# LEARNING ATTRIBUTE REPRESENTATIONS WITH DEEP CONVOLUTIONAL NEURAL NETWORKS FOR WORD SPOTTING

Dissertation
zur Erlangung des Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

SEBASTIAN SUDHOLT

Dortmund

2018

|  |  |
|---|---|
| Tag der mündlichen Prüfung: | 17.12.2018 |
| Dekan: | Prof. Dr.-Ing. Gernot A. Fink |
| Gutachter: | Prof. Dr.-Ing. Gernot A. Fink<br>Prof. Dr. Lambert Schomaker |

# NOTATION

When using mathematical expressions in this thesis all variables, functions and arguments are defined where they appear first. The following notation is used for specific mathematical elements:

| | |
|---|---|
| $a, b, c, \ldots$ | scalar |
| $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$ | vector |
| $\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$ | matrix or tensor |
| $A, B, C, \ldots$ | univariate random variable |
| $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \ldots$ | multivariate random variable |
| $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ | integer value for, e.g., counts, cardinalities or dimensionalities |
| $\boldsymbol{\mathsf{A}}, \boldsymbol{\mathsf{B}}, \boldsymbol{\mathsf{C}}, \ldots$ | set |
| $\lvert\boldsymbol{\mathsf{A}}\rvert$ | the cardinality of set $\boldsymbol{\mathsf{A}}$ |
| $\boldsymbol{\mathsf{A}}^{(i)}$ | the $i$-th element in set $\boldsymbol{\mathsf{A}}$ |
| $\bar{\boldsymbol{\mathsf{A}}}$ | average value of the set $\boldsymbol{\mathsf{A}}$, i.e., $\frac{1}{\lvert\boldsymbol{\mathsf{A}}\rvert}\sum_{i=1}^{\lvert\boldsymbol{\mathsf{A}}\rvert}\boldsymbol{\mathsf{A}}^{(i)}$ |
| $\mathbf{a}^T$ | the transpose of vector $\mathbf{a}$ |
| $\hat{a}, \hat{\mathbf{a}}$ | estimate or prediction of a scalar or vector |
| $a_i$ | the $i$-th element of vector $\mathbf{a}$ |
| $a_{i,j}$ | the element of the matrix $\mathbf{A}$ at row $i$ and column $j$ |
| $\mathrm{diag}(\mathbf{A})$ | the main diagonal of $\mathbf{A}$ |
| $a_i^{(j)}$ | the $i$-th element of the $j$-th vector in set $\boldsymbol{\mathsf{S}} = \left\{\mathbf{a}^{(j)}\right\}_{j=1}^{n}$ |
| $f(x), f(\mathbf{x})$ | scalar function with scalar or vector argument |
| $\mathbf{f}(\mathbf{x})$ | vector function with vector argument |
| $\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_i}$ | the partial derivative of $\mathbf{f}$ with respect to $x_i$ |
| $\mathbb{E}[X]$ | the expected value of the univariate random variable $X$ |
| $\langle \mathbf{a}, \mathbf{b} \rangle$ | inner product between the vectors $\mathbf{a}$ and $\mathbf{b}$ |

All vectors are assumed to be column vectors if not specified otherwise. When referring to the norm of vectors, the notation $||\cdot||$ is short for the Euclidean norm $||\cdot||_2$. The operator $\odot$ denotes the Hadamard product (element-wise vector or matrix multiplication):

$$\mathbf{a} \odot \mathbf{b} = (a_1 b_1, a_2 b_2, \ldots, a_D b_D)^T .$$

For all functions that are only defined on scalars, a vectorial argument indicates element-wise application of the function, e.g.:

$$\begin{aligned}\operatorname{sigm}(x) &= \frac{1}{1 + e^{-x}} \\ \operatorname{sigm}(\mathbf{x}) &= (\operatorname{sigm}(x_1), \operatorname{sigm}(x_2), \ldots, \operatorname{sigm}(x_D))^T .\end{aligned}$$

# CONTENTS

# 1 INTRODUCTION

Understanding the contents of handwritten texts from document images has long been a traditional field of research in computer science. The ultimate goal is to automatically transcribe the text in the images into an electronic format. This would make the documents from which the images were generated much easier to access and would also allow for a fast extraction of information. Especially for historical documents a possibility to easily sift through large document image collections would be of high interest. There exist vast amounts of manuscripts all over the world storing substantial amounts of yet untapped information on cultural heritage. Being able to extract this information for large and different corpora would allow historians unprecedented insight into various aspects of ancient human life.

The desired goal is thus to obtain information on the text embedded in digital document images with no manual human interaction at all. A well known approach for achieving this is to make use of models known from the field of pattern recognition and machine learning in order to classify the text in the images into electronic representations of characters or words. This approach is known as Optical Character Recognition (OCR) or text recognition and belongs to the oldest applications of pattern recognition and computer science in general with the first works in this field dating back as far as 1914 [31]. Despite its long history, handwritten text recognition is still considered an unsolved task as classification systems are still not able to consistently achieve results as are common for machine printed text recognition. This is especially true for historical documents as the text to be recognized typically exhibits different amounts of degradation as well as large variability in handwriting for the same characters and words.

Depending on the task at hand, a full transcription of the text might, however, not be necessary. If a potential user is only interested in whether a certain word or text portion is present in a given document collection or not, retrieval-based approaches are able to produce more robust results than recognition-based ones. These retrieval-based approaches compare parts of the document images to a sought-after query and decide if the individual parts are similar to the query. The similarity measure does not need to be binary and is often times a real number representing the level of similarity. For a given method, the result is then a list of parts of the document images which are deemed relevant by the method. Typically, this list is sorted according to the determined similarity in descending order. In the field of document image analysis, this retrieval approach is known as *keyword spotting* or simply *word spotting*.

Word spotting is the problem of interest in this thesis. In particular, a method will be presented which allows for using neural network models in order to approach different word spotting tasks. This method is inspired by a recent state-of-the-art approach by Almazán *et al.* [9] which utilizes semantic attributes for word spotting. In pattern recognition and computer vision, semantic attributes describe characteristics of classes which may be shared between classes. This sharing ability enables an attribute representations to encode parts of different classes which are common and those which are not. For example, when classifying animals, the classes *tiger* and *zebra* may share an attribute *striped*.

In the context of document images analysis, Almazán *et al.* [9] encode the characters of a word in combination with their position as attributes. This way, a powerful representation is obtained. Using this approach, they were able to establish state-of-the-art performance at the time of their first publication.

The success of any attribute-based method is, of course, highly dependent on the ability of a classifier to correctly predict the individual attributes. In order to accomplish an accurate prediction, the use of Convolutional Neural Networks (CNNs) is proposed in this thesis. CNNs have recently attracted a substantial amount of research interest as they are able to consistently achieve state-of-the-art results in virtually all fields of computer vision. Their main advantage compared to other methods is their ability to jointly optimize a classifier and the feature representations obtained from the images. This characteristic is known as end-to-end learning. While CNNs have been used extensively for classifying data into one of multiple classes for various tasks, predicting attributes with these neural networks has largely been done for face and fashion attributes only.

For the method presented in this thesis a CNN is trained to predict attribute representations in an end-to-end fashion. These attributes are leveraged in order to perform word spotting. The core contribution lies in the design and evaluation of different neural network architectures which are specifically designed to be applied to document images. A big part of this design is to determine suitable loss functions for the CNNs. Loss functions are a crucial ingredient in the training of neural networks in general and largely determine what kind of annotations the individual networks are able to learn for the given images. It will be shown experimentally, that the obtained architectures achieve state-of-the-art results for various word spotting benchmarks.

This thesis presents five different contributions to the fields of word spotting and deep learning. These individual contributions will be discussed in Sec. 1.1. The section also states if and where parts of the contributions have been published before. The chapter is concluded by giving an outline over the following chapters of this thesis in Sec. 1.2.

## 1.1 CONTRIBUTIONS

Word spotting methods in general can be discriminated with respect to whether they need a segmentation for the document images into word images as well as the types of query they are able to process. The method in this thesis is able to perform segmentation-based Query-by-Example (QbE) and Query-by-String (QbS) word spotting[1]. This is achieved using attribute and attribute-like representations of strings. While using attributes for word spotting was initially presented in [9], using a CNN for predicting the attributes in an end-to-end is a novel approach. For designing the presented method, a number of contributions are made regarding the fields of document image analysis and neural networks. These contributions are explained in detail in the following. Please note that some of the contributions in this thesis have previously been published in peer reviewed conferences or journals. According to the regulations governing this thesis (German: Promotionsordnung), the author is required to report his individual contributions in joint publications which will be done in this section also.

1 The presented method is in principal also able to support a lesser known query paradigm called Query-by-Online-Trajectory (QbO). While QbO is not at the primary focus in this thesis, there exists a chapter in the appendix which is concerned with using the approach presented here for QbO word spotting (cf. Appendix C).

*Loss functions for attribute representations*

The big advantage of CNNs is their ability to combine the training of a classifier and the learning of feature representations for the input images. While this characteristic has been exploited in multi-class classification in a large number of scenarios, the prediction of attribute representations has mainly focused on human characteristics or fashion. Before the initial publication of the presented method in 2016, learning attribute representations with CNNs had not been done for document image analysis in general and word spotting in particular. The standard procedure to train neural networks in general is an algorithm known as backpropagation. While the specifics of training will be explained later, at this point it suffices to say that the aforementioned loss function is critical for the training process. A major contribution of this thesis is to derive suitable loss functions for attribute representations. These loss functions are derived from a statistical model. This allows for interpreting the training process from a probabilistic point of view. In particular, the well-known Binary Cross Entropy Loss (BCEL) and Cosine Loss functions can be obtained through the method used in this thesis. This method was first published in [189] and more thoroughly discussed in [190]. In these two publications, the author was responsible for proposing the Generalized Linear Model (GLM) framework for a theoretically sound derivation of the loss functions, the derivation of the Cosine Loss from the von Mises-Fisher distribution, finding the connection between the Euclidean Loss and the Cosine Loss and for conducting the experimental evaluations.

*CNN architectures for word spotting*

A common approach for computer vision tasks using CNNs is to make use of network architectures which have been shown to achieve state-of-the-art results in other scenarios and apply them to the problem at hand. The most commonly used CNN architectures like AlexNet and VGGnet, however, where initially proposed for processing natural images. In contrast, document or word images feature different traits with respect to number of objects, object sizes and variability. It can be shown that standard CNN architectures do achieve competitive performance when applied to document images [177]. However, it is reasonable to expect that architectures which are designed to cope with the defining characteristics of document and word images are enabled to perform better than these standard architectures. Thus, three CNN architectures are proposed in this thesis designed for word images. While all these architectures build on successful CNNs such as VGGnet and Residual Network (ResNet), they are able to accept word images of varying sizes while still producing a fixed-size attribute representation. This allows the CNNs to process the images without the need for rescaling or cropping. In order to accomplish this, the first architecture makes use of the well known Spatial Pyramid Pooling (SPP) layer. For the other two architectures a novel layer is used which is referred to as Temporal Pyramid Pooling (TPP) layer. While this layer also allows the CNNs to accept images of varying sizes, it is explicitly constructed for word images as input.

The three proposed architectures are called PHOCNet, TPP-PHOCNet and PHOCResNet. The PHOCNet was first proposed in [188] while the TPP-PHOCNet and the TPP layer where presented in [189]. In both publications, the author was responsible for designing the CNN architectures as well as conducting the different word spotting experiments which they were involved in.

*Augmentation method for word images*

The CNNs used in this thesis generally fall into the category of deep learning models. A common challenge with these models is that the number of trainable parameters is usually in excess of $10^8$. If no counter measures are taken, the vast amount of parameters makes these neural networks prone to overfitting during training. This means that the model learns the characteristics of the training data "by heart" and is unable to achieve a satisfactory generalization ability for new and unseen data. Approaches for countering overfitting behavior are known as regularization techniques. Goodfellow *et al.* define regularization as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error" [58, p. 224]. A prominent and oft-used method for regularization is dataset augmentation. For this, images are added to the dataset based on label-preserving transformations of images from the dataset. For natural images, these label-preserving transformations include for example rotation, scaling and cropping or flipping of the images. However, not all of these techniques are directly applicable to document or word images. Hence, a novel method is proposed which allows for randomly sampling a transformation for word images which incorporates a number of image transformations which may be applicable to word images such as shear, rotation, scaling and translation. This augmentation strategy was first proposed by the author in [188] as part of a joint publication.

*Experimental evaluation of the different architectures and attribute representations*

Most works on attribute-based word spotting make use of the attribute representation proposed by Almazán *et al.* [9]. However, there exist other attribute and attribute-like embeddings for word strings which can be used for word spotting in general and for the presented method in particular as well. The question that arises is which representation is best suited for performing attribute-based word spotting with CNNs. This question will be answered through a thorough experimental evaluation comparing the three word string embeddings known as Pyramidal Histogram of Characters (PHOC), Spatial Pyramid of Characters (SPOC) and Discrete Cosine Transform of Words (DCToW). What all three representation have in common is that pairs of attributes in a single representation may be correlated. This correlation may lead the neural network to falsely predict the presence of attributes based on other attributes which are correlated with the desired one. In order to assess whether this is the case, two decorrelation techniques for attribute as well as attribute-like representations are proposed and evaluated.

Part of the evaluation is the investigation of how much the amount of annotated training images can be reduced in order for the network to still perform well. Using the proposed CNNs this way was first done in [61]. Here, the author was responsible for the idea of training CNNs for word spotting under this increasingly weak supervision.

The evaluation will investigate the two most common query types namely images (QbE) and word strings (QbS).

*Analysis of what the CNNs have learned*

Deep neural networks still carry the stigma of being black boxes when it comes to explaining the reasons for their predictions. However, there exist visualization techniques which allow for gaining insight into what the trained filter kernels of a CNN in a specific layer have learned to detect (cf. e.g. [180, 183, 213]). In this thesis, the guided backpropation method [183] will be used in order to investigate the filter kernels of the proposed CNNs. This analysis allows for interesting observations regarding the behavior of filters in the final convolutional layers. As will be seen, these filters often times function as character detectors without ever having been supplied with segmented characters.

## 1.2 OUTLINE

The main part of this thesis is split into 8 chapters with the current chapter serving as introduction and motivation. The remaining chapters are organized as follows:

CHAP. 2 - PATTERN RECOGNITION AND COMPUTER VISION FUNDAMENTALS
The proposed method can be accounted towards the fields of pattern recognition in general and computer vision in particular. This chapter explains the basic fundamentals of these two fields of research which are necessary in order to understand the method presented later. This includes a detailed presentation of attributes in computer vision as they build an integral part of the method.

CHAP. 3 - NEURAL NETWORKS AND DEEP LEARNING As neural networks are the model used at the core of the presented method, this chapter gives a thorough overview over these machine learning models. In particular, important aspects for neural networks from the field of deep learning are explained as the CNNs used for word spotting fall into this category.

CHAP. 4 - WORD SPOTTING While neural networks are the models of interest in this thesis, word spotting is the application of interest. As word spotting applications can be discriminated into various tasks depending on the problem at hand, this chapter first explains established terminology with respect to this field of document images analysis. Afterwards, important contributions to word spotting are presented and their methodologies are explained in detail.

CHAP. 5 - ATTRIBUTE CNNS This chapter presents the methodology proposed for word spotting in this thesis. As this methodology hinges on using the right loss functions for training the proposed neural networks, a statistical model is explained first which allows for deriving loss functions based on an assumption on the distribution of attributes. This model is subsequently used to derive the loss functions used for learning attribute representations. The chapter is concluded by presenting the three CNN architectures proposed for the word spotting problem at hand. In particular, their design choices are explained and justified.

CHAP. 6 - RELATED WORK The presented method has relations to other methods proposed previously in the literature. This chapter presents other works related to the approach proposed in the previous chapter. Especially, the similarities and differences between the CNN-based method presented in this thesis to the related methods are pointed out and explained.

CHAP. 7 - EXPERIMENTAL EVALUATION In this chapter, the results obtained with the proposed approach for six different word spotting benchmarks are presented. For this, the benchmarks and their specific characteristics are explained first. Afterwards, the obtained performance values are listed and compared to other methods from the literature. Finally, an analysis is conducted which allows for interpreting what the final convolutional layer has learned to detect.

CHAP. 8 - CONCLUSION The final chapter concludes the main part of the thesis by summarizing the obtained results, findings and insights as well as giving an outlook on potential future work.

APPENDIX AND BACKMATTER After the main part of the thesis, there is an appendix containing a detailed list of publications by the author, additional mathematical derivations, a chapter on QbO word spotting with the proposed CNNs and further results concerning the significance tests from the experimental evaluation. The backmatter consists of the bibliography, a definition of all acronyms and an index.

Besides referencing articles from the literature, some sources such as, e.g., the software libraries used for implementing the presented method or the datasets used for evaluation are referenced by internet URLs. All URL references in this thesis were last checked on December 21, 2018 for availability and content.

# 2 PATTERN RECOGNITION AND COMPUTER VISION FUNDAMENTALS

The method presented in this thesis can be assigned to the field of pattern recognition. Pattern recognition methods in general aim at replicating human perception abilities (cf. e.g. [40, p. 2]). The branch of pattern recognition which is concerned with visual perceptions is known as computer vision. Computer vision problems include classifying the content of images into one of many classes, detecting or segmenting images into salient objects or retrieving relevant instances from a database of images. The application of interest in this thesis, i.e., word spotting, can generally be considered a computer vision problem, too, as the data to be processed consists of images of handwritten words. The approach for word spotting presented in this thesis and other related works for word spotting make use of computer vision methodology used for classifying images. Hence, the rest of this chapter will focus on presenting the fundamentals for solving classification problems in computer vision.

Despite the large variety in tasks, classic approaches for image classification share, to some extent, a common pipeline (cf. e.g. [40, p. 5]). This pipeline is depicted in Fig. 1: The first step in this pipeline is the acquisition of images through some form of camera device. These images may then be preprocessed in order to reduce the amount of unwanted variability. Typical preprocessing steps include pixel value scaling, noise removal or edge enhancement (cf. e.g. [195, p. 101]). After preprocessing, the next step is to extract so-called features from the images. The feature representation of a given image is then used in order to classify the image into one of the available classes. This classification step requires a classifier which is responsible for generating the prediction. Using parametrized models which are able to automatically learn the statistical characteristics of the feature representations has shown itself to be a successful approach for finding good classifiers. The structure for these models, however, is not determined automatically but by a human expert. Interestingly, it can be shown that there does not exist a single best classifier without *a priori* knowledge of the data to be processed [209]. In the context of pattern recognition, this fact is known as the No Free Lunch theorem.

Having decided on a type of classifier, finding suitable model parameters can be achieved by setting them such that the classifier predicts the desired classes for a given sample set of images. This process is known as *supervised training*. It is typically an iterative optimization procedure during which the accuracy of the classifier's predictions is gradually increased. Besides the images themselves, supervised training requires a corresponding annotation for each image indicating the desired class. The set of tuples of images and annotations used for training is called training set. The classifier is trained using the feature representation of the respective training images.

When performing supervised training, it is important that the sample set is representative for the data to be classified later. This is due to the classifier being tasked to extrapolate the knowledge obtained from the training to new data when making predictions.

One notable aspect of the classic computer vision pipeline shown in Fig. 1 is that the classifier does not have the ability to influence the feature extraction or preprocessing step

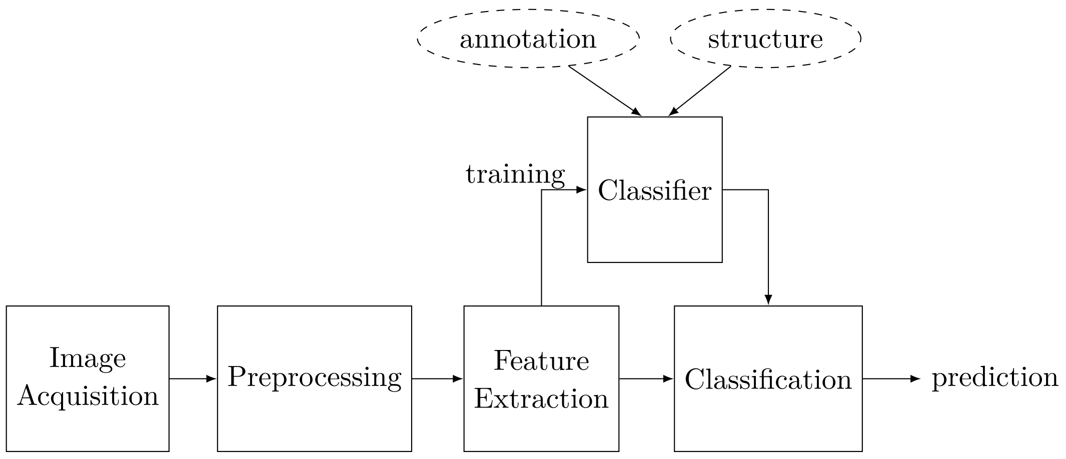


Figure 1: The figure displays the standard pipeline for computer vision methods: First, an image is obtained from some sort of camera device which may then be preprocessed in order to reduce unwanted variability. Afterwards, features are extracted from the image. A classifier is then used in order to predict the class membership of content of the image using the feature representation. While the general structure of the classifier is obtained through expert design, the parameters of the respective classification model are usually trained using a set of annotated feature vectors.

in order to optimize the classification accuracy. Preprocessing, feature extraction and classifier are thus typically optimized individually. A notable exception to this are the recently successful Convolutional Neural Networks (CNNs) from the field of deep learning. Fig. 2 visualizes a modern deep learning pipeline for computer vision using these neural networks. As can be seen in the figure, CNNs integrate preprocessing and feature extraction into the classification model. This characteristic is their main advantage compared to classic approaches: Instead of optimizing each block in the classic pipeline individually, a CNN can be optimized in an *end-to-end* fashion. This term refers to the fact that preprocessing, feature extraction and classification are optimized combinedly in order to obtain the best classification accuracy. It needs to be noted, that the No Free Lunch theorem still dictates that a CNN is not superior to other classification models which make use of the classic pipeline *a priori*. However, deep learning approaches can be shown empirically to outperform classic ones on different computer vision benchmarks by a substantial margin (cf. e.g.[54, 92, 172]). Especially the prestigious ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [167] has been dominated by deep CNNs ever since they were made famous by Krizhevsky *et al.* [92] in 2012 in this competition.

Both classic and deep learning approaches face a problem when being presented with images of classes which were not among the training classes, i.e., are unknown from a classifier's point of view. Without modifications, a classifier can typically only predict one of its known classes [95]. It would, of course, be desirable to enhance a classifier such that it is able to easily incorporate knowledge about unknown classes without the need of collecting training samples for them. This approach is known as zero-shot learning [96]. Zero-shot learning is of high importance for this thesis as the presented methodology requires to predict representations for word images of classes which may have not been observed during training.

The rest of this chapter presents relevant methodology for the classic computer vision pipeline as well as zero-shot learning techniques. Describing the classic approaches is necessary as they are at the core of a number of important works related to the method presented

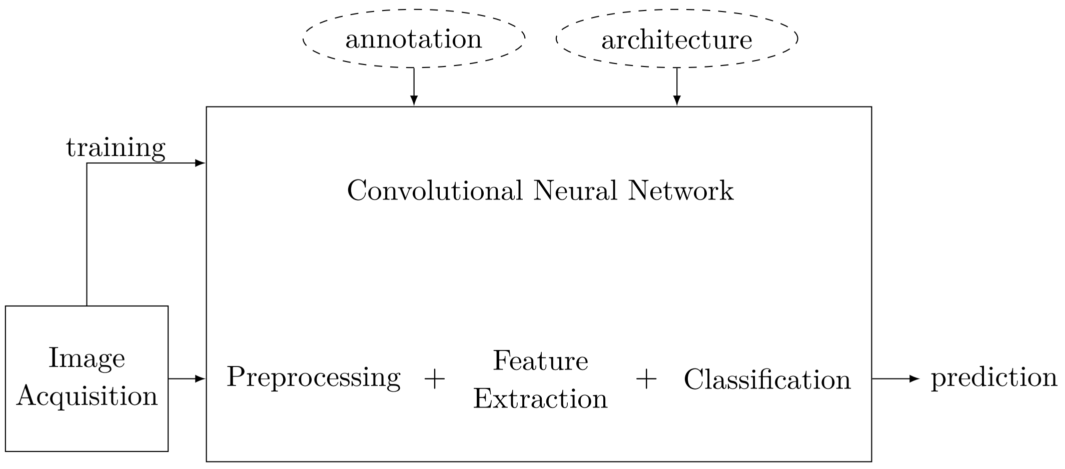


Figure 2: The figure conceptually visualizes how a Convolutional Neural Network (CNN) is used in computer vision for classification: The acquired image is used as input to the respective neural network which encapsulates a potential preprocessing, feature extraction and classification. Each of the three parts can be learned in an *end-to-end* fashion. For this, a CNN typically requires the training images to be manually labeled. Besides for supplying the annotation, an expert is also required for defining the concrete architecture of the CNN to be used.

in this thesis. As the presented method makes use of a deep learning approach, CNNs and relevant fundamentals will be extensively discussed in a dedicated chapter (Chap. 3). General preprocessing techniques will not be covered here as most related works ignore a preprocessing step and extract features directly from the given images. If preprocessing is applied in a related method, it is very specific for the problem of processing images of handwritten documents. For these related works, the preprocessing step will be described when explaining the method. The interested reader may find detailed descriptions for general preprocessing techniques in computer vision in, e.g., [195, p. 99]. In contrast to preprocessing, feature extraction techniques and classifier types found in the related word spotting literature are typically successful ones from other fields of computer vision or pattern recognition in general. The next section (Sec. 2.1) will hence describe common approaches for extracting feature representations from images in the classic computer vision pipeline. The ensuing section (Sec. 2.2) then focuses on describing different classifier models. The chapter is concluded by describing important fundamentals concerning zero-shot learning (Sec. 2.3). These fundamentals are relevant for some of the related works as well as the presented method.

## 2.1 FEATURE REPRESENTATIONS FOR IMAGE CLASSIFICATION

The goal of encoding an image into a feature representation is to obtain a numerical vector which may further be processed by the ensuing classifier. Choosing a discriminative feature representation is paramount for solving the classification task: If for all classes the feature vectors of a specific class are compact while the regions in feature space representing the different classes are far apart, a classifier will be presented with a much easier task compared to when different classes overlap in feature space. For determining useful feature representation, one would like a closed loop optimization where the goal is to maximize the classification performance for the training samples with respect to the feature representation. In the case of classic approaches, however, the loop is open as feature representation and classifier are optimized independently (cf. Fig. 1). Hence, a common approach is to define the feature representation heuristically.

A very famous feature representation for computer vision methods is the so-called Bag of Features (BoF) (cf. e.g. [129, 181]). Its principals derive from Bag-of-Words models known from natural language processing. The BoF representation is essentially a histogram of quantized local image statistics obtained at various points of an image. In order to create a BoF representation, local descriptors are computed from the image first. These descriptors capture characteristics of the respective image in a defined and small pixel neighborhood. Typically, the descriptors of the Scale Invariant Feature Transform (SIFT) [112] are used in BoF representations but other local features such as Speeded Up Robust Features (SURF) [15] are possible as well and have been used in the literature (cf. e.g. [79]). While in the original SIFT method the descriptors were calculated at specified keypoint locations, current methods for image classification usually compute the descriptors in a dense grid across the entire image when using BoF representations (cf. e.g. [25]).

Having computed the local descriptors for a set of images, the next step for obtaining a BoF representation is to quantize them. This quantization step is necessary for creating the BoF histogram as the individual descriptors have to be assigned to a specific bin in the histogram. In order to obtain the assignment of local descriptors to bins, the descriptors obtained from the images are clustered. This can be achieved with, e.g., Lloyd's algorithm [111] or MacQueen's k-means algorithm [114]. The codebook obtained after clustering is referred to as *visual vocabulary* and the cluster centers as *visual words*. For quantizing the descriptors, each descriptor is assigned to its closest visual word in descriptor space. Having found a visual word representative for each descriptor in an image, the BoF histogram is obtained by counting the number of descriptors which have been assigned to the individual visual words.

One notable characteristic of BoF representations is the loss of localization information for specific descriptors. This is due to the position of the individual descriptors not being taken into account for creating the BoF histogram. While this does bare certain advantages with respect to, e.g., translation invariance, a complete loss of localization may lead to a decreased overall performance in certain situations [98]. *Spatial pyramid* representations [98] seek to counter this problem. The spatial pyramid concept is visualized in Fig. 3. The dots represent local descriptors with the color depicting one of three possible visual words the respective descriptor has been assigned to during clustering. The spatial pyramid representation is then obtained by creating individual BoF histograms for different parts of the image in a pyramidal fashion. In the first level, the standard BoF representation is used. In the second level, the image is subdivided into four equally sized regions and a BoF histogram is computed for each region. This process is continued for the ensuing pyramid levels with each layer having double the amount of regions along the horizontal and vertical axis compared to its previous level. All levels make use of the same visual vocabulary obtained during the initial clustering. Finally, all individual histograms are concatenated to form the spatial pyramid representation. While the amount of levels can be chosen arbitrarily, typical spatial pyramid approaches use three levels (cf. e.g. [25, 96, 98]). As a 3-level spatial pyramid has a grand total of 21 bins across all levels, this also increases the overall representation size by this factor compared to a standard BoF representation. However, spatial pyramids are typically sparse and can thus be stored and processed quite efficiently [169].

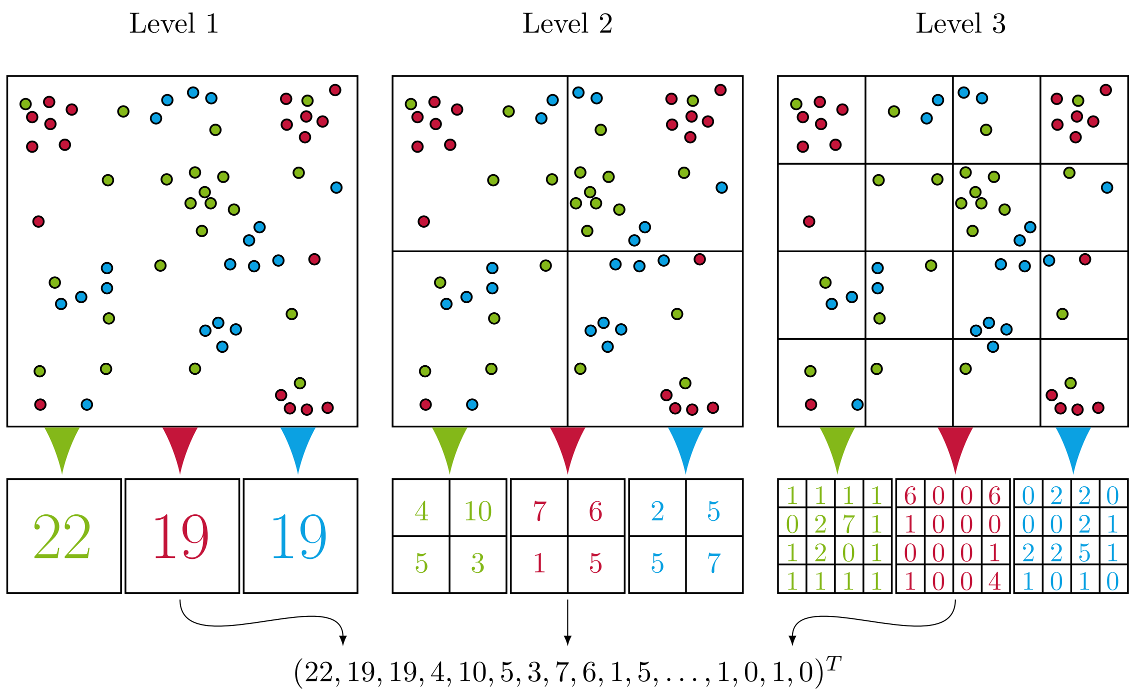

Level 1
Level 2
Level 3
22
19
19
$(22, 19, 19, 4, 10, 5, 3, 7, 6, 1, 5, \ldots, 1, 0, 1, 0)^T$

Figure 3: The figure visualizes how to create a 3-level spatial pyramid representation from quantized features belonging to one of three visual words. In each level the image to be processed is split into different regions in a quad tree-like structure. For each region a BoF histogram is extracted. The individual histograms may further be scaled or normalized in order to account for the expected loss in absolute counts for levels with a higher index. Finally, all histograms from all bins are concatenated in order to form the spatial pyramid representation. The figure is inspired by Fig. 1 in [98].

Both BoF and spatial pyramid representations share the hard assignment of a descriptor to a visual word. It can be shown empirically that this hard assignment performs worse compared to soft-assignment strategies for a large variety of tasks [25, 169]. A very successful representation making use of soft-assignment has been the Fisher Vector [169]. The Fisher Vector contains information about log-likelihood gradients of a probabilistic model given the descriptors. In order to compute a Fisher Vector representation, the descriptors of a set of images are assumed to be realizations of a random variable. For modeling this generative process, a Gaussian Mixture Model (GMM) is chosen. A GMM is a mixture model of $\mathsf{M}$ different normal distributions. The Probability Density Function (PDF) of a GMM can be defined as weighted sum of the mixture components:

$$f_{\text{GMM}}\left(\mathbf{x} \,\middle|\, \left\{w^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\Sigma}^{(i)}\right\}_{i=1}^{\mathsf{M}}\right) = \sum_{i=1}^{\mathsf{M}} w^{(i)} f_{\mathcal{N}}\left(\mathbf{x} \,\middle|\, \boldsymbol{\mu}^{(i)}, \boldsymbol{\Sigma}^{(i)}\right), \text{ where} \tag{1}$$

$$\sum_{i=1}^{\mathsf{M}} w^{(i)} = 1, \forall i : w^{(i)} \geq 0 \text{ and} \tag{2}$$

$$f_{\mathcal{N}}\left(\mathbf{x} \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{\sqrt{(2\pi)^{\mathsf{D_d}} \cdot |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}\left(\mathbf{x} - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1}\left(\mathbf{x} - \boldsymbol{\mu}\right)\right). \tag{3}$$

In Eq. 1, $w^{(i)}, \boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\Sigma}^{(i)}$ are the respective weight, mean and covariance matrix for the $i$-th mixture component. In Eq. 3, $\mathsf{D_d}$ is the dimensionality of the descriptors and $|\boldsymbol{\Sigma}|$ the determinant of the covariance matrix. The descriptors obtained from the images are used in order to estimate the parameters of the GMM using the Expectation-Maximization algorithm. In analogy to the visual vocabulary from BoF, the GMM involved in creating a Fisher Vector is sometimes referred to as *probabilistic visual vocabulary* [169]. Typically, only diagonal covariance matrices are used for the GMM's mixture components. In order to generate the Fisher Vector representation for a given image, the derivative of the GMM's log-likelihood given the descriptors in the image is computed with respect to the mixture weight, mean and covariance matrix of each component. For this, the weights are first re-parametrized using the softmax formalism [89]:

$$w^{(i)} = \frac{\exp\left(\alpha^{(i)}\right)}{\sum_{j=1}^{m} \exp\left(\alpha^{(j)}\right)}. \tag{4}$$

This way, the constraint from Eq. 2 is implicitly encoded into the model. The gradients for each descriptor $\mathbf{x}$ are then computed as follows [89]:

$$\frac{\partial \log f_{\text{GMM}}}{\partial \alpha^{(i)}} = p_i(\mathbf{x}) - w^{(i)} \tag{5}$$

$$\frac{\partial \log f_{\text{GMM}}}{\partial \boldsymbol{\mu}^{(i)}} = p_i(\mathbf{x}) \cdot {\boldsymbol{\Sigma}^{(i)}}^{-1} \left(\mathbf{x} - \boldsymbol{\mu}^{(i)}\right) \tag{6}$$

$$\frac{\partial \log f_{\text{GMM}}}{\partial {\boldsymbol{\Sigma}^{(i)}}^{-1}} = \frac{p_i(\mathbf{x})}{2} \cdot \left(\boldsymbol{\Sigma}^{(i)} - \left(\mathbf{x} - \boldsymbol{\mu}^{(i)}\right)\left(\mathbf{x} - \boldsymbol{\mu}^{(i)}\right)^T\right). \tag{7}$$

In the equations above, $p_i(\mathbf{x})$ is the probability of the descriptor being generated by the $i$-th mixture component and is computed as follows [169]:

$$p_i(\mathbf{x}) = \frac{w^{(i)} f_{\mathcal{N}}\left(\mathbf{x} \,\middle|\, \boldsymbol{\mu}^{(i)}, \boldsymbol{\Sigma}^{(i)}\right)}{\sum_{j=1}^{\mathsf{M}} w^{(j)} f_{\mathcal{N}}\left(\mathbf{x} \,\middle|\, \boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right)} \tag{8}$$

For building the Fisher Vector, the gradients for all descriptors from the available images are computed, averaged and concatenated. Note that for this only the diagonal of the gradient in Eq. 7 is used. The size $\mathsf{D}_{\text{FV}}$ of the resulting Fisher Vector is thus

$$\mathsf{D}_{\text{FV}} = (2\mathsf{D}_{\text{d}} + 1) \cdot \mathsf{M}. \tag{9}$$

The concept of subdividing the image into regions in a pyramidal form as was done for the spatial pyramids can be used for the Fisher Vector as well. This is achieved by computing an individual GMM per region in the pyramid and concatenating all resulting Fisher Vectors [169]. As Fisher Vectors are almost always dense, they, however, typically consume more memory than spatial pyramid representations of the same dimensionality.

## 2.2 IMAGE CLASSIFICATION

Having obtained a feature representation for the training images, the next step in the classic computer vision pipeline is to train a classifier (cf. Fig. 1). While the feature representations for computer vision approaches are typically designed specifically for images, the classifiers used in computer vision methods are usually not specific to this subfield of pattern recognition.

There exists a variety of classifiers in the pattern recognition literature. A central component of each of these classifiers is the *classification rule*. The classification rule is essentially the function a classifier uses to make its prediction. For example, statistical classifiers predict a data sample to belong to a class $c$ such that the costs for a wrong classification are minimized (cf. e.g. [40, p. 20]). If all false classifications carry the same cost, statistical classifiers assign a given sample to the class $c$ if $c$ is the class with maximum posterior probabilities given the feature vector $\mathbf{x}$ of the sample. Using Bayes' rule, this classification rule can be expressed as follows (e.g. [40, p. 24]):

$$\underset{c}{\operatorname{argmax}}\, p\left(c \mid \mathbf{x}\right) = \underset{c}{\operatorname{argmax}} \frac{p(c)p\left(\mathbf{x} \mid c\right)}{p(\mathbf{x})} = \underset{c}{\operatorname{argmax}}\, p(c)p\left(\mathbf{x} \mid c\right), \tag{10}$$

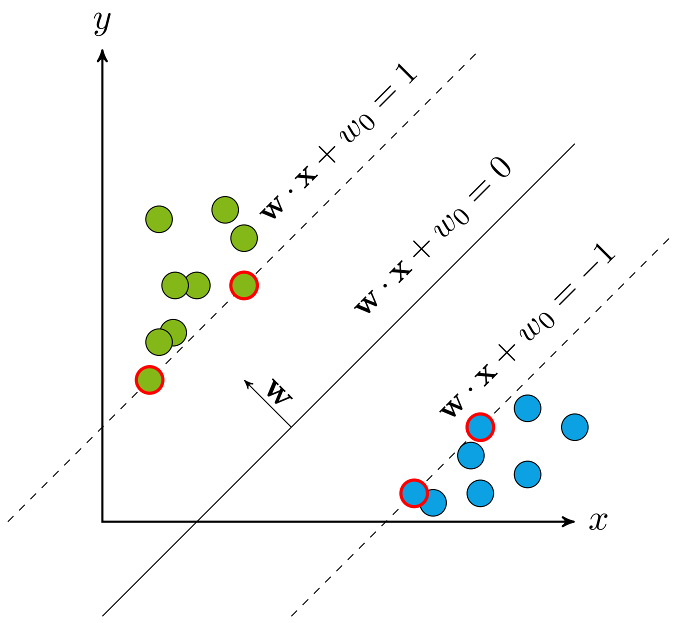
where $p(c)$ is the prior probability for class $c$ and $p\left(\mathbf{x} \mid c\right)$ the probability for observing the vector $\mathbf{x}$ given the class $c$. These two probabilities need to be estimated from the training samples.

While statistical classifiers are used quite regularly in classic computer vision approaches, other classifier types such as the $k$-Nearest Neighbor classifier (kNN) (e.g. [124, p. 16]), Random Forests [21] or Support Vector Machines (SVMs) [30] can be found there as well. Please recall that no classifier can be identified as optimal for a given task *a priori* [209]. The best classifier and its respective parametrization for a given set of data can thus only be determined empirically. For this, the available data can be split into a training set, which is used for training the classifier, and a validation set, which is then used in order to determine the classifiers' performance (cf. e.g. [124, p. 23]). Training a classifier and using it for predicting new data will be explained exemplarily in the following using an SVM as classifier. SVMs are of general interest for this thesis as a number of works closely related to the presented method make use of these classifier models.

An SVM classifies data into one of two classes which is commonly known as binary classification. Given a training set $\mathsf{S} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{\mathsf{N}_s}$ of tuples of feature representations $\mathbf{x}$ and labels $y$, an SVM tries to find a hyperplane in feature space which separates the two classes and has maximum distance to the samples from either class. For convenience, it is assumed that the two classes are encoded as $-1$ and $1$ in the labels. The reason for this will be seen when formally discussing the training algorithm for the SVM.

Having found the hyperplane with maximum distance to the samples, a new sample can be classified by determining in which half space it lies with respect to the hyperplane. This is the SVM's classification rule. The concept of an SVM is visualized in Fig. 4. In the figure, the dashed lines represent the so-called margins and the solid black line the separating hyperplane of the SVM. The two margins are always parallel and the separating hyperplane is always halfway between them.

The two major shortcomings of the SVM as explained above are that it is only able to classify binary data and only if the data is linearly separable. The first aspect can be countered by either *one-vs-all* or *one-vs-one* classification. In one-vs-all there exists one SVM per class which tries two find the hyperplane separating the samples of the respective class from all other samples. In one-vs-one, there exists one SVM per pair of classes. In both scenarios, the final classification result is obtained through some form of voting mechanism (cf. e.g. [124, p. 503]). In order for the SVM to handle non-linearly separable data, there exist two common concepts. The first is concerned with individual outliers: If only a small amount of outliers from each class causes the data to not be linearly separable, the SVM may be allowed to position the hyperplane such that the outliers are classified incorrectly during training. The second approach for classifying non-linearly separable data is concerned with the situation when a non-linear class boundary is not caused by outliers but is rather a structural characteristic of the data. In these situations



$y$
$x$
$\mathbf{w} \cdot \mathbf{x} + w_0 = 1$
$\mathbf{w} \cdot \mathbf{x} + w_0 = 0$
$\mathbf{w} \cdot \mathbf{x} + w_0 = -1$
$\mathbf{w}$

Figure 4: The figure visualizes the concept of an SVM. The samples of the two classes are represented by green and blue dots. The hyperplane found during training is the one with maximum distance to all samples (solid black line). The margins of the hyperplane are depicted by dashed lines. Data samples, which lie on the margins, are called support vectors. All data points with a red outline are the support vectors for the specific classes.

the data can be projected into a high-dimensional space using a non-linear transformation. Intuitively speaking, the higher dimensionality increases the probability of being able to find a hyperplane for which the projected data is again linearly separable. As will be seen in the following, an SVM only requires inner products for training and predicting a class. Thus, the computationally expensive non-linear transformation can be replaced by a kernel function (cf. e.g. [124, p. 488]). A kernel function allows for computing the dot product in a projected space using only the original data samples.

When using feature representations such as spatial pyramids or Fisher Vectors as input to an SVM, the dimensionality of the respective representations is usually in the order of $10^5$ or even higher [25, 153]. These input spaces are typically large enough such that an SVM without a kernel function can find a hyperplane which separates the training samples into the desired classes. SVMs, which do not make use of a kernel, are referred to as *linear SVMs* (cf. e.g. [75]). In the context of this thesis, only linear SVMs are of interest which is why the application of kernel functions will not be discussed in the following. The interested reader is referred to the literature on kernel functions, e.g., [124, p. 479].

Any SVM is parametrized by the normal vector $\mathbf{w}$ and the offset $w_0$ of its hyperplane. If these two values have been determined, the SVM can predict the label $\hat{y}$ of a new sample $\mathbf{x}^*$ as follows:

$$\hat{y} = \operatorname{sgn}\left(\mathbf{w}^T\mathbf{x}^* + w_0\right). \tag{11}$$

For obtaining the parameters of the hyperplane from the training samples, the margins for the two classes are first defined as $\mathbf{w}^T\mathbf{x} + w_0 = 1$ and $\mathbf{w}^T\mathbf{x} + w_0 = -1$ respectively. The distance between these two margin hyperplanes can be shown to be $\frac{2}{||\mathbf{w}||}$ (cf. e.g. [124, p. 501]). Instead of maximizing this distance, one can also minimize the inverse of it. The optimization process must, of course, be constrained to account for the data at hand: Recalling that the margins define the edge for the respective classes, the hyperplane obtained from optimization must fulfill the following constraint (cf. e.g. [19, p. 328]):

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + w_0\right) - 1 \geq 0 \quad \forall\, i \in \{1, \ldots, n_s\}. \tag{12}$$

In order to account for potential outliers as explained above, so-called *slack variables* are introduced into the constraints of the optimization problem. These slack variables allow the hyperplane to be placed such that samples of a specific class are outside of their class margin (cf. e.g. [19, p. 332]):

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + w_0\right) - 1 + \xi_i \geq 0 \quad \forall\, i \in \{1, \ldots, n_s\}. \tag{13}$$

The slack variables $\xi_i$ need to be positive, which is another constraint for the optimization. Using Lagrange multipliers, the constraint optimization problem for finding the normal vector and offset of the hyperplane can be defined as follows:

$$\begin{aligned}\hat{\mathbf{w}}, \hat{w_0}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}} = \underset{\mathbf{w}, w_0, \boldsymbol{\alpha}, \boldsymbol{\beta}}{\operatorname{argmin}}\ & \frac{1}{2}\,||\mathbf{w}||^2 + C\sum_{i=1}^{\mathsf{N}_s}\xi_i \\ & + \sum_{i=1}^{\mathsf{N}_s}\alpha_i\left(y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + w_0\right) - 1\right) \\ & + \sum_{i=1}^{\mathsf{N}_s}\beta_i\xi_i.\end{aligned} \tag{14}$$

In the optimization, the positive scalar $C$ is a meta-parameter which defines how the slack variables will be penalized. If $C$ is large, the total sum of slack variables may only be small meaning that only few training sample misclassifications are allowed (cf. e.g. [19, p. 332]). In contrast, a $C$ close to zero would allow the hyperplane of the SVM to be placed almost arbitrarily.

It can be shown that the resulting estimate $\hat{\mathbf{w}}$ for the hyperplane is a linear combination of the training samples with the Lagrange multipliers $\boldsymbol{\alpha}$ serving as weights for the individual samples (cf. e.g. [124, p. 499]):

$$\hat{\mathbf{w}} = \sum_{i=1}^{\mathsf{N}_s}\alpha_i\mathbf{x}^{(i)}. \tag{15}$$

Moreover, $\alpha_i$ is only non-zero for those training samples $\mathbf{x}^{(i)}$ which reside on or beyond their respective margin. These samples are called *support vectors*. In Fig. 4, the support vectors for the two classes are depicted by a red outline around the respective data samples.

## 2.3 ZERO-SHOT LEARNING WITH ATTRIBUTES

The classifiers mentioned in the previous section are all able to classify data samples into one of $k$ classes. As hinted at before, they are faced with a problem when being presented with samples of classes which were not present during the training. These classes are generally referred to as unknown [96]. The problem arises due to the classifiers only being able to predict one of the known classes. Even if a specific classifier is able to reject a prediction through some form of thresholding, determining suitable threshold values is a cumbersome task, especially when considering that unknown classes are by definition unavailable during the time the classifier is trained. The problem discussed above can be mitigated if information about unknown classes can be incorporated into a classifier without requiring samples of these classes. Methods which follow this paradigm belong to the field of zero-shot learning [96]. For zero-shot learning, classes are represented by a set of semantic properties instead of a scalar label [73]. These properties may be shared between classes. It is, however, required that each class is represented by a unique set of properties and that no two classes have the same representation. When presented with new samples, classification can be performed by first predicting the set of properties for the given sample and then determining the class for which this intermediate encoding fits the best. Using the zero-shot learning framework, unknown classes can be made available for classification without requiring any training samples if an expert can supply the set of semantic properties for the new class. In the following, these classes will be referred to as *zero-shot classes*.

For computer vision problems, the semantic properties used in zero-shot learning are known as *attributes* [73]. The concept of attributes was independently proposed by Lampert *et al.* [95, 96] and Farhadi *et al.* [43]. Lampert *et al.* define an attribute as a characteristic of an object for which a human has the ability to decide if it is present or not. For example, if the object class is *banana*, possible attributes would be *yellow, fruit, curved* or *grows on trees*. It is important to note, that attributes do not necessarily need to be visually assessable[1]. As an example for this, Lampert *et al.* [96] list the habitat of an animal. An important aspect of attributes is that they are shared among a subset of classes [43]. This way, when learning attributes for known classes, an attribute classifier should be able to transfer knowledge to unknown classes which share the respective attributes and be able to predict the respective attributes for the unknown classes also. Zero-shot learning using attributes can thus also be viewed as a transfer learning problem [93, 153].

The definition of attributes by Lampert *et al.* leaves a certain leeway for interpretation. In particular, the definition does not state whether attributes are assigned to classes or image instances. Although it may seem as classes and image instances can be considered equally, there is an important difference between the two. If attributes are simply labels for a given image, finding these attributes is a problem known for decades as multi-label classification. Multi-label classification describes the problem of assigning $k$ out of $n$ possible labels to a given input sample (cf. e.g. [57, 115, 144]). Predicting attributes would thus be nothing else than multi-label classification. In contrast, if an attribute belongs to a class the focus of the problem shifts. In this scenario, each image instance does not carry multiple but only a single class label from which attributes can be derived. Farhadi *et al.* [43] use attributes under this second paradigm, i.e., as belonging to a class. They make this explicit as they claim to use attributes in order to effectively recognize object categories. Lampert *et al.* [96] also use attributes in this sense, i.e., in order to recognize object classes. The computer vision community in general, however, has adopted a looser interpretation of what defines an attribute. A prominent example for this is facial attribute prediction which has received considerable research interest lately, e.g., [109, 161, 215]. While the attributes used here are also assigned to classes, the attribute vector for a given class may not be unique. It is rather used as a so-called soft biometric in order to increase the performance of traditional face recognition methods by including a number of additional clues, i.e., attributes, for the face to be classified [120]. The attributes used here can thus rather

1 For this very reason, the term *visual attributes* will not be used in this thesis although it can also be found in the literature for describing attributes as well (cf. e.g. [133])

be interpreted as an additional multi-label annotation for a given face image. Rudd *et al.* [161] agree on this fact as they state "that facial attribute recognition inherently seeks multiple labels for the same image". Another area of computer vision where attributes are used is the classification of scene images. Here, the term attribute again refers to a multi-label annotation for a given scene image with no unique representations of scene classes through attributes [136].

As can be seen, attribute classification is sometimes used as a synonym for multi-label classification. This is in line with the definition by Sharif Razavian *et al.* [176] who consider an "attribute within the context of computer vision [...] as some semantic or abstract quality which different instances/categories share". In order to discriminate between the two definitions, attributes as defined by Lampert *et al.* [96] will be referred to as *class attributes* and those as defined by Sharif Razavian *et al.* [176] as *instance attributes* in this thesis. If no further qualification is given, the term attributes refers to class attributes in the following. Despite the different definitions for attributes, it should be noted that methods for predicting class attributes can often times be used for instance attributes as well and vice versa: As each sample is annotated with a binary vector $\mathbf{y} \in \{0, 1\}^n$, predicting class and instance attributes can be done using the same methods from a technical point of view.

For classifying data based on attributes, a mapping from attribute representation to the desired classes has to be defined. Typically, this mapping is obtained by a human expert establishing the relationships. However, recent work seeks to find these relationships in an unsupervised fashion by mining information from encyclopedias such as Wikipedia [5]. Besides allowing for zero-shot learning, attributes also enable a classifier to predict unusual attributes, i.e., unusual characteristics of known classes [43].

According to the definition by Lampert *et al.* [96], attributes are binary values. An oft-used approach is thus to use a set of SVMs for predicting the attribute representation for a given data sample (cf. e.g. [43, 73, 93, 96, 153]). For this, there exists one SVM per attribute which is responsible for predicting the presence or absence of this attribute. Having predicted the attribute representation, the class prediction can be done in different ways: Farhadi *et al.* [43] compute the distance from the predicted attribute representation to the available known encodings which are made up of the attribute representation known from the training samples and those supplied for the zero-shot classes. The class of the nearest known encoding is then used as prediction for the given sample. In contrast to Farhadi *et al.*, Lampert *et al.* [96] propose a probabilistic approach for predicting classes from attributes which they term Direct Attribute Prediction (DAP)[2]. The DAP approach is visualized in Fig. 5. The basis for DAP is a joint class label set $\mathsf{C}$ which consists of all unique class labels $y$ of the known classes and all unique class labels $z$ for zero-shot classes. The first step is then to obtain a vector $\mathbf{a}$ of attribute predictions for a given sample $\mathbf{x}$ from the attribute classifier. In order to obtain a class prediction from the predicted attribute vector, the attribute classifier has to be able to predict the probabilities $p\,(a_i \,|\, \mathbf{x})$ for each

2 Lampert *et al.* [96] also propose a second approach for predicting classes from attributes which they call Indirect Attribute Prediction (IAP). IAP is, however, rarely used in the literature and has no relevance for this thesis.

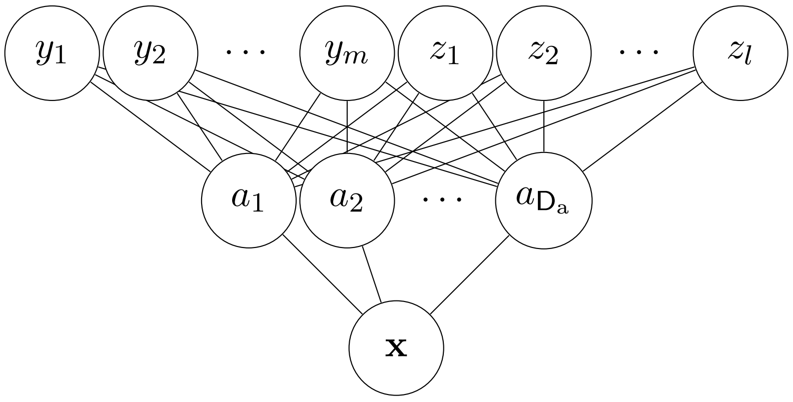


Figure 5: Visualization of the DAP method: First, the probability for each of the $\mathsf{D_a}$ attributes to be present given sample $\mathbf{x}$ is predicted from an attribute classifier. These probabilities are then used in order to predict probabilities for the known classes $y_i$ as well as the zero-shot classes $z_i$ for which there do not exist any training samples (figure recreated from Fig. 2 in [96]).

attribute $a_i$ being present given the sample $\mathbf{x}$. The class $\hat{c}$ can then be predicted from $\mathbf{a}$ through

$$\hat{c} = \operatorname*{argmax}_{c \in \mathsf{C}} p\left(c \,|\, \mathbf{x}\right) \tag{16}$$

$$= \operatorname*{argmax}_{c \in \mathsf{C}} \frac{p(c)}{p\left(\mathbf{a}^{(c)}\right)} \prod_{i=1}^{\mathsf{D_a}} p\left(a_i = a_i^{(c)} \,\middle|\, \mathbf{x}\right) \quad \text{with} \tag{17}$$

$$p\left(a_i = a_i^{(c)} \,\middle|\, \mathbf{x}\right) = \begin{cases} p\left(a_i \,|\, \mathbf{x}\right) & \text{if } a_i^{(c)} = 1 \\ 1 - p\left(a_i \,|\, \mathbf{x}\right) & \text{otherwise} \end{cases} \tag{18}$$

where $\mathsf{D_a}$ is the dimensionality of the attribute representation and $a_i^{(c)} \in 0, 1$ defines whether the $i$-th attribute for class $c$ is present or not. If no further knowledge about the known and unknown classes is available, the class prior $p(c)$ is set to a uniform value and may hence be disregarded when optimizing the posterior probability. The same applies for the prior $p\left(\mathbf{a}^{(c)}\right)$ for the different attribute vectors.

Class attributes are a powerful tool which have been successfully used for a variety of different computer vision applications such as human action recognition [108, 211], object retrieval [38, 96], scene recognition [104] and clothing recognition [192]. They were also used for determining whether a human finds an image aesthetically pleasing or not [36]. Besides the aforementioned facial attribute recognition, instance attributes have successfully been used for predicting characteristics of natural scene images [136] and human poses [214]. In all of the applications mentioned above, attributes are predicted for a single image. An interesting expansion to this is to define attributes as relations between two images [134]. This way, an attribute is not detected as present or not but it is rather determined whether a specific attribute applies more to a given image than another. Relative attributes have successfully been used for product retrieval, e.g., [87, 88]. For example, if the user queries a specific shoe, he or she can then decide to retrieve other shoes which have a higher heel than the current one. In this case, the relative attribute would be *has higher heel*.

Attributes have also been used for document image analysis applications. Works in these regard have a higher relevance than from other fields of computer vision and will be discussed separately in Chap. 6.

# 3 NEURAL NETWORKS AND DEEP LEARNING

After explaining general concepts of computer vision in the previous chapter, this chapter gives a detailed overview of neural networks as these machine learning models play an integral part in the methodology presented later in this thesis. First, concepts for feedforward neural networks are presented in Sec. 3.1 as they represent the fundamentals of neural networks used in this thesis. The ensuing Sec. 3.2 then elaborates on how to train feedforward architectures. Afterwards, Sec. 3.3 introduces Convolutional Neural Networks (CNNs) which serve as the cornerstone of the presented methodology. Finally, Sec. 3.4 and Sec. 3.5 discuss the field of deep learning and the use of increasingly deeper models.

In the following, the terms neural network and network will be used interchangeably. These terms refer strictly to the neural network models known from pattern recognition and machine learning and not any circuit networks from electrical engineering or models of the brain.

## 3.1 FEEDFORWARD NEURAL NETWORKS

Feedforward neural networks build an important class of neural networks with some authors even suggesting that they are the "quintessential deep learning models" [58, p. 168]. The are made up of computational blocks which are stacked up in a hierarchical order and form a directed and acyclic graph.

Work on feedforward neural networks can be traced back at least as far as the 1950s. As the name suggests, the main inspiration was mimicking the structures of the human brain. This way, abstract neuron models could be created which where able to approximate binary functions and later real-valued functions as well. It should be noted, though, that neural networks are only loosely inspired by the human brain and especially do not copy its functionally.

A number of concepts that where developed with the early neural networks models are still used in todays deep learning architectures which is why they will be presented in detail here. First, this section discusses the Perceptron in Sec. 3.1.1. Then, the Multilayer Perceptron is presented in Sec. 3.1.2.

### 3.1.1 *Perceptron*

The Perceptron model by Rosenblatt [155] is widely regarded as one of the most influential early works on neural networks. A graphical outline of a Perceptron is given in Fig. 6. The Perceptron classifies input samples $\mathbf{x} \in \mathbb{R}^{\mathsf{D}}$ into either of two classes $y \in \{-1, 1\}$. This is done by determining the sign of a weighted sum of the inputs. The classification rule for the Perceptron can formally be defined as

$$y(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + b > 0 \\ -1 & \text{otherwise} \end{cases} \tag{19}$$

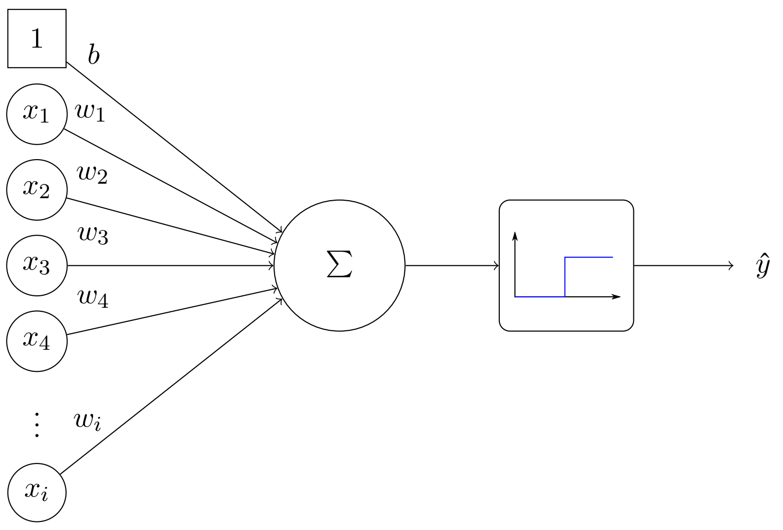


Figure 6: The Perceptron model in its "updated" version as proposed by Minsky and Papert [122]. The input to the model is a real-valued vector. A weighted linear combination of the vector elements is forwarded to a step function which effectively extracts the sign of the linear combination. This is the class $\hat{y}$ predicted from the Perceptron.

where $\mathbf{w}$ is the vector of weights for the input and $b$ is a bias. Originally, the input to the Perceptron was restricted to binary vectors [155]. Contemporary literature, however, usually refers to the "updated" version by Minsky and Papert [122] which handles real-valued vectors as well.

A Perceptron shares a number of characteristics with a Support Vector Machine (SVM) (cf. Sec. 2.2). Just as an SVM, a Perceptron is only able to perform binary classification. Upgrading a Perceptron in order to perform multi-class classification can be achieved the same way as is done for SVMs, i.e., through one-vs-all or one-vs-one classification. Another similarity is how an SVM and a Perceptron determine the class prediction for a given sample. For the Perceptron, $\mathbf{w}$ represents the normal vector of a hyperplane which defines the boundary between the two classes. Using a hyperplane for classification is done in SVMs as well. The major difference of the two models lies in the way they are trained, i.e., fitted to data. While the optimization of an SVM model makes use of the max-margin criterion in order to find the separating hyperplane, the Perceptron uses a per-sample classification result as optimization criterion which is also referred to as *perceptron criterion* [19, p. 193]. For this, each sample $\mathbf{x}$ and its corresponding label $y$ is processed by the Perceptron individually and in an iterative fashion. If the criterion

$$\mathbf{w}^T\mathbf{x} \cdot y > 0 \tag{20}$$

is satisfied, $\mathbf{w}$ is not changed. Otherwise, the weight vector $\mathbf{w}^{(\tau)}$ at iteration $\tau$ is updated as follows:

$$\mathbf{w}^{(\tau)} \leftarrow \mathbf{w}^{(\tau-1)} + y \cdot \mathbf{x}. \tag{21}$$

This algorithm is guaranteed to converge to an exact solution if the training samples are linearly separable [122, 156].

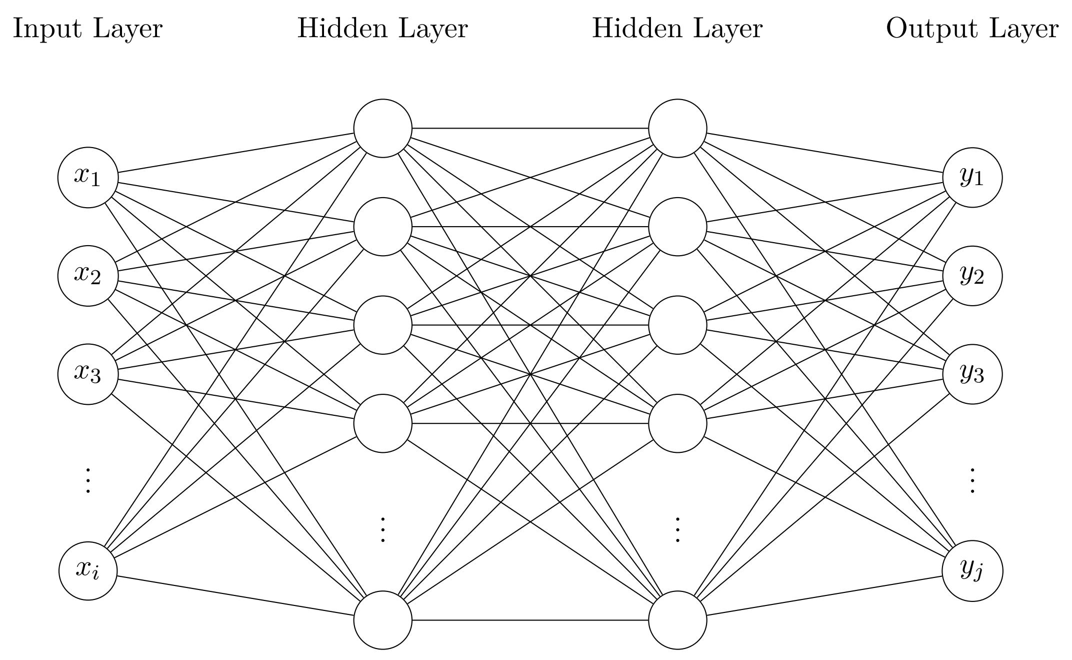


Figure 7: The figure visualizes an MLP. Except for the input, each layer is fully connected, meaning that each neuron is connected to all neurons in the preceding layer. All inner layers of the MLP are called hidden layers, while the last layer is called output layer.

### 3.1.2 *Multilayer Perceptron*

The main drawback of a Perceptron is its inability to handle non-linearly separable data. This was famously documented by Minsky and Papert [122] in showing the Perceptron's inability to represent the XOR-function. However, Minsky and Papert went on to show that this property could be remedied by stacking multiple layers of Perceptrons. This approach gave rise to a neural network model which is known as Multilayer Perceptron (MLP)

An MLP consists of a number of sequentially stacked Perceptrons. Fig. 7 exemplarily visualizes a four layer MLP. The first layer simply delivers the input $\mathbf{x}$ to the network and is called *input layer*. The ensuing layers all compute intermediate representations with the last layer (the *output layer*) finally producing the output $\hat{\mathbf{y}}$ of the MLP. All layers beside the input and output layer are called *hidden layers*. In general, the MLP is a feedforward network [19, p. 229] as the output of any layer does only depend on the output of the immediately preceeding layer (except for the input layer).

As each layer $s$ in an MLP is a Perceptron, the layer-wise output is computed the same as is done in a multi-class Perceptron: First the weighted sum

$$\mathbf{i}^{(s)}\left(\mathbf{x}, \mathbf{W}^{(s)}, \mathbf{b}^{(s)}\right) = \mathbf{W}^{(s)}\mathbf{x} + \mathbf{b}^{(s)} \tag{22}$$

of the layer's input vector is calculated. For simplicity, the input vector $\mathbf{x}$ can be augmented by a single element with value 1. This way, the bias is encoded into the weight matrix which greatly simplifies notation:

$$\mathbf{i}^{(s)}\left(\mathbf{x}, \mathbf{W}^{(s)}\right) = \mathbf{W}^{(s)}\mathbf{x}. \tag{23}$$

The result of the weighted sum of inputs is then forwarded to a nonlinear activation function

$$\mathbf{f}^{(s)}\left(\mathbf{x}, \mathbf{W}^{(s)}\right) = \Theta\left(\mathbf{i}^{(s)}\left(\mathbf{x}, \mathbf{W}^{(s)}\right)\right) \tag{24}$$

in order to compute the layer's output $\mathbf{f}^{(s)}$. It is necessary that $\Theta$ is nonlinear as otherwise the MLP can be represented by a single linear function and thus be no more expressive than a linear classifier. In contrast to the step function used in the Perceptron, MLPs traditionally made use of the sigmoid function

$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}. \tag{25}$$

The sigmoid can be interpreted as a soft step function which has the property of being differentiable everywhere. This is an important aspect for training MLPs which will be further discussed in Sec. 3.2.

Computing the output $\hat{\mathbf{y}}$ of the MLP is done by a so-called *forward pass* of the input data through the network (weight arguments left out for cleaner visualization):

$$\hat{\mathbf{y}} = \mathbf{f}^{(\mathsf{N}_l)}\left(\mathbf{f}^{(\mathsf{N}_l - 1)}\left(\ldots \mathbf{f}^{(2)}\left(\mathbf{f}^{(1)}\left(\mathbf{x}\right)\right)\right)\right). \tag{26}$$

Here, $\mathsf{N}_l$ represents the number of layers in the MLP. There are $\mathsf{N}_l - 1$ hidden layers with $\mathbf{f}^{(\mathsf{N}_l)}$ serving as the output layer.

An interesting theoretical aspect of MLPs is, that an MLP with two layers containing weights, i.e., one hidden and one output layer, is able to approximate any Borel measurable function with arbitrary accuracy [71]. As all functions $\mathbf{g} : \mathbb{R}^{\mathsf{D}} \rightarrow \mathbb{R}^{\mathsf{D}'}$ are Borel measurable, an MLP is thus theoretically able to approximate all real-valued multivariate functions [170]. The proof concerning this characteristic is known as the *universal approximation theorem* [71].

## 3.2 TRAINING FEEDFORWARD NETWORKS

Although MLPs can theoretically approximate any function, finding the correct weight configuration to represent a specific function with desired accuracy is non-trivial and an open research question. In essence, one would like to accurately approximate the correct classification rule for a given classification problem. This is, of course, impossible to achieve since the true data distribution and the corresponding classification rule can generally not be determined. Hence, neural networks have to approximate the correct classification rule based on a set of training samples they are supplied with.

While a Perceptron learns its classification rule using the perceptron criterion, this criterion can no longer be used when dealing with MLPs. Instead, the standard approach is to minimize the difference between the output obtained for a specific input and the desired output with respect to the weights of the MLP. Fitting a neural network to data this way is called *training* or *learning*. While this approach has been used for training neural networks since at least the 1980's, it is still the de-facto standard for supervised training of modern neural networks. This section gives a detailed description of this standard training procedure for feedforward networks.

### 3.2.1 *Gradient Descent and Error Backpropagation*

The goal of training is to find the weights and biases such that a neural network produces a desired output $\mathbf{y}$ for a given input $\mathbf{x}$. For this, a so-called loss function $l$ is required.

This function computes a numerical value that represents a measure of deviation of the network's prediction $\hat{\mathbf{y}}$ from $\mathbf{y}$. A straight forward loss function is the Euclidean Loss:

$$l_{\mathcal{N}}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{\mathsf{D}} \frac{1}{2} (\hat{y}_i - y_i)^2 . \tag{27}$$

The loss here is simply the squared Euclidean distance of $\hat{\mathbf{y}}$ and $\mathbf{y}$ scaled by a factor.

Having defined a loss function, training a neural network is then achieved by adapting the weights of the network such that $l$ and thus the deviation is minimized. Unfortunately, finding an analytic solution to the optimization problem is cumbersome at best and most often simply impossible, especially with increasing depth of the network to be trained. Thus, the training procedure resorts to an iterative update of the weights known as *gradient descent*: The weight $w_{i,j}^{(s)}$ in layer $s$ is updated by adding a fraction of the negative gradient of the loss with respect to $w_{i,j}^{(s)}$:

$$w_{i,j}^{(s)} \leftarrow w_{i,j}^{(s)} - \eta \frac{\partial l}{\partial w_{i,j}^{(s)}} = w_{i,j}^{(s)} + \Delta w_{i,j}^{(s)} . \tag{28}$$

This way, the gradient descent algorithm iteratively finds a local minimum for $l$. The meta-parameter $\eta$ is called *learning rate* and is critical for the optimization process. If chosen too large, the optimization might oscillate or even diverge. On the other hand, a small learning rate might require a large amount of iterations for the algorithm to converge to a local minimum. In order to stabilize the training process, Rumelhart *et al.* [162] propose to add an inertia term into the optimization: At iteration $\tau$ the weight update $\Delta w_{i,j}^{(s)}(\tau)$ is not only determined by the current gradient but also by a fraction of the previous weight update:

$$\Delta w_{i,j}^{(s)}(\tau) = -\eta \frac{\partial l}{\partial w_{i,j}^{(s)}(\tau)} + \alpha \cdot \Delta w_{i,j}^{(s)}(\tau - 1). \tag{29}$$

Here, $\alpha$ is called *momentum* and is another meta-parameter for gradient descent. It can be shown empirically that incorporating the momentum term makes optimization less sensible to otherwise disadvantageous learning rate values and greatly speeds up the training process [162].

The above formulations suffice for training a neural network. However, it requires computing the gradient $\frac{\partial l}{\partial w_{i,j}^{(s)}}$ which is straight forward for the weights in the output layer but less so for the weights in the hidden layers. An important insight is, that a neural network computes its output through a composition of functions $\mathbf{f}^{(s)}$ (cf. Eq. 26). Thus, the gradient of the loss function with respect to a weight $w_{i,j}^{(s)}$ in layer $s$ can be rewritten by applying the chain rule:

$$\frac{\partial l}{\partial w_{i,j}^{(s)}} = \frac{\partial l}{\partial \mathbf{f}^{(\mathsf{N}_l)}} \cdot \frac{\partial \mathbf{f}^{(\mathsf{N}_l)}}{\partial \mathbf{f}^{(\mathsf{N}_l - 1)}} \cdot \ldots \cdot \frac{\partial \mathbf{f}^{(s+1)}}{\partial \mathbf{f}^{(s)}} \cdot \frac{\partial \mathbf{f}^{(s)}}{\partial \mathbf{i}^{(s)}} \cdot \frac{\partial \mathbf{i}^{(s)}}{\partial w_{i,j}^{(s)}} \tag{30}$$

$$= \frac{\partial l}{\partial \mathbf{f}^{(\mathsf{N}_l)}} \cdot \left\{ \prod_{k=s+1}^{\mathsf{N}_l} \frac{\partial \mathbf{f}^{(k)}}{\partial \mathbf{f}^{(k-1)}} \right\} \cdot \frac{\partial \mathbf{f}^{(s)}}{\partial \mathbf{i}^{(s)}} \cdot \frac{\partial \mathbf{i}^{(s)}}{\partial w_{i,j}^{(s)}} \tag{31}$$

The last factor in Eq. 31 is the gradient of the desired weight with respect to the linear output, i.e., before applying the activation function. The first two factors in Eq. 31 effectively represent the gradient of the loss with respect to the previous layer $s + 1$. This implies that for computing the gradient of any weight $w_{i,j}^{(s)}$ one needs to first find the gradient for the previous layer. Hence, for updating all the weights in the network the gradients are computed starting at the last layer and ending at the first layer as follows: The output layer computes the gradient of its weights directly from the loss function. It also computes the gradient with respect to its input and sends this gradient "back" to its preceding layer. Thus the preceding layer is handed the gradient for its output. All hidden layers then compute their gradients the same way: Use the gradient of the respective output (i.e. the gradient handed back from the succeeding layer) in order to determine the gradient of the loss with respect to the weights and the input of this layer. The gradient with respect to the input is then propagated back to the preceding layer. Computing the gradients of the weights of a neural network this way is known as *error backpropagation* or simply *backpropagation*. Although first versions of this algorithm were proposed as early as 1974 [202], the work by Rumelhart *et al.* [162] is widely considered as being most comprehensive and thorough with respect to backpropagation.

The formulation of the backpropagation algorithm in Eq. 31 shows why using the step function as activation function $\Theta$ as is done for the Perceptron is not a feasible approach when training a network with gradient descent as was claimed earlier: The gradient of the step function is 0 everywhere except for when the argument is 0 for which the gradient is not defined. While technically a gradient could simply be defined for this specific argument, training a neural network would still not work as the gradient would be 0 everywhere else. Thus $\frac{\partial \mathbf{f}^{(s)}}{\partial \mathbf{i}^{(s)}}$ in Eq. 31 would always be 0 as well. As $\frac{\partial \mathbf{f}^{(s)}}{\partial \mathbf{i}^{(s)}}$ is a factor in the gradient computations, the overall gradient would be the zero-vector and the weights would never receive any updates.

### 3.2.2 *Stochastic Gradient Descent*

The description of the training of neural networks so far only considered a single data point for training. In order to train a neural network with a number of data samples, Rumelhart *et al.* [162] initially proposed to compute the gradient for each sample individually and then update the weights of the network according to the mean of all individual gradients. This training procedure is referred to as *batch training* [100]. One major drawback of this approach is that computing the gradients for the entire training set can be very time consuming if the amount of training samples is large.

A possible alternative to batch training is using a small, randomly drawn subset of training samples in order to estimate the true gradient for updating the weights. Using this approach, gradient computations and hence training can be sped up substantially. This concept is known as Stochastic Gradient Descent (SGD) or *stochastic learning*. In its most extreme case, a single sample is used for approximating the "true" gradient. This approach is also known as *online learning*.

While earlier works define stochastic learning as online learning, e.g. [20, 101], stochastic learning nowadays is usually defined as using a subset larger than one for computing the gradient estimate. In the context of Stochastic Gradient Descent (SGD), this subset is referred to as *mini-batch*. While LeCun *et al.* [101] suggest, that the mini-batch size is set to small values in the beginning of training and then gradually increased, it is common practice nowadays to simply define a mini-batch size and keep it fixed during the entire training process (cf. e.g. [92, 179]).

Stochastic gradient descent has a number of advantages over batch training. One of them is the increased number of updates per weight given a number of training samples. Apart from this, however, there exist structural advantages as well. In order to explain them, one has to visualize the loss to be optimized as a function of the weights of the network. This function is highly non-convex and usually exhibits a large amount of local minima (cf. e.g. [42]). When training with batch gradient descent, the optimization process always finds the minimum corresponding to the basin of extraction of the initial weights [101]. SGD, however, allows for the optimization to "hop" out of the current basin of attraction as the gradient used is only a noisy approximation. However, in order to not fall victim to too noisy gradients and also to have the algorithm converge eventually, the use of momentum during training is practically mandatory when performing SGD.

Besides its intuitive characteristics, SGD has a number of interesting theoretical properties. For example, it can be proven that it almost always finds a local minimum despite its stochastic nature [103]. Moreover, SGD is not affected by saddle points [131].

The size of a mini-batch is a crucial meta-parameter when training neural networks with SGD. Recent work suggests that when the mini-batch size is too big the optimization process finds a local minimum with bad generalization characteristics [81]. This means, that, although the labels for the training samples are predicted with high accuracy, the accuracy on the test samples is generally poor.

## 3.3 CONVOLUTIONAL NEURAL NETWORKS

A basic aspect of MLPs is that each layer is fully connected to the preceeding layer (cf. Sec. 3.1). When dealing with images as input data as is done in this thesis, this approach has the notable drawback that the network requires a large amount of parameters in the first layer. For example, using an image size of $28 \times 28$ pixels[1] and a relatively small first hidden layer of 500 neurons, the layer would require 392 000 parameters. Having this many parameters in the first layer alone would require a large amount of annotated training samples for the network not to overfit during training. Additionally, the network could not exploit the knowledge that objects can appear at different positions in the image. For example, if a neural network learned to detect cars in the upper left corner of an image, it would need to again learn a set of weights for detecting a car if it appeared in the upper right corner of an image. Ideally, one would like to incorporate translation invariance here in the sense that a neural network is able to learn a concept, i.e., detect an object, and find this learned concept in arbitrary parts of the supplied image. This is exactly the approach used in modern architectures known as Convolutional Neural Networks (CNNs). In comparison to MLPs, CNNs do not exhibit a full connection to the previous layer but rather use small filters or kernels of trainable weights to slide over the input. These special types of neural networks were originally proposed by Fukushima [48] under the name of Neocognitron but did not receive major attention until made famous by LeCun *et al.* [99] as CNNs.

The main building blocks of CNNs are *convolutional layers*. Fig. 8 exemplarily visualizes such a layer. Each convolutional layer is made up of $n$ different filters and a bias $b$ for each filter. It is common practice to chose a fixed kernel size for all filters in a single convolutional layer. This way, the learnable parameters of the layer can be represented by a tensor $\mathbf{W}$ and a vector of biases $\mathbf{b}$. During the forward pass, the input image $\mathbf{I}_{h,w,c}$ of height $h$, width

1 $28 \times 28$ pixels is the image size for the well-known MNIST dataset [101]

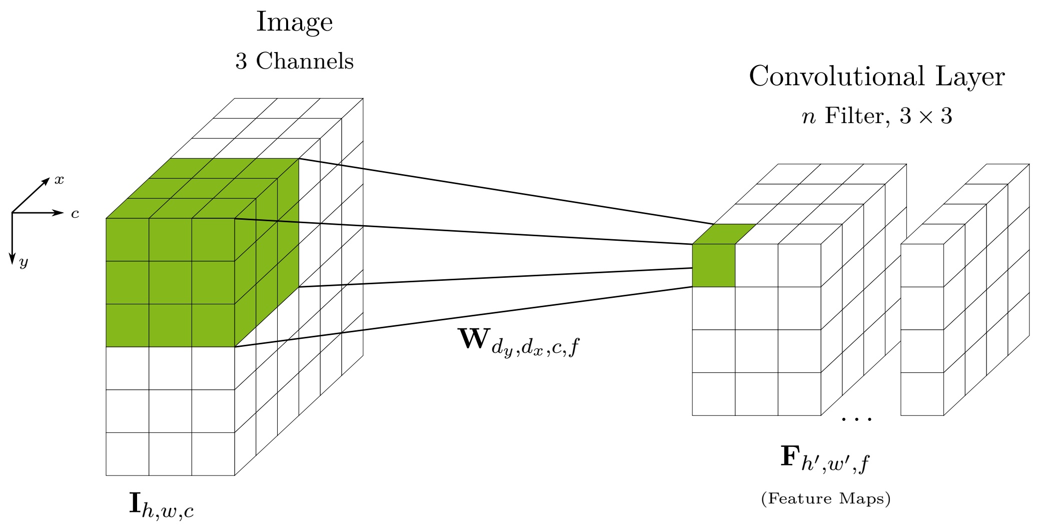


Figure 8: The figure depicts a convolutional layer using $n$ different $3 \times 3$ filters. The weights of the layer are combined into the four dimensional tensor $\mathbf{W}$. The filters produce feature maps $\mathbf{F}_{h',w',f}$ from the input image $\mathbf{I}$.

$w$ and $c$ channels is discretely convolved with the different filters to produce $n$ distinct outputs which in this context are known as *feature maps*. Each convolution is only applied in a small spatial region but spans across all channels of the input[2]. More formally, the point $x, y$ of the $f$-th feature map is computed by

$$\mathbf{F}_{x,y,f} = \Theta\left(\sum_{c=1}^{\mathsf{K}} \sum_{i=-\lfloor d_y/2 \rfloor}^{\lceil d_y/2 \rceil - 1} \sum_{j=-\lfloor d_x/2 \rfloor}^{\lceil d_x/2 \rceil - 1} \mathbf{W}_{i,j,c,f} \cdot \mathbf{I}_{x+i,y+j,c} + b_f\right), \tag{32}$$

where $d_x$ and $d_y$ are the width and height of the filter $f$ and $b_f$ its corresponding bias and $\mathsf{K}$ the number of channels in the input to the convolutional layer. Typically, a convolutional layer makes use of square filter kernels, i.e., $d_x$ and $d_y$ are equal. As can be seen from the equation, a feature map value is undefined when the corresponding filter kernel would be placed over pixels which are outside of the input. For these special situations, the behavior has to be defined by the user. A common approach is to only compute feature map values for which the corresponding filter does not exceed the bounds of the input. The drawback of this approach is, of course, that the resulting feature maps are smaller than the original image. Using square filters with a width and height of $d$, the feature maps have $2 \cdot \lfloor \frac{d}{2} \rfloor$ less pixels in height and width. In order to mitigate this effect, an oft-used approach is to pad the input along the horizontal and vertical axes by the amount of pixels that would otherwise by erased due to the convolution. While the padding values can principally be arbitrarily chosen, they are typicall set to 0.

Just as is done for layers used in an MLP, which are referred to as fully connected layers when comparing them to convolutional layers, the result of the discrete and linear convolution is forwarded to an activation function $\Theta$. This activation function is typically a scalar function and thus applied individually for each element of the output of the convolution in order to produce the resulting feature map.

2 There exists so-called *nd*-convolutions as well which take the channel dimension as depth of the supplied input and thus do not span the entire channels. These special convolutions, however, are not relevant for this thesis and will not be discussed further.

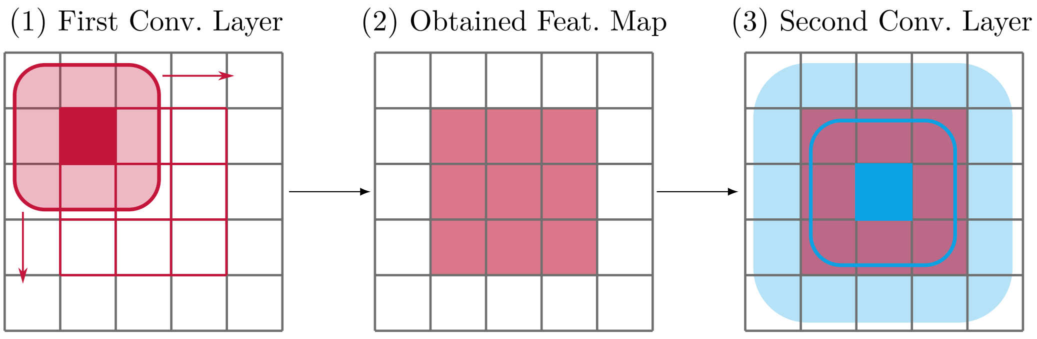


Figure 9: Visualization of the receptive fields for two consecutive convolutional layers. Each of the convolutional layers uses filter kernels of size $3 \times 3$. The first convolutional layer is applied to the input image (1). Here, the receptive field is simply the input pixels under the current filter kernel (light red). The resulting feature map (2) is then used as input for the second convolutional layer. Although the filter kernel is also of size $3 \times 3$ the receptive field is now the entire image.

Up to this point, the input to a convolutional layer was considered to be in image. Of course, convolutional layers can also accept the output of other convolutional layers, i.e., feature maps. This way, convolutional layers can be stacked to form parts of a neural network. The filters in the convolutional layers can be considered feature detectors that get activated by certain structures or objects in the image. It can be shown, that the layers close to the input image typically learn to detect color blobs and edges. The filters of layers which are further away from the input image are typically activated by objects parts or entire objects [180, 183, 213]. A CNN is thus able to learn hierarchical representations of the input data. They are most effective if such a hierarchy exists, meaning that the supplied data exhibits hierarchical characteristics. These characteristics are typically given for natural images where objects can be composed of object parts which in turn can be composed of edges and color blobs. The data used in this thesis are document images of hand written text, which also exhibit the desired characteristics: Word images can be decomposed into characters which in turn can be decomposed into strokes. Hence, it is a reasonable assumption that CNNs will fare well on document and word images, too.

An important concept for CNNs is the *receptive field*. It describes all pixels in the original image that influence the value of the current “pixel” in the output feature map of a specific layer. Fig. 9 visualizes this concept for two consecutive convolutional layers which use filter kernels of size $3 \times 3$ using an input image of $5 \times 5$ pixels. The feature map values for the first convolutional layer are only affected by the input image pixels which are within the view of the filter kernel (red rounded rectangle). The resulting feature map is then processed by the second convolutional layer. The filter kernels in this layer are also of size $3 \times 3$ (dark blue rounded rectangle). However, all pixels in the input image influence the feature map value (light blue filled rectangle). The receptive field is not only defined for each pixel in a respective feature map but also for a filter in a convolutional layer in general. Here, the term refers to the spatial extent a filter kernel is affected by in the input image. For example, the blue filter in Fig. 9 has a receptive field size of $5 \times 5$.

In general, the receptive field gradually increases for each convolutional layer in a CNN. In order to increase the receptive field size without adding more convolutional layers and also to introduce a certain amount of translation invariance in the intermediate feature maps, a common approach is to insert so-called *pooling layers* after a certain number

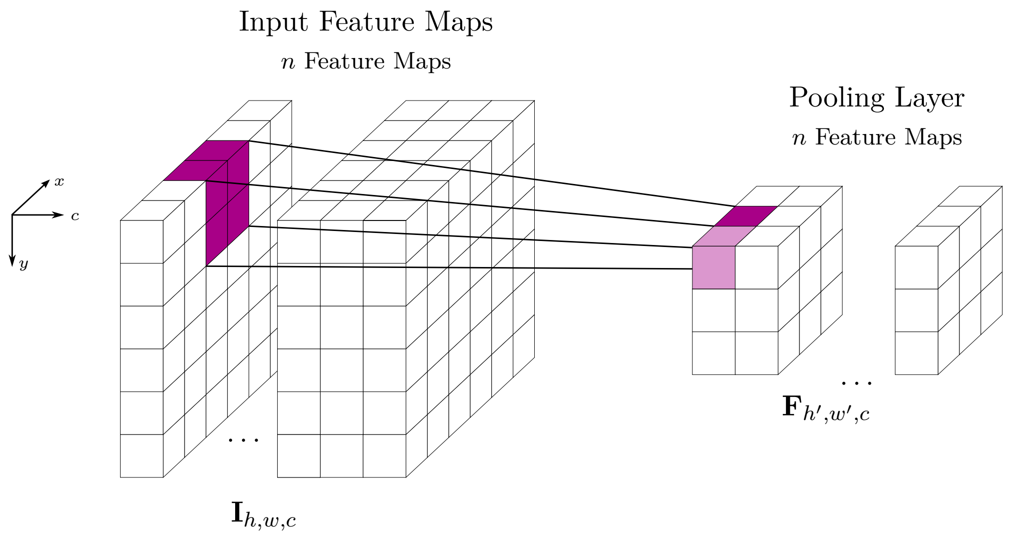


Figure 10: The figure displays a $2 \times 2$ pooling layer with a stride of two.

of convolutional layers. Pooling layers use feature maps as input and pool their values in predefined areas into single scalar value in order to produce an output feature map. Fig. 10 exemplarily visualizes such a pooling layer. Each $2 \times 2$ region in the input feature maps is pooled into a single value. The pooling regions are applied with a stride of two, meaning that the next pooling region has an offset of two pixels compared to the previous region. This is a common approach when applying pooling layers and effectively downsamples the input feature maps. This downsampling increases the receptive field size by a factor of two when using $2 \times 2$ pooling regions and a stride of two. The most commonly used pooling strategy is *max pooling*, e.g., [92, 179]. Here, the output feature map is computed by taking the maximum value from each pooling region in the input feature map. In average pooling, e.g., [107, 217] output feature map values are computed from averaging the values in the respective pooling regions. A strategy used less often is stochastic pooling [212]. Here, the values in each pooling region are first $L1$-normalized in order to form pseudo-probabilities. The pooling result is then determined by drawing from a multinomial distribution using the previously computed pseudo-probabilities as priors for selecting the respective location.

Convolutional and pooling layers form the so-called convolutional part of a CNN. When using CNNs for image classification, this convolutional part is typically connected to a neural network capable of classification, usually an MLP, e.g., [92, 179], or a Perceptron, e.g., [67, 193]. The convolutional part can thus be interpreted as being responsible for selecting suitable features from the input image which are then used as input to the succeeding classifier. This view bares a certain resemblance to the computer vision pipeline in traditional approaches (cf. Chap. 2). The difference is that the feature extractors (convolutional part) and the classifier (MLP, Perceptron) can be optimized in a combined fashion instead of optimizing each part individually. This concept is referred to as *end-to-end optimization* in the literature [101]. The term stems from the fact that a CNN is only presented with an image and its corresponding label during training and is tasked to find a good feature representation during the optimization process.

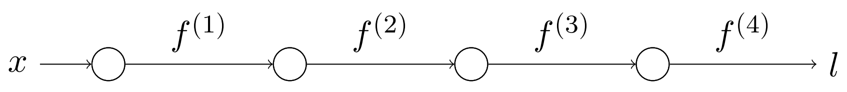


Figure 11: Network architecture for demonstrating the vanishing gradient problem. The input $x$ is forwarded through four layers containing neurons with sigmoid activation. The result from the last layer is then forwarded to a loss function $l$.

## 3.4 DEEP LEARNING AND THE VANISHING GRADIENT PROBLEM

CNNs are the backbone of computer vision methods from the field of *deep learning*. In the context of this thesis, deep learning is defined as follows: Deep learning refers to a field of research in pattern recognition and machine learning concerned with neural networks which encompass a large amount of layers. Viewing the layers as nodes in a graph, the depth of a neural network is the maximum length of any path from input to output. The term *deep* is used rather loosely and there exists no clear definition as to when a neural network can be regarded as deep. Bengio [16] defines deep architectures simply as having "many hidden layers". Glorot *et al.* [56] state that an architecture can be considered as deep if it has three or more hidden layers. Nowadays, deep neural networks are often times made up of tens or hundreds of layers (cf. e.g. [67, 92, 179]). These networks are generally regarded as deep.

While it can be shown empirically, that deep neural networks achieve better performance than other approaches in a vast amount of tasks, there exists no theoretical evidence concerning the reason for their current superiority. While the No Free Lunch theorem [210] forbids such evidence without knowledge of the data anyways, even when making assumptions on the data there exists no theoretical explanation. Most explanations rely on intuition and cite the ability to extract intricate structures in hierarchical representations of the supplied data [102].

Although a large depth was expected to be crucial for learning powerful representations long before the current success of deep architectures, e.g., [16], training deep architectures proved to be difficult. For these types of neural network, training requires a large amount of computational power and can only be handled effectively by Graphical Processing Units (GPUs). In addition, an increasing number of model parameters, i.e., weights, requires a substantial amount of annotated training samples in order to be optimized. But even after having satisfied both conditions, training deep neural networks suffered from a structural problem in the beginning which is known today as vanishing gradient problem. While it was first discovered for Recurrent Neural Networks (RNNs) [70], it occurs in deep feedforward architectures as well and can be illustrated by the following toy example[3]: Consider an MLP architecture with $\mathsf{N}_l$ layers, each consisting of a single neuron using the sigmoid activation function. Fig. 11 depicts such a network for the case of four layers. The output for a layer $i$ is computed by

$$f^{(i)}(x) = \sigma\left(w^{(i)} f^{(i-1)}(x) + b^{(i)}\right) \text{ with} \tag{33}$$

$$f^{(0)}(x) = x. \tag{34}$$

3 The example is inspired by a similar one presented in [128]

The gradients of $f^{(i)}$ with respect to the layer's input and the weight $w^{(i)}$ are

$$\frac{\partial f^{(i)}}{\partial f^{(i-1)}} = \sigma'\left(w^{(i)} f^{(i)} + b^{(i)}\right) \cdot w^{(i)} \text{ and} \tag{35}$$

$$\frac{\partial f^{(i)}}{\partial w^{(i)}} = \sigma'\left(w^{(i)} f^{(i-1)} + b^{(i)}\right) \cdot f^{(i-1)}. \tag{36}$$

The gradient of the loss with respect to an arbitrary weight $w^{(s)}$ is thus

$$\frac{\partial l}{\partial w^{(s)}} = \frac{\partial l}{\partial f^{(n)}} \cdot \prod_{i=s-1}^{\mathsf{N}_l} \left[\sigma'\left(w^{(i)} f^{(i-1)} + b^{(i)}\right) \cdot w^{(i)}\right] \cdot \sigma'\left(w^{(s)} f^{(s-1)} + b^{(s)}\right) \cdot f^{(s-1)}. \tag{37}$$

The derivative of the sigmoid function $\sigma'$ has its maximum at 0 where its value is $\frac{1}{4}$. When backpropagating the error, the gradient is thus scaled by a maximum value of $\frac{1}{4}$ for each layer:

$$\frac{\partial l}{\partial w^{(s)}} \leq \frac{\partial l}{\partial f^{(n)}} \cdot \prod_{s=i-1}^{\mathsf{N}_l} \left[w^{(i)}\right] \cdot f^{(s-1)} \cdot \frac{1}{4^s}. \tag{38}$$

While this scaling factor has a smaller influence on gradients for layers at the end of the network, the gradient almost vanishes for the layers at the beginning of the network due to the exponential decrease. Using the four-layer network from Fig. 11, the scaling factor is already at $\frac{1}{256}$ for the first layer. As Hochreiter [70] mentions, this problem can not be combated by an increased learning rate.

Initial attempts for solving the vanishing gradient problem didn't target it directly but rather used a form of unsupervised pretraining of the weights in order to have each layer output useful representations [69, 168]. These pretrained weights can then be used as initialization for supervised training of the neural network. This approach, however, does not mitigate the structural problem of vanishing gradients during the supervised training stage as the network still uses sigmoid activation functions.

A more direct approach for overcoming the vanishing gradient problem is to replace the sigmoid with a more suitable activation function for increasingly deeper architectures. The desired properties for such an activation function are that it does not scale the gradient but still is non-linear. A function that fits these criteria is the Rectified Linear Unit (ReLU). While originally proposed by Hahnloser *et al.* [63] for hardware circuits representing neural networks, it was made popular in machine learning and computer vision by Glorot *et al.* [56] and Nair and Hinton [126] for use in deep feedforward architectures. The Rectified Linear Unit (ReLU) is arguably one of the deciding factors for the current success of deep learning as it allows for architectures with many layers while not exhibiting the structural risk of a vanishing gradient[4].

The ReLU is defined as

$$f_{\text{ReLU}}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}. \tag{39}$$

When used in a neural network, it has the effect of linearly propagating values if they are positive. Fig. 12 visualizes this concept for an MLP with two hidden layers of six neurons each using ReLU activations. Neurons for which the weighted sum produces a negative result are "switched off" by the ReLU. These neurons are depicted in light gray in the

4 Gradients may still vanish due to disadvantageous initialization of the weights, see, e.g., [179]. However, the described scaling behavior is eliminated.

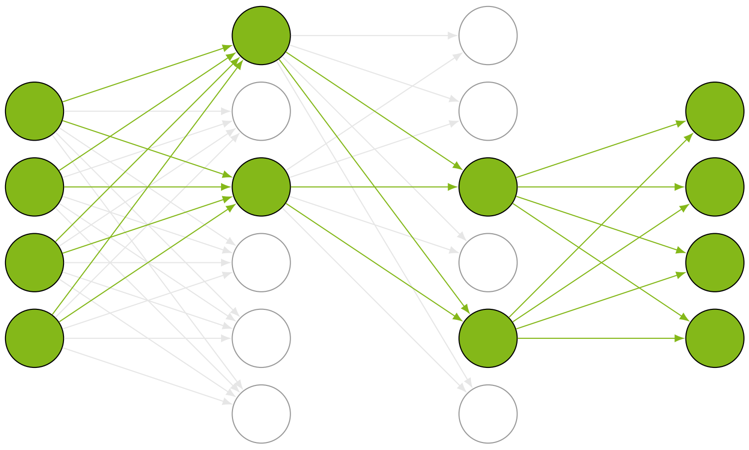


Figure 12: The figure visualizes the effect of ReLUs in an MLP (inspired by [56]). All gray nodes have a negative output before applying the activation function and are thus effectively switched off by the ReLU.

figure. Using the ReLU activation, only a subset of neurons is active for a given input, i.e. the output is different from 0 [56]. The final output of the network is then a linear function of the subset and the input. Essentially, a neural network using ReLU activations in the hidden layers is a mixture of linear classifiers [126]. The number of linear classifiers increases exponentially with the number of neurons, i.e. model parameters.

Shortly after being proposed as activation function for deep neural networks, ReLUs where used in the CNN known today as AlexNet which is often times cited as being the neural network which kicked off the current success of deep learning. Since then, a number ReLU variants have been proposed. Leaky Rectified Linear Units (LReLUs) [113] do not set negative values to zero but rather scale them by a constant factor:

$$f_{\text{LReLU}}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{else} \end{cases}. \tag{40}$$

Improving on this concept, Parametric Rectified Linear Units (PReLUs) [66] use the factor as learnable parameter of the activation function, effectively learning the negative slope $\alpha$ for each unit:

$$f_{\text{PReLU}}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}. \tag{41}$$

The one thing that all of the three ReLU variants have in common is that they are not differentiable in 0. The recently proposed Exponential Linear Unit (ELU) [29] seeks to eliminate this kink in the function. It is defined by the following equation:

$$f_{\text{ELU}}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(e^x - 1\right) & \text{else} \end{cases}. \tag{42}$$

Fig. 13 visualizes the ReLU, LReLU and ELU activation functions. The PReLU is not shown as the parameter $\alpha$ might cause different slopes for the negative x-axis. The visualized LReLU can be seen as an instance of the PReLU function with $\alpha = 0.01$. As can be seen from both the equations and the plot, the described activation functions all

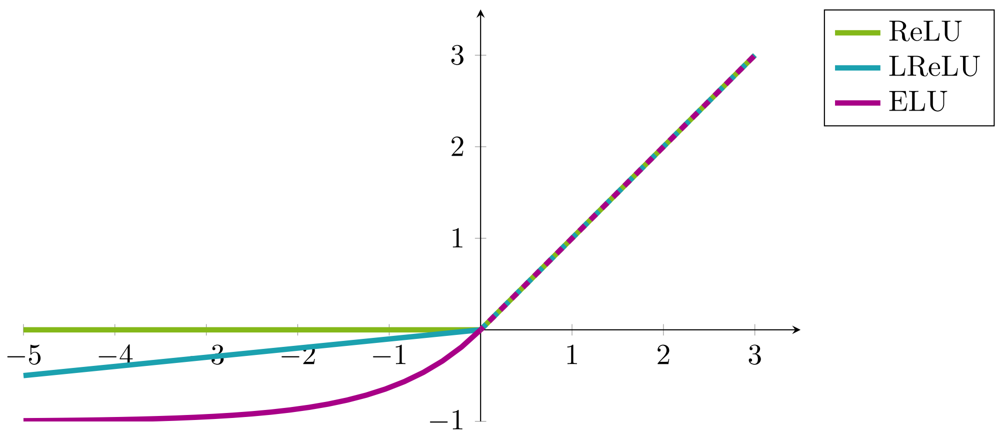


Figure 13: Visualization of the ReLU, LReLU and ELU activation functions.

negate the vanishing gradient problem as the respective gradients are linearly propagated for positive values.

Although specific activation functions may perform better in specific tasks, recent work suggests that all activation functions mentioned above perform more or less similarly in general [106]. Hence, it is common practice to simply use the ReLU for its computational simplicity and effectiveness.

## 3.5 ARCHITECTURES FOR COMPUTER VISION

The types of layers, their parametrization, connections and ordering constitute the architecture of a CNN and neural networks in general. Of course, there exists an infinite amount of possible network architectures and designing architectures for a specific task is still considered an art rather than a science. Most of the time, designing neural networks is based on intuition and design patterns without a clearly defined and principled method. There have been some attempts at a data-driven approach of finding the best architecture for a given task [13, 218]. However, these approaches make use of evolutionary algorithms, effectively treating the problem as a black box and applying a brute-force solution. Moreover, the approach requires a large amount of time in order to find suitable architectures in addition to a considerable amount of computation power in terms of GPUs. At last, using an evolutionary algorithm one is, of course, able to create only very constrained architectures.

Due to the given reasons, the prevailing approach for designing neural network architecture is to pick an established one, which performs well on other tasks, and adapt it to the specifics of the data at hand. The CNN architecture used for the presented method in this thesis makes use of this approach as well. In order to understand specific design choices presented later on, this section gives an overview over a set of CNN architectures which have become de-facto reference architectures and explains their respective unique features.

### 3.5.1 *AlexNet*

The emergence of deep learning is in large parts due to an architecture which is publicly known as *AlexNet*. Originally published under the name of SuperVision by Krizhevsky *et al.*

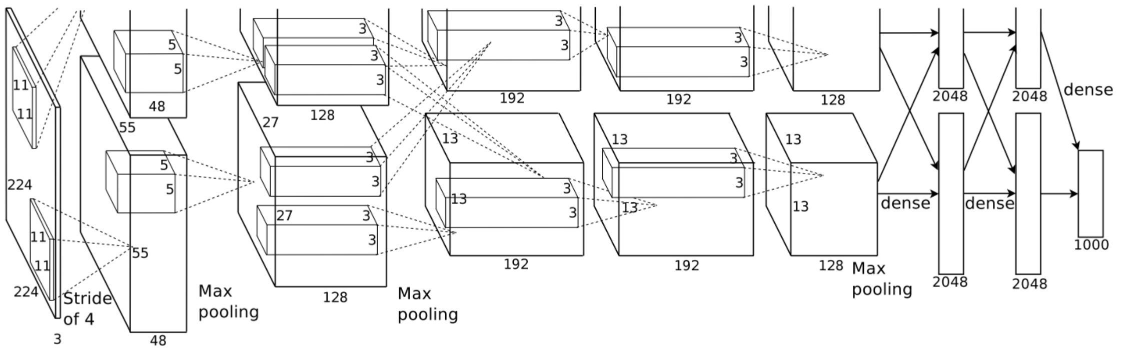


Figure 14: The figure displays the AlexNet architecture. The cuboids represent convolutional layers with the number of filters shown at the bottom and the kernel dimensions shown in the inner cuboids. The *dense* connections represent fully connected layers with the number of neurons shown at the bottom. *Max pooling* represents max pooling layers with a kernel size of $3 \times 3$ and a stride of 2 (image taken from [92]).

[92], it was the first CNN to win the prestigious ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [167] in 2012 beating the competition by an unprecedented margin.

Fig. 14 schematically depicts the AlexNet architecture. It is made up of five convolutional, three pooling and three fully connected layers. The CNN accepts input images of $224 \times 224$ pixels. As can be seen in the figure, the CNN is actually a combination of two CNNs which operate on the same image and have connections at certain layers. In the context of neural networks, this design is known as a two stream network [44]. This design choice, however, was not due to a single stream giving inferior performance but rather a more practical reason: At the time the AlexNet was proposed, high-end GPUs did only have a memory size of roughly 3 GB. The one stream version of the AlexNet was simply too large to fit on a single GPU. Hence, Krizhevsky *et al.* decided to split the net in two and have each stream of the CNN handled by one of two GPUs. As nowadays graphical memories are large enough to store the entire AlexNet, it is usually used in a variant using a single stream only in recent publications (cf. e.g. [53, 175]).

Each of the convolutional and fully connected layers in the AlexNet uses ReLUs as nonlinear activation function. In addition, the first two fully connected layers make use of a technique known as *dropout* [185]. In dropout, the output of a neuron is randomly set to 0 with a fixed probability during training. This way, succeeding neurons cannot rely on a neuron to be active given a certain input as it might have simply been “deactivated” during dropout. This has the effect of the network building more robust feature detectors and in return generalizing better to unseen data. At each training iteration, dropout allows only certain paths through the network from input to output. This can be seen as training an ensemble of smaller neural networks with the individual networks sharing their weights. Typically, the probability of dropping out a neuron is set to 0.5 which is also the value used in the AlexNet. At test time, no neuron is dropped out but the outputs are multiplied by 0.5 in order to account for the expected increase in neuron outputs.

### 3.5.2 *VGGnet*

The architecture known as VGGnet is one of the most frequently used CNNs today. Proposed by Simonyan and Zisserman [179], its simplicity in design yet strong performance

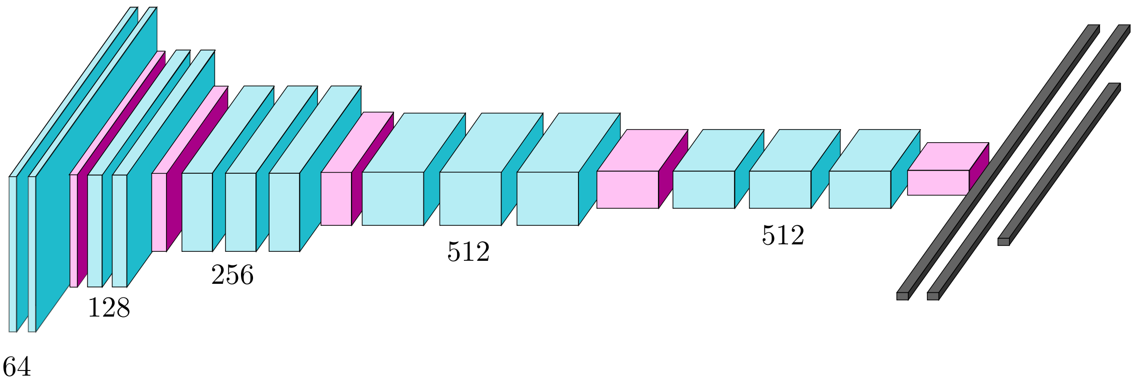


Figure 15: Visualization of the VGG16 architecture. The light blue boxes represent convolutional layers, pink boxes max pooling layers and block boxes fully connected layers. The convolutional layers can be grouped into blocks with the max pooling layers serving as separators. In every block, each convolutional layer makes use of the exact same parameters. The number of filters in the respective blocks are shown at the bottom. The two hidden fully connected layers make use of 4096 neurons each while the output layer size depends on the number of classes.

on a large variety of benchmarks lead to the VGGnet being used for a number of different applications There actually exist five different variants of the VGGnet which only differ in the amount of layers used. The most often used variants are the ones with 16 and 19 weight layers which have since been denoted VGG16 and VGG19 respectively. The VGG16 architecture is visualized in Fig. 15. It is made up of blocks of convolutional layers with max pooling layers serving as separators between the blocks. The parameters for the convolutional layers are the same within a block. However, all convolutional layers in the VGG16 make use of $3 \times 3$ filter kernels. The max pooling layers are applied in what can be considered the standard downsampling approach: Use $2 \times 2$ kernels for pooling and apply each pooling region with an offset of 2 to the previous one. After a final max pooling layer, an MLP serves as classifier. In comparison to the depicted VGG16 architecture, the VGG19 architecture simply adds one more block in the convolutional part.

The key observation for designing this CNN was that the AlexNet and other successful AlexNet-like architectures such as the improved AlexNet by Zeiler and Fergus [213] and the OverFeat net by Sermanet *et al.* [172] all use larger filter kernels for the convolutional layers in the earlier parts of the architectures while exclusively employing $3 \times 3$ kernels in the final convolutional layers. Larger filter kernels, of course, lead to more trainable parameters and are thus more prone to overfitting. Instead of using large convolutions, Simonyan and Zisserman use only $3 \times 3$ convolutions throughout the entire VGGnet. They argue that a receptive field size of $7 \times 7$ as is used by Zeiler and Fergus [213] can be obtained by stacking three convolutional layers, each containing $3 \times 3$ filter kernels. Assuming a constant number of input and output channels $c$, each layer needs $c$ filters, each having $9c$ parameters. Thus the total amount of parameters in this three layer block is $27c^2$. Using a single layer of $7 \times 7$ convolutions, this layer requires $49c^2$ parameters. Simonyan and Zisserman claim that this can be seen as imposing a regularization on the CNN, namely that the $7 \times 7$ can be decomposed into $3 \times 3$ convolutions, each using ReLUs as activation function.

### 3.5.3 *Residual Networks*

A recurrent finding in papers on deep CNNs is that an increased architecture depth leads to better performance, e.g., [92, 179]. However, in their VGGnets Simonyan and Zisserman [179] note that deeper networks do not converge when trained from scratch. Thus they initialize the nets from previously trained shallower ones. He *et al.* [66] propose to initialize weight layers which make use of ReLUs as activation function from a normal distribution with zero mean and variance of $\frac{2}{n}$. Here, $n$ is the number of input connections for a specific layer. Using this approach, even architectures like VGG19 can be trained from scratch.

For increasingly deeper architectures than VGG19, He *et al.* [67] find that a network's performance first saturates and then deteriorates rapidly even when using suitable weight initializations. In their experiments they show that this effect is not due to overfitting and that the training error is worse for extremely deep networks than shallower ones. The authors argue that this should not be the case as deep network can be constructed which has the same training error as a shallower one by simply requiring the additional layers to perform identity mappings. This observation leads the authors to the following hypothesis: Assuming that a number of layers in a CNN can be trained to approximate any underlying function $\mathbf{H}(\mathbf{X})$, it may be easier for a CNN trained with SGD to learn the residual function

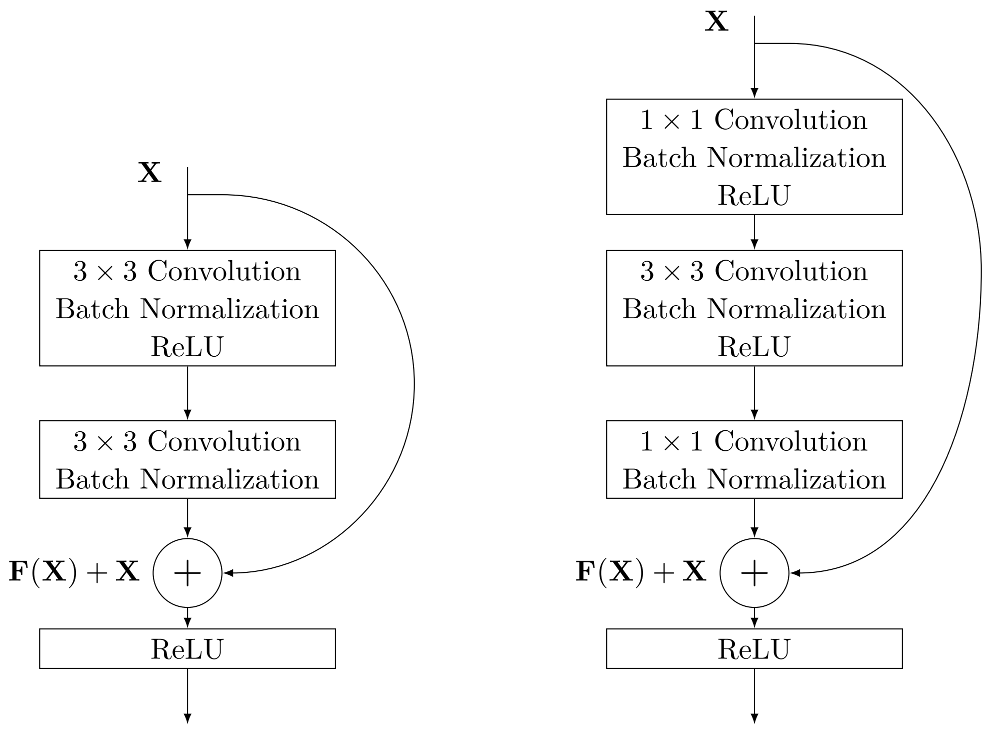
$$\mathbf{F}(\mathbf{X}) = \mathbf{H}(\mathbf{X}) - \mathbf{X} \tag{43}$$

instead of $\mathbf{H}(\mathbf{X})$. The function $\mathbf{H}$ is thus approximated by a learned function $\mathbf{F}$ and an identity mapping:

$$\mathbf{H}(\mathbf{X}) = \mathbf{F}(\mathbf{X}) + \mathbf{X}. \tag{44}$$

This concept can be integrated into CNNs through so-called *residual blocks*. Fig. 16a and Fig. 16b display the two residual block types proposed by He *et al.* [67]. Both block types make use of a so called skip-connection which allows for the input of the residual block to skip a number of layers during the forward pass. The residual standard block (Fig. 16a) is made up of two convolutional layers, each followed by a batch normalization. First proposed by Szegedy *et al.* [193], batch normalization transforms its input (in this case the output of the preceeding convolution) to have zero mean and unit variance. It is inspired by the observation, that some neural networks can be trained more effectively when presented with input data which has been normalized to zero mean and unit variance as well. As could be shown by Szegedy *et al.* [193], the training time of neural networks can be reduced substantially if batch normalization is injected into the architecture. In the residual standard block, batch normalization is applied after the convolution and before the ReLU activation function. For simplicity, the batch normalization will be seen as part of each convolutional layer. After the second convolutional layer the output is added to the input $\mathbf{X}$ of the first convolutional layer. The two convolutional layers thus represent the function $\mathbf{F}$ in Eq. 44. While the first convolutional layer makes use of the ReLU activation function, the second convolutional layer applies the activation function after having added the original input. The authors do not disclose what lead to this architectural decision.

In contrast to the residual standard block, the *residual bottleneck block* (Fig. 16b) makes use of three convolutional layers for learning the residual function $\mathbf{F}$. While the first and last convolutional layers use filter kernels of size $1 \times 1$, the second layer uses $3 \times 3$ filter kernels. The reason for constructing such a bottleneck block is the limited amount of memory available in GPUs. Using the bottleneck blocks, deeper architectures can be constructed compared to standard blocks given a fixed memory size.



(a) Residual Standard Block (b) Residual Bottleneck Block

Figure 16: The figure visualizes the two residual block types proposed by He *et al.* [67]. The figure on the left portrays a standard residual block while the structure on the right is known as bottleneck residual block. In comparison to the standard block, the bottleneck block allows for more layers in a Residual Network (ResNet) considering a fixed memory size the network may be stored in.

Residual blocks can be stacked in order to form ResNets. In order to allow for downsampling specific feature maps, ResNets typically do not make use of pooling layers but rather use convolutional layers with a stride of two pixels. The identity shortcut makes use of a linear convolutional layer with a stride of two pixels as well. This approach is essentially the same as is done for the All Convolutional Net [183] which only consists of convolutional layers and one fully connected layer.

Although residual blocks can be stacked in almost arbitrary ways to form neural networks, it is common practice to use one of four architectures initially proposed in [67], namely ResNet34, ResNet50, ResNet101 and ResNet152. The number at the end of each name symbolizes to overall amount of convolutional layers used in the respective architecture and thus directly correlates to the depth of the CNN. All but ResNet34 make use of residual bottleneck blocks while the former uses residual standard blocks.

# 4 WORD SPOTTING

After having established relevant methodological fundamentals in Chap. 2 and Chap. 3, this chapter presents relevant fundamentals with respect to the application of interest in this thesis.

*Keyword spotting* or simply *word spotting* has attracted a substantial amount of research interest over the last years. The goal in word spotting is to retrieve relevant word images from a given document collection given a query. This makes word spotting a form of Content-based Image Retrieval (CBIR). It is important to differentiate between word spotting and text recognition. In text recognition, the goal is to find the transcription of a given image portraying text. The classifier used here is tasked to produce only one recognition result. While it is generally accepted that text recognition is solved for machine printed text, recognition-based approaches often times do not yield a comparable level of accuracy for handwritten text. This characteristic emerges especially when dealing with historical handwritten documents. The main reason for the increased difficulty of these document types are large variabilities in writing style and degeneration of the original documents such as fading ink, bleed through or other aging effects [143, 165]. In contrast to recognition, word spotting methods are tasked to retrieve all relevant elements of a given document image collection in a retrieval list. Thus, instead of one single prediction, approaches used for word spotting present the user with a number of possibly relevant items. This makes word spotting-based approaches more robust than recognition-based ones.

There exists a de-facto standard terminology in word spotting which has been largely accepted by the document image analysis community. In order to have a clear understanding of specific word spotting nomenclature, this chapter first gives definitions for established and important word spotting terms in Sec. 4.1. Afterwards, an overview of relevant older (Sec. 4.2) as well as modern methods (Sec. 4.3) in the field of word spotting is given. The presentation of these methods is necessary as certain characteristics serve as motivation as well as foundation for the method presented in this thesis.

## 4.1 WORD SPOTTING TERMINOLOGY

Methods for word spotting are usually discriminated with respect to three different categories: possible query modalities, necessity for an existing segmentation of word images and availability of annotated training samples. The general word spotting scenario is that there exists a database of document images from which to extract relevant parts by querying the database. If a method is capable of using word images as queries it is said to support Query-by-Example (QbE). If the query can be provided as textual representation, a method is able to perform Query-by-String (QbS). QbE and QbS are the most common query modalities in the literature. However, there has been an increasing amount of research concerned with using other query types lately. Rusiñol *et al.* [166] propose Query-by-Speech (QbSP) word spotting. Here, the query is the digital signal of a spoken word for which relevant word images are to be retrieved. Wieprecht *et al.* [204] present Query-by-Online-

Trajectory (QbO) word spotting. In QbO, retrieval is performed by finding relevant word images with respect to an online trajectory of a handwritten word.

Apart from possible query types, a word spotting method is either dependent on a previous segmentation of the word images from the page or can operate without any such prerequisite. While approaches depending on a segmentation are referred to as *segmentation-based*, all others are able to perform *segmentation-free* word spotting.

Finally, if a word spotting method requires a set of annotated training samples it is referred to as *training-based* and *training-free* otherwise [51]. The naming convention regarding the requirement of annotated training samples is somewhat misleading. A substantial amount of methods falling into the training-free category make use of training on the supplied samples albeit in an unsupervised fashion, e.g., [159, 165, 187]. Hence, it would be more fitting to discriminate methods with respect to supervised training, unsupervised training and no training. While the initially described wording has been generally accepted in the word spotting community, word spotting methods will be categorized as *supervised*, *unsupervised* or *training free* in the following where *training free* refers to the scenario were neither supervised nor unsupervised training is happening for a given method. This is done in order to allow for a clearer discrimination between different methods.

## 4.2 ORIGINS AND EARLY WORD SPOTTING METHODS

A large number of early word spotting methods have their roots in the signal processing subfield of acoustics. This was a general trend in the early 1990's: Methods which had shown itself to be successful for audio and speech signals were transfered over to the field of document image analysis, e.g., [186]. The main idea behind this is that both audio signals and text images exhibit sequential characteristics thus allowing sequence-based methods to be applied to both fields. For example, Rohlicek *et al.* [152] use Hidden Markov Models (HMMs) with Gaussian output modeling in order to process features extracted from speech in order to perform acoustic word spotting, i.e., retrieving sections of a speech signal which represent a desired spoken word. In a very similar fashion Chen *et al.* [26] use HMMs for segmentation-free, supervised QbS word spotting in printed text: First, word bounding boxes are detected by applying a number of morphological operations and finding the connected components. Then the height of each bounding box is normalized to a fixed size with the width being scaled such that the aspect ratio is kept constant. From each scaled image, sequential contour features are then extracted which serve as input to an HMM. For word spotting, a keyword HMM is constructed by using the trained character models which correspond to the characters in the query. Each word in the corpus is then scored by the keyword HMM and a so-called non-keyword HMM. Afterwards, the retrieval list is constructed by taking the words from the corpus which have higher probability to have been generated by the keyword HMM than the non-keyword HMM. In a similar approach to the one by Chen *et al.* [26], Kuo and Agazzi [94] use a pseudo 2D HMMs in order to also perform segmentation-free, supervised QbS word spotting in printed documents.

In contrast to the sequence-based approaches used for early QbS methods, initial methods for QbE word spotting resorted to holistic representations. One of the first approaches to do so was presented by Khoubyari and Hull [82] in the form of XOR distance maps for segmentation-based, training-free QbE word spotting in printed documents. Here, a binary image is computed for the query and each word image in the corpus. Then, a so-called XOR image is computed by scaling the images in the corpus to the size of the query and computing the pixel-wise XOR values. Each pixel in the XOR image which is 1, i.e., where the corpus word image doesn't match the query word image is then replaced by the closest distance to any 0 pixel. As a measure for determining the distance from the word images in the corpus to a query word image, the squared pixel values are summed per XOR map and the corpus entries are ranked according to this distance.

While all preceding works on word spotting focus on printed text, Manmatha *et al.* [116] present the first work on QbE word spotting for handwritten text in historical documents. Using handwritten text as data is widely considered to be more difficult than printed text as the visual variability is much larger. Historical documents pose an even harder problem as they typically exhibit additional unwanted variabilities such as fading ink, bleed through or other types of image degradations [143]. For their word spotting approach, Manmatha *et al.* use the XOR distance map approach from [82]. Manmatha *et al.* also prune the retrieval list by eliminating word images which do not approximately match the query image in terms of size and aspect ratio. In addition, they evaluate a second matching algorithm which allows for a larger amount of inter-class variability. For this, the affine transformation is computed from the pixels with value 1 which best allows for obtaining the XOR distance map for a given corpus image from the XOR distance map of the query image. The distance map of the query image is then processed by the affine transform and the result is compared to the original distance map for the corpus image. Manmatha *et al.* argue that if the difference between the projection on the original distance map is large, the query and corpus image are likely to not match and vice versa.

Ensuing works on word spotting eliminated the XOR map as image representation and replaced it with more discriminative representations. Geometric features used to be a very popular choice in this regard. These features describe geometric properties of objects in images such as their outline [138, p. 96]. For binarized word images, the typically used geometric features include the upper, lower, left and right outlines of the word, cavities and column-wise transition counts of black and white pixels. One of the first approaches to use geometric features is presented by Keaton *et al.* [80]. In this method for segmentation-free, unsupervised QbE word spotting, Keaton *et al.* use the outline as well as cavity features of word images as input to a discrete cosine transform. The resulting representation is defined as keyword signature and word spotting is performed by computing the Minkowski distance of order four from the query to the corpus representations and ranking the word images according to this distance. In a different approach with geometric features, Kołcz *et al.* [85] use the upper and lower contour as features in their method for segmentation-free, unsupervised QbE word spotting. In addition, they also add the number of transition counts to the feature representation. For this, the number of changes from black to white pixels are counted for each pixel column in a given word image. Having extracted the features, the resulting representation is then normalized and scaled to a unit length, i.e., scaled along the axis of writing in order to be able to compare representations for differently sized word images. Comparing the representations is done through Dynamic Time Warping (DTW). In general, DTW finds an optimal alignment between two sequences [123]. The cost for this alignment is used by [85] as similarity measure for comparing two sequences of the image features explained above. This measure is also known as DTW distance For word spotting, they extract hypothesized word images from a document image and rank these hypotheses based on the DTW distance of their representation to the representation of a specified query image. In a similar approach, Rath and Manmatha [142] also use the DTW distance on sequences of geometric features in order to perform word spotting. For this, the vertical projection profile of a word image is extracted and normalized. In contrast to Kołcz *et al.*, however, Rath and Manmatha do not scale the extracted profiles along the direction of writing. They are able to show, that the combination of geometric features and DTW vastly outperform approaches based on XOR maps.

## 4.3 MODERN WORD SPOTTING METHODS

One common approach in early word spotting methods was the use of heuristic features based on binarized word images. Especially when dealing with historical documents, however, binarization is a difficult task and often times falls victim to the aforementioned degradations in the document images. Recent approaches for word spotting thus aimed at introducing more robust feature representations for word images. For this, state-of-the-art methods from the late 2000's drew inspiration from other computer vision tasks. One trend that could be observed for other tasks was the use of local image descriptors based on gradient statistics, e.g, classification or retrieval of natural images [25]. In contrast to the previously presented features, these descriptors can be obtained from gray-level image directly thus eliminating the necessity for binarization. One of the most prominent local descriptors used across different computer vision tasks is the one used in the Scale Invariant Feature Transform (SIFT) approach [112]. This local feature representation is generally known as SIFT descriptor [25]. Its success in natural image classification and retrieval could be transfered over to word spotting as well. The first word spotting method to make use of SIFT descriptors is presented by Ataer and Duygulu [10]. Here, a Bag of Features (BoF) histogram is obtained from the descriptors which is then used as holistic word image representation in order to perform segmentation-based, unsupervised QbE on word images of Ottoman script. For this, Ataer and Duygulu apply the "classic" SIFT approach in that they first detect keypoints in the word images at which the descriptors are computed. Then, the obtained descriptors are clustered and quantized into a BoF representation. Finally, the representation is $L_1$-normalized. For retrieval, they rank the word images from the corpus according to the Kullback-Leibler divergence between the query and the corpus representation.

An interesting observation when using SIFT descriptors for classification and retrieval of natural images is, that the accuracy can be improved if the descriptors are sampled at fixed locations instead of at keypoints [25]. Typically, this is done in a regular grid spanning the entire image. The stride in this grid is chosen such that the descriptors overlap. The latest works on word spotting using descriptor-based representations exclusively use this dense grid approach. In an influential approach, Rusiñol *et al.* [164] pursue this concept by representing regions of a document images by spatial pyramids using quantized SIFT descriptors in order to perform segmentation-free, QbE wordspotting. The query images are also represented through a spatial pyramid. At query time, a window is slid over the document images in order to extract small patches of the given document. The respective spatial pyramid representation is then computed for each patch. The query and patch representations are then projected into a lower dimensional space using latent semantic indexing [35]. Finally, the projected patch representations are ranked according to their cosine dissimilarity

$$d_{\cos}(\mathbf{a}, \mathbf{b}) = 1 - \frac{\mathbf{a}^T \mathbf{b}}{||\mathbf{a}|| \cdot ||\mathbf{b}||}, \tag{45}$$

to the projected query representation in order to obtain the retrieval list. The respective representations are denoted as **a** and **b** in Eq. 45. The method by Rusiñol *et al.*

can be characterized as segmentation-free and unsupervised while allowing for QbE word spotting. In [165], Rusiñol *et al.* improve upon their method by incorporating a Product Quantization-based indexing structure. This way, the computation of patch representations can be reduced to a small amount of regions in the document image which are deemed as potentially relevant by the indexing structure. Not having to compute a spatial pyramid for all possible patch positions results in a decrease in retrieval time and allows the method to be applied to large corpora.

In an approach similar to the one by Rusiñol *et al.*, Aldavert *et al.* [6] use spatial pyramids on top of SIFT descriptors in their approach for segmentation-based, supervised QbS word spotting. For this, the available annotations for the word images are transformed into a histogram representation counting all different $n$-grams in the annotation. The obtained histogram is then $L_2$-normalized and concatenated with a SIFT descriptor-based spatial pyramid representation of the respective word image. All concatenated vectors are used for determining a transformation matrix in order to project the vectors into a low-dimensional subspace using truncated Singular Value Decomposition (SVD). This subspace is used as basis for the ensuing word spotting approach. The general idea is to be able to compute purely textual and purely visual representations of annotations and word images respectively. For the word images in the corpus, there doesn't exist an annotation. They are projected into the subspace by padding the visual representation, i.e., a spatial pyramid, with a vector of zeros such that the resulting vector has the same dimensionality as the combination of spatial pyramid and textual representation. Likewise, query strings can be projected into the subspace by computing the textual representation and then padding with a vector of zeros of the same dimensionality as the spatial pyramid. The assumption is that the projections of the padded visual representations and the padded textual representations are close for corresponding annotations and word images in the subspace. The presented approach gave then state-of-the-art results for segmentation-based QbS word spotting.

While the approaches by Rusiñol *et al.* [164] and Aldavert *et al.* [6] make use of SIFT descriptors in holistic representations of word images, another method incorporates them into a sequential model: In [158], Rothacker *et al.* combine BoF representations with HMMs in order to represent word images. Like the spatial pyramid approach from [164], these so called BoF-HMMs are able to perform segmentation-free, unsupervised QbE word spotting. A sliding window approach is used here as well in order to extract patch regions from the document images. For each patch region, a dense grid of quantized SIFT descriptors is computed. The descriptors in each column of the dense grid are then pooled into a sequence of BoF histograms. For the desired query image, the sequential representation is generated the same way. An HMM is then trained on the sequence of BoF histograms extracted from the query image. After training, this HMM is used in order to score each patch region extracted from the corpus with respect to how likely it is that the BoF histogram sequence from the patch was generated from the HMM. For generating the retrieval list, the patches are ranked according to their score. The concept of BoF-HMMs is extended in [157] to allow for segmentation-free, supervised QbS word spotting. This is done by training separate BoF-HMMs for the individual characters present in a given dataset.

Apart from the SIFT descriptor, other gradient-based descriptors which have been used for word spotting are Local Gradient Histograms (LGHs) [148, 151], Histograms of Oriented Gradients (HOGs) [8, 86] or HOG-like descriptors [7, 197]. Independent of the feature representation used though, word spotting methods presented in the literature are

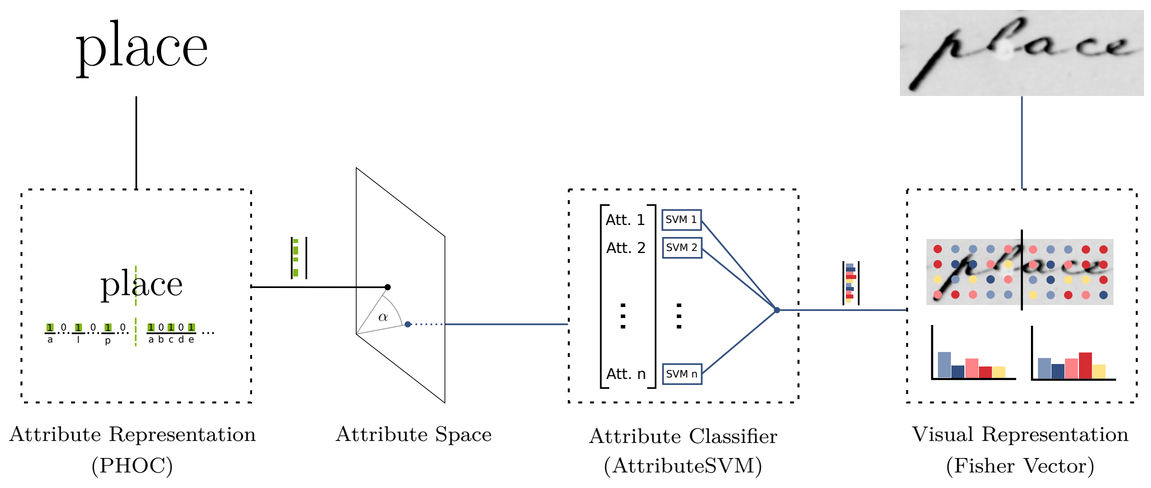

place
place
Att. 1
Att. 2
Att. n
SVM 1
SVM 2
SVM n
α
Attribute Representation (PHOC)
Attribute Space
Attribute Classifier (AttributeSVM)
Visual Representation (Fisher Vector)

Figure 17: Visualization of the embedded attributes framework as presented by Almazán *et al.* [9]: For a given word image the corresponding Pyramidal Histogram of Characters attribute representation is predicted from an ensemble of Support Vector Machines (SVMs) based on a  representation of the word image (right side). This attribute representation can also be obtained directly for a given word string (left side). QbE and QbS are then performed by ranking all word images of a given corpus based on their cosine distance to the query representation. This representation is either obtained directly for word strings or also obtained from the SVM ensemble. The figure is inspired by a similar one found in [204]. Parts of the figure are courtesy of Christian Wieprecht and Leonard Rothacker.

typically designed to work under either of the QbE, QbS, QbO or QbSP paradigms. If a method shall be enabled to accept other types of queries as well it usually needs to be adapted. An example for this are the aforementioned BoF-HMMs which where initially proposed for QbE [158] and then adapted in order to accept string queries for QbS [157]. A very elegant solution for combining different query paradigms is proposed by Almazán *et al.* [9] in the form of the embedded attributes framework. This framework is also the foundation of the method presented in this thesis. Due to its high relevance, it will be presented extensively in the following section.

## 4.4 EMBEDDED ATTRIBUTES

The embedded attributes framework has been a very influential concept, not only in word spotting but in document image analysis in general. Initially proposed by Almazán *et al.* [9], it allows for a unified approach for supervised QbE and QbS word spotting. While Almazán *et al.* initially proposed a segmentation-based method for the embedded attributes framework, recently proposed methods extend it to segmentation-free scenarios as well (cf. e.g. [50, 160, 207]).

Fig. 17 visualizes the embedded attributes framework using the approach by Almazán *et al.* The basic concept is to encode word strings and word images as common representation using attributes (cf. Sec. 2.3). If these attribute representations can be obtained from word strings and word images, word spotting boils down to a simple nearest neighbor search in the attribute space. For this, there need to exist two mappings into the common attribute space: one for the word strings, i.e., word classes, and another one for the word

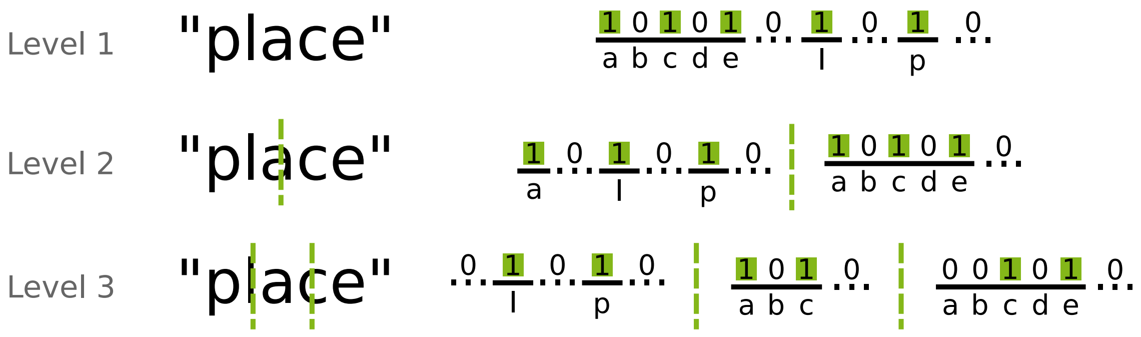


Figure 18: The figure visualizes the construction of a Pyramidal Histogram of Characters representation for the word string *place*. The example shown here makes use of a PHOC with levels 1,2 and 3. For each region in the respective level, an indicator vector is obtained showing whether a given character is present in this region or not. Not shown is the final representation which is obtained by concatenating all individual vectors. Parts of the figure are courtesy of Christian Wieprecht and Leonard Rothacker.

images. While Almazán *et al.* require the mapping for the word classes to be directly obtainable from the respective string representation of the class, the mapping from word images to attribute space is best obtained through some form of trainable model. For simplicity, the mapping for the word strings will be referred to as the textual model while the one for word images will be called visual model. Having the visual model not predict class labels but attributes allows for predicting attribute representations for word images from classes which where not seen during training. Especially QbS word spotting benefits from this characteristic: In QbS the textual model is responsible for predicting the query representations. As it is able to compute this representation directly from a string unknown at training time, the queries are not constrained to come from a predefined, i.e., closed lexicon. In analogy to zero-shot learning, this paradigm is also known as zero-shot retrieval in image retrieval in general [33]. For word spotting applications, this is essentially a requirement as it typically cannot be known beforehand which queries are of interest to a potential user.

The attribute representation used to represent strings and word images plays a crucial role in the embedded attributes framework. Almazán *et al.* propose an embedding for this which they term Pyramidal Histogram of Characters (PHOC). The PHOC encodes the presence or absence of characters in certain sections of a string in a pyramidal fashion. Fig. 18 visualizes how to construct a PHOC vector for the string *place*. In order to construct the representation, an alphabet of characters or unigrams needs to be defined first. Then, an attribute is created for each unigram of the alphabet indicating the presence or absence of this specific character in a certain split of the string. At the first level of the PHOC, all unigrams in the string are considered and the respective attribute is set to 1 if the corresponding unigram appears at least once in the word. In the following levels, the string is split into regions of same sizes. For each unigram and region it is then evaluated whether the region contains at least one of the respective unigrams. For determining whether a certain unigram in the given string lies in a specific region, all unigrams in the string to be processed are first defined to have an equal width of one. In the example in Fig. 18, the total width of the word would thus be five. Almazán *et al.* then define a unigram to be present in a given region if it overlaps at least 50% with the particular region. For example, the character *a* in Fig. 18 has an overlap of 50% with both left and right region in the second level. Hence, it is marked as present for both regions. In the ensuing levels, the string is split into increasingly smaller regions. At the third level there are three regions and at the fourth four. This pattern continues for as many levels as are desired. Almazán *et al.* propose to not use the first level in the PHOC as the resulting vector for this level can not discriminate between anagrams, e.g., *asleep* and *please*. In their PHOC definition, they use the levels $2, 3, 4$ and 5. In addition to the attributes for the unigrams, Almazán *et al.* also add attributes for bigrams to the PHOC representation. For this they determine the 50 most common bigrams in the English language and add a level with two regions accounting for bigram attributes. This setup results in the PHOC being obtained from 14 individual vectors for the unigrams and 2 vectors for the bigrams. As unigrams Almazán *et al.* use all lower-case characters from the Latin alphabet plus the ten digits. This is justified by their assumptions made for a potential application: They assume that a word image should be considered relevant for a given query if the transcriptions match without considering whether characters are capitalized or not. The resulting PHOC is of size 604 ($14 \cdot 36$ elements for the unigram attributes and $2 \cdot 50$ elements for the bigrams).

While the PHOC represents the textual model, Almazán *et al.* make use of an ensemble of SVMs as the visual model which they term AttributeSVM. The AttributeSVM requires that the image is encoded into some form of holistic feature representation. For this, Almazán *et al.* choose the Fisher Vector on top of enriched SIFT descriptors which are extracted in a grid. For enriching the SIFT descriptors, they add the normalized width and height coordinates of the point in the grid the descriptor was extracted at to the descriptor. The descriptors are then projected into a lower dimensional space using Principal Component Analysis (PCA). In order to introduce even more spatial information, the Gaussian Mixture Model (GMM) used for constructing the Fisher Vector is not computed for all descriptors but only for certain ones: At training time, the descriptors are pooled into twelve bins (six along the width and two along the height of the word image) depending on the location they were extracted at. For each bin, an individual GMM with 192 mixture components is fitted using the enriched descriptors from the respective bin as data. Afterwards, all GMMs are joined and the weights of the individual components are $L_1$-normalized in order to sum to 1. The Fisher Vector is then created with this joint GMM.

Having defined the global feature representation, the goal is to predict the PHOC representation corresponding to the transcription of the word image at hand. In order to do so, there exists an SVM for each dimension, i.e., attribute, in the PHOC vector, which is responsible for predicting this attribute. The combination of all SVMs then predicts the entire PHOC. In [9], Almazán *et al.* use linear SVMs as their Fisher Vector representation has a size of 24 576. The large dimensionality allows for disregarding a kernel in the SVMs.

The embedded attributes framework allows for QbE as well as QbS word spotting in a straight forward way: For the corpus to be retrieved from, the PHOC is predicted for each word image from the AttributeSVMs. For QbE, the query representations is also predicted from the AttributeSVMs while for QbS it is obtained directly from the query string. The retrieval list is then generated by ranking all representations obtained from the corpus with respect to the cosine distance to the query representation.

For obtaining the predicted representations, Almazán *et al.* do not use the class predictions as attributes but rather the raw distance from a given Fisher Vector to the hyperplane of the respective SVM. As these scores exhibit a different range than the PHOCs they are compared with in QbS, the are calibrated. In order to do so, three different techniques are evaluated: The first is Platt's scaling. Here, the generalized logistic function

$$f(x) = \frac{1}{1 + e^{\alpha x + \beta}} \tag{46}$$

is fitted to the scores using Maximum Likelihood Estimation (MLE) were $\alpha$ and $\beta$ are the trainable parameters. The final attribute prediction is obtained by applying this function to the aforementioned distance values. The other two approaches used by Almazán *et al.* for calibration are Canonical Correlation Analysis (CCA) and Kernelized Canonical Correlation Analysis (KCCA). For both CCA and KCCA the goal is to find the projections to a common subspace for two data sources which maximize the correlation between corresponding elements. In the context of the method proposed by Almazán *et al.* both can thus be used to correlate the distance values obtained from the AttributeSVMs with the desired PHOC representation. While CCA achieves this through an individual linear projection for both data sources, KCCA adds a kernel function to the projection in order to allow for non-linear correlations. Almazán *et al.* evaluate the effectiveness of the three calibration methods and find that Platt's scaling gives better results for smaller datasets while the other two methods work better if there exists more training data. However, there is no consistent improvement when using KCCA instead of CCA.

The embedded attributes framework is very flexible and does not only allow for a simple and effective way to perform QbE or QbS word spotting but also other query modalities. For example, Wieprecht *et al.* [204] use this framework in order to perform QbO word spotting. For this, the textual model is replaced with a model which is able to predict the corresponding PHOC representation given an online-handwritten trajectory. This allows for using the online trajectories as queries to a corpus of offline word images. In general, any query modality can be used in the presented framework as long as a desired attribute representation can be obtained for it.

There exists a number of other approaches for word spotting in the literature which are based on embedded attributes as well. As they bare a high relatedness to the proposed method, they will, however, be discussed in the chapter on related work (Chap. 6).

# 5 ATTRIBUTE CNNS

Having established the necessary fundamentals in the previous chapters, this chapter presents the methodology proposed in this thesis. The approach shown here is able to perform supervised Query-by-Example (QbE) and Query-by-String (QbS) word spotting and falls into the segmentation-based category[1]. The main contribution is the design of Convolutional Neural Networks (CNNs) which are able to predict attribute representations. These neural networks can then directly be used in the previously described embedded attribute framework (cf. Sec. 4.4). Due to their special characteristic of predicting attributes, the CNNs presented here will be referred to as *Attribute CNNs*. The motivation for using a CNN is based on the observation that supervised word spotting methods typically follow the traditional computer vision pipeline in which feature representation and classification model are optimized separately (cf. Chap. 2). This is also the case for the AttributeSVM approach by Almazán *et al.* [9]. In addition, the different AttributeSVMs used here are unable to share trained knowledge which leads to each Support Vector Machine (SVM) being trained independently. This thesis proposes to use a single Attribute CNN which, in contrast to the AttributeSVM, predicts all attributes in a combined fashion. In addition, it is able to learn the attribute representations in an end-to-end fashion thus eliminating the need to design heuristic feature representations.

There are two crucial aspects when designing Attribute CNNs: The first is to choose a suitable network architecture. The second is to have the network output a representation which is suitable for the word spotting problem at hand. For the second aspect, an appropriate way needs to be determined how the respective CNNs can be trained with a given representation. The key ingredient for this is a suitable loss function which can be used for training. This chapter thus explains the design choice for the proposed CNN architectures as well as how to derive loss functions which allow for training a CNN with attribute representations. Before getting into the details of the CNNs though, it is first explained how Attribute CNNs can be used in the embedded attributes framework in order to perform word spotting.

Afterwards, the attribute embeddings are explained which will be used in conjunction with the presented networks (Sec. 5.2). The ensuing section then explains how neural networks in general can be trained to predict these attribute representations (Sec. 5.3). The last section of this chapter finally presents the CNN architectures used for the proposed method (Sec. 5.4).

Please note that the sole focus of this chapter is to present the proposed method. A comparison to related work including the discussion of the respective differences will be given in the next chapter.

1 The method presented here is in principal also able to perform supervised Query-by-Online-Trajectory (QbO) word spotting. However, in order to do so, it requires additional considerations compared to QbE and QbS. For the sake of a clean presentation, QbO will not be presented in this chapter. There exists a chapter in the appendix which concentrates on QbO with Attribute CNNs exclusively (see Appendix C).

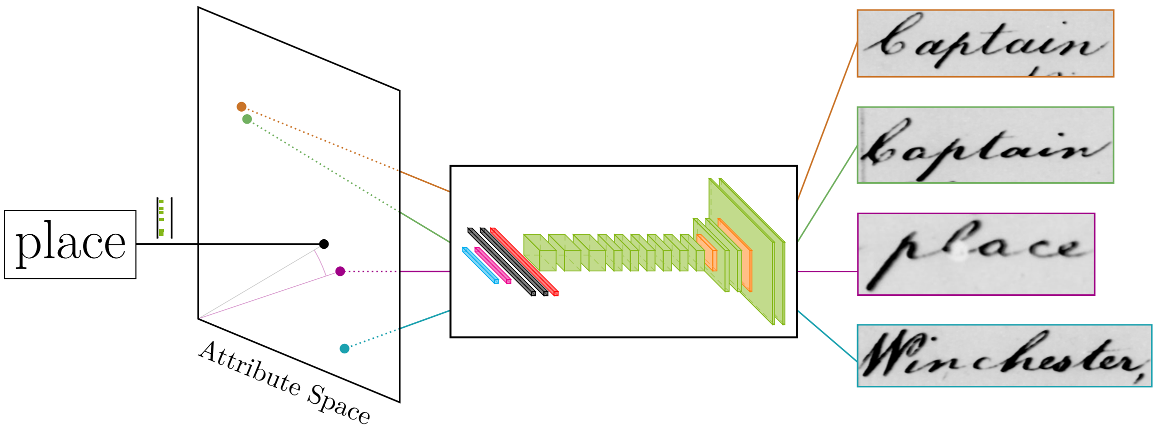


Figure 19: Overview of the presented method for QbE and QbS word spotting: Word images (right) as well as word strings (left) are embedded in an attribute space. For strings, this mapping is computed directly, while for word images an Attribute CNN predicts the attribute representation. The word spotting problem then boils down to a simple nearest neighbor search in the attribute space.

## 5.1 WORD SPOTTING WITH ATTRIBUTE CNNS

Suppose a CNN is able to predict a desired attribute representation for a given word image and the attribute representation for the transcription can be obtained directly, i.e., is a function of the word string, as is the case for, e.g., the Pyramidal Histogram of Characters (PHOC). Hence, the attribute representation for all word images in a desired corpus can be predicted from the CNN prior to running retrieval. QbE word spotting can then be performed by predicting the attribute representation for a given query word image and ranking all images in the database according to the distance to the query representation. As the attribute representation is a function of word strings, QbS word spotting can be performed by computing the attribute representation for the given query string and ranking all attribute representations of word images in the corpus as was done for QbE. Fig. 19 gives an overview of the described approach. Of course, the proposed approach requires a suitable distance metric for determining distances between representations. As the attribute representations are typically of high dimensionality, the cosine dissimilarity (Eq. 45) can be reasonably expected to be suitable for the task at hand. This measure is actually the one predominantly used when comparing high dimensional representations, not only for word spotting, e.g., [6, 165, 206], but in other pattern recognition problems as well, e.g., [28, 126, 127].

## 5.2 ATTRIBUTE REPRESENTATIONS FOR WORD SPOTTING

When performing word spotting in the embedded attributes framework, the attribute representation used plays a critical role. In order to choose a suitable representation, one has to define which behavior should be expected from a word spotting method with respect to what it considers similar, i.e., relevant for a given query. Of course, exact matches for a sought-after query are always similar but the question is whether the user prefers word images in the retrieval list which are semantically close to the query or rather images which show words which are close from a point of characters and their respective order. An example for this decision would be, whether a word spotting method should rather return word images showing military ranks for the query *Captain* or if word images showing different inflections or abbreviations of the query are of higher interest.

For the rest of this thesis, it is assumed that a word spotting method is of interest which focuses on the latter, i.e., lexical similarity. Thus, methods like *Word2Vec* [121] are not considered in the following[2]. This paradigm does not only allow for the retrieval aspect of word spotting but also enables word spotting methods to be used directly for handwriting recognition [9] or use them in order to support other methods for Optical Character Recognition (OCR). For example, Silberpfennig *et al.* [178] use word spotting as an additional confidence measure for an unreliable baseline OCR method.

The second constraint which is made regarding the representation is that it should be obtainable for words which were not seen during training, effectively allowing for querying classes unknown from training. The representation must thus allow for zero-shot learning (cf. Sec. 2.3) which rules out representations like the one proposed by Weston *et al.* [203].

In the context of the thesis, there exist three prominent attribute representations for word strings which can effectively be integrated into the proposed methods, namely the PHOC, the Spatial Pyramid of Characters (SPOC) and the Discrete Cosine Transform of Words (DCToW). The inevitable question, of course, is, whether any of these representations is more suitable to be used with the proposed Attribute CNNs than the others. This question will be evaluated in the experiments presented later in this thesis. In the rest of this section, the SPOC and DCToW representations are presented and their individual characteristics are discussed. The PHOC will not be elaborated on again as it was already presented in Sec. 4.4.

The SPOC proposed by Rodriguez-Serrano *et al.* [150] for both scene text recognition and retrieval is very similar to the PHOC. Like is done for the PHOC, a given string is split into multiple levels and histograms are computed for each of these levels in order to form this representations. Different to the PHOC, though, not the presence of characters is stored in the individual histograms but the actual count of characters or fractions thereof. Fig. 20 visualizes how a 3-level SPOC descriptor can be extracted from the given string *aabcc*. At each level, the string is split into a number of regions for which a character histogram is computed. For this, each character is treated as having equal width and the total width of a string is considered to be the number of characters it contains. If only a fraction of a character overlaps with a region, this fraction is added to the respective bin in the histogram as well. Afterwards, all individual histograms are concatenated. Rodriguez-Serrano *et al.* note that the individual histograms may optionally be $L_2$-normalized. They do not perform normalization for their recognition experiments but do so for their retrieval experiments. In addition, they propose to use four levels for the SPOC, each doubling the number of splits compared to the previous layer. Thus at the first level, the histogram is extracted for all characters in the word. At the second level there is a split into left and right part while the third splits these two again in two parts each. At the fourth and final layer, the string is split into eight different regions.

Different from the PHOC and SPOC, the string embedding known as DCToW [206] does not make use of a pyramidal splitting of a given string. Fig. 21 visualizes this embedding approach. First, the word is transformed into an indicator matrix. This is done by encoding

2 In principal, the proposed approach would still allow for easily integrating Word2Vec representation. However, they are not in the scope of this thesis

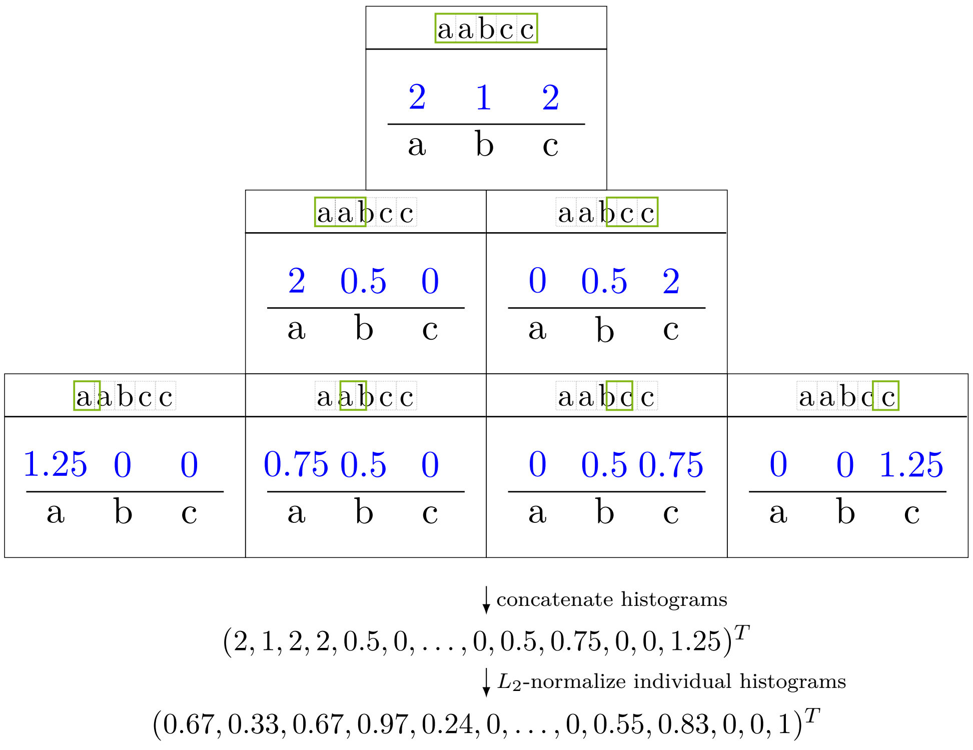

aabcc
2 1 2
a b c
aabcc
2 0.5 0
a b c
aabcc
0 0.5 2
a b c
aabcc
1.25 0 0
a b c
aabcc
0.75 0.5 0
a b c
aabcc
0 0.5 0.75
a b c
aabcc
0 0 1.25
a b c
concatenate histograms
$(2, 1, 2, 2, 0.5, 0, \ldots, 0, 0.5, 0.75, 0, 0, 1.25)^T$
$L_2$-normalize individual histograms
$(0.67, 0.33, 0.67, 0.97, 0.24, 0, \ldots, 0, 0.55, 0.83, 0, 0, 1)^T$

Figure 20: Visualization of how to extract a 3-level SPOC embedding from the string *aabcc*. At each level the string is split into a number of regions for which individual histograms are computed. The regions, which are considered in the respective levels, are indicated in green at the top of each region in the pyramid. If a fraction of a character falls into a region, this fraction is used in the histogram as well. Afterwards, all histograms are concatenated and optionally normalized.

every character in the string as a one-hot vector with the characters in the alphabet used serving as classes. The resulting vectors are then stacked in order to form the matrix. Then, a discrete cosine transform is applied to each row. The three largest values per row are then concatenated in order to form the DCToW descriptor. As only the three largest coefficients are chosen, strings of variable length can be encoded in a fixed size representation this way.

Strictly speaking, both SPOC and DCToW are not attribute representations as they are not binary (cf. Chap. 2). However, both can be considered attribute-like. Looking at the SPOC, a human can still determine whether a certain number of characters is present in a split or not or if a fraction of a character belongs to a split. For the DCToW, the final representation is obtained from a binary, i.e., attribute representation. The mere transformation involved in SPOC and DCToW should not render them as strictly non-attribute representations. For simplicity, they will be referred to as attribute-like representations in the following.

In addition to the three embeddings mentioned above, two novel feature embeddings for word spotting are presented and evaluated in this thesis. The main idea behind creating both embeddings is that the elements in the pyramidal representations, i.e., PHOC and SPOC, may be heavily correlated. For a neural network, it may be easier to learn an embedding if its elements are decorrelated. A well known method which achieves decorrelation is, of course, Principal Component Analysis (PCA). However when using PCA, one relies on being able to estimate reliable covariance values. Obtaining good covariance

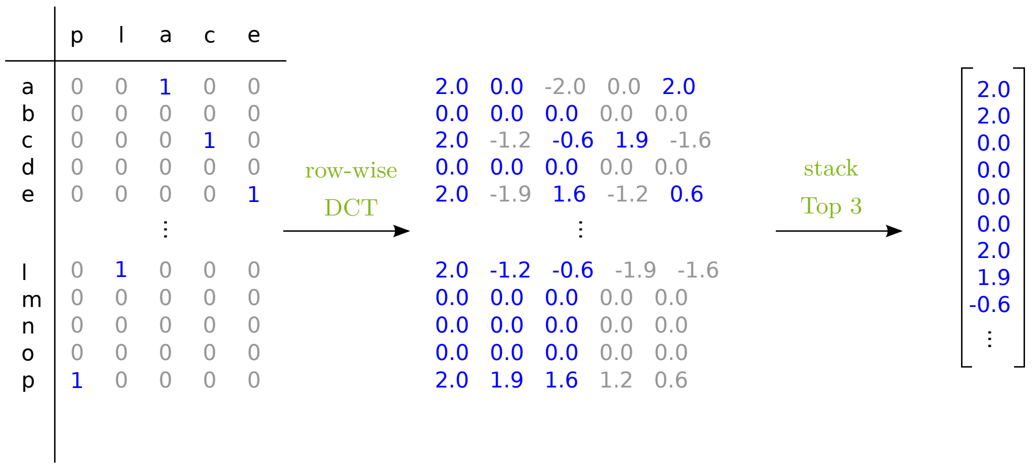


Figure 21: Visualization of how the DCToW embedding is created for the word *place*. First the word is transformed into an indicator matrix which resembles the alphabet used for the DCToW in its rows and the characters of the word in the columns. Then a row-wise discrete cosine transform is performed. The three largest values of each row are then concatenated in order to form the DCToW descriptor.

estimates is rather problematic for the high dimensional string embeddings. Hence, two other decorrelation methods are used. While the first requires the string embedding to be binary vectors, the second approach works for all types of data as long as there exists a suitable metric for determining similarity between two embedded representations.

For decorrelating binary vectors, Chollet [28] proposes to use a transformation based on the Pointwise Mutual Information (PMI). The PMI is a measure for determining the discrepancy between the joint probability of two univariate and discrete random variables $X$ and $Y$ and the product of their individual probabilities. It is defined as

$$\text{pmi}(x, y) = \log \frac{p(x, y)}{p(x)\, p(y)}. \tag{47}$$

where $x$ and $y$ are realizations of $X$ and $Y$ respectively. Using the PMI, a matrix $\mathbf{M}$ is computed in [28] where each element $m_{i,j}$ captures the PMI between the $i$-th and $j$-th attribute of a binary vector $\mathbf{y}$:

$$m_{i,j} = \text{pmi}(y_i, y_j). \tag{48}$$

For simplicity, this matrix will be referred to as PMI matrix. It can be decomposed via Singular Value Decomposition (SVD) into

$$\mathbf{M} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{U}^T, \tag{49}$$

where $\mathbf{U}$ is a unary matrix and $\mathbf{S}$ the matrix of singular values. Defining the matrix $\mathbf{E}_{\text{PMI}} = \mathbf{U} \cdot \sqrt{\mathbf{S}}$, the matrix $\mathbf{M}$ of individual PMI values can be rewritten as

$$\mathbf{M} = \mathbf{E}_{\text{PMI}} \cdot \mathbf{E}_{\text{PMI}}^T. \tag{50}$$

Chollet [28] argues that using $\mathbf{E}_{\text{PMI}}$ as projection matrix one can project the labels $\mathbf{y}$ onto a latent space where each dimensionality describes an independent factor of variation of the PMI. These projections can then be used as surrogates for the string embeddings in order to train a neural network. It is obvious that the proposed approach can only work if $\mathbf{S}$ is at least positive semidefinite. This might, however, not always be the case. Even worse, the elements $m_{i,j}$ can obtain the value $-\infty$ if the corresponding joint probability $p\left(y_i, y_j\right)$ is zero. In order to overcome this problem, a small change to the approach by Chollet is proposed in this thesis. Looking at the original PMI-based approach, one can see that the PMI matrix represents positive as well as negative correlations between attributes. For training a CNN, however, only the possible positive correlations are of interest. This is due to the neural network possibly learning the aspects of the data that actually belong to the correlated attributes opposed to the "correct" ones. A simple example for this for natural images would be the attributes *sunny* and *blue sky*. Depending on the data, a neural network might learn to predict *blue sky* based solely on the presence of the sun in an image. This, of course, leads to errors if the sun can be seen on an image showing a partially clouded sky. As only the positive correlation between attributes is of interest, one can adapt Eq. 47 to disregard negative correlations:

$$\operatorname{ppmi}(x, y) = \max\left(0, pmi\left(x, y\right)\right). \tag{51}$$

This measure is known as Positive Pairwise Mutual Information (PPMI) [77, p. 276]. Using the PPMI in Eq. 48, $\mathbf{S}$ contains only positive and finite values. This way, the projection matrix $\mathbf{E}$ is defined and can be used in order to project binary attribute representations onto a latent space of independent factors of variation of the PPMI.

The approach proposed above can only be applied to binary vectors as otherwise the matrix $\mathbf{M}$ could not be constructed. Hence, it is applicable to the PHOC but not the SPOC and DCToW. In order to decorrelate these real-valued attribute representations, an approach is proposed which is based on Multi Dimensional Scaling (MDS). MDS is usually used as an algorithm to solve the inverse distance problem which is defined as finding the location of a fixed amount of points given only their pairwise distances and the dimensionality $d$ of the target space. This is achieved by computing a matrix $\mathbf{B}$ using the matrix $\mathbf{D}$ of pairwise distances as follows:

$$\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H}, \text{ where} \tag{52}$$

$$\mathbf{H} = \mathbf{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T. \tag{53}$$

Here, $\mathbf{I}_n$ is the $n \times n$ identity matrix and $\mathbf{1}_n\mathbf{1}_n^T$ the $n \times n$ matrix of all ones. From $\mathbf{B}$ the $d$ largest eigenvalues $\lambda^{(i)}$ and their corresponding eigenvectors $\mathbf{v}^{(i)}$ are extracted and stacked into a matrix as follows:

$$\mathbf{E}_{\text{MDS}} = \begin{pmatrix} \sqrt{\lambda^{(1)}} \cdot \mathbf{v}^{(1)T} \\ \sqrt{\lambda^{(2)}} \cdot \mathbf{v}^{(2)T} \\ \vdots \\ \sqrt{\lambda^{(d)}} \cdot \mathbf{v}^{(d)T} \end{pmatrix}. \tag{54}$$

The shape of $\mathbf{E}_{\text{MDS}}$ is $d \times n$ and each column represents the embedding for a specific data point.

In addition to solving the inverse distance problem, MDS can be exploited to decorrelate attribute representations given a suitable distance metric. This is achieved by computing the matrix of pairwise distances from the obtained attribute representations and obtaining the MDS embedding while keeping all positive eigenvalues and their corresponding eigenvectors. In this thesis, the cosine distance (cf. Eq. 45) is used as metric. The embedding thus aims at preserving the angles as distances between the given attribute representations. In addition, the dimensions with eigenvalues of 0 in the embedding $\mathbf{E}_{\text{MDS}}$ are disregarded in the proposed approach as well. This way, the dimensionality of the embedding is kept to a minimum while preserving all relevant information.

Attribute representations, which were not observed at training time, can easily be transformed into the space obtained from MDS through *out-of-sample embedding* [17]. Given an attribute representation $\mathbf{a}$, its embedding $\mathbf{e}$ is obtained by the following transformation:

$$\mathbf{e} = \frac{1}{2}\mathbf{E}^{\#}\left(\mathbf{m} - \mathbf{d}^2\right), \text{ where} \tag{55}$$

$$\mathbf{E}^{\#} = \begin{pmatrix} \frac{1}{\sqrt{\lambda^{(1)}}} \cdot \mathbf{v}^{(1)^T} \\ \frac{1}{\sqrt{\lambda^{(2)}}} \cdot \mathbf{v}^{(2)^T} \\ \vdots \\ \frac{1}{\sqrt{\lambda^{(d)}}} \cdot \mathbf{v}^{(d)^T} \end{pmatrix}, \tag{56}$$

$\mathbf{d}$ is a vector of distances from $\mathbf{a}$ to all samples used in obtaining the original MDS embedding and $\mathbf{m}$ is the column mean of $\mathbf{D}^2$ (cf. Eq. 52).

In the following, if a decorrelation method is used for a given attribute representation, the resulting representation will be denoted by prepending the decorrelation method to the attribute representation's name. For example, the PPMI-PHOC representation is obtained by applying the PPMI embedding technique to the standard PHOC representation.

## 5.3 LOSS FUNCTIONS FOR ATTRIBUTE CNNS

Having defined the attribute representations to be used for the proposed approach, this section presents how loss functions can be obtained in order to train a CNN with the desired representation. This is one of the central contributions of this thesis.

Traditionally, CNNs have been heavily used in the domain of multi-class classification. The task here is to predict one out of $k$ classes for a given image. Usually, this is achieved by computing the softmax

$$\hat{y}_i(\mathbf{o}) = \frac{e^{o_i}}{\sum_{j=1}^{\mathsf{N_c}} e^{o_j}} \tag{57}$$

for each element $o_i$ of the last layer's output $\mathbf{o}$ in order to obtain the posterior probability $\hat{y}_i$ for the $i$-th of $N_c$ classes (cf. e.g. [22]). The predicted class is the one with highest probability. For training, a one-hot-encoded vector is supplied to the CNN with the sought-after class having a numerical value of 1 and all other classes a value of 0. For optimizing the weights, the categorical cross entropy loss between the CNN's output and the label vector is computed. The gradient of the computed loss is then backpropagated through the neural network and the weights are iteratively optimized by changing them in the direction of the negative gradient (cf. Sec. 3.1.2).

When dealing with the previously mentioned attribute representations, it is infeasible to use the combination of softmax and the categorical cross entropy loss function for training. The reason for this is that the softmax produces a vector of probabilities where at most one element can become 1 and all elements have to sum to 1. In the attribute representations considered in this thesis, however, there may exist multiple binary values (PHOC) or the representation to be predicted might be real-valued (SPOC, DCToW and the decorrelated representations). For binary representations, one can of course use the sigmoid as activation function in the last layer instead of the softmax in order to predict individual probabilities for each attribute. This, however, leads to the question what loss function is most suitable for this situation. A straightforward approach would be to make use of the Euclidean Loss

$$l_{\mathcal{N}}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \sum_{i=1}^{\mathsf{N}_s} \left|\left|\mathbf{y}^{(\mathrm{i})} - \hat{\mathbf{y}}^{(\mathrm{i})}\right|\right|^2 \tag{58}$$

in order to train the network with the $\mathsf{N}_s$ available samples. This, however, bares the drawback, that the overall gradient is scaled by the derivative of the sigmoid in the last layer [128][3]. If the initialization of the network's weights leaves the sigmoid neurons in a saturated state after a forward pass, i.e., the input to the sigmoid has a large absolute value, this causes a slow convergence behavior or even a complete stall in training. The sigmoid activation function could, of course, be replaced by a linear function. While this would eliminate the gradient being scaled by the derivative of the sigmoid, the output of the CNN would not be bounded anymore. It could thus produce values below 0 or above 1 when tasked to predict the PHOC representation. Whether a sigmoid or a linear activation is used, there exists another disadvantage: By using the Euclidean Loss one implicitly assumes that the Euclidean distance is a feasible metric for comparing the output of the CNN and the label vectors. For high-dimensional vectors the ratio of closest and farthest points approaches 1 for the Euclidean distance [1, 37]. As the dimensionality of the attribute representations to be used is typically between 500 and 1000, the Euclidean Loss is not feasible for the proposed method.

It is obvious, that the activation function in the last layer of a neural network and the loss function used for training are tightly coupled. Determining suitable combinations is thus paramount for designing networks which allow for word spotting using attribute representations. A very elegant framework for finding these combinations are Generalized Linear Models (GLMs). In the following, loss functions and corresponding activation functions for the last layer of a neural network are derived from these statistical models. This allows for interpreting the training from a probabilistic perspective.

The rest of this chapter is organized as follows: First, an introduction to GLMs is given in Sec. 5.3.1. The ensuing sections then explain, how GLMs can be used in order to derive loss functions for binary (Sec. 5.3.2) as well as real-valued attribute representations (Sec. 5.3.3). Please note that, for the sake of a clean presentation, mathematical deductions are kept to a minimum in the text. The interested may find complete deductions for certain equations in dedicated sidebars throughout the chapter.

### 5.3.1 *Generalized Linear Models*

Generalized Linear Models (GLMs) are statistical tools for predicting the conditional expected value $\mathbb{E}\left[Y \mid \boldsymbol{X} = \mathbf{x}\right]$ of a random variable $Y$ conditioned on an independent random variable $\boldsymbol{X}$, e.g., [174, p. 281]. When using a GLM, two important assumptions are made: It is assumed that $Y$ follows a distribution from the exponential family and the expected value $\mathbb{E}[Y]$ depends on a transformation of a linear combination of $\boldsymbol{X}$. The first assumption can be expressed by the inner product

$$\eta(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \tag{59}$$

3 The proof for this can also be found in the appendix, Sec. B.2.2

which is called *linear predictor* in the context of GLMs. Here, $\mathbf{w}$ are the weights of the GLM and $\mathbf{x}$ is a realization of $\boldsymbol{X}$. In order to transform the linear prediction into the correct range, a so-called *link function* $g$ is used. It is defined as the function mapping from conditional expectation to the linear prediction (cf. e.g. [174, p. 283]):

$$\eta(\mathbf{x}) = g\left(\hat{\mathbb{E}}\left[Y \mid \boldsymbol{X} = \mathbf{x}\right]\right). \tag{60}$$

The inverse of this function can then used in order to perform the mapping from linear prediction of the GLM to the predicted conditional expected value:

$$\hat{\mathbb{E}}\left[Y \mid \boldsymbol{X} = \mathbf{x}\right] = g^{-1}\left(\mathbf{w}^T\mathbf{x}\right). \tag{61}$$

In general, any link function can be used in a GLM as long as its inverse is defined and maps the linear prediction $\eta$ into a range suitable for the assumed distribution of $Y$ [174, p. 286]. However, there exists a *canonical link function* for each distribution in the exponential family for which a number of statistical characteristics can be shown. In the context of the thesis, the most important of these is that minimizing the negative log-likelihood of the model with respect to some data converges to the maximum likelihood estimate [174, p. 286]. For obtaining the canonical link function the formulation of the Probability Density Function (PDF) of $Y$ (or Probability Mass Function (PMF) in the case of $Y$ being discrete) is first rearranged into the standard exponential family form

$$f_{\exp}\left(\mathbf{x} \mid \boldsymbol{\theta}\right) = h(\mathbf{x}) \cdot \exp\left(\boldsymbol{\nu}(\boldsymbol{\theta})^T\mathbf{t}(\mathbf{x}) - \mathbf{a}(\boldsymbol{\theta})\right), \tag{62}$$

where $\boldsymbol{\theta}$ are the parameters of the distribution (cf. e.g. [124, p. 282]). The canonical link function is then equivalent to the function $\boldsymbol{\nu}$ (cf. e.g. [124, p. 291]). Sidebar 1 exemplarily shows how the canonical link function can be obtained if $Y$ follows a Bernoulli distribution.

Training a GLM happens in a supervised fashion, meaning it requires an annotated dataset

$$\mathsf{S} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{\mathsf{N}_s}, \tag{63}$$

where $\mathsf{N}_s$ is the number of training samples. The weights of the GLM can then be obtained through Maximum Likelihood Estimation (MLE). This training procedure shall be explained using the following example: It is assumed that all samples in $\mathsf{S}$ are independent and identically distributed (i.i.d.) and that all labels $y^{(i)}$ follow a normal distribution. The canonical link function for the normal distribution is simply the identity function [174, p. 283]. Thus the expected value is predicted by

$$\hat{\mathbb{E}}\left[Y \mid \boldsymbol{X} = \mathbf{x}\right] = \mathbf{w}^T\mathbf{x} = \hat{y}\left(\mathbf{x} \mid \mathbf{w}\right) \tag{64}$$

For the sake of presentation, $\hat{y}\left(\mathbf{x} \mid \mathbf{w}\right)$ will be abbreviated to $\hat{y}$ in the following. The goal now is to maximize the likelihood of the data given the predictions from the model. As $Y$ is assumed to follow a normal distribution, its PDF is

$$f_{\mathcal{N}}\left(y \,\middle|\, \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right). \tag{65}$$

As the variance $\sigma^2$ is not predicted by the GLM, it needs to be defined by the user. Typically, it is simply set to 1. The likelihood of the data given the model can then be computed by

$$L(\mathbf{w}|S) = \prod_{i=1}^{\mathsf{N}_s} f_{\mathcal{N}}\left(y^{(i)} \,\middle|\, \hat{y}^{(i)}, 1\right). \tag{66}$$

**Sidebar 1: Obtaining the canonical link function for a GLM**

The following example shows how to obtain the canonical link function for a specific GLM[4]. For the sake of this example, it is assumed that $Y$ follows a Bernoulli distribution. The PMF of $Y$ is defined as follows:

$$f_{\mathcal{B}}\left(Y = y \,|\, p\right) = p^y \left(1-p\right)^{1-y} \text{ for } y \in \{0,1\}$$

The next step is to bring the PMF into the standard exponential family form:

$$\begin{aligned} f_{\mathcal{B}}\left(Y = y \,|\, p\right) &= p^y \left(1-p\right)^{1-y} \\ &= \exp\left(\log\left(p^y \left(1-p\right)^{1-y}\right)\right) \\ &= \exp\left(y\log(p) + (1-y)\log(1-p)\right) \\ &= \exp\left(y\log(p) - y\log(1-p) + \log(1-p)\right) \\ &= \exp\left(\log\left(\frac{p}{1-p}\right)\cdot y + \log(1-p)\right) \end{aligned}$$

The canonical link function is thus

$$g_{\mathcal{B}}(x) = \log\left(\frac{x}{1-x}\right).$$

The model parameters $\mathbf{w}$ now need to be chosen such that $L$ is maximized. As is common for similar MLE problems, the parameters can also be estimated by minimizing the negative log-likelihood of the model given the data (see sidebar 2 for a complete derivation):

$$\begin{aligned} \hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} - \log L(\mathbf{w}|S) &= -\sum_{i=1}^{\mathsf{N}_s} f_{\mathcal{N}}\left(y^{(i)} \,\middle|\, \hat{y}^{(i)}, 1\right) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2}\sum_{i=1}^{\mathsf{N}_s}\left(y^{(i)} - \hat{y}^{(i)}\right)^2. \end{aligned} \tag{67}$$

Similar to neural networks, the optimization is then carried out by running (stochastic) gradient descent in order to minimize the function in Eq. 67 [124, p. 292].

As can be seen from Eq. 61, a GLM can be interpreted as a Perceptron (cf. Sec. 3.1.1) with the inverse of the link function serving as activation function. The output of a Perceptron is thus the estimated conditional expected value of a random variable $Y$. This characteristic can be generalized to Multilayer Perceptrons (MLPs) and deeper neural networks: In order to obtain a possibly deep MLP from a GLM one needs to replace the linear predictor with a neural network. Replacing the linear predictor can be done without invalidating any other aspect of the GLM [174, p. 287]. In fact, this is the exact approach chosen by Generalized Nonlinear Models (GNLMs) [97]. These variants of GLMs replace the linear predictor by a non-linear relation between the independent and dependent variable. A neural network can thus be interpreted as a form of GNLM and its output as a prediction for the conditional expected value. This allows for applying all the insights gained from GLMs to neural networks as well. Especially, training a neural network with

4 This example originates from an online blog available at `http://willwolf.io/2017/05/18/minimizing_the_negative_log_likelihood_in_english/`

**Sidebar 2: Fitting a GLM with labels following a normal distribution**

Given a dataset $\mathsf{S} = \left\{\mathbf{x}^{(s)}, y^{(s)}\right\}_{i=1}^{N_s}$, the assumption is that the labels follow a normal distribution with the variance $\sigma^2$ to be determined by the user. The PDF for the normal distribution is defined as

$$f_{\mathcal{N}}\left(y \,\middle|\, \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right).$$

The output

$$\hat{y}\left(\mathbf{x}^{(i)} \,\middle|\, \mathbf{w}\right) = \hat{y}^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)}.$$

is the GLM's prediction for the expected value $\mu$. For training the GLM the likelihood function

$$L\left(\mathbf{w} \,|\, \mathsf{S}\right) = \prod_{i=1}^{\mathsf{N}_s} f_{\mathcal{N}}\left(y^{(i)} \,\middle|\, \hat{y}^{(i)}, \sigma^2\right)$$

is maximized or, alternatively, the negative log-likelihood is minimized in order to estimate the weights $\hat{\mathbf{w}}$ of the GLM:

$$\begin{aligned}
\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{i=1}^{\mathsf{N}_s} \log f_{\mathcal{N}}\left(y^{(i)} \,\middle|\, \hat{y}, \sigma^2\right) \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{i=1}^{\mathsf{N}_s} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2\sigma^2} \sum_{i=1}^{\mathsf{N}_s} \left(y^{(i)} - \hat{y}^{(i)}\right)^2
\end{aligned}$$

specific loss functions and activation functions in the last layer can be interpreted as MLE of the network's weights. In addition, it allows for determining the assumptions made when training with different loss functions. For example, assuming that the labels follow a multivariate normal distribution with the identity matrix as covariance matrix one obtains the Euclidean Loss (Eq. 58) as function to be minimized (a complete derivation for this is given in sidebar 3). This means that training a neural network with the Euclidean Loss using a linear activation in the last layer is equivalent to MLE of the network's parameters while assuming that all labels are observations of a random variable following a multivariate normal distribution with the identity matrix as covariance matrix.

Of course, this probabilistic perspective on neural networks can also be used in reverse: By defining the distribution $Y$ of the labels one can derive an activation function for the last layer of the neural network as well as a loss function specifically for this distribution: The activation function in the last layer is the inverse of the canonical link function for $Y$. The loss function is simply the negative log-likelihood function for $Y$ given a set $\mathsf{S}$ of i.i.d. data samples. If these two conditions are met, the neural network predicts the conditional expected value $\mathbb{E}\left[Y \,|\, \boldsymbol{X}\right]$ as does the GLM. If a multivariate distribution is used, the above formulation can be generalized to the neural network predicting $\mathbb{E}\left[\boldsymbol{Y} \,|\, \boldsymbol{X}\right]$ (cf. e.g. [124, p. 295] for how this is done for GLMs). The insight presented above will be

**Sidebar 3: Derivation of the Euclidean Loss**

Given a dataset $\mathsf{S} = \left\{\left(\mathbf{x}^{(s)}, \mathbf{y}^{(s)}\right)\right\}_{i=1}^{\mathsf{N}_s}$, the assumption is that the labels follow a multivariate normal distribution of dimensionality $\mathsf{D}$ with the identity matrix as covariance matrix. The PDF can thus be expressed as composition of $\mathsf{D}$ independent normal distributions

$$f_{\mathcal{N}}\left(y \,|\, \mu, 1\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y-\mu)^2\right).$$

The prediction $\hat{\mathbf{y}}$ of the neural network for a specific sample $\mathbf{x}$ is equivalent to the conditional expected value $\boldsymbol{\mu}$ given the sample and the weights $\mathbf{w}$ of the neural network. As the individual dimensions are independent the likelihood function given the set $\mathsf{S}$ can be expressed as

$$L\left(\mathbf{w} \,|\, \mathsf{S}\right) = \prod_{i=1}^{\mathsf{N}_s} \prod_{j=1}^{D} f_{\mathcal{N}}\left(y_j^{(i)} \,\middle|\, \hat{y}_j^{(i)}, 1\right),$$

where $\mathsf{N}_s$ is the number of samples. Instead of maximizing this function one can also minimize the negative log-likelihood:

$$\begin{aligned}
\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{i=1}^{\mathsf{N}_s} \sum_{j=1}^{D} \log f_{\mathcal{N}}\left(y_j^{(i)} \,\middle|\, \hat{y}_j, 1\right) \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{i=1}^{\mathsf{N}_s} \sum_{j=1}^{\mathsf{D}} \log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{1}{2}\left(y_j^{(i)} - \hat{y}_j^{(i)}\right)^2 \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{\mathsf{N}_s} \sum_{j=1}^{\mathsf{D}} \frac{1}{2}\left(y_j^{(i)} - \hat{y}_j^{(i)}\right)^2 \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{\mathsf{N}_s} \sum_{j=1}^{\mathsf{D}} \left(y_j^{(i)} - \hat{y}_j^{(i)}\right)^2 \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{\mathsf{N}_s} \left|\left|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right|\right|^2
\end{aligned}$$

The function to be minimized is thus exactly the Euclidean Loss (cf. Eq. 58).

leveraged in the following two subsections in order to obtain loss functions for training a neural network with attribute as well as attribute-like representations.

### 5.3.2 *Loss Function for Attribute Representations*

The goal of this section is to derive a loss function which allows for using true attribute representations, i.e., binary vectors such as the PHOC as labels. For illustration purposes, the labels are considered to be binary scalars first. The derived loss function is then generalized in order to account for binary vectors as labels.

When using binary scalars as labels, it can be reasonably assumed that the dependent variable $Y$ follows a Bernoulli distribution. As explained in the previous section, the activation function in the last layer of the neural network used must be the inverse of the canonical link function. For a Bernoulli distributed random variable $Y$, the inverse canonical link function is the sigmoid function (cf. e.g. [124, p. 291]:

$$\operatorname{sigm}(x) = \frac{1}{1+\exp(-x)}. \tag{68}$$

It squashes the output of the network to the range $(0, 1)$. In order to find the corresponding loss function, the negative log-likelihood function is computed. For this, the PMF of a Bernoulli distributed variable $Y$ is required, which is defined as

$$f_{\mathcal{B}}\left(Y = y \,|\, p\right) = p^y \left(1-p\right)^{1-y} \text{ for } y \in \{0,1\}, \tag{69}$$

where $p$ is the probability $P(Y = 1)$. The negative log-likelihood function given a training set $\mathsf{S}$ of i.i.d. samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ then computes to

$$\begin{aligned} l_{\mathcal{B}} &= -\sum_{i=1}^{N_s} f_{\mathcal{B}}\left(y^{(i)} \,\middle|\, \hat{y}^{(i)}\right) \\ &= -\sum_{i=1}^{N_s} y^{(i)} \log \hat{y}^{(i)} + \left(1-y^{(i)}\right) \log\left(1-\hat{y}^{(i)}\right), \end{aligned} \tag{70}$$

where $\hat{y}^{(i)}$ is the prediction of the network for the $i$-th sample. This value can be shown to be the predicted posterior probability of $Y$ given $\boldsymbol{X}$: As $Y$ follows a Bernoulli distribution, its expected value is $p$. The neural network predicts the conditional expected value $\mathbb{E}\left[Y \,|\, \boldsymbol{X}\right]$ which is $P\left(Y = 1 \,|\, \boldsymbol{X}\right)$, i.e., the posterior probability. As explained in the previous section, Eq. 70 is exactly the loss function to be optimized. It can be shown, that $l_{\mathcal{B}}$ is equivalent to the cross entropy between the distribution of the network's predictions and the label distribution [128] which is why $l_{\mathcal{B}}$ is often times referred to as Binary Cross Entropy Loss (BCEL). Due to the sigmoid function being used here, another common name is Sigmoid Cross Entropy Loss [135, 175].

As explained before, the labels are assumed to be scalars up to this point. A straight forward approach in order to account for vectors of binary variables would be to assume that $\boldsymbol{Y}$ follows a multivariate Bernoulli distribution [32]. This approach, however, has the drawback that the neural network used would have to have an output layer size of $2^{\mathsf{D}}$ where $\mathsf{D}$ is the dimensionality of the attribute representation. Typical attribute representations for word spotting have a size greater or equal to 540 [9, 150] which would demand an output layer size greater than $3.599 \cdot 10^{162}$. This is, of course, technically impossible to realize. In order to make the problem tractable, the assumption is made that each label vector is a collection of $\mathsf{D}$ pairwise independent and Bernoulli distributed variables, each having their own probability $p$ of evaluating to 1. This way, the negative log-likelihood (and thus the loss function) is simply the sum over all negative log-likelihoods for the different variables:

$$l_{\mathcal{B}} = -\sum_{i=1}^{\mathsf{N}_s} \sum_{j=1}^{\mathsf{D}} y_j^{(i)} \log \hat{y}_j^{(i)} + \left(1-y_j^{(i)}\right) \log\left(1-\hat{y}_j^{(i)}\right). \tag{71}$$

A complete derivation for this generalization of the BCEL can be found in sidebar 4. In most deep learning tool boxes, BCEL functions typically make use of the formulation presented in Eq. 71 thus assuming independence between the individual attributes. This

**Sidebar 4: Derivation of the Binary Cross Entropy Loss**

Given the dataset $\mathsf{S} = \left\{\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right\}_{i=1}^{\mathsf{N}_s}$ where $y_j^{(i)} \in \{0,1\} \quad \forall \quad j$, the assumption is that each label $\mathbf{y}$ is a collection of independent random variables were each variable follows an individual Bernoulli distribution with PMF

$$f_{\mathcal{B}}\left(y \,|\, p_j\right) = p_j^y \left(1 - p_j\right)^{1-y} \text{ for } y \in \{0,1\}$$

Here, $p_j$ is the probability for the $j$-th element of $\mathbf{y}$ to evaluate to 1. The logarithmic PMF for the Bernoulli distribution is hence

$$\log f_{\mathcal{B}}\left(y \,|\, p_j\right) = y \log p_j + (1-y) \log\left(1 - p_j\right).$$

The prediction $\hat{\mathbf{y}}$ of the neural network for a specific sample $\mathbf{x}$ is equivalent to the conditional expected value $\mathbf{p}$ (the vector of all probabilities $p_j$) given the sample and the weights $\mathbf{w}$ of the neural network. As the individual dimensions are independent the likelihood function given the set $\mathsf{S}$ can be expressed as

$$L\left(\mathbf{w} \,|\, \mathsf{S}\right) = \prod_{i=1}^{\mathsf{N}_s} \prod_{j=1}^{\mathsf{D}} f_{\mathcal{B}}\left(y_j^{(i)} \,\middle|\, \hat{y}_j^{(i)}\right),$$

where $\mathsf{N}_s$ is the number of samples and $\mathsf{D}$ is the dimensionality of the labels. As was done for the Euclidean Loss (cf. sidebar 3) the negative log-likelihood is minimized instead of maximizing the likelihood in order to obtain the weights $\mathbf{w}$ of the neural network:

$$\begin{aligned}\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{i=1}^{\mathsf{N}_s} \sum_{j=1}^{\mathsf{D}} \log f_{\mathcal{B}}\left(y_j^{(i)} \,\middle|\, \hat{y}_j^{(i)}\right) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{i=1}^{\mathsf{N}_s} \sum_{j=1}^{\mathsf{D}} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right) + \left(1 - y_j^{(i)}\right) \log\left(1 - \hat{y}_j^{(i)}\right)\end{aligned}$$

As can be seen, the function to be minimized is the Binary Cross Entropy Loss.

loss function will be used when training the proposed Attribute CNNs with the PHOCs as labels. The training process can then be interpreted as follows: Training a neural network with sigmoid activation functions in the last layer using the BCEL is equivalent to MLE of the network's weights given the training set and assuming that the labels are vectorial realizations of independent random variables, each following a Bernoulli distribution.

### 5.3.3 *Loss Function for Attribute-like Representations*

When dealing with attribute-like, i.e., real-valued representations, a straight forward approach would be to assume that the labels follow a multivariate normal distribution. Computing the activation function and loss function for this distribution leads to the Euclidean Loss $l_{\mathcal{N}}$ (Eq. 58, also cf. sidebar 3). For high-dimensional representations such as the ones used in this thesis, the Euclidean Loss is infeasible due to the drawbacks explained in Sec. 5.3. Hence, a different distribution family is assumed for the real-valued representa-

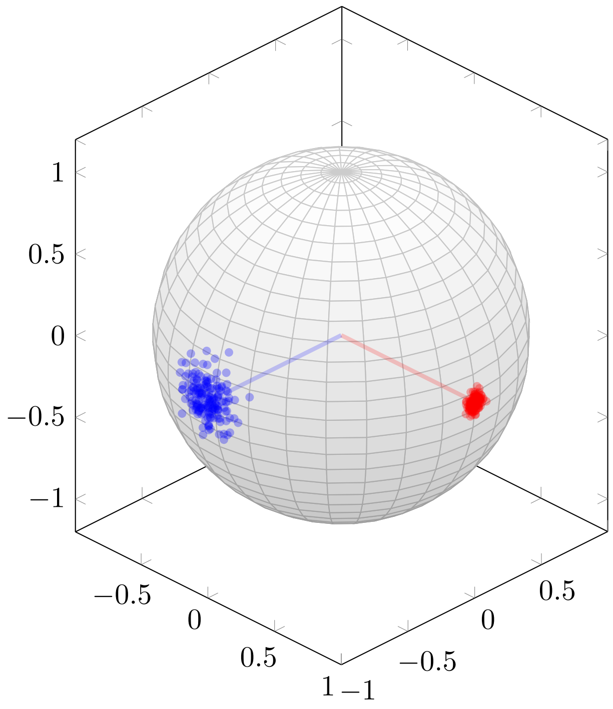


Figure 22: Visualzation of samples drawn from two different von Mises-Fisher distributions of dimensionality 3 (red and blue). The mean vector of each distribution is shown as line from the origin. The distribution painted in red has a larger concentration parameter than the one painted in blue.

tions in the following. For this, a distinct trait of these embeddings is exploited: When dealing with real-valued attribute representations, the cosine dissimilarity (cf. Eq. 45) has shown itself to be effective for computing similarity between two vectors, e.g., [150, 206]. As Sra [184] and Mardia [118] claim, if the cosine distance is an effective metric for data from a certain domain, this data has intrinsic *directional* characteristics. One of these characteristics is that the direction of the data vectors is the only relevant aspect for comparing them while the individual magnitudes do not matter. As the direction is the defining criterion, all data samples can as well be normalized, effectively projecting them onto the unit hypersphere.

For real-valued representations, the loss function is thus derived based on a directional distribution[5]. In order to obtain a suitable loss function, a directional distribution needs to be chosen which belongs to the exponential family. One possible distribution for this is the von Mises-Fisher distribution. Its PDF for a D-dimensional vector is defined as

$$f_{\mathcal{MF}}(\mathbf{y}, \boldsymbol{\mu}, \kappa) = c(\mathsf{D}, \kappa) \exp\left(\kappa \boldsymbol{\mu}^T \mathbf{y}\right) \tag{72}$$

where $\boldsymbol{\mu}$ is the mean direction, $\kappa$ is the concentration parameter and $c_\mathsf{D}$ is a normalization constant, depending on the dimensionality D of the data and $\kappa$ [184]. The vectors $\boldsymbol{\mu}$ and $\mathbf{x}$ are required to be have unit length. The von Mises-Fisher distribution can be considered as a normal distribution on the D-dimensional unit hypersphere with the mean direction as analogy to the mean and the concentration parameter as (inverse) analogy to the variance.

5 The term *directional distribution* refers to distributions which are defined on a $d$-dimensional unit hypersphere. An extensive discussion of directional distributions and directional statistics can be found in, e.g., [184].

The density value for a given sample, however, depends on the angle of the sample to the mean direction and not its Euclidean distance. Fig. 22 displays two examples of von Mises-Fisher distributions. In the figure, the distribution shown in blue has a lower concentration than the one shown in red.

In order to derive the corresponding activation and loss functions from this distribution, the simplifying assumption is made that $\kappa = 1$. The PDF of the von Mises-Fisher distribution thus boils down to

$$f_{\mathcal{MF}}(\mathbf{y}, \boldsymbol{\mu}, 1) = c(\mathsf{D}, 1) \exp\left(\boldsymbol{\mu}^T \mathbf{y}\right) \tag{73}$$

This formulation of the PDF is already in the standard form of the exponential family (cf. Eq. 62). The natural parameter for the von Mises-Fisher distribution with $\kappa = 1$ is simply the mean vector $\boldsymbol{\mu}$. This means that the canonical link function is simply the identity function. As a consequence, the neural network has to make use of a linear activation in the last layer when predicting labels from a von Mises-Fisher distribution. However, the output of the network still needs to be $L_2$-normalized. This is due to the network predicting $\mathbb{E}\left[\boldsymbol{Y} \mid \boldsymbol{X}\right]$. In the case of the von Mises-Fisher distribution, this expected value is $\boldsymbol{\mu}$ which by definition has to be normalized.

Assuming that the concentration parameter is set to 1, minimizing the negative log-likelihood leads to the following loss function (a complete derivation can be found in sidebar 5):

$$l_{\mathcal{MF}} = \sum_{i=1}^{\mathsf{N}_s} 1 - \cos\left(\mathbf{y}^{(\mathrm{i})}, \hat{\mathbf{y}}^{(\mathrm{i})}\right). \tag{74}$$

The loss is thus simply the cosine distance between the distribution of predictions and the label distribution (Eq. 45) which is why it is known as Cosine Loss. Although this loss in itself is not novel, e.g., [28], this is, to the best of the author's knowledge, the first time it has been theoretically motivated from a statistical point of view. As was done for the BCEL, this allows for interpreting the training process: Using the Cosine Loss for training a neural network with linear activation function in the last layer is equivalent to MLE of the network's parameters and assuming that the labels follow a von Mises-Fisher distribution.

Interestingly, there is a connection between the Cosine Loss and the Euclidean Loss (Eq. 58) which to the best of the author's knowledge has not been shown before: If the output of the neural network and the labels are both normalized then training with the Euclidean Loss is equivalent to training with the Cosine Loss (see sidebar 6).

Apart from being used for real-valued attribute representations, the Cosine Loss can also be used for binary representations such as the PHOC. This is feasible as PHOC representations are compared effectively using the cosine dissimilarity, e.g., [9, 91, 165, 206], hence exhibiting directional characteristics as well. The added benefit of using the Cosine Loss for binary representations from a theoretical point of view is that the assumption of independence between the different attributes is eliminated.

## 5.4 ATTRIBUTE CNN ARCHITECTURES FOR WORD SPOTTING

The loss functions from the previous section could simply be attached to any CNN architecture in order for the network to learn to predict attribute representations. However, most CNN architectures have been designed with the task of operating on natural images in

**Sidebar 5: Derivation of the Cosine Loss**

Given the dataset $\mathsf{S} = \left\{\left(\mathbf{x}^{(s)}, \mathbf{y}^{(s)}\right)\right\}_{i=1}^{\mathsf{N}_s}$, the assumption is that the labels follows a von Mises-Fisher distribution with concentration parameter 1. The PDF for this von Mises-Fisher distribution is defined as

$$f_{\mathcal{MF}}\left(\mathbf{y} \,|\, \boldsymbol{\mu}, 1\right) = c(\mathsf{D}, 1) \exp\left(\boldsymbol{\mu}^T \mathbf{y}\right)$$

Where $c(\mathsf{D}, 1)$ is a constant normalization factor. As was shown in Sec. 5.3.1, the prediction $\hat{\mathbf{y}}$ of the neural network is equivalent to the conditional expected value of the label distribution. In the case of the von Mises-Fisher distribution this is simply the expected value $\boldsymbol{\mu}$. Note that both $\boldsymbol{\mu}$ and $\mathbf{y}$ are expected to have unit length. The neural network's output $\hat{\mathbf{y}}$ is hence also expected to be normalized. Training the weights of the neural network through MLE is then done by minimizing the negative log-likelihood as was done previously for the Euclidean Loss (sidebar 3) and BCEL (sidebar 4):

$$\begin{aligned}
\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{s=1}^{\mathsf{N}_s} \log f_{\mathcal{MF}}\left(\mathbf{y}^{(\mathrm{s})} \,\middle|\, \hat{\mathbf{y}}^{(\mathrm{s})}, 1\right) \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{s=1}^{\mathsf{N}_s} \log c(\mathsf{D}, 1) + \left\langle \hat{\mathbf{y}}^{(\mathrm{s})}, \mathbf{y}^{(\mathrm{s})} \right\rangle \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} - N_s \log c(\mathsf{D}, 1) + \sum_{s=1}^{\mathsf{N}_s} - \left\langle \hat{\mathbf{y}}^{(\mathrm{s})}, \mathbf{y}^{(\mathrm{s})} \right\rangle \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{s=1}^{\mathsf{N}_s} 1 - \left\langle \hat{\mathbf{y}}^{(\mathrm{s})}, \mathbf{y}^{(\mathrm{s})} \right\rangle \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{s=1}^{\mathsf{N}_s} 1 - \cos\left(\hat{\mathbf{y}}^{(\mathrm{s})} \mathbf{y}^{(\mathrm{s})}\right)
\end{aligned}$$

The equivalence relation for the last step is only valid as long as $\hat{\mathbf{y}}$ and $\mathbf{y}$ are normalized as was assumed above. Instead of performing the normalization beforehand, it can also be explicitly integrated into the loss function:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{s=1}^{\mathsf{N}_s} 1 - \frac{\left\langle \hat{\mathbf{y}}^{(\mathrm{s})}, \mathbf{y}^{(\mathrm{s})} \right\rangle}{\left|\left|\hat{\mathbf{y}}^{(\mathrm{s})}\right|\right| \cdot \left|\left|\mathbf{y}^{(\mathrm{s})}\right|\right|}$$

The resulting function to be minimized is exactly the Cosine Loss.

mind. The problem in this thesis is concerned with document or word images which often times have different properties than can be found in typical natural images. Hence, three different CNN architectures are proposed in this thesis which are specifically designed to be applied to document images. These architectures are presented in the following subsections.

**Sidebar 6: Connection between Euclidean Loss and Cosine Loss**

Assuming that the prediction $\hat{\mathbf{y}}$ of a neural network and the desired label $\mathbf{y}$ are both normalized then the Euclidean Loss (Eq. 58) can be reformulated as follows:

$$\begin{aligned}\frac{1}{2}\sum_{i=1}^{\mathsf{N_s}}\left|\left|\mathbf{y}^{(\mathrm{i})}-\hat{\mathbf{y}}^{(\mathrm{i})}\right|\right|^2 &= \sum_{i=1}^{\mathsf{N_s}}\frac{1}{2}\left(\left\langle\mathbf{y}^{(\mathrm{i})},\mathbf{y}^{(\mathrm{i})}\right\rangle-2\cdot\left\langle\mathbf{y}^{(\mathrm{i})},\hat{\mathbf{y}}^{(\mathrm{i})}\right\rangle+\left\langle\hat{\mathbf{y}}^{(\mathrm{i})},\hat{\mathbf{y}}^{(\mathrm{i})}\right\rangle\right)\\ &= \sum_{i=1}^{\mathsf{N_s}}\frac{1}{2}\left(2-2\cdot\left\langle\mathbf{y}^{(\mathrm{i})},\hat{\mathbf{y}}^{(\mathrm{i})}\right\rangle\right)=\sum_{i=1}^{\mathsf{N_s}}1-\left\langle\mathbf{y}^{(\mathrm{i})},\hat{\mathbf{y}}^{(\mathrm{i})}\right\rangle\\ &= \sum_{i=1}^{\mathsf{N_s}}1-\cos\left(\mathbf{y},\hat{\mathbf{y}}\right).\end{aligned}$$

The function on the right side of the equation is the Cosine Loss (Eq. 74). Hence using the Euclidean Loss is equivalent to using the Cosine Loss if the labels and the output of the neural network are both normalized.

### 5.4.1 *PHOCNet*

As the first proposed architecture was originally used to only predict PHOCs representations [188], it was dubbed *PHOCNet*. The architecture is visualized in Fig. 23. The PHOCNet is inspired by the successful VGG16 architecture [179]. Just as for the VGG16, the convolutional layers which are closer to the image use a small amount of filters which is doubled after each pooling layer. This forces the CNN to learn to detect less and thus more general features in the first layers while giving it the possibility to learn a large number of more abstract features for the “deeper” layers. In addition to this, only $3 \times 3$ filters are used in all convolutional layers. This imposes a regularization on the filter kernels as the number of weights per layer is kept to a minimum[6] for the convolutional part of the CNN [179].

The previous two design choices work for both natural images in the case of the VGG16 as well as document or word images in the context of this thesis. However, there are certain aspects to be taken into account when designing a CNN architecture for document image analysis applications. In the case of segmentation-based word spotting, the CNN is asked to predict a representation for previously segmented word images. Usually, these word images exhibit a substantial variability in size. Typically when applying CNNs to natural images, size variations are combated by either anisotropical rescaling of the images to a fixed size [92] or by sampling different crops from the images [67, 179]. Both approaches are, however, infeasible when dealing with word images: Rescaling the input images leads to severe distortions whenever the original aspect ratio does not approximately match the desired aspect ratio. Likewise, cropping can not be easily applied as it is unclear how the obtained attribute representations for the individual crops can be merged. While in multi-class classification problems the outputs for different crops can simply be averaged, this is not possible in this case as the considered attribute embeddings contain a certain

6 The term minimum here refers to the minimal kernel size that can be used while still considering spatial information for a given filter. One could, of course, use $1 \times 1$ convolutions to reduce the number of weights even further but this would not allow the filters to learn any spatial context.

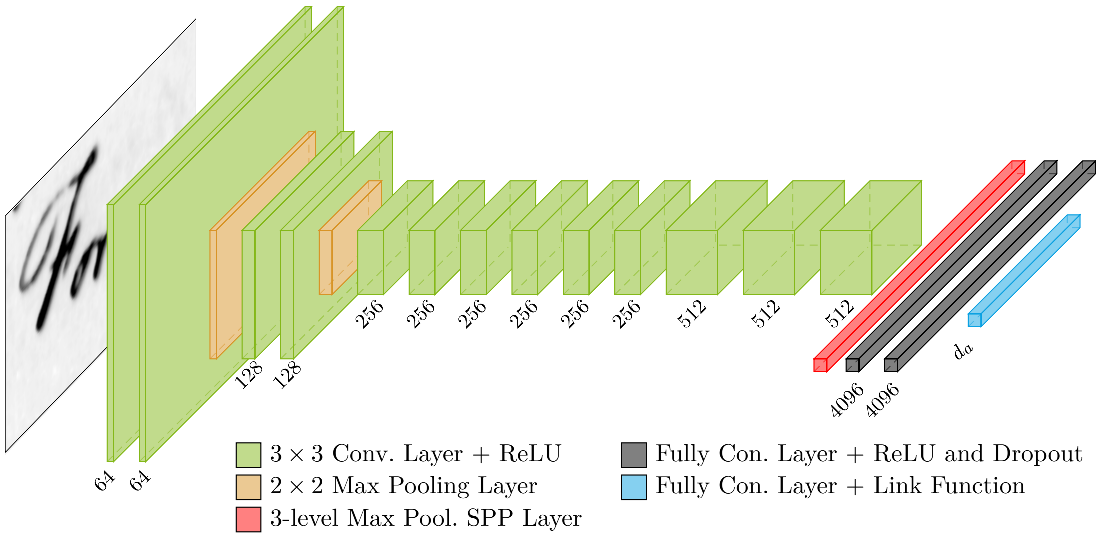

64
64
128
128
256
256
256
256
256
256
512
512
512
4096
4096
$d_a$
🟩 3 × 3 Conv. Layer + ReLU
🟧 2 × 2 Max Pooling Layer
🟥 3-level Max Pool. SPP Layer
⬛ Fully Con. Layer + ReLU and Dropout
🟦 Fully Con. Layer + Link Function

Figure 23: Visualization of the PHOCNet architecture: Green layers represent convolutional layers using $3 \times 3$ filter kernels with the respective number of filters shown underneath. All convolutional layers make use of Rectified Linear Unit (ReLU) activation functions. Orange layers depict $2 \times 2$ max pooling layers with a stride of 2. These layers effectively sample down the feature maps they are presented with. The red layer indicates a 3-level SPP layer while both block layers are fully connected layers using 4096 neurons, dropout of 50% and ReLU as activation function. The output layer size $\mathsf{D}_a$ depends on the attribute representation used as does the activation function which is simply the link function from the respective GLM model.

level of positional encoding of the attributes. Fusing outputs for different crops is thus very cumbersome and not straightforward at all.

The main problem with varying image sizes is the classifier deployed at the end of a CNN. Whether this classifier is a MLP or a Perceptron, both expect a fixed-sized input. On the other hand, the convolutional layers can be applied to arbitrarily sized images. In order to alleviate this problem, He *et al.* [65] propose to use a so called Spatial Pyramid Pooling (SPP) layer. Inspired by the original spatial pyramid paradigm [98], this layer type performs multiple pooling operations over a fixed number of regions of the last feature map of the convolutional part of a CNN. These regions vary in size in order to always span the entire feature maps obtained for a given image. Pooling is performed at various levels of granularity in a quadtree-like structure (cf. e.g. [195, p. 407]). As for the original spatial pyramid, the number of regions in a certain level along width and height is doubled with respect to the previous level. Typically, a three-level SPP is used for CNNs [65]. The first level performs a global pooling over each feature map, while the second and third level split the feature maps in four and 16 regions respectively. As pooling is performed for a fixed amount of feature map regions, an SPP layer has a fixed output representation of size

$$\mathsf{D}_{\mathrm{SPP}} = \sum_{i=0}^{\mathsf{L}_{\mathrm{SPP}}-1} 4^i \cdot \mathsf{N}_f, \tag{75}$$

where $\mathsf{L}_{\mathrm{SPP}}$ is the number of levels used in the SPP layer and $\mathsf{N}_f$ the number of feature maps presented as input to the SPP layer. While all pooling types which are applicable to normal pooling layers can be used for an SPP layer as well, max pooling is the predominantly used one (e.g. [52, 65]).

In order to be able to process input images of different sizes, an SPP layer is employed in the PHOCNet after the last convolutional layer (red layer in Fig. 23). This way, the first fully connected layer following the convolutional part is always presented with a fixed size image representation, independent of the input image size. As parameters, the standard ones are chosen, i.e., the SPP layer uses three levels and max pooling.

As the input image sizes are kept, the number of pooling layers in the convolutional part of the PHOCNet architecture is reduced with respect to the VGG16. This way, even the smallest word images in the datasets used for the experimental evaluation can be processed ($26 \times 26$ pixels for the George Washington dataset). The pooling layers are placed rather close to the input layer in order to lower the computational cost (orange layers in Fig. 23).

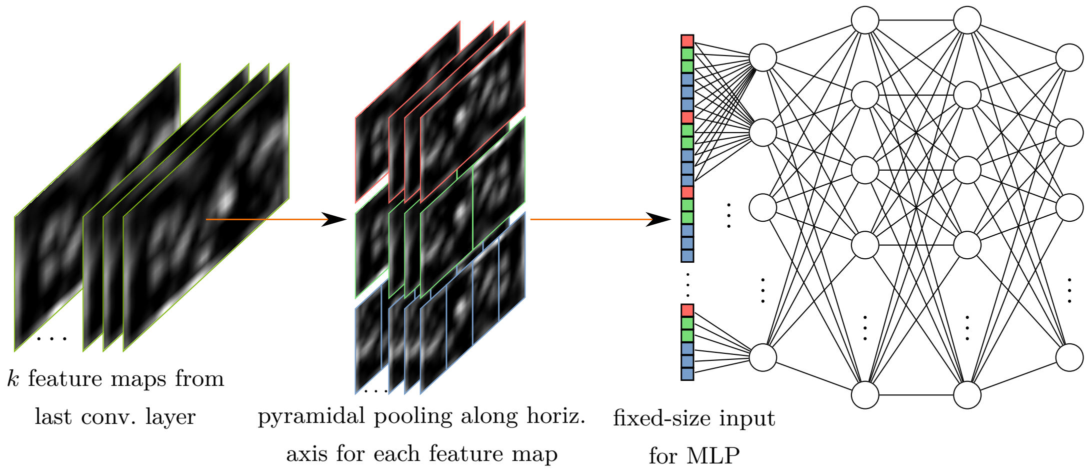
After the convolutional part, the PHOCNet makes use of an MLP with two hidden layers. The number of neurons in the two hidden layers is set to 4096 which is the same number used in the VGG16 and the AlexNet CNNs [92, 179]. Just as is done for these two CNNs, a 50% dropout is applied to the neurons in the hidden layers of the PHOCNet's MLP at training time in order to avoid overfitting. The output layer has size $d_a$ which is the dimensionality of the attribute representation to be predicted. The activation function in this layer is the link function corresponding to the loss function used during training. For the BCEL this is the sigmoid and for the Cosine Loss it is the $L_2$-normalization.

In total, the PHOCNet has $70,230,492$ trainable weights (not including the weights for the output layer as the number here depends on the size of the attribute representation used). The convolutional part of the CNN accounts for 13.4% of all weights while the hidden layers contain 86.6% of the weights. The layer with the most parameters is the first fully connected layer (the layer following the SPP layer) which contains 62.7% of all weights in the PHOCNet.

### 5.4.2 *TPP-PHOCNet*

The use of the SPP layer enables the PHOCNet to accept images of varying size as input while producing an output of constant size. The design of the SPP layer follows the one used for the "classic" spatial pyramid proposed by Lazebnik *et al.* [98]: the number of regions along width and height in a level is doubled with respect to the previous level. It can be shown that using this standard bin partitioning scheme for spatial pyramids on top of Bag of Features (BoF) representations leads to inferior results compared to other schemes for word spotting problems (e.g. [6, 164, 165]). Here, the retrieval results can be increased when using spatial pyramids featuring a fine-grained split along the horizontal axis while only using a rough partitioning for the vertical axis of a word image. This concept is pursued even further by Hidden Markov Model (HMM)-based approaches, such as SC-HMMs [149], BoF-HMMs [157, 159] or HMMs using word graphs [200], as well as methods based on Recurrent Neural Networks (RNNs) such as Bidirectional Long Short-Term Memory Networks (BLSTMs) [47]. In the case of these sequential models, the vertical axis is not partitioned at all while the partitioning of the horizontal axis is implicitly done by splitting the word image in frames and processing them as sequence. In a way, this can be seen as a probabilistic version of a spatial pyramid.

In general, choosing a fine grained partitioning along the horizontal axis and a coarser partitioning along the vertical axis is important when dealing with word images. Incorporating this observation into a neural network layer, a modified version of the SPP layer is proposed which will be referred to as Temporal Pyramid Pooling (TPP) layer. Pooling in



$k$ feature maps from last conv. layer
pyramidal pooling along horiz. axis for each feature map
fixed-size input for MLP

Figure 24: The figure visualizes a TPP layer. For every feature map from the output of the convolutional part of a CNN (left) a sequence of max pooled values is extracted in a pyramidal fashion (middle). Here, a 3-level TPP layer is visualized. The layer produces output values for 9 max pooled regions per feature map. These values are stacked as is done for the SPP layer in order to form a representation of fixed size for variably sized input feature maps. This representation is then fed to the MLP-part of the network (right).

this layer is done similarly to the PHOC pooling of binary attributes: In each level, the feature maps used as input to the TPP layer are split into horizontal regions. Each region covers the entire vertical axis of the feature map. The values obtained after pooling thus represent features from consecutive intervals of the word image along the axis of writing. When stacking multiple of these pooling layers with different amounts of splits along the axis of writing, one ends up with a pyramidal representation encoding the progression of writing, hence the name Temporal Pyramid Pooling. The TPP concept is visualized in Fig. 24. Here, the feature maps of the output of the convolutional part of the CNN used (left part of the figure) are followed by a TPP layer as described above. The number of regions are doubled for each level compared to the previous one as is done in the SPP layer.

For evaluating the effectiveness of the proposed layer, the SPP layer in the PHOCNet is swapped with the TPP layer. The rest of the PHOCNet architecture is left unchanged. In order to discriminate the new architecture from the previous one, this variant of the PHOCNet will be referred to as TPP-PHOCNet. In the experimental evaluation, a 3-level TPP layer will be used which encompasses 1, 2 and 4 regions in the respective levels. A notable benefit of using a TPP instead of a SPP layer is that the number of weights in the CNN is drastically reduced. The output dimensionality of a TPP layer is

$$\mathsf{D}_{\text{TPP}} = \sum_{i=0}^{\mathsf{L}_{\text{TPP}}-1} 2^i \cdot \mathsf{N}_f, \tag{76}$$

where $\mathsf{L}_{\text{TPP}}$ is the number of levels used in the layer and $N_f$ is, again, the number of feature maps used as input. In the TPP-PHOCNet architecture, the layer following the TPP has $1.468 \cdot 10^7$ weights while for the PHOCNet the layer following the SPP layer has $4.404 \cdot 10^7$ weights. The TPP layer hence reduces the necessary parameters in this layer by 66.7%.

### 5.4.3 *PHOCResNet*

The previous two CNN architectures are both inspired by the VGG16 CNN. Using this architecture for tasks on natural images, it could be shown that Residual Networks (ResNets) can achieve considerably better results. For evaluating the effectiveness of residual connections in CNNs for word spotting, a third architecture is proposed in this thesis. For easy reference, it will be referred to as PHOCResNet.

The proposed PHOCResNet is inspired by the successful ResNet50 [67]. One of the characteristics of ResNets is that they do not use pooling layers. Downsampling of feature maps is instead achieved through strided convolutions. In analogy to the PHOCNet, only the first two strided convolutions are used in the PHOCResNet while the others are set to have a stride of one, i.e., do not perform downsampling. The reason for this is, again, the capability of processing small word images. In order to allow the PHOCResNet to accept arbitrarily sized images, a TPP layer is used after the convolutional part of the CNN.

In addition, the classifier at the end of the CNN is replaced in the PHOCResNet with respect to the original ResNet50. This is due to the fact that ResNets typically only make use of a Perceptron after the convolutional part in order to perform classification. While in typical multi-class classification problems this setup achieves state-of-the-art results, this might not be the case for the attribute representations to be used for word spotting as they typically exhibit a substantial amount of dependencies. Even after decorrelation, a simple Perceptron may not be powerful enough in order to reliably predict the desired representation. Hence, the Perceptron is replaced by an MLP in the PHOCResNet. Thus, this specialized ResNet has a closer resemblances to architectures such as the VGGnet or the AlexNet (cf. Sec. 3.5).

The last adaption made to the original ResNet architecture is to eliminate the Batch Normalization (BN) layers. The reason for this is of a technical nature: When dealing with images of varying size, the gradients for a single batch are computed by passing the images through the network one by one and accumulating the individual gradients. Essentially, the effective mini-batch size is reduced to one and the desired mini-batch size $n_b$ is obtained by averaging $n_b$ single image mini-batches. This approach is problematic for BN layers as they expect to see an entire mini-batch of feature maps of the same size in order to compute the mean and variance for each feature map pixel. Computing mean and variance for each pixel using a mini-batch of size one does, of course, not make sense. One could compute all inputs for each mini-batch presented to a BN layer and then apply the layer to these values. Unfortunately, this increases the time necessary for a single forward pass by a large amount as this requires $n_b$ forward passes for each BN layer used in the respective CNN. But even if the added computational expenses were to be accepted, there exists another problem: If the input images and hence the feature maps vary in size, the mean and variance can only be computed for those pixel locations that are shared among all images. For example, if for a given layer half of the output feature maps have a shape of $50 \times 50$ and the other half a shape of $100 \times 100$ all pixels with a width or height index greater than 50 would not be defined for the first half. The only chance in the presented scenario for applying BN layers is to simply compute the mean and variance of all pixels in the given feature map. It is, however, questionable whether this leads to meaningful results. Due to the problems mentioned above, the BN layers are simply neglected in the PHOCResNet architecture.

Overall, the PHOCResNet has $98,980,864$ trainable parameters not including the output layer. The convolutional part is responsible for 23.7% of the weights while the MLP accounts for 76.3%.

# 6 RELATED WORK

The approach proposed in the previous chapter shares a number of characteristics with other methods from the literature. This chapter presents these related works. For this, the chapter is split into two sections: The first section deals with the related literature concerned with the core methodology of this thesis, namely predicting attributes with neural networks. The second section then examines related work in document image analysis. A major focus of this section is the use of neural networks or attribute representations for word spotting.

## 6.1 PREDICTING ATTRIBUTE REPRESENTATIONS USING CNNS

For predicting attributes using Convolutional Neural Networks (CNNs), there exist two popular approaches: For the first approach, a CNN is first trained on a multi-class classification dataset like the well-known ImageNet [167]. The resulting CNN is then used as a feature extractor by using the output of a certain layer as feature representation for a given image. This feature representations is then processed by a dedicated attribute classifier, typically one Support Vector Machine (SVM) per attribute. The concept of using the output of a certain layer in a CNN as feature representation is known as *deep feature* approach (cf. e.g. [11, 216]). It has been successfully used for predicting class or instance attributes for human pose detection [214], animal classification [4, 176], objects [135], faces [109] and natural scenes [24].

In contrast to using deep features, the second approach for predicting attributes using CNNs is to train the neural network to predict the desired attributes directly from a given image, i.e., in an end-to-end fashion. For instance attributes of objects in natural scenes, this second approach could already be shown to achieve superior results compared to using deep features [135]. Methods using the end-to-end approach are related closer to the presented Attribute CNNs than those using deep features as the neural networks presented in this thesis also predict the desired attribute representations in an end-to-end fashion.

As class and instance attributes can both be represented by a vector of binary values (cf. Sec. 2.3), using the combination of sigmoid activation function and Binary Cross Entropy Loss (BCEL) for end-to-end training of these representations is well motivated. One of the first methods which predicts attributes using this combination of activation and loss function is presented by Shankar *et al.* [175] for predicting instance attributes of natural scenes. Their approach is to learn the respective instance attributes given only a weak supervision. Shankar *et al.* define weak supervision in this context as being given only a single label for an image for which other labels would be correct as well but are missing from the annotation. The goal is then to use all images with this single annotation in order to discover all instance attributes present for a new image. For this, they iteratively determine pseudo labels from the feature map responses during training: First, an AlexNet CNN (cf. Sec. 3.5) is used to predict the single attribute annotation for each training image. After a certain amount of training iterations, the average activation in each feature map is calculated for all training images of a certain attribute label $a_i$. Then, the feature map responses for all other training images, i.e., images which are not annotated with $a_i$, are computed. These responses are then used in order to determine a pseudo-probability for $a_i$ to actually be present even though it is not contained in the annotation. Having obtained the potentially missing attributes for all images, the network is then trained again for a number of iterations using the newly obtained annotation of pseudo-probabilities for all attributes. For training the network, Shankar *et al.* make use of the BCEL. While their approach for predicting attributes in an end-to-end fashion is similar to the use of the BCEL in the approach presented in this thesis, the problem of missing attributes is not given for the word spotting task considered in this work. As the attribute representations for word strings are generated in a deterministic and automatic way from the training labels, they are always complete.

Patterson and Hays [135] use the combination of sigmoid activation function and BCEL in order to predict instance attributes for objects in natural images. As CNN architecture they choose CaffeNet which is an AlexNet-like architecture which comes with the Caffe deep learning library [74][1]. The CNN is first pretrained using the ImageNet dataset. After pretraining, Patterson and Hays replace the softmax and Categorical Cross Entropy Loss with a sigmoid activation function and the BCEL. The resulting network is then finetuned using the instance attributes for each image.

Hand and Chellappa [64] use the sigmoid activation function in tandem with the BCEL for instance attribute prediction of faces. Instead of using an existing architecture, though, they design a novel CNN which they refer to as *Multi-Task CNN*. This CNN architecture is made up of two shared convolutional layers and then branches out into six individual CNNs. Each of these branches is responsible for predicting the instance attributes from one of six previously defined groups. Each group contains semantically similar instance attributes. For example, the instance attributes *male* and *female* belong to the *gender* group. The resulting CNN is trained concatenating all outputs of the individual branches and applying the sigmoid and BCEL. Hand and Chellappa also present a second architecture in which they attach an additional fully connected layer to their previously trained Multi-Task CNN which is of the same size as the original output layer. This new last layer is then trained by only updating the new parameters in this layer and freezing all other weights in the previously trained CNN. The goal of this is to enable this new layer to incorporate knowledge from all other instance attribute predictions when making the "final" decision whether a specific instance attribute is present or not. Unfortunately, it is unclear whether the second training step is done using a second training set or with the original training data. It seems, though, that the original training data is used. The presented results support this assumption as the performance of the second architecture is almost identical to that of the first.

All related works using an end-to-end prediction of class and instance attributes presented so far have made use of the combination of sigmoid activation in the last layer and applying the BCEL during training. To the best of the author's knowledge, the only other approach for training a CNN with class attribute representations using the Cosine Loss is presented by [28] for classification of natural images[2]. The goal in [28] is to incorporate information about attribute co-occurrences in the training process in order for the CNN to find more discriminative representation of a given class. For this, Chollet projects the

1 The exact definition of the CaffeNet architecture can be found at https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet

2 The Cosine Loss is also used in [201] for multi-label classification which can be considered an instance attribute prediction.

binary attribute vectors for each class into the $\mathbb{R}^d$ by using the Pointwise Mutual Information (PMI) embedding approach as shown in Sec. 5.2. As projection matrix, he only uses those eigenvectors of the PMI matrix which have a positive corresponding eigenvalue. Having obtained a projection for each class, a CNN is trained in order to minimize the Cosine Loss of its output and the desired projected representation. As architecture, Chollet uses the Inception v3 CNN by Szegedy *et al.* [194].

## 6.2 DOCUMENT IMAGE ANALYSIS

The focus of this section is on discussing related work with respect to using attribute representations in document image analysis in general and word spotting in particular. In addition, word spotting methods will be examined which make use of neural networks concerning the relationships with the presented Attribute CNNs.

### *Related Works for Non-Word Spotting Tasks*

While attribute representations have been used quite frequently lately for word spotting, they are scarcely used for other document image analysis tasks. To the best of the author's knowledge, there exist only three approaches using attributes for non-word spotting related tasks. In the first of these three tasks, Almazán *et al.* [9] use their embedded attribute approach in order to perform lexicon-based recognition. A presentation of the embedded approach was already given in Sec. 4.4. In order to perform recognition, Almazán *et al.* simply use the nearest neighbor Pyramidal Histogram of Characters (PHOC) representation available from the lexicon as recognition result. The rest of their methodology stays the same.

The second non-word spotting related method using attributes is presented by He *et al.* [68]. They propose a method for learning attribute representations for writing styles in an unsupervised fashion in order to predict the dates of historical document images. They motivate the unsupervised approach by stating that manually creating an attribute annotation may be tedious and that defining attributes for writing styles may be hard even for experts. For obtaining the attributes, He *et al.* first represent the individual document images through a concatenation of four heuristic features which originate from writer identification. The authors then assume, that documents with the same attributes are close in feature space. In order to obtain these attributes, He *et al.* cluster the feature space into $K$ clusters. Each centroid then represents an attribute. In order to predict the attributes, $K$ linear SVMs are trained where each SVM is responsible for predicting one of the $K$ clusters, i.e., attributes. The final representation given the feature vector $\mathbf{x}$ is then obtained by computing and concatenating all scores

$$s_i = \frac{1}{1 + \exp\left(\left\langle -\mathbf{w}^{(i)}, \mathbf{x} \right\rangle + w_0^{(i)}\right)} \tag{77}$$

for the $K$ SVMs. In the equation above $\mathbf{w}^{(i)}$ and $w_0^{(i)}$ are the normal vector and offset of the hyperplane for the $i$-th SVM. He *et al.* define this as their attribute representation. They are able to show that this representation can be used effectively for the document dating task at hand. While the approach by He *et al.* has merits for obtaining writing style attributes in an unsupervised fashion, the attribute and attribute-like representations used for word spotting in this thesis do not require such an approach as they can be obtained from a given translation through a direct mapping.

The final of the three approaches for using attributes in non-word spotting related tasks is presented by Poznanski and Wolf [140]. Their proposed method is able to perform handwritten text recognition using attribute representations which are predicted from a CNN. The method is hence highly related to the proposed Attribute CNNs. In their work, Poznanski and Wolf train a CNN using a PHOC representation for word images using the sigmoid in the last layer and training with the BCEL. This approach is essentially the same as the one presented in this thesis for training Attribute CNNs with class attribute representations[3]. For creating their PHOC representation, Poznanski and Wolf not only incorporate bigram attributes but also add attributes for trigrams. Compared to the original PHOC, the global level of unigrams is added to the PHOC as well, giving it unigram levels 1 to 5. The PHOC is thus made up of 15 individual histograms for the unigram levels, two for the bigrams and another two for the trigrams, hence 19 histograms in total. Rather than predicting the entire PHOC, the CNN used by Poznanski and Wolf predicts each histogram of the attribute embedding individually in a similar fashion as is done by [64]. For this the network uses a shared convolutional part, while there exists an individual Multilayer Perceptron (MLP) with one hidden layer for each of the 19 individual histograms. Each MLP uses the sigmoid activation function in the last layer in order to predict the attributes. For training, all outputs of the individual MLPs are concatenated and the BCEL is applied. In their CNN architecture, Poznanski and Wolf [140] make use of Batch Normalization (BN) layers. In order to employ these layers, they anisotropically rescale the word images during training and evaluation to a fixed image size. In order to perform recognition, a lexicon is used and for each entry the PHOC representation is computed. Instead of using the output of the network as the representation for classification directly, Poznanski and Wolf instead concatenate the outputs of each hidden layer of the 19 MLPs and train a Canonical Correlation Analysis (CCA) model for predicting a subspace representation from the concatenated activations and the desired PHOC vectors. Recognition is then performed by projecting all PHOC encodings of the lexicon into the CCA subspace, predicting the representation from the CNN and computing the nearest neighbor from the lexicon. The authors perform an extensive experimental evaluation, determining the influence of the bi- and trigrams, the influence of using the 19 MLPs "branches" compared to a single one as well as the influence of the CCA. The results suggest, that both bi- and trigram attributes are insignificant for the final performance and that one can obtain the same results when using only unigrams in the PHOC. The CNN proposed by Poznanski and Wolf differs quite substantially from the Attribute CNNs used in this thesis. The BN layers used in the architecture demand that the input images are scaled to a fixed size. This introduces unwanted variability in the word images the CNN has to process. The Spatial Pyramid Pooling (SPP) and Temporal Pyramid Pooling (TPP) layers used for the Attribute CNNs presented in this thesis eliminate the necessity of fixed sized images.

### *Related Works for Word Spotting*

Related work considering word spotting encompasses the methods either using neural networks or those using attributes as proposed by Almazán *et al.* [9] (cf. Sec. 4.4) in the

3 Both the original work on training Attribute CNNs with the PHOC [188] and the work by Poznanski and Wolf [140] were published simultaneously.

context of this thesis. Please note that the focus of this section is on methods for word spotting in document image analysis. Methods for word spotting in audio data are not relevant for this thesis and will thus not be considered here

The first works on word spotting with neural networks did actually not make use of CNNs but rather Recurrent Neural Networks (RNNs). Inspired by similar works for word spotting in audio data [45, 208], Frinken *et al.* [47] use a Bidirectional Long Short-Term Memory Network (BLSTM) for spotting words in historical as well as contemporary handwritten document images. Their presented method is able to perform Query-by-String (QbS) word spotting but not Query-by-Example (QbE). Furthermore, Frinken *et al.* operate under a line spotting approach, meaning that they require segmented text lines for training their neural networks. In addition, instead of returning word images in the retrieval list, it is made up of text lines as well. An element in the retrieval list is considered to be relevant if the corresponding line contains the word which was used as query. The actual method by Frinken *et al.* follows the classic computer vision pipeline: As a preprocessing step, the text lines are skew and slant corrected, normalized to a unit height and then binarized. In the feature extraction step, nine geometric features are obtained for each pixel column of the text lines. Afterwards, a BLSTM is trained as is done for handwriting recognition: Given the sequential features for a given line, the neural networks is trained to predict the sequence of characters from the supplied annotation using the Connectionist Temporal Classification (CTC) algorithm by Graves *et al.* [59]. For this, the BLSTM first predicts posterior probabilities for each character, i.e., character posteriors, for every element of the feature sequence. In addition, a "blank" label is added as possible prediction. Combining all character posteriors from all frames leads to a lattice in which the paths represent possible predictions. For training the network, the probability for the path representing the annotation is maximized[4]. Having obtained a trained BLSTM, Frinken *et al.* then adapt the standard approach for handwriting recognition in order to account for word spotting: Instead of using the path with highest probability for a new text line in order to classify it into characters, they compute the entire lattice for the text lines in the corpus to be retrieved from. Then, the log-probability of observing the query string in any path is calculated for a given lattice. The retrieval list is then constructed by sorting the text lines according to their log-probabilities. Evaluating the approach on historical as well as contemporary document images of handwritten text, Frinken *et al.* were able to achieve then state-of-the-art results. In contrast to the approach presented in this thesis, the BLSTM-based approach by Frinken *et al.* does not depend on a word-level segmentation for the training data. A notable disadvantage of their method, however, is that it relies on the ability to binarize the document images in order to extract the geometric features. For document images featuring a certain amount of degradation, binarization is a difficult task. Almazán *et al.* [9] were able to show that the BLSTMs can be outperformed using SIFT-based Fisher Vectors in a sliding window-based approach on historic document images.

Another method for QbS word spotting using neural networks in sequential models is presented by Thomas *et al.* [198]. Instead of using an RNN, though, they combine an MLP with a Hidden Markov Model (HMM). In this hybrid model, the HMM is made up of individual character models with the MLP being responsible for predicting the probability of being in a given state when observing elements of a feature sequence, i.e., the state

4 A more detailed description of the exact technical properties of the CTC algorithm is not important in the context of this thesis. The interested reader is referred to [59] for a complete presentation.

posterior. As is done by Frinken *et al.*, Thomas *et al.* use their model in a classic approach: Text lines are first extracted using the connected components-based method from [132]. The text lines are then preprocessed by correcting for skew and slant, smoothing the contours of the text as well as normalizing the line images to a unit height of 54 pixels. The lines are then represented by overlapping sequential frames which each have a width of 8 pixels and stride of 3 pixels. These frames are used as input for the MLP. For training the neural network in order to predict the state of the HMM, each frame needs to be assigned a state label. In order to obtain these labels, Thomas *et al.* use the standard approach for predicting the state posteriors in hybrid models of neural networks and HMMs (cf. e.g. [105, 196]): First, a surrogate HMM using Gaussian output modeling is trained using the 26 geometric features for each of the extracted frames. Then, the Viterbi algorithm (cf. e.g. [46, p. 86]) is used in order to extract the most probable sequence of states for each training line. These states are used as labels for the corresponding frames. This frame-level annotation is then used in order to train the MLP directly on the pixels of the individual frames. As the last step, the HMM is trained on the output of the neural network for the different frames of a sequence in order to obtain the complete hybrid model. Thomas *et al.* are able to show that their hybrid approach is able to outperform a Gaussian Mixture Model (GMM)-HMM on the RIMES database[5]. One of the challenges of using an MLP for processing the frame images instead of a convolutional architecture is the number of parameters in the first layer of the neural network (cf. Sec. 3.3). The architecture used by Thomas *et al.* is hence rather small compared to other architectures published at that time, e.g., the AlexNet (cf. Sec. 3.5): Their proclaimed deep neural network is made up of three fully connected layers of $400, 350$ and $300$ neurons respectively.

Before being used for word spotting, CNNs were already used for a task called *text spotting* [72]. Although word and text spotting sound similar, the goal of the latter is to perform text recognition in natural scenes (cf. e.g. [60, 139]). The methods for text spotting are thus not directly related and will not be discussed in the following.

The first work on using CNNs for word spotting is presented by Sharma and Pramod [177]. The presented approach is able to perform segmentation-based QbE. The first step is to train an AlexNet CNN on the ImageNet dataset. The resulting CNN is then finetuned to the word images available for training for a given dataset. For the finetuning step, the 1000 most often occurring word classes are determined and the corresponding word images are extracted from the dataset. Training on this subset of word images is then carried out by having one neuron in the output layer represent each of the 1000 word classes and applying the softmax function in combination with the Categorical Cross Entropy Loss (cf. Sec. 5.3). As is done by Poznanski and Wolf [140], Sharma and Pramod scale the word images to a fixed size in order to serve as input to the neural network during training. After having trained the CNN, each word in the corpus is also scaled to the previously used fixed size and forwarded through the CNN. The output of the last hidden layer is then used as representation. At query time, this representation is predicted for the query word image. The retrieval list is obtained by sorting all word images in the respective database according to the distance of their representation to the query representation. As distance metrics, Sharma and Pramod experiment with both the $L_1$- and $L_2$-norm. They evaluate their approach on the IAM Handwriting Database (IAM-DB) as well as a second dataset featuring Telugu script. The reported results show that their approach

5 RIMES is a database of contemporary document images of handwritten text, cf. http://www.a2ialab.com/doku.php?id=rimes_database:start

is able to outperform the AttributeSVM and Platt's scaling combination proposed by Almazán *et al.* [9] on the IAM-DB but fall short of the results obtained for the tandem of AttributeSVM and Kernelized Canonical Correlation Analysis (KCCA) on this dataset. Unfortunately, Sharma and Pramod do not explain, why they choose to pretrain their CNN on natural images instead of document images. While the initial layers of a CNN are generally accepted to detect blobs and edges, the ensuing layers are trained towards detecting parts and entire objects. It is not obvious why this is generally a good starting point for finetuning the CNN to document images. Besides this, there exist a notable methodological drawback: First, the CNN is finetuned in order to learn word classes. Hence, the network is forced to discriminate two words differing only in a single letter as two different classes. Knowledge about parts of the words which are shared, i.e., the characters, is disregarded. Attribute representations have an advantage here as they are able to encode similarities between differing word classes as well.

The method by Krishnan *et al.* [91] proposes remedies for these two drawbacks of the approach by Sharma and Pramod. As a first step in their method for segmentation-based QbE and QbS word spotting, the authors also train a CNN on classes of word images. In contrast to the approach by Sharma and Pramod, however, Krishnan *et al.* do not pretrain their network on natural images but rather use the HW-SYNTH dataset from [90]. This dataset consists of synthetically generated word images of handwritten-like fonts. This fact makes it a more suitable alternative for pretraining than the ImageNet for the word spotting task. After pretraining, the resulting CNN is further finetuned to the available training images of the respective datasets. For this, all word classes are used for training. Krishnan *et al.* then go on to also use the output of the last hidden layer as feature representations for the word images. Instead of using this representation for retrieval directly, though, it serves as input to an AttributeSVM. Hence, these deep features effectively replace the Fisher Vector used as feature representations by Almazán *et al.* [9] in the attribute embedding framework. The obtained attribute classifiers are then used as is done by Almazán *et al.* in order to perform QbE as well as QbS word spotting. As attribute representation, Krishnan *et al.* also use the PHOC. They are able to show empirically that their deep feature approach is able to outperform the AttributeSVMs by Almazán *et al.* [9] as well as the results obtained for the initial publication of the PHOCNet [188] on the IAM-DB and George Washington Database (GW). The main conceptual difference between the method by Krishnan *et al.* and the proposed Attribute CNNs is that the deep feature approach does not allow for end-to-end training as the CNN and the SVMs need to be trained separately. As the ability of end-to-end training has already shown to obtain superior results in other computer vision tasks compared to the classic pipeline (cf. Chap. 2), it can be reasonably expected that this is also the case for word spotting.

The approach sharing the closest relation to the presented Attribute CNNs is proposed by Wilkinson and Brun [206]. In their method for segmentation based QbE and QbS word spotting, they also uses a CNN in order to directly predict attribute and attribute-like representations which are then used for retrieval as is done by Almazán *et al.* [9] and in this thesis. The first step in their approach is to train a CNN in order to predict discriminative feature representations from the given word images This is achieved by training the network using a triplet loss function. The general goal when training neural networks with triplet loss functions is to obtain embeddings in $\mathbb{R}^D$ for images based solely on knowledge about whether pairs of images belong to the same class or not. For this the network first computes an output representation for three different samples as input with the constraint that the classes of two of these samples match while the third class is not a match for either of the other two. The loss function then produces large values if the representations for the two matching samples are far apart in $\mathbb{R}^D$ or if the representation for the non-matching sample is close to either of the two others. When training the neural network to minimize this loss function, it learns an embedding space where samples from same classes are bunched together while those of different classes are "pushed apart". CNNs trained with a triplet loss are generally referred to as *Triplet-CNNs* in the literature (cf. e.g. [27]). As triplet loss function, Wilkinson and Brun use the SoftPN loss presented by Baltnas *et al.* [14]. It is computed by

$$\begin{aligned} l_{\text{SPN}}\left(\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \mathbf{b}\right) &= \left(\frac{e^{d\left(\mathbf{a}^{(1)},\mathbf{a}^{(2)}\right)}}{s\left(\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \mathbf{b}\right)}\right)^2 + \left(\frac{e^{\min\left(d\left(\mathbf{a}^{(1)},\mathbf{b}\right),d\left(\mathbf{a}^{(2)},\mathbf{b}\right)\right)}}{s\left(\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \mathbf{b}\right)}\right)^2, \text{ where} \\ s\left(\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \mathbf{b}\right) &= e^{\min\left(d\left(\mathbf{a}^{(1)},\mathbf{b}\right),d\left(\mathbf{a}^{(2)},\mathbf{b}\right)\right)} + e^{d\left(\mathbf{a}^{(1)},\mathbf{a}^{(2)}\right)}, \end{aligned} \tag{78}$$

where $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$ are the representations of the matching samples, $\mathbf{b}$ is the representation for the non-matching sample and $d$ is a distance metric for the embedding space. For training, the gradients with respect to each of the three samples are computed individually. For updating the weights, they are then averaged. As CNN they use the ResNet34 architecture from [67]. Having obtained the representations from the Triplet-CNN, they are then fed as input to an MLP of the same dimensionality as the ones used for the proposed Attribute CNNs. This MLP is responsible for predicting the desired attribute representation for the word image which was previously processed by the Triplet-CNN. In their work, Wilkinson and Brun propose to use the Discrete Cosine Transform of Words (DCToW) as attribute representation. As the DCToW is real-valued, they propose to use the Cosine Embedding Loss for training the MLP. This loss requires two representations $\mathbf{r}^{(1)}$ and $\mathbf{r}^{(2)}$ as input in addition to a boolean value $t$ which specifies whether the neural network shall be trained in order to minimize the angle between the two representations:

$$l_{\text{CosEmb}}\left(\mathbf{r}^{(1)}, \mathbf{r}^{(2)}, t\right) = \begin{cases} 1 - \cos\left(\mathbf{r}^{(1)}, \mathbf{r}^{(2)}\right), & \text{if } t = \text{ true,} \\ \max\left(0, \cos\left(\mathbf{r}^{(1)}, \mathbf{r}^{(2)}\right) - \gamma\right) & \text{otherwise} \end{cases}. \tag{79}$$

In the equation above, $\gamma$ is the minimum desired angle between the two representations. The Cosine Embedding Loss is very similar to loss functions used for training Siamese networks [23] which can be seen as a precursor to the Triplet-CNNs. For training, Wilkinson and Brun use the output of the MLP and a DCToW representation randomly sampled from the available training representations. If the randomly sampled representation matches the desired representation, $t$ is set to true and to false otherwise. After having trained the Triplet-CNN and the MLP, Wilkinson and Brun, combine both neural networks to a single one and perform a small amount of end-to-end training steps. Their method is able to outperform the AttributeSVM approach [9] as well as the results obtained for the initial publication of the PHOCNet [188] on the IAM-DB and GW.

# 7 EXPERIMENTAL EVALUATION

The approach using Attribute CNNs for word spotting as proposed in Chap. 5 raises two immediate questions:

**Is any combination of attribute or attribute-like representation, loss function and CNN-architecture superior or inferior to others?** Between the seven different attribute or attribute-like representations, two loss functions and three CNN architectures there exist 24 unique combinations. The question is whether any of these combinations generally stands out as being superior or inferior to all or a subset of other combinations.

**Does end-to-end learning of attributes for word spotting lead to a significant improvement in performance compared to "classic" approaches such as AttributeSVMs?** The main argument for using Attribute CNNs over non end-to-end approaches is the ability of the neural network to integrate the process of learning feature representations and the corresponding attribute classifier. In order to support this argument, there needs to be evidence for a statistically significant difference in performance.

As there does not exist a strategy for answering these two questions theoretically, the approach chosen in this thesis is to run a number of different experiments in order to supply empirical evidence for either of the answers. In order to accomplish this and keep the number of experiments handleable, the possible combinations of attribute or attribute-like representations and loss functions are first narrowed down to a feasible subset using selected datasets. The obtained subset is then used in order to determine whether any architecture is superior or inferior to the others as well as compare the performance to that of other word spotting approaches reported in the literature.

In order to run the different experiments, standard word spotting benchmarks will be used. Each of these benchmarks comes with its own evaluation protocol which may differ from other protocols. For all protocols and experiments the same measure is used, however, in order to determine the performance of a specific Attribute CNN configuration or other word spotting approaches. For this, the mean Average Precision (mAP) is chosen which has become the de-facto standard for comparing word spotting methods. Hence, this chapter starts by giving a detailed description and formal definition of the mAP (Sec. 7.1).

As will be seen in the experiments, the mAP values for different Attribute CNN configurations or word spotting approaches on a single benchmark are often times very close from a numerical point of view. In order to determine whether a difference in performance stems from mere chance or rather a systematic superiority of one configuration, a statistical significance test is run for the obtained results. For this the permutation test is chosen. Sec. 7.2 presents this specific test while also justifying its applicability for the word spotting scenario.

Sec. 7.3 presents the results obtained for the different experiments. First, it is evaluated which combination of attribute or attribute-like representation, loss function and Attribute CNN architecture is best suited for word spotting. Afterwards, the obtained results are compared to others reported in the literature. The section is closed by a discussion considering the obtained results, run times and requirements concerning training set size

Besides the two questions stated above, which generally refer to performance of the proposed Attribute CNNs, there exists a third one which is more concerned with interpretability of the predictions:

**When predicting attributes, what do trained Attribute CNNs base their decision on?** Deep neural networks today still carry the stigma of being black boxes when it comes to interpreting what they base their decision on. However, there exist a number of different techniques which allow for at least partially explaining what parts of a given image did cause a neural network to decide for or against a certain class. The last section of the chapter (Sec. 7.4) aims at examining just that, i.e., what the Attribute CNNs base their attribute predictions on. For this, the guided backpropation method is used in order to visualize the filters of the last hidden layer.

## 7.1 PERFORMANCE MEASURES

As word spotting is essentially an information retrieval problem, performance measures for specific word spotting methods are typically taken from this field of research as well. In order to describe these measures, important information retrieval terms are explained first: Information retrieval deals with the problem of extracting relevant information from a database. For this, a query is issued to the database which returns a list of items which is known as retrieval list. In the case of word spotting, the retrieval list consists of a number of word images from the corpus which serves as database. The retrieval list consist of items which are either *relevant* or *irrelevant* with respect to the query. In information retrieval in general, relevance is not globally defined and depends on the context of the application. Most works on word spotting deem an item in the retrieval list as relevant with respect to the query if the transcriptions of the query and the word image match, e.g., [9, 158, 164]. Hence, an image showing the word *place* would be considered relevant to this very textual query but irrelevant for the textual query *places*.

Typically, a user demands an information retrieval system to only return relevant items in the retrieval list. On the other hand, the system should also return all elements from the database which are relevant. Determining the performance of a retrieval system can thus be based on the fraction of relevant items in the retrieval list and the fraction of relevant items returned from the database. While the former measure is known as *precision* the later is called *recall* [12, p. 135]. It is important to always consider both measures when determining the performance of a retrieval system as the precision can typically be traded for recall and vice versa. For example, returning only the item of the database for which the system asserts the highest likelihood of being relevant likely leads to a precision of 100% but yields a small recall if more than one relevant item is in the database. On the other hand, returning all items in the database leads to a recall of 100% but typically yields a number of irrelevant items which causes a drop in precision.

Comparing information retrieval systems based on two values is difficult as only pareto-superiority can be assessed this way. For comparing two methods, a scalar performance measure is much more desirable (cf. e.g. [12, p. 139]). A straight forward approach to get a single value from the precision and recall values is to compute the harmonic mean of two values which is known as *$F$-score* or *$F$-measure* (cf. e.g. [12, p. 144]). However, there is a distinct disadvantage to the $F$-measure: When querying a database, a user would typically like those items to appear first in the retrieval list for which the retrieval system asserts the highest likelihood of being relevant. The $F$-measure, however, is agnostic to the order of

Relevance List:
[1, 0, 1, 0, 0, 1, 1, 0]

$\text{pr}_q(1) = 1$
$\text{pr}_q(2) = 0.5$
$\text{pr}_q(3) = 0.67$
$\text{pr}_q(4) = 0.5$
$\text{pr}_q(5) = 0.4$
$\text{pr}_q(6) = 0.5$
$\text{pr}_q(7) = 0.571$
$\text{pr}_q(8) = 0.5$

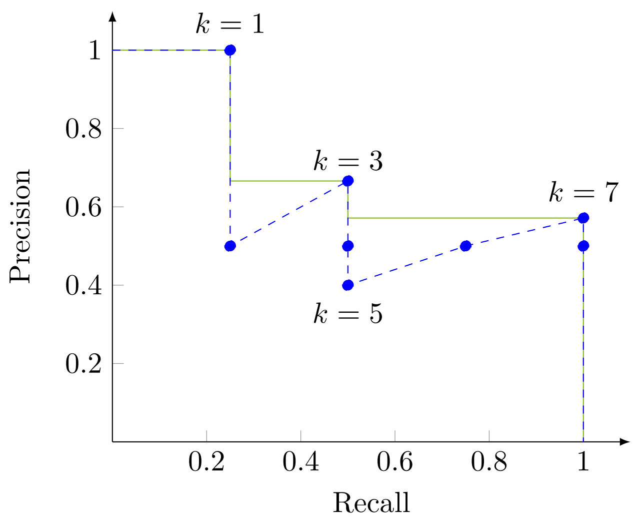


Figure 25: Visualization of the precision at $k$ and average precision: Given a list determining the relevance of the respective item in a corresponding retrieval list, the different precision values $\text{pr}_q$ can be computed when cutting off the list after one of the $k$ possible elements. In order to represent the precision as a function of the recall, an interpolation scheme is applied (green line). The average precision is then the area under this interpolated precision recall curve. The presented example assumes, that the retrieval list corresponding to the relevance list contains all items available in the queried database, i.e., is complete.

the retrieval list and simply treats it as set of retrieved items. A very well known measure for assessing the precision, recall as well as sorting of a retrieval list simultaneously is the Average Precision (AP) (cf. e.g. [117, p. 158]). In order to compute the AP, the precision values at different recall levels are averaged in order to obtain a single value. The problem is, that the recall levels for a given retrieval list are inherently discrete. The precision values can thus only be determined for discrete levels. Determining precision this way, one ends up with a concept known as *precision at k* [12, p. 140]. It is a function $\text{pr}_q(k)$ which determines the precision of the retrieval list for query $q$ if the retrieval list is cut-off after $k$ elements. Fig. 25 visualizes a toy example illustrating the principals of precision at $k$. Here, the relevance list for a generic query $q$ is displayed. This list simply contains a 1 at all positions which contain relevant items in the corresponding retrieval list and 0 for all other positions. The graph on the right side plots the precision against the recall for the different cut-off positions $k$. In the example, it is assumed that the retrieval list is complete, meaning that it contains all items available in the queried database. As can be seen from the plot, transforming the precision at $k$ into a function of the recall is impossible as single recall values often-times have a number of precision values. Thus a common approach is to interpolate the precision at $k$ to the next highest recall level [117, p. 158]. This *interpolated precision* can be computed by

$$\text{ipr}_q(r) = \max_{r' \geq r} \text{pr}_q\left(\text{pos}\left(r'\right)\right) \tag{80}$$

where $r$ is the desired recall level, $r'$ a recall level bigger or equal to $r$ and pos an indicator function determining the index of the retrieval list where the recall $r'$ is obtained. In the example from Fig. 25, the interpolated precision is indicated as a green curve in the plot. The intuition behind using the interpolated precision for the AP is that a user would likely be willing to look at a certain amount of irrelevant items in the retrieval list if it could increase the chance at obtaining a higher precision [117, p. 159]. The interpolated precision has two nice characteristics: First, compared to the precision at $k$, it is a function from a mathematical point of view as any recall level is mapped to a single precision value only. Second, it is also well defined for the recall level 0.

For determining the AP, the interpolated precision doesn't have to be computed explicitly. Instead, one may only use the precision at $k$ values for which the recall level changes. The AP is then simply the average of these selected values:

$$\mathrm{AP}(q) = \frac{\sum_{i=1}^{\mathsf{L}} \mathrm{pr}_q(i) \cdot \mathrm{rel}_q(i)}{n_{\mathrm{rel}}(q)}. \tag{81}$$

In the equation, $\mathsf{L}$ is the length of the retrieval list obtained for a query $q$, $\mathrm{rel}_q(i)$ an indicator function yielding 1 if the $i$-th element of the retrieval list is relevant with respect to $q$ and 0 otherwise, and $n_{\mathrm{rel}}(q)$ the total amount of relevant items in the database for the query. The average precision is equivalent to the area under the interpolated precision-recall curve for a given query (cf. e.g. [117, p. 160]).

The AP can be used as performance measure for a single query. However, a user would typically like a retrieval system to not only perform well for a single query but rather a number of different ones. A straight forward approach is thus to use a number of queries for testing. The resulting AP values are then averaged in order to obtain a scalar performance measure. This way, all queries are treated as equally important. The resulting averaged value is known as mean Average Precision (mAP) and is one of the most commonly used performance measures for retrieval systems (cf. e.g. [117, p. 159]). This is also true for word spotting where the mAP has become the de-facto standard metric for determining the performance of different methods. Hence, it will be used in this thesis as well in order to determine the performance of different Attribute CNN configurations. This not only allows for comparing the configurations against each other but also for comparing the obtained results to others reported in the literature.

## 7.2 SIGNIFICANCE TESTING

When examining different word spotting methods or configurations, one is tempted to simply compare the obtained mAP values in order to determine which method performs better or worse. However, assessing superiority based on this single number bares a number of risks. For example, it may be that a given method performed better by chance on a given set of queries used for evaluation and might perform substantially worse when using slightly different queries. Additionally, a number of word spotting methods make use of training some sort of model on a given dataset, be it in an unsupervised or supervised fashion. The process of training these models almost always involves some sort of stochastic element, e.g., sampling a suitable number of descriptors for creating a visual vocabulary in Bag of Features (BoF)-based approaches (cf. e.g. [159, 164]) or the stochastic gradient descent used for training neural networks (cf. e.g. [91, 206]). It thus may be that superior performance stems from a “lucky” solution found during training rather than a methodological advantage.

Hence, it is desirable to take into account additional evidence besides the pure mAP values when comparing word spotting methods. One possible approach for this is to run *statistical hypothesis tests* (cf. e.g. [34, p. 3-28]). These tests allow for a data-driven assessment whether a hypothesis made about a certain set of data is unlikely enough to be rejected. This hypothesis is referred to as *null hypothesis*. Among others, a commonly tested null hypothesis is the assumption, that the mean of two random variables is equal. This hypothesis can be used for determining whether there exists statistical evidence for one of two word spotting methods being better than the other on a specific dataset: First, the average precisions obtained from the two methods for the given dataset are treated as realizations of two random variables. The respective mean of these two random variables is the "true" mAP of the corresponding method. If the null hypothesis can be rejected, these two mAP values are significantly different from one another, i.e., the larger of the two mAPs is significantly higher than the other. A significantly higher mAP value indicates superiority beyond chance. This can be interpreted as the method with a significantly larger mAP value than another being significantly better than the other method on a given dataset.

An important element of any statistical test is the so-called *p-value*. It represents the probability of observing the data (e.g. the two different sets of APs) under the assumption that the null hypothesis is true. If this value is smaller than a previously defined threshold, the null hypothesis is rejected and the *alternate hypothesis* is accepted, which is simply the opposite of the null hypothesis. The threshold mentioned above is called *significance level* and is a meta-parameter of the test which has to be set by the user. If the null hypothesis is rejected, the result of the test is called *statistically significant* or simply *significant*. It is very important to note that a statistical test can only reject the null hypothesis if the evidence allows so. In particular, it can never serve as statistical evidence for accepting the null hypothesis. In case the obtained $p$-value is bigger than the significance level, the test only *fails to reject* the null hypothesis but does not accept it. This means, that there does not exist enough statistical evidence for making a definitive statement regarding the null hypothesis. In the context of word spotting, the difference between two mAP values may still be significant in case of a failure to reject the null hypothesis that both are equal. However, this can not be determined from the two sets of obtained AP values. A failure to reject happens if the two empirically determined mAPs are too close from a numerical point of view for the test to definitively say, that the underlying "true" mAPs are different. In this case, it can be assumed that the two corresponding methods perform (almost) the same.

There exist a number of statistical tests for determining whether the means of two sets of data are significantly different. Most of these tests, however, make some form of assumption regarding the data used for testing, i.e., the AP values in the case of word spotting. For example, the well known two-sample Student's t-Test assumes that the two random variables each follow a normal distribution with identical variance values (cf. e.g. [78, p. 8]). The same assumption is made for the Wilcoxon-Mann-Whitney-Test (cf. e.g. [78, p. 101]). A notable exception regarding assumptions is the permutation test which is also known as resampling or randomization test [182]. The basic idea behind the permutation test is the following: If the null hypothesis (means of both random variables are equal) is true, the difference between the observed mean values for each variable should not be smaller if the realizations of the two variables are randomly assigned to either variable. Taking all possible permutations of assigning the data to either of the two variables, the true observed difference should thus not be an outlier compared to the difference in means obtained after permutation. Here, the fraction of permutations where the difference of the randomly assigned average precision values is greater than the true observed difference is exactly the $p$-value for the permutation test (cf. e.g. [41, 130, 182]).

For an exact permutation test, all possible permutations of the data at hand have to be evaluated. In practice, however, computing all permutations quickly becomes impossible as the sample sizes grow [130]. A solution to this problem is to approximate the $p$-value of the test by sampling an adequate number of permutations. Let $\hat{p}$ denote the approximated $p$-value. The variance $s_{\hat{p}}^2$ of $\hat{p}$ is

$$s_{\hat{p}}^2 = \frac{p(1-p)}{\mathsf{K}} \tag{82}$$

where $\mathsf{K}$ is the number of sampled permutations and $p$ is the underlying true $p$-value [41, p. 208]. This formula can be rearranged in order to find the number of iterations necessary to obtain a desired standard deviation:

$$\mathsf{K} = \frac{p(1-p)}{s_{\hat{p}}^2}. \tag{83}$$

In the formula above, the true $p$-value is unknown. However, the upper limit of permutations necessary to obtain a desired standard deviation for any $p$-value can be computed by finding the maximum of $p(1-p)$. The maximum value is obtained for $p = 0.5$. Inserting 0.5 into Eq. 83 for $p$ yields the following function:

$$\mathsf{K} = \frac{1}{4s_{\hat{p}}^2}. \tag{84}$$

Using Eq. 84, the user can specify an allowed standard deviation $s_{\hat{p}}^2$ of the $p$-value and obtain the number of permutations necessary to obtain this standard deviation.

Having layed out the general principles, Alg. 1 shows how to assess the significance of difference in performance of two word spotting methods using the permutation test: Given two sets $\mathbf{A}_1$ and $\mathbf{A}_2$ of average precision values for two different methods obtained for the same queries and retrieval database, the difference between the mAP values $t_{\text{obs}}$ is computed first. This is the true observed difference. Then, $\mathbf{A}_1$ and $\mathbf{A}_2$ are joined in order to form a combined set $\mathbf{B}$. Whether an AP value in $\mathbf{B}$ belongs to the first or second method can not be differentiated anymore. The number of permutations $k$ is computed from the desired standard deviation $s_{\hat{p}}$ which has to be defined by the user. Afterwards, $k$ different sets $\mathbf{P}_1$ and $\mathbf{P}_2$ are drawn from $\mathbf{B}$ such that all samples in $\mathbf{B}$ get randomly assigned to one of the two sets. These two sets must be different from all other previously drawn sets, i.e., be a unique permutation for the current test. In order to assure that the current permutation is unique with respect to all previously drawn permutations, a hash value of the current permutation is computed[1]. If this hash value has been observed for a previous permutation, a new permutation is drawn. This is repeated until an unseen permutation was drawn. By only using permutations with a unique hash value, it can be guaranteed that the drawn permutation is also unique with respect to all previous permutations. The algorithm then proceeds to compute the difference of the means $t_{\text{perm}}$ for the current sets $\mathbf{P}_1$ and $\mathbf{P}_2$. If this difference is larger than the true observed difference $t_{\text{obs}}$, a counter variable $n$ is increased. After having evaluated $\mathsf{K}$ unique permutations, the fraction $\frac{n}{\mathsf{K}}$ represents the estimated $p$-value $\hat{p}$. If it is smaller than the defined significance level $\alpha$, the null hypothesis can be rejected, meaning that the difference in mAPs is deemed significant. Otherwise, the test fails to reject the null hypothesis.

1 The implementation used in this thesis uses the built-in `hash` function of the Python programming language as this hash function allows for an efficient computation of hash values. In principle, other hash functions such as the Message-Digest Algorithm 5 (MD5) could be used here as well.

**Algorithm 1** Determining the significance of difference in mAP between two word spotting methods using the permutation test

```
Inputs:
    $\mathbf{A}_1$: set of average precision values obtained for the first method,
    $\mathbf{A}_2$: set of average precision values obtained for the second method,
    $s_{\hat{p}}^2$: desired variance of the estimated $p$-value $\hat{p}$
    $\alpha$: significance level of the test
Initialize:
    $t_{\text{obs}} \leftarrow \left|\bar{\mathbf{A}}_1 - \bar{\mathbf{A}}_2\right|$
    $\mathbf{B} \leftarrow \mathbf{A}_1 \cup \mathbf{A}_2$
    $\mathsf{K} \leftarrow \left(4 s_{\hat{p}}^2\right)^{-1}$
    $n \leftarrow 0$
    $\mathbf{H} \leftarrow \varnothing$
for i = 1 to k do
    while TRUE do
        Draw sets $\mathbf{P}_1$ and $\mathbf{P}_2$ from $\mathbf{B}$ without replacement
        Concatenate the byte representations of $\mathbf{P}_1$ and $\mathbf{P}_2$
        $h \leftarrow$ hash value for the concatenated byte string
        if $h \notin \mathbf{H}$ then
            $\mathbf{H} \leftarrow \mathbf{H} \cup \{h\}$
            BREAK
        end if
    end while
    $t_{\text{perm}} \leftarrow \left|\bar{\mathbf{P}}_1 - \bar{\mathbf{P}}_2\right|$
    if $t_{\text{perm}} > t_{\text{obs}}$ then
        $n \leftarrow n + 1$
    end if
end for
if $\frac{n}{\mathsf{K}} < \alpha$ then
    Difference in mAP is significant for level $\alpha$
else
    Failure to reject
end if
```

## 7.3 EVALUATION OF QBE AND QBS WORD SPOTTING

In this section, the results obtained for the Query-by-Example (QbE) and Query-by-String (QbS) experiments are presented. In order to be able to compare the results obtained from the presented Attribute CNNs with those from other methods from the literature, established word spotting benchmarks are used. The same benchmarks are also used for comparing the different configurations of the proposed CNNs.

First, the datasets used for the evaluation are presented in Sec. 7.3.1. This description also contains how a given dataset is split up into training and test partitions, where the test partition consists of a set of queries and a retrieval set which is used as database. The ensuing Sec. 7.3.2 then covers the protocols for the different benchmarks. Having defined the word spotting specifics, Sec. 7.3.3 presents the meta-parameters used for training the

Attribute CNNs and justifies their choice. Finally, the obtained results are presented in Sec. 7.3.4 and discussed in Sec. 7.3.5.

### 7.3.1 *Description of the Benchmarks*

The evaluation of the presented CNNs for QbE and QbS is done on six publicly available benchmarks. These datasets used in these benchmarks are made up of document images showing both historical and contemporary handwriting. In addition, a benchmark featuring Arabic handwritten text is included in order to determine the robustness of the presented method to a change in script. In the following, the different datasets are explained in detail.

#### *George Washington*

The George Washington Database (GW) has become the data source for one of the standard benchmarks for word spotting. It is also known as *George Washington 20 (GW20)* as it encompasses 20 pages of correspondences between George Washington and his associates dating from 1755. The dataset is an excerpt of a larger collection available at the library of congress of the United States[2]. As the documents in the GW are obtained from the letter book 2, which is not an original, but a later re-copied volume, it can be assumed that the dataset has been produced by a single writer only.

There actually exist two versions of this dataset which have been used to evaluate word spotting methods. The first version contains binarized word images which have been slant corrected[3]. The second version[4] contains plain gray-level document images and is by far the more commonly one used for evaluating word spotting methods (e.g. in [159, 164–166, 177]). For the following experiments, the plain gray-level document images will be used as well. Although not challenging from the number of writers, the GW exhibits a number of aging artifacts such as fading ink and bleed-through.

The annotation for the GW contains word bounding boxes for 4860 word images with 1124 different transcriptions. Originally, the GW was used for unsupervised word spotting methods which is why there exists no official partition for supervised methods into training, query and retrieval set. Typically, when using this dataset for supervised approaches, a fourfold cross validation is performed (cf. e.g. [9]). For segmentation-free experiments, the dataset is split up into cross validation splits of five consecutive pages [157]. However, for segmentation-based evaluations, as are done in this thesis, a randomized fourfold split of all words is typically used. Here, the splits by Almazán *et al.* [9] serve as de-facto standard partitioning for the GW. In order to be able to compare the results obtained for the Attribute CNNs to those reported by Almazán *et al.* for AttributeSVMs, the same cross validation splits are used in this thesis[5].

#### *IAM-DB*

Although originally designed for handwriting recognition, the IAM Handwriting Database (IAM-DB) [119] has recently been used as data for word spotting benchmarks as well. The

2 https://memory.loc.gov/ammem/gwhtml/
3 http://www.fki.inf.unibe.ch/databases/iam-historical-document-database/washington-database
4 http://ciir.cs.umass.edu/downloads/old/data_sets.html
5 cross validation partitions available at https://github.com/almazan/watts/tree/master/data

database is made up of document images showing English handwritten text of contemporary style. In its latest version 3.0, it consists of 1539 scanned pages containing text from 657 different writers[6]. The annotation contains transcriptions for 13 353 text lines as well as bounding boxes for 115 320 word images.

The document images in the IAM-DB are gray-level images and the quality is very good. The challenge lies in the fact that the official partitioning into training and test is writer independent meaning that a single writer did only contribute to either the training or the test set. This partitioning has been adopted for word spotting benchmarks in the literature as well (cf. e.g. [9, 91, 206]). In order to allow for a direct comparison of the Attribute CNNs to these approaches, the standard partitioning is used in this thesis as well.

A common benchmark for segmentation-based, supervised word spotting on the IAM-DB is defined by Almazán *et al.* [9]. The first step in this benchmark is to remove all text lines from the dataset where the annotation states that the transcription may be dubious or contains errors. The training images from the handwriting recognition benchmark are then also used as training partition. As retrieval set, all images in the test partition of the recognition benchmark are used.

### *Esposalles*

The *Esposalles Database* [154] is an excerpt of the larger *Llibres d'Esposalles* collection of marriage license books at the archives of the Cathedral of Barcelona dating from 1451 to 1905. For the following experiments, the first version of the Esposalles databases will be used as presented in [154]. It consists of 173 pages of historical documents. The major difficulties of the dataset lie in several forms of degradation of the document images such as uneven illumination, smearing or bleed-through as well as high variability in the text stemming from multiple writers.

The accompanying annotation gives bounding boxes and transcriptions for 45 100 word images. The official partitioning for offline handwriting recognition uses 32 052 of these word images as training set and the remaining 13 048 as test set. As there exists no dedicated word spotting partitioning, the approach chosen for the Esposalles is follows: the official training partition is used for training also while the test set is used for obtaining the query and retrieval set.

### *IFN/ENIT*

The IFN/ENIT Database (IFN/ENIT) [137] contrasts all other datasets used as it features Arabic script. It consists of word images of Tunisian town or village names. The total amount of word images is 26 459 which were contributed by 411 different writers. There exists an official partitioning of the database into four subsets A, B, C and D. For handwriting recognition, the custom benchmark using the IFN/ENIT uses sets A,B and C for training and D for testing. As, again, there exists no dedicated word spotting partitioning, the same approach is chosen for IFN/ENIT as was done for IAM-DB and Esposalles.

While for all other datasets a Latin alphabet can be used in order to extract attribute representations, the IFN/ENIT requires an Arabic alphabet. The problem here is that this alphabet can easily become very large considering all characters with diacritics and

6 Version 3.0 of the IAM-DB is available at `http://www.fki.inf.unibe.ch/databases/iam-handwriting-database`

Table 1: Number of training images for the respective partitions of the Botany and Konzilsprotokolle datasets after the removal of word images wider than 2000 pixels.

| | Botany | Konzilsprotokolle |
|---|---|---|
| Train I | 1683 | 1849 |
| Train II | 5289 | 7816 |
| Train III | 21 964 | 16 918 |

ligatures. In order to keep the size of the attribute representations at a manageable number, the alphabet is reduced to a smaller set of characters in the following way[7]: First all character shapes are mapped to their representative Arabic characters. Characters with optional Shadda diacritic are replaced with characters without the Shadda diacritic. Special two-character-shape ligature models are mapped to two-character ligature models without the shape contexts. This mapping produces an alphabet of size 50 for this dataset[8]. Please note that Arabic script is written from right to left opposed to Latin script. In order to account for this, the word images from the IFN/ENIT are mirrored along the vertical axis for the following experiments.

*Botany and Konzilsprotokolle*

The two datasets Botany in Britsh India (Botany) and Alvermann Konzilsprotokolle (Konzilsprotokolle) where introduced and used in the Handwritten Keyword Spotting Competition held during the 2016 International Conference on Frontiers in Handwriting Recognition[9]. While the former covers botanical topics such as gardens, botanical collections and plants, the latter is a collection of protocols from the central administration at the university library of Greifswald, Germany, dating from 1794 to 1797. While Botany is written in Latin script, the script used in the Konzilsprotokolle database is Kurrent.

Different from the other datasets presented before, Botany and Konzilsprotokolle are dedicated word spotting datasets and come with a partitioning into training set, query images for QbE, query strings for QbS as well as a retrieval set. As part of the competition was to evaluate how well the participating systems could deal with small to large amounts of training images, each dataset comes with three increasingly larger training sets. Tab. 1 lists the sizes for the three different sets. Please note that both datasets contain word images which are wider than 2000 pixels. When using images of this size, a pre-experimental evaluation showed that it was impossible to fit the neural networks into the memory of the Graphical Processing Unit (GPU) used during training. Hence, images which are wider than 2000 pixels are removed from the training partitions for both datasets. This procedure still allows for comparing the obtained results to those reported in the literature as the number of available training images is decreased. The images in the query or retrieval sets are not altered.

The respective retrieval set sizes are 3230 for Botany and 3533 for Konzilsprotokolle. For QbS, there exist 101 query strings for each dataset. For QbE, the query word images in Botany amount to 150 while for Konzilsprotokolle there exist 200.

7 The idea and implementation of this specific reduction are courtesy of Irfan Ahmad.
8 The approach bares some similarities to the Arabic sub-character modeling proposed in [2] which was later extended on in [3].
9 https://www.prhlt.upv.es/contests/icfhr2016-kws/data.html

The major challenges for both datasets are similar to ones present in the other historical datasets, namely aging effects. In addition, Botany features a considerable amount of size variability in word images of the same class.

### 7.3.2 *Evaluation Protocols*

As explained before, the presented datasets can be discriminated with respect to whether they come with dedicated sets of training, query and retrieval images or not. While Botany and Konzilsprotokolle possess official partitions for these three sets, they need to be obtained for the other datasets first. Almazán *et al.* [9] present a method how the three sets can be obtained from a standard training-test partition found in datasets used for handwriting recognition and how word spotting can be performed with these three derived sets. The approach by Almazán *et al.* is used for GW, IAM-DB, Esposalles and IFN/ENIT and will be dubbed *Almazán Protocol*. For Botany and Konzilsprotokolle, the protocol from the competition will be used which will be referred to as *Competition Protocol*. For both protocols, the mAP is used for comparing the performance of different methods. In order to determine the significance of difference in performance between two methods on a given dataset, the permutation test is used as described in Sec. 7.2. For all experiments in this thesis a small desired standard deviation $s_{\hat{p}}$ of 0.001 is chosen. Choosing this standard deviation, the probability for the true $p$-value being in the interval $\hat{p} \pm 0.003$ is more than 99%. The desired standard deviation is obtained after a maximum of 250 000 randomly chosen permutations (see Eq. 84 for determining the upper bound of permutations to be evaluated). It needs to be pointed out here, that only the results obtained from the different Attribute CNNs will be compared by means of the permutation test. Comparing the obtained results to those reported in the literature would require the other results to report the individual AP values as well, which is not the case for the methods the Attribute CNNs will be compared to.

In the rest of this section, the Almazán and Competition protocols will be explained in detail.

#### *Almazán Protocol*

Adapting the word spotting protocol proposed by Almazán *et al.* [9] in order to obtain mAP values for different Attribute CNNs and datasets is done as follows in the ensuing experiments: For each dataset, the annotation is used in order to create a segmentation for each word image. The alphabet to be used for creating the desired attribute representation is then obtained by collecting each unique character from the annotations for the training set. Here, the lower case version for each character is used only. Each transcribed word is then transformed into an attribute or attribute-like representation using this alphabet. The word images in the training partition and their corresponding representations are then used in order to train a given Attribute CNN.

At query time, the segmented word images in the test set are used for retrieval as follows:. For QbE, each word image in the test set is used once as query to rank all the other test word images, effectively using them as retrieval set. For ranking, an attribute representation is predicted for the query and each word image in the retrieval set from the Attribute CNN to be evaluated. The word images in the retrieval set are then ranked according to the cosine dissimilarity from their predicted attribute representation to the query representation. A requirement for a word image to be used as query is that its corresponding word class appears more than once in the retrieval set as otherwise the retrieval list would not contain any relevant results. However, if a word class only appears once, the corresponding word image is kept in the retrieval set as distractor for all other queries.

For QbS, all unique word classes in the retrieval set are used as queries. The attribute or attribute-like embedding can be computed directly from the word class string. For each query, the entire test set is used as retrieval set. Similar to QbE, the Attribute CNN to be evaluated is used to predict attribute representations for all word images in the retrieval set. As in QbE, the obtained representations are then ranked according to their Cosine distance to the query representation.

In both QbE and QbS the entire retrieval set is returned in the retrieval list. Hence, the recall for each query is always 100%. An item in the retrieval list is considered relevant if its annotated word class matches the query word class without considering upper or lower cases as different. For example, a word image showing *Captain* is relevant for the query string *captain* but not for *Captains*.

The Almazán protocol is used for the datasets GW, IAM-DB, Esposalles and IFN/ENIT. For IAM-DB, however, stop words are excluded as queries for both QbE and QbS. Again, they are kept as “distractions” among the test words though.

*Competition Protocol*

The segmentation-based protocol used for the 2016 Keyword Spotting Competition [141] is very similar to the Almazán protocol. The training of the respective Attribute CNNs is performed just as before. However, there are dedicated query sets for both QbE and QbS for which it can be assumed that at least one relevant item is present in the respective retrieval set. Before query time, the attribute representations are predicted for the word images in the retrieval set under both query paradigms. For QbE, the query representations are then also predicted from the query word images. The retrieval list is then generated as was done for the Almazán protocol. Relevance is also assessed in the same way.

### 7.3.3 *Training Configurations*

All Attribute CNNs used in the experiments are trained in an end-to-end fashion given word images as input and their corresponding attribute representation as labels. The respective CNNs are always trained from a random initialization without the help of additional pretraining. No word image is preprocessed before being forwarded through either of the CNNs except for scaling the pixels to floating point values in the range of $[0, 1]$ with 0 representing white pixels and 1 representing black pixels. As the word images in the respective datasets are all made up of black strokes, this lets those parts of the image be closer to 1 in pixel values while the background is closer to 0. The rational behind this is that the convolution layers employed in a CNN effectively multiply each pixel in the input with a corresponding weight value from the filter kernel. Using the non-zero values for ink, the filters will thus be enabled to be active when “seeing” portions of the word image containing ink, i.e., the supposedly relevant portions.

The initial experiment is conducted using the three presented attribute representations Pyramidal Histogram of Characters (PHOC), Spatial Pyramid of Characters (SPOC) and Discrete Cosine Transform of Words (DCToW) in addition to their decorrelated versions using Positive Pairwise Mutual Information (PPMI) and Multi Dimensional Scaling (MDS) (cf. Sec. 5.2). For training the Attribute CNNs, the Binary Cross Entropy Loss (BCEL) (Eq. 70) is used for PHOC representations while the Cosine Loss (Eq. 74) is used for all others. In addition, the Cosine Loss is evaluated in combination with PHOC representations as well.

Due to the massive amount of free parameters in the fully connected parts of the Attribute CNNs, the neural networks are prone to overfitting. Hence, a number of regularization techniques is applied in order to mitigate the overfitting problem. The regularization measures used are described in the following subsection. Afterwards, the section closes by explaining the training of the neural networks using the Stochastic Gradient Descent (SGD) algorithm.

### *Regularization*

The regularization techniques used for the presented Attribute CNNs are well-known ones which have become standard approaches for deep learning architectures in general. First, dropout is applied to all but the last fully connected layers in the Multilayer Perceptron (MLP) parts of the neural networks. In dropout, the output of a neuron is randomly set to 0 with a certain probability. This prevents a neural network from learning certain paths for a given input image "by heart" as neurons can no longer rely on a neuron in a previous layer to always be active for a given image. Another way to think of dropout is that a the network trained with dropout actually represents an ensemble of smaller networks which all share some of their weights [185]. The size of this ensemble is exponential in the number of neurons used in the layer applying dropout. Deep learning architectures typically use dropout with a probability of 50% meaning that for each forward pass half the neurons in a layer are dropped on average (cf. e.g. [92, 172, 179]).

In addition to dropout, the training images are augmented with additional images which are created in an unsupervised way. The goal in augmentation is to present the neural network with variability of the data which may not be covered by the training set but can be expected for new images. The transformations used for creating the new images have to be *label preserving*, meaning that the depicted content may not change in an augmented image given the original label. For example, in natural images this is typically the case when flipping an image around the vertical axis. However for word images, this approach is not valid for augmentation as the mirrored image would almost always not produce a valid word image. In order to allow for augmenting word images, a new approach is presented in this thesis: The unwanted variability in new word images can be expected to fall into the categories usually addressed with preprocessing in other document image analysis tasks. Most prominently, these are variabilities in size, slant, shear, translation and rotation. These variabilities can be accounted for in preprocessing by presenting the neural network with a number of affine transformations of a given word image. In order to not have to define a set of parameters for each of the stated transformations individually, a homography projection can be used. Fig. 26 illustrates the proposed augmentation approach. The process of creating an augmented image using a homography is done as follows: Three points at fixed relative positions in the middle of an image are taken and each coordinate is multiplied with a random number uniformly sampled from the interval $[0.9; 1.1]$. The boundaries of the interval represent the meta-parameters for the presented augmentation. Then, the homography is computed for which the second set of points is obtained from the first. This homography is the transformation used for generating a single augmented image from an original word image. Of course, a number of different augmented images

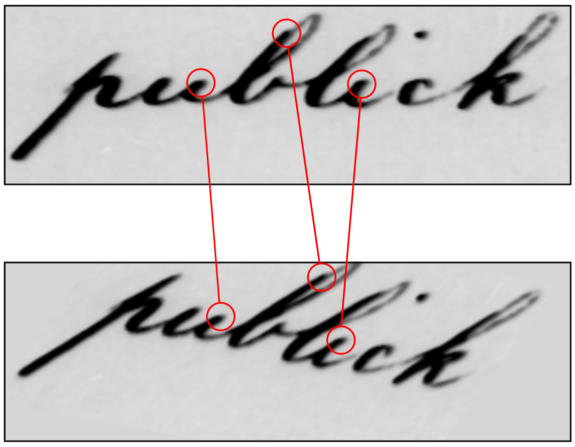


Figure 26: Visualization of how to extract a synthetic (bottom) from an original image (top) for dataset augmentation: A homography is computed which allows for transforming three points into the same three points offset by a random amount. This homography can then be used in order to create a synthetic image in an unsupervised fashion while preserving the original label of the word image.

can be obtained from a single "original" one by sampling the corresponding number of random offsets for the three points.

*Optimization of the Weights in the CNNs*

Traditionally, the optimization strategy of choice in deep learning has been SGD with momentum (cf. Sec. 3.2). The drawback with this approach is that each weight uses the same learning rate during the update step which could possibly hamper training convergence. Recent improvements for classic SGD have thus incorporated additional information into the training process for adapting learning rates individually. For example, AdaGrad [39] assigns low learning rates to frequently occurring features while giving high learning rates to those occurring only rarely. On the other hand, RMSprop [199] normalizes the gradient length for a given weight by a moving average over recent gradient lengths. The optimization strategy Adaptive Moment Estimation (Adam) [83] combines the advantages of AdaGrad and RMSprop. It works by computing a sliding average for the mean and variance of the gradient for each weight. The weights are then updated by applying the mean gradient normalized with its mean standard deviation. The Adam approach has increasingly been used lately as optimization strategy for deep neural networks, e.g., in [49, 76, 81, 218]. In the following experiments, both standard SGD and Adam are evaluated as optimization strategies for Attribute CNNs.

All Attribute CNNs are trained using a mini-batch size of 10. The initial learning rate values are determined by taking the largest value for which training started to converge. For networks being trained with standard SGD this value is $10^{-4}$ when using the BCEL and $10^{-2}$ when using the Cosine Loss. For Adam based optimization the maximum initial learning rate to achieve convergence was found to be $10^{-4}$ which matches the default value proposed by Kingma and Ba [83]. In order to stabilize the gradients for standard SGD, a momentum of 0.9 is used. For Adam, the recommended momentum hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are chosen [83]. In addition, all networks are trained with a weight decay of $5 \cdot 10^{-5}$ as is standard for VGG-style and Residual Network (ResNet) architectures [67, 179]. Training is run for a maximum of 80 000 iterations with the learning rate being divided by 10 after 70 000 iterations. The step size of 70 000 was determined by monitoring the training loss for plateaus. After a total of 80 000 iterations the loss could not be improved upon anymore by lowering the learning rate which is why training was stopped at this point. Only for the experiments using the IAM-DB a maximum number of 240 000 iterations was used as the training loss could still be improved by training beyond 80 000 iterations. On this dataset, training is carried out for 100 000 iterations before lowering the learning rate. Please note that a training iteration here refers to calculating the gradient for a given mini-batch and updating the weights of the network accordingly.

As the neural networks in this thesis are trained with gradient descent, the final solution is inherently dependent on the initial weights of the network. Hence, a crucial step in training is the weight initialization strategy. Throughout the literature, various strategies have been used. The influential AlexNet architecture [92], for example, is initialized by drawing weights from a normal distribution with zero mean and a standard deviation of 0.01. However, choosing initial weights this way hampers training for increasingly deep architectures (cf. e.g. [66, 179]). According to Glorot and Bengio [55], the difficulty in training is caused by a large variance in gradients in the beginning of the training. They propose an initialization scheme based on a uniform distribution which minimizes the variance in the gradients. However, their initialization scheme is based on the assumption that the respective network only makes use of linear activation functions. He *et al.* [66] adapt the work by Glorot and Bengio [55] in order to account for networks using Rectified Linear Unit (ReLU) activation functions. Here, the weights for initializing a layer $s$ are obtained by drawing from a normal distribution with zero-mean and variance $\frac{2}{n_s}$ where $n_s$ is the number of parameters in $s$. For example, if a convolution layer with a kernel size of $3 \times 3$ is presented with a feature map of 512 channels, $n_s$ computes to $3^2 \cdot 512 = 4608$. This theoretically sound initialization strategy allows He *et al.* to train a 30-layer network from scratch. In the experiments presented in this thesis, all weights in the convolutional and fully connected layers of the Attribute CNNs are initialized according to the strategy presented by He *et al.* [66]. As is common for this strategy, the biases in the layers carrying weights are initialized to zero. All CNNs are always trained from scratch and do neither depend on nor require pretrained weights.

All the meta-parameters regarding training were chosen based on pre-experiments on the GW. Only for IAM-DB the number of training iterations was altered as the training loss could still be decreased with the added amount of iterations. Training is carried out on a single Nvidia Pascal P100 using a customized version of the Caffe library [74].

### 7.3.4 *Results*

In total, there exist 48 possible configurations of Attribute CNN architecture, loss function, word string embedding and optimization strategy. Evaluating all these combinations on all QbE and QbS benchmarks would require 288 different experiments when assuming that QbE and QbS can be evaluated in a single experiment. In order to keep the amount of experiments tractable, a subset of suitable configurations is defined first. The resulting configurations are then evaluated on all datasets. For this, the TPP-PHOCNet is trained with all possible configurations of attribute embedding, loss function and optimization strategy on GW and IAM-DB. The choice of these two datasets is based on the following considerations: The GW is a rather small dataset which allows for assessing whether certain configurations are more suitable when faced with fewer training data. While the IAM-DB

Table 2: Comparison of results using different configurations for the TPP-PHOCNet on the GW and IAM-DB datasets in mAP [%].

| Loss | Emb. | Opt. | GW | | IAM-DB | |
|---|---|---|---|---|---|---|
| | | | QbE | QbS | QbE | QbS |
| BCEL | PHOC | Adam | 97.53 | *95.39* | **85.33** | 92.10 |
| Cosine | PHOC | Adam | *96.44* | *87.56* | 66.70 | 85.28 |
| Cosine | SPOC | Adam | 97.17 | *89.68* | 75.43 | 90.53 |
| Cosine | DCToW | Adam | *96.63* | *90.47* | 73.17 | 90.05 |
| Cosine | PPMI-PHOC | Adam | *96.31* | *91.23* | 67.24 | 84.02 |
| Cosine | MDS-PHOC | Adam | *96.92* | *91.34* | 74.93 | 90.25 |
| Cosine | MDS-SPOC | Adam | *96.94* | *91.60* | 77.85 | 91.91 |
| Cosine | MDS-DCToW | Adam | *96.57* | *89.66* | 69.99 | 83.59 |
| BCEL | PHOC | SGD | 97.27 | *96.67* | 79.85 | 90.68 |
| Cosine | PHOC | SGD | 97.47 | 97.20 | 82.62 | 92.20 |
| Cosine | SPOC | SGD | 97.64 | 97.68 | 83.26 | 93.22 |
| Cosine | DCToW | SGD | 97.48 | 97.56 | 83.80 | **93.48** |
| Cosine | PPMI-PHOC | SGD | 97.48 | 97.37 | 76.48 | 88.58 |
| Cosine | MDS-PHOC | SGD | 97.70 | 97.38 | 77.48 | 90.44 |
| Cosine | MDS-SPOC | SGD | **97.72** | 97.63 | 80.33 | 92.06 |
| Cosine | MDS-DCToW | SGD | 97.55 | **98.06** | 65.18 | 78.34 |

has the largest amount of training images of all datasets, the variability in the word images is probably the largest among all datasets. This makes it a great challenge for the CNNs used with respect to learning the attribute representations.

Tab. 2 compares the results obtained for the different string embeddings, loss functions and optimization strategies for the TPP-PHOCNet. In the table, the classic stochastic gradient descent optimization is denoted as *SGD* and the Adam optimization [83] as *Adam* although technically Adam is a form of stochastic gradient descent as well. The best results are printed in bold. Results printed in italics are significantly worse than the respective best result. For all other results, a significant difference to the respective best result could not be determined. The significance of difference is determined by running a Monte Carlo permutation test with a significance level of 0.01 and the standard deviation of the $p$-value set to 0.001 (cf. Sec. 7.2).

As can be seen in Tab. 2, no combination of word string embedding, loss function and optimization always produces better results than the others. As a clear winner in terms of configuration can not be determined, two configurations are chosen for evaluating the three different Attribute CNN architectures on the six benchmarks, one for training with the BCEL and one for the Cosine Loss (see Sec. 7.3.5 for further discussion): The first is the combination of BCEL, PHOC embedding and Adam optimization (BPA) while the second is Cosine Loss, SPOC embedding and standard SGD optimization (CSS).

Tab. 3 presents the results obtained for the other CNN architectures using these two configurations for the benchmarks with Almazán protocol. In addition, the table also lists

Table 3: Comparison of the results obtained with the Attribute CNNs on the benchmarks using the Almazán Protocol. All values are mAP percentages.

| Method | GW | | IAM | | Esposalles | | IFN/ENIT | |
|---|---|---|---|---|---|---|---|---|
| | QbE | QbS | QbE | QbS | QbE | QbS | QbE | QbS |
| PHOCNet (BPA) | 97.25 | *95.13* | *84.61* | *91.71* | 97.30 | 93.62 | 96.57 | 94.81 |
| PHOCNet (CSS) | 97.42 | 97.55 | *82.46* | 92.70 | 97.12 | 94.01 | *93.21* | 94.00 |
| TPP-PHOCNet (BPA) | 97.53 | *95.39* | *85.34* | *92.10* | **97.31** | 93.72 | 96.50 | 93.25 |
| TPP-PHOCNet (CSS) | 97.64 | 97.68 | *83.27* | 93.22 | 97.17 | **94.29** | *92.75* | *93.25* |
| PHOCResNet (BPA) | *96.95* | *94.56* | **86.82** | 93.34 | 97.15 | 93.68 | **96.93** | **95.29** |
| PHOCResNet (CSS) | **97.75** | **98.01** | 85.51 | **94.07** | 97.10 | 93.92 | *92.64* | 93.48 |
| Deep Feat. Emb. [91] | 94.41 | 92.84 | 84.24 | 91.58 | – | – | – | – |
| Attribute SVM [9] | 93.04 | 91.29 | 55.73 | 73.72 | – | – | – | – |
| Finetuned CNN [177] | – | – | 46.53 | – | – | – | – | – |
| LSA Embedding [6] | – | 56.54 | – | – | – | – | – | – |
| Triplet-CNN* [206] | 98.00 | 93.69 | 81.58 | 89.49 | – | – | – | – |
| BLSTM* [47] | – | 84.00 | – | 78.00 | – | – | – | – |
| SC-HMM* [149] | 53.10 | – | – | – | – | – | 41.60 | – |

the results reported in the literature in order to compare them to those of the Attribute CNNs. In this table, significant differences in performance are displayed as was done for Tab. 2. Please note that the results obtained from the literature do not allow for assessing the significance of the obtained differences: In order to conduct a statistical test, the individual average precision values of each method to be compared must be known. As this is not the case, the significance of differences can not be determined for the results obtained from the literature for other approaches. Approaches marked with an asterisk do not share the exact same evaluation protocol and can thus not be compared directly to the results obtained for the Attribute CNNs. In particular, Rodríguez-Serrano and Perronnin [149] use different splits for training and test for their experiments on IFN/ENIT, effectively reducing the number of word images in the retrieval set. The Bidirectional Long Short-Term Memory Network (BLSTM) approach by Frinken *et al.* [47] follows a line spotting protocol instead of the segmentation-based Almazán protocol. Finally, Wilkinson and Brun [206] make use of the CVL-Database [84] in order to pretrain their neural networks thus effectively extending their annotated training dataset. While the results are not directly comparable, they nevertheless give a general idea of the performance of the proposed method in comparison to others. Corresponding to Tab. 3, Fig. 27 shows the mAP values obtained after increasing amounts of training iterations for the QbE experiments.

Tab. 4 and Tab. 5 display the mAP results obtained for the benchmarks using the competition protocol. The results used for comparison were all obtained from the official competition report [141]. Significant differences in performance are displayed as was done in the previous two tables. Fig. 28 visualizes the mAP values obtained after different amounts of training iterations for Botany and Konzilsprotokolle during the QbE experiments.

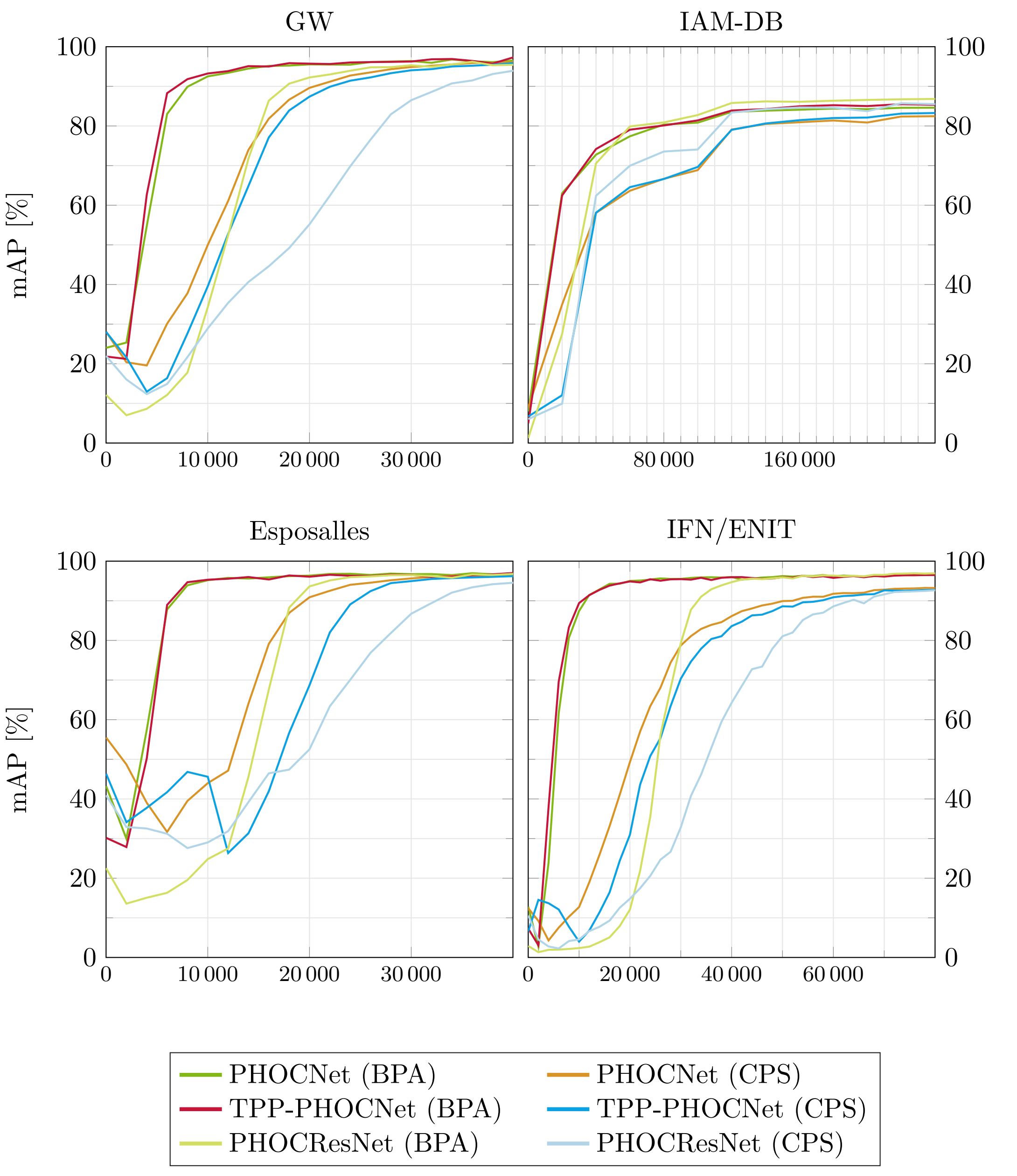


Figure 27: The figure displays the mAP over the different training iterations for the four QbE experiments using the two different Attribute CNNs.

Table 4: Results for the experiments run on the Botany dataset in mAP [%]. All results for other methods were obtained from the official report of the 2016 Handwritten Keyword Spotting Competition [141].

| Method | Train I | | Train II | | Train III | |
|---|---|---|---|---|---|---|
| | QbE | QbS | QbE | QbS | QbE | QbS |
| PHOCNet (BPA) | 41.04 | *31.94* | 77.03 | *76.73* | 92.75 | *95.91* |
| PHOCNet (CSS) | *33.29* | *32.97* | *71.74* | *80.83* | *80.11* | *90.51* |
| TPP-PHOCNet (BPA) | *39.86* | *36.47* | 76.55 | 84.23 | 92.89 | 96.61 |
| TPP-PHOCNet (CSS) | *38.80* | 38.19 | 73.35 | 84.42 | *78.39* | *87.78* |
| PHOCResNet (BPA) | 41.70 | *34.10* | **80.07** | 86.43 | **95.92** | **98.53** |
| PHOCResNet (CSS) | 47.68 | 49.32 | 78.44 | **88.48** | *77.47* | *87.61* |
| AttributeSVM | **75.77** | **65.69** | – | 65.69 | – | – |
| HOG/LBP | 50.64 | – | – | – | – | – |
| Triplet-CNN | 54.95 | 3.40 | – | – | – | – |

Table 5: Results for the experiments run on the Konzilsprotokolle dataset in mAP [%]. All results for other methods were obtained from the official report of the 2016 Handwritten Keyword Spotting Competition [141].

| Method | Train I | | Train II | | Train III | |
|---|---|---|---|---|---|---|
| | QbE | QbS | QbE | QbS | QbE | QbS |
| PHOCNet (BPA) | *84.00* | *76.56* | *95.16* | 94.11 | 96.62 | 96.79 |
| PHOCNet (CSS) | *85.60* | *79.66* | *94.85* | 94.74 | *95.24* | 94.97 |
| TPP-PHOCNet (BPA) | *84.40* | *77.31* | 96.46 | 95.77 | 97.58 | **97.69** |
| TPP-PHOCNet (CSS) | *88.47* | 84.89 | *95.98* | 95.23 | *95.51* | 94.44 |
| PHOCResNet (BPA) | *90.44* | 84.32 | **98.15** | 97.28 | **98.10** | **97.69** |
| PHOCResNet (CPS) | **94.80** | **90.74** | 96.96 | **97.31** | *96.21* | 96.97 |
| AttributeSVM | 77.91 | 55.27 | – | 82.91 | – | – |
| HOG/LBP | 71.11 | – | – | – | – | – |
| Triplet-CNN | 82.15 | 12.55 | – | – | – | – |

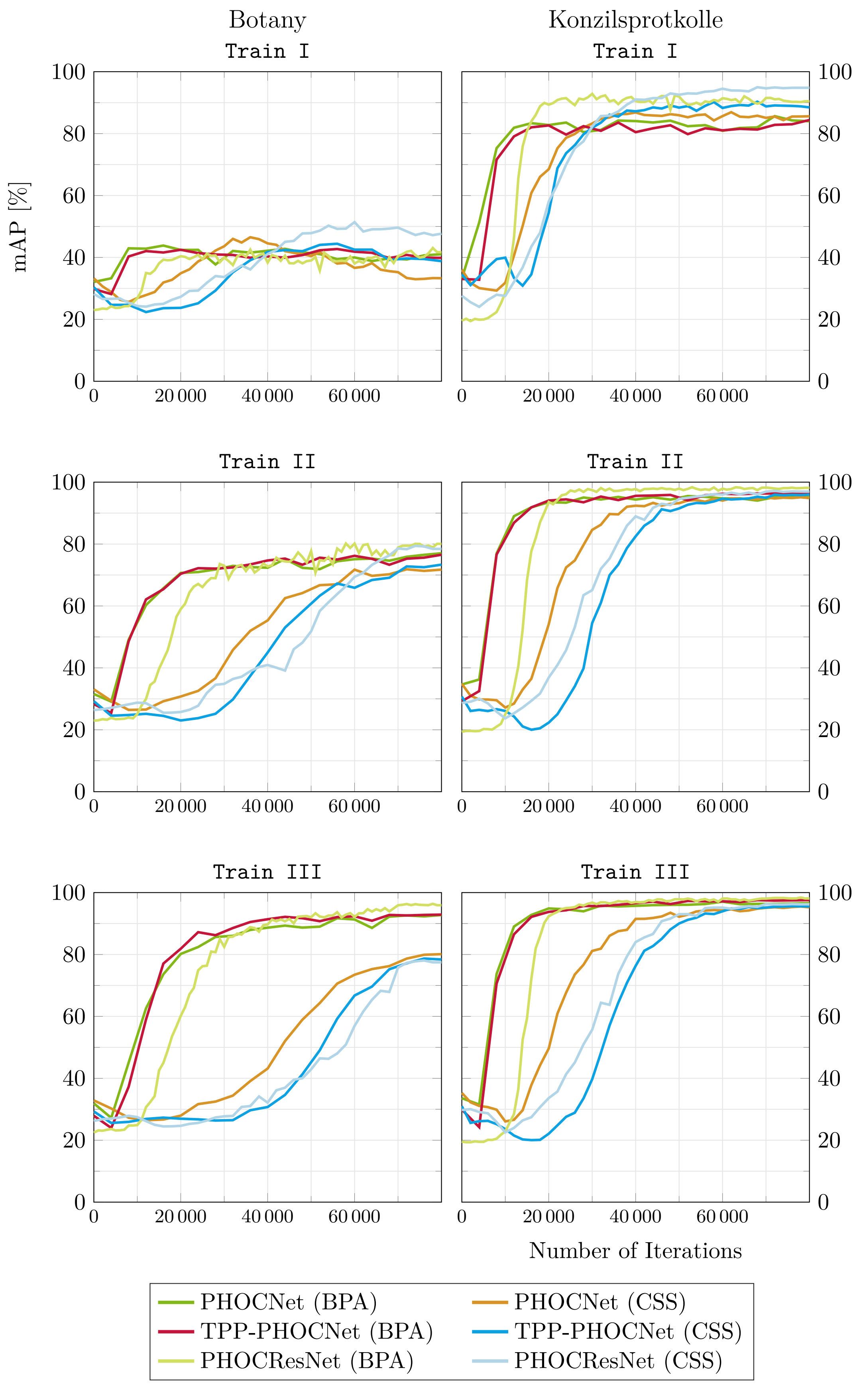


Figure 28: Comparison of the evolution of the QbE experiments for the Botany and Konzilsprotokolle datasets.

### 7.3.5 *Discussion*

#### *Examining the Results Obtained for the Different Configurations*

Tab. 2 lists the results for the different combinations of loss function, attribute or attribute-like embedding and optimization strategy using the TPP-PHOCNet. The first observation is that there does not exist a clear cut winning combination. For most combinations, the permutation test was unable to find a significant difference. This indicates that performances in these cases is almost identical. The PHOC embedding might be considered the only exception here as it always achieves state-of-the-art results. In addition, this attribute representation consistently allows for the fastest training of all configurations when combined with BCEL and Adam optimization (Fig. 27 and Fig. 28).

One fact that can be observed is that the networks trained with the Cosine Loss generally do not perform well when paired with the Adam optimization for either representation. While this characteristic does show less severe for the GW dataset, using Cosine Loss and Adam for the more challenging IAM-DB leads to a substantial drop in performance. Optimizing with SGD is more suitable for this combination as the results obtained for PHOC, SPOC and DCToW embedding are very similar.

Using a decorrelated embedding does not provide an advantage for any of the tested configurations and datasets. For the experiments on the IAM-DB, decorrelating the attribute representations even leads to worse results for QbE. The results suggest that the dependencies of the individual variables in the PHOC vectors do not pose a problem when training the TPP-PHOCNet with BCEL. This is a notable result as one assumes independence between the individual components of a binary vector when training with the BCEL (cf. Sec. 5.3.2).

For comparing the different Attribute CNN architectures, one configuration using the BCEL and one using the Cosine Loss is selected. The idea behind this is to examine whether the correlation of attributes does not play a role for training the TPP-PHOCNet only or if the same results can be obtained for the other architecture as well. For the configuration using the BCEL, Adam is chosen as optimization strategy and the PHOC as attribute representation. This combination achieved the best results on the IAM-DB for all examined configurations.

#### *Comparing the Attribute CNN Architectures*

Comparing the results obtained from the PHOCNet with the TPP-PHOCNet allows for assessing the suitability of the Temporal Pyramid Pooling (TPP) layer compared to the Spatial Pyramid Pooling (SPP) layer as both architecture only differ in this layer. As can be seen in Tab. 4 and Tab. 5, the TPP layer achieves better results for the QbS experiments if the number of training images is rather small as is the case for the first two training partitions of Botany and Konzilsprotokolle. This is probably due to the reduced number of parameters: The fully connected layer following the TPP layer has 66.7% less parameters in the TPP-PHOCNet than the layer following the SPP layer in the PHOCNet (cf. Sec. 5.4.2). This reduction leads to less trainable parameters and thus acts as an additional regularization measure. On the other hand, the representation obtained from the TPP layer is still discriminative enough to be able to achieve state-of-the-art results if there exists a large amount of training images. The results obtained with the PHOCNet and TPP-PHOCNet are almost identical in these cases. An interesting aspect is that the TPP layer consistently produces more robust representations which generalize better than

Table 6: Comparison of the results obtained when using the output of the SPP and TPP layers as word image descriptors for QbE word spotting. The displayed numbers are mAP values [%].

| Attribute CNN | GW | IAM-DB | Esposalles | IFN/ENIT |
|---|---|---|---|---|
| PHOCNet (SPP, BPA) | 89.10 | 62.69 | 96.22 | 88.56 |
| PHOCNet (SPP, CSS) | 92.95 | 69.88 | 96.56 | 90.60 |
| TPP-PHOCNet (TPP, BPA) | 93.23 | 73.14 | 96.40 | 90.50 |
| TPP-PHOCNet (TPP, CSS) | 95.82 | 76.39 | 97.05 | 92.63 |

those obtained from an SPP layer. Thus the TPP layer is especially suitable in situations, where a pretrained CNN is used as a deep feature extractor for document images as is done by, e.g., Retsinas *et al.* [146] and Krishnan *et al.* [91]. In order to show that the TPP layer learns to predict better representations than the SPP layer, an additional QbE experiment is run[10]: A total of four PHOCNets and TPP-PHOCNets are trained using the same setup as presented in 7.3.3 where each of the architectures is trained with the BPA and CPS configuration once. At query time, the representations for the word images in the query and retrieval sets are obtained directly from the SPP and TPP layers in the respective CNNs. Using these representations, the previously presented protocols are then applied in order to evaluate the word spotting performance. Tab. 6 shows the results obtained for the different architectures using the configurations BPA and CPS for training.

The PHOCResNets are generally able to achieve superior or at least similar results compared to the PHOCNet and TPP-PHOCNet. This is true for even the datasets with smaller training partitions such as the GW or the benchmarks with smaller training partitions for Botany and Konzilsprotokolle (cf. Tab. 3, Tab. 4 and Tab. 5). This noteworthy as the PHOCResNets contain considerably more parameters than the PHOCNets or TPP-PHOCNet architectures and do not make use of additional regularization strategies. Training a PHOCResNet with the BPA configuration only produces significantly inferior results to the CSS configuration in three experiments, namely QbE and QbS for GW and QbS for the `Train I` partition of Botany. For all other experiments, this setup either produces the significantly best results or results which could not be determined to be significantly different from all other configurations. In combination, training a PHOCResNet with either the BPA or CSS configuration resulted in the best performance in 18 of the 20 QbE and QbS experiments. For the two experiments, where a PHOCResNet did not achieve the top performance (QbE and QbS on Esposalles) the obtained performance could not be determined to be significantly different from best performance.

### *Comparison to Results from the Literature*

As can be seen from Tab. 3, Tab. 4 and Tab. 5 the proposed Attribute CNN approach achieves state-of-the-art results on all datasets in both QbE and QbS scenarios except for the `Train I` partition of the Botany dataset. The most likely reason for this is that the training set size of this partition in combination with the difficulty of the Botany dataset is too small in order for the Attribute CNNs to learn robust representations. In

10 Please note that a QbS experiment can not be run directly using the feature representations obtained from the SPP and TPP layers as a mapping from string to feature representation is missing.

other datasets such as GW and Konzilsprotokolle, however, a small training set is not detrimental to the performance. In fact, the Attribute CNNs are all able to outperform all other approaches on Konzilsprotokolle for even the smallest training partition of 1849 images.

As the methods from the literature do not report average precision values, the permutation test cannot be used in order to assess the significance of difference in performance between the Attribute CNNs and the other approaches. However, the PHOCResNets already achieve a significantly higher performance than the PHOCNet with BPA configuration which in turn achieves better results than the next best results reported for the Deep Feature Embedding [91] approach. This allows for the reasonable assumption, that the results obtained for the PHOCResNet would also be significantly better than those for the Deep Feature Embedding and the Triplet-CNN [206]. For the GW it can be expected that the permutation test would not find a significant difference of the mAP values for QbE between the proposed CNNs and the Triplet-CNN as there is already no significant difference between the two mAP values obtained for the PHOCResNet CSS and PHOCNet CSS configurations.

### *Run Time Considerations*

When integrating a method into real world word spotting applications, there exist two main aspects which have to be considered. On the one hand, the method must have high retrieval performance in order to give accurate results. On the other and the user expects the system to have low retrieval times. The experiments did already show that the proposed Attribute CNNs are very capable of fulfilling the first goal. For evaluating their retrieval times, one has to consider two different points in time: training time and query time. At training time, the CNNs are allowed to be fitted to the data at hand. Run times here are not as critical as they can be considered offline precomputations. The eventual user of a word spotting application will not be affected by these times directly when querying a database. However, if the user supplies a new document image collection, these precomputations should still be in a reasonable time frame and not take weeks or months before the user can even begin with retrieving elements. In contrast to training time, a user typically demands a responsive system at query time which is able to run the retrieval in a minimal amount of time.

In order to assess the applicability of the proposed Attribute CNNs in word spotting applications, the training and query times are evaluated. Of course, the exact training times depend on the number and size of the word images in the datasets and the architecture used. Training of the different CNNs finished after 9 to 18 hours for all datasets and architectures when run on a Nvidia Pascal P100 GPU. In addition to training the model, the respective retrieval set representations can be obtained at training time as well as they are not dependent on a query. Hence, this computation does not count towards the query time.

The individual query times for the proposed method thus only depend on the time it takes to generate the query representation plus the time for comparing it to the retrieval set representations and sorting the obtained distance values. For QbS, generating the query representation does effectively not consume any time as it can be directly obtained from the string using very simple computational operations. For QbE, the generation time is the time it takes for the CNN to compute the predicted representations of the given query image, i.e., one forward pass. Tab. 7 lists the described timings in milliseconds for

Table 7: Timings for single a forward pass using a TPP-PHOCNet predicting a PHOC representation. The unit of the presented values is milliseconds.

| Dataset | Forward Pass | | Retrieval |
|---|---|---|---|
| | CPU | GPU | |
| GW | 2191 | 6.2 | 2.0 |
| IAM-DB | 3474 | 6.1 | 1.7 |
| Esposalles | 2676 | 5.0 | 1.0 |
| IFN/ENIT | 4309 | 11.5 | 1.2 |
| Botany | 8493 | 15.5 | 0.2 |
| Konzilsprot. | 4318 | 13.8 | 0.2 |

a TPP-PHOCNet using the PHOC as word string embedding. Computing the distance between the query and retrieval set representations as well as sorting the resulting values was done using Python and the `sklearn` library[11]. For running the forward pass for a given query image, a Nvidia Pascal P100 GPU was used. It should be noted here that using an advanced graphics card like the P100 might not be possible for some applications at query time. The scenario for this might be that a previously trained model shall be employed on a mobile device. Hence, forward pass timings are also evaluated for the scenario when only a CPU is available at query time. For this an Intel Xeon E5-2650 processor is used which is similar to the modern consumer CPU Intel Core i7. It is still assumed here that for training time a GPU was available such that model training and the generation of word image representation for the retrieval set can be completed in an adequate time frame. Training the presented Attribute CNNs on a CPU is practically not possible as is the case for all modern deep neural networks.

From the timings in Tab. 7 it can be seen that a single QbE query takes at most 8.5 s total (forward pass + retrieval) on the CPU and can be as fast as 6 ms when using a GPU. It should be emphasized that the retrieval time for the CPU scenario is almost exclusively due to the forward pass of the query image. The retrieval set size could be increased by a factor of 100 in all experiments which would only add at most 200 ms to the query time irrelevant of the usage of a CPU or GPU. This makes the Attribute CNNs suitable for real-world applications even when faced with limited computation power at query time.

### *Training Set Size Considerations*

One of the stigmas that is still attached to CNNs today is that they require large amounts of annotated training data. The presented Attribute CNNs can be trained from scratch with very limited data as demonstrated in the GW and Konzilsprotokolle experiments. Here, the training sets encompasses only 3615 and 1849 annotated samples respectively. As can be seen from the evaluations during training (Fig. 27 and Fig. 28), the regularizations added to the proposed networks largely prevent overfitting.

However, an inevitable question that arises is how far the number of training images can be reduced without losing too much of the retrieval performance. When faced with few training samples for training a CNN, there exists a common regularization strategy called

11 http://scikit-learn.org/

Table 8: Results for experiments the experiments on GW, IAM-DB and Esposalles when using synthetic data for pretraining and small amounts of data for finetuning. All values are mAP percentages. The results presented here were first published in [61].

| Training Subset Size | GW | | IAM | | Esposalles | |
|---|---|---|---|---|---|---|
| | QbE | QbS | QbE | QbS | QbE | QbS |
| 100 | 83.05 | 86.69 | 38.45 | 56.47 | 89.67 | 71.15 |
| 250 | 90.76 | 92.39 | 43.78 | 60.90 | 94.06 | 82.43 |
| 500 | 93.86 | 94.82 | 52.41 | 68.33 | 95.14 | 85.42 |
| 1000 | 95.74 | 96.59 | 55.39 | 74.09 | 96.27 | 89.18 |
| complete | 97.64 | 97.68 | 85.34 | 92.10 | 97.31 | 93.72 |

finetuning which was not used for the previously presented results. In finetuning, a large annotated dataset is used in order to put the weights of a CNN in a supposedly good range for a smaller dataset to which the CNN is then adapted, i.e., finetuned. This approach, however, only shifts the manual effort as the larger dataset still has to be annotated in order to be used for pretraining. Of course, this manual effort can be mitigated to almost zero if the annotation can be generated synthetically.

For assessing the possibility of pretraining an Attribute CNN, a small amount of additional experiments is conducted[12]. In these experiment, a TPP-PHOCNet is trained using synthetically generated data. The resulting network is then finetuned to small portions of the training partition of a given dataset. In order to keep the amount of experimental evaluations small, only the benchmarks using the Almazán protocol are evaluated. As synthetic dataset the HW-SYNTH [90] is chosen. This dataset was created by using 750 different fonts for rendering word images for 10 000 classes from the Hunspell dictionary. For each class, 100 from the possible 750 fonts are randomly sampled and each font is used in order to render one word image per font. For each rendering process, the string to be used, i.e., the string representing the word class, is randomly transformed to either capitalizing the first letter, capitalizing the whole string or using the lower case version. A single word image is obtained by randomly rendering the individual characters with different distances, different stroke widths and random ink and background pixel distributions. Overall, 1 000 000 word images are created this way which form the HW-SYNTH. The official training partition contains 750 000 word images and is used for pretraining a TPP-PHOCNet. The data used for finetuning is obtained by randomly sampling word images from the respective training partitions. As the HW-SYNTH only encompasses Latin script, the following experiments are only conducted for the GW, IAM-DB and the Esposalles. The word spotting protocol is the same as specified in Sec. 7.3.2. Finetuning is carried out for 4000 iterations after which the training loss could not be decreased any further.

Tab. 8 lists the results obtained when randomly sampling 100, 200, 500 and 1000 word images for finetuning. In the experiments, the entire test partition was used as retrieval set and the numbers reported for GW are, again, obtained through cross validation. As can be seen from the table, for GW and Esposalles competitive results can be obtained with only 1000 randomly sampled training images. This is roughly four pages of manual annotation effort for the GW. The results on the IAM-DB are considerably worse compared to using

12 The results shown here were previously published in [61].

the entire training set. A reasonable explanation is that the IAM-DB training partition was created by more then 600 writers. The large variability of the different writing styles can not be captured using only 1000 training samples. However, in comparison to the results obtain with an AttributeSVM on the entire dataset the results are not worse (cf. Tab. 3). This goes to show that the CNN does not fail when faced with heavily reduced training sets provided that there exists a synthetically generated dataset which can be used for pretraining.

A notable aspect of pretraining the TPP-PHOCNet with the HW-SYNTH is that the final results after finetunung are usually obtained after 1000 iterations. These 1000 iterations take between 7 and 11 minutes for the evaluated datasets. This is especially important for word spotting applications. If a CNN pretrained on synthetic data is available, the user can be asked to only annotate about 1000 words and have the CNN obtain competitive results after a very short amount of time. As the pretrained CNN does only depend on the synthetic data and can be used for any dataset matching the script used for synthesis, this allows for very fast adaption to new and unseen data.

## 7.4 VISUALIZATION OF THE FILTERS

The results presented in the previous section show that the proposed Attribute CNNs achieve results similar to or beyond the state-of-the-art. It is, of course, interesting to investigate the cause of this strong performance, i.e., what the different Attribute CNNs base their predictions on. A possible approach for this is to visualize the parts of certain images for which filter kernels in a given layer get activated the most. Informally speaking, these visualizations show what the filters have learned, i.e., which structures in the respective images they detect.

The visualization of the filter kernels in the first layer is trivial as these are simply the weights of the specific kernels. These visualizations are generally not informative as the first layer of a CNN can generally be shown to learn filters which serve as edge or blob detectors (cf. e.g. [92, 183, 213]). Visualizing the filter kernels of "deeper" hidden layers is more interesting but also not as straight forward as visualizing the filters of the first layer. There exist a number of different approaches for inspecting these hidden kernels but the three most prominent ones are proposed by Zeiler and Fergus [213], Simonyan *et al.* [180] and Springenberg *et al.* [183]. These will be discussed in the following.

Zeiler and Fergus [213] make use of a deconvolutional neural network which is attached to the layer to be inspected. The goal is to map back the activations obtained from a desired layer to the input space, i.e., image pixels. For this, the deconvolutional network has the same architecture as the network to be inspected but flips the layers. Thus convolutional become deconvolutional layers and pooling become unpooling layers. Without going into the specifics of the layers, this flipping procedure is essentially equivalent to a backwards pass through the network. The only difference to a standard backwards pass is how the non-linearities, i.e., ReLU activation functions, are handled. Zeiler and Fergus argue, that the deconvolutional neural network should always produce valid feature maps in each of its layers. Hence, the ReLU function is applied to the output of each deconvolution. The combination of the network to be inspected plus the attached deconvolutional neural network is thus very similar to a convolutional auto-encoder architecture. The weights of the deconvolutional part are however not trained but are simply the transposed weights of the original network.

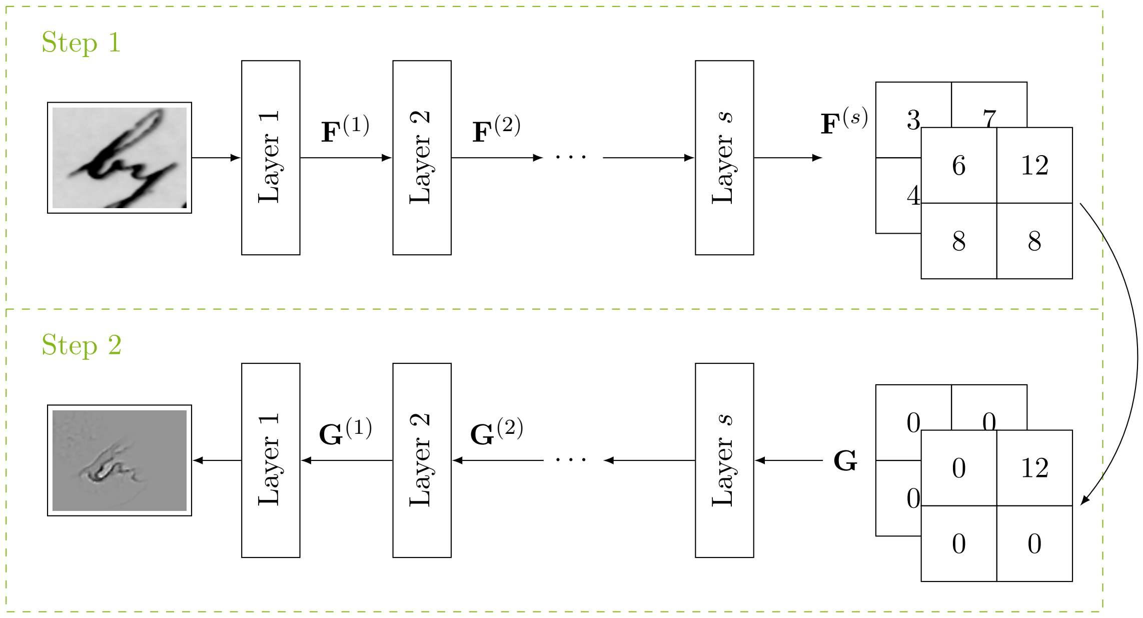


Figure 29: The figure displays the guided backpropation approach for visualizing filter kernels. The first step is to run a forward pass of a given image up to the layer $s$ in which the filter to be visualized resides. The second step is then to set all feature maps in this layer to zero except for the largest element of the feature map corresponding to the desired filter. The resulting feature maps are then used as gradient for layer $s$ and are backpropagated through the neural network until the gradient for the image is obtained.

A different method for visualizing the inner filter kernels of a CNN is the class saliency approach by Simonyan *et al.* [180]. Here, the approach is to compute the gradient of a desired neuron with respect to the image. This way, one can find the pixels in an image which need to be adapted the least to have the strongest influence on the inspected neuron. The class saliency approach is, of course, strongly related to the deconvolutional neural network approach by Zeiler and Fergus. Essentially, the only difference is how the activation function is handled. Interpreting the deconvolutional network as backward pass, the ReLU units are reversed with respect to a real backward pass. In contrast, the class saliency approach computes the gradient of the ReLU.

In the third prominent work on visualizing filter kernels, Springenberg *et al.* [183] argue that the standard backwards pass as done by Simonyan *et al.* [180] incorporates negative gradients as well, i.e., gradients which lower the activation of the unit to be inspected. They claim that this is detrimental to the visualization as the goal is to find the image pixels which need to be changed in order to have a positive influence on the neuron to be inspected. Hence, Springenberg *et al.* propose to combine the deconvolutional neural network approach with the class saliency method. They term this visualization strategy guided backpropation as it guides the gradient to the pixels with a positive influence. Guided backpropation has not only been used for visualizing inner filters of CNNs but also for weakly supervised learning [171].

In the following, guided backpropation will be used in order to visualize what certain internal filters of the Attribute CNNs have learned. Fig. 29 gives an abstract overview over this visualization technique. An image is passed through the neural network up to the layer $s$ which contains the filter to be visualized. All feature maps in this layer are then set to

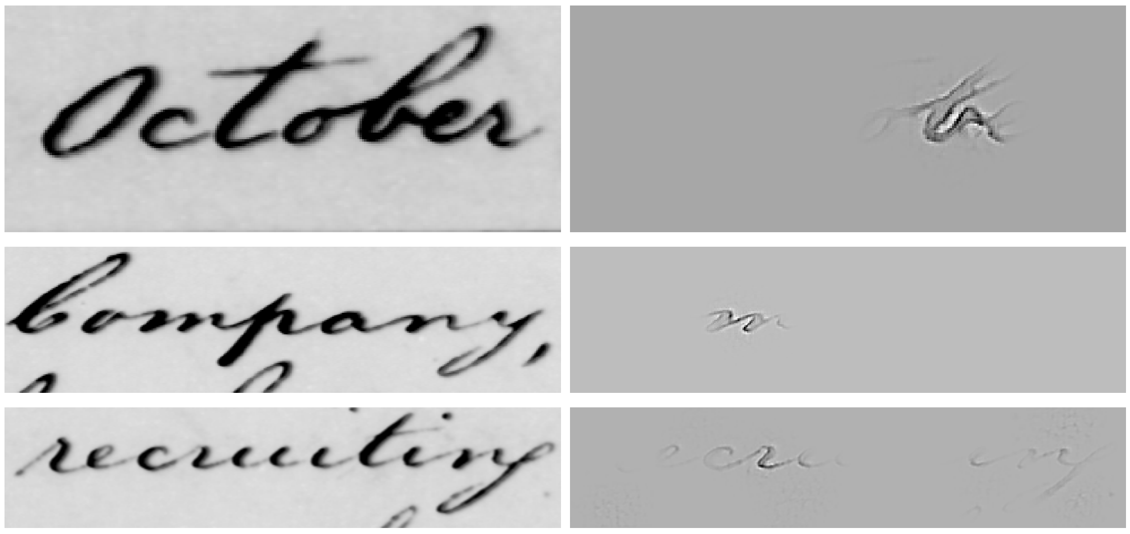


Figure 30: The figure shows examples of visualizations obtained for the last convolutional layer of a TPP-PHOCNet trained on the GW dataset. As can be seen, the filters concentrate on individual characters.

zero except for the largest value in the feature map obtained from the desired filter. This way, a very sparse feature map is obtained which is subsequently used as “start” gradient for the layer $s$ in a backpropagation step. Instead of computing only the gradients for the inputs of the different layers before $s$, however, the gradient of the image is computed. This gradient is then the visualization for the desired filter kernel. In order to obtain a guided gradient of the input image, the ReLU non-linearities are handled specially: For any ReLU in the network, the gradient is computed as follows:

$$\mathbf{G}^{(r)} = \left(\mathbf{F}^{(r)} > 0\right) \odot \left(\mathbf{G}^{(r+1)} > 0\right) \odot \mathbf{G}^{(r+1)}, \tag{85}$$

where $\mathbf{G}^{(r+1)}$ is the gradient from the layer after the ReLU in the architecture, i.e., the preceding layer during backpropagation, $\mathbf{F}^{(r)}$ is the result from the ReLU function from the forward pass and $\mathbf{G}^{(r)}$ is the guided gradient of the ReLU.

Due to the massive amount of layers and filters in the different architectures, only a selected number of visualizations obtained from guided backpropation for the Attribute CNNs will be shown. In the following, filters from the last convolutional layer of the TPP-PHOCNet are visualized. The reason for this is, that these filters are expected to be the most interesting in terms of detecting entire structures in the word images. For natural images, it could be shown already that the filters in the last convolutional layer of a CNN learn entire objects such as heads of animals and humans or parts of cars (cf. e.g. [180, 183, 213]). It is assumed that for a network trained on word images a similar behavior can be expected, i.e., that individual parts of word images are detected by the filters in the last convolutional layer.

Fig. 30 displays a selected number of visualizations for a TPP-PHOCNet trained on the GW dataset. As can be seen, the individual filters are activated the most by certain characters that can be seen in the word image. This behavior can be observed consistently for the other network architectures and datasets as well. To a degree, the individual filters in the last convolutional layer can thus be considered character detectors. The detections obtained for a single word image are then composed into an attribute representation by the MLP in the proposed Attribute CNN architectures. In order to assess this compositional

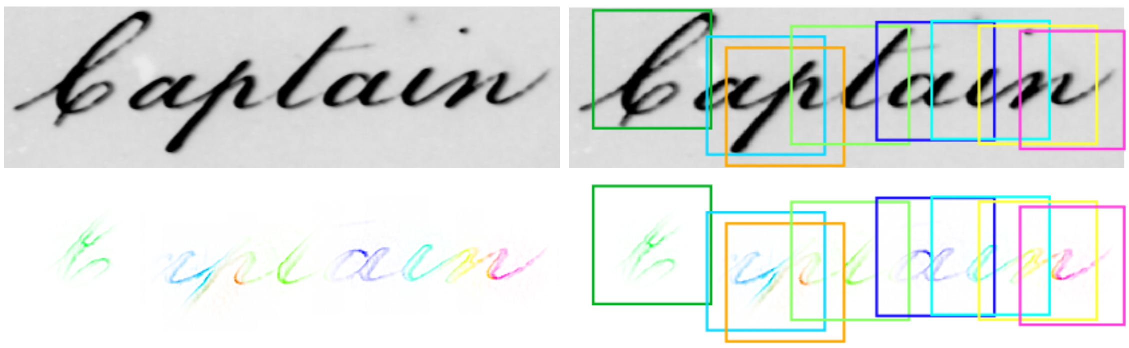


Figure 31: Visualization of the eight filters showing the highest activation for the given word image.

behavior, Fig. 31 visualizes the eight filters with the numerically largest activations in the last convolutional layer of the TPP-PHOCNet for a word image from the GW showing the word *Captain*. On the left side of the figure the word image itself as well as the gradients obtained from guided backpropation are displayed. The different colors in the gradient image correspond to the different filters. The right side of the figure additionally shows the receptive fields for the different filters at their highest activation. First, it can be seen that the different filters really split the task of character detection with each filter concentrating on a different character. In addition, there exists a filter which is responsible for detecting the transition between the first *a* and the *p* (light blue). An interesting observation is, that the different filters are activated the most by the core area of the word image and largely ignore the ascender and descender parts. As can be seen, this behavior is not due to ascenders and descenders vanishing due to fading ink. Apparently, these parts of the characters are not relevant for making a decision about attributes later in the architecture.

The filter displayed in pink can be considered a “false detection” as this filter has actually learned to detect the character *r*. For the given word image, it is wrongfully detecting an *r* in the last character *n*. This behavior is actually plausible as the *r* characters in the GW dataset look quite similar to the last portion of an *n* (cf. Fig. 30). Despite the strong performance of the Attribute CNNs, these false detections of characters occur throughout the different architectures. It seems, that the MLP used at the end of each Attribute CNN is able to eliminate implausible detections quite efficiently when predicting the attribute representations.

# 8 CONCLUSION

After reporting the results of the experimental evaluation of the proposed Attribute CNNs in the previous chapter, this chapter concludes the thesis. For this, a summary of the contents, results and findings is given first (Sec. 8.1). Afterwards, it is discussed how the proposed Attribute CNNs can be extended in order to cope with segmentation-free word spotting tasks (Sec. 8.2). The following section discusses the most recent activities in the literature concerning Attribute CNNs (Sec. 8.3) while the ensuing section discusses fitness of applicability for Attribute CNNs in a real world scenario (Sec. 8.4). Finally, section Sec. 8.5 closes the thesis by given an outlook on potential future work in the field of attribute-based word spotting.

## 8.1 SUMMARY

In this thesis, a method for word spotting was proposed which is capable of segmentation-based, supervised Query-by-Example (QbE) and Query-by-String (QbS) word spotting. For this, a CNN is trained to predict attribute or attribute-like representations of word strings. Three different CNN architectures have been investigated in this regard. All of them are capable of accepting word images of arbitrary size while producing a fixed-size output representation. In order to determine which combination of attribute representation and CNN architecture is most suitable for the word spotting task an extensive empirical investigation has been conducted. It could be shown, that for easier datasets, i.e., datasets with a small amount of visual variability, the different architectures and attribute representations perform almost alike.

The core contribution of this thesis is a probabilistic interpretation of the different loss functions used to train the Attribute CNNs. This interpretation allows for determining the assumptions made when employing the specific functions. Using the Generalized Linear Model (GLM), the well-known Binary Cross Entropy Loss (BCEL) could be derived. As was shown, the assumptions made when training with the BCEL as loss function are that the attribute representations used as labels are a set of pairwise independent binary values. This assumption is violated in one of the most-often used attribute representations for word spotting, the Pyramidal Histogram of Characters (PHOC) representation. In order to alleviate this violation, a different loss function is derived with the prerequisite that the attribute vectors used as labels follow a von Mises-Fisher distribution. The resulting loss function is the Cosine Loss which can not only be used for attribute representations but also attribute-like representation such as the Spatial Pyramid of Characters (SPOC) or the Discrete Cosine Transform of Words (DCToW).

The experimental evaluation, however, shows that the independence assumption is not critical for training the proposed Attribute CNNs. Using three different attribute representations and the two loss functions mentioned above, no combination stood out as consistently superior to the others. Even decorrelating the representations did not improve performance and sometimes even diminished it. While performance-wise no clear winner can be determined, the combination of BCEL, PHOC and Adam optimization consistently lead to the fastest convergence times. It can thus be considered the recommended combination for the presented method.

The experimental evaluation also showed that the Attribute CNN-approach leads to state-of-the-art results for segmentation-based, supervised QbE and QbS word spotting. While for the easier benchmarks such as the one defined on the George Washington Database (GW) no significant difference to other recent approaches can be determined, the presented approach outperforms them on harder benchmarks such as the one defined on the IAM Handwriting Database (IAM-DB). The organizers of the 2016 Handwritten Keyword Spotting Competition even attest the TPP-PHOCNet "overwhelming superiority" with respect to other methods entered into the competition [141] given an adequate amount of training data.

An interesting aspect about the proposed CNNs is that they learn character detectors in the final convolutional layer without being given a character-level segmentation. This trait could potentially be exploited for building weakly supervised character detectors which only require word-level annotations.

As could be shown in the experimental evaluation, the approach of using CNNs for predicting attributes for word spotting bares a number of advantages over methods presented in the literature. In comparison to the influential AttributeSVM [9] or Deep Feature Embedding [91], Attribute CNNs allow for an end-to-end learning approach which greatly increases the performance compared to the classic approach of handling features and attribute predictors separately. Additionally, the presented approach does not require extensive pretraining on labeled data as is done for Triplet-CNNs [206]. In contrast to other CNN-based methods for word spotting, predicting attribute representations with the presented neural networks is based on a solid theoretical foundation: The loss functions are designed specifically to account for the characteristics of the respective attribute or attribute-like representations which shall be predicted.

## 8.2 SEGMENTATION-FREE WORD SPOTTING

One aspect, which was not in the scope of this thesis, is how the presented Attribute CNNs can be used effectively in situations were segmented word images are not available for retrieval, i.e., segmentation-free word spotting. In order to be used in real word applications, this is an important aspect as manually creating a word-level annotation for document images of handwritten text is very cumbersome. A user would typically expect a word spotting system to handle the segmentation step automatically with only a bare minimum of user interaction. An approach how to integrate the presented Attribute CNNs into a segmentation-free method was presented by Rothacker *et al.* [160]. Here, a number of word hypotheses are predicted for a given document image for which a TPP-PHOCNet predicts the attribute representations. In contrast to typical sliding-window based approaches for QbE word spotting, e.g., [159, 165], the word hypotheses-based approach has the advantage that it can effectively deal with words in the document image which are relevant to the query but exhibit a substantial difference in size. Another important aspect is, that the attributes only need to be predicted for the regions of the document image for which a word is hypothesized. Besides the approach by Rothacker *et al.* [160], Ghosh and Valveny [50] use the PHOCNet for segmentation-free word spotting by incorporating a region of interest layer as is done for R-CNNs used for object detection (cf. e.g. [52]). A similar approach, into which the Attribute CNNs could also be integrated, is presented by Wilkinson *et al.* [207]. Here, the authors employ a region proposal network in combination with a heuristic method in order to predict word hypotheses. The resulting hypotheses are also processed by a CNN in order to predict attribute representations as is done by Rothacker *et al.* [160]. This CNN could easily be replaced by one of the proposed Attribute CNNs.

## 8.3 ENSUING WORK ON ATTRIBUTE CNNS

The work on Attribute CNNs proposed in [188] and [189] has sparked a number of research activities. In [147], Retsinas *et al.* examine two properties of the TPP-PHOCNet: First, they investigate whether using a word image's original size gives better results compared to using fixed-size images. This way, they evaluate the necessity of the Spatial Pyramid Pooling (SPP) or Temporal Pyramid Pooling (TPP) layers used in the PHOCNet and TPP-PHOCNet respectively. They find that the approach using fixed-size images leads to results comparable to those obtained for keeping the original image size.

They second question they investigate is whether a pyramidal approach as used in the SPP and TPP layers is necessary in order to achieve state-of-the-art performance or if a single pyramid level suffices in order to obtain a similar performance. This approach is also known as zoning (cf. e.g. [173]). They find that the zoning-based approach leads to a small performance drop but do not evaluate whether the drop in performance is significant. These results lead them to the conclusion that the zoning-based approach performs as well as using SPP or TPP layers in a CNN architecture designed for predicting attributes.

The findings by Retsinas *et al.* [147] are somewhat contradicted by a different work on using Attribute CNNs for word spotting presented by Rusakov *et al.* [163]. In this work, the authors extend the attribute-based word spotting approach by not performing retrieval based on a nearest neighbor search but rather a probabilistic retrieval: After predicting a PHOC representation for a given query, the posterior probability for all PHOC representations in the database to retrieve from given the query representation. This approach is very similar to how Lampert *et al.* [96] use Direct Attribute Prediction (DAP) in order to predict classes from attribute representations (cf. Sec. 2.3). Having obtained the posterior probabilities, Rusakov *et al.* [163] construct the retrieval list by sorting all word images in the database according to the probability of their PHOC representation given the query representation. Here, the query representation is either obtained directly when performing QbS or predicted from an Attribute CNN when performing QbE. In their work, Rusakov *et al.* [163] extend on the original TPP-PHOCNet architecture by adding a SPP and a TPP layer in parallel. The concatenation of the output of both layers serves as input to the Multilayer Perceptron (MLP) part of the CNN. The resulting Attribute CNN is able to outperform the results obtained by Retsinas *et al.* [147] by a considerable margin. This indicates that pyramidal pooling may actually help in obtaining top performances.

## 8.4 DISCUSSION OF APPLICABILITY

The major motivation for word spotting, as stated in Chap. 1, is the possibility for historians to effortlessly browse through large corpora of digitalized historic documents. The question is whether the presented Attribute CNNs pose a viable option to be used as core component in such a word spotting application. The question can be paraphrased by asking whether Attribute CNNs are able to deal with large corpora of diverse and historic writing styles in scans of degraded document images. In the experimental evaluation it could be shown, that this is exactly the case (cf. Sec. 7.3). While the GW and IAM-DB might not be ideally suited to support this claim, the other datasets clearly allow for making this statement: The Esposalles dataset was collected from multiple writers and exhibits a considerable amount of degradation due to its age. The datasets Botany in Britsh India (Botany) and Alvermann Konzilsprotokolle (Konzilsprotokolle) can both also be considered historic with Konzilsprotokolle even featuring a different script than the others (Kurrent). In addition, the number of training samples of Botany and Konzilsprotokolle is quite limited compared to Esposalles for even the largest training partitions. Yet despite all these aspects, the presented Attribute CNNs are able to achieve state-of-the-art results on all of them. This demonstrates that they are very capable of handling historic documents as well as multiple scripts and languages. The last point is emphasized even more by the superb results obtained for IFN/ENIT Database (IFN/ENIT) which features Arabic script. In addition to the performance, the retrieval run times are also adequate for using the proposed Attribute CNNs in real applications (cf. Tab. 7).

## 8.5 OUTLOOK

Over the recent past, results for segmentation-based word spotting have come close to perfect even for the most challenging benchmarks like IAM-DB. Today, different approaches are able to achieve mean Average Precision (mAP) values in the mid or high 90's on all common segmentation-based benchmarks. In comparison to this, segmentation-free word spotting methods still have a larger room for improvement when it comes to performance. Recent approaches here have also started to incorporate deep neural networks in hopes of replicating their success for segmentation-based scenarios. Whether neural networks are used or not, the vast amount of recent work on segmentation-free word spotting does not incorporate an end-to-end training procedure. Judging from the results obtained for other fields of research as well as from the ones obtained in this thesis, it can generally be expected that a method for segmentation-free word spotting making use of the end-to-end paradigm would principally be better from a performance point of view than others making use of the classic computer vision approach. To the best of the authors knowledge, the only work on end-to-end learning for segmentation-free word spotting is presented by Wilkinson *et al.* [207]. Here, the authors make use of a Region Proposal Network (RPN) (cf. [145]) in combination with a heuristic approach in order to predict hypotheses for regions in a digital document image where word images may lie. For these regions, a PHOC or DCToW representation is predicted which in turn can then be used for word spotting using the nearest neighbor approach as is done for segmentation-based word spotting. The region hypotheses are obtained by combining both the output of the RPN and other hypotheses generated from an approach known as Dilated Text Proposals (DTP) [205] which is based on finding connected components in a document image. While the results reported for the segmentation-free benchmarks using GW and IAM-DB are state-of-the-art or at least comparable to it, the results also suggest that the DTP plays a more important role for creating reliable word hypotheses compared to the RPN: The recall for DTP on the segmentation-free IAM-DB benchmark is at 97.9 % while the RPN is only able to find 39.0 % of all word images[1] . The total recall of 98.1 % indicates that the RPN helps almost nothing in detecting words in a document image. While the authors do not elaborate on this, a possible explanation for this behavior may be that RPNs are

1 Recall values obtained at 50 % overlap.

designed to usually detect typically well below 50 elements in an image. In contrast to this, the document images in the presented tasks typically contain 200 or more word images which are sometimes overlapping making it very hard to reliably detect boundaries. In contrast to document images, RPNs work very well for text detection in natural images (cf. e.g. [125]) where the number of instances to be detected is also well below 200. This serves as an indicator that the standard RPN might not be ideally suited for document images of handwritten words. Hence, a new approach would be required which can be trained in an end-to-end fashion all while being able to predict in the order of 200 word images per page. Finding such an approach could potentially help in achieving as high mAP values in segmentation-free scenarios as are obtained for segmentation-based word spotting using the proposed Attribute CNNs.

# A PUBLICATIONS BY THE AUTHOR

As stated in Sec. 1.1, the main contributions presented in this thesis were previously published in peer-reviewed conferences or journals. While in Sec. 1.1 the contributions in previous publications were stated in a brief fashion, this chapter gives a more detailed overview of the different publications which served as base for this dissertation. In the following, the publications are listed in order of publication date.

**Sebastian Sudholt and Gernot A Fink. "A Modified Isomap Approach to Manifold Learning in Word Spotting." In: *Proc. of the German Conference on Pattern Recognition*. 2015, pp. 529–539**,
*Bibliography entry: [187]*
In the work presented in [187], a dimensionality reduction method for sparse and high-dimensional Bag of Features (BoF) representations is proposed. Based on Multi Dimensional Scaling (MDS), this method is used in order to obtain smaller and dense representations for spatial pyramid representations of word images for word spotting. It is shown experimentally, that the obtained representations maintain their individual distances with respect to the original representation more faithfully than the well-known Latent Semantic Indexing (LSI). The method presented in [187] is used for obtaining the decorrelated attribute representations in this thesis (cf. Sec. 5.2): All embeddings denoted with the MDS prefix in the experiments (cf. Tab. 2) are obtained by transforming them into their respective MDS embedding using the cosine distance for computing the required distance values.

**Sebastian Sudholt and Gernot A Fink. "PHOCNet : A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents." In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition*. 2016, pp. 277–282**,
*Bibliography entry: [188]*
The work presented in [188] constitutes the initial ideas and concepts of how Convolutional Neural Networks (CNNs) can be used in an end-to-end fashion in order to predict attribute representations for word spotting. As part of this, the CNN architecture known as PHOCNet (cf. Sec. 5.4.1) is proposed. In contrast to previous works on neural networks for document image analysis, this CNN does not require its input images to be scaled to a fixed size. Another notable difference is that it predicts Pyramidal Histograms of Characters (PHOCs), i.e., attribute representations compared to other approaches predicting either word class labels or character sequences. In [188], the PHOCNet is trained to predict the PHOC representations as proposed by Almazán *et al.* [9], i.e., using the levels 2,3,4 and 5 for unigrams and a left-right split level for the 50 most common bigrams in the English language.

**Sebastian Sudholt and Gernot A. Fink. "Evaluating Word String Embeddings and Loss Functions for CNN-based Word Spotting." In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2017, pp. 493–498**,
*Bibliography entry: [189]*
The work presented in [189] extends on [188] in two critical aspects: The first is to evaluate the three word string embeddings used in this thesis also (cf. Sec. 5.2) in order to determine whether there exists a single best performing embedding. This question can also be interpreted as whether there exists an embedding which can be learned "easier" than the others in an end-to-end setting. The Cosine Loss is used in order to be able to learn the attribute-like embeddings. As the experimental evaluation shows, this is not the case, i.e., no significant difference in performance can be determined for the different embeddings.

The second part of this work examines the PHOCNet and proposes a new and improved version of it by replacing the originally used Spatial Pyramid Pooling (SPP) layer with the Temporal Pyramid Pooling (TPP) layer (cf. Sec. 5.4.2). This layer also accepts variably sized feature maps and produces a fixed-size representation but is specifically designed for word images as input to the respective neural network. Using this layer, a new CNN architecture is proposed for word spotting known as TPP-PHOCNet. As the experimental evaluation shows, the TPP-PHOCNet is able to significantly improve on the performance obtained by the original PHOCNet.

**Sebastian Sudholt, Leonard Rothacker, and Gernot A. Fink. "Query-by-Online Word Spotting Revisited: Using CNNs for Cross-Domain Retrieval." In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2017, pp. 481–486**,
*Bibliography entry: [191]*
The method proposed in [191] constitutes the first method for Query-by-Online-Trajectory (QbO) word spotting using CNNs. In order for the CNN to accept online trajectories as input, the trajectories are first rendered into offline images. An added benefit of this approach is that it allows to disregard preprocessing techniques as the CNN is capable of coping with the data at hand without this initial step. Using the rendered images, two approaches for QbO are then proposed which are also examined in this thesis (cf. Sec. 5.1): For the first, two networks are trained, one for the query image and one for the test image domain. The second approach uses a single CNN to learn to predict attribute representations for both domains combined. As is shown in the paper, the two proposed approaches are able to outperform the previous state-of-the-art using handcrafted features and AttributeSVMs by a considerable margin.

**Leonard Rothacker, Sebastian Sudholt, Eugen Rusakov, Matthias Kasperidus, and Gernot A. Fink. "Word Hypotheses for Segmentation-free Word Spotting in Historic Document Images." In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2017**,
*Bibliography entry: [160]*
As stated in the conclusion, one of the major drawbacks of the Attribute CNNs proposed in this thesis, [188] and [189] is that they require previously segmented word images in the database to retrieve from. The approach proposed in [160] improves upon this fact in enabling the previously mentioned CNNs to allow for segmentation-free Query-by-Example (QbE) and Query-by-String (QbS) word spotting. In order to accomplish this, a number of word hypothesis regions are predicted for the document images of a given corpus. These regions are then processed by the TPP-PHOCNet in order to predict attribute represen-

tations. The presented approach is similar to the well-known R-CNNs used for object detection [54]. As part of the hypothesis predictions, a CNN is used which allows for weakly supervised character detection by generalizing the Class Activation Maps (CAM) approach from [217] to attributes.

**Sebastian Sudholt and Gernot A. Fink. "Attribute CNNs for Word Spotting in Handwritten Documents." In: *International Journal on Document Analysis and Recognition* 21.3 (2018), pp. 199–218**,
*Bibliography entry: [190]*
The work presented in [190] combines the methods and findings from [188] and [189] and adds a more thorough experimental evaluation. In addition, a probabilistic interpretation of the training and loss functions used is given which allows for assessing the assumptions made when training with the respective loss functions. This probabilistic interpretation makes use of Generalized Linear Models (GLMs) and is the base for the loss functions derived in Sec. 5.3. Using this approach, the Cosine Loss is derived from the von Mises-Fisher distribution. This connection had not been shown before in the literature and allows for determining the implicit assumptions made when training with the Cosine Loss.

# B MATHEMATICAL DEDUCTIONS

In the following chapter, mathematical deductions concerning earlier claims in Chap. 3 and Chap. 5 are presented.

## B.1 OBTAINING THE CATEGORICAL CROSS ENTROPY LOSS

In the following, it will be shown how to derive the Categorical Cross Entropy Loss using the Generalized Linear Model (GLM) framework as shown in Sec. 5.3.1. The training dataset $\mathsf{S} = \left\{\left(\mathbf{x}^{(i)}, c^{(i)}\right)\right\}_{i=1}^{\mathsf{N}_s}$ is considered to consists of tuples of images or feature representations $\mathbf{x}$ and class labels $c \in \{1, \dots, \mathsf{N}_c\}$ where $\mathsf{N}_s$ is the number of samples and $\mathsf{N}_c$ the number of classes. For each sample, the vector $\mathbf{y}^{(i)}$ indicates the one-hot encoding of $c^{(i)}$. If the annotations $c^{(i)}$ are class labels, it can reasonably be assumed that the dependent variable $Y$ in the GLM follows a categorical distribution. The categorical distribution is a special case of the multinomial distribution with only a single draw. Its Probability Mass Function (PMF) is defined as follows:

$$f_{\mathcal{C}}\left(\mathbf{y} \mid \mathbf{p}\right) = \prod_{j=1}^{\mathsf{N}_c} p_j^{y_j}. \tag{86}$$

As $\mathbb{E}[Y] = \mathbf{p}$ in the case of $Y$ following a categorical distribution, the corresponding GLM directly predicts the conditional posterior probability for each of the classes given the corresponding sample $\mathbf{x}^{(i)}$. For this, it makes use of $N_c$ independent linear projections in order to obtain a vector $\mathbf{g}$ which is subsequently processed by the inverse of a single link function. Bringing the categorical distribution into the standard exponential family form (cf. Eq. 62), one obtains the softmax function (cf. Eq. 57) as inverse of the canonical link function (cf. e.g. [18]). The prediction $\hat{\mathbf{p}}$ of probabilities for all classes is then obtained from the GLM as follows:

$$\hat{\mathbf{p}} = \frac{\exp\left(\mathbf{W}\mathbf{x}\right)}{\sum_{j=1}^{\mathsf{N}_c} \exp\left(\mathbf{w}_j^T \mathbf{x}\right)}, \tag{87}$$

where $\mathbf{W}$ are the trainable parameters of the GLM and $\mathbf{w}_j$ is the $j$-th row of $\mathbf{W}$. For training, the negative log-likelihood is minimized:

$$\begin{aligned}\hat{\mathbf{W}} &= \underset{\mathbf{W}}{\operatorname{argmin}} - \sum_{i=1}^{n} \log f_{\mathcal{C}}\left(\mathbf{y}^{(\mathrm{i})} \,\middle|\, \hat{\mathbf{p}}^{(i)}\right) \\ &= \underset{\mathbf{W}}{\operatorname{argmin}} - \sum_{i=1}^{\mathsf{N}_s} \sum_{j}^{\mathsf{N}_c} y_j^{(i)} \log \hat{p}_j^{(i)}\end{aligned}$$

As $\mathbf{y}^{(i)}$ has only one non-zero entry, the function from above can also be rewritten as

$$\hat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} - \sum_{i=1}^{\mathsf{N}_s} \log \hat{p}_{c^{(i)}}^{(i)}$$

This is actually the formulation for the Categorical Cross Entropy Loss that can be found in most deep learning toolboxes[1].

## B.2 DERIVATIVES

In this section, a number of gradients are derived which are important for training neural networks in the context of this thesis.

### B.2.1 *Derivative Sigmoid*

In this section it is shown that the derivative of the sigmoid activation function $\frac{\partial \operatorname{sigm}(x)}{\partial x}$ is indeed $\operatorname{sigm}(x)(1-\operatorname{sigm}(x))$:

$$\begin{aligned}
\operatorname{sigm}(x) &= \frac{1}{1+e^{-x}} \\
\frac{\partial \operatorname{sigm}(x)}{\partial x} &= -\left(\frac{1}{1+e^{-x}}\right)^2 \cdot (-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{e^{-x}}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}} \\
&= \frac{1+e^{-x}-1}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}} \\
&= \left(1-\frac{1}{1+e^{-x}}\right) \cdot \frac{1}{1+e^{-x}} \\
&= \operatorname{sigm}(x)(1-\operatorname{sigm}(x))
\end{aligned}$$

### B.2.2 *Combined Gradient Sigmoid Activation and Euclidean Loss*

It was claimed in Sec. 5.3 that using the Euclidean Loss in combination with sigmoid activation functions in the output layer leads to the overall gradient being scaled by the gradient of the sigmoid. This claim will be proven in this section. Recall that the Euclidean Loss is defined as

$$l_{\mathcal{N}} = \frac{1}{2}\sum_{i=1}^{\mathsf{N}_s} ||\mathbf{y}-\hat{\mathbf{y}}||^2 .$$

The otherwise linear output $\mathbf{o}$ of the last layer is transformed by a sigmoid in order to obtain the output $\hat{\mathbf{y}}$ of the neural network:

$$\hat{\mathbf{y}} = \operatorname{sigm}(\mathbf{o})$$

The combined gradient of both loss function and sigmoid activation function can be expressed by means of the chain rule as is standard for the backpropagation algorithm:

$$\frac{\partial l_{\mathcal{N}}}{\partial \mathbf{o}} = \frac{\partial l_{\mathcal{N}}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{o}}.$$

1 For example, in the Caffe toolbox [74]: https://caffe.berkeleyvision.org/doxygen/classcaffe_1_1MultinomialLogisticLossLayer.html

In order to obtain this gradient, the partial derivatives $\frac{\partial l_{\mathcal{N}}}{\partial o_j}$ are computed separately and then stacked in order to obtain the gradient:

$$\begin{aligned}\frac{\partial l_{\mathcal{N}}}{\partial o_j} &= \frac{\partial l_{\mathcal{N}}}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial o_j} \\ &= -(y_j - \hat{y}_j) \cdot \frac{\partial \operatorname{sigm}(o_j)}{\partial o_j} \\ &= (\hat{y}_j - y_j) \cdot \operatorname{sigm}(o_j)(1 - \operatorname{sigm}(o_j)) \\ \Rightarrow \frac{\partial l_{\mathcal{N}}}{\partial \mathbf{o}} &= (\hat{\mathbf{y}} - \mathbf{y}) \odot \operatorname{sigm}(\mathbf{o}) \odot (1 - \operatorname{sigm}(\mathbf{o}))\end{aligned}$$

As can be seen from the equation above, if the linear output $\mathbf{o}$ has large absolute values, the gradient is scaled by a very small factor. This can be seen as a version of the vanishing gradient problem.

### B.2.3 *Combined Gradient Sigmoid Activation and Binary Cross Entropy Loss*

In this section it will be shown that, in contrast to the combination of sigmoid activation function and Euclidean Loss, the combined gradient of the loss and activation function is not scaled by the derivative of the sigmoid if the Binary Cross Entropy Loss (BCEL) is used as loss. Recall that the BCEL is defined as

$$l_{\mathcal{B}} = -\sum_{i=1}^{\mathsf{N}_s} \sum_{j=1}^{\mathsf{D}} y_j^{(i)} \log \hat{y}_j^{(i)} + \left(1 - y_j^{(i)}\right) \log \left(1 - \hat{y}_j^{(i)}\right).$$

For simplicity, the gradient will be computed for a single sample only. As was done in the previous section, the combined gradient for sigmoid activation function and BCEL is derived by computing the partial derivative with respect to each dimension $j$ of the linear output vector $\mathbf{o}$ to the sigmoid function separately. The results are then stacked in order to obtain the gradient:

$$\begin{aligned}\frac{\partial l_{\mathcal{B}}}{\partial o_j} &= \frac{\partial l_{\mathcal{B}}}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial o_j} \\ &= -\left(\frac{y_j}{\hat{y}_j} - \frac{1 - y_j}{1 - \hat{y}_j}\right) \cdot \frac{\partial \operatorname{sigm}(o_j)}{\partial o_j} \\ &= \left(\frac{1 - y_j}{1 - \hat{y}_j} - \frac{y_j}{\hat{y}_j}\right) \cdot \operatorname{sigm}(o_j)(1 - \operatorname{sigm}(o_j)) \\ &= \frac{(1 - y_j)\hat{y}_j - y_j(1 - \hat{y}_j)}{\hat{y}_j(1 - \hat{y}_j)} \cdot \hat{y}_j(1 - \hat{y}_j) \\ &= \hat{y}_j - \hat{y}_j y_j - y_j + \hat{y}_j y_j \\ &= \hat{y}_j - y_j\end{aligned}$$

The gradient of $l_{\mathcal{B}}$ with respect to the entire linear output vector $\mathbf{o}$ is thus

$$\frac{\partial l_{\mathcal{B}}}{\partial \mathbf{o}} = \hat{\mathbf{y}} - \mathbf{y}.$$

As can be seen, the combined gradient is not scaled by the derivative of the sigmoid anymore.

### B.2.4 *Gradient for the Cosine Loss*

In order to be able to train a neural network with the Cosine Loss its gradient with respect to the output of the network needs to be computed. As was done in the previous section, only a single sample will be considered in order to derive this gradient. For multiple samples the gradient is simply the mean of all gradients. If only a single sample is used, the Cosine Loss is defined as follows:

$$\begin{aligned} l_{\mathcal{MF}} &= 1 - \frac{\mathbf{y}^T\,\hat{\mathbf{y}}}{||\,\mathbf{y}\,||\cdot||\,\hat{\mathbf{y}}\,||} \\ &= 1 - \frac{\sum_{j=1}^{\mathsf{D}} y_j \hat{y}_i}{\sqrt{\sum_{j=1}^{\mathsf{D}} y_j^2}\cdot\sqrt{\sum_{j=1}^{\mathsf{D}} \hat{y}_i^2}} \\ &= 1 - \underbrace{\left(\sum_{j=1}^{\mathsf{D}} y_j \hat{y}_j\right)}_{a(\mathbf{y},\hat{\mathbf{y}})} \cdot \underbrace{\left(\sum_{j=1}^{\mathsf{D}} \hat{y}_j^2\right)^{-\frac{1}{2}}}_{b(\hat{\mathbf{y}})} \cdot \underbrace{\left(\sum_{j=1}^{\mathsf{D}} y_i^2\right)^{-\frac{1}{2}}}_{c(\mathbf{y})} \end{aligned}$$

In order to make the following computations more visually appealing, the Cosine Loss is expressed in terms of three functions $a, b$ and $c$. With respect to the prediction of the neural network, the function $c$ is constant. Thus for finding the derivative $\frac{\partial l_{\mathcal{MF}}}{\partial \hat{\mathbf{y}}}$, the partial derivatives $\frac{\partial a}{\partial \hat{y}_j}$ and $\frac{\partial b}{\partial \hat{y}_j}$ need to be computed and aggregated:

$$\begin{aligned} \frac{\partial a}{\partial \hat{y}_j} &= y_i \\ \frac{\partial b}{\partial \hat{y}_j} &= 2\hat{y}_j \cdot \left(-\frac{1}{2}\right) \cdot \left(\sum_i \hat{y}_i^2\right)^{-\frac{3}{2}} \\ &= -\hat{y}_i \cdot g^3 \end{aligned}$$

Applying the product rule leads to the gradient $\frac{\partial l_{\mathcal{MF}}}{\partial \hat{y}_j}$:

$$\begin{aligned} \frac{\partial l_{\mathcal{MF}}}{\partial \hat{y}_j} &= \frac{\partial(1 - c \cdot a \cdot b)}{\partial \hat{y}_j} \\ &= -c \cdot \left(\frac{\partial a}{\partial \hat{y}_j} b + a \frac{\partial b}{\partial \hat{y}_j}\right) \\ &= -c \cdot \left(y_i b - a \cdot \hat{y}_j b^3\right) \\ &= cb \cdot \left(ab^2 \cdot \hat{y}_j - y_j\right) \\ &= \frac{1}{||\,\mathbf{y}\,||\cdot||\,\hat{\mathbf{y}}\,||}\left(\frac{\mathbf{y}^T\,\hat{\mathbf{y}}}{||\,\hat{\mathbf{y}}\,||^2} \cdot \hat{y}_j - y_j\right) \end{aligned}$$

$$\Rightarrow \frac{\partial l_{\mathcal{MF}}}{\partial\,\hat{\mathbf{y}}} = \frac{1}{||\,\mathbf{y}\,||\cdot||\,\hat{\mathbf{y}}\,||}\left(\frac{\mathbf{y}^T\,\hat{\mathbf{y}}}{||\,\hat{\mathbf{y}}\,||^2} \cdot \hat{\mathbf{y}} - \mathbf{y}\right)$$

# C QUERY-BY-ONLINE WORD SPOTTING

## C.1 ATTRIBUTE CNNS FOR QBO

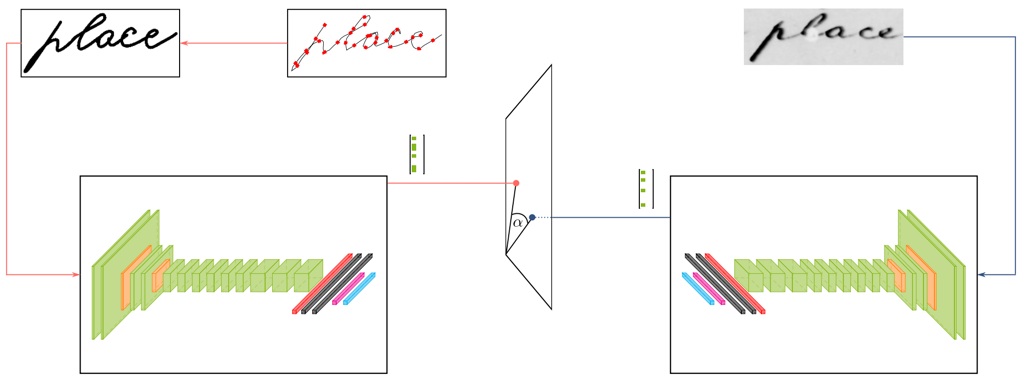
Query-by-Online-Trajectory (QbO) word spotting bares a subtle difference to QbE and QbS: The trajectories of online-handwriting used as queries under this paradigm are inadequate to be used with CNNs as these neural networks are designed for images as input. A simple and straight forward approach is to render the trajectories into word images. These word images can then be processed by a given CNN. The advantage of using the rendered "offline versions" of the trajectories is that CNNs can usually cope with a large amount of variability and do not require excessive preprocessing. In contrast, when dealing with online trajectories, one typically needs to make use of a number of elaborate preprocessing techniques in order to obtain good results even when using advanced methods such as RNNs [59]. This is all done away with when rendering the trajectories to offline word images and using a CNN. Of course, the images produced this way can be expected to be vastly different from the corpus images, especially when dealing with historical documents. Hence, a straight forward approach for QbO word spotting with Attribute CNNs is to have one CNN predict attributes for the rendered online trajectories and another CNN doing so for the corpus images. This way, the individual CNNs can specialize to the specific data domain they are presented with. Fig. 32a visualizes this approach. In contrast to this, a second approach would be to predict attribute representations for both rendered and corpus images from a single CNN (Fig. 32b). If the CNN is powerful enough to learn the characteristics of both data distributions, this approach may even be better as it would allow the single model to draw connections between the different datasets and attributes.

## C.2 EVALUATION OF QBO WORD SPOTTING

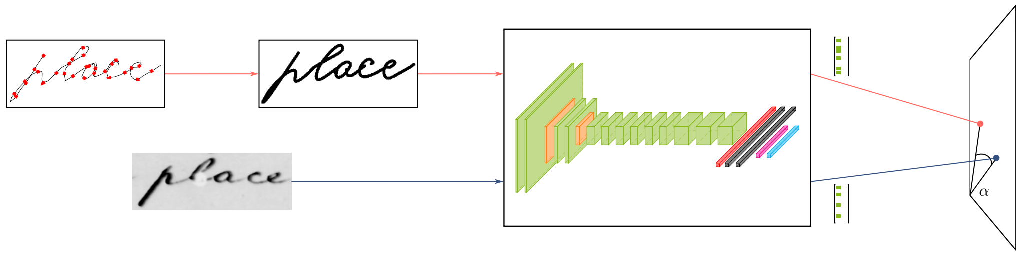
In the following section, the QbO experiments conducted in this thesis are reported. QbO is a rather new query paradigm and the AttributeSVM-based approach by Wieprecht *et al.* [204] was the only one for this specific task before Attribute CNNs were proposed for this in [191]. The benchmarks and protocols proposed by Wieprecht *et al.* differ quite substantially from the Almazaán and competition protocols. In order to compare the results obtained for the presented Attribute CNNs for QbO to the ones reported by Wieprecht *et al.*, the same benchmarks and protocols will be used in this thesis.

The goal in QbO is to retrieve offline word images based on an online trajectory. The benchmarks for evaluating QbO thus consist of two different datasets. The first is a set of online trajectories, which are used as queries. The second is a dataset of offline word images. In the case of supervised word spotting methods, there exists a training partition for both datasets. The remaining words of the online dataset are then used as queries while the remaining offline data is used as retrieval set.

Wieprecht *et al.* [204] propose two QbO benchmarks: The first uses a single writer which contributes online trajectories to both training and query set. For the second benchmark, online trajectories from different writers are used for training, while the query dataset only contains trajectories from a single writer who did not contribute to the training par-



(a) 2-Net Approach



(b) 1-Net Approach

Figure 32: Visualization of the two proposed approaches for using Attribute CNNs for QbO word spotting. In the 2-Net approach (a), one Attribute CNN is trained on images rendered from online-handwritten trajectories while the other is trained on word images extracted from document images. For retrieval, the "online" network predicts the query representations while the "offline" network predicts to the corpus representation. In contrast to this, the 1-Net approach (b) uses a single CNN which is trained on both rendered online trajectories and extracted word images. Both query and corpus representations are predicted from this network at query time.

tition. For both benchmarks, the online as well as offline data are made up of publicly available datasets. In order to compare the performance of the Attribute CNNs for supervised QbO word spotting with the AttributeSVM approach by Wieprecht *et al.*, the same benchmarks will be used. In addition, a third benchmark is proposed. This benchmark focuses on the scenario, where different users issue online queries to a supervised QbO word spotting method while none of these users contributed any online trajectories to the training partition.

In the following, the online datasets used for the three benchmarks will be described first (Sec. C.2.1). As offline datasets, the previously presented GW and IAM-DB will be used (cf. Sec. 7.3.1). These datasets will not be presented again at this point. Having established the online datasets to be used, Sec. C.2.2 presents the three different benchmarks in more detail specifically pointing out the datasets and word spotting protocol used. The section is concluded by presenting the obtained results, comparing them to the ones from Wieprecht *et al.* and a final discussion (Sec. C.2.3). For the experiments, all Attribute CNNs use the combination of BCEL, PHOC and Adam as this configuration has shown itself to be the most effective for the QbE and QbS experiments (cf. Sec. 7.3.5).

### C.2.1 *Online Datasets*

#### *George Washington Online*

The GWO[1] is a dataset of online-handwriting trajectories. It was created for the experiments in [204] in order to perform QbO word spotting using the GW dataset as retrieval set. For this, every word in the GW was copied by a single writer as online trajectory using a stylus pen on an Android tablet. Hence, the GWO consists of 4860 trajectories of online-handwritten text. The writing style is very homogeneous due to being created by a single writer only. Likewise, the scale of the individual trajectories is very consistent.

#### *UNIPEN*

The UNIPEN dataset is a collection of online-handwritten trajectories from the UNIPEN foundation [62][2]. It was originally designed to facilitate a database for evaluating and comparing methods for online handwriting recognition. The dataset comes with annotations and transcriptions at line, word and character level. In order to be able to directly compare the obtained results to the ones reported in [204], the subset *sta0* of trajectories of Latin script is used in the following experiments. This subset consists of online-handwritten trajectories from 62 writers, each contributing roughly 400 trajectories. In total, the *sta0* partition of the UNIPEN dataset consists of 27 112 online trajectories.

#### *IAM On-Line Handwriting Database*

The IAM-OnDB [110] is used as online dataset for the newly proposed QbO benchmark. Like the UNIPEN, the IAM-OnDB was originally designed for online-handwriting recognition. It is made up of 13 049 trajectories of online-handwritten text lines, contributed by 221 writers. The IAM-OnDB does only come with line-level segmentations for the trajectories. As the Attribute CNNs used in this thesis are dependent on word-level segmentations, this segmentation was obtained from the line segmentations using a BoF-HMM by means of a forced alignment. The forced alignment used in this thesis is courtesy of Leonard Rothacker. The exact parametrization of this segmentation procedure are beyond the scope of this thesis. The interest reader is referred to the original publication of Attribute CNNs for QbO [191] for a detailed description.

The IAM-OnDB comes with a number of official partitionings. In the following experiments the `f` partition is used. Similar to the IAM-DB, a single writer does only contribute to either the training or the test split for this partition. Using the test partition as queries thus allows for QbO with multiple writers, unknown from the training partition.

### C.2.2 *QbO Benchmark Description*

This section outlines the benchmarks used for the QbO word spotting experiments in this thesis. For all benchmarks, rendering of the online trajectories is done by sliding a circle along the trajectory. The diameter of this circle differs for the three online datasets and will be referred to as *stroke width* in the following.

In order to simplify explanations, only the training, query and retrieval set used in each experiment are pointed out in the following. For the 2-Net approach, the rendered online

1 The GWO dataset is available at `https://github.com/cwiep/gw-online-dataset`
2 UNIPEN dataset available at `http://www.unipen.org/products.html`

images and offline document images of the respective training set are used to train two different Attribute CNNs while for the 1-Net approach both sets are combined to train a single CNN (cf. Fig. 32). For the query set PHOCs are predicted from the network trained on the rendered online images (2-Net) or the joint network (1-Net). Likewise, representations for the word images in the retrieval set are obtained from the CNN trained on the offline images (2-Net) or the joint network (1-Net). For each experiment, word spotting is performed following the Almazán protocol: Each query PHOC is used to rank the PHOCs obtained for the retrieval set using the Cosine distance. A query is discarded if the corresponding retrieval list has no relevant entry.

*Benchmark 1: Single Known Writer (SKW)*

For the first benchmark, the queries originate from a single writer, while all training trajectories also originate from this writer. As online dataset the GWO is used while the GW serves as offline dataset. In order to be able to train the Attribute CNNs on the online trajectories, they are rendered with a stroke width of 10 pixels. The stroke width was determined by taking five trajectories of the training partition and creating a visually appealing rendered word image. The GWO and GW are then divided into four cross validation splits. For this, the splits defined by Almazán *et al.* [9] are used (cf. Sec. 7.3.1). The training set for this experiment is then the combination of both training splits for GWO and GW. The query set is the test split of the GWO while the retrieval set consists of the test split of the GW.

*Benchmark 2: Single Unknown Writer (SUW)*

For the second experiment, the queries originate from a single writer, who did not contribute other trajectories to the training partition. The datasets used are the UNIPEN, GWO and GW. As was done for the first benchmark, the GWO is rendered with a stroke width of 10 pixels. For UNIPEN, a stroke width of 20 pixels is chosen. The stroke widths were determined as for the first experiment. Afterwards, GWO and GW are again split up into the four cross validation sets. As training set for this benchmark, the entire UNIPEN dataset as well as the respective GW training split is used. Query and retrieval set are, again, the test splits for the GWO and GW respectively. During pre-experimental evaluations it became evident that simply using the query set this way does not allow the CNN to reliably predict PHOCs. This is due to the vastly different scales of the UNIPEN and GWO trajectories and hence the rendered word images. In order to cope with this size mismatch, a scaling factor is determined which is applied to the word images of the GWO. For this, all common words from the UNIPEN and the respective training partition of the GWO are determined first. Then, the average height of these common words is calculated for both datasets. The scaling factor is then the quotient between the average height of the UNIPEN and the average height of the GWO. This procedure roughly aligns the distribution of word image heights for the two datasets.

*Benchmark 3: Multiple Unknown Writers (MUW)*

For the third experiment, the queries originate from multiple writers, who did not contribute other trajectories to the training set. For this, the IAM-DB and IAM-OnDB datasets are used. The IAM-OnDB is first rendered with a stroke width of 20 pixels (stroke width was determined as for Exp. 1 and 2). The training set for this experiment is the

Table 9: Summary of the different datasets for the three QbO benchmarks.

| Benchmark | Training Set | Query Set | Corpus |
|---|---|---|---|
| Ben. 1: SKW | GWO Train + GW Train | GWO Test | GW Test |
| Ben. 2: SUW | UNIPEN + GW Train | GWO Test | GW Test |
| Ben. 3: MUW | IAM-DB Train + IAM-OnDB Train | IAM-OnDB Test | IAM-DB Test |

Table 10: Results for the three QbO experiments. All values are mAP percentages.

| Architecture/Method | Ben. 1: SKW | Ben. 2: SUW | Ben. 3: MUW |
|---|---|---|---|
| PHOCNet (2-Net) | *94.68* | *77.26* | *84.96* |
| PHOCNet (1-Net) | 95.89 | 87.14 | *84.53* |
| TPP-PHOCNet (2-Net) | *94.41* | *77.20* | *84.51* |
| TPP-PHOCNet (1-Net) | 96.42 | **87.22** | *84.92* |
| PHOCResNet (2-Net) | *94.76* | *77.03* | **86.59** |
| PHOCResNet (1-Net) | **96.60** | 86.63 | *85.02* |
| AttributeSVM [204] | 86.49 | 21.71 | – |

union of the two training splits of IAM-DB and IAM-OnDB. The query set is the official test split *f* of the IAM-OnDB. All words from the *f* split are used as queries. Finally, the retrieval set consists of the test split of the IAM-DB. Table 9 summarizes the training, query and test sets for the three experiments.

### C.2.3 *Results and Discussion*

The results for the three experiments and two word spotting systems are reported in table 10. As can be seen in the table, the approach of using CNNs instead of AttributeSVM leads to a considerable performance gain.

Another interesting observation is that the 1-Net approach is able to consistently outperform the 2-Net approach for the SKW and SUW experiments. This result might seem counter intuitive at first as using individual CNNs for the rendered online images and offline document images should enable each CNN to focus better on the characteristics of the respective data. A possible explanation for why the 1-Net approach performs better is that it is able to draw connections between characters of the different data sources: If the net is presented with word images of the same class from both rendered and document images, it is implicitly shown which portions of the word images "match" from an attribute point of view. This enables it to learn a much more robust representation compared to seeing data from one domain only. What is really interesting about this is that the online trajectories are rendered as binary images while the offline word images are grayscale images. Yet, the CNN is able to draw knowledge from combining both binary and grayscale images. This demonstrates that the Attribute CNN approach is very capable of dealing with data of different modalities.

# D RESULTS FOR THE SIGNIFICANCE TESTS

In order to supply a clean visualization, significant differences in performance as displayed in the tables in Chap. 7 were only shown in terms of whether a certain mAP value was significantly better than a second one. In this chapter, the results of the significance tests are displayed more verbosely. For each dataset used in the experimental evaluation reported in Chap. 7, there exist two tables stating the used Attribute CNN and configuration (BCEL, PHOC, Adam (BPA)) or Cosine Loss, SPOC, SGD (CSS)). The columns of the tables are always set up the same way: The first, second and third column state the architecture, configuration and mAP values. The fourth column displays the $p$-value obtained through the Monte Carlo permutation test for testing against the best result. The range of the $p$-value is $[0, 1]$. The fifth column displays the significance of the $p$-value in the star notation of the programming language R[1]. The last column states whether the obtained result is significant with a significance level of 0.01. The first row of the table always shows the best performing configuration for the respective experiment. All other rows show the results in comparison to the first row, i.e., significance is assessed by comparing the result in a given row to the one in the first. If a result is significantly worse than the best result, it is marked with a check mark (✓).

The displayed $p$-values were obtained for $250,000$ Monte Carlo runs of the permutation test. The $p$-value is obtained by dividing the number of permutations with a bigger difference in means than the original difference by the total number of runs (cf. Sec. 7.2). Thus, the number of permutations with a bigger difference can be obtained by multiplying the respective $p$-values with $250,000$.

Table 11: Results for **QbE** on the **GW** database

| Attribute CNN | Configuration | mAP | $p$-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC, SGD | 97.75 | | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 97.64 | 0.667368 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 97.53 | 0.405704 | | |
| PHOCNet | Cosine, SPOC, SGD | 97.42 | 0.224636 | | |
| PHOCNet | BCEL, PHOC, Adam | 97.25 | 0.072532 | . | |
| PHOCResNet | BCEL, PHOC, Adam | 96.95 | 0.005712 | ** | ✓ |

1 A description of R's star notation can be found here: https://www.rdocumentation.org/packages/gtools/versions/3.5.0/topics/stars.pval

Table 12: Results for **QbS** on the **GW** database

| Attribute CNN | Configuration | mAP | $p$-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC, SGD | 98.01 | | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 97.68 | 0.383952 | | |
| PHOCNet | Cosine, SPOC, SGD | 97.55 | 0.243536 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 95.39 | 0.000000 | *** | ✓ |
| PHOCNet | BCEL, PHOC, Adam | 95.13 | 0.000000 | *** | ✓ |
| PHOCResNet | BCEL, PHOC, Adam | 94.56 | 0.000000 | *** | ✓ |

Table 13: Results for **QbE** on the **IAM-DB**

| Attribute CNN | Configuration | mAP | $p$-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | BCEL, PHOC, Adam | 86.82 | | | |
| PHOCResNet | Cosine, SPOC, SGD | 85.51 | 0.019744 | * | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 85.34 | 0.008576 | ** | ✓ |
| PHOCNet | BCEL, PHOC, Adam | 84.61 | 0.000132 | *** | ✓ |
| TPP-PHOCNet | Cosine, SPOC, SGD | 83.27 | 0.000000 | *** | ✓ |
| PHOCNet | Cosine, SPOC, SGD | 82.46 | 0.000000 | *** | ✓ |

Table 14: Results for **QbS** on the **IAM-DB**

| Attribute CNN | Configuration | mAP | $p$-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC, SGD | 94.07 | | | |
| PHOCResNet | BCEL, PHOC, Adam | 93.34 | 0.165812 | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 93.22 | 0.112992 | | |
| PHOCNet | Cosine, SPOC, SGD | 92.70 | 0.011184 | * | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 92.10 | 0.000412 | *** | ✓ |
| PHOCNet | BCEL, PHOC, Adam | 91.71 | 0.000032 | *** | ✓ |

Table 15: Results for **QbE** on the **Esposalles** database

| Attribute CNN | Configuration | mAP | $p$-value | R Symb. | Significant |
|---|---|---|---|---|---|
| TPP-PHOCNet | BCEL, PHOC, Adam | 97.31 | | | |
| PHOCNet | BCEL, PHOC, Adam | 97.30 | 0.956788 | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 97.17 | 0.283956 | | |
| PHOCResNet | BCEL, PHOC, Adam | 97.15 | 0.230356 | | |
| PHOCNet | Cosine, SPOC, SGD | 97.12 | 0.154584 | | |
| PHOCResNet | Cosine SPOC, SGD | 97.10 | 0.116056 | | |

Table 16: Results for **QbS** on the **Esposalles** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| TPP-PHOCNet | Cosine, SPOC, SGD | 94.29 | | | |
| PHOCNet | Cosine, SPOC, SGD | 94.01 | 0.703616 | | |
| PHOCResNet | Cosine, SPOC, SGD | 93.92 | 0.610524 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 93.72 | 0.440540 | | |
| PHOCResNet | BCEL, PHOC, Adam | 93.68 | 0.411240 | | |
| PHOCNet | BCEL, PHOC, Adam | 93.62 | 0.360332 | | |

Table 17: Results for **QbE** on the **IFN/ENIT** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | BCEL, PHOC, Adam | 96.93 | | | |
| PHOCNet | BCEL, PHOC, Adam | 96.57 | 0.078540 | . | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 96.50 | 0.036652 | * | |
| PHOCNet | Cosine, SPOC, SGD | 93.22 | 0.000000 | *** | ✓ |
| TPP-PHOCNet | Cosine, SPOC, SGD | 92.75 | 0.000000 | *** | ✓ |
| PHOCResNet | Cosine, SPOC, SGD | 92.64 | 0.000000 | *** | ✓ |

Table 18: Results for **QbS** on the **IFN/ENIT** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | BCEL, PHOC, Adam | 95.29 | | | |
| PHOCNet | BCEL, PHOC, Adam | 94.81 | 0.529952 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 94.27 | 0.183472 | | |
| PHOCNet | Cosine, SPOC, SGD | 94.00 | 0.091228 | . | |
| PHOCResNet | Cosine, SPOC, SGD | 93.48 | 0.020040 | * | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 93.25 | 0.009288 | ** | ✓ |

Table 19: Results for **QbE** on the **Botany Train I** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC, SGD | 43.70 | | | |
| PHOCResNet | BCEL, PHOC, Adam | 41.70 | 0.472120 | | |
| PHOCNet | BCEL, PHOC, Adam | 41.04 | 0.349052 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 39.86 | 0.171732 | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 38.80 | 0.095748 | . | |
| PHOCNet | Cosine, SPOC, SGD | 33.29 | 0.000224 | *** | ✓ |

Table 20: Results for **QbS** on the **Botany Train I** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC, SGD | 42.27 | | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 38.19 | 0.354536 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 36.47 | 0.173352 | | |
| PHOCResNet | BCEL, PHOC, Adam | 34.10 | 0.054192 | . | |
| PHOCNet | Cosine, SPOC, SGD | 32.97 | 0.036260 | * | |
| PHOCNet | BCEL, PHOC, Adam | 31.94 | 0.019144 | * | |

Table 21: Results for **QbE** on the **Botany Train II** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | BCEL, PHOC, Adam | 80.07 | | | |
| PHOCResNet | Cosine, SPOC, SGD | 78.44 | 0.539032 | | |
| PHOCNet | BCEL, PHOC, Adam | 77.03 | 0.255864 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 76.55 | 0.205992 | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 73.35 | 0.017644 | * | |
| PHOCNet | Cosine, SPOC, SGD | 71.74 | 0.003044 | ** | ✓ |

Table 22: Results for **QbS** on the **Botany Train II** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC SGD | 88.48 | | | |
| PHOCResNet | BCEL, PHOC, Adam | 86.43 | 0.383400 | | |
| TPP-PHOCNet | Cosine, SPOC SGD | 84.42 | 0.113032 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 84.23 | 0.095292 | . | |
| PHOCNet | Cosine, SPOC SGD | 80.83 | 0.006276 | ** | ✓ |
| PHOCNet | BCEL, PHOC, Adam | 76.73 | 0.000068 | *** | ✓ |

Table 23: Results for **QbE** on the **Botany Train III** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | BCEL, PHOC, Adam | 95.92 | | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 92.89 | 0.054260 | . | |
| PHOCNet | BCEL, PHOC, Adam | 92.75 | 0.057732 | . | |
| PHOCNet | Cosine, SPOC, SGD | 80.11 | 0.000000 | *** | ✓ |
| TPP-PHOCNet | Cosine, SPOC, SGD | 78.39 | 0.000000 | *** | ✓ |
| PHOCResNet | Cosine, SPOC, SGD | 77.47 | 0.000000 | *** | ✓ |

Table 24: Results for **QbS** on the **Botany Train III** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | BCEL, PHOC, Adam | 98.53 | | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 96.61 | 0.026676 | * | |
| PHOCNet | BCEL, PHOC, Adam | 95.91 | 0.004448 | ** | ✓ |
| PHOCNet | Cosine, SPOC, SGD | 90.51 | 0.000000 | *** | ✓ |
| TPP-PHOCNet | Cosine, SPOC, SGD | 87.78 | 0.000000 | *** | ✓ |
| PHOCResNet | Cosine, SPOC, SGD | 87.61 | 0.000000 | *** | ✓ |

Table 25: Results for **QbE** on the **Konzilsprotokolle Train I** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC, SGD | 94.80 | | | |
| PHOCResNet | BCEL, PHOC, Adam | 90.44 | 0.001180 | ** | ✓ |
| TPP-PHOCNet | Cosine, SPOC, SGD | 88.47 | 0.000008 | *** | ✓ |
| PHOCNet | Cosine, SPOC, SGD | 85.60 | 0.000000 | *** | ✓ |
| TPP-PHOCNet | BCEL, PHOC, Adam | 84.40 | 0.000000 | *** | ✓ |
| PHOCNet | BCEL, PHOC, Adam | 84.00 | 0.000000 | *** | ✓ |

Table 26: Results for **QbS** on the **Konzilsprotokolle Train I** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC, SGD | 90.74 | | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 84.89 | 0.067064 | . | |
| PHOCResNet | BCEL, PHOC, Adam | 84.32 | 0.047216 | * | |
| PHOCNet | Cosine, SPOC, SGD | 79.66 | 0.002460 | ** | ✓ |
| TPP-PHOCNet | BCEL, PHOC, Adam | 77.31 | 0.000136 | *** | ✓ |
| PHOCNet | BCEL, PHOC, Adam | 76.56 | 0.000232 | *** | ✓ |

Table 27: Results for **QbE** on the **Konzilsprotokolle Train II** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | BCEL, PHOC, Adam | 98.16 | | | |
| PHOCResNet | Cosine, SPOC, SGD | 96.96 | 0.080852 | . | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 96.46 | 0.025128 | * | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 95.98 | 0.002948 | ** | ✓ |
| PHOCNet | BCEL, PHOC, Adam | 95.16 | 0.000548 | *** | ✓ |
| PHOCNet | Cosine, SPOC, SGD | 94.85 | 0.000052 | *** | ✓ |

Table 28: Results for **QbS** on the **Konzilsprotokolle Train II** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | Cosine, SPOC, SGD | 97.31 | | | |
| PHOCResNet | BCEL, PHOC, Adam | 97.28 | 0.982264 | | |
| TPP-PHOCNet | BCEL, PHOC, Adam | 95.77 | 0.318436 | | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 95.23 | 0.68208 | | |
| PHOCNet | Cosine, SPOC, SGD | 94.74 | 0.114684 | | |
| PHOCNet | BCEL, PHOC, Adam | 94.11 | 0.051304 | . | |

Table 29: Results for **QbE** on the **Konzilsprotokolle Train III** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | BCEL,PHOC, Adam | 98.10 | | | |
| TPP-PHOCNet | BCEL,PHOC, Adam | 97.58 | 0.427148 | | |
| PHOCNet | BCEL,PHOC, Adam | 96.62 | 0.029036 | * | |
| PHOCResNet | Cosine, SPOC, SGD | 96.21 | 0.007344 | ** | ✓ |
| TPP-PHOCNet | Cosine, SPOC, SGD | 95.51 | 0.000280 | *** | ✓ |
| PHOCNet | Cosine, SPOC, SGD | 95.24 | 0.000080 | *** | ✓ |

Table 30: Results for **QbS** on the **Konzilsprotokolle Train III** database

| Attribute CNN | Configuration | mAP | *p*-value | R Symb. | Significant |
|---|---|---|---|---|---|
| TPP-PHOCNet | BCEL, PHOC, Adam | 97.69 | | | |
| PHOCResNet | BCEL, PHOC, Adam | 97.69 | 0.999908 | | |
| PHOCResNet | Cosine, SPOC, SGD | 96.97 | 0.502012 | | |
| PHOCNet | BCEL, PHOC, Adam | 96.79 | 0.412868 | | |
| PHOCNet | Cosine, SPOC, SGD | 94.97 | 0.064156 | . | |
| TPP-PHOCNet | Cosine, SPOC, SGD | 94.44 | 0.023748 | * | |

The following tables indicate the results of the significance test for the QbO experiments. As all configurations used the PHOC, BCEL and Adam optimization, only the general QbO setup is displayed as configuration, i.e., 1-Net or 2-Net (cf. Sec. 5.1).

Table 31: Results for **QbO** for the **SKW** experiments

| Attribute CNN | Configuration | mAP | $p$-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | 1-Net | 96.60 | | | |
| TPP-PHOCNet | 1-Net | 96.42 | 0.524020 | | |
| PHOCNet | 1-Net | 95.89 | 0.017052 | * | |
| PHOCResNet | 2-Net | 94.76 | 0.000000 | *** | ✓ |
| PHOCNet | 2-Net | 94.68 | 0.000000 | *** | ✓ |
| TPP-PHOCNet | 2-Net | 94.41 | 0.000000 | *** | ✓ |

Table 32: Results for **QbO** for the **SUW** experiments

| Attribute CNN | Configuration | mAP | $p$-value | R Symb. | Significant |
|---|---|---|---|---|---|
| TPP-PHOCNet | 1-Net | 87.22 | | | |
| PHOCNet | 1-Net | 87.14 | 0.892648 | | |
| PHOCResNet | 1-Net | 86.63 | 0.302504 | | |
| PHOCNet | 2-Net | 77.26 | 0.000000 | *** | ✓ |
| TPP-PHOCNet | 2-Net | 77.20 | 0.000000 | *** | ✓ |
| PHOCResNet | 2-Net | 77.03 | 0.000000 | *** | ✓ |

Table 33: Results for **QbO** for the **MUW** experiments

| Attribute CNN | Configuration | mAP | $p$-value | R Symb. | Significant |
|---|---|---|---|---|---|
| PHOCResNet | 2-Net | 86.59 | | | |
| PHOCResNet | 1-Net | 85.02 | 0.000000 | *** | ✓ |
| PHOCNet | 2-Net | 84.96 | 0.000000 | *** | ✓ |
| TPP-PHOCNet | 1-Net | 84.92 | 0.000000 | *** | ✓ |
| PHOCNet | 1-Net | 84.53 | 0.000000 | *** | ✓ |
| TPP-PHOCNet | 2-Net | 84.51 | 0.000000 | *** | ✓ |

# BIBLIOGRAPHY

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Spaces." In: *International Conference on Database Theory.* 2001, pp. 420–434.

[2] Irfan Ahmad, Leonard Rothacker, Gernot A Fink, and Sabri A Mahmoud. "Novel Sub-character HMM Models for Arabic Text Recognition." In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2013, pp. 658–662.

[3] Irfan Ahmad, Gernot A Fink, and Sabri A Mahmoud. "Improvements in Sub-character HMM Model Based Arabic Text Recognition." In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition.* 2014, pp. 537–542.

[4] Ziad Al-Halah and Rainer Stiefelhagen. "How to Transfer? Zero-Shot Object Recognition via Hierarchical Transfer of Semantic Attributes." In: *Proc. of the Winter Conference on Applications of Computer Vision.* 2015, pp. 837–843.

[5] Ziad Al-Halah and Rainer Stiefelhagen. "Automatic Discovery, Association Estimation and Learning of Semantic Attributes for a Thousand Categories." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2017, pp. 614–623.

[6] David Aldavert, Marcal Rusinol, Ricardo Toledo, and Josep Llados. "Integrating Visual and Textual Cues for Query-by-String Word Spotting." In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2013, pp. 511–515.

[7] J. Almazan, A. Fornes, and E. Valveny. "Deformable HOG-Based Shape Descriptor." In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2013, pp. 1022–1026.

[8] Jon Almazán, Albert Gordo, Alicia Fornés, and Ernest Valveny. "Segmentation-free Word Spotting with Exemplar SVMs." In: *Pattern Recognition* 47.12 (2014), pp. 3967–3978.

[9] Jon Almazán, Albert Gordo, Alicia Fornés, and Ernest Valveny. "Word Spotting and Recognition with Embedded Attributes." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.12 (2014), pp. 2552–2566.

[10] Esra Ataer and Pınar Duygulu. "Matching Ottoman Words: An Image Retrieval Approach to Historical Document Indexing." In: *Proc. of the Int. Conf. on Image and Video Retrieval.* 2007, pp. 341–347.

[11] Artem Babenko and Victor Lempitsky. "Aggregating Local Deep Features for Image Retrieval." In: *Proc. of the Int. Conf. on Computer Vision.* 2015, pp. 1269–1277.

[12] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search.* Vol. 82. 2011.

[13] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. "Designing Neural Network Architectures using Reinforcement Learning." In: *Proc. of the Int. Conf. on Learning Representations.* 2017.

[14] Vassileios Balntas, Edward Johns, Lilian Tang, and Krystian Mikolajczyk. *PN-Net: Conjoined Triple Deep Network for Learning Local Image Descriptors*. 2016. arXiv: 1601.05030v1.

[15] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features." In: *Proc. of the European Conference on Computer Vision*. 2006, pp. 404–417.

[16] Y. Bengio. "Learning Deep Architectures for AI." In: *Foundations and Trends in Machine Learning* 2.1 (2009), pp. 1–127.

[17] Yoshua Bengio, Jean-Francois Paiement, Pascal Vincent, Olivier Delalllaux, Nicholas Le Roux, and Marie Ouimet. "Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps and Spectral Clustering." In: *Advances in Neural Information Processing Systems*. 2004, pp. 177–184.

[18] Elia M. Biganzoli, Patrizia Boracchi, Federico Ambrogi, and Ettore Marubini. "Artificial Neural Network for the Joint Modelling of Discrete Cause-Specific Hazards." In: *Artifical Intelligence in Medicine* 37.2 (2006), pp. 119–130.

[19] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2006.

[20] Léon Bottou and Olivier Bousquet. "The Tradeoffs of Large Scale Learning." In: *Advances in Neural Information Processing Systems*. 2007, pp. 161–168.

[21] Leo Breiman. "Random Forests." In: *Machine Learning* 45.5 (2001), pp. 5–32.

[22] John S. Bridle. "Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition." In: *Neurocomputing*. 1990, pp. 227–236.

[23] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. "Signature Verification Using a "Siamese" Time Delay Neural Network." In: *Advances in Neural Information Processing Systems*. 1993, pp. 737–744.

[24] Soravit Changpinyo, Wei-Lun Chao, Boqing Gong, and Fei Sha. "Synthesized Classifiers for Zero-Shot Learning." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. 2016, pp. 5327–5336.

[25] Ken Chatfield, Victor Lempitsky, Andrea Vedaldi, and Andrew Zisserman. "The Devil is in the Details: An Evaluation of Recent Feature Encoding Methods." In: *Proc. of the British Machine Vision Conference*. 2011, pp. 76.1–76.12.

[26] Francine R Chen, Lynn D Wilcox, and Dan S Bloomberg. "Word Spotting in Scanned Images using Hidden Markov Models." In: *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*. 1993.

[27] De Cheng, Yihong Gong, Sanping Zhou, Jinjun Wang, and Nanning Zheng. "Person Re-identification by Multi-Channel Parts-Based CNN with Improved Triplet Loss Function." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. 2016, pp. 1335–1344.

[28] Francois Chollet. "Information-theoretical Label Embeddings for Large-scale Image Classification." In: *arXiv* (2016). arXiv: 1607.05691v1.

[29] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." In: *Proc. of the Int. Conf. on Learning Representations*. 2016.

[30] Corinna Cortes and Vladimir Vapnik. “Support-vector Networks.” In: *Machine Learning* 20.3 (1995), pp. 273–297.

[31] Edmund Edward Fournier D’Albe. “The Type-Reading Optophone.” In: *Nature* 94.2340 (1914), pp. 109–110.

[32] Bin Dai, Shilin Ding, and Grace Wahba. “Multivariate Bernoulli Distribution.” In: *Bernoulli* 19.4 (2013), pp. 1465–1483.

[33] Jeffrey Dalton, James Allan, and Pranav Mirajkar. “Zero-shot video retrieval using content and concepts.” In: *Proc. of the Int. Conf. on Information And Knowledge Management.* 2013, pp. 1857–1860.

[34] Michael J. De Smith. *Statistical Analysis Handbook.* 2015.

[35] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. “Indexing by latent semantic analysis.” In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407.

[36] Sagnik Dhar, Vicente Ordonez, and Tamara L. Berg. “High Level Describable Attributes for Predicting Aesthetics and Interestingness.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2011, pp. 1657–1664.

[37] Pedro Domingos. “A Few Useful Things to Know about Machine Learning.” In: *Communications of the ACM* 55.10 (2012), p. 78.

[38] Matthijs Douze, Arnau Ramisa, and Cordelia Schmid. “Combining Attributes and Fisher Vectors for Efficient Image Retrieval.” In: *Proc. of the IEEE Comp. Soc. Con. on Computer Vision and Pattern Recognition.* 2011, pp. 745–752.

[39] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.” In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159.

[40] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* 2nd. Wiley-Interscience, 2001.

[41] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall/CRC, 1994.

[42] Dumitru Erhan, Aaron Courville, and Pascal Vincent. “Why Does Unsupervised Pre-training Help Deep Learning ?” In: *Journal of Machine Learning Research* 11 (2010), pp. 625–660.

[43] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. “Describing Objects by their Attributes.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2009, pp. 1778–1785.

[44] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. “Convolutional Two-Stream Network Fusion for Video Action Recognition.” In: *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* 2016, pp. 1933–1941.

[45] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. “An Application of Recurrent Neural Networks to Discriminative Keyword Spotting.” In: *Proc. of the Int. Conf. on Artificial Neural Networks.* 2007, pp. 220–229.

[46] Gernot A. Fink. *Markov Models for Pattern Recognition - From Theory to Applications.* 2nd. Springer, 2014.

[47] Volkmar Frinken, Andreas Fischer, R. Manmatha, and Horst Bunke. “A Novel Word Spotting Method Based on Recurrent Neural Networks.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (2012), pp. 211–224.

[48] Kunihiko Fukushima. “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.” In: *Biological Cybernetics* 36.4 (1980), pp. 193–202.

[49] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning.” In: *Proc. of the Int. Conf. on Machine Learning.* 2016.

[50] Suman Ghosh and Ernest Valveny. “R-PHOC: Segmentation-Free Word Spotting using CNN.” In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2017, pp. 801–806.

[51] Angelos P. Giotis, Giorgos Sfikas, Basilis Gatos, and Christophoros Nikou. “A Survey of Document Image Word Spotting Techniques.” In: *Pattern Recognition* 68 (2017), pp. 310–332.

[52] Ross Girshick. “Fast R-CNN.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision.* 2015, pp. 1440–1448.

[53] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.” In: *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.* 2014, pp. 580–587.

[54] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. “Region-Based Convolutional Networks for Accurate Object Detection and Segmentation.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.1 (2016), pp. 142–158.

[55] Xavier Glorot and Yoshua Bengio. “Understanding the Difficulty of Training Deep Feedforward Neural Networks.” In: *Proc. of the Int. Conf. on Artificial Intelligence and Statistics.* Vol. 9. 2010, pp. 249–256.

[56] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks.” In: *Proc. of the Int. Conf. on Artificial Intelligence and Statistics.* Vol. 15. 2011, pp. 315–323.

[57] Shantanu Godbole and Sunita Sarawagi. “Discriminative Methods for Multi-labeled Classification.” In: *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining.* 2004, pp. 22–30.

[58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[59] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. “A Novel Connectionist System for Unconstrained Handwriting Recognition.” In: *Transactions on Pattern Analysis and Machine Intelligence* 31.5 (2009), pp. 855–868.

[60] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. “Synthetic Data for Text Localisation in Natural Images.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2016, pp. 2315–2324.

[61] Neha Gurjar, Sebastian Sudholt, and Gernot A. Fink. "Learning Deep Representations for Word Spotting Under Weak Supervision." In: *Proc. of the Int. Workshop on Document Analysis Systems*. 2018, pp. 7–12.

[62] Isabelle Guyon and Lambert Schomaker. "UNIPEN Project of Online Data Exchange and Recognizer Benchmarks." In: *Proc. of the Int. Conf. on Pattern Recognition*. Vol. 2. 1994, pp. 29–33.

[63] Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. "Digital Selection and Analogue Amplification Coexist in a Cortex-inspired Silicon Circuit." In: *Nature* 405.6789 (2000), pp. 947–951.

[64] Emily M. Hand and Rama Chellappa. "Attributes for Improved Attributes: A Multi-Task Network for Attribute Classification." In: *Proc. of the AAAI Conference on Artificial Intelligence*. 2017, pp. 4068–4074.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." In: *Proc. of the European Conference on Computer Vision*. 2014, pp. 346–361.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *Proc. of the Int. Conf. on Computer Vision*. 2015, pp. 1026–1034.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[68] Sheng He, Petros Samara, Jan Burgers, and Lambert Schomaker. "Historical Document Dating Using Unsupervised Attribute Learning." In: *Proc. of the Int. Workshop on Document Analysis Systems*. 2016, pp. 36–41.

[69] Geoffrey E. Hinton and Simon Osindero. "A Fast Learning Algorithm for Deep Belief Nets." In: *Neural Computation* 18 (2006), p. 2006.

[70] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions." In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06.02 (1998), pp. 107–116.

[71] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer Feedforward Networks are Universal Approximators." In: *Neural Networks* 2.5 (1989), pp. 359–366.

[72] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Deep Features for Text Spotting." In: *Proc. of the European Conference on Computer Vision*. 2014, pp. 512–528.

[73] Dinesh Jayaraman and Kristen Grauman. "Zero-Shot Recognition with Unreliable Attributes." In: *Proc. of the Neural Information Processing Systems Conference*. 2014, pp. 3464–3472.

[74] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell, and U C Berkeley Eecs. "Caffe : Convolutional Architecture for Fast Feature Embedding." In: *Proc. of the ACM Conference on Multimedia*. 2014, pp. 675–678.

[75] Thorsten Joachims. “Training linear SVMs in linear time.” In: *Proc. of the Int. Conf. on Knowledge Discovery and Data Mining*. 2006, pp. 217–226.

[76] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. “DenseCap: Fully Convolutional Localization Networks for Dense Captioning.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. Las Vegas (NV), USA, 2016, pp. 4565–4574.

[77] Daniel Jurafsky and James H. Martin. *Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3rd Editio. Prentice Hall, 2018.

[78] Gopal K Kanji. *100 Statistical Tests*. 3rd. SAGE Publications, 2006.

[79] Yoshiyuki Kawano and Keiji Yanai. “Real-Time Mobile Food Recognition System.” In: *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*. 2013, pp. 1–7.

[80] Patricia Keaton, Hayit Greenspan, and Rodney Goodman. “Keyword Spotting for Cursive Document Retrieval.” In: *Proc. of the Workshop on Document Image Analysis*. 1997, pp. 74–81.

[81] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.” In: *Proc. of the Int. Conf. on Learning Representations*. 2017.

[82] Siamak Khoubyari and Jonathan J. Hull. “Keyword Location in Noisy Document Images.” In: *Proc. of the Symp on Document Analysis and Information Retrieval*. 1993, pp. 217–231.

[83] Diederik P. Kingma and Jimmy Lei Ba. “Adam: A Method for Stochastic Optimization.” In: *Proc. of the Int. Conf. on Learning Representations*. 2015.

[84] Florian Kleber, Stefan Fiel, Markus Diem, and Robert Sablatnig. “CVL-Database: An Off-line Database for Writer Retrieval, Writer Identification and Word Spotting.” In: *Proc. of the Int. Conf. on Document Analysis and Recognition*. 2013, pp. 560–564.

[85] A. Kołcz, J. Alspector, M. Augusteijn, R. Carlson, and G. Viorel Popescu. “A Line-Oriented Approach to Word Spotting in Handwritten Documents.” In: *Pattern Analysis & Applications* 3.2 (2000), pp. 154–168.

[86] Alon Kovalchuk, Lior Wolf, and Nachum Dershowitz. “A Simple and Fast Word Spotting Method.” In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition*. 2014, pp. 3–8.

[87] Adriana Kovashka and Kristen Grauman. “Attribute Pivots for Guiding Relevance Feedback in Image Search.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 297–304.

[88] Adriana Kovashka, Devi Parikh, and Kristen Grauman. “WhittleSearch: Interactive Image Search with Relative Attribute Feedback.” In: *International Journal of Computer Vision* 115.2 (2015), pp. 185–210.

[89] Josip Krapac, Jakob Verbeek, and Frederic Jurie. “Modeling Spatial Layout with Fisher Vectors for Image Categorization Josip.” In: *Proc. of the Int. Conf. on Computer Vision*. 2011, pp. 1487–1494.

[90] Praveen Krishnan and C.V. Jawahar. “Matching Handwritten Document Images.” In: *European Conference on Computer Vision*. 2016, pp. 766–782.

[91] Praveen Krishnan, Kartik Dutta, and C.V. Jawahar. “Deep Feature Embedding for Accurate Recognition and Retrieval of Handwritten Text.” In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition*. 2016, pp. 289–294.

[92] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[93] Praveen Kulkarni, Gaurav Sharma, Joaquin Zepeda, and Louis Chevallier. “Transfer Learning via Attributes for Improved On-the-fly Classification.” In: *Proc. of the IEEE Winter Conference on Applications of Computer Vision*. 2014, pp. 220–226.

[94] Shyh Shiaw Kuo and Oscar E. Agazzi. “Keyword Spotting in Poorly Printed Documents Using Pseudo 2-D Hidden Markov Models.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16.8 (1994), pp. 842–848.

[95] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. “Learning To Detect Unseen Object Classes by Between-Class Attribute Transfer.” In: *Computer Vision and Pattern Recognition*. 2009, pp. 951–958.

[96] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. “Attribute-Based Classification for Zero-Shot Visual Object Categorization.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.3 (2014), pp. 453–465.

[97] Peter William Lane. “Generalized Nonlinear Models.” In: *Proc. in Computational Statistics*. 1996, pp. 331–336.

[98] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. Vol. 2. 2006, pp. 2169–2178.

[99] Yann LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Handwritten Digit Recognition with a Back-Propagation Network.” In: *Advances in Neural Information Processing Systems*. 1990, pp. 396–404.

[100] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient BackProp*. Vol. 53. 9. 1998, pp. 1689–1699.

[101] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient Based Learning Applied to Document Recognition.” In: *Proc. of the IEEE* 86.11 (1998), pp. 2278–2324.

[102] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning.” In: *Nature* 521.7553 (2015), pp. 436–444.

[103] Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. *Gradient Descent Converges to Minimizers*. 2016.

[104] Li Jia Li, Hao Su, Yongwhan Lim, and Li Fei-Fei. “Objects as Attributes for Acene Classification.” In: *Proc. of the European Conference on Computer Vision*. 2012, pp. 57–69.

[105] Longfei Li, Yong Zhao, Dongmei Jiang, Yanning Zhang, Fengna Wang, Isabel Gonzalez, Enescu Valentin, and Hichem Sahli. "Hybrid Deep Neural Network–Hidden Markov Model (DNN-HMM) Based Speech Emotion Recognition." In: *Proc. of the Humaine Association Conf. on Affective Computing and Intelligent Interaction*. 2013, pp. 312–317.

[106] Yang Li, Chunxiao Fan, Yong Li, Qiong Wu, and Yue Ming. "Improving Deep Neural Network with Multiple Parametric Exponential Linear Units." In: *arXiv* (2016). arXiv: 1606.00305.

[107] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network." In: *Proc. of the Int. Conf. on Learning Representations*. 2014.

[108] Jingen Liu, Benjamin Kuipers, and Silvio Savarese. "Recognizing Human Actions by Attributes." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. 2011, pp. 3337–3344.

[109] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Deep Learning Face Attributes in the Wild." In: *Proc. of the Int. Conf. on Computer Vision*. 2015, pp. 3730–3738. ISBN: 9781467383912. arXiv: 1411.7766.

[110] Marcus Liwicki and Horst Bunke. "IAM-OnDB - An on-line English Sentence Database Acquired from Handwritten Text on a Whiteboard." In: *Proc. of the Int. Conf. on Document Analysis and Recognition*. 2005, pp. 956–961.

[111] Stuart P. Lloyd. "Least Squares Quantization in PCM." In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[112] David G. Lowe. "Distinctive Image Features From Scale-invariant Keypoints." In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110.

[113] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models." In: *Proc. of the Int. Conf. on Machine Learning*. 2013.

[114] James B. MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations." In: *Proc. Berkeley Symposium on Mathematical Statistics and Probability*. Vol. 1. 1967, pp. 281–297.

[115] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. "An Extensive Experimental Comparison of Methods for Multi-label Learning." In: *Pattern Recognition* 45.9 (2012), pp. 3084–3104.

[116] R Manmatha, Chengfeng Han, and E.M. Riseman. "Word Spotting: A New Approach to Indexing Handwriting." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. 1996, pp. 1–29.

[117] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2009.

[118] Kanti V. Mardia. "Characterization of Directional Distributions." In: *Statistical Distributions in Scientific Work*. 1975, pp. 365–385.

[119] U. V. Marti and Horst Bunke. "The IAM-database: An English Sentence Database for Offline Handwriting Recognition." In: *International Journal on Document Analysis and Recognition* 5.1 (2002), pp. 39–46.

[120] Domingo Mery and Kevin Bowyer. “Automatic Facial Attribute Analysis via Adaptive Sparse Representation of Random Patches.” In: *Pattern Recognition Letters* 68.2 (2015), pp. 260–269.

[121] Tomas Mikolov, Greg Corrado, Kai Chen, and Jeffrey Dean. “Efficient Estimation of Word Representations in Vector Space.” In: *Proc. of the Int. Conf. on Learning Representations* (2013).

[122] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Vol. 165. 1969, p. 258.

[123] Meinard Müller. “Dynamic Time Warping.” In: *Information Retrieval for Music and Motion*. Springer, 2007. Chap. 4, pp. 69–84.

[124] Kevin P. Murphy. *Machine Learning*. MIT Press, 2012.

[125] Yoshito Nagaoka, Tomo Miyazaki, Yoshihiro Sugaya, and Shinichiro Omachi. “Text Detection by Faster R-CNN with Multiple Region Proposal Networks.” In: *Proc. of the Int. Conf. on Document Analysis and Recognition*. 2017, pp. 15–20.

[126] Vinod Nair and Geoffrey E Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines.” In: *Proc. of the Int. Conf. on Machine Learning*. 2010, pp. 807–814.

[127] Bai Li Nguyen, Hieu V. “Cosine Similarity Metric Learning for Face Verification.” In: *Asian Conference on Computer Vision* (2010), pp. 709–720.

[128] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[129] Stephen O’Hara and Bruce A. Draper. *Introduction to the Bag of Features Paradigm for Image Classification and Retrieval*. 2011.

[130] Markus Ojala and Gemma C. Garriga. “Permutation Tests for Studying Classifier Performance.” In: *Journal of Machine Learning Research* 11 (2010), pp. 1833–1863.

[131] Ioannis Panageas and Georgios Piliouras. *Gradient Descent Converges to Minimizers: The Case of Non-Isolated Critical Points*. 2016. arXiv: 1605.00405.

[132] Thierry Paquet, Laurent Heutte, Guillaume Koch, and Clément Chatelain. “A Categorization System for Handwritten Documents.” In: *International Journal on Document Analysis and Recognition* 15.4 (2012), pp. 315–330.

[133] Devi Parikh and Kristen Grauman. “Interactively Building a Discriminative Vocabulary of Nameable Attributes.” In: *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*. 2011, pp. 1681–1688.

[134] Devi Parikh and Kristen Grauman. “Relative Attributes.” In: *Proc. of the Int. Conf. on Computer Vision*. 2011, pp. 503–510.

[135] Genevieve Patterson and James Hays. “COCO Attributes: Attributes for People, Animals, and Objects.” In: *Proc. of the European Conference on Computer Vision*. 2016, pp. 85–100.

[136] Genevieve Patterson, Chen Xu, Hang Su, and James Hays. “The SUN Attribute Database: Beyond Categories for Deeper Scene Understanding.” In: *International Journal of Computer Vision* 108.1-2 (2014), pp. 59–81.

[137] Mario Pechwitz, Ss Maddouri, and Volker Märgner. “IFN/ENIT-database of handwritten Arabic words.” In: *Colloque International Francophone sur l’Ecrit et le Document* (2002), pp. 1–8.

[138] Jean Ponce and David A. Forsyth. *Computer Vision: A Modern Approach.* 2012.

[139] Ingmar Posner, Peter Corke, and Paul Newman. “Using Text-Spotting to Query the World.” In: *Proc. of the Int. Conf. on Intelligent Robots and Systems.* 2010, pp. 3181–3186.

[140] Arik Poznanski and Lior Wolf. “CNN-N-Gram for Handwriting Word Recognition.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2016, pp. 2305–2314.

[141] Ioannis Pratikakis, Konstantinos Zagoris, Basilis Gatos, Joan Puigcerver, Alejandro H. Toselli, and Enrique Vidal. “ICFHR2016 Handwritten Keyword Spotting Competition (H-KWS 2016).” In: *International Conference on Frontiers in Handwriting Recognition.* Shenzhen, China, 2016, pp. 613–618.

[142] Toni M. Rath and R. Manmatha. “Word Image Matching Using Dynamic Time Warping.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* Vol. 2. 2003, pp. II–521–II–527.

[143] Tony M. Rath and R. Manmatha. “Word Spotting for Historical Documents.” In: *International Journal on Document Analysis and Recognition* 9 (2007), pp. 139–152.

[144] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. “Classifier Chains for Multi-label Classification.” In: *Machine Learning* 85.3 (2011), pp. 333–359.

[145] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” In: *Advances in Neural Information Processing Systems* (2015), pp. 1–10.

[146] George Retsinas, Giorgos Sfikas, and Basilis Gatos. “Transferable Deep Features for Keyword Spotting.” In: *Proc. of the European Signal Processing Conference.* 2017.

[147] George Retsinas, Giorgos Sfikas, Nikolaos Stamatopoulos, Georgios Louloudis, and Basilis Gatos. “Exploring Critical Aspects of CNN-based Keyword Spotting. A PHOCNet Study.” In: *Proc. of the Int. Workshop on Document Analysis Systems.* 2018, pp. 13–18.

[148] José A. Rodríguez-Serrano and Florent Perronnin. “Handwritten Word-Spotting Using Hidden Markov Models and Universal Vocabularies.” In: *Pattern Recognition* 42.9 (2009), pp. 2106–2116.

[149] José A. Rodríguez-Serrano and Florent Perronnin. “A Model-Based Sequence Similarity with Application to Handwritten Word Spotting.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2108–2120.

[150] Jose A. Rodriguez-Serrano, Albert Gordo, and Florent Perronnin. “Label Embedding: A Frugal Baseline for Text Recognition.” In: *International Journal of Computer Vision* 113.3 (2015), pp. 193–207. ISSN: 15731405.

[151] José A. Rodríguez and Florent Perronnin. “Local Gradient Histogram Features for Word Spotting in Unconstrained Handwritten Documents.” In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition.* 2008, pp. 7–12.

[152] Robin Rohlicek, William Russel, Salim Roukos, and Herbert Gish. "Continous Hidden Markov Modelling for Speaker-independent Word Spotting." In: *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing.* 1989, pp. 627–630.

[153] Marcus Rohrbach, Michael Stark, and Bernt Schiele. "Evaluating Knowledge Transfer and Zero-Shot Learning in a Large-Scale Setting." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* 2011, pp. 1641–1648.

[154] Verónica Romero, Alicia Fornés, Nicolás Serrano, Joan Andreu Sánchez, Alejandro H. Toselli, Volkmar Frinken, Enrique Vidal, and Josep Lladós. "The ESPOSALLES database: An ancient marriage license corpus for off-line handwriting recognition." In: *Pattern Recognition* 46.6 (2013), pp. 1658–1669.

[155] Frank Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." In: *Psychological Review* 65.6 (1958), pp. 386–408.

[156] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, 1962.

[157] Leonard Rothacker and Gernot A. Fink. "Segmentation-free Query-by-String Word Spotting with Bag-of-Features HMMs." In: *Int. Conf. on Document Analysis and Recognition.* 2015, pp. 661–665.

[158] Leonard Rothacker, Szilard Vajda, and Gernot A. Fink. "Bag-of-Features Representations for Offline Handwriting Recognition Applied to Arabic Script." In: *International Conference on Frontiers in Handwriting Recognition.* 2012.

[159] Leonard Rothacker, Marcal Rusinol, and Gernot A. Fink. "Bag-of-Features HMMs for Segmentation-Free Word Spotting in Handwritten Documents." In: *Int. Conf. on Document Analysis and Recognition.* 2013, pp. 1305–1309.

[160] Leonard Rothacker, Sebastian Sudholt, Eugen Rusakov, Matthias Kasperidus, and Gernot A. Fink. "Word Hypotheses for Segmentation-free Word Spotting in Historic Document Images." In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2017.

[161] Ethan M. Rudd, Manuel Günther, and Terrance E. Boult. "MOON: A Mixed Objective Optimization Network for the Recognition of Facial Attributes." In: *Proc. of the European Conference on Computer Vision.* 2016, pp. 19–35.

[162] David E. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation." In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Vol. 1. MIT Press, 1986, pp. 318–362.

[163] Eugen Rusakov, Leonard Rothacker, Hyunho Mo, and Gernot A Fink. "A Probabilistic Retrieval Model for Word Spotting based on Direct Attribute Prediction." In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition.* 2018.

[164] Marçal Rusiñol, David Aldavert, Ricardo Toledo, and Josep Lladós. "Browsing Heterogeneous Document Collections by a Segmentation-Free Word Spotting Method." In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2011, pp. 63–67.

[165] Marçal Rusiñol, David Aldavert, Ricardo Toledo, and Josep Lladós. "Efficient segmentation-free keyword spotting in historical document collections." In: *Pattern Recognition* 48.2 (2015), pp. 545–555.

[166] Marçal Rusiñol, David Aldavert, Ricardo Toledo, and Josep Lladós. “Towards Query-by-Speech Handwritten Keyword Spotting.” In: *Proc. of the Int. Conf. on Document Image Analysis.* 2015, pp. 501–505.

[167] Olga Russakovsky *et al.* “ImageNet Large Scale Visual Recognition Challenge.” In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[168] Ruslan Salakhutdinov and Geoffrey Hinton. “Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure.” In: *Proc. of the Int. Conf. on Artificial Intelligence and Statistics.* 2007, pp. 412–419.

[169] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. “Image Classification with the Fisher Vector: Theory and Practice.” In: *International Journal of Computer Vision* 105.3 (2013), pp. 222–245.

[170] Jürgen Schmidhuber. “Deep Learning in Neural Networks: An Overview.” In: *Neural Networks* 61 (2015), pp. 85–117.

[171] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.” In: *Proc. of the Int. Conf. on Computer Vision.* 2017, pp. 618–626.

[172] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks.” In: *Proc. of the Int. Conf. on Learning Representations.* 2014.

[173] Giorgos Sfikas, George Retsinas, and Basilis Gatos. “Zoning Aggregated Hypercolumns for Keyword Spotting.” In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition.* 2016, pp. 283–288.

[174] Cosma Rohilla Shalizi. *Advanced Data Analysis from an Elementary Point of View.* Cambridge University Press, 2013.

[175] Sukrit Shankar, Vikas K. Garg, and Roberto Cipolla. “DEEP-CARVING: Discovering Visual Attributes by Carving Deep Neural Nets.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2015, pp. 3403–3412.

[176] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2014, pp. 806–813.

[177] Arjun Sharma and Sankar K. Pramod. “Adapting Off-the-Shelf CNNs for Word Spotting & Recognition.” In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2015, pp. 986–990.

[178] Adi Silberpfennig, Lior Wolf, Nachum Dershowitz, Seraogi Bhagesh, and Bidyut B. Chaudhuri. “Improving OCR for an Under-Resourced Script Using Unsupervised Word-Spotting.” In: *Proc. of the Int. Conf. on Document Analysis and Recognition.* 2015, pp. 706–710.

[179] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” In: *Proc. of the Int. Conf. on Learning Representations.* 2015.

[180] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.” In: *Proc. of the Int. Conf. on Learning Representations*. 2014.

[181] Josef Sivic and Andrew Zisserman. “Efficient Visual Search of Videos Cast as Text Retrieval.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.4 (2009), pp. 591–606.

[182] Mark D. Smucker, James Allan, and Ben Carterette. “A Comparison of Statistical Significance Tests for Information Retrieval Evaluation.” In: *Conference on Information and Knowledge Management*. Lisbon, Portugal, 2007, pp. 623–632. ISBN: 9781595938039.

[183] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. “Striving for Simplicity: The All Convolutional Net.” In: *Proc. of the Int. Conf. on Learning Representations*. 2015.

[184] Suvrit Sra. “Directional Statistics in Machine Learning: A Brief Review.” In: *Modern Statistical Methods for Directional Data*. Ed. by Christoph Ley and Thomas Verdebout. CRC Press, 2016. Chap. 1.

[185] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[186] T. Starner, J. Makhoul, R. Schwartz, and G. Chou. “On-Line Cursive Handwriting Recognition Using Speech Recognition Methods.” In: *Proceedings of ICASSP ’94. IEEE International Conference on Acoustics, Speech and Signal Processing*. 1994, pp. 125–128.

[187] Sebastian Sudholt and Gernot A Fink. “A Modified Isomap Approach to Manifold Learning in Word Spotting.” In: *Proc. of the German Conference on Pattern Recognition*. 2015, pp. 529–539.

[188] Sebastian Sudholt and Gernot A Fink. “PHOCNet : A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents.” In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition*. 2016, pp. 277–282.

[189] Sebastian Sudholt and Gernot A. Fink. “Evaluating Word String Embeddings and Loss Functions for CNN-based Word Spotting.” In: *Proc. of the Int. Conf. on Document Analysis and Recognition*. 2017, pp. 493–498.

[190] Sebastian Sudholt and Gernot A. Fink. “Attribute CNNs for Word Spotting in Handwritten Documents.” In: *International Journal on Document Analysis and Recognition* 21.3 (2018), pp. 199–218.

[191] Sebastian Sudholt, Leonard Rothacker, and Gernot A. Fink. “Query-by-Online Word Spotting Revisited: Using CNNs for Cross-Domain Retrieval.” In: *Proc. of the Int. Conf. on Document Analysis and Recognition*. 2017, pp. 481–486.

[192] Guang-Lu Sun, Xiao Wu, Hong-Han Chen, and Qiang Peng. “Clothing Style Recognition using Fashion Attribute Detection.” In: *Proc. of the Int. Conf. on Mobile Multimedia Communications*. 2015.

[193] Christian Szegedy *et al.* “Going Deeper with Convolutions.” In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. 2014, pp. 1–9.

[194] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. "Rethinking the Inception Architecture for Computer Vision." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[195] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 5th. Springer, 2010.

[196] Xian Tang. "Hybrid Hidden Markov Model and Artificial Neural Network for Automatic Speech Recognition." In: *Proc. of the Pacific-Asia Conf. on Circuits, Communications and Systems*. 2009, pp. 682–685.

[197] Kengo Terasawa and Yuzuru Tanaka. "Slit Style HOG Feature for Document Image Word Spotting." In: *Proc. of the Int. Conf. on Document Analysis and Recognition*. 2009, pp. 116–120.

[198] Simon Thomas, Clément Chatelain, Laurent Heutte, Thierry Paquet, and Yousri Kessentini. "A Deep HMM Model for Multiple Keywords Spotting in Handwritten Documents." In: *Pattern Analysis and Applications* 18.4 (2015), pp. 1003–1015.

[199] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5 - RMSprop: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural Networks for Machine Learning* (2012).

[200] Alejandro Héctor Toselli, Enrique Vidal, Verónica Romero, and Volkmar Frinken. "HMM Word Graph Based Keyword Spotting in Handwritten Document Images." In: *Information Sciences* 370-371.20 (2016), pp. 497–518.

[201] Andreas Veit, Maximilian Nickel, Serge Belongie, and Laurens van der Maaten. *Separating Self-Expression and Visual Content in Hashtag Supervision*. 2017. arXiv: 1711.09825.

[202] Paul Werbos. "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences." PhD thesis. Harvard University, 1974.

[203] Jason Weston, Samy Bengio, and Nicolas Usunier. "Large Scale Image Annotation: Learning to Rank with Joint Word-Image Embeddings." In: *Machine Learning* 81.1 (2010), pp. 21–35.

[204] Christian Wieprecht, Leonard Rothacker, and Gernot A. Fink. "Word Spotting in Historical Document Collections with Online-Handwritten Queries." In: *Proc. of the Int. Workshop on Document Analysis Systems*. 2016.

[205] Thomas Wilkinson and Anders Brun. "A Novel Word Segmentation Method Based on Object Detection and Deep Learning." In: *Advances in Visual Computing*. 2015, pp. 231–240.

[206] Tomas Wilkinson and Anders Brun. "Semantic and Verbatim Word Spotting using Deep Neural Networks." In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition*. 2016, pp. 307–312.

[207] Tomas Wilkinson, Jonas Lindström, and Anders Brun. "Neural Ctrl-F: Segmentation-free Query-by-String Word Spotting in Handwritten Manuscript Collections." In: *Proc. of the Int. Conf. on Computer Vision*. 2017, pp. 4433–4442.

[208] Martin Wöllmer, Florian Eyben, Joseph Keshet, Alex Graves, and Gerhard Rigoll. "Robust Discriminative Keyword Spotting for Emotionally Colored Spontaneous Speech Using Bidirectional LSTM Networks." In: *Proc. of the Int. Conf. on Acoustics, Speech and Signal Processing.* 2009, pp. 3949–3952.

[209] David H. Wolpert. "The Lack of A Priori Distinctions Between Learning Algorithms." In: *Neural Computation* 8.7 (1996), pp. 1341–1390.

[210] David H. Wolpert and William G. Macready. "No free lunch theorems for optimization." In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82.

[211] Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas Guibas, and Li Fei-Fei. "Human Action Recognition by Learning Bases of Action Attributes and Parts." In: *Proc. of the Int. Conference on Computer Vision.* 2011, pp. 1331–1338.

[212] Matthew D. Zeiler and Rob Fergus. "Stochastic Pooling for Regularization of Deep Convolutional Neural Networks." In: *Proc. of the Int. Conf. on Learning Representations.* 2013.

[213] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks." In: *Proc. of the European Conference on Computer Vision.* 2014, pp. 818–833.

[214] Ning Zhang, Manohar Paluri, Marc'Aurelio Ranzato, Trevor Darrell, and Lubomir Bourdev. "PANDA: Pose Aligned Networks for Deep Attribute Modeling." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2014, pp. 1637–1644.

[215] Yang Zhong, Josephine Sullivan, and Haibo Li. "Face Attribute Prediction Using Off-the-Shelf CNN Features." In: *Proc. of the Int. Conf. on Biometrics.* 2016, pp. 1–7.

[216] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. "Learning Deep Features for Scene Recognition Using Places Database." In: *Advances in Neural Information Processing Systems.* 2014, pp. 487–495.

[217] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. "Learning Deep Features for Discriminative Localization." In: *Proc. of the IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition.* 2016, pp. 2921–2929.

[218] Barret Zoph and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning." In: *Proc. of the Int. Conf. on Learning Representations.* 2017.

# ACRONYMS

| | |
|---|---|
| Adam | Adaptive Moment Estimation |
| AP | Average Precision |
| BCEL | Binary Cross Entropy Loss |
| BLSTM | Bidirectional Long Short-Term Memory Network |
| BN | Batch Normalization |
| BoF | Bag of Features |
| Botany | Botany in Britsh India |
| CAM | Class Activation Maps |
| CBIR | Content-based Image Retrieval |
| CCA | Canonical Correlation Analysis |
| CNN | Convolutional Neural Network |
| CTC | Connectionist Temporal Classification |
| DAP | Direct Attribute Prediction |
| DCToW | Discrete Cosine Transform of Words |
| DTP | Dilated Text Proposals |
| DTW | Dynamic Time Warping |
| ELU | Exponential Linear Unit |
| GLM | Generalized Linear Model |
| GMM | Gaussian Mixture Model |
| GNLM | Generalized Nonlinear Model |
| GPU | Graphical Processing Unit |
| GW | George Washington Database |
| GWO | George Washington Online Database |
| HMM | Hidden Markov Model |
| HOG | Histogram of Oriented Gradients |
| i.i.d. | independent and identically distributed |
| IAM-DB | IAM Handwriting Database |
| IAM-OnDB | IAM On-Line Handwriting Database |

| | |
|---|---|
| IAP | Indirect Attribute Prediction |
| IFN/ENIT | IFN/ENIT Database |
| ILSVRC | ImageNet Large Scale Visual Recognition Challenge |
| KCCA | Kernelized Canonical Correlation Analysis |
| kNN | $k$-Nearest Neighbor classifier |
| Konzilsprotokolle | Alvermann Konzilsprotokolle |
| LGH | Local Gradient Histogram |
| LReLU | Leaky Rectified Linear Unit |
| LSI | Latent Semantic Indexing |
| mAP | mean Average Precision |
| MDS | Multi Dimensional Scaling |
| MLE | Maximum Likelihood Estimation |
| MLP | Multilayer Perceptron |
| OCR | Optical Character Recognition |
| PCA | Principal Component Analysis |
| PDF | Probability Density Function |
| PHOC | Pyramidal Histogram of Characters |
| PMF | Probability Mass Function |
| PMI | Pointwise Mutual Information |
| PPMI | Positive Pairwise Mutual Information |
| PReLU | Parametric Rectified Linear Unit |
| QbE | Query-by-Example |
| QbO | Query-by-Online-Trajectory |
| QbS | Query-by-String |
| QbSP | Query-by-Speech |
| ReLU | Rectified Linear Unit |
| ResNet | Residual Network |
| RNN | Recurrent Neural Network |
| RPN | Region Proposal Network |
| SGD | Stochastic Gradient Descent |
| SIFT | Scale Invariant Feature Transform |

| | |
|---|---|
| SPOC | Spatial Pyramid of Characters |
| SPP | Spatial Pyramid Pooling |
| SURF | Speeded Up Robust Features |
| SVD | Singular Value Decomposition |
| SVM | Support Vector Machine |
| TPP | Temporal Pyramid Pooling |

# INDEX

# ACKNOWLEDGMENTS

When I think back four years to the start of my PhD studies, I could not have imagined the path that my studies eventually took me which ultimately culminated in this thesis. I consider myself fortunate that this path was lined with people who enjoyed the good parts of my journey and were there to support me during the harder phases. The final words of my thesis are dedicated to you.

First and foremost, I want to thank my supervisor Prof. Dr.-Ing. Gernot A. Fink. Thank you, Gernot, for always taking time and sharing your knowledge when I had a question. You were always patient with me when I wanted to discuss my ideas and always challenged me to get the best out of me.

I would also like to thank my second reviewer Prof. Dr. Lambert Schomaker. Not only did you immediately agree to review my thesis, you were one of the first persons outside of our group which I discussed my ideas for the PHOCNet with.

My sincere thanks also goes to Prof. Dr. Johannes Fischer for his willingness to be my mentor in the structured PhD program.

Besides my advisors and mentor, I was surrounded by a great group of colleagues during my PhD studies. From this group, a special thank you goes to my "roommate" Leonard Rothacker. You have accompanied my way ever since my master program and showed me the ropes in my first years as a PhD student. I never feared any presentation or discussion because I was able to run everything by you first. You wouldn't let me get away with anything and always pushed me to make my ideas and arguments bullet proof. Besides Leonard, I want to thank René Grzeszick, Axel Plinge, Fernando Moya and Eugen Rusakov for countless short-lived and fun hours at the lab.

Finally, I want to thank my family. My parents Walter and Bettina Sudholt always supported me in pursuing the education and degree I wanted and I cannot thank you enough for this. It means the world to me to see how proud you are of me.

My biggest thank you goes to my wife Hanna and my son Theo. You made the greatest of sacrifices letting me spend long hours at the lab and touring the world for my PhD program. Without you, I could have not endured everything that comes with being a PhD student. You make me feel like I'm the smartest person in the world. I love both of you.