technische universität
dortmund

The Generation of Synthetic Data for CNN-Based Word Spotting

Bachelor Thesis

Jin Ke

September 18, 2020

Supervisors:
Fabian Wolf, M.Sc.
Prof. Dr.-Ing. Gernot A. Fink

Fakultät für Informatik
Technische Universität Dortmund
http://www.cs.uni-dortmund.de

CONTENTS

1 MOTIVATION

Historical handwritten documents contain valuable information that can be used in some modern research and projects. Document Image Analysis and Recognition is dedicated to preserving these documents and extracting valuable information from them. Optical Character Recognition (OCR) uses technology to convert characters or words in handwritten or printed text images into electronic representations that can be used for data processing. OCR has been widely used in pattern recognition, artificial intelligence, and computer vision to input data from the text images. These data can then be electronically edited, searched, stored, or employed in machine processes such as cognitive computing.

Currently, many document OCR systems can work almost error-free in printed and high-quality scanned documents. However, they have faced a bottleneck in handwriting, especially in historical handwritten documents and manuscripts, because the word appearing in the handwritten text images will be affected by many factors such as font, direction, and image distortion. The scribbled handwriting, the overlap of letters, and the tight connections between words in the handwritten text significantly increase the experiment's difficulty. Therefore, handwritten text recognition is still a challenging but wildly useful task.

As an alternative solution to OCR, word spotting is receiving more and more attention from researchers. It does not require a full transcription of the text and is a retrieval-based approach to locate a specific word a given document collection. This method compares each part of the document images with the required query and determines whether they are similar to the query. The query can be a text string (Query-By-String) or a word image (Query-By-Example). The result is a list of document image parts and is usually sorted in descending order according to the determined similarity. People are increasingly inclined to use this approach to solve document indexing problems, which has led to a surge in word spotting methods today [GSGN17] [NAK+17].

In recent years, deep learning-based word spotting has managed to get remarkable achievements [SZ15] [JSVZ14], but it also exposed their desire for a wide range of training data sets. In pattern recognition problems, the model's performance for word spotting depends not only on the features and classification algorithms but also on the quality and scale of the training data. With word spotting becoming a significant field of study in the handwritten domain, researchers could no longer ignore the severe problem in document image analysis: the deep CNN-based word spotting methods rely heavily on numerous high-quality labeled samples.

The experiments in [JSVZ14] and [PW16] show that the larger the training dataset, the better the performance of the CNNs. However, it takes much effort to create a training dataset, since data collection to annotation and verification require many manual works. A solution to this is to create synthetic data automatically. Synthetic handwritten word images with multiple sizes and distortions meet our requirements for training data combining more variability and realism, and also provide endless possibilities for word spotting [GSF18] [WBF20].

The generation of synthetic word images is the problem of interest in this thesis. In particular, we will use open-source fonts to generate synthetic data that is highly realistic and sufficient to replace actual data for CNN-based word spotting systems. Following suggestions of Krishnan *et al.* [KJ16b], we complete word image generation through font rendering and apply some distortions to the generated images. Based on the method of Krishnan *et al.*, we also use elastic deformation to reduce the gap between synthetic samples and the real samples. This method's advantage is that it does not need any real data and can provide a large amount of training data to reduce manually annotated data significantly.

For evaluation purposes, we consider an existing CNN-based word spotting model, the PHOCNet. This deep neural network model is trained solely on data produced by our synthetic text generation engine. Finally, all trained models are evaluated in the segmentation-based case. By comparing these performance results, we want to know the contribution of each stage of synthetic data generation to word spotting accuracy. Besides, we are interested in how the vocabulary size and the number of fonts used will affect the results.

We first give an overview of word spotting methods and the deep neural network in Chap. 2. An overview of the synthetic handwritten word follows and a popular model for word spotting, the PHOCNet, in terms of the technology described in Chap. 3. Our synthetic data generation system is then presented in Chap. 4, and evaluated in Chap. 5, finally conclusions are drawn in Chap. 6.

2 FUNDAMENTALS

The application of neural networks and deep learning has given a significant performance improvement in word spotting. Many word spotting methods, such as [WB16] [KDJ16], use neural networks to learn various feature embeddings. Convolutional neural networks (CNNs) are used in word spotting and their success is well documented [SF16].

This chapter presents the fundamentals of interest in this thesis. Sec. 2.1 introduces the concept of word spotting. And then, Sec. 2.2 gives a detailed overview of the neural network. Sec. 2.2.1 outlines then the basics of the neural networks used in this thesis. Afterward, Sec. 2.2.2 elaborates on how to train neural networks. Finally, Sec. 2.2.3 introduces CNNs that are the cornerstone of the model we used in this work.

2.1 WORD SPOTTING

With the drastic growth of demand for processing and accessing handwritten documents in digital form, word spotting has aroused many interests of researchers from the document image research field. Especially in historical document images, word spotting techniques are widely regarded as the best alternative to the OCR-based techniques, because these documents usually have low visualization quality, diverse writing styles, deformed fonts, and uneven text spacing [GSGN17].

The purpose of word spotting is to retrieve related word images from a given document collection under a given query. In [GSGN17], many different approaches for word spotting are introduced. For these methods, the query term may be a text string (Query-by-String), it may also be an image (Query-by-Example) or rely on other query methods [RATL15] [WRF16]. Query-by-String (QbS) and Query-by-Example (QbE) are the query methods used in this thesis.

Besides, the word spotting method's operation also depends on whether the word images need to be segmented from the page. The approaches depending on segmentation are called the segmentation-based [SF18] [KJ19], and all other methods can be collectively referred to as segmentation-free [DM20] [TP19].

According to the requirement of training data, the word spotting methods can also be distinguished into annotation-free [WF20] and annotation-based [SF18] [KJ19]. If a word spotting method does not rely on any manually labeled samples, it is annotation-free and annotation-based otherwise.

As the word spotting research has received more and more attention, many different models have been employed to the word spotting problem. Nowadays, neural network-based models for word spotting become popular, and they have shown excellent performance on most traditional data sets [SF16] [WB16] [KDJ16]. However, these methods rely heavily on annotated training data. By using a synthetic data set to pre-train a neural network, the experiment in [GSF18] significantly reduces the requirement for manually created annotations while ensuring performance. The word spotting method in [WF20] does not require any manually annotated data.

The word spotting process can be roughly divided into preprocessing, feature extraction, and matching [GSGN17]. The preprocessing stage's main task is to remove the noise from input images to enhance image quality. In the feature extraction stage, a set of meaningful features are extracted for each word image, and they will be used to calculate the similarity between words in the subsequent matching process. The general feature extraction method is to embed a word image's visual appearance in a feature vector. And then, a distance measure can be used to compare this representation with other document areas. The word spotting's heart process matches the word query image and the set of word images stored in the database using the extracted features. Most word spotting-based techniques build indexes with features extracted from all word images and extract the same features from the given query image. They then measure and record how similar the query image is to each word image in the database. Finally, the word images will be sorted according to their similarity to the query and returned as a list [GSGN17].

2.2 NEURAL NETWORKS

After explaining the concept of word spotting in the previous section, this section introduces neural networks in detail. These machine learning models play a vital role in this work. First, concepts for feedforward neural networks are displayed in Sec. 2.2.1 as they represent the fundamentals of neural networks used in this work. Then, Sec. 2.2.2 introduces how to train feedforward architectures. Finally, Sec. 2.2.3 details the Convolutional Neural Networks (CNNs).

2.2.1 *Feedforward Neural Networks*

Feedforward neural networks are artificial neural networks made up of connected nodes that form a directed and acyclic graph. Many concepts developed in early neural network models are still used in today's deep learning architectures. The two representatives of them are Perceptrons and Multi-Layer Perceptrons (MLP). This section first discusses perceptrons and then introduces multilayer perceptrons.

Perceptron

The perceptron proposed by Frank Rosenblatt [Ros58] is regarded as the simplest form of a feedforward neural network. In artificial neural networks research, the perceptron is also referred to as a single-layer artificial neural network to distinguish it from the more complex multilayer perceptron. Despite its simple structure, the (single-layer) perceptron can learn and solve a variety of problems.

The graphical visualization of the perceptron is in Fig. 2.2.1. The perceptron maps its n-dimensional input vectors $\mathbf{x} \in \mathbb{R}^n$ to the output $f(\mathbf{x}) \in \{0, 1\}$ [Ros58] [Nie19]. The classification rule for the perceptron can be defined as

$$f(\mathbf{x}) = \begin{cases} 1 & \text{, if } \mathbf{w}^T\mathbf{x} + b > 0 \\ 0 & \text{, otherwise} \end{cases} \tag{2.2.1}$$

where $\mathbf{w}$ is the vector of real-valued weights, $\mathbf{w}^T\mathbf{x}$ is the dot product $\sum_{i=1}^{n} w_i x_i$, where n is the number of inputs to the perceptron, and b is the bias which shifts the decision boundary away from the origin and does not depend on any input value. Bias can be regarded as a measure of how difficult the perceptrons output a "1". For perceptrons with large bias, it is easy for the perceptron to output a "1" and vice versa [Nie19]. The step function f can divide the input space into two parts, which is the case of binary classification.

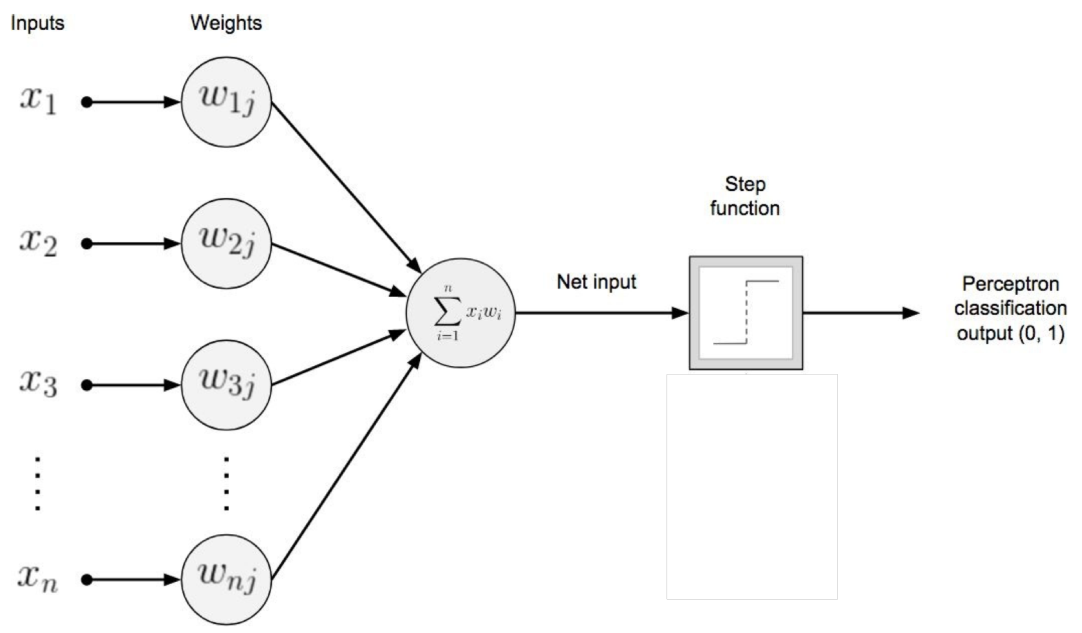


Figure 2.2.1: Visualization of Perceptron: In this process, the inputs are first multiplied by their associated weights representing their importance to the output. These calculated values are then added to create a weighted sum, which will be sent to a step function to produce the output (0 or a 1) of the perceptron (image taken from [PG17]).

If the training samples are linearly separable, a single-layer perceptron can work as a linear classifier. However, the perceptron's main flaw is that it cannot deal with the linear inseparability problem [MP87]. Multi-layer perceptrons have more powerful processing capabilities than the perceptrons with one layer, and they are sufficient to solve many problems that the perceptrons cannot handle.

Multi-Layer Perceptron

A multi-layer perceptron (MLP) is formed by stacking many perceptrons in sequence and can model complex non-linear relationships. As shown in Fig. 2.2.2, MLP can be regarded as a directed graph composed of multiple node layers, each of which is fully connected to the next layer. The basic structure of multi-layer perception consists of three parts: the first input layer, the middle hidden layers,

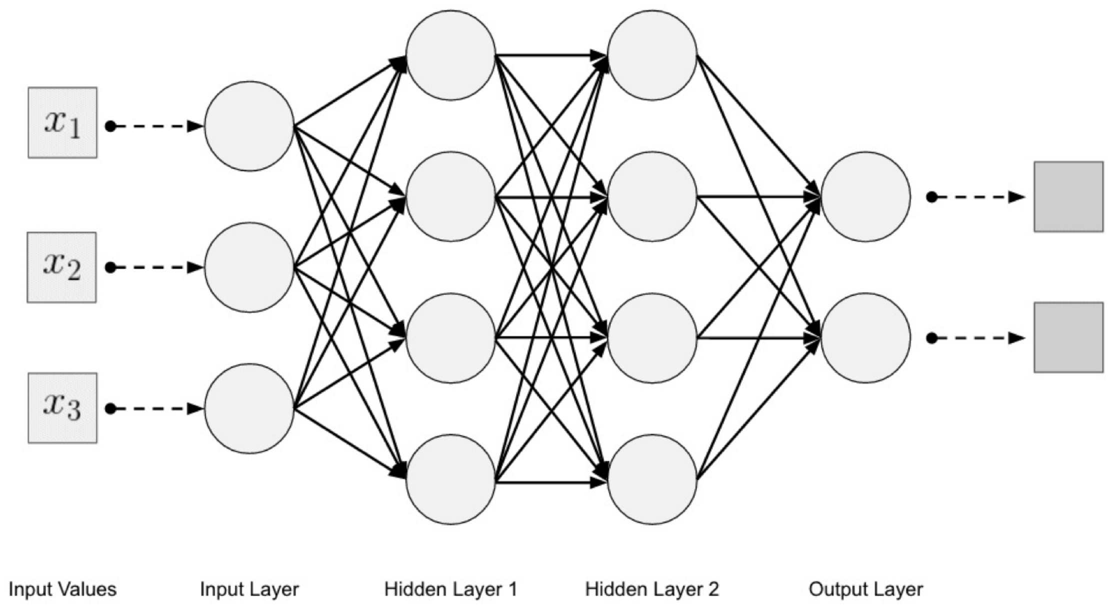


Figure 2.2.2: Visualization of an Multi-Layer Perceptron: The first layer is the input layer, all inner layers of MLP are called hidden layers, and the last layer is named the output layer (image taken from [PG17]).

and the final output layer. The input layer delivers inputs to the network. All layers, besides the input and output layer, are called hidden layers and compute intermediate representations. The computing process in which the product of the input values and the weights is fed to the summing node with neuron bias is the same as described in Sec. 2.2.1. Finally, the output layer provides the output of the MLP.

Inspired by the principles of the human nervous system, multilayer perceptrons can learn and make data predictions. In the training process, they can use learning algorithms to adjust the weights and reduce the bias, which is the error between the actual value and the predicted value [Nie19]. The main advantage of MLPs lies in its ability to solve complex problems quickly. They are the promotion of perceptrons and overcome perceptrons' weakness that cannot recognize linearly inseparable data. If each neuron's activation function is linear, then the MLP of any number of layers can be reduced to an equivalent single-layer perceptron.

Activations

In theory, MLPs can use any form of the activation function. However, in order to use the backpropagation algorithm for effective learning (see Sec. 2.2.2), the activation function used in MLPs is limited to the nonlinear function. In MLPs, the sigmoid function is adopted traditionally as the activation function due to its good differentiability and is defined by

$$\mathrm{sigm}(x) = \frac{1}{1+e^{-x}}. \tag{2.2.2}$$

For a neuron with inputs $x_1, x_2, ..., x_n$, weights $w_1, w_2, ..., w_n$ and bias b, its output is $\frac{1}{1+e^{-z}}$ where z is the weighted sum of input connections $\mathbf{w}^T\mathbf{x} + b$. The sigmoid function can be explained as a smoothed version of the step function, with the characteristic that it can be differentiated anywhere. The big difference between the step function and the sigmoid function is that the latter not only outputs 0 or 1, it can also output any real number between 0 and 1. [Nie19].

However, for deeper neural networks, the sigmoid function has the problem of vanishing gradient. Therefore, in the latest deep learning development, the Rectifier Linear Unit (ReLU) is more frequently used to overcome numerical problems related to the s-shaped functions. It was originally proposed by Hahnloser *et al.* for hardware circuits that represent neural networks [HSM+00]. In [NH10] and [SF16], the ReLU is used in the deep feedforward architecture. The ReLU is defined by:

$$f_{\mathrm{ReLU}}(x) = \max(0, x) \tag{2.2.3}$$

where x is the input of neuron. In the neural network, the ReLU is used as the activation function of the neuron to define the nonlinear output result of the neuron after the linear transformation $\mathbf{w}^T\mathbf{x} + b$. In other words, for the input vector x from the upper layer of the neural network, the neuron using the ReLU activation function will output the greater value of 0 and $\mathbf{w}^T\mathbf{x} + b$ to the neurons at the next layer or as the output of the entire neural network, depending on where the current neuron is located in the network structure. Fig. 2.2.3 visualizes the sigmoid and ReLU activation functions.

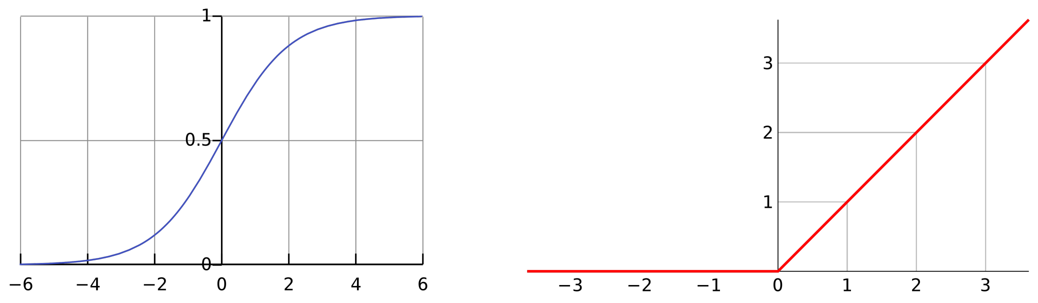


Figure 2.2.3: Plots of the sigmoid (left) and the ReLU (right) function. The sigmoid logistic function ranges from 0 to 1, while the ReLU function ranges from 0 to ∞.

2.2.2 *Training*

Training a neural network involves using the training data set to update the model weights to create an approximately errorless mapping from input to output [Nie19]. A non-linear optimization method called gradient descent is generally used to adjust the weights appropriately in the training process. To significantly accelerate the speed of gradient calculation and training, the Stochastic Gradient Descent (SGD) approach has become popular. For multi-layer networks, the most widely used learning technique is back-propagation. This section gives a detailed description of the standard training process of neural networks.

The training purpose is to minimize the difference between the output obtained for specific input and the desired output. This difference needs to be represented by a loss function. This function calculates the deviation of the output value $\hat{\mathbf{y}}$ predicted by the network from the desired output value $\mathbf{y}$. A possible loss function is defined as follows:

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2. \tag{2.2.4}$$

The training process is iterative, each iteration will make small updates to the model weights, and these updates also affect the model's performance. After repeating this training process enough times, the network will usually converge to a state where the calculation error is minimum.

Gradient Descent

A commonly used optimization method is called gradient descent, which adjusts the weights based on the error caused by the output values and the correct answers. The gradient we care about describes the relationship between the network error and a single weight and denotes how the error changes as the weight is adjusted. In order to use gradient descent to find the local minimum of the loss function E, the training process resorts to steps proportional to the negative value of the gradient of the function at the current point:

$$w_{ij}^{l} \leftarrow w_{ij}^{l} - \mu \frac{\partial E}{\partial w_{ij}^{l}} = w_{ij}^{l} + \Delta w_{ij}^{l} \tag{2.2.5}$$

where w_{ij}^{l} represents the weight between the i-th neuron in layer $l-1$ and the j-th neuron in layer l, and μ is known as the learning rate [Nie19]. The learning rate is vital for the optimization process. If μ is too small, the weights will be adjusted too slowly, and it will take a long time for us to converge to a local minimum. If it is set too high, we may overshoot and not converge in a local optimum [Nie19]. Therefore, such learning programs can be prolonged. It has been found that adding the momentum term into the optimization can significantly increase the rate of convergence and stabilize the training process [RHW86]. With this method, at iteration t the weight update Δw_{ij}^{l} takes the form:

$$\Delta w_{ij}^{l}(t) = -\mu \frac{\partial E}{\partial w_{ij}^{l}(t)} + p \cdot \Delta w_{ij}^{l}(t-1) \tag{2.2.6}$$

where p is the momentum parameter. The equation 2.2.6 means that the modification of the current weight vector depends on the current gradient as well as the weight change of the previous step. Gradient descent with momentum takes into account the past gradient to smoothly update. So it works faster than the standard gradient descent algorithm [Nie19].

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a current optimization method in machine learning. SGD is much faster than methods such as batch gradient descent when training large amounts of data and does not lose model accuracy (cf. [PG17], p.

189). In gradient descent, the overall loss of all training examples is calculated. However, in SGD, the gradient is calculated from a small, randomly drawn subset of training samples. When training a neural network with many data samples, this method can reduce the amount of calculation and achieve faster training.

The SGD convergence is limited by random noise caused by the random selection of an example in each iteration. Therefore, momentum is necessary during training when performing SGD. It has been proved in [BB07] that when the training set is large, even without access to all training examples, SGD could find a desired local minimum.

Backpropagation

Backpropagation is the abbreviation of error backpropagation and is a common method used in combination with optimization methods to train artificial neural networks. This algorithm is proposed in [Wer74]. Then, Rumelhart *et al.* make it popular [RHW86]. This algorithm computes the gradient of the loss function respect to all weights in the network. The gradient is then fed back to the optimization method to update weights to minimize the loss efficiently. Today, the backpropagation algorithm has become the most widely used training algorithm in machine learning [Nie19].

Output $\hat{\mathbf{y}}$ of the MLP is calculated through a forward pass of the input data in the network:

$$\hat{\mathbf{y}} = f^L(f^{L-1}(\cdots f^2(f^1(\mathbf{x})))) \tag{2.2.7}$$

where f is an activation function and L is the number of layers in the MLP. Therefore, a technique called chain rule can be used in backpropagation algorithm to compute the gradient of the loss function for the weight w_{ij}^l in the layer l:

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial w_{ij}^l} = \frac{\partial E}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{ij}^l} \tag{2.2.8}$$

$$z_j^l = \sum_{k=1}^{n} w_{kj}^l \cdot a_k^{l-1} + b_j^l \tag{2.2.9}$$

$$a_i^l = f(z_i^l) \tag{2.2.10}$$

where n is the number of neurons in $l-1$ layer, z_j^l represents neuron value at the hidden layer l, and the activation function f is non-linear and differentiable [Nie19]. The equation 2.2.8 means that in order to calculate the gradient of any weight w_{ij}^l, the gradient for the previous layer needs to be known first. In the last factor of the right-hand side of the equation 2.2.8, only one term in the z_j^l depends on w_{ij}^l, so that

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{ij}^l} = \frac{\partial E}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot a_i^{l-1} = a_i^{l-1}\delta_j^l \tag{2.2.11}$$

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_{k=1}^{n} \frac{\partial E}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_{k=1}^{n} \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_{k=1}^{n} \delta_k^{l+1} w_{kj}^{l+1} \cdot f'(z_j^l) \tag{2.2.12}$$

where δ_j^l is the error of the neuron j in the hidden layer l and $f'(z_j^l)$ measures how fast the activation function f is changing at z_j^l. The error in the hidden layer l is represented as

$$\delta^l = ((\mathbf{w}^{l+1})^T \delta^{l+1}) \odot f'(z_j^l). \tag{2.2.13}$$

Therefore, backpropagation requires a known and desired output for each input value to calculate the loss function's gradient. This algorithm calculates the error vector backward from the last layer.

2.2.3 *Convolutional Neural Network*

A Multilayer perceptron is usually represented as a fully connected network (cf. Sec. 2.2.1). This complete connection of the network causes probably the network to encounter the problem of over-fitting during training because the network requires a large number of parameters. Overfitting is an appearance that the learning algorithm cannot adapt to the training set well. Techniques learned too well on the training set may not achieve good performance results on the test set [Mur05]. Compared with MLPs, CNNs do not show a full connection of each layer with the previous layer but slides different convolution kernels or small filters over the input image. This model can be trained using backpropagation algorithms (cf. Sec. 2.2.2). Moreover, compared to other deep, feed-forward neural

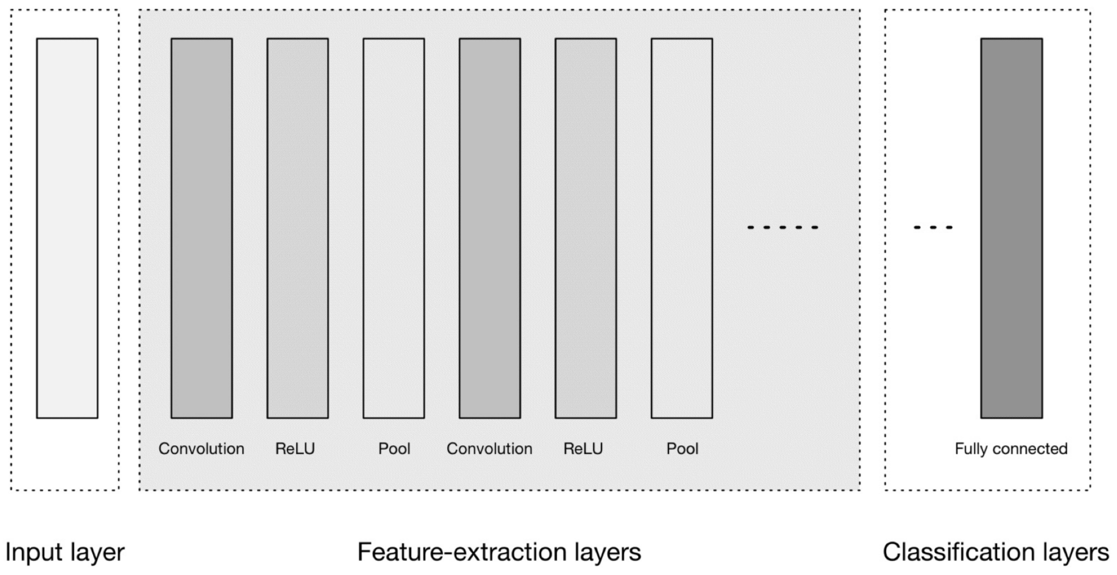
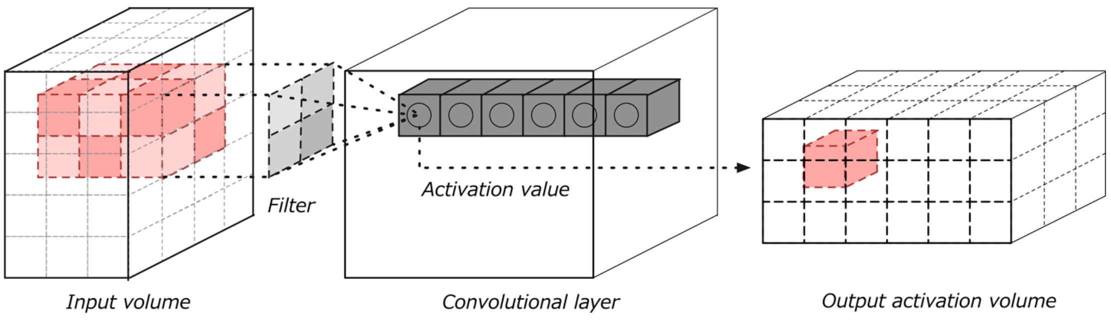


Figure 2.2.4: Schematic diagram of a convolutional neural network (CNN) architecture (image taken from [PG17]).

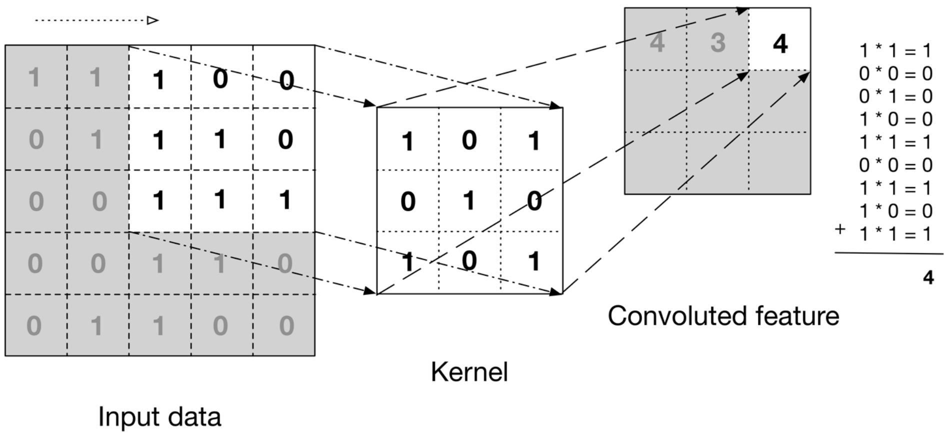
networks, convolutional neural networks need to consider fewer parameters and give better results in image recognition, which makes it an attractive deep learning model. Deep convolutional networks have been successfully used in the field of pattern recognition and word spotting methods [SF16].

As demonstrated in Fig. 2.2.4, the convolutional neural network consists of three major parts: the input layer, the classification layers, and the feature-learning layers that include one or more convolutional as well as pooling layers. The input layer usually accepts a three-dimensional input consisting of the image's length and width and a depth representing the color channels. The feature-extraction layers can extract many features in the inputs and gradually construct higher-order features. Finally, there are fully connected layers in the classification layer that receive high-order features and generate class probabilities or scores [PG17].

The convolutional layers are considered the core building block of a CNN. Each convolutional layer consists of a collection of learnable filters (kernels) passed over the entire input and viewing some pixels at a time, for example, 3×3 or 5×5 pixels. As shown in Fig. 2.2.5, each filter is convolved with the input volume to compute an activation map composed of neurons. The convolutional layer's



(a)



(b)

Figure 2.2.5: (a) is a convolution layer with input and output volume. (b) shows an example for the convolution operation (image retrieved from [PG17]).

output volume is obtained by stacking the activation maps of all filters in the depth direction. Each entry in the output can also be interpreted as the neuron's output that observes a small area in the input and shares parameters with neurons in the same activation map. This parameter sharing scheme reduces the number of parameters and shortens training time [PG17]. The convolution operation is also denoted as the feature detector of a CNN. Fig. 2.2.5 visualizes this concept with filter kernels of size 3×3 and an input data of 5×5 pixels. A convolution can use raw data or a feature map output from another convolution as input. Then,

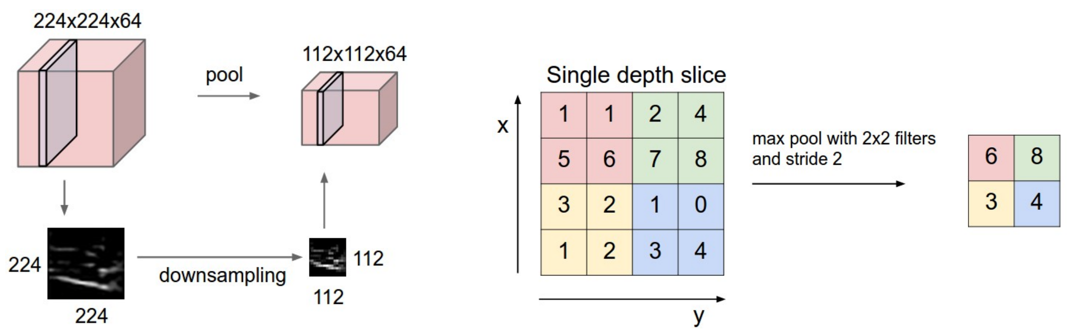


Figure 2.2.6: An example for the max pooling (right). The downsampling in a pooling layer (left): Input volume of size $[224 \times 224 \times 64]$ is pooled with filter size 2×2 and a stride 2 into output volume of size $[112 \times 112 \times 64]$ (the source of images : [AR20]).

the results are summarized into a number representing all the pixels observed by the filter.

Pooling is another crucial concept of CNNs and is a nonlinear form of downsampling. Pooling layers are generally inserted between two successive convolutional layers to gradually reduce the spatial size (width and height) of data representation and help control over-fitting. The pooling layer downsamples the volume spatially, independently in each depth slice of the input volume [PG17]. Fig. 2.2.6 visualizes this concept for a pooling layer which uses filters of size 2×2 and a stride of 2 to downsample at every depth slice in the input along both width and height. During this process, the depth dimension remains unchanged. The pooling layer can continuously reduce the spatial size of data, so the number of parameters and the amount of calculation will also decrease, which also avoids overfitting to a certain extent. The pooling operation provides another form of translation invariance. Because the convolution kernel is a feature detector, different edges in the input data can be easily found in the convolution layer. However, the features found by the convolutional layer are often too accurate. The pooling layer can reduce the sensitivity of the convolutional layer to edges [PG17].

The max-pooling is the most common form of nonlinear pooling function [KSH17]. An excellent example of max-pooling is given in Fig. 2.2.6. The max-pooling divides the input image into several rectangular areas and outputs the maximum value for each sub-area.

The classification layers are used to calculate the class scores used as the output of the network. In classification layers, as shown in Fig. 2.2.4, multiple fully connected layers are stacked together to form a standard MLP as a classifier.

3 RELATED WORK

This chapter gives an overview of handwritten word synthesis and the PHOCNet. Generally, data creation is a time-consuming and expensive process because it requires much manual work from the data collection stage to annotation and verification. Using a synthetic handwritten text generator to replace the traditional data generation process has become very popular. This ideal allows us to have a large number of training samples without any manual effort, which is undoubtedly good for CNN-based word spotting methods that require a large amount of labeled training data. Much research has been done in synthetic text generation, but only a part of them is dedicated to handwritten text. Sec. 3.1 details the current researches creating synthetic handwritten data.

The present thesis also studies a word spotting method based on segmentation, which can perform query-by-example and query-by-string. In order to realize these two application scenarios, with the help of embedded PHOC attributes, images of text and visual words are converted into attribute representations in attribute space. The PHOC representation will be explained in detail in Sec. 3.2. In order to use PHOC vectors to map word images in the n-dimensional vector space, a deep PHOCNet is used, and its architecture and functions are described in Sec. 3.2.2.

3.1 GENERATION OF SYNTHETIC WORD IMAGE

Many efforts have been implemented in the field of synthetic data generation. The idea of generating synthetic data is constructive for overcoming the inherent difficulty of obtaining data. Data sets of various sizes can be synthesized according to experimental requirements in this way, which can alleviate the desire for massive data in many machine learning architectures. This section introduces the most mainstream data synthesis methods. The methods specifically for handwritten data generation can be roughly divided into the following three categories: glyphcentric approach, generational method, and font-based method.

3.1.1 *Glyphcentric approach*

In [HAB16], Haines *et al.* propose a glyph-based algorithm to render the desired string with a writer's handwriting. This method is called the glyphcentric approach because its rendering process does not use existing fonts but the specific natural handwriting style learned from a given sample of an individual's handwriting. With learned parameters for spacing, line thickness, and pressure, novel images of handwriting can be produced that imitate a specific author's handwriting style.

Fig. 3.1.1 shows how this glyphcentric method works. The first step is to collect annotated samples of the author's handwriting. Then, the glyph structure from the samples is analyzed. After collecting and analyzing samples, the rendering system selects a glyph to represent each character. The glyphs are then placed on the page, and ligatures are added if the author uses joined-up writing. Finally, the texture is transferred from the original input to the vector output, and, if being printed, color correction is applied. This generative model is built around glyphs. The synthetic data generated by this model looks like the handwriting written by a specific author.

3.1.2 *Generational method*

In [Gra13] [BPC+17] and [JBMN09], the ideas of synthesis aim to generate the handwriting strokes of word synthetically, usually employing RNNs or GANs. This method is a great advantage in working with online data since the sequential structure is inherent to online data. Similarly to the Glyphcentric approach described in Sec. 3.1.1, this method also needs real, already labeled samples, and considers the personal handwriting style.

In [Gra13], Graves considers handwriting as a trajectory and generate synthetic text image for a given the word by producing a sequence of pen positions, as shown in Fig. 3.1.2. In [Gra13], a Long Short-Term Memory (LSTM) network is used to predict such a sequence, and the network trains its prediction on the target string to synthesize handwriting. However, Graves' method does not allow the handle of some standard features in offline data, such as background texture or stroke thickness variations.

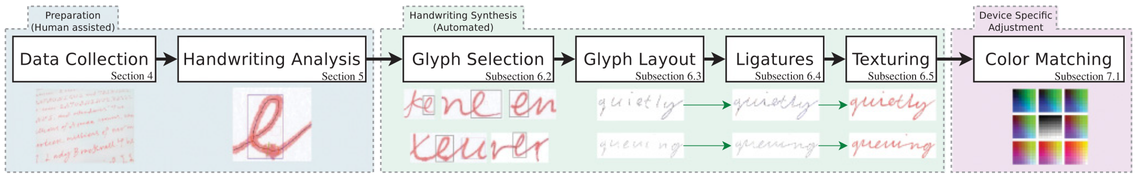


Figure 3.1.1: System diagram showing the processing pipeline of glyphcentric approach, with representative images for each stage (image taken from [HAB16]).

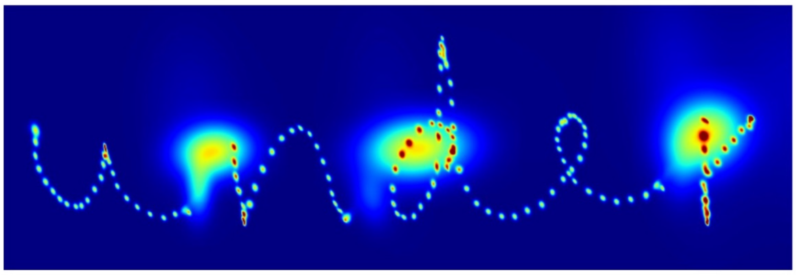


Figure 3.1.2: This figure demonstrates a scheme of synthesis using online handwritten samples with strokes. A LSTM network predicts the next pen position based on the previous stroke, according to the sequence and handwriting style. The small blobs are the predictions when the strokes are being written. And the three large blobs are the predictions at the ends of the strokes for the first point in the next stroke (image taken from [Gra13]).

3.1.3 *Font-based method*

This method is very similar to the Glyphcentric approach, but it does not need any real samples. In these representative experiments [AF15][RMC18][KJ16a], the available font classes are used to render word images and applying some distortions to increase realism and variability. This concept was successfully implemented in [AF15], where a training data set was created using various available Arabic fonts and many different deformations; finally, this data set is used to train an HMM (Hidden Markov Model). Then, [RMC18] and [KJ16a] extend this idea. Especially in [KJ16a], the famous synthetic handwriting data set, IIIT-HWS, is released, which contains a large number of synthetic word images rendered out of 750 publicly available handwritten fonts.

As more and more handwritten font families are available recently, we continued the framework of Krishnan *et al.* [KJ16a] to render large-scale synthetic handwritten data to train deep neural networks. The specific synthesis process will be elaborated in Chap. 4. Based on the framework of Krishnan *et al.*, we also use elastic distortion to mimics variations of the writing style of single characters in a word.

3.2 PHOCNET

This section provides a detailed introduction to the PHOCNet used in this work. In [SF16], Sudholt and Fink propose a very successful deep CNN architecture, the PHOCNet. It is inspired by the VGGNet [SZ15] and designed for word spotting. The task of the PHOCNet is to predict the Pyramidal Histogram of Characters (PHOC) representation for a given word image, and then use this attribute representation to perform word spotting. The PHOC is the attributes based representation of a word image, where words are represented as binary PHOC vectors in a d-dimensional space. It is used as the source of character attributes to encode whether certain characters exist in certain spatial regions of the string. This important concept is introduced in Sec. 3.2.1.

One of the advantages of the PHOCNet is that it can perform word spotting through both Query-by-String (QbS) and Query-by-Example (QbE). Another significant advantage is that it can accept input images of arbitrary size due to the use of the Spatial Pyramid Pooling (SPP) layer [HZRS14] which enables the networks to receive a variable-size input while still producing a constant size representation and pass this resulting representation to the directly following fully connected layer. Because training a CNN with PHOCs as labels can be seen as a multi-label classification task, the PHOCNet places sigmoid activation in its last layer and use Binary Cross-Entropy Loss as the cost function to train the network. The specific architecture of the PHOCNet is explained in Sec. 3.2.2.

3.2.1 *PHOC Representation*

The Pyramidal Histogram of Characters (PHOC) is proposed by Almazán *et al.* in [AGFV14]. In the embedded PHOCNet framework, the PHOC representation plays a vital role in representing strings and word images. The PHOC encodes visual attributes of the corresponding word image, where intuitive attributes of a word image are its characters. Fig. 3.2.1 shows the process of constructing a PHOC vector for a given text string. The first level of the PHOC for a given text string "place" shows which letters appear in the entire string, and at the 2nd level, the PHOC encodes whether the word contains a particular character in the first or second half of the word. On 3nd level, the word is divided into 3 regions.

A binary vector is determined for each word region, which shows whether a specific character appears in this word region. When using ten digits plus 26 Latin alphabet, this results in a 36-dimensional histogram, where each dimension represents whether the text string contains a specific character. This model requires multiple levels because the resulting vector for the first level in PHOC cannot distinguish the difference between two similar words. For example, "team" and "meat" have the same representation at the first level. The PHOC focuses on finding characters in different areas of a word string, not just on the whole word.

In [AGFV14], Almazán *et al.* use levels 2, 3, 4, and 5 to represent word images, leading to a histogram of $(2+3+4+5) \times 36 = 504$ dimensions. They also add the 50 most common bigrams in English at Level 2, which means that there are $50 \times 2 = 100$ extra dimensions. So the total size of PHOC is $504 + 100 = 604$. Besides, Fig. 3.2.1 also shows a special situation that the word split falls on a certain letter, which causes this letter to overlap with both regions, for example, the character "a" on the second level and the letters "l" and "c" on the third level (see Fig. 3.2.1). Almazán *et al.* defines that if a letter overlaps with a specific area by at least 50% in this case, it is marked as present in this area.

In [AGFV14], Almazán *et al.* use the advanced Fisher vector coding method [PSM10] to encode word images into feature vectors, and then used these feature vectors with PHOC labels together to learn SVM-based attribute models. Another successful application of PHOC is in [SF16], where Sudholt and Fink use PHOC representation to train a new CNN-based architecture, the PHOCNet. Their experiments prove that the PHOCNet can show excellent performance over other word spotting models with a short training and test time.

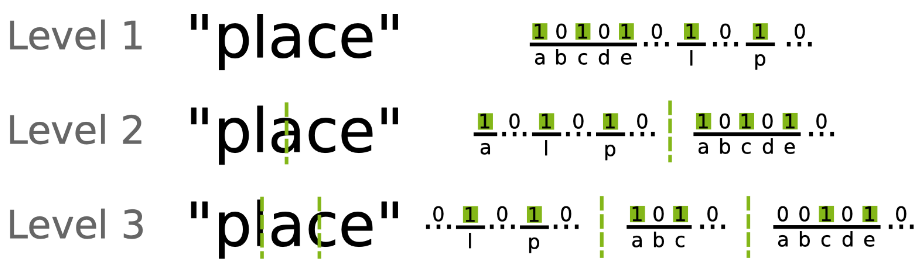


Figure 3.2.1: The figure visualizes the extraction of a PHOC from string "place" at levels 1, 2 and 3 (image from [SF16]).

3.2.2 *PHOCNet Architecture*

The PHOCNet model is a CNN-based deep neural network specially designed for word spotting. Fig. 3.2.2 visualizes the PHOCNet architecture. This architecture of is similar to VGGNet [SZ15]. Lower convolutional layers use fewer filters, while the amount of filters in higher layers is doubled. However, the PHOCNet expands more possibilities based on VGGNet. The main difference from VGGNet is the use of a Spatial Pyramid Pooling (SPP) layer after the convolution part.

In the case of segment-based word spotting, each text image is preprocessed into many segmented word images, and then, the CNN predicts a representation for the segmented word images. These segmented word images show great variability in size, which is related to the composition of the word; for example, the number of letters that make up the word "internationalization" and the word "a" differs greatly. Generally, the classifier deployed at the end of a CNN allows only a fixed-sized input. For this, word images can be cropped (e.g. in [HZRS16]) or anisotropically rescaled (e.g. in [KSH17]) to a fixed size. Nevertheless, these methods might cause drastic changes to images: Cropping may damage some valuable information from images, and if the anisotropic rescaling changes the images' original aspect ratio, the images will be distorted.

The Spatial Pyramid Pooling (SPP) Layer proposed by He *et al.* can alleviate this problem. The convolutional and the pooling layers can handle various input sizes, but the fully connected layers cannot cope with different input sizes. Therefore, the SPP layer is placed between the convolution part and the fully connected part as the last pooling layer before the classifier. In this way, the PHOCNet can process the input of different sizes while producing constant size output. The PHOCNet

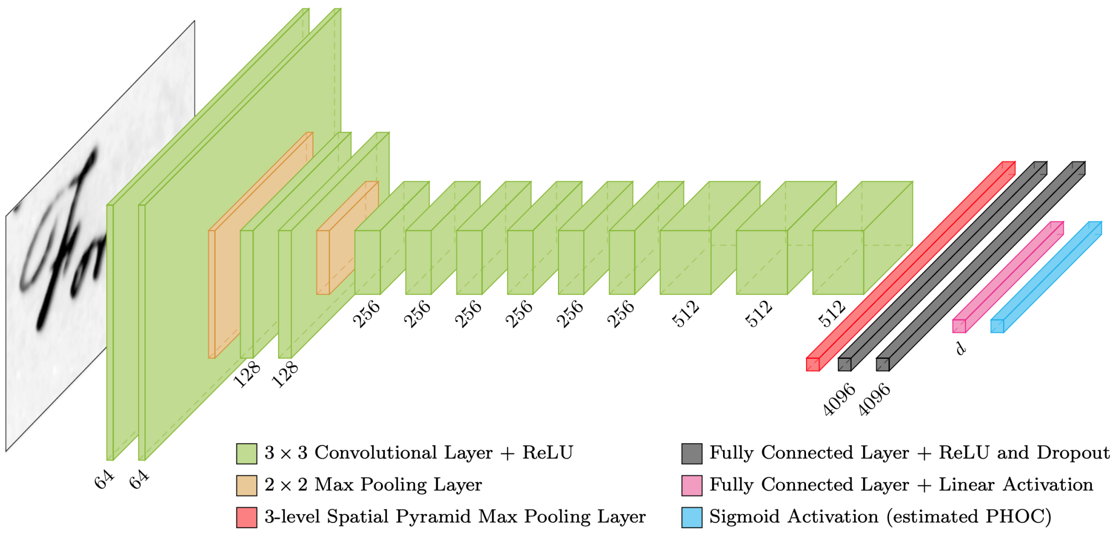

64
64
128
128
256
256
256
256
256
256
512
512
512
4096
4096
d
3 × 3 Convolutional Layer + ReLU
2 × 2 Max Pooling Layer
3-level Spatial Pyramid Max Pooling Layer
Fully Connected Layer + ReLU and Dropout
Fully Connected Layer + Linear Activation
Sigmoid Activation (estimated PHOC)

Figure 3.2.2: The figure visualizes the PHOCNet architecture: The green layers represent the convolutional layers, and the numbers marked below represent the number of filters in the corresponding layer. 3×3 convolutions and a ReLU activation function are used in all convolutional layers. All 2×2 max-pooling layers are depicted by orange, where stride is set as 2. The red layer represents a 3-level SPP layer. Moreover, the fully connected layers are shown by a black box, where "4096" represents the number of neurons in each block of fully connected layers, the dropout is 50%, and ReLU is used as the activation function. The size of the last layer (blue) for output depends on the size of the PHOC (image taken from [SF18]).

uses a three-level SPP that follows the layout rules of the original spatial pyramid [LSP06], where the number of regions in each level along each dimension is twice that of the previous level. For a 3-level SPP, each feature map is globally pooled at the first level, and the feature map at the second level is proportionally divided into four regions, and the third level splits the feature map into 16 regions totally.

Like VGGNet, the number of neurons in the two fully connected layers in the PHOCNet is also set to 4096, which means that there are many free parameters in the fully connected part. During training, a dropout probability of 50% is applied to neurons in the fully connected layer to avoid overfitting.

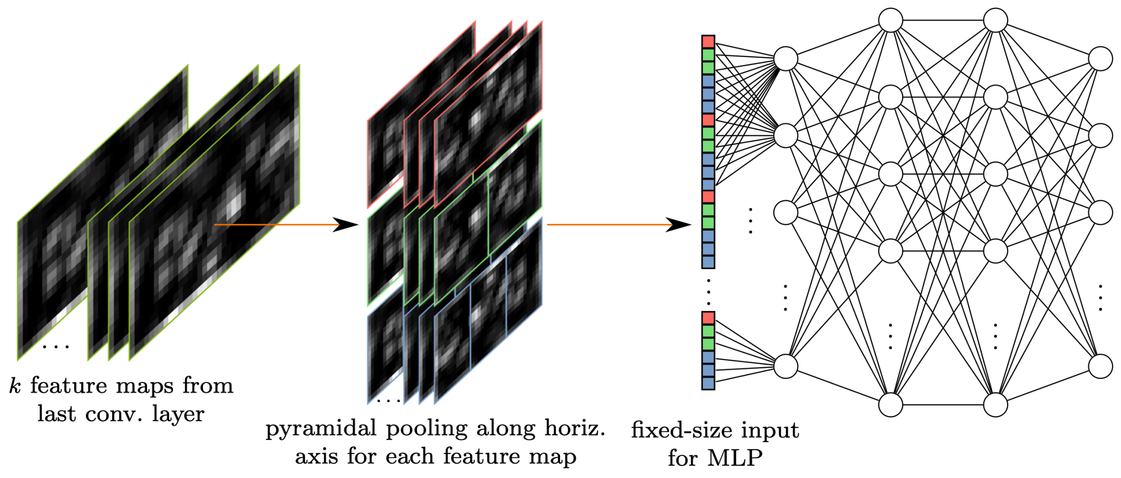

k feature maps from last conv. layer
pyramidal pooling along horiz. axis for each feature map
fixed-size input for MLP

Figure 3.2.3: The figure shows a 3-level TPP layer. This layer can extract a fixed-size representation from feature maps of arbitrary size while only considering the splits along the horizontal axis. And then, this representation is fed to the MLP part of the network (image taken from [SF18]).

In [SF18], Sudholt and Fink propose a modified version of the PHOCNet and called it the TPP-PHOCNet. In the advanced TPP-PHOCNet, the SPP layer is replaced by the Temporal Pyramid Pooling (TPP) layer. As mentioned above, SPP layer follows the design of the spatial pyramid proposed by Lazebnik *et al.* [LSP06]. However, the application of this layout on Bag of Features (BoF) representations leads to undesirable results [ARTL13]. However, the results can be improved, when employing spatial pyramids with a fine-grained split for the horizontal axis and a coarse split for the vertical axis of a word image. To improve the retrieval results, Sudholt and Fink use this concept to optimize the SPP layer and propose the Temporal Pyramid Pooling (TPP) layer, as visualized in Fig. 3.2.3. At each level in this layer, the feature maps used as input are divided only along the horizontal axis. Each horizontal region covers the entire vertical axis of the feature map. The values obtained after pooling hence reflect sequential features of the word image along the writing axis. A pyramidal representation encoding writing progression can be obtained by stacking numerous pooling layers with different amounts of splits along the writing axis.

4 METHODS

In the past few years, research on handwritten documents has been very active. Despite the significant progress made in these studies, handwritten document word spotting still relies heavily on available data. However, in application scenarios, the early exploration of historical documents usually does not have enough data with different writing styles and qualities. With the emergence of many handwritten data with different styles and deformations, this problem can be largely alleviated.

The previous chapters have established the necessary fundamentals. This chapter will present the methodology used in this work. This thesis intends to generate highly realistic synthetic data for CNN-based word spotting methods. This work follows some successful experiences in generating synthetic datasets [BW17] [JSVZ14] [KJ16a] [RMC18] and create a font-based synthetic data generator. The generator can take a string as input and generate the corresponding human-like handwritten word images as output, as shown in Fig. 4.0.1. The main contribution of this generator designed to generate synthetic data is that it can easily create a large number of handwritten word images, which mimics the natural writing features, and then these images can be directly used for training on the previously described PHOCNet (see Sec. 3.2.2).

Two crucial sub-tasks need to be considered to complete this task: The first is to render the word images (cf. Sec. 4.1). The second is to apply distortions on the rendered images (cf. Sec. 4.2). This framework is inspired by the method described in [KJ16a]. Based on the method of Krishnan *et al.*, we additionally use the elastic distortion during processing images. In [SSP03], Patrice *et al.* have used this deformation to expand the MNIST data set of handwritten digital images and have proven its effectiveness. In this work, we use it in synthetic handwritten word images to imitate the variations produced by the muscles' natural vibration and the inertia imposed on the writing medium.

This chapter's focus is to introduce the methods used to generate synthetic data, and the word spotting performance of the trained model will be evaluated in Chap. 5.

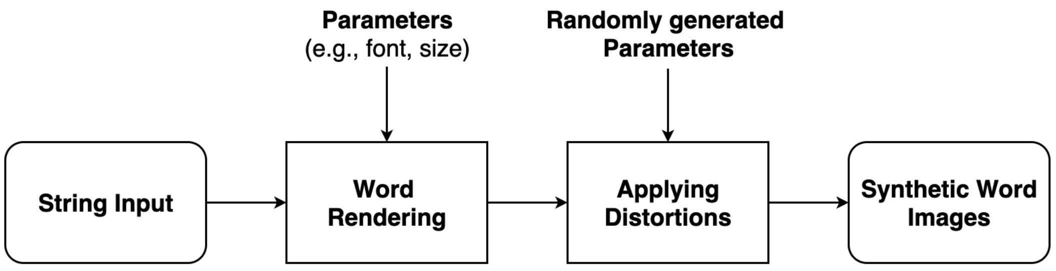


Figure 4.0.1: Synthetic word generation approach

4.1 SYNTHETIC WORD IMAGE GENERATION

The successful framework in [KJ16a] inspires our generator of synthetic word images used in this thesis. Like the method of Krishnan *et al.*, first, the vocabulary of words is selected from a dictionary, and then the generator for each word samples randomly one font and renders its corresponding image with this font. In this process, the generator will randomly choose various parameters within the given intervals. To make the synthesized word images as realistic as possible, we only focus on fonts with evident handwriting characteristics.

Section 4.1.1 explains 6 different vocabularies that will be used in our experiments, while section 4.1.2 describes in detail the process of rendering word images.

4.1.1 *The Vocabulary of Words*

For a synthetic data set used to train the word spotting models, the choice of the vocabulary of words is crucial. As the synthetic data set representatives, HW-SYNTH [KJ16b] and IIIT-HWS dataset [GSF18] selected $10,000$ and $90,000$ English words from the Hunspell[1] dictionary, respectively. The experiments in [WBF20] have proved that the vocabulary of HW-SYNTH is quite different from the vocabulary of the traditional real data sets, which means that the vocabulary of HW-SYNTH data set does not have universally adaptability. Although the vocabulary of IIIT-HWS is large and has a high degree of overlap with the

1 Hunspell is a spell checker and morphological analyser, https://github.com/hunspell/hunspell

vocabulary of the real data sets, a large part of it is non-representative words. These irrelevant words not only failed to improve the performance results but may also interfere with the model to learn useful information. Compared with them, focusing on the most common words in [WBF20] is more reasonable.

In the work of synthesizing word images, the choice of vocabulary is the first step. Like the method in [WBF20], we select the most frequently used words into the vocabulary list because they have an exceptionally high probability of appearing in new documents that need to be retrieved. To obtain a representative vocabulary, we use a Python library, Wordfreq [SCL+18], where all data comes from the Exquisite Corpus [SCL+18] generated from a variety of modern English text sources.

To explore the impact of vocabulary size on the model's performance at the experimental part, we created six vocabularies with sizes of 5k, 10k, 15k, 20k, 25k, and 30k. They all only consider the most frequently occurring words and will be used to generate different synthetic data sets. Moreover, these 6 vocabularies are composed of numbers, words, and phrases without special symbols.

4.1.2 *Font Rendering*

In recent years, the availability of more and more public handwritten fonts has encouraged the development of font-based synthetic data generation methods. The main challenge in the synthesis of realistic handwritten word images comes from differences in writing styles: different authors for the same word may have completely different writing styles. To generate high-quality synthetic data imitating real word-images, we picked 410 handwritten fonts with different writing styles from Google Fonts[2] and 1001Fonts[3].

To obtain a composite image of the target word, we use ImageMagick[4] as an image rendering tool to convert images for each word in the vocabulary. ImageMagick is a powerful free software for creating, editing, compositing as well as converting bitmap images. We randomly sample a font from the fonts set for each given string, then access the "**convert**" command of ImageMagick and change the following necessary parameters:

2 Google Fonts is a open-source library of 999 font families, `https://fonts.google.com`
3 1001Fonts is a library of 12,220 free licensed font families, `https://www.1001fonts.com`
4 ImageMagick toolbox, `http://www.imagemagick.org`

- **"label:** *string* **"** :

  receives the target word. Creating a font image using a "label:" image is a typical way of drawing a font quickly in ImageMagick.

- **-font** *name* :

  sets the font to use for creating images with the given label. The font is randomly selected from the font list.

- **-pointsize** *value* :

  sets the height of the drawing area of the selected font. The value is selected randomly from the interval $(69, 81]$.

- **-strokewidth** *value* :

  sets the stroke width. It must cooperate with "-stroke *color*". Otherwise, it will not make any changes to the generated image. Here, the value is selected randomly between 0 and 1.5. Furthermore, the color is set as *black*.

- **-kerning** *value* :

  sets the space between two letters. The value is randomly selected from an interval $(-2.1, 2.1]$.

- **-gravity center** :

  specifies that the target word will be placed in the center of the created image.

- **-trim** :

  is used to remove the vast amount of extra white space in the created image to simulate the word segmentation.

Because very bad parameters can lead to distortions that are usually not found in real handwriting. Therefore, to ensure the non-aggressiveness of all parameters, all the above parameter intervals are chosen after the informal experiments prove their workability.

By using this step, we can generate the synthetic datasets of handwriting word images without any distortion. Fig. 4.1.1 shows some samples of the rendered undistorted word image. To address the in [KJ16a] mentioned problem that the

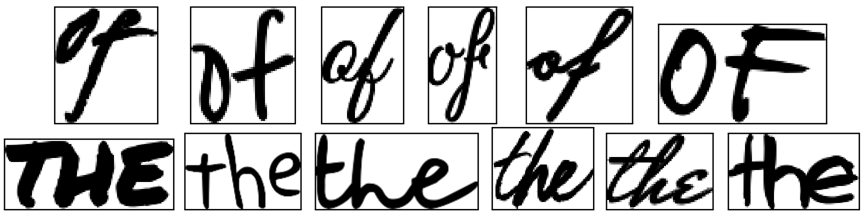


Figure 4.1.1: Sample synthetic handwritten word images without any distortion rendered in this work.

number of samples of certain words in real data sets is too small, we generate an equal number of images for each word in the vocabulary for each of our synthetic datasets.

4.2 APPLYING DEFORMATIONS

The goal of this work is to generate synthetic data very similar to real samples. To this end, the process of generating a synthetic image should simulate as much as possible every aspect of the natural handwriting process. As well known, many factors affect the natural handwriting process, and the writers' handwriting style is only one aspect. In Sec. 4.1.2, the existing handwritten fonts that mimic the different handwritings from different writers have helped mainly us alleviate the problem of diversity in writing styles to a certain extent.

To enhance the quality of the synthetic images generated in the previous chapter, we use some deformation operations to increase the image's realism. The following sections will detail each deformation method's principles and the goals to be achieved by using them.

4.2.1 *Affine Transformation*

In Euclidean geometry, the affine transformation is a geometric transformation that preserves straight lines and parallelism. After using affine transformation, in the image, the angle between straight lines and the distance between points may be changed, but the ratios of the distances between points on the same straight line will be maintained. The affine transformation can scale, rotate, translate, reflect, and shear the images, as shown in Fig. 4.2.1.

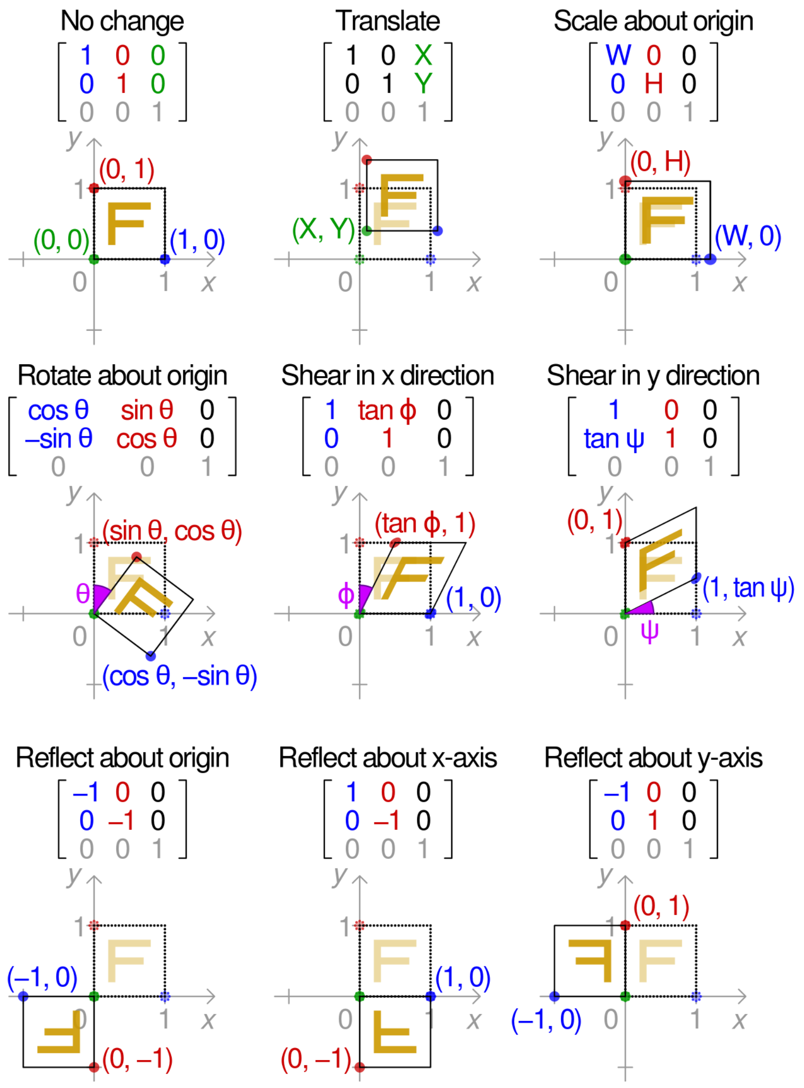


Figure 4.2.1: 2D Affine Transformation Matrix (image source : [Com20]).

In this work, we only choose the rotation and the shear mapping to imitate the writing habits of different writers to a certain extent, such as placing the writing medium at different angles, writing with different hands, or scribble on the unlined paper. This deformation will be applied to the rendered word images when generating the data set. We also use the “**convert**” command of ImageMagick, which can complete rotation and shearing for the generated word images at the same time. However, in addition to the parameter "-trim," the affine transformation also employs some new parameters to restrict the rotation and shearing at specific angles to mimic the skew and scribbling in natural handwriting:

- **-virtual-pixel** *color* :

  defines what color source should be used when a color lookup completely misses the source image. Since all the generated word images' background color is white, the color used to fill the area surrounding the image is set as *white*.

- **+distort AffineProjection** 's_x, r_x, r_y, s_y, t_x, t_y ' (see array 4.2.1) :

  distorts an image linearly, using the given affine Matrix of 6 coefficients forming a set of affine equations to map the source image to the destination image. The conversion matrix has 3×3 elements, as shown in array 4.2.1, but since three of them are constant, these three are omitted from the input. This method is used to apply the shearing along the horizontal direction on images, so the input is set as "1, 0, x, 1, 0, 0", where we employ a random amount of shear "x" ($+/-0.5$ degrees).

$$\begin{bmatrix} s_x & r_y & t_x \\ r_x & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2.1}$$

- **+distort SRT** *Rotate-Angle* :

  rotates the image according to the given angle. We apply a random amount of rotation ($+/-3$ degrees).

4.2.2 *Elastic Distortion*

The elastic transformation algorithm (Elastic Distortion) was first proposed by Patrice *et al.* in [SSP03], and it was initially used in the MNIST data set of handwritten digit images. It has been found that the recognition effect of handwritten digits has been significantly improved after expanding the MNIST data set through

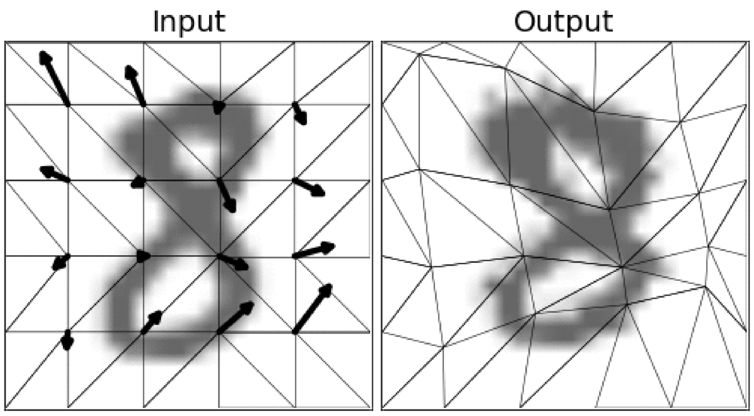


Figure 4.2.2: An example of a randomized elastic distortion (image taken from [BSH17]).

elastic distortions. Since then, it has become a prevalent technique for expanding image data sets to combat overfitting in deep CNNs and has been widely used to improve the performance as well as generalization of models, for example, in [BSH17], [WBF20], and [BIK+20].

In the natural writing process, we can hardly find two identical letters or words, even if the same author writes them. So this operation is intended to mimic this variability. Inspired by the method in [BSH17], image elastic deformations in this work is completed by first generating random displacement fields. We specify a random number between 6 and 21 as the grid size, which affects the deformations' granularity and the strength of displacement within the grid. To perturb control points, we sample a value randomly from a normal distribution with zero mean. We set a random standard deviation for this normal distribution from the interval $(0.3, 2.1]$. Fig. 4.2.2 visualizes this process, where the length of arrows shows the strength of the displacement within the grids.

4.2.3 *Naturalization of Pixel Distribution*

To model the distribution of fore- and background pixels, we use the Gaussian distribution in this work. The Gaussian (or normal) distribution is regarded as one of the most influential distributions in statistics. It is a continuous probability distribution that roughly describes some objects concentrated on their mean

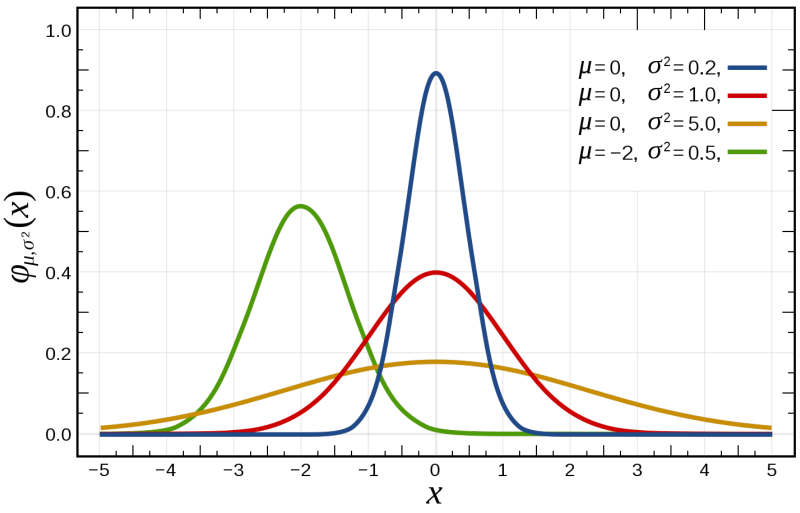


Figure 4.2.3: The one-dimensional Gaussian distributions: the probability density function is bell-shaped, peaking at the mean (image from [Com18]).

[SW10]. The density function of Gaussian distribution with mean μ and standard deviation σ is as follow:

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \text{ for } -\infty < x < \infty \tag{4.2.2}$$

where the Gaussian spread parameter σ determines the width of the Gaussian [JKS95]. As shown in Fig. 4.2.3, since its probability density function is bell-shaped, it is also regarded as a bell-shaped curve [DW14]. If a distribution is normal, the mean, median, and mode of it have then the same value, where "mean" means the average of all values, "median" signifies the value at the center point of the distribution, and "mode" denotes the value that appears most frequently during the measurement [DW14].

Due to the central limit theorem (CLT), Gaussian distribution is advantageous for representing real-valued random variables whose distribution is unknown. The average of random variables converges to a normal distribution and is normally distributed when the number of random variables is large. Therefore, the Gaussian distribution is particularly suitable as a simple model for many theoretical and practical problems in statistics and natural sciences [AFM17] [SW10].

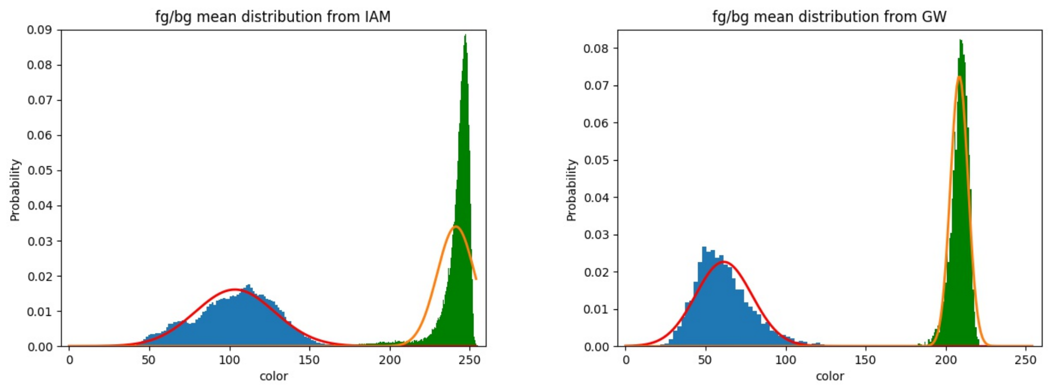


Figure 4.2.4: Pixel distributions of foreground and background regions estimated from IAM (left) and GW (right): The blue parts mean the distribution of foreground pixels, and the green parts are for the background pixels. This figure shows also the fitted normal curves to the distribution histograms.

Up to this stage, in this work, the synthetic images we have generated have only imitated a very idealized writing process with black ink on pure white paper. However, it is not easy to see a pure white background and text with the even black ink (as the foreground) in a real handwritten text image. In order to make the pixel distribution of foreground pixels and background pixels more natural, we use a method similar to that in [KJ16a]: evaluating a pixel distribution corresponding for the foreground and background regions of a real data set and then sampling the corresponding pixels for both regions of our synthetic images from these pixel distributions. This method will be referred to as the naturalization of pixel distribution of fore- and background pixels in this thesis.

The difference from the method in [KJ16a] is that we learn the parameters not only from the IAM data set but also from the George Washington (GW) data set. Fig. 4.2.4 shows Gaussian distributions of pixels from the foreground and background in the real data sets IAM and GW, while Fig. 4.2.5 gives some images processed by this method.

4.2.4 *Gaussian Smoothing*

As shown in Fig. 4.2.5, after naturalizing the foreground and background pixels of synthetic images, the junction between the foreground and background regions in the image becomes too apparent. Moreover, these tooth-like edges look very unnatural because the edges of ink usually look smooth in the natural handwritten text. To make the synthetic image look more real, we choose to use a Gaussian filter to smooth the generated word images. Another purpose of using the Gaussian filters is to simulate the blur probably caused when scanning or photographing handwritten documents.

Gaussian filter is a type of linear smoothing filter with the weights chosen according to the shape of a Gaussian function. Furthermore, the Gaussian smoothing filter is an excellent general-purpose filter, and it is the current standard method used to blur the image softly and remove noise and details [JKS95].

The specific operation of Gaussian filtering is first defining a mask with the size of the considered neighborhood, and the scanning each pixel in the image with this mask, finally using the weighted average gray value based on the pixels in the neighbor area covered by the mask to replace the value of the pixel located the center of the mask [GW19]. For image processing, the two-dimensional Gaussian function:

$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.2.3}$$

is used as a smoothing filter [SS01] [NA08]. In two dimensions, the Gaussian function is rotationally symmetric, which means that the amount of smoothing performed by the filter is the same in all directions. Generally, the edges in an image will not orient in a specific direction known in advance. Therefore, the Gaussian smoothing filter is very friendly to the edges of the image. The nature of rotational symmetry means that the Gaussian smoothing filter will not bias the subsequent edge detection in any specific direction [JKS95].

Besides, the two-dimensional Gaussian function has only one lobe, which means that the Gaussian filter performs smoothing processing by replacing each pixel in the image with the weighted average of adjacent pixels so that the weight is given to adjacent pixels monotonously decreases with the distance from the center pixel. This attribute is vital for image edge processing because the edge is a local feature

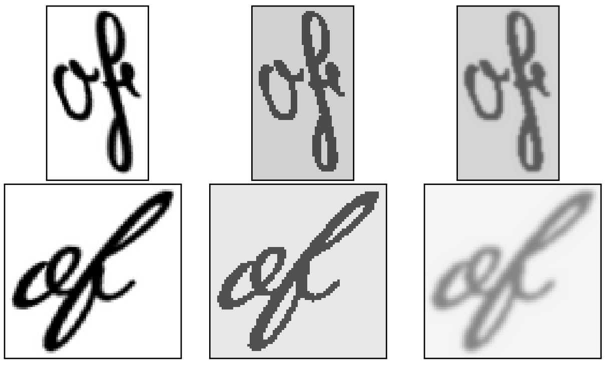


Figure 4.2.5: Left: Examples of simple synthetic word images without any distortion. Middle: The pixel distributions of the simple image are adjusted to the natural pixel distributions learned from the IAM dataset. Right: Gaussian smoothing is used on the images in the middle, and the smoothing σ is randomly selected from the interval $(0.3, 1.2]$ (top) or $(0.6, 1.5]$ (bottom).

in the image. It will deform this feature when a smoothing operation gives more importance to the farther pixel [JKS95].

The width of a Gaussian filter and the degree of smoothing is parameterized by σ. The relationship between σ and the degree of smoothing is straightforward: a larger σ implies a wider Gaussian filter and greater smoothing [JKS95]. In order to discuss the influence of Gaussian smoothing with different degree of smoothing on the experimental results later in Sec. 5.4.3, we selecte two value-intervals for "*sigma*", namely $(0.3, 1.2]$ and $(0.6, 1.5]$. Moreover, the examples of Gaussian smoothing are shown in Fig. 4.2.5.

5 EXPERIMENTS

In word spotting, the goal is to retrieve all relevant instances concerning a given query. All retrieved elements are sorted based on their similarity to a given query. We generate synthetic databases to train our models. During generating synthetic data, we have the following questions:

- Does the size of the vocabulary affect the performance of the model?

  As explained in Sec. 4.1.1, we have defined six vocabularies with different sizes, which focus on the most commonly used n-words: the vocabulary of size 10k also contains all the words in the vocabulary of 5k words. The question is, does a more extensive vocabulary contribute more to improving model performance? Sec. 5.4.1 chooses to find the answer by conducting different experiments and then comparing the results.

- Will the number of fonts used to render word images affect the performance of the model?

  We use the font-based method to generate word images. A large number of free handwritten fonts make the synthesized word images have a variety of handwriting styles. Therefore, the relationship between the number of fonts used and the model's performance is our concern. In order to explore this issue, we generated multiple synthetic databases different numbers of fonts. Sec. 5.4.2 will give the experimental results for this problem.

- Which deformations applied to synthetic data will result in significant performance improvement?

  Experiment [JSVZ14] shows that as more sophisticated synthetic training data of natural scene text is used, the models' recognition accuracy increases. Inspired by [JSVZ14], in Sec. 5.4.3 and Sec. 5.4.4, we will analyze the impact of several deformation methods we used on the performance of the PHOCNet for word spotting.

Sec. 5.1 introduces first the two real data sets used in the experiment. For experimentation, we evaluate the trained model according to the protocol detailed in Sec. 5.2. The details of the training are described in Sec. 5.3. Finally, the obtained results are presented and discussed in Sec. 5.4 and compared with the literature results in Sec. 5.5.

5.1 DATASETS

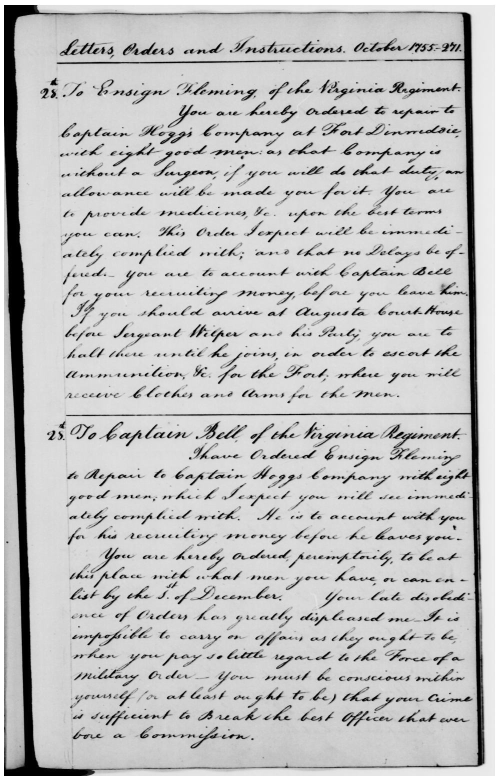
Besides our synthetic datasets, two standard databases are also used for the evaluation of our systems, namely George Washington and IAM database. While the IAM is a contemporary data set, the George Washington is a historic one. In the following, the two datasets will be explained in detail, and examples in them will be shown in Fig. 5.1.1.

5.1.1 *George Washington*

The George Washington database (GW) was created from the George Washington Papers at the Library of Congress[1] and has become the standard benchmark for word spotting [GSGN17]. It contains a total of 4,894 English words from 20 pages of letters between George Washington and his colleagues in 1,755. The text images in this data set show little variability in writing style.

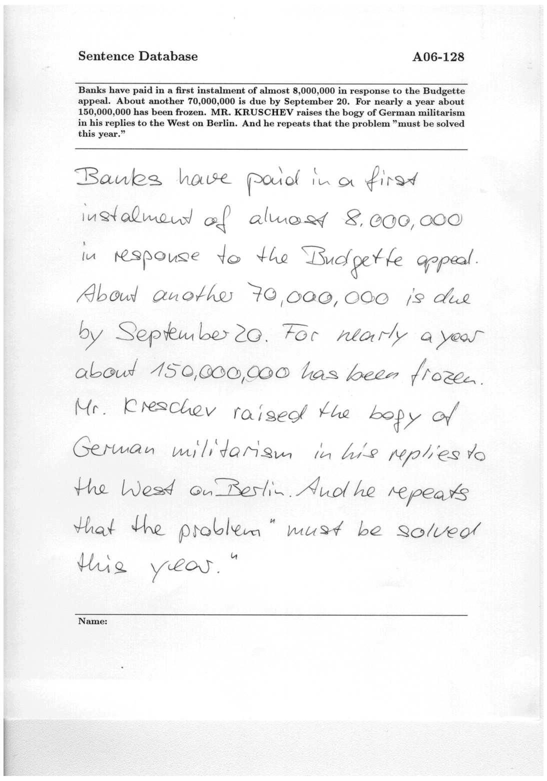
In GW, there is not an official training and testing partition. Like [GSF18], we implemented the four-fold cross-validation proposed in [AGFV14]. In 4-fold cross-validation, the original dataset is randomly divided into four equal-sized partitions. Among the four partitions, one partition is employed as the test set, and the remaining three partitions are used as training sets. Then, repeat this process four times. To compare our results with those in [GSF18] and [AGFV14], we use the same cross-validation partitions[2]. In this work, all GW-related results shown are the average of the results from the four folds.

1 https://www.loc.gov/collections/george-washington-papers/about-this-collection/
2 Cross-validation splits are available at https://github.com/almazan/watts/tree/master/data


Letters, Orders and Instructions. October 1755-271.

28th. To Ensign Fleming, of the Virginia Regiment.
You are hereby Ordered to repair to
Captain Hoggs Company at Fort Dinwiddie,
with eight good men: as that Company is
without a Surgeon, if you will do that duty, an
allowance will be made you for it. You are
to provide medicines, &c. upon the best terms
you can. This Order I expect will be immedi-
ately complied with; and that no Delays be of-
fered. you are to account with Captain Bell
for your recruiting money, before you leave him.
If you should arrive at Augusta Court-House
before Sergeant Wilper and his Party, you are to
halt there until he joins, in order to escort the
Ammunition, &c. for the Fort; where you will
receive Clothes and Arms for the men.

28th. To Captain Bell, of the Virginia Regiment.
I have Ordered Ensign Fleming
to Repair to Captain Hoggs Company with eight
good men; which I expect you will see immedi-
ately complied with. He is to account with you
for his recruiting money before he leaves you.
You are hereby ordered, peremptorily, to be at
this place with what men you have, or can en-
list by the 1st of December. Your late disobedi-
ence of Orders has greatly displeased me. It is
impoſsible to carry on affairs as they ought to be,
when you pay so little regard to the Force of a
Military Order. You must be conscious within
yourself (or at least ought to be) that your Crime
is sufficient to Break the best Officer that ever
bore a Commiſsion.


Sentence Database A06-128

Banks have paid in a first instalment of almost 8,000,000 in response to the Budgette appeal. About another 70,000,000 is due by September 20. For nearly a year about 150,000,000 has been frozen. MR. KRUSCHEV raises the bogy of German militarism in his replies to the West on Berlin. And he repeats that the problem "must be solved this year."

Banks have paid in a first
instalment of almost 8,000,000
in response to the Budgette appeal.
About another 70,000,000 is due
by September 20. For nearly a year
about 150,000,000 has been frozen.
Mr. Kreschev raised the bogy of
German militarism in his replies to
the West on Berlin. And he repeats
that the problem " must be solved
this year. "

Name:

Figure 5.1.1: Samples handwritten text images from GW (left) and IAM (right).

5.1.2 *IAM*

The IAM Off-line Handwriting Database [MB02] is one of the most widely used public handwritten text datasets in the document analysis. It contains 1,539 pages of scanned text and more than 100K labeled words written by 657 different authors[3].

IAM has officially defined training and test sets. The text lines of these two sets are different from each other, which means that each writer only contributed to one set. In [GSF18], Gurjar *et al.* have also adopted this official training and test partitioning. In order to straight compare our results with results in [GSF18], this official partition is used in this thesis as well.

3 IAM Handwriting Database is available at http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

5.2 EVALUATION PROTOCOL

According to the protocol in [GSF18] and [AGFV14], the PHOCNet is evaluated under segmentation-based QbE and QbS word spotting scenarios. We are training only with no manually annotated synthetic training data. The PHOCNet is trained for $80,000$ iterations on each of our synthetic datasets, similar in [GSF18].

For QbE, each word image in the corresponding test partition of the handwritten data sets is used once as the query, while all remaining images in test partition are regarded as the retrieval set. The PHOCNet predicts the PHOC representation for the given query as well as word images in the retrieval set, and then the images from the retrieval set are sorted according to the cosine distance between its predicted PHOC representation and the predicted representation of the query. The queries that have no related words in the retrieval set will be discarded; however, they will remain for other queries.

When performing QbS, each string that appears in the test set is used as a query only once, even if it appears multiple times, while the entire test partition is used as the retrieval set. Like QbE, the PHOC representation of the query string is used as the basis for ranking the representations in the retrieval set.

As in [GSF18] and [AGFV14], the entire retrieval set is returned in the search ranking list in both QbE and QbS, so the recall rate for each query is always 100%. We use mean Average Precision (mAP) as the accuracy measure for both QbE and QbS, which is a standard measure for retrieval task performance (cf. [ZZ09], p. 1691). The mAP of a set of queries refers to the mean of the average precision scores

$$AP = \frac{\sum_{i=1}^{n} P(i) \cdot r(i)}{\text{number of relevant elements}} \tag{5.2.1}$$

of each query, where i is the index in the ranked retrieval list, n is the number of retrieved elements, $P(i)$ is the precision

$$P = \frac{|\{\text{relevant elements}\} \cap \{\text{retrieved elements}\}|}{|\{\text{retrieved elements}\}|} \tag{5.2.2}$$

of the top-i retrieved elements, and $r(i)$ is an indicator function equaling 1 if the item at index i is a relevant element as well as 0 otherwise (cf. [ZZ09], p. 98).

5.3 TRAINING DETAILS

To compare our results to others reported in literatures (e.g., [GSF18] , [WBF20] and [WF20].), we use the same parameters for training. The stochastic gradient descent method (see Sec. 2.2.2) is used to train the PHOCNet, and each batch contains ten training samples. The momentum is equal to 0.9, and the weight decay is $5 \cdot 10^{-5}$. The learning rate for the first $70,000$ iterations is 10^{-4}, and for the last $10,000$ iterations is 10^{-5}. Furthermore, the initial weights of the PHOCNet are randomly sampled from a uniform distribution with mean 0 and standard deviation $\frac{2}{n}$, where n is the number of parameters in a given layer. Moreover, layer biases are initialized to 0.

5.4 RESULTS AND DISCUSSION

To elucidate the questions raised at the beginning of this chapter, this section will give corresponding experimental results for different questions, and then discuss these based on the obtained results. In the experiment, the training for the PHOCNet uses only the methods in Chap. 4 to create synthetic data sets, and then the test data from each real data set are used to evaluate the trained models.

5.4.1 *The Size of Vocabulary*

The experiment's purpose is to investigate the effect of vocabulary size on model performance. To this end, we use six vocabularies of varying sizes to create six corresponding synthetic data sets. As reported in Sec. 4.1.1, these six vocabularies we generated contain 5k, 10k, 20k, 25k, and 30k, respectively. For the six synthetic data sets, except for the different vocabulary used, the rest of the generation process is the same (see Sec. 4.1.2). We sample a font from 410 fonts and render the corresponding undistorted image for each word from the vocabulary. In this way, we generate 100 images for each word class. To make the six synthetic data sets comparable, we only generate images for word classes that do not yet have images. For example, when generating the synthetic dataset with the vocabulary with a size of 10k, we only render images for 5k new words, and the other half of this dataset inherits all word images in the dataset generated using 5k words.

Table 5.4.1: Results for experiments with different vocabularies in mAP [%]

Size of Vocabulary	IAM		GW	
	QbE	QbS	QbE	QbS
5,000	27.82	37.97	29.64	45.59
10,000	28.67	40.66	30.97	45.18
15,000	27.69	40.99	29.78	46.08
20,000	27.39	36.66	28.91	45.18
25,000	27.49	40.60	28.77	44.93
30,000	27.56	39.73	29.18	44.65

Tab. 5.4.1 compares the results obtained training the PHOCNet on synthetic data sets generated with vocabularies of different sizes. In the table, we do not see the performance improvement of models as the vocabulary size increases. Our observation is that there is no one clear winner in these six synthetic data sets. The performance of the PHOCNet trained on these six synthetic databases is almost the same.

One fact that can be discerned is that vocabulary with more words does not provide any advantage to the synthetic data set. As described in Sec. 4.1.1, although the six vocabularies we made are of different sizes, they all only concentrate on the most frequently occurring words. It means that the 5,000 words with the highest frequency appear in all six vocabularies.

As can be seen in Tab. 5.4.1, compared with the results of the synthetic data set containing 5,000 words, the synthetic data set containing 30,000 words does not get outstanding results in the experiment both on IAM and GW, even though the size of vocabulary has been drastically increased to 30k words. So we have reason to believe that even in such an extensive vocabulary containing 30k words, only the 5k words with the highest frequency contribute to the experimental results.

Table 5.4.2: Results for experiments with different number of Fonts in mAP [%]

Number of Fonts	IAM		GW	
	QbE	QbS	QbE	QbS
100	22.16	30.38	22.82	33.34
200	24.77	34.78	26.91	39.87
300	26.54	37.71	29.76	44.11
400	**28.55**	**40.03**	**30.19**	**44.81**

5.4.2 *The Number of Fonts*

This experiment intends to explore whether the number of handwritten fonts used to render word images impacts the performance of the model. Based on the rendering method detailed in Sec. 4.1.2, we randomly sample four subsets of fonts with a size of 100, 200, 300, and 400 from 410 different handwritten fonts to render images. In this experiment, for each synthetic dataset, we use one font set of them to generate ten undistorted images for each word class in the vocabulary of 10k words. The percentages of mAP scores for experiments conducted on these synthetic databases are listed in Tab. 5.4.2. The best results are shown in bold.

We can draw several conclusions from our experiment. Firstly, according to the results in Tab. 5.4.2, we observe that using synthetic data sets generated with more different fonts to train the PHOCNet can make the CNN have relatively better performance results on the historical data sets. Secondly, as can be seen in Fig. 5.4.1, the performances on the IAM dataset increase steadily with the increase in the number of fonts: every additional 100 fonts used when generating synthetic data can increase the performance by 2% on QbE, and by almost 3% on QbS, while on GW, a significant performance growth tends to be flat, after the number of fonts increases to 300. This difference may be due to not the same generation process of the two traditional data sets. As introduced in Sec. 5.1, more than 600 writers jointly complete IAM, while GW is regarded as a data set generated by a single author, so the diversity of writing styles in IAM is a big challenge, but its influence in GW is limited.

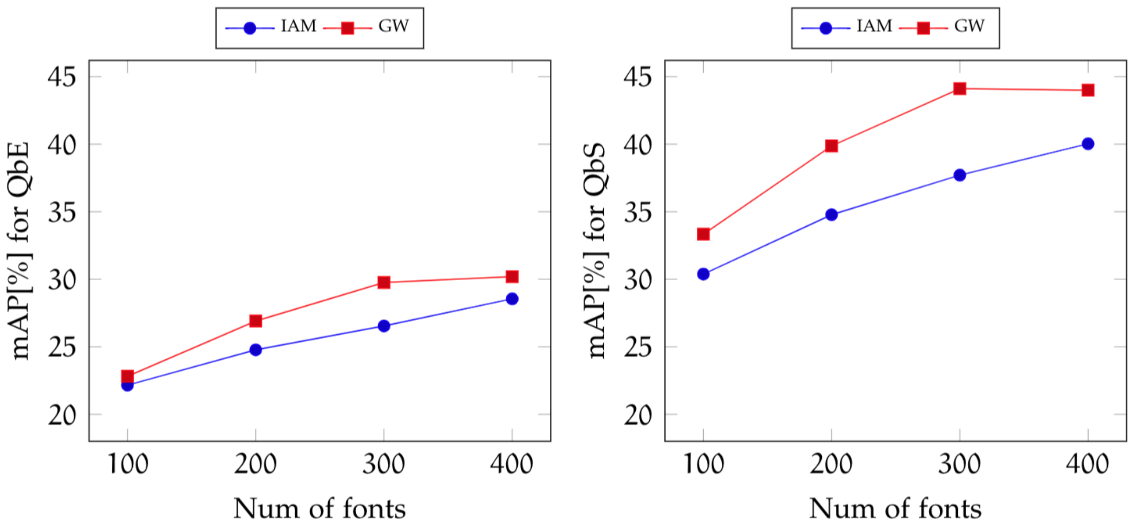


Figure 5.4.1: The mAPs of the PHOCNets trained on synthetic data sets with different number of fonts on the IAM and GW.

5.4.3 *Pixel Distribution of Fore- and Background Pixels*

As described in Sec. 4.2.3, in order to make the pixel distribution of both foreground and background pixels of the synthesized word images more real, we learn the parameters related to the pixel distribution from two real data sets, IAM and GW, and use them to adjust the pixel distribution of an undistorted synthetic data set that uses 410 fonts and contains 100 images for each word from the vocabulary of size 10k.

Tab. 5.4.3 shows the results of the PHOCNet using two synthetic data sets with different pixel distribution as the training set. Besides, the table also lists the results corresponding to the synthesized data set without distortion to compare them with the results of the data sets applied Gaussian distribution of fore- and background pixels. Comparing the results obtained from the PHOCNet trained on two synthetic data sets with different pixel distribution allows for assessing the suitability of the pixel distribution of both foreground and background pixels learned from IAM compared to the learned from GW as both synthetic data sets differ only in this respect.

Table 5.4.3: Results for experiments with different Gaussian distribution of fore- and background pixels in mAP [%].

Pixel Distribution learned from	IAM		GW	
	QbE	QbS	QbE	QbS
None	28.67	40.66	30.97	45.18
IAM	**38.88**	**58.09**	**52.41**	**60.74**
GW	37.15	55.10	47.26	59.40

As shown in Tab.5.4.3, after naturalizing the foreground and background pixels of the word images in the synthetic data set, the distorted synthetic data obtained better results than the original synthetic data. It shows that this method can effectively narrow the gap between the synthesized word images and the real samples.

Another exciting aspect is that the data set only with IAM pixel distribution performs better than the with GW pixel distribution, both in the experiment on IAM and the GW. This is probably due to the few real samples available for learning pixel distribution in GW. As we have already introduced in Sec. 5.1, the IAM dataset contains more than 100k word images, while there are less than 5,000 word images in GW. There is a too small number of samples available to not learn a universal pixel distribution from GW. Therefore, our point of view is that the parameters summarized from IAM are more valuable than those summarized from GW when performing Gaussian distribution processing for the pixels of the synthetic word images.

5.4.4 *Synthetic Data Sets with Deformations*

This experiment studies the contribution of the various stages of the synthetic data generator to the word spotting model's performance. All generation stages have been detailed in Chap. 4. In this experiment, we used synthetic data with different sophistication levels, where each sophistication level corresponds to one

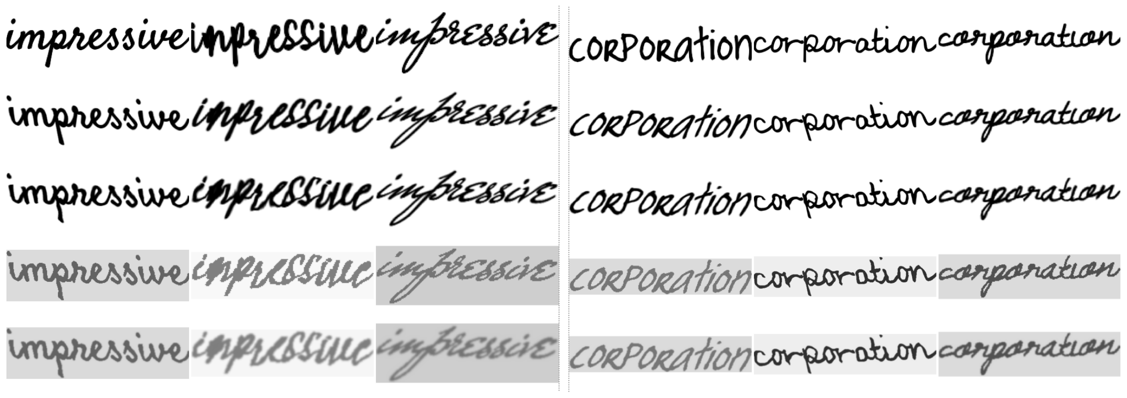

(a) (b)

Figure 5.4.2: Comparison of generated synthetic images with different levels of sophistication. (a) and (b) represent the same word in three different handwritten fonts. From top to bottom, each row corresponds to level 1-5 of sophistication of the synthetic data, respectively.

or several generation stages. Level 1: Word with black ink rendered on a white background with a handwritten font. Level 2: Adding affine transformation. Level 3: Adding elastic distortion. Level 4: Applying the normal distribution of fore- and background pixels learned from the IAM data set. Level 5: Adding smoothing by Gaussian filtering (a low-smoothing with $\sigma \in (0.3, 1.2]$ or a normal smoothing with $\sigma \in (0.6, 1.5]$). In this experiment, we choose the undistorted synthetic data generated in Sec. 5.4.3 as the 1-level data. Fig. 5.4.2 shows some sample rendered word images with different levels of sophistication.

Tab. 5.4.4 displays the mAP results obtained by using purely synthetic data with different distortion levels to train the PHOCNet. Besides, Fig. 5.4.3 shows the changing trend of the PHOCNet performance when trained with increasing levels of sophistication of synthetic training data.

It can be seen from Tab. 5.4.4 and Fig. 5.4.3 that the affine transformation causes a notable performance drop on IAM in both QbE and QbS scenarios, but it also gives a nearly 4% additional performance on the GW. The most likely reason for this is that GW, as a historical data set generated by a single writer, has a distinct and uniform writing style. As shown in Fig. 5.1.1, the words in GW have a pronounced rightward tilt caused by the writer's inherent writing style. Our

Table 5.4.4: Comparison of results obtained by using purely synthetic data with different distortion levels to train the PHOCNet in mAP [%]. Both two sets of fifth-level synthetic data perform Gaussian smoothing on the synthetic data of level 4. The only difference between them is that they use different smoothing intensity intervals ($\sigma \in (0.3, 1.2]$ for low-smoothing, and $\sigma \in (0.6, 1.5]$ for normal smoothing).

Levels of sophistication of synthetic data	IAM		GW	
	QbE	QbS	QbE	QbS
1	28.67	40.66	30.97	45.18
2	22.24	30.38	34.53	49.58
3	34.77	48.05	40.16	54.33
4	**44.70**	**65.19**	**60.40**	**69.02**
5 (low-smoothing)	34.75	57.66	58.06	67.79
5 (normal smoothing)	32.49	57.79	58.31	66.00

synthetic data generator uses the shearing to simulate this handwriting tilt caused by a cursive script. Moreover, all pages in GW are scanned from a book. The scanning process made a noticeable warping of the entire page. The rotation operation can good simulate this kind of paper warping. Therefore, the affine transformation can improve the performance on the GW dataset. However, IAM is composed of document images showing contemporary style English handwritten text. Although the words in IAM present various writing styles, most of them are not scribbled, so they do not show apparent tilt. Therefore, shearing may be an interference operation for synthetic data in the experiment on a modern data set. When making the IAM data set, the writers write on the separate piece of paper, and then each piece of written paper is flattened and scanned. So the pages remain very flat during the process of collecting text images; this may make the rotation operation irrelevant.

One aspect worthy of attention is that both elastic deformation and naturalization of pixel distribution cause a significant increase in performance. As the

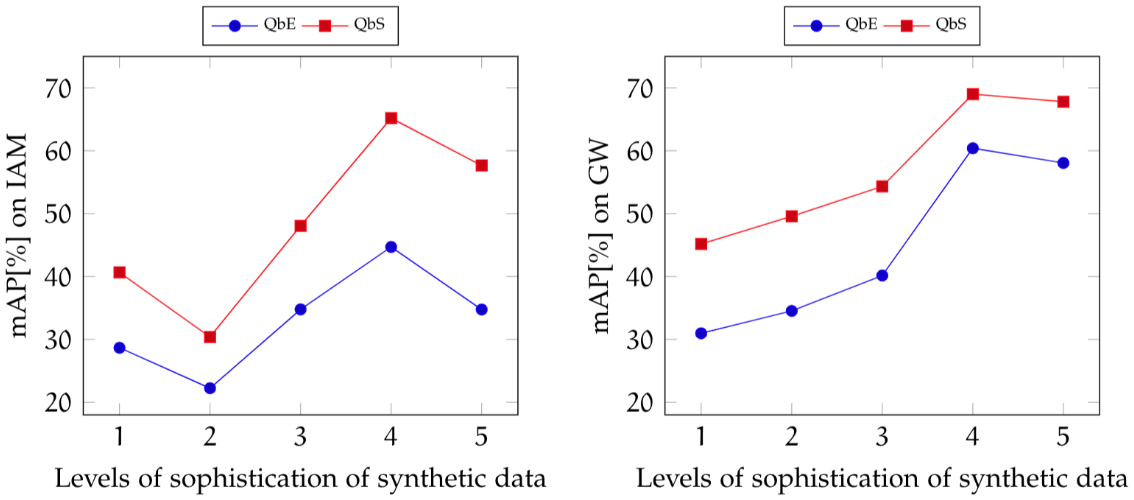


Figure 5.4.3: The performance of the PHOCNet models evaluated on IAM (left) and GW (right) respectfully. The models are trained on purely synthetic data with level 1 to 5 of sophistication of the synthetic data. The fifth level of synthetic data shown in this figure uses slight smoothing ($\sigma \in (0.3, 1.2]$).

training data with the 4th level of sophistication of synthetic data, the performance reaches the peak. In the past, elastic deformation in [SSP03] has been successfully used for recognizing handwritten digits. This experiment proves that it can also play a positive role in word spotting tasks.

Last but not least, after adding Gaussian smoothing, there is a sharp performance degradation on IAM, but only a very slight decline on GW. To study the adverse effect of Gaussian smoothing on performance, we compared the results of two sets of five-level synthetic data using different smoothness. When the smoothing reduces lightly, there is almost no change in performance on GW, while the result for QbE on IAM has slightly improved. Even with such a slight improvement, the performance result of 5-level synthetic data with low-smoothing is still 10% behind the highest point of performance. This situation may be because the quality of text images in IAM and GW is significantly different. Fig. 5.1.1 illustrates this difference well: GW exhibits apparent aging phenomena that do not appear in IAM, such as fading ink and bleeding. Hence, the blurring effect produced by Gaussian smoothing will have a more clear adverse effect on the performance on IAM than the on GW.

5.5 COMPARISON TO RESULTS FROM THE LITERATURE

In this section, to compare with the state-of-the-art, we only focus on the 4-level synthetic data obtained the best results in previous experiments, generated with all distortion methods except Gaussian smoothing. Tab. 5.5.1 displays the mAP results obtained by using different purely synthetic data to train models. One of the results used for comparison is from the method in [GSF18] using the synthetic data set HW-SYNTH [KJ16b], and the other is from the in [WF20] using IIIT-HWS dataset [GSF18]. Besides, the best results so far are reported in [WBF20], where the authors combined the synthesis approach with further augmentation techniques.

Compared with [GSF18] and [WF20], our method obtains more competitive results both on the GW and IAM dataset. The difference in vocabulary is that HW-SYNTH and IIIT-HWS respectively selected 10k and 90k unique words both from the Hunspell dictionary, and our vocabulary only contains the $10,000$ most frequently occurring English words based on the Exquisite Corpus [SCL^{+}18]. In the process of generating synthetic images, we additionally use elastic distortion compared with HW-SYNTH and IIIT-HWS. These may be the reasons for the big difference in results.

Although the synthetic data generated by Wolf *et al.* in the literature [WBF20] obtained ideal results on GW, we can still achieve better performance on IAM. The source of the vocabulary we use is the same as that in [WBF20], and we also use the most common $10,000$ English words. The method we use to generate synthetic word images is also similar to the method described in [WBF20], but there are still some differences. We use 410 different handwritten fonts to render word images, but only 134 fonts are used in the method of Wolf *et al.*. As we have discussed in Sec. 5.4.2, the experimental results are proportional to the number of handwritten fonts limited to 410 used. The demand for diversification of fonts is particularly evident on the IAM. Besides, for the pixel distribution of synthetic images, the method in [WBF20] is to select the values of the back- and foreground pixels randomly from the uniform distributions limited with $[180, 255]$ and $[0, 100]$, that are significantly different from the pixel distributions learned from the IAM

Table 5.5.1: Comparison with the literature. Results reported as mAP [%]. Best results are marked in bold.

Method	IAM		GW	
	QbE	QbS	QbE	QbS
Ours	**44.70**	**65.19**	60.40	69.02
Gurjar *et al.* [GSF18]	26.21	36.57	39.89	48.92
Wolf *et al.* [WF20]	16.00	39.50	46.60	57.90
Wolf *et al.* [WBF20]	39.00	64.10	**69.20**	**72.30**

data set shown in Fig. 4.2.4. These factors probably cause the advantages of our method on the IAM.

We also noticed that the method in [WBF20] randomly selects a slant angle of $[-40, -20, 0, 20, 40]$ for each synthetic word image. However, our method is to adopt a random amount of shear ($+/-0.5$ degrees along horizontal direction), equivalent to a random slant angle almost between -26 and 26 degrees. From Fig. 5.1.1, we can see that the text in GW has a vast slant angle. Accordingly, the slant angles set in the literature are more suitable for GW than us. However, the excessively large slant angles may adversely affect the performance of synthetic data on the IAM.

6 CONCLUSION

This thesis proposes a framework for generating large-scale synthetic data by rendering handwritten word images. Inspired by [KJ16a], this work extends the font-based method of rendering words, and we add some natural variations in the handwriting domain to the rendered images. To generate highly realistic synthetic data sufficient to replace actual data, we introduce four artificial distortion schemes: Affine Transformation, Elastic Distortion, Gaussian Distribution, and Gaussian Smoothing. A large amount of synthetic data eases manual annotation efforts for CNN-based word spotting tasks.

This thesis's core is to generate multiple synthetic datasets with different sophistication levels of synthetic training data and use them to evaluate the contribution of each stage of the synthesis process to the model performance. Experimental evaluation shows that using more different handwritten fonts to render words can make the synthetic data set obtain better results, especially on IAM. We used up to 410 handwritten fonts in this work. In future work, we plan to employ more handwritten fonts, such as 657, which is the same as the number of writers generating the IAM dataset.

As shown in the experimental evaluation, the pixel distribution learned from IAM has better general adaptability than that learned from GW. In future experiments, we plan to continue using the parameters learned from IAM to naturalize the pixels of the synthesized images. We have also proved that the size of the vocabulary only focusing on the most frequently occurring words has no significant effect on performance results.

The experimental evaluation also shows that elastic distortion and Gaussian distribution of fore- and background pixels achieve significant performance growth in the experiments both on IAM and GW. Although affine transformation leads to a model performance decline in the experiment on IAM, it still plays a positive role in the experiment on GW. On the contrary, Gaussian smoothing causes performance degradation in all experiments, and this degradation is particularly prominent on the IAM database.

We have proved through experiments that our 4-level synthetic data is very suitable as pretraining data for word spotting models. In the literature [GSF18], a PHOCNet only uses $1,000$ labeled samples and mainly relies on synthetic data HW-SYNTH for pretraining, which can achieve excellent performance. Therefore, we can make a reasonable conjecture: if using our synthetic data set to pre-train the PHOCNet, the demand for real data can be further reduced. We plan to prove this conjecture in future work.

BIBLIOGRAPHY

[AF15] Ahmad, Irfan ; Fink, Gernot A.: Training an Arabic handwriting recognizer without a handwritten training data set. In: *International Conference on Document Analysis and Recognition, ICDAR*, 2015, S. 476–480

[AFM17] Athanasiou, Lambros S. ; Fotiadis, Dimitrios I. ; Michalis, Lampros K.: *Propagation of Segmentation and Imaging System Errors*. 2017. – 151 – 166 S.

[AGFV14] Almazán, Jon ; Gordo, Albert ; Fornés, Alicia ; Valveny, Ernest: Word Spotting and Recognition with Embedded Attributes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36 (2014), Nr. 12, S. 2552–2566

[AR20] Akhtar, Nadeem ; Ragavendran, U.: Interpretation of intelligence in CNN-pooling processes: a methodological survey. In: *Neural Computing and Applications* 32 (2020), Nr. 3, S. 879–898

[ARTL13] Aldavert, David ; Rusiñol, Marçal ; Toledo, Ricardo ; Lladós, Josep: Integrating Visual and Textual Cues for Query-by-String Word Spotting. In: *International Conference on Document Analysis and Recognition, ICDAR*, 2013, S. 511–515

[BB07] Bottou, Léon ; Bousquet, Olivier: The Tradeoffs of Large Scale Learning. In: *Advances in Neural Information Processing Systems 20, NIPS*, 2007, S. 161–168

[BIK$^+$20] Buslaev, Alexander ; Iglovikov, Vladimir I. ; Khvedchenya, Eugene ; Parinov, Alex ; Druzhinin, Mikhail ; Kalinin, Alexandr A.: Albumentations: Fast and Flexible Image Augmentations. In: *Information* 11 (2020), Nr. 2, S. 125

[BPC$^+$17] Bhattacharya, Ujjwal ; Plamondon, Réjean ; Chowdhury, Souvik D. ; Goyal, Pankaj ; Parui, Swapan K.: A sigma-lognormal model-based approach to generating large synthetic online handwriting sample databases. In: *International Journal on Document Analysis and Recognition, IJDAR* 20 (2017), Nr. 3, S. 155–171

[BSH17] Bloice, Marcus D. ; Stocker, Christof ; Holzinger, Andreas: Augmentor: An Image Augmentation Library for Machine Learning. In: *Journal of Open Source Software* 2 (2017), Nr. 19, S. 432

[BW17] Balreira, Dennis G. ; Walter, Marcelo: Handwriting Synthesis from Public Fonts. In: *SIBGRAPI Conference on Graphics, Patterns and Images*, 2017, S. 246–253

[Com18] Commons, Wikimedia: *File:Normal Distribution PDF.svg — Wikimedia Commons, the free media repository*. https://commons.wikimedia.org/w/index.php?title=File:Normal_Distribution_PDF.svg. Version: 2018

[Com20] Commons, Wikimedia: *File:2D affine transformation matrix.svg — Wikimedia Commons, the free media repository*. https://commons.wikimedia.org/w/index.php?title=File:2D_affine_transformation_matrix.svg. Version: 2020

[DM20] Das, Sugata ; Mandal, Sekhar: Segmentation-free word spotting in historical Bangla handwritten document using Wave Kernel Signature. In: *Pattern Analysis and Applications* 23 (2020), Nr. 2, S. 593–610

[DW14] Dasgupta, Amitava ; Wahed, Amer: Chapter 4 - Laboratory Statistics and Quality Control. In: *Clinical Chemistry, Immunology and Laboratory Quality Control*. 2014, S. 47 – 66

[Gra13] Graves, Alex: Generating Sequences With Recurrent Neural Networks. In: *CoRR* abs/1308.0850 (2013)

[GSF18] Gurjar, Neha ; Sudholt, Sebastian ; Fink, Gernot A.: Learning Deep Representations for Word Spotting under Weak Supervision. In: *IAPR International Workshop on Document Analysis Systems, DAS*, 2018, S. 7–12

[GSGN17] Giotis, Angelos P. ; Sfikas, Giorgos ; Gatos, Basilis ; Nikou, Christophoros: A survey of document image word spotting techniques. In: *Pattern Recognition* 68 (2017), S. 310–332

[GW19] Goodman, Dou ; Wei, Tao: Cloud-based Image Classification Service Is Not Robust To Simple Transformations: A Forgotten Battlefield. In: *CoRR* abs/1906.07997 (2019)

[HAB16] Haines, Tom S. F. ; Aodha, Oisin M. ; Brostow, Gabriel J.: My Text in Your Handwriting. In: *ACM Transactions on Graphics* 35 (2016), Nr. 3, S. 26:1–26:18

[HSM+00] Hahnioser, Richard H.R. ; Sarpeshkar, Rahul ; Mahowald, Misha A. ; Douglas, Rodney J. ; Seung, Hyunjune Sebastian: Digital selection and analogue amplification coexist in a cortex- inspired silicon circuit. In: *Nature* 405 (2000), Nr. 6789, S. 947–951

[HZRS14] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: *European Conference on Computer Vision, ECCV* Bd. 8691, 2014, S. 346–361

[HZRS16] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016, S. 770–778

[JBMN09] Jawahar, C. V. ; Balasubramanian, A. ; Meshesha, Million ; Namboodiri, Anoop M.: Retrieval of online handwriting by synthesis and matching. In: *Pattern Recognition* 42 (2009), Nr. 7, S. 1445–1457

[JKS95] Jain, R. ; Kasturi, R. ; Schunck, B.G.: *Machine Vision*. 1995. – 112–139 S.

[JSVZ14] Jaderberg, Max ; Simonyan, Karen ; Vedaldi, Andrea ; Zisserman, Andrew: Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition. (2014)

[KDJ16] Krishnan, Praveen ; Dutta, Kartik ; Jawahar, C. V.: Deep Feature Embedding for Accurate Recognition and Retrieval of Handwritten Text. In: *International Conference on Frontiers in Handwriting Recognition, ICFHR*, 2016, S. 289–294

[KJ16a] Krishnan, Praveen ; Jawahar, C. V.: Generating Synthetic Data for Text Recognition. In: *CoRR* abs/1608.04224 (2016)

[KJ16b] Krishnan, Praveen ; Jawahar, C. V.: Matching Handwritten Document Images. In: *European Conference on Computer Vision, ECCV*, 2016, S. 766–782

[KJ19] Krishnan, Praveen ; Jawahar, C. V.: HWNet v2: an efficient word image representation for handwritten documents. In: *International Journal on Document Analysis and Recognition, IJDAR* 22 (2019), Nr. 4, S. 387–405

[KSH17] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet classification with deep convolutional neural networks. In: *Communications of the ACM* 60 (2017), Nr. 6, S. 84–90

[LSP06] Lazebnik, Svetlana ; Schmid, Cordelia ; Ponce, Jean: Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, 2006, S. 2169–2178

[MB02] Marti, Urs-Viktor ; Bunke, Horst: The IAM-database: an English sentence database for offline handwriting recognition. In: *International Journal on Document Analysis and Recognition* 5 (2002), Nr. 1, S. 39–46

[MP87] Minsky, Marvin ; Papert, Seymour: *Perceptrons - an introduction to computational geometry*. MIT Press, 1987

[Mur05] Murakoshi, Kazushi: Avoiding overfitting in multilayer perceptrons with feeling-of-knowing using self-organizing maps. In: *Biosystems* 80 (2005), Nr. 1, S. 37 – 40

[NA08] Nixon, Mark ; Aguado, Alberto S.: *Feature Extraction Image Processing*. 2008. – 113 S.

[NAK+17] Niaz, Hafiz A. ; Akram, M. U. ; Khan, Muazzam A. ; Usman, Anam ; Rafique, Awais: A Study on Word Spotting Techniques for Document Image Analysis. In: *International Conference on Multimedia Systems and Signal Processing, ICMSSP*, 2017, S. 17–21

[NH10] Nair, Vinod ; Hinton, Geoffrey E.: Rectified Linear Units Improve Restricted Boltzmann Machines. In: *International Conference on Machine Learning ,ICML*, 2010, S. 807–814

[Nie19] Nielsen, Michael A.: *Neural Networks and Deep Learning*. 2019

[PG17] Patterson, J. ; Gibson, A.: *Deep Learning: A Practitioner's Approach*. 2017. – 250–255 S.

[PSM10] Perronnin, Florent ; Sánchez, Jorge ; Mensink, Thomas: Improving the Fisher Kernel for Large-Scale Image Classification. In: *European Conference on Computer Vision, ECCV* Bd. 6314, 2010, S. 143–156

[PW16] Poznanski, Arik ; Wolf, Lior: CNN-N-Gram for Handwriting Word Recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition,CVPR*, 2016, S. 2305–2314

[RATL15] Rusiñol, Marçal ; Aldavert, David ; Toledo, Ricardo ; Lladós, Josep: Towards query-by-speech handwritten keyword spotting. In: *International Conference on Document Analysis and Recognition, ICDAR*, 2015, S. 501–505

[RHW86] Rumelhart, David E. ; Hinton, Geoffrey E. ; Williams, Ronald J.: Learning Internal Representations by Error Propagation. 1986, S. 318–362

[RMC18] Roy, Partha P. ; Mohta, Akash ; Chaudhuri, Bidyut B.: Synthetic data generation for Indic handwritten text recognition. In: *CoRR* abs/1804.06254 (2018)

[Ros58] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. In: *Psychological Review* 65 (1958), Nr. 6, S. 386–408

[SCL$^+$18] Speer, Robyn ; Chin, Joshua ; Lin, Andrew ; Jewett, Sara ; Nathan, Lance: *LuminosoInsight/wordfreq: v2.2*. oct 2018

[SF16] Sudholt, Sebastian ; Fink, Gernot A.: PHOCNet: A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents. In: *International Conference on Frontiers in Handwriting Recognition, ICFHR*, 2016, S. 277–282

[SF18] Sudholt, Sebastian ; Fink, Gernot A.: Attribute CNNs for word spotting in handwritten documents. In: *International Journal on Document Analysis and Recognition, IJDAR* 21 (2018), Nr. 3, S. 199–218

[SS01] Shapiro, L.G. ; Stockman, G.C.: *Computer Vision*. 2001. – 153–154 S.

[SSP03] Simard, Patrice Y. ; Steinkraus, David ; Platt, John C.: Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In: *International Conference on Document Analysis and Recognition, ICDAR*, 2003, S. 958–962

[SW10] Sammut, Claude ; Webb, Geoffrey I.: Normal Distribution. In: *Encyclopedia of Machine Learning*. 2010, S. 731–731

[SZ15] Simonyan, Karen ; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *International Conference on Learning Representations, ICLR*, 2015

[TP19] Thontadari, C. ; Prabhakar, C. J.: Segmentation Free Word Spotting for Handwritten Documents Using Bag of Visual Words Based on Co-HOG Descriptor. In: *International Journal of Information Retrieval Research, IJIRR* 9 (2019), Nr. 2, S. 49–65

[WB16] Wilkinson, Tomas ; Brun, Anders: Semantic and Verbatim Word Spotting Using Deep Neural Networks. In: *International Conference on Frontiers in Handwriting Recognition, ICFHR*, 2016, S. 307–312

[WBF20] Wolf, Fabian ; Brandenbusch, Kai ; Fink, Gernot A.: Improving Handwritten Word Synthesis for Annotation-free Word Spotting. In: *International Conference on Frontiers of Handwriting Recognition, ICFHR*, 2020

[Wer74] Werbos, P. J.: *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Harvard University, Diss., 1974

[WF20] Wolf, Fabian ; Fink, Gernot A.: Annotation-Free Learning of Deep Representations for Word Spotting Using Synthetic Data and Self Labeling. In: *IAPR International Workshop on Document Analysis Systems, DAS*, 2020, S. 293–308

[WRF16] Wieprecht, Christian ; Rothacker, Leonard ; Fink, Gernot A.: Word Spotting in Historical Document Collections with Online-Handwritten Queries. In: *IAPR Workshop on Document Analysis Systems, DAS*, 2016, S. 162–167

[ZZ09] Zhang, Ethan ; Zhang, Yi: *Encyclopedia of Database Systems*. 2009