Scene Text Recognition using Deep Learning

Szenentexterkennung mithilfe von Deep Learning

by

Christian Bartz

A thesis submitted to the
Hasso Plattner Institute
at the University of Potsdam, Germany
in partial fulfillment of the requirements for the degree of

Master of Science in IT-Systems Engineering

Supervisors

Prof. Dr. Christoph Meinel
Dr. Haojin Yang

Internet-Technologies and Systems
Hasso Plattner Institute
University of Potsdam, Germany

August 1, 2016

Abstract

In the last decade large amounts of multimedia data like images and videos have become publicly available. These data artifacts contain a huge amount of semantic metadata that could be used for creating systems that are able to search for content in multimedia data. Because of the availability and the advances in high performance computing hardware the automatic extraction of metadata using techniques based on deep learning has recently gained a lot of attention.

In this work we concentrate on textual metadata (scene text) found in images and videos using Deep Neural Networks (DNNs). Thereby we are solely focusing on recognizing characters from images that only contain a line of text (scene text recognition), that has been cropped from the original image. In the first part of this thesis we introduce the foundations of traditional Optical Character Recognition (OCR) systems and motivate the need for new approaches to OCR. We then provide an overview of the basic concepts of deep neural networks.

In the second part of this thesis we introduce and evaluate different approaches for recognizing text in scene images. We first introduce an approach that uses a Convolutional Neural Network (CNN) to recognize text contained in a text line image. We then introduce several approaches that use Recurrent Neural Networks (RNNs) besides a convolutional feature extractor to further improve the recognition results. Finally we provide an extensive evaluation of the described approaches on standard benchmarks for scene text recognition. We also show that we can use our trained models in an end-to-end scene text recognition system that achieves real-time performance.

Zusammenfassung

Die Menge an frei verfügbaren Multimediadaten, wie Fotos und Videos hat sich im letzten Jahrzehnt drastisch erhöht. Diese Multimediadaten enthalten eine Menge wertvoller Metadaten, die genutzt werden können um diese Datenmengen durchsuchbar zu machen. Durch die Verfügbarkeit von diesen riesigen Datenmengen und einer erheblich gesteigerten Rechenleistung ist es möglich geworden Systeme auf Basis von Deep Neural Networks (DNNs) zu bauen, die in der Lage sind Metadaten automatisch aus Multimediadaten zu extrahieren.

Diese Arbeit befasst sich mit der Extraktion von Text aus Bildern und Videos, unter Zuhilfenahme von DNNs. Dabei befasst sich diese Arbeit nur mit der Erkennung von Buchstaben und Wörtern in Bildern, nicht mit der Detektion von einzelnen Textzeilen in einem Bild. Im ersten Teil der Arbeit stellen wir die Grundlagen von traditionellen Systemen zur optischen Erkennung von Buchstaben (OCR) vor und erklären, warum diese nicht für die Aufgabe, Text aus Fotos und Videos zu extrahieren, geeignet sind. Im weiteren Verlauf geben wir eine Einführung in die Grundlagen von Neuronalen Netzen, die in dieser Arbeit genutzt werden.

Im zweiten Teil dieser Arbeit zeigen und evaluieren wir verschiedene Ansätze um Text in Fotos und Videos erkennen zu können. Dabei zeigen wir zuerst einen Ansatz, der ein Convolutional Neural Network (CNN) nutzt um aus einem gegebenem Eingabebild relevante Features zu extrahieren und diese für die Erkennung der Buchstaben im diesem Bild nutzt. Danach zeigen wir weitere Verfahren, die neben einem CNN auch ein Recurrent Neural Network (RNN) nutzen um die Ergebnisse, die mit einem CNN erzielt werden können weiter zu verbessern. Wir evaluieren alle vorgestellten Ansätze auf standardisierten Benchmarks für die Erkennung von Text aus Fotos. Wir zeigen weiterhin, dass mit den von uns trainierten Modellen ein Ende-zu-Ende System, das in der Lage ist die Detektion und Erkennung von Text in Echtzeit durchzuführen, erstellt werden kann.

Acknowledgments

I would like express my gratitude to my supervisors Professor Dr. Christoph Meinel and Dr. Haojin Yang. I especially want to thank Dr. Yang for his helpful comments, ideas and the fruitful discussions we had about the topic of my master thesis. I further want to thank my colleagues Dimitri Korsch and Hannes Rantzsch for the great times we had in our office and that they have always been there and helped me through the difficult times of writing my master thesis. I also want to thank all people that I have met at the Hasso-Plattner-Institute who made the time of my studies as awesome as it was.

Last but not least I want to thank my parents who have always supported and encouraged me throughout the years.

Vielen Dank! Thank You! — Christian

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

OCR	Optical Character Recognition
CNN	Convolutional Neural Network
DNN	Deep Neural Network
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
CTC	Connectionist Temporal Classification
SGD	Stochastic Gradient Descent
BPTT	Back-Propagation Through Time
LSTM	Long Short-Term Memory
CRF	Conditional Random Field
SVM	Support Vector Machine
k-NN	k-Nearest Neighbor
HOG	Histogram of Oriented Gradient
WDCH	Weighted Direction Code Histogram
ICDAR	International Conference on Document Analysis and Reading
BLSTM	Bidirectional LSTM
PIL	Python Imaging Library
Caffe	Convolutional Architecture for Fast Feature Embedding
ILSVRC14	ImageNet Large-Scale Visual Recognition Challenge 2014
SVT	Street View Text
MSER	Maximally Stable Extremal Region

1. Introduction

In the last decade large amounts of image and video data have become publicly available. Due to the development of smartphones and other wearable devices it is possible for users to create multimedia content at every time and everywhere. Users post their multimedia content on Social media platforms like Flickr[1], Instagram[2] and YouTube[3]. These platforms reach a large amount of different users. YouTube for instance has over a billion users and every day these users watch over hundreds of millions of hours of videos[4] and in 2013 The Verge reported that Flickr has 87 million registered users that upload 3.5 million images every day[5].

All of this data contains a huge amount of semantic information that is not directly available to the users that search for specific content. The only metadata available for users is most often a user defined caption, or category tags provided by users. This information does not cover all semantic metadata contained in an image. Semantic metadata includes objects contained in images and videos, concepts depicted, text and many more pieces of information. It would be possible to extract this metadata by hand but the massive amount of content created makes it infeasible for humans. Therefore the need to automatically extract the semantic metadata from multimedia data arises. Text as a high level semantic feature is a very interesting type of semantic metadata. Text in images and videos can directly depict content shown in the video or image. For instance images taken by Google for Google StreetView[6] contain a lot of text that show signs of restaurants, house numbers or street signs. Extracting the textual content from these images may help to improve results of queries by of users for certain places by also showing these images with text next to the ordinary search results. Extracting text from news videos is also a very interesting area for extracting text as text shown in news videos directly shows information about the content currently displayed in the video.

Besides semantic metadata extraction for improving retrieval systems and search engines it is also interesting to be able to extract text from live video scenes for creating text-to-speech systems that may help blind people to read texts. Being able to recognize text could also be beneficial for autonomous cars that can use the extracted text together with other information to provide more helpful advice to the algorithm controlling the car.

[1] https://www.flickr.com/
[2] https://www.instagram.com
[3] https://www.youtube.com
[4] https://www.youtube.com/yt/press/statistics.html last access: 12/07/2016
[5] http://www.theverge.com/2013/3/20/4121574/flickr-chief-markus-spiering-talks-photos-and-marissa-mayer last access: 12/07/2016
[6] https://www.google.com/intl/en_ALL/maps/streetview/

In this work we focus on recognizing text that can be found in real-world images (scene text) that have been captured by a smartphone camera, a webcam, camera in an autonomous car or any other type of camera. The method for recognizing the text depicted in these images is called OCR. OCR systems for printed documents have been around since the late 1960s and have improved since then to be very good at recognizing simple text with clearly defined foreground and background. Text contained in real-world images like image from Google StreetView do not fulfill these requirements and thus traditional OCR systems fail in recognizing the text contained in these images. In the past five years Deep Learning and DNNs have gained great attention from the computer vision research community. Krizhevsy et al. were able to decrease the error rate of image classification on the ILSVRC-2012 test set by more than 10 % using CNNs [33]. Since then a lot of research work has been done using CNNs. This research has also been applied to the field of OCR where different researchers showed that using deep learning allows great performance gains for their experiments [60, 4, 27, 13, 28, 16, 51]. In this work we will follow the path of applying deep learning for recognizing text from scene text images.

1.1. Contributions

In this thesis we address the problem of applying deep learning methods to scene text recognition. In order to apply deep learning for recognizing text from videos and images we needed a huge amount of training data. In this thesis we introduce a data generation tool that is capable of generating synthetic scene text images that are virtually indistinguishable from real-world data. We show that we can create text recognition models that reach near state-of-the-art accuracies on standard benchmark datasets by maintaining the possibility to perform real-time scene text recognition when integrated into an end-to-end scene text recognition system. Besides that we provide an extensive comparison of convolutional networks and recurrent networks as deep network architectures for scene text recognition.

1.2. Outline of the Thesis

This thesis is organized in the following way:

We will present and introduce the foundations of OCR systems in chapter 2. In section 2.5 we will provide an overview on the related work in the field of scene text recognition.

In chapter 3 we will describe our data generation tool that we used to generate millions of training images for our experiments with different deep network architectures. In chapter 4 and chapter 5 we introduce the network layouts and principal concepts of the different architectures we compare in this work.

In chapter 6 we showcase the experimental results we received while performing different experiments with each described network architecture. In section 6.5 we also showcase the real-time performance of our generated scene text recognition

models. In chapter 7 we conclude this thesis by summarizing our achievements. We also provide a brief outlook on future work in this chapter.

2. Foundations of OCR

In this chapter we will first present the development of OCR through time. We will then provide an overview of the standard OCR techniques. The challenges for traditional print OCR systems will be discussed. Based on these challenges fundamental techniques to overcome these challenges will be introduced. In the last part we will give detailed review of related work in the field of scene text recognition.

2.1. Optical Character Recognition

Making machines able to read has been a dream of humans since ancient times. With the development of digital computers and more and more processing power it has been possible to enable machines to read texts. Until now it was possible to solve this problem for printed document text very well, but with recent advances in machine learning, the availability of big data and fast processing hardware it becomes feasible for computers to also read text from real-world images.

One of the first OCR devices was the Optophone introduced in 1913 by Dr. Edmund Fournier d'Albe [9]. This device used photosensors to detect black print and convert the detection into characteristic sounds for each letter, enabling a blind to read a text. In 1928 Emanuel Goldberg filed a patent for a machine that was able to do pattern recognition on rolls of microfilmed documents, helping users to find documents containing specific data [53].

With the development of digital computers it was possible to process more and more data in shorter time intervals. In order to simplify the input of data special fonts like the American font OCR-A and European font OCR-B have been developed in 1968. These fonts have a simple design which makes it easy for a machine to read these letters. Figure 2.1 shows the font faces of OCR-A and OCR-B.

The first OCR machines were only able to read a very limited set of fonts. Already in 1970 a company called Scan-Data Corporation produced one of the first OCR machines that was able to recognize text written in various fonts and therefore helpful in digitalizing official documents like acts [43].

With computers becoming more powerful and software more important the prices for OCR machines and software dropped drastically and OCR was more widely used. Throughout the 1990s and early years of 2000 OCR for printed documents achieved a mature state and products like ABBYY FineReader[1] or Tesseract[2]

[1] https://www.abbyy.com
[2] https://github.com/tesseract-ocr/tesseract

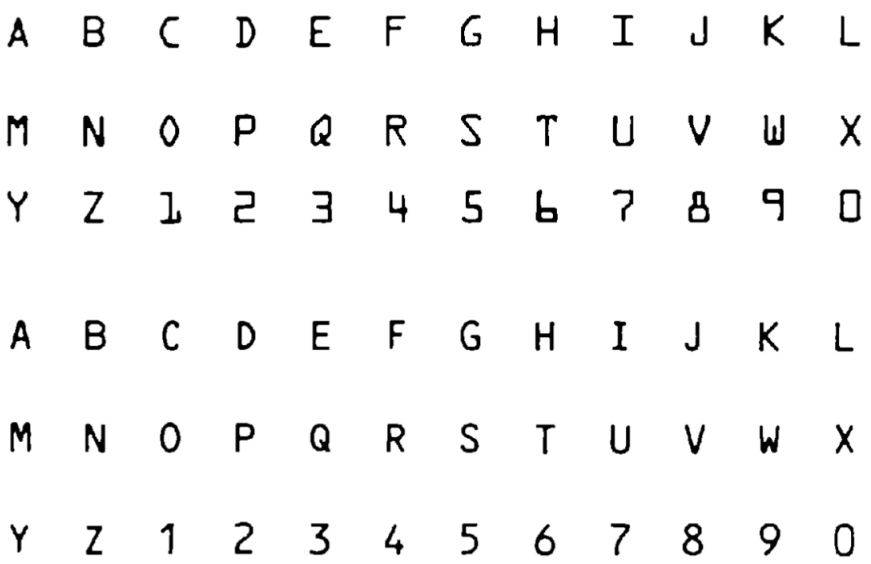


Figure 2.1.: *(top)* OCR-A, *(bottom)* OCR-B (credit [10])

became available. These Print OCR systems are very good for recognizing text in printed documents but they fail in recognizing text seen in natural scenes, see figure 2.2 for an example of text lines typical for real-world images. In order to overcome this issue current research tries to use recent advances in machine learning and artificial intelligence for recognizing scene text. The next section will show the components of a typical OCR system.

2.2. Components of OCR systems

A typical OCR system consists of the following components:

Optical Scanning converts a document into a digital form by using optical document scanners or cameras. In classical Print OCR the document is then converted into a black and white image using binarization technologies. As scanned documents show a large range of different contrasts this step is very crucial as it heavily influences the performance of the recognition engine used. Thanks to the nature of CNNs manual binarization of scanned documents for scene text recognition is not necessary anymore. We will give a detailed introduction into CNNs in section 2.4.

Location and segmentation analyzes the layout of the scanned document. Regions that might contain text are extracted. These regions are then further divided into areas containing only single characters. This localization and segmentation is mostly done using connected component analysis methods like Maximally Stable Extremal Regions (MSERs). In typical scene text recognition systems this step is not always necessary as CNNs are able to perform location and segmentation on their own.

Preprocessing tries to eliminate noise that appears in the segmented areas of text. Besides smoothing processes that eliminate small gaps, breaks and holes in the identified areas, normalization is a widely used preprocessing technique. Normalization aims to produce characters of uniform rotation, slant and size leading to an overall improvement of the recognition accuracy of the used system.

Feature extraction is seen as one of the most challenging steps in OCR that aims at capturing representative traits of characters. The robustness of the extracted features hereby determines the overall performance of the OCR system. Features can be extracted through the following methods: structural analysis of characters, analysis of point distributions, transformations and series expansions. Features can be hand crafted or automatically learned as it is done by CNNs.

Classification takes the extracted features and assigns these features to the most probable character class. This can be done by using techniques like matching, where the similarity between extracted feature vectors and the description of each class is calculated using the euclidean distance. Furthermore physical structure based classifiers that distinguish characters by their characteristic strokes that may have been extracted during the feature extraction process can be used for assigning classes to extracted features. Besides these classification approaches neural network classifiers have recently gained a lot of attention. We provide a detailed introduction into neural networks in section 2.4.

Post processing aims at improving the recognition result. On the one hand post processing is used to group individually recognized characters into words. On the other hand post processing is used to detect and correct errors in the recognition result. There are two different methods for error detection and correction. The first is using a spell checker together with a dictionary that determines whether the predicted word is correct. If the predicted word is not correct the dictionary can be used to produce a suggestion of the correct word that has the smallest edit distance to the predicted word. The second method is to generate a language model that detects missing or ambiguous characters and corrects these errors to their most likely interpretation.

In the next section we will present problems of using conventional print OCR engines for scene text recognition and the motivation for inventing new methods for text recognition.

2.3. From traditional Print OCR to Scene Text Recognition

The accuracy of an OCR system is affected by a number of different factors. One of these factors is the resolution of the scanned document. If the resolution is not high enough the OCR system will not be able to correctly recognize all characters as there is not enough information available. Although commercial OCR systems are reporting accuracy rates of 99 % these rates can only be achieved on optimal scans of documents. Typical real-world scene text images may be captured with

low quality cameras, that neither provide a high resolution of the image nor a good contrast. Text in real-world images can furthermore be found on backgrounds of varying colors and shapes, which is different to text of scanned documents where the background is plain white. Although it is possible for traditional print OCR systems to recognize text written in various fonts the fonts appearing in scene text images are even more challenging and versatile. Figure 2.2 shows some examples of typical scene text text-line images that contain challenging fonts, backgrounds, various rotations and lighting effects like reflections that make it very difficult for a print OCR engine to correctly recognize the text. In the next section we will introduce fundamental techniques used in scene text recognition to overcome the limitations of print OCR systems.

2.4. Fundamentals of Scene Text Recognition Systems

Compared to a standard print OCR System a basic Scene Text Recognition System only consists of three to four basic parts. The first part is the Optical Scanning part which is the same as in print OCR systems. The second part is the text detection stage which creates hypotheses for text lines that may be verified in this stage or the next. The third part, which this thesis deals with, is the text recognition stage that gets all identified text lines as input and generates a prediction of the characters that might be present in the text line. The last part is the post processing part which is an optional part and in general the same as in print OCR systems.

This section will introduce the fundamental concepts of DNNs which are the basic building blocks for state-of-the-art text recognition systems. We will first introduce the main concept of neural networks and deep neural networks. After this we will introduce CNNs which play a crucial role in text recognition systems and lastly we will present RNNs which help to further improve the overall recognition rate when using CNNs as feature extractors.

2.4.1. History of Neural Networks

Neural networks have gained a lot of attention in recent years. At first glance this seems surprising as the actual concept of having a neural network as computational model is known since the 1940s. In 1943 McCulloch and Pitts where the first to introduce a calculus based on the inner workings of neural activities in the human brain [41]. In 1958 Rosenblatt introduced the perceptron which was the first model of a neuron that could be trained by an algorithm [48]. The first models of neurons where only able to compute linear functions on their inputs. This limitation let to a fast decay of popularity in the late 1960s as a wide area of problems could not be solved by only being able to compute linear functions. In the 1980s a second wave of research in the field of neural networks emerged and brought up the basics that are used today in deep learning. The central idea during that time was to create a large network of neurons achieving intelligent behavior by training these networks of neurons using the back-propagation method [49]. Adding a non-

Figure 2.2.: Challenging scene text images including a wide variety of fonts, background colors, distortions and lighting effects

linearity allowed these neural networks to also approximate non-linear functions, thus they were able to overcome the significant limitation of the early approaches. Hornik et al. [21] and Cybenko [8] showed that neural networks with a single hidden layer are able to approximate any function from a finite dimensional space to another finite dimensional space with any desired accuracy if the number of hidden neurons is large enough. Hinton et al. showed that adding more than just a single hidden layer to a neural network increases the representational capacity of the network by keeping the overall number of used neurons low [17]. This second wave of extensive research on neural networks lasted until the mid-1990s where other machine learning models like kernel machines achieved better results on important tasks. One of the main problems was that it was deemed that neural networks are difficult to train. In fact all algorithms used nowadays already exist since the 1980s and they work quite well. Neural networks also lost their appeal to the community because training a network is computationally costly and the large datasets used today were simply not available at this time. The third wave of using neural networks, called deep learning, emerged around 2006 when the computational power of multi-core CPUS and general purpose GPUs became more and more available. That increase in processing power allows researchers to train very large models in quite short time frames making it feasible to do research on and with large/deep neural networks. Deeper models have more hidden layers than shallower models and are thus able to learn more complex abstractions than shallower models. The power of deep learning models lies in their ability to decompose the overall task into several easy tasks to solve.

Nowadays the availability of "Big Data" makes machine learning and especially deep learning easier as the data itself can be used to learn the best features for solving a task are and also what the best way for extracting the features is. This reduces the amount of skill necessary for creating a successful deep learning solution because there is no need for creating sophisticated methods that extract features selected by human operators, the networks can learn to identify and extract necessary features by themselves. A large amount of training data also lowers the generalization error, because the subset of oberved data increases which in turn increases the range of different observed feature distributions.

2.4.2. Neurons and Fully Connected Layers

A neural network consists of neurons that compute an affine transformation

$$f(x, w) = w^T x + b \tag{2.1}$$

that takes n inputs $x \in \mathbb{R}^n$ with values $x_1, \ldots, x_n$ and a weight vector $w \in \mathbb{R}^n$ with values $w_1, \ldots, w_n$. The scalar value b is the bias of that function, which is a learnable parameter that forces the result of the matrix multiplication of the input vector x with the weight vector W into a defined direction.

On top of that neuron it is possible to employ an activation function that is able to transform the output of the linear function to an output of a non-linear function.

One of these activation functions is the binary step function that depends on an inner threshold τ:

$$output = \begin{cases} 0 & \text{if } f(x,w) \leq \tau \\ 1 & \text{if } f(x,w) > \tau \end{cases} \tag{2.2}$$

The step function is not a very good function to activate neurons because small changes in the weights of a neuron might lead to crucial changes in the output of the neuron activated with the step function. Over the course of time several other activation functions like the sigmoid function have been employed that are better because small changes in the weights do not lead to drastically changes in the values that are the output of the neuron. Using the sigmoid function as activation function makes it easier to train the network with back propagation because it is easier to compute the gradient of that function which is essential for doing back-propagation. A second possible activation function is the rectifier non-linearity which was found to be an even better activation function for neurons than the sigmoid function [12]. Rectifier units can be found in most modern DNNs as Rectified Linear Unit (ReLU). The function computed by a rectifier unit is the following:

$$output = max(0, f(x,w)) \tag{2.3}$$

Although the function is not differentiable at 0 (like the step function, which is not differentiable at threshold τ) and imposes a hard non-linearity it could be shown that a deep neural network containing ReLUs is able to converge while being trained.

A neural network is a network consisting of many of the neurons described above. Neural networks consist of layers. The first layer is always the input layer where a vector $x \in \mathbb{R}^n$ of input data is fed into the network for computation. The last layer is always the output layer that produces an output vector $y \in \mathbb{R}^m$ of length m. In between there may be an arbitrary number of layers (hidden layers) that actually compute the function learned by the neural network. Figure 2.3 shows the structure of such a neural network with one hidden layer. The hidden layer in figure 2.3 is a fully connected layer where each input is connected to each neuron of the layer itself. Each output neuron is also connected to each neuron of the hidden layer making the network a fully connected network.

2.4.3. Training of Neural Networks

Supplying input to a neural network and letting the neural network produce an output by computing the function specified by the learned weights is called a forward pass. A neural network can be seen as an acyclic directed graph and each step of a forward pass through the network can be saved in the form of a computational graph which can be used later for training the network.

After creation a network has weights that are randomly initialized making it necessary to adapt the weights of the network so that the function computed by the network fits to the task it shall perform. This is the process of learning. It is possible to learn a network in an unsupervised way, meaning that the network

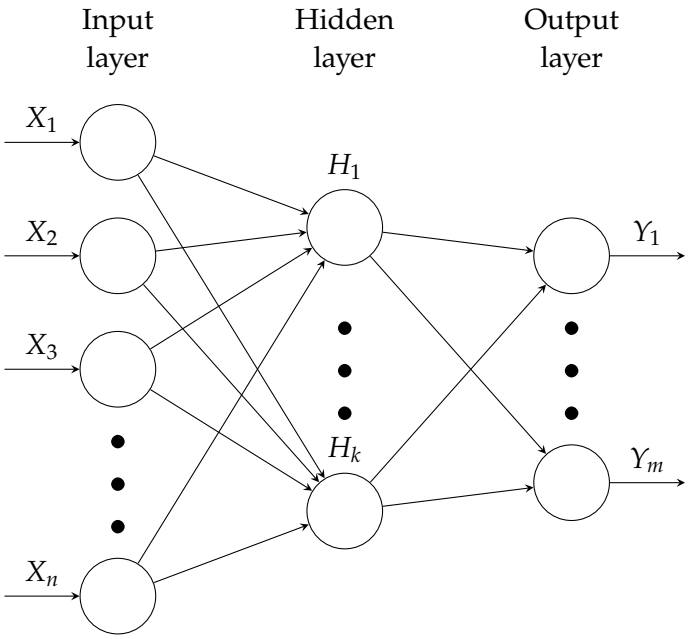


Figure 2.3.: Neural Network with one fully connected hidden layer

gets training data as input and tries to find clusters in the input data by itself. The other is supervised learning, which we used for all experiments shown in this thesis. While performing supervised learning the network gets an input vector x and a label l which is the desired outcome of the forward pass performed by the network.

In order to know how good the current prediction of the network is, it is necessary to have some kind of performance measure. Such a performance measure is a loss function L that takes the output of the network, the desired label and computes how well the network predicted the desired label. The quality value computed by the loss function is then used to maximize the probability that the output of the network is the label corresponding to the input vector. We could more formally say that the network tries to maximize $P(y = l|x)$.

We used two different loss functions in our work. The first one is the softmax cross entropy loss and the second one is the Connectionist Temporal Classification (CTC) loss which will be explained in more detail in section 2.4.6.

Softmax Loss

The softmax loss function is based on the softmax function which takes the output vector $y \in \mathbb{R}^m$ and produces a vector $z \in \mathbb{R}^m$ with the following properties:

- the value of each element of z lies in the interval $[0, 1]$
- the sum of all values of z is exactly 1

Thus, the softmax function computes a probability distribution over the output vector y. The loss L computed by the softmax loss function is now the negative log

probability (negative log probability is used to avoid numerical underflows) of the desired label l predicted by the network (model). More formally:

$$loss = -logP_{model}(l|x) \tag{2.4}$$

The loss computed in that way is then used to compute the gradients of each layer in the network, which is then used to update the weights of each layer.

Gradient Based Learning
Most modern neural networks use a gradient based approach like gradient descent for updating their weights. A gradient based approach uses the derivative of a function $y = f(x)$, which provides the slope of the function at every point x. If we know the point x we are currently at we can take a small step ϵ in the opposite direction of the sign of the derivative at this point to further minimize the output y. This approach is called gradient descent [6] and uses the the gradient computed over all samples in the training set used to train the network. As training sets for machine learning are usually quite large with several million of samples it is not feasible to compute an exact gradient for each learning step by running a forward pass for every sample in the train set. Stochastic Gradient Descent (SGD) is an extension of the gradient descent algorithm which only uses a small amount of samples at each learning step for computing the gradient, which produces an estimation of the exact gradient for the complete training set. It has been shown that this estimation is sufficient enough to learn a neural network.

In most modern neural networks the gradient necessary for doing gradient descent is computed by the back-propagation algorithm [49]. A neural network computes the composition of several functions. Suppose we have a neural network computing $y = f(g(x))$. Back-Propagation is an algorithm that efficiently computes the derivatives of the function created by the composition of f and g by using the chain rule of calculus. Together with SGD this can be used to update the weights of a neural network, thus minimizing the loss.

2.4.4. Deep Neural Networks

Neural networks showed so far only had a very limited number of layers. Traditional neural networks mostly consist of one hidden layer, although neural networks with one hidden layer can be seen as universe approximators [21, 8] and thus a network with one hidden layer is able to compute an arbitrary function it is not clear how large a such a single hidden layer has to be in order to compute the desired function. Stacking multiple hidden layers allows to compute complex functions with a lower overall number of neurons and weight parameters used in the network. This is the idea of deep learning. Stacking multiple layers on top of each other allows a neural network to learn how to extract features from the input data that get more abstract the deeper the network is. A special type of deep neural networks that is popular in computer vision is the CNN which we will introduce in the following section.

2.4.5. Convolutional Neural Networks

CNNs [7] are neural networks that behave differently than other neural networks, which we have introduced before. CNNs are based on findings of neurophysiologists that did research on how the mammalian vision system works. It was discovered that the visual cortex of a mammalian brain might be organized into different layers that each have their own specific task. The first layer is responsible for recognizing simple forms like edges and corners. In subsequent layers the level of abstraction that can be recognized by the neurons in that layers rises and neurons in higher layers are for example able to recognize faces or objects [55].

CNNs are play the role of receptive fields [7]. In that sense we can say the structure of CNNs is different to the structure of a typical neural network in such a way that a convolutional layer it does not do a matrix multiplication with every input at once, but rather performs a convolutional operation on the input. This convolutional operation is learned by the layer itself, by finding weights for the filters that are used to perform the convolutional operation. CNNs are popular for tasks in computer vision because they are very well suited for finding features describing images. A convolutional filter can for example be used to find horizontal or vertical edges in an image. Figure 2.4 shows the application of a filter that can find vertical edges.

A single convolutional layer can learn a large number of these filters and thus extract a large magnitude of different features from one image. While extracting these features from an image the filter with a kernel of width w and height h is slided across the image and is reused several times on the same input. This concept of weight sharing enables CNNs to find features like edges at different parts of each image with just using one set of $w \cdot h$ parameters. This property makes the application of CNNs also translation invariant. Suppose we have an image with one vertical line in it. If we translate the vertical line to the left by some pixels, it is still possible for a convolutional filter to extract the edges of that line in our image. Weight sharing also helps to reduce the overall computational complexity necessary for performing an operation on an input. The need to only save $w \cdot h$ parameters per convolutional feature map also drastically reduces the memory usage of the whole network.

A typical layer of a CNN employs three different operations:

Convolution The convolution operation is the operation described above which extracts features from the input by performing an affine transformation on the input.

Detector The detector on top of the convolution applies a non-linearity to the result of the convolutional operation. A typical non-linearity applied is the ReLU non linearity discussed in section 2.4.2.

Pooling The last operation of a convolutional layer may be pooling. Pooling is an operation that summarizes the output of several neighboring convolutional neurons. The most typical pooling operation found is the max pooling operation that takes the maximum activation value of a predefined neighborhood. Other

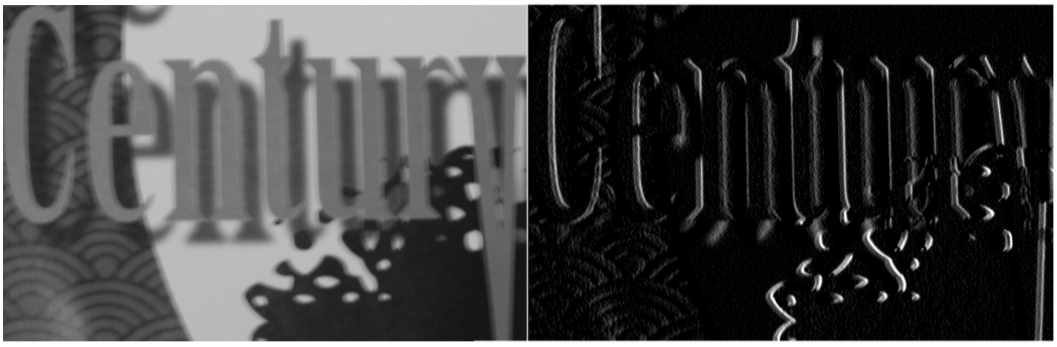


Figure 2.4.: Example of application of convolutional filters. The image on the right shows the strength of all vertical edges from the left picture. This effect has been achieved by convolving the source image with a $2x1$ filter applied to to each pixel of the original image. The filter subtracts the value of the left neighbor of the current pixel and stores the resulting value.

possible pooling variants are average pooling or using the L^2 norm of a neighborhood. The result of pooling is that the outputs of the convolutional layer become more invariant to local translations. Figure 2.5 contains an example showing the translation invariance introduced by a max pooling layer that keeps the dimensionality of the input. Besides the translation invariance produced by the pooling layer. A pooling layer can also be used to reduce the dimensionality of the resulting feature map, which in turn reduces the computational complexity and hence the speed in which convolutional operations can be performed.

2.4.6. Recurrent Neural Networks

Another building block that can be used for deep learning are Recurrent Neural Networks (RNNs) [49]. RNNs are neural networks that do not form an acyclic directional graph but rather a cyclic directional graph, meaning that information that has been seen before can be reused in a later step performed with other data. This makes RNNs are very well suited for processing sequential data of arbitrary length. RNNs consist of recurrent layers that use parameter sharing across all of their parts. This enable them to process sequences of arbitrary length. Each of these recurrent layers has connections to the input and output layer and a connection from its own computed output (the hidden state) to itself.

As it is not possible for the back-propagation algorithm to do back-propagation on a graph that is not acyclic it is necessary to use a trick and transform the computational graph of a RNN into an acyclic graph. This can be done by unfolding the recurrence relation in time. Meaning that each time step x^t with time step index t ranging from $1, \text{to } \tau$ is represented as a single node in the graph. A graphical representation of a folded and unfolded RNN can be found in figure 2.6.

The output h^t at time step t computed by a RNN directly depends on the last hidden state computed by the recurrent neurons h^{t-1}, the current input x^t and

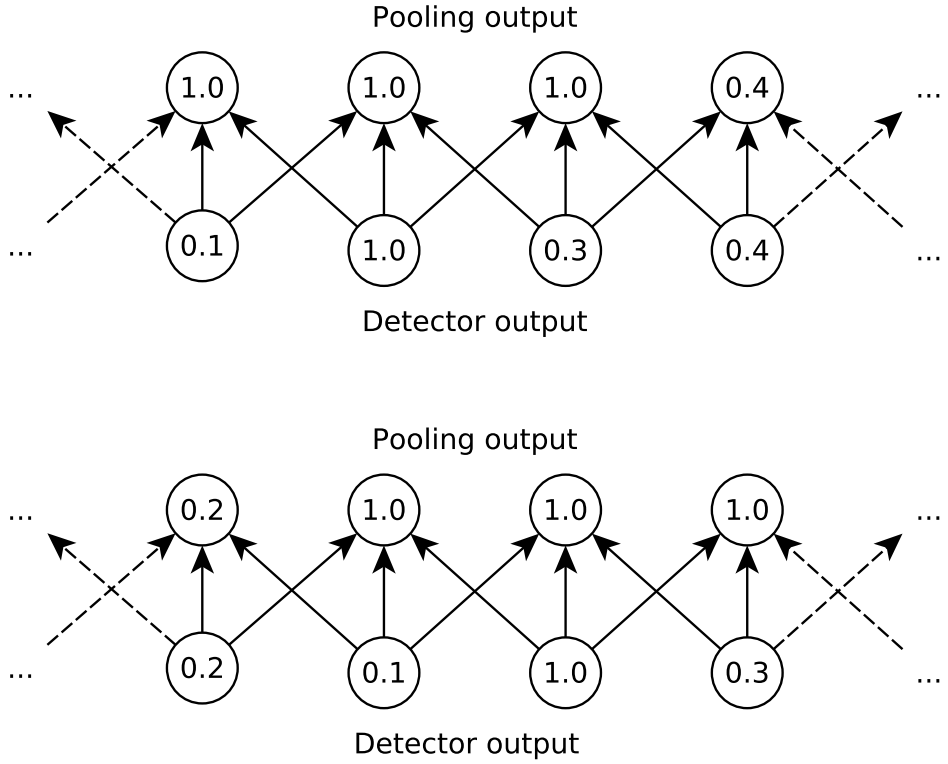


Figure 2.5.: Max pooling makes strong features translation invariant. *(Top)* The output of the detector operation (non-linearity) is pooled by a max pooling layer with stride one and a pooling kernel of three pixels. *(Bottom)* The output of the detector operation has been shifted to the right by one pixel. The output of the layer without pooling is very different than the last output. The output of the pooling layer is very similar output compared to the last output, showing that a max pooling layer is more robust against translation

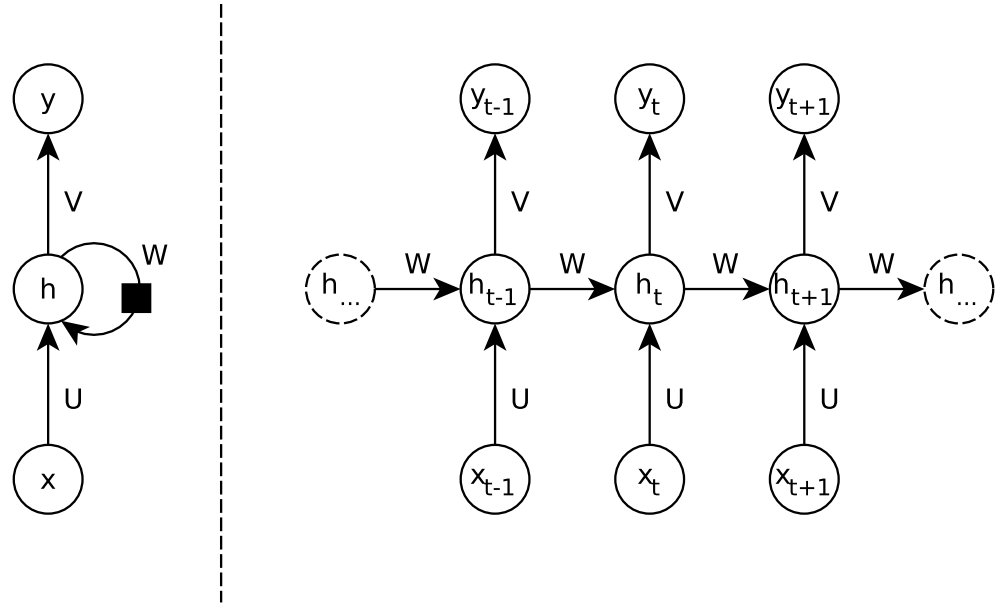


Figure 2.6.: Computational graph of a RNN in folded and unfolded view. *(Left)* Folded view of a RNN. The folded version of a RNN is a cyclic directed graph. At each time step the RNN gets an input vector x that is multiplied with a weight matrix U. The RNN keeps its own hidden state h in a feedback loop and together with the transformed input x this hidden state is multiplied with a weight matrix W that creates the next hidden state. The current hidden state is then multiplied with a weight matrix V creating the output y at each time step. *(Right)* Unfolded view of the RNN. The graph is now acyclic making it possible to perform back propagation. It is now possible to see that each of the matrices U, W, V is reused at every time step t.

the parameters θ of the network. The new hidden state is computed as shown in Equation 2.5.

$$h^t = f(h^{t-1}, x^t, \theta) \tag{2.5}$$

If the recurrence relation gets unfolded in time it is possible to see that RNNs share their parameters across different invocations as the **same** function f is used again and again. Equation 2.6 shows a sample unfolding operation for a RNN computing three time steps.

$$h^2 = f(f(f(h^0, x^0, \theta), x^1, \theta), x^2, \theta) \tag{2.6}$$

Reusing of the same function f also implies that the number of time steps can be arbitrary long because the network defines the transition from one state at time step t to the next state at time step $t + 1$, rather than a variable-length history of several states. This characteristic of RNNs leads to a low number of parameters that need to be optimized during learning.

RNNs can follow one of the following design patterns:

1. Recurrent networks that produce an output at each time step.
2. Recurrent networks that read the entire input sequence and produce one output at the last time step.
3. Recurrent networks that read the entire input sequence and produce a sequence of outputs.

For the application of scene text recognition using RNNs we will mainly focus on RNNs that follow the first design pattern. The following functions describe the forward pass of a RNN for each time step from $t = 1$ to $t = \tau$. The linear function which is the non activated output of the hidden neurons is computed like that:

$$a^t = Wh^{t-1} + Ux^t + b \tag{2.7}$$

The hidden state is then computed by applying a non-linear function on top of that result. In most RNNs the hyperbolic tangent function is used:

$$h^t = tanh(a^t) \tag{2.8}$$

The output of the RNN at each time step involves a second bias parameter c which is summed with the result of the matrix multiplication of the hidden state and the weight matrix for the hidden-to-output connections:

$$o^t = Vh^t + c \tag{2.9}$$

The last step may be to apply the softmax function on the output of the network in order to get a probability distribution over all outputs at each time step

$$y^t = softmax(o^t) \tag{2.10}$$

The probability distribution obtained by applying the softmax function on top of the output can then be used to compute the loss of the forward pass. The total loss of the forward pass of a given sequence x with their according labels l is the sum of the losses over all time steps.

$$-\sum_{t} logP_{model}(y^t|\{x^0,\cdots,x^t\}) \tag{2.11}$$

Backward computation in a RNN can be done by applying the standard back-propagation algorithm to the unfolded computational graph of the RNN. This application of back-propagation is called *Back-Propagation Through Time (*BPTT*)*. The gradients obtained by applying BPTT can be used by a weight update technique like SGD to train the RNN.

Limitations of the standard RNN model
RNNs provide a good way for processing sequences of arbitrary length with a neural network, but the use of shared parameters introduces the problem of exploding/vanishing gradients while performing BPTT. While computing the gradient the same weight matrices are multiplied with each other several times leading to an interesting problem [20]:

1. If an element of the weight matrix is larger than $abs(1.0)$ the resulting error gradient will increase exponentially, leading to unstable learning.
2. If an element of the weight matrix is smaller than $abs(1.0)$ the resulting error gradient will vanish exponentially, leading to a very limited learning success at the first time steps.

The problem of the vanishing/exploding gradient can be mitigated by either adding shortcut connections from earlier time steps to later time steps allowing the gradient to vanish later [39]. Another approach is to use a RNN unit that is not only capable of keeping information from the past, but also forgetting the current internal state if the need for that arises. The Long Short-Term Memory (LSTM) model by Hochreiter and Schmidhuber [19] aims at doing exactly this. A LSTM unit contains a number of gates that control the inner state and output of the unit at each time step. Besides these gates an LSTM unit also holds an internal state c that can be seen as a shortcut connection from earlier time steps to later time steps. The first gate of an LSTM unit is the forget gate which uses the hidden state h^{t-1} and input x^t to determine whether the current internal state c should be forgotten or not. The input gate is the second gate. This gate determines how much information of the current input and last hidden state will be saved into the cell state. The last gate is the output gate which determines how much information of the current internal state is used as output (next hidden state) of the LSTM unit. Together all these units mitigate the vanishing/exploding gradient problem because now the gradient is not propagated back through time by just multiplying one weight matrix with the hidden state, but rather by using multiple weight matrices and also summing and multiplying the results of the matrix multiplications. Figure 2.7

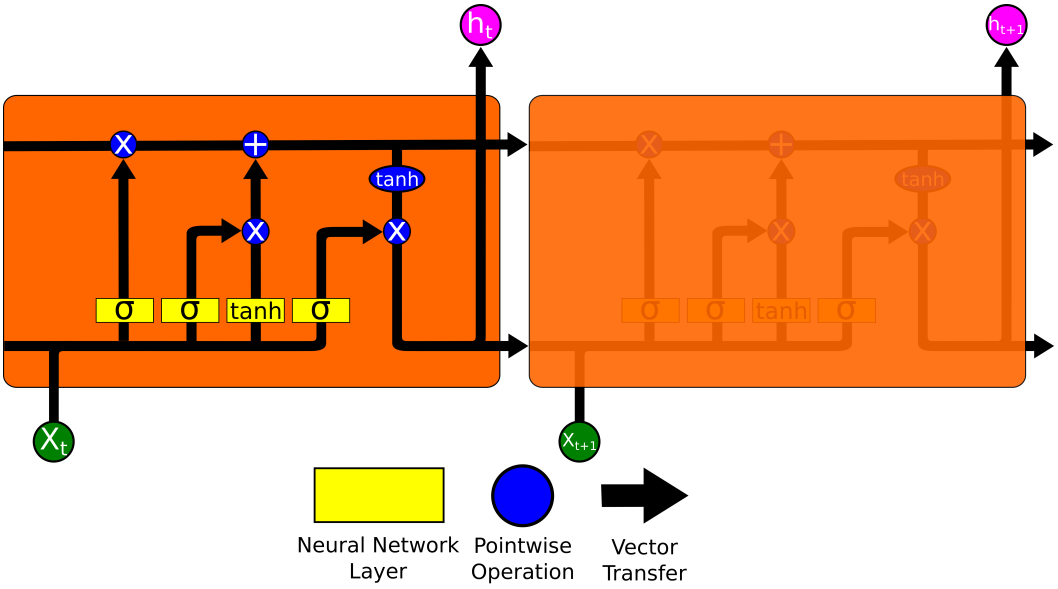


Figure 2.7.: Structure of LSTM unit that consists of multiple multiplicative gates. The top most arc is the cell state c. The first neural network layer on the left decides how much of the cell state shall be forgotten in each time step. The next two layers form the input gate that determines how much new data shall be stored in the cell state. The last layer is the output gate that determines how much information of the current cell state shall stay in the hidden state of the LSTM unit. Credit: [47]

provides a schematic overview of the inner structure of a LSTM unit. Please note that σ stands for the sigmoid function which is used as activation function on top of each internal gate.

Connectionist Temporal Classification

A RNN can be trained using the negative log probability of the target class which is determined by applying the softmax function to the output of the RNN at time step t (see Equation 2.10). This is only possible if for each output y^t produced at time step t there exists an according label l^t that is directly aligned to this output. When training on sequences of input data this is not always the case. Especially in the field of speech recognition it is common to have several time steps of noise or no interesting data before data is put into the network that is actually relevant for the task of speech recognition. The same problem also applies to scene text recognition where it is not always possible to know the alignment of the ground truth labels $l^0, \ldots, l^k$ consisting of k labels and the input sequence $x^0, \ldots, x^\tau$ consisting of τ inputs ($k \leq \tau$).

CTC [15] is a function that tries to align the label sequence l of length k to the output sequence y of length τ of the RNN. In order to do this each output o^t is put into a softmax output layer producing a probability distribution y^t with one more element than there are labels in the alphabet L. Each y_k^t corresponds to the activation of unit k (probability of label at index k in alphabet L) at time step t. The extra element corresponds to the probability that the network did not

see any of the other labels (the network does not produce an output for this time step). This effectively creates a new alphabet $L' = L \cup blank$. A valid sequence L'^t that can be created using the elements of the alphabet L' is called a path π. Using this information it is possible to define a map B that maps each path π to a possible labeling that removes all repeated labels and blanks from the path (e.g. $B(h-ee-l--l-o) = B(hh-e-l-loo) = hello$). It is not feasible to do this for every possible path of length τ as the number of possible paths grows exponentially with the number of time steps. A fast solution, that does not yield the best solution but a solution that is good enough, would be to perform best path decoding where the most probable path π is always taken:

$$y = argmax_{\pi \in L'^t} P(\pi|x) \tag{2.12}$$

$$\text{with } P(\pi|x) = \prod_{t=1}^{\tau} y^t_{\pi^t}, \forall \pi \in L'^\tau \tag{2.13}$$

2.5. Related Work

In this section we will review the research done in the field of text recognition, especially scene text recognition. Generally speaking it is possible to divide the methods proposed by other researchers into two broad categories. Methods that belong to the first category are methods that perform scene text recognition using hand crafted features and shallow machine learning methods like Support Vector Machines (SVMs), k-Nearest Neighbors (k-NNs) or Conditional Random Fields (CRFs). Most approaches hereby first determine the location of each character in a given word line. A second stage recognizes the characters in that identified character regions. Methods belonging to the second category employ automatic feature extraction through neural networks and deep learning for performing scene text recognition.

2.5.1. Methods based on Shallow Machine Learning

Character based approaches try to tackle the task of scene text recognition by decomposing the overall task into subtasks that correspond to the tasks performed by traditional print OCR systems (see section 2.2). By doing that these character based approaches use sophisticated approaches for character detection and segmentation paired with the use shallow machine learning for character recognition. Epshtein et al. [11] introduce an image operator (stroke width transform) that is capable of creating binary text/non-text masks from images. These masks are created by transforming every pixel of an image into a representation where the value of each pixel corresponds to the width of the stroke that most likely contains that pixel. Regions with similar stroke width will then be merged together by a connected component analysis into character candidates. These character candidates will then be grouped into words by further analyzing the differences between the

stroke information of each identified character candidate. Recognition can then be performed by using print OCR software.

Yao et al. [62] propose to learn a whole set of classifiers that extract essential structures of characters (strokes) from the presented text line image using classifiers trained to extract specific sets of strokes from the image at different scales. The extracted strokes will be used to approximate the character bounding box and all extracted strokes in that bounding box are used as features for the character classifiers. Yao et al. use 62 SVMs in order to distinguish 62 different characters.

Wang and Belongie [59] generate character positions and predictions by cross correlating (using Normalized Cross Correlation) each part of the text line image, which they transform into Histogram of Oriented Gradient (HOG) features, with the HOG features of every char in their training set. The proposed character predictions are then fed into a k-Nearest Neighbor (κ-NN) classifier that produces a prediction for the character. The predicted characters are then refined by performing a matching search in a dictionary.

Mishra et al. [42] propose to use a sliding window classifier based on HOG features and a SVM classifier to detect and recognize single characters in the text line image. They use a higher order language model on top of these predictions for correcting possible errors made by the character classifier. Predictions are chosen based on a CRF classifier that uses a cost graph over the predicted single characters and their possible higher order language model extensions.

Almazan et al. [1] propose a multimodal method that embeds labels and images into a common feature space. Labels/dictionary words are thereby embedded into a high dimensional space by computing a pyramidal histogram of characters where each dimension corresponds to the presence or absence of a character in a word, and at different locations in that word. Input images are transformed into fisher vectors of same dimensionality as the embedded labels. Both representations are then transformed into a common feature space by performing a common subspace regression where they try to find projections for each type of features that maximize the correlation of the projected results in the common subspace.

Neumann and Matas [45, 44] propose to detect character candidate regions by using MSERs and extremal regions, which will be refined in subsequent steps by applying a text/non text SVM. Verified text regions are then merged into a large word region by applying another SVM on the remaining character candidate regions. Character regions belonging to a single word are then used to extract geometric features which will then be used by another SVM to recognize each character individually.

All of these approaches use handcrafted features that may work well for a specific range of tasks but may fail if the task domain changes slightly. It is also not clear whether the handcrafted features are representative features for the problem they shall solve. Handcrafted features furthermore add a lot of complexity to scene text recognition systems as the methods used to extract and compute these need a quite complex stack of single steps.

2.5.2. Methods based on Deep Learning

Methods based on deep learning try to overcome the complexity of the approaches mentioned before. Most of the following deep learning approaches do not have the need for sophisticated preprocessing steps. They rather take complete images as input or use a fixed sliding window as input to their feature extractors. Most methods based on deep learning do not use hand crafted features and therefore seem to be superior compared to shallow machine learning based approaches.

Bissacco et al. [4] propose "Photo OCR" a scene text recognition system that can be seen as on the edge between deep learning and shallow machine learning methods. They first extract characters from a text line by using a segmentation approach that generates a huge amount of possible word line segmentations. They try to find segmentation points of each character by first performing a connected component analysis on a binarized word line image and if that approach fails by using a binary classifier based on HOG and Weighted Direction Code Histogram (WDCH) features. The identified segmentation points are then used for character recognition by a 99 class classifier based on a fully connected deep neural network. A post processing step using a language model tries to correct recognition errors. The overall approach by Bissacco et al. is very complex, but it yielded the best results in the International Conference on Document Analysis and Reading (ICDAR) 2013 challenge on robust reading.

Wang et al. [60] use two two layer CNNs for both character detection and recognition. They perform a multiscale text detection by sliding the text detection CNN across the full scene image. That CNN decides whether the currently investigated image patch contains a centered character or not. Using that approach they detect possible word lines and also word boundaries. After the detection of all word lines and word boundaries the text recognition CNN is used to recognize characters of each word.

Jaderberg et al. [27] propose a similar approach to Wang et al. [60]. They also slide a text detection CNN across a full scene image, but they train a deeper CNN that uses maxout as activation function [14]. Furthermore they use different kinds of CNNs as classifier. The final word prediction is generated by scoring the prediction results of three CNN classifiers, a 37-way case insensitive character classifier, a 62 way case sensitive character classifier and a bigram classifier that can recognize arbitrary combinations of two characters.

Jaderberg et al. [25] refined their approach for text recognition by introducing a network that takes a text line image as input and provides the text in that text line as output. The network they use is constrained to 90000 possible classes, each class representing one word from a dictionary. They furthermore introduce the design specifics of a synthetic data engine that can be used to generate an arbitrary number of training samples necessary for a successful training of a CNN for text recognition.

Goodfellow et al. [13] propose a CNN taking an image containing house numbers as input and providing the numbers in that image as output. The CNN itself

consists of five classifiers. Each classifier handles the recognition of one number. The first classifier always deals with recognizing the first number, the second with recognizing the second and so on. They also add a sixth classifier that predicts the length of the house number in the provided text line image. All classifiers share their convolutional feature extractors with each other but predict the results independently.

Jaderberg et al. [28] further developed the approach of Goodfellow et al. [13] for unconstrained text recognition. The network proposed by Jaderberg et al. now has 23 classifiers instead of five classifiers, allowing them to recognize words with a length of up to 23 characters. Instead of an extra classifier that predicts the length of the word in the text line image they add a new class to each classifier indicating whether the word has already ended or not.

He et al. [16] introduce a CNN paired with a RNN that can be used to recognize words without length constraints. They use a sliding window over the text line and extract features from every sliding window step. These extracted features can then be seen as single time steps which are the input to the RNN on top of the CNN.

Shi et al. [51] also handle scene text recognition as a sequence problem by using a CNN with a RNN on top. They do not use a sliding window approach but rather use the whole image as input to the CNN part of the network. The CNN produces a feature vector that has a larger width than height. Each vector of width elements is then used as input to the RNN. The RNN used is a deep Bidirectional LSTM (BLSTM) network that uses past and future context for the text prediction.

3. Data Generation

3.1. Motivation

In order to successfully train a deep neural network a large amount of data is required. A general rule of thumb is that a supervised deep learning algorithm will achieve an acceptable performance with around 5000 labeled examples per category and even human level performance with a dataset having at least 10 million labeled examples [2]. If we consider the task of scene text recognition where we want to distinguish between all numbers from 0-9, latin letters from a-z and special characters like the exclamation mark, colon, semi colon, question mark and others we end up with a total of $(10 + 26 + 14) \cdot 5000 = 5\,200\,000$ samples we need at least in order to successfully train a network with an acceptable performance for scene text recognition.

There are some benchmark datasets for scene text recognition like the ICDAR dataset for the robust reading challenge[1] or the IIIT5K [42] dataset. These datasets include pre-partioned datasets. There is always one train dataset containing images that can be used for training the system and a test dataset which is the actual benchmark dataset. Train datasets do not include more than 2000 images each. As there are not enough of these benchmark datasets it is necessary to get more training datasets that have the same visual complexity as real-world scene text images have.

Image hosting/sharing services like flickr[2], pinterest[3] or instagram[4] provide lots of user uploaded images that may contain scene text but none of these images has annotations or a ground truth that can be used for training a deep neural network. It would be possible to create that ground truth by hand or semi automatically, but this is a cumbersome task that would take a lot of time.

A better solution is to generate synthetic data that has nearly the same properties as real world data and therefore acts as a good estimator for the data a trained network has to expect when being fed with real-world data. A system that generates synthetic data should be capable of performing the following operations:

1. Render arbitrary text strings using a huge amount of different font faces that can also be found in real world images.
2. Add different colors, shadows/borders with different displacements to that rendered text lines in order to further increase the visual complexity.

[1] http://rrc.cvc.uab.es/?ch=2&com=downloads
[2] https://www.flickr.com/
[3] https://www.pinterest.com/
[4] https://www.instagram.com/

3. Rotate and distort the generated text lines in order to train a text recognizer that is robust to small projective distortions.
4. Blend the generated image with a natural image containing no text, creating a sample that has the same properties as most real world images.

Jaderberg et al. [26] used such a generator for training samples and created a publicly available dataset consisting of 8 million training and validation samples[5]. We used this dataset for training our different models but because the dataset does not contain text lines with blurred text, reflections on the texts, or special characters like exclamation marks, colons, semi colons or question marks, we decided to implement such a data engine on our own. In this chapter we will introduce the design and implementation of our own data generation tool that overcomes the problems of the training samples generated by Jaderberg et al. and adds more visual complexity making the generated images even more realistic.

3.2. Data Generation Pipeline

The proposed data generation tool is written in Python and uses Pillow[6] a fork of the Python Imaging Library (PIL) and numpy[7] as external libraries. The generation tool is able to generate samples in various scripts, with various fonts, various backgrounds ranging from single color backgrounds to natural image backgrounds and various forms of image distortions blur effects and colors. The generated samples are always gray scale images as we do not need colored images for the training of our network, but it would be easy to change that behavior to also allow colored images.

The data generator itself consists of 4 different main building blocks:

Generator The generator is the brain of the data generator. The Generator creates the words that will be rendered and applies all image operations on the generated samples.

Sample The Sample class is a wrapper around the various image operations that are involved in creating a sample with the data generator.

Config The config class is responsible for reading the config file and also for deciding whether to apply an image operation or not. An object of the Config class also decides what parameters are passed to the image operation methods provided by the Sample class.

FileManager The FileManager is responsible for loading all fonts and background images used during the generation of samples. It is also responsible for saving the generated samples.

We will now describe the process of generating a sample using our data generation tool. A graphical representation of that description can also be found in figure 3.1. The first step of sample generation is data preparation:

[5] http://www.robots.ox.ac.uk/~vgg/data/text/
[6] http://python-pillow.org/
[7] http://www.numpy.org/

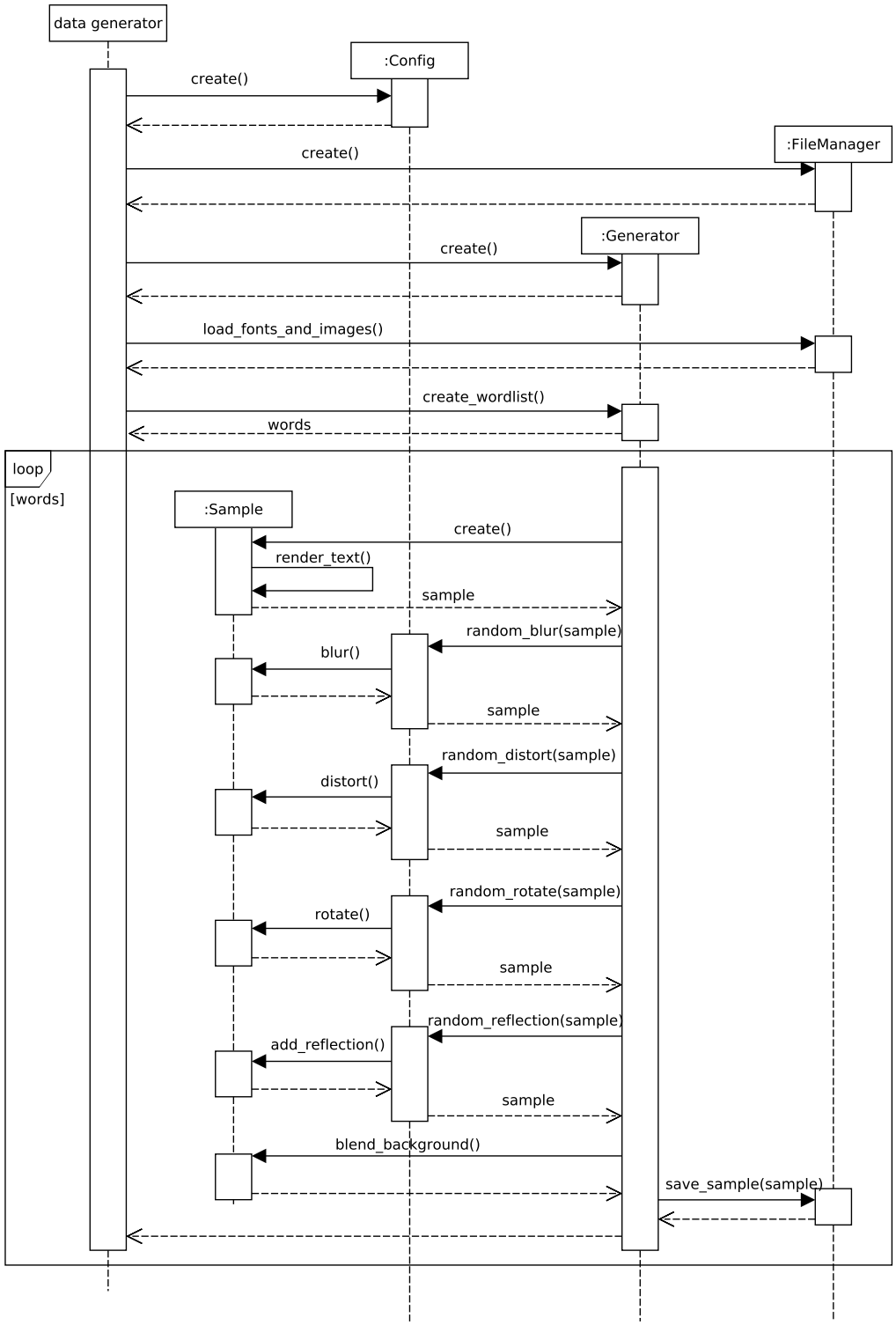


Figure 3.1.: Sequence diagram showing the sequence of events and function calls involved in creating a sample with the data generator

Config parsing The first step is to read the config file, where the user can specify all data sources like fonts, background images, output size of the generated samples, what kind of characters to use (only use lower case, upper case, numbers or any possible combination), whether to use words from a wordlist and the parameters for each image operation like distortion.

Font loading The data generator loads all fonts that reside in the font directory specified in the config file. For generating words using a language that uses latin script we found that it is highly beneficial to use a large set of different fonts. Google[8] provides a very large set of different fonts that can be used for that purpose.

Background image loading The next step is to create a list of all natural background files that are supplied to the tool. Background files are lazily loaded into memory because the total amount of blending files might exceed the memory capacity of the system where the data generator is running on. In order to save time by not having to load every file if it is reused again we use a least recently used cache that can hold up to 4096 images.

Wordlist creation After loading all auxiliary files it is time to generate all words that will be put on samples. Words can be generated in two different ways:

1. The user supplied a wordlist file that contains one word or sentence in each line of the file.
2. The user specified which kinds of characters to use (lower case characters, upper case characters, numbers) and the wordlist generation script will generate random word strings up to a specified maximum length.

While creating the wordlist, the data generator may add special characters like "!, ?, :, ; (,), [,]" if it is configured to do so.

Start of sample generation It is now possible to generate a sample with a random font, random font size and random background that contains one word or sentence from the wordlist generated in the last step.

The sample generation process now renders a text and a shadow image using the supplied font and the supplied font size. Text image, shadow image and background color have randomly selected colors, but background color and shadow color are constrained to have a minimum contrast compared to the text color. That minimum contrast value can be configured in the config and could also be set to zero to allow any color as background and shadow color. The next step is to blend shadow image and text image together using alpha compositing. After the generation of the base sample, the data generation tool applies image transformations in the following order:

random gaussian blur The already created text sample is blurred using gaussian blur, where the value of each pixel P is computed as the weighted average of the neighborhood of that pixel. The weighted average of all neighborhood

[8] https://fonts.google.com/

pixels can be determined by convolving the image with a gaussian filter. The gaussian filter used by the data generator uses a random blur radius. A gaussian filter with a blur radius of 3 may look like this:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \tag{3.1}$$

RANDOM DISTORTION The data generation tool adds random distortions to the image by applying perspective transformations. This creates images of text lines where the text is not perfectly aligned but rather skewed and incidental oriented.

RANDOM ROTATION Besides that random distortion that does a three dimensional transformation of the image, the data generator is also able to further rotate the already blurred and distorted text.

RANDOM REFLECTION After the image has been rotated it is possible to add a randomly selected reflection. This can be done using one of two different strategies:

1. Use predefined reflection overlays that are alpha composited with the generated image and simulate a white glare on the image. It is save to assume that a white glare is sufficient enough to model a natural reflection as we are using grayscale images.
2. The second possibility for creating a reflection is to use a set of metaballs [5]. A metaball is an algebraic surface that is based on quantum mechanics where an electron is represented as a density function of its spatial location. For the application of metaballs in this case the density function has been redefined to represent a distance D which is parameterized by the location in the image x and y. This distance depends on the mass of the metaball and the position x_i, y_i of the metaball center in the image:

$$D(x,y) = \frac{(x-x_i)^2 + (y-y_i)^2}{\text{mass}_i} \tag{3.2}$$

Using this distance and two predefined thresholds T_1, T_2 with $T_1 < T_2$ it is now possible to determine the value of a pixel in the image depending on its location to the center of a metaball and its mass:

$$Pixel(x,y) = \begin{cases} 255 & \text{if } D(x,y) < T_1 \\ (1 - D(x,y)) * 255 + D(x,y) - 1 & \text{if } T_1 < D(x,y) < T_2 \\ 0 & \text{if } D(x,y) > T_2 \end{cases} \tag{3.3}$$

A resulting reflection image can be found in figure 3.2.

BACKGROUND BLENDING The last step in the sample generation process is to add a background to the sample. This background can be a plain color or a natural image background. The first blending step is to add a plain colored background with a random opacity to the generated text sample. In the second

Figure 3.2.: Image with reflection overlay created using metaballs with mass 500 and thresholds $T_1 = 0.2$ and $T_2 = 10.2$

Figure 3.3.: Comparison of generated samples with real world images from the ICDAR dataset of the robust reading challenge. *(Left)* Images generated by our sample generation tool. *(Right)* Images taken from the robust reading challenge.

step the tool decides whether to further add a natural background or not. The opacity and probability of a natural background image to add can be configured using the config file.

We argue that the data generator is able to create samples that look similar to real world examples used in opened scene text recognition benchmarks like the dataset for robust reading challenge of the International Conference on Document Analysis and Reading (ICDAR). Figure 3.3 shows a comparison of samples generated with our tool and samples of the robust reading challenge.

4. Scene Text Recognition using CNNs

Besides having a large amount of training data at hand it is necessary to have a good network design that is capable of extracting relevant features from the training data. Very strong feature extractors that have been used for scene text recognition are based on CNNs. In this chapter we will introduce an approach for scene text recognition based on a convolutional neural network architecture that has been proposed by Jaderberg et al [28]. We will first describe the approach. We will then introduce the basic structure of the used network. We will then discuss strategies that could be employed to successfully train such a network.

4.1. Description of Approach

There are some deep learning approaches to scene text recognition that utilize a CNN based text recognizer with a single classifier, that has to be centered on top of a single character and thus can only recognize single characters [60, 23]. The input to these networks is a cropped image containing a single character that is preferably in the middle of the cropped region. If the character cropping is not done well enough these approaches suffer from bad recognition accuracies.

To overcome the cropping problem Goodfellow et al. [13] proposed a network that takes a full image as input and produces a sequence of characters/numbers as output. Goodfellow et al. constrained their solution to the problem domain of recognizing house numbers in Google Street View[1] images. The idea of the introduced approach is to use a CNN that consists of multiple classifiers on top of the last layer of the CNN. For the purpose of house number recognition Goodfellow et al. used a network consisting of five classifiers that are used for recognizing individual numbers and one classifier that predicts the length of the house number in the given input image. Each of the five classifiers is responsible for recognizing a number at a given location in the sequence of individual digits. The first classifier is responsible for recognizing the first digit, the second classifier is responsible for recognizing the second digit and so on. All classifiers use the same feature vector that has been extracted by a CNN.

A prediction of the network is derived by:

$$\text{number of length l} = transcribe((l, s_1, \ldots, s_l)), \text{ with} \tag{4.1}$$

$$(l, s_1, \ldots, s_l) = argmax_{L,S_1,\ldots,S_L} log P(S|X), \text{ with} \tag{4.2}$$

$$P(S = \text{number of length l}|X) = P(L = l|X) \prod_{i=1}^{l} P(S_i = s_i|X) \tag{4.3}$$

[1] https://www.google.de/intl/de/maps/streetview/

With X being the input image and each s_i being the output of classifier i. The function *transcribe* takes the generated length prediction l and the prediction of the class id for each digit s_i and returns a number with l digits.

Based on this approach Jaderberg et al. [28] propose a refined version of this network design that is capable of recognizing lower-case words with a length of up to 23 characters. The proposed network consists of $N_{max} = 23$ classifiers where each classifier i is responsible for predicting a character $c_i \in C = [a - z0 - 9\varnothing]$ at position i in the given text line. The character class $\varnothing$ corresponds to the case that the character classifier predicts no character at its position. Adding the extra character class $\varnothing$ makes the extra classifier that predicts the length of the given text line obsolete. If all characters of a word with length n have been classified, classifier c_{n+1} predicts the class $\varnothing$ to indicate that the word has ended. A word prediction w^* is a sequence of n characters such that $w^* = (c_1, c_2, \ldots, c_n)$ with

$$w^* = argmax_{c_i} P(c_i|X) \text{ for each } i \in [1, N_{max}] \tag{4.4}$$

The predictions of all classifiers are then combined using a *transcribe* function similar to the function of Goodfellow's approach. The *transcribe* function simply removes the prediction of all $\varnothing$ classes.

A graphical representation showing the basic idea of both approaches can be found in figure 4.1.

4.2. System Structure

We created a network based on the approach introduced by Jaderberg et al. We created the network using the deep learning framework Convolutional Architecture for Fast Feature Embedding (Caffe) by Jia et al. [29]. The network as such consists of a convolutional feature extractor based on the AlexNet by Krizhevsky et al. [33]. The convolutional feature extractor consists of five convolutional layers and two fully connected layers. All input images are scaled to have a width of 100 pixels and a height of 32 pixels, regardless of the aspect ratio of the source image. The five convolutional layers perform convolutions with padding of feature maps to preserve spatial dimensionality. The first, second and third layer are followed by $2x2$ max-pooling with stride 2. Each max-pooling step cuts the spatial dimensionality of the feature maps into half, reducing the amount of memory and computational resources required for training the network. ReLUs are used as non-linearity after each convolutional layer. The two fully connected layers consist of 4096 fully connected neurons each. The last layer is a fully connected layer with $23 \cdot 51 = 1173$ neurons that acts as a layer containing 23 independent classifiers that can distinguish between 51 different classes. The network has been trained using SGD. Table 4.1 provides an overview of the described network layout.

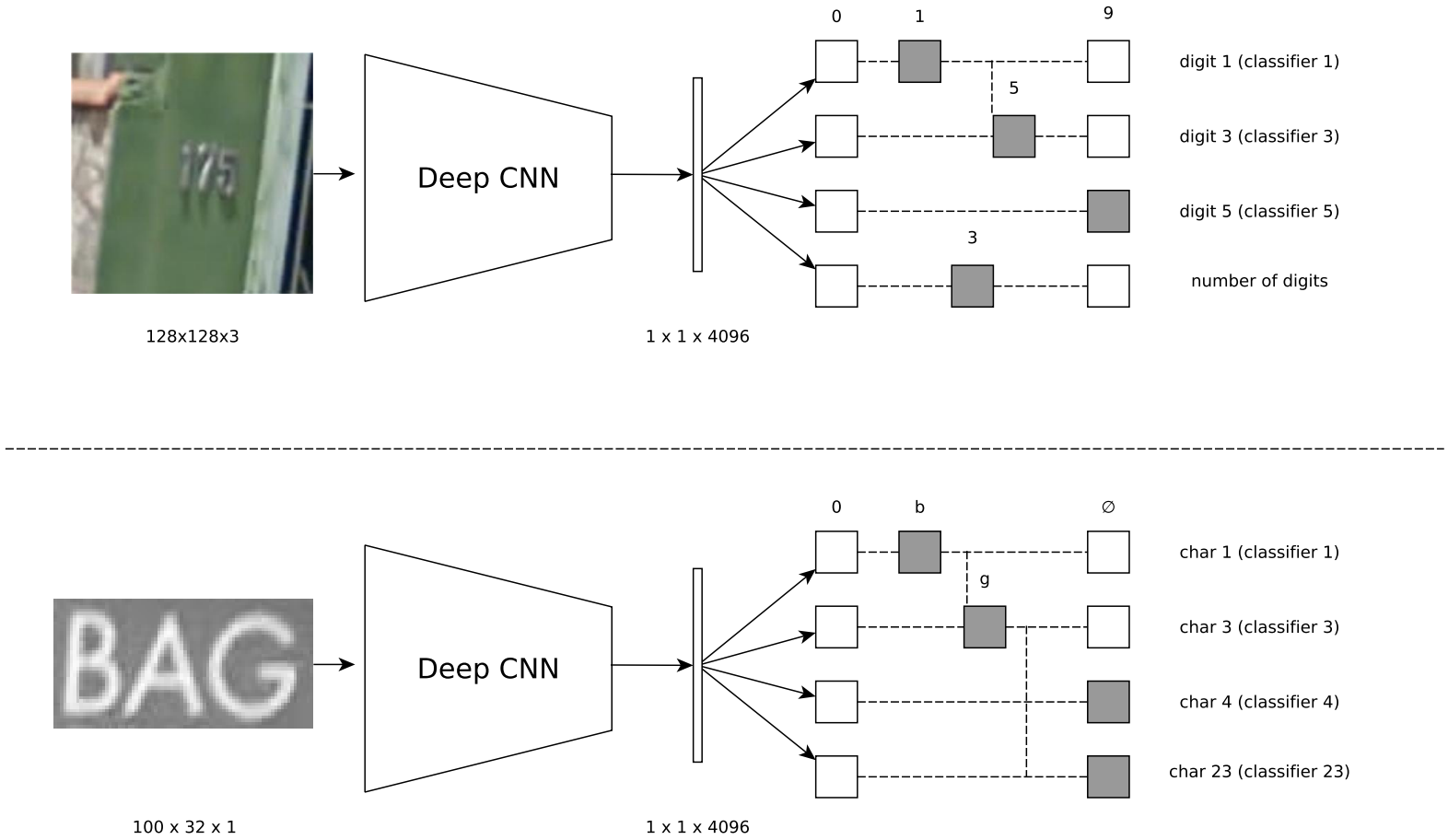


Figure 4.1.: Approach of Goodfellow and Jaderberg in comparison. *(Top)* Approach of Goodfellow et al. that takes a single street view image as input and produces a prediction with up to 5 digits and one prediction that encodes the length of the text line shown. *(Bottom)* Approach of Jaderberg et al. that takes a text line image and creates 23 predictions. One for each possible character in the word. If there is no character left the model predicts the no-char class ∅.

Type	input maps	output maps	kernel	stride	padding
Convolution	1	64	5 x 5	1	0
Max Pooling	64	64	2 x 2	2	0
Convolution	64	128	5 x 5	1	2
Max Pooling	128	128	2 x 2	2	0
Convolution	128	256	3 x 3	1	1
Convolution	256	512	3 x 3	1	1
Max Pooling	512	512	2 x 2	2	0
Convolution	512	512	3 x 3	1	1
FullyConnected	24576	4096			
FullyConnected	4096	4096			
FullyConnected	4096	1173			

Table 4.1.: Structural overview of CNN used for text recognition

4.3. Achieving Convergence

While experimenting with the described network we observed that it is difficult to make the network converge on the given training data. An investigation of the learned filters of the convolutional part of the network showed that these filters still have randomized parameters. We also observed that the accuracy and loss reached a constant value very fast. This indicated that the network did not learn anything in the lower convolutional layers but rather in the two fully connected layers. It might be that the task we want the model to learn is to complex and it is to difficult to optimize the used network due to the deepness of the network. The problem is well known in literature and there are several different ways of mitigating such a problem:

Layer-wise pre-training It has been found that it is beneficial to train the layer of the actual network step by step [18, 3]. In the case of the CNN we used for our experiments this means that we start the training from scratch with only one convolutional layer and both fully connected layers. This network is then trained until the loss does not decrease significantly anymore. The next step is to add the second convolutional layer and perform a fine-tuning on the pre-trained model. This process continues until all convolutional layers have been added to the network and the network finally converges during the last fine-tuning step. It is believed that this process works well because the initialization of the weights in the lower layers is better suited for the task, when a fine-tuning based on a small set of pre-trained layers is performed.

Finetuning based on a different network A similar way of mitigating the problem of the network not learning is to fine-tune a model that has been used to solve a different task. For achieving our results we used a model with the same overall structure as our model that has been trained to decide whether an image patch contains text or not. This is a two way classification task that is way easier than our $23 \cdot 51$ way classification task. Instead of pre-training over a

fully trained network where only the last layer gets exchanged, it is also possible to take already trained convolutional layers of a different network and train the network based on these layers [52].

Usage of multiple loss layers during training A further way of making the model learn is to add auxiliary classifiers at different parts of the network. Adding extra classifiers that are learned on the same task as the main classifier during training and adding small portions of the loss produced by these extra classifiers to the overall loss of the network increases the strength of the gradient signals that get propagated back and also adds additional regularization. This approach has been used by Szegedy et al. for their GoogLeNet which won the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14) [54].

Usage of Batch Normalization Another method that helps training deep networks is batch normalization which has been introduced by Ioffe and Szegedy [22]. Ioffe and Szegedy determined that a change of the parameters in a lower network layer leads to a change in the distribution of the input data in higher layers of the network, which makes it more difficult for the network to learn individual layers. They call this phenomenon *internal covariate shift*. The idea of batch normalization is to reduce this internal covariate shift by fixing the output distribution of a layer. The normalization done by batch normalization normalizes the output of a layer by subtracting each output value by the mean of the outputs produced for the current mini-batch and dividing the result by the variance of the mini-batch values effectively creating a distribution with mean zero and the same covariance over the training set, which is found to be beneficial for training neural networks [37].

Using the layer-wise pre-training, fine-tuning based on a different network and batch normalization approach made it possible for us to create a model that converged on the given training set.

As we wanted to compare the performance of this model to the models introduced in the next section we performed some experiments with this network layout. A description and the results of these experiments can be found in section 6.2.1.

5. Scene Text Recognition using RNNs

The CNN described in the last section consists of 23 classifiers that produce a sequence of up to 23 characters. The length of this sequence is constrained to a maximum number of 23 characters making it not possible to recognize words with more than 23 characters. In order to remove the constraint of being only able to recognize words with a length of up to 23 characters we thought it might be possible to use a RNN that can handle sequences of arbitrary length which seems to be a natural solution for the problem of scene text recognition as we are trying to recognize a sequence of characters from a text line image. In this chapter we will first introduce the basic approaches we examined for performing scene text recognition with RNNs. We will then describe implementation details of the networks we used for our experiments. This is followed by a description of our BLSTM implementation in the deep learning framework Chainer.

5.1. Description of Approaches

While experimenting with networks using RNNs for scene text recognition we came across three different system designs. The first design is based on a sliding window approach [16], where a sub window is pushed across the text line image. This sub window has a predefined height and width and is pushed across the image with a fixed stride. Each image area that is contained in the sub window is then processed by a CNN, which extracts features from that region of the image. The resulting feature vector is then used as input to a RNN consisting of a single BLSTM, see figure 5.1 for a graphical representation of the network structure. Although the network uses a sliding window approach it is not necessary to have a label for each individual extracted sub window. The usage of CTC as loss function allows a sequence of predictions with length l_p to be mapped to a sequence of labels of length l_l if $l_p \geq l_l$. This is a huge improvement over other sliding window approaches [60, 24] as this means that it is not necessary to explicitly design input dimensions and scan an image using different scales.

The second approach does not utilize a sliding window to generate the input sequence to the recurrent part of the network [51]. This approach takes the full text line image as input to a CNN and extracts a sequential feature representation of that image. This sequential feature representation is a feature vector that forms the input sequence to the RNN on top of the convolutional feature extractor. The feature extraction happens in a similar way to the feature extraction done in the CNN approach. The recurrent part of this network consists of two BLSTMs stacked on top of each other, see figure 5.2. As it is not known which sequence

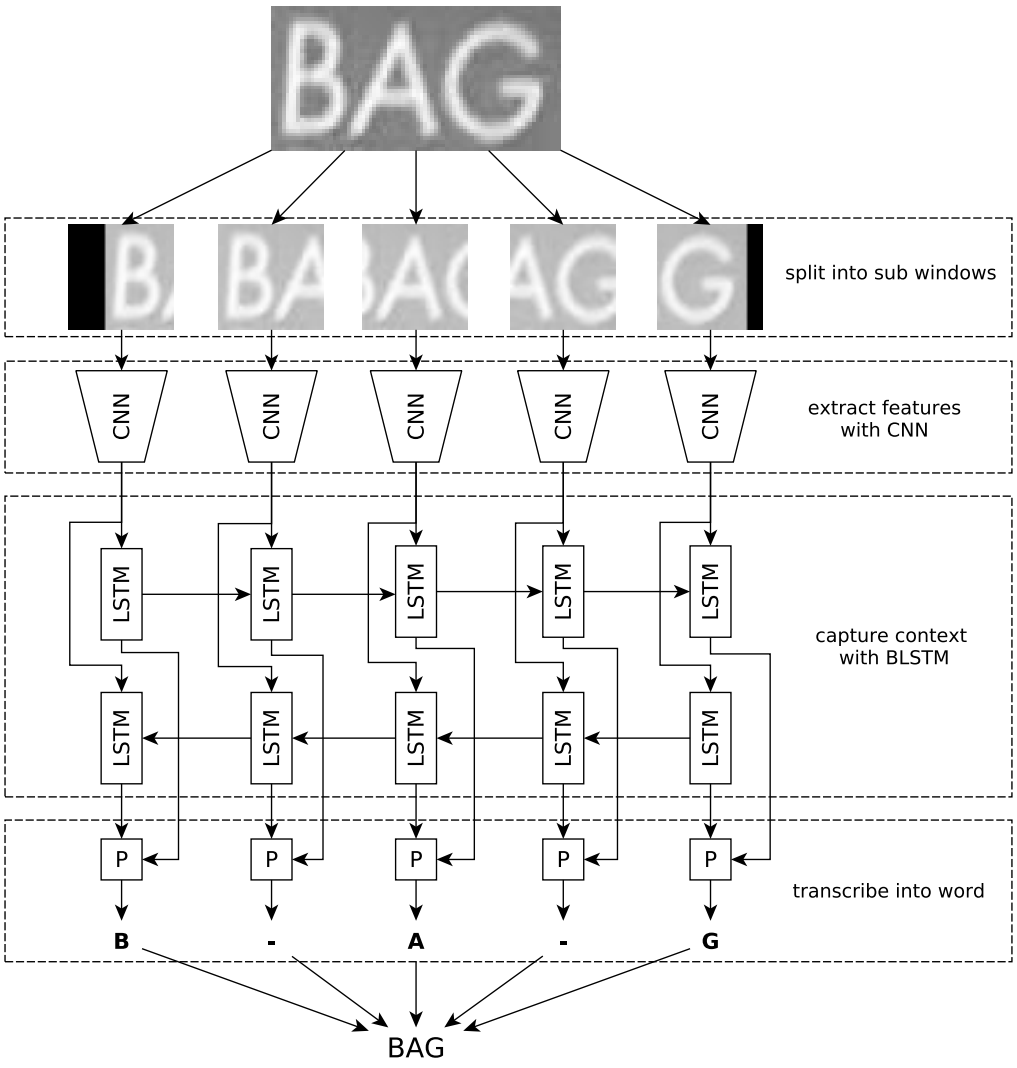


Figure 5.1.: Structural overview of sliding window approach for scene text recognition

item contains features that describe a character of the word the same loss function (CTC) as in the sliding window based approach is used here.

For the third approach we used the convolutional approach described in chapter 4 to train a convolutional feature extractor. On top of that feature extractor we added a LSTM that produces a sequence of characters given the image features and the last predicted character. This work is based on networks using BLSTMs for image captioning systems as proposed by Wang et al. [57]. The convolutional feature extractor of this network is a pre-trained CNN based on the AlexNet (see table 4.1 for more information about the network layout) where we used the features computed by the last layer as input features for the LSTM. The second input to the LSTM is the last character embedded in a word vector. The output of the network is the character that follows the last character or an end of sequence token. The computed image features and the last prediction are always fed into the network until the system observes the end of sequence token. The network is trained using softmax with loss at each time step. Figure 5.3 provides a graphical overview of this process.

5.2. System Configuration

We created individual networks for each of the three described approaches. The first two experiments were conducted using the deep learning framework Chainer [56]. The third experiment was conducted using the deep learning framework Caffe [29]. We used different network designs for the convolutional feature extractors and also performed experiments with BLSTMs of different depths.

5.2.1. Sliding Window Model

For our experiments with the network based on input generated by sliding a window across the text line image we conducted experiments with two different types of convolutional feature extractors. The first feature extractor used was a CNN using maxout [14] units as described in the original paper by He et al. We also conducted experiments using a CNN that is based on the VGGNet by Simonyan and Zisserman [52]. The maxout CNN consists of four convolutional layers followed by a maxout layer each. The CNN based on the VGGNet consists of seven convolutional layers, where each convolutional layer is followed by the ReLU nonlinearity. The first, second, fourth and sixth convolutional layers are followed by a $2x2$ max pooling with stride 2. A detailed description of the network design can be found in table 5.1.

5.2.2. Feature Extraction Model

The second approach we used for performing scene text recognition using RNNs takes the full text line image as input to the network. The authors of the original paper [51] suggest to scale the input text line image to a predefined height (32

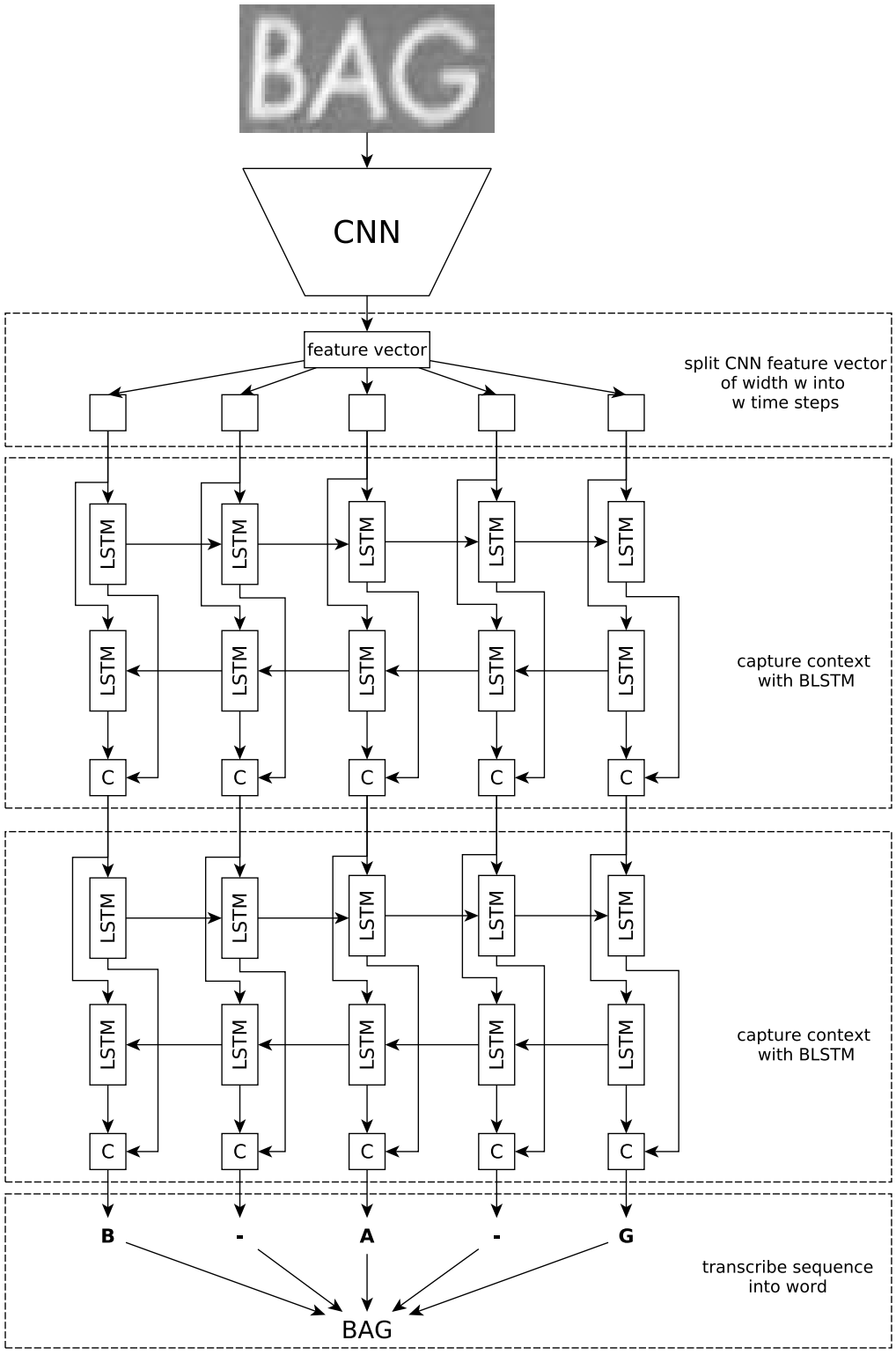


Figure 5.2.: Structural overview of feature extraction approach for scene text recognition

Type	input maps	output maps	kernel	stride	padding
Convolution	1	96	9 x 9	1	0
BatchNormalization	96	96			
Maxout	96	48	2		
Convolution	48	128	9 x 9	1	0
BatchNormalization	128	128			
Maxout	128	64	2		
Convolution	64	256	9 x 9	1	0
BatchNormalization	256	256			
Maxout	256	256	2		
Convolution	128	512	8 x 8	1	0
Maxout	512	128	4		
BLSTM	128	1024			
FullyConnected	1024	51			

(a) Maxout network structure

Type	input maps	output maps	kernel	stride	padding
Convolution	1	64	3 x 3	1	1
Max Pooling	64	64	2 x 2	2	0
Convolution	64	128	3 x 3	1	1
Max Pooling	128	128	2 x 2	2	0
Convolution	128	256	3 x 3	1	1
Convolution	256	256	3 x 3	1	1
Max Pooling	256	256	2 x 2	2	0
Convolution	256	512	3 x 3	1	1
Batch Normalization	512	512			
Convolution	512	512	3 x 3	1	1
Batch Normalization	512	512			
BLSTM	512	1024			
FullyConnected	1024	51			

(b) VGGNet like network structure

Table 5.1.: Network structures for the sliding window approach.

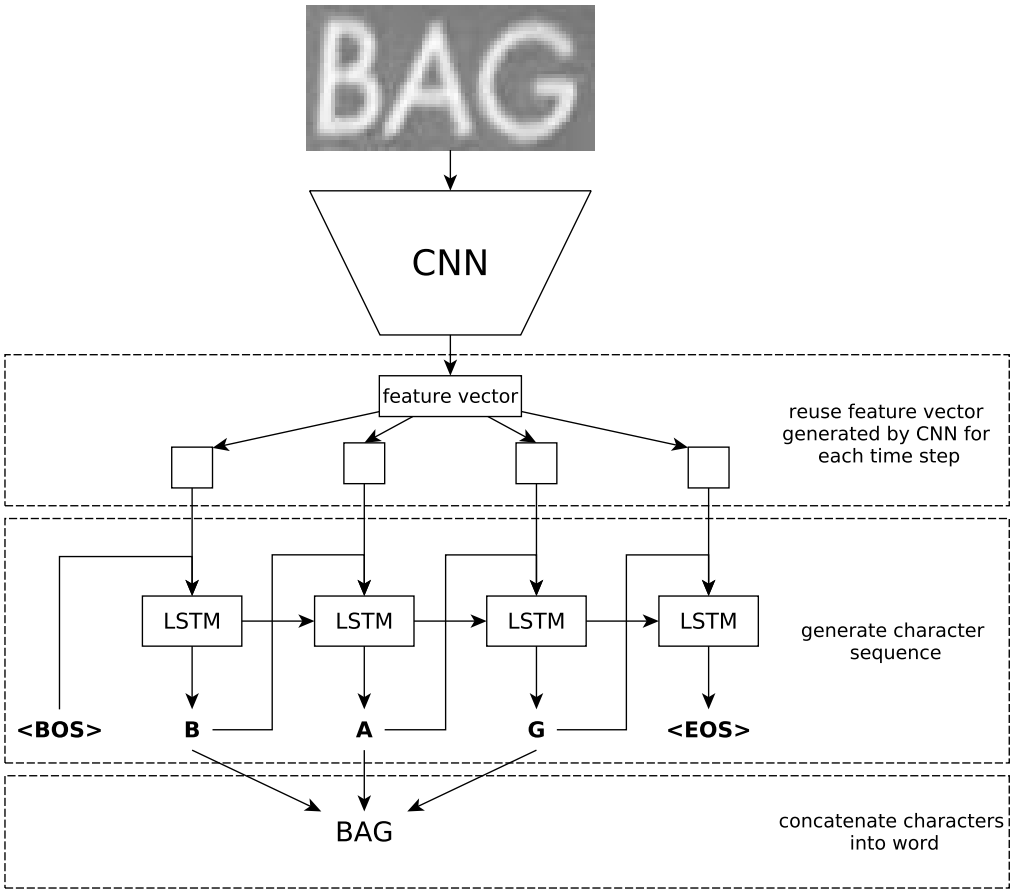


Figure 5.3.: Structural overview of CNN Sequence approach

pixels) while keeping the aspect ratio of the image. A CNN, similar to the VGGNet, that consists of seven convolutional layers is then used to extract a vector of features from the image. The feature extraction process produces a feature map with a height of one and a width corresponding to the sequence length which is used as the input to the recurrent part of the network. This shape of the feature map is achieved by using rectangular pooling windows and having input images that have a larger width than height. During the training of the network we followed the suggestion of the authors to fix the width of an input image to 100 pixels. This width generates a feature vector with 24 time steps, which is enough for most english words. A more detailed listing of the network configuration used for this approach can be found in table 5.2.

5.2.3. CNN Sequence Model

Our third approach also takes a full text line image as input an produces a sequence of characters as output. All input images are scaled to be 100 pixels wide and 32 pixels high. The network has been trained using Caffe with the LSTM

Type	input maps	output maps	kernel	stride	padding
Convolution	1	64	3 x 3	1	1
Max Pooling	64	64	2 x 2	2	0
Convolution	64	128	3 x 3	1	1
Max Pooling	128	128	2 x 2	2	0
Convolution	128	256	3 x 3	1	1
Convolution	256	256	3 x 3	1	1
Max Pooling	256	256	2 x 1	2, 1	0
Convolution	256	512	3 x 3	1	1
Batch Normalization	512	512			
Convolution	512	512	3 x 3	1	1
Batch Normalization	512	512			
Max Pooling	512	512	2 x 1	2, 1	0
Convolution	512	512	2 x 2	1	0
BLSTM	512	1024			
BLSTM	1024	1024			
FullyConnected	1024	51			

Table 5.2.: Network structure for the feature extraction model.

implementation by Jeff Donahue[1]. The convolutional network is based on an AlexNet [33]. The whole network is trained using softmax loss with SGD and the initial weights for the convolutional network have been provided by a former experiment with that network. Table 5.3 shows the parameters of all used layers.

We will describe all experiments that we conducted using the three different base approaches for scene text recognition with RNNs in chapter 6.

5.3. Implementation of Bidirectional LSTMs (BLSTMs) in Chainer

As Chainer does not include an implementation of bidirectional LSTMs we implemented this kind of recurrent network layer on our own. A bidirectional LSTM consists of two LSTMs. The first LSTM $\overrightarrow{L}$ always uses the past context $\overrightarrow{h}^{t-1}$ and produces a new context $\overrightarrow{h}^{t}$ for each time step t. The second LSTM $\overleftarrow{L}$ uses the future context $\overleftarrow{h}^{t+1}$ and produces a new context $\overleftarrow{h}^{t}$ for each time step t. this means that bidirectional LSTMs are able to create feature vectors that include past and future context for each time step making it possible for the LSTM to learn intra sequence dependencies. Our implementation adds a new layer to Chainer. This layer receives a sequence of CNN features which are then processed in forward and backward order by the BLSTM layer. The BLSTM layer returns two sequences of features. The first sequence is the result of the forward LSTM and the second sequence is the result of the backward LSTM. These feature sequences are then

[1]Caffe Pull request by Jeff Donahue - https://github.com/BVLC/caffe/pull/3948 - last access: 07/07/2016

Type	input maps	output maps	kernel	stride	padding
Convolution	1	64	5 x 5	1	0
Max Pooling	64	64	2 x 2	2	0
Convolution	64	128	5 x 5	1	2
Max Pooling	128	128	2 x 2	2	0
Convolution	128	256	3 x 3	1	1
Convolution	256	512	3 x 3	1	1
Max Pooling	512	512	2 x 2	2	0
Convolution	512	512	3 x 3	1	1
FullyConnected	24576	4096			
FullyConnected	4096	4096			
FullyConnected	4096	851			
WordVec	50	1000			
LSTM	1000	1000			
LSTM	1000 & 851	1000			
FullyConnected	1000	50			

Table 5.3.: Structural overview of CNN Sequence Model used for text recognition

concatenated and can be used as input to the next BLSTM or as input to a fully connected layer that maps each element of the sequence vector to a feature vector with as many elements as there are output classes. Listing A.1 shows the code used for implementing a BLSTM in Chainer.

6. Experiments and Evaluation

In this chapter we will show and discuss the results of the experiments we performed using the network designs and different approaches described in chapter 4 and chapter 5. We performed experiments that are meant to enable us to compare the different approaches. In the first section of this chapter we will describe the setup for all of our experiments. We will then show and discuss the results of the experiments with each of the introduced scene text recognition approaches. The next sections of this chapter deal with the evaluation of the trained models on publicly available text recognition benchmarks. We will first introduce the different benchmarks that we tested our models on. We will then evaluate the trained models on these datasets and compare the results to results of other researchers. This is followed by a discussion of the weaknesses of each approach. In the last section we will evaluate the real-time scene text recognition capabilities of each model by evaluating the inference time needed for recognizing a given scene text image.

6.1. Experiment Set-Up

We conducted most of our experiments using the deep learning framework Chainer [56]. All experiments that have not been conducted using Chainer have been conducted using the deep learning framework Caffe. We've implemented the specification of each network type and also added a new layer to the framework in order to work with BLSTMs. Our implementation of a BLSTM can be seen in section 5.3. For conducting our experiments we had access to three different machines equipped with CUDA compatible hardware [46]. The overall configuration of each machine is as follows:

MACHINE ONE The first machine has an Intel(R) Xeon(R) E5-1607 CPU, 8 GB of RAM and computations are running on a GeForce NVIDIA GTX 780 with 3 GB of graphic memory

MACHINE TWO The second machine has an Intel(R) Core(TM) i7-4790K CPU, 16 GB of RAM and computations are running on one of two NVIDIA GeForce GTX 980 with 4 GB of graphic memory each.

MACHINE THREE The third machine has an AMD FX-8370 CPU, 16 GB of RAM and computations are running on a NVIDIA GeForce GTX TITAN X with 12 GB of graphic memory.

For training the models we used a large dataset consisting of more than 10 000 000 training samples and 1 500 000 test samples. The largest part (7 200 000 training

samples and 890 000 test samples) was taken from the dataset provided by Jaderberg et al. [26, 25]. The rest consists of data generated by our data generation tool introduced in chapter 3. The data we added to the dataset of Jaderberg et al. was designed to overcome shortcomings of the data distribution we observed during the evaluation of our first models. One of the first shortcomings we noticed were that our first trained models based on the CNN approach were not able to recognize text lines with more than 15 individual characters. Therefore we created one million extra images containing words with a length of 16 to 23 characters and one million images containing words with a length of 11 to 23 characters. We further added more training samples with larger shadows, complex backgrounds (natural images), not so complex backgrounds (plain color) and also new words containing special characters like exclamation marks or brackets leaving us with the described amount of training data.

In order to achieve comparable results we used the following training environment for the training of each model:

Initial Weights The initial weights of each model have been randomly initialized using a scaled Gaussian distribution with scale factor 1.0[1]. This random initialization is used because this provides all models with a similar starting point for their parameters. It is not possible to use the same random numbers on each model as the number of parameters to fill varies a lot between each of the different model types.

Data Loading All training data is loaded using scipy [30]. All data is resized to the input dimensions as needed by the network specification. Resizing is done using the `imresize` function of scipy. The pixel values of each image are normalized to have a mean of 0. This is achieved by subtracting the mean of the image from each of the pixels in the image. This is done to improve the numerical stability of the network as proposed in [37].

Optimizer We used ADADELTA [64] as solver/optimizer for the training of each model. We chose ADADELTA as this solver is more robust to gradient spikes during training and also because it is not necessary to manually adapt the learning rate, making it easier to train the network.

Metrics During training we collected the following metrics:

- **Loss** The loss as a measure of how well the network is able to perform its task. See section 2.4.3 for more information on the role of loss during the training of a network.
- **Accuracy** The accuracy gives the percentage of how many words have been correctly recognized from each training or test mini batch.
- **Edit Distance** The edit distance represents the number of changes (insertions, deletions and substitutions) required to transform the recognized string into the ground-truth string. We utilize the Levenshtein distance [38] to determine this number of changes.

[1] HeNormal initializer in Chainer Docs: http://docs.chainer.org/en/v1.9.1/reference/initializers.html - last accessed 27/06/2016

End of Training We trained each network until we reached convergence. This means that we stopped the training if we were not able to see much improvement in accuracy and edit distance over the course of at least 10 000 iterations.

6.2. Experiments

In this work we were to compare the performance of several deep learning approaches to scene text recognition. We therefore conducted a number of different experiments with the approaches introduced in chapter 4 and chapter 5. In this section we will describe all experiments we conducted and interpret the training progress for each experiment.

6.2.1. Experiments with CNNs

In order to evaluate the performance of a end-to-end trained model based on a convolutional feature extractor with a fully connected classifier at the end we conducted two experiments. In our first experiment we generated a CNN that has been trained using the synthetic training data provided by Jaderberg et al. [26, 25]. We found that the model created with the provided training data did not achieve the results that Jaderberg et al reported. In order to increase the accuracy of the model on the test dataset we tried to fine-tune the already created model. We therefore added more and more training and test data to our dataset until we reached the number of training samples shown in section 6.1.

Besides fine-tuning with more data we thought that it might be beneficial to fine-tune the model by increasing the importance of the loss of individual classifiers, while decreasing the importance of other classifiers at the same time. We performed this kind of fine-tuning for multiple rounds always increasing the importance of individual classifiers that performed the worst. Using this approach we were able to increase the average accuracy of the 23 classifiers from 83 % to 95 % on the test set.

This experiment (***CNN-1***)was our first experiment with deep learning models for scene text recognition and left us with a model that performed very well on public benchmark datasets. Because we invested so much engineering effort into generating a good text recognition model we deemed that this model is not very good for a comparison with other models that just have been trained once without fine-tuning. That is why we conducted a second experiment with the CNN model.

In our second experiment (***CNN-2***) we used the same network layout as described in table 4.1, but we added BatchNormalization layers after each convolutional layer in order to make the model converge on the training data as described in section 4.3. We trained this model using Chainer. We used a batch size of 64 for training the model. We plotted the progress of the training in figure 6.1. The plot shows that the training starts with a model that is very good at recognizing text (edit distance was 8 at iteration 100). Over the course of time the train and test accuracy increased slowly and always stayed at the same level. This indicates that

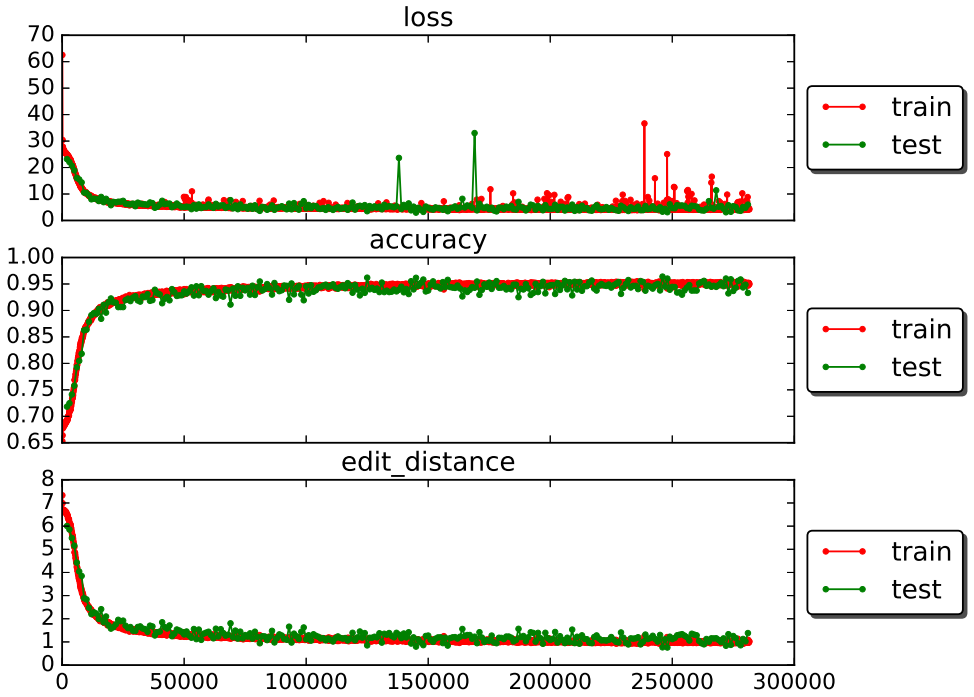


Figure 6.1.: Progress of experiment with CNN approach

the model does not over-fit on the training data and is able generalize the learned concepts. Besides the increase in accuracy, loss and edit distance decreased simultaneously also showing the success of the training. The final average edit distance value of around 2.3 paired with the high accuracy of around 90 % indicates that the model seems to work on a lot of examples very well but seems to fail on some others. We will evaluate the generated models on public datasets in section 6.4 and compare them to the other models that we will introduce in the next section.

6.2.2. Experiments with RNNs

The second family of approaches we experimented with were approaches that employ a convolutional feature extractor that extracts features that are used as input to a RNN. We found three different promising approaches in the literature. The first approach relies on a sliding window that is pushed across an input image and the second approach generates feature representations of an image containing a full text line. The third approach uses the RNN to generate a sequence of characters using the same extracted image features over and over again (see chapter 5 for further information about the approaches).

Sliding Window Approach

Using the sliding window approach we conducted four different experiments. The first two experiments where to compare the performance of the two different convolutional feature extractor designs for this network described in table 5.1. The networks used in the first experiments had only one BLSTM as proposed in the paper by He et al. [16]. The last two experiments again compare the performance of the feature extractors, but this time on a network with two BLSTMs stacked on top of each other. We performed all experiments on machine three and trained the models with training batches of size 30. Plots showing the progress of each experiment can be seen in figure 6.1.

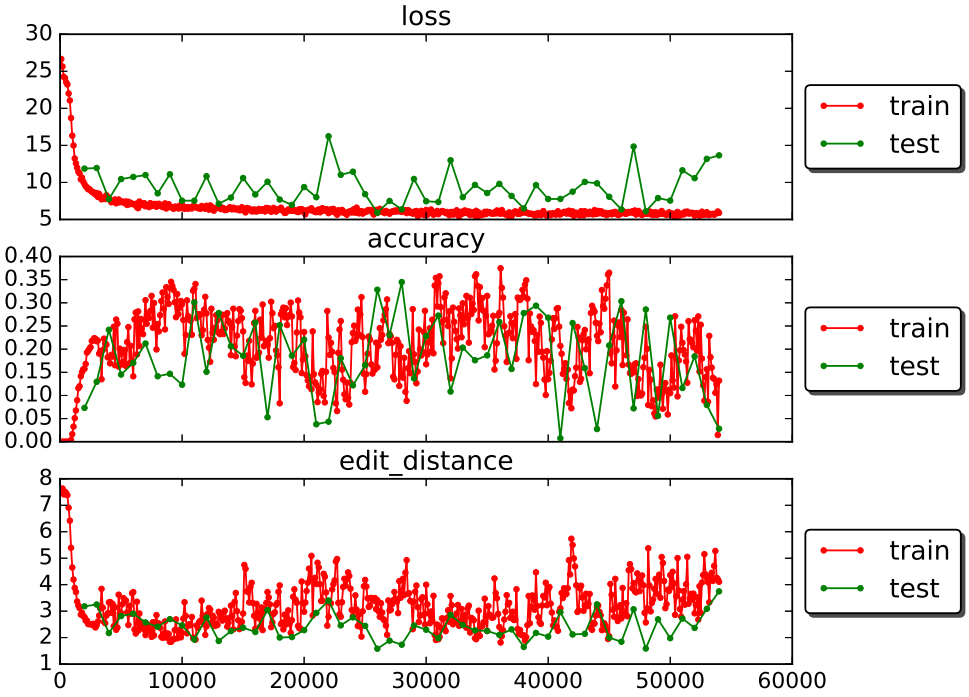
1. **Maxout CNN with one BLSTM (*SL-Maxout-1-BLSTM*)** The training progress of the first experiment where the network consisted of one maxout CNN with one BLSTM on top can be seen in figure 6.1 (a). In contrast to the curve of the CNN this network needs some more iterations until accuracy, loss and edit distance begin to improve. Over the course of time edit distance and accuracy do not decrease/increase in a constant way but rather spike around a certain value. This behavior is most likely due to the small batch size used for training and also because the network gets a lot of different input. On the one hand the network gets very short words that do not produce many time steps for training and on the other hand there are also batches with very long words, producing many time steps. These differences in the input lead to noisy inputs that the network needs to compensate.
2. **VGG CNN with one BLSTM (*SL-VGG-1-BLSTM*)** Figure 6.1 (b) shows the training progress of the network consisting of a convolutional feature extractor based on the VGG network by Simonyan and Zisserman [52]. The graph looks similar to the graph of the network using a maxout CNN as feature extractor. This network type also needs some time before it can recognize any text correctly and reduces the average edit distance per batch. Edit distance and accuracy also show a lot of spikes rather than a smooth curve, which is due to the same reasons as indicated before, but in contrast to the maxout feature extractor this feature extractor seems to be able to extract better features because loss and edit distance have smaller values over the time of the training. The network with a feature extractor based on the VGG net also seems to generalize better on the input data as it shows better performance on the test dataset as the network with the maxout CNN as feature extractor.
3. **Maxout CNN with two BLSTMs (*SL-Maxout-2-BLSTM*)** Figure 6.1 (c) shows the results of the training progress for the network with a maxout CNN feature extractor and two BLSTM layers on top of that feature extractor. This network achieves a higher accuracy on train and test set compared to the maxout network with only one BLSTM layer on top. Loss and edit distance are lower than loss and edit distance of the other model. It seems that adding a second BLSTM layer improves the performance of this network type.
4. **VGG CNN with two BLSTMs *(SL-VGG-2-BLSTM)*** In figure 6.1 (d) we show the training progress of the network with the VGG like feature extractor and

two BLSTMs on top. This plot shows that adding a second BLSTM on top of the first BLSTM seems to be beneficial, which stands in contrast to the hypothesis that adding more depth to the network does not increase the overall performance by He et al. Although the training loss stays the same as the loss for the experiment with only one BLSTM the overall accuracy is better and the average edit distance is also less noisy and lower than the edit distance of the former experiment (see figure 6.1 (b) for comparison).

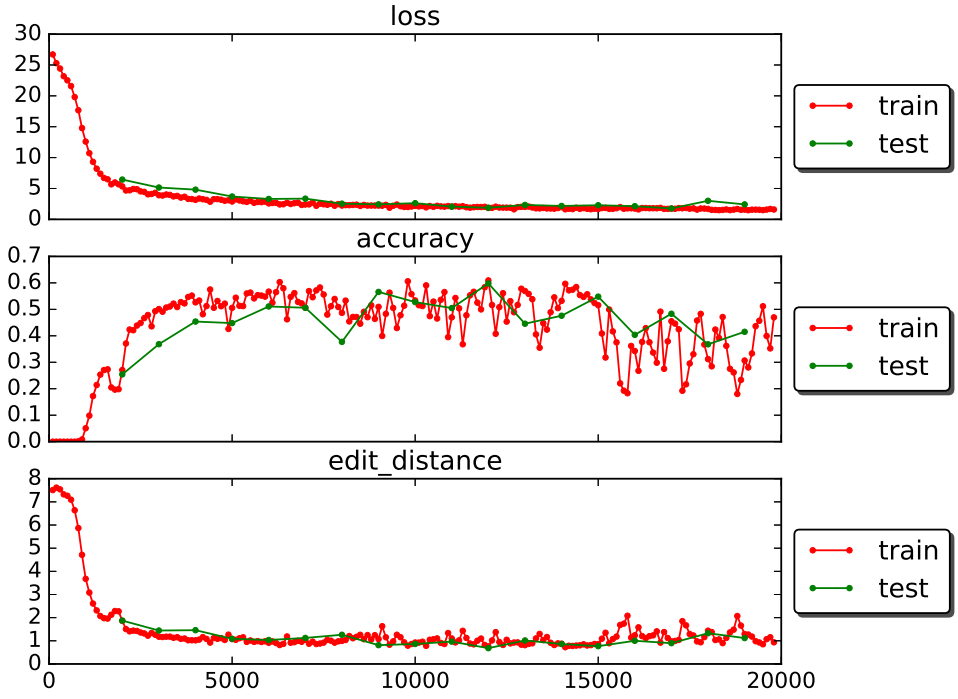
Feature Extraction Approach

We conducted four different experiments using the feature extraction approach. All experiments used the same basic layout shown in table 5.2. The only difference for each experiment was the number and type of RNNs on top of the convolutional feature extractor. In the first experiment we trained a model with only one LSTM on top of the feature extractor. In the second experiment we trained a model with a BLSTM on top of the feature extractor. In the third experiment we trained a model with two BLSTMs stacked on top of each other (the model layout as described by Shi et al. [51]). In the last experiment we wanted to see whether adding a third BLSTM can further improve the recognition accuracy of this approach. The plots of the training progress for each experiment can be found in figure 6.2.

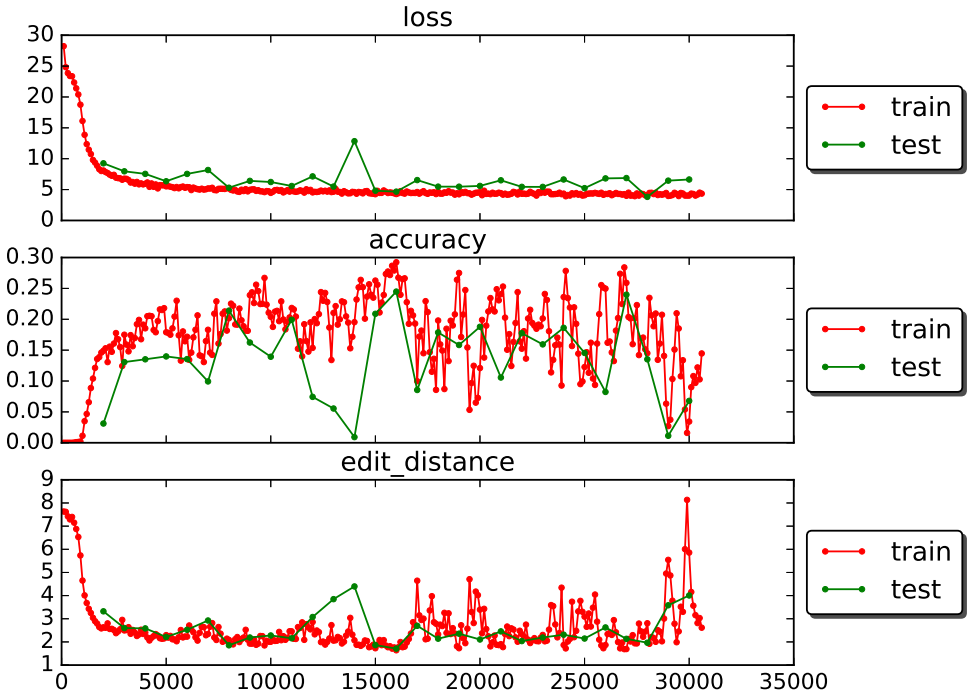
1. **Network with one LSTM *(FE-1-LSTM)*** In our first experiment we wanted to see how well a network with only one LSTM on top of the feature extractor can solve the problem of scene text recognition. Figure 6.2 (a) shows the plot of the training progress. Loss and edit distance plots show the same characteristics as the networks trained using the sliding window approach. The accuracy plot shows that the network seems to learn very stable feature representations leading to very good accuracy results of nearly 65 % accuracy on the test dataset.
2. **Network with one BLSTM *FE-1-BLSTM*** For our next experiment we exchanged the LSTM with a BLSTM. Figure 6.2 (b) shows the training progress of that network. This network shows the same characteristics as the network with only one LSTM, but the accuracy of the trained model is slightly better than the accuracy of the model with only one LSTM. This is due to the higher representational capacity of a BLSTM which includes past and future information for each time step which seems to beneficial for the performance of a model based on RNNs for scene text recognition.
3. **Network with two BLSTMs (*FE-2-BLSTM*)** Here we trained the network with the structure described in table 5.2. Figure 6.2 (c) shows the learning progress of this network. The plots show the same characteristics as for the first two experiments described. From this data it is not possible to conclude whether a second layer of BLSTMs helps to increase the performance of the trained model.
4. **Network with three BLSTMs *(FE-3-BLSTM)*** In our last experiment we trained a model consisting of three BLSTMs stacked on top of each other. The training results of this approach can be seen in figure 6.2 (d). Accuracy, loss and edit



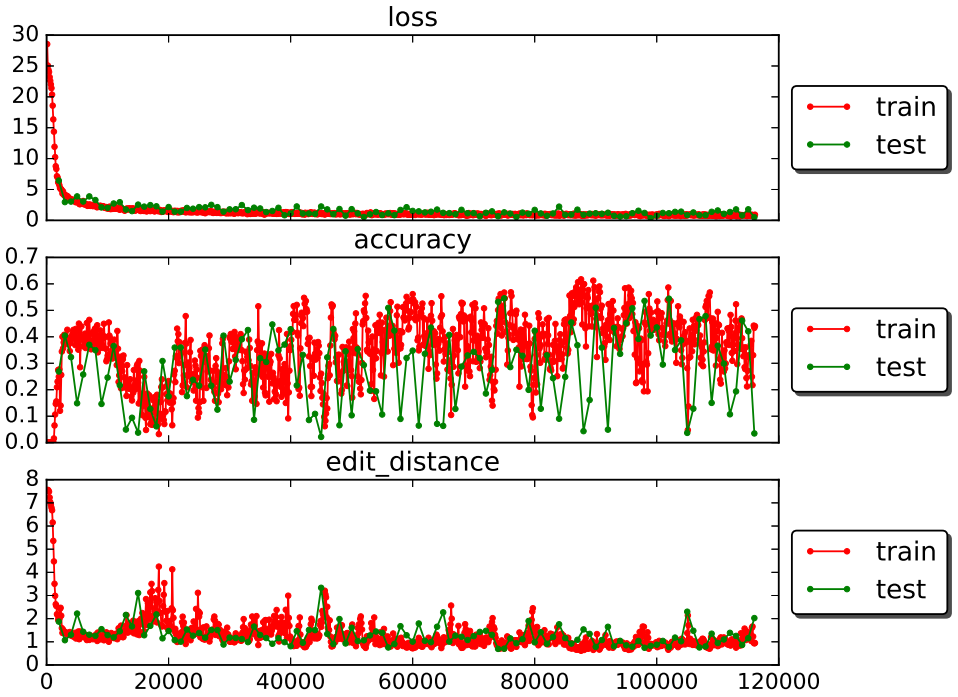
(a) Maxout CNN with one BLSTM layer on top



(b) VGG like CNN with one BLSTM layer on top



(c) Maxout CNN with two BLSTM layers on top



(d) VGG like CNN with two BLSTM layers on top

Figure 6.1.: Training progress of experiments with sliding window approach

distance show the same behavior as accuracy loss and edit distance did in the previous plots.

CNN Sequence Approach
With the CNN Sequence approach we only conducted one experiment. We started with a model trained on all data provided by Jaderberg et al [26, 25] and then we fine-tuned the model in a multi step approach where we added more and more data. We did this until the performance gain of further fine-tuning went near zero. In our experiments on real life benchmark datasets we show the results of two experiments with this approach. *CNN-S-1* is the model trained on only the data provided by Jaderberg et al. and *CNN-S-2* is the model fine-tuned on this model with more and more data in a multistep approach.

6.3. Public Benchmark Datasets

Over the course of years some public benchmark datasets have been released in order to make it possible to compare scene text recognition approaches implemented by different researchers.

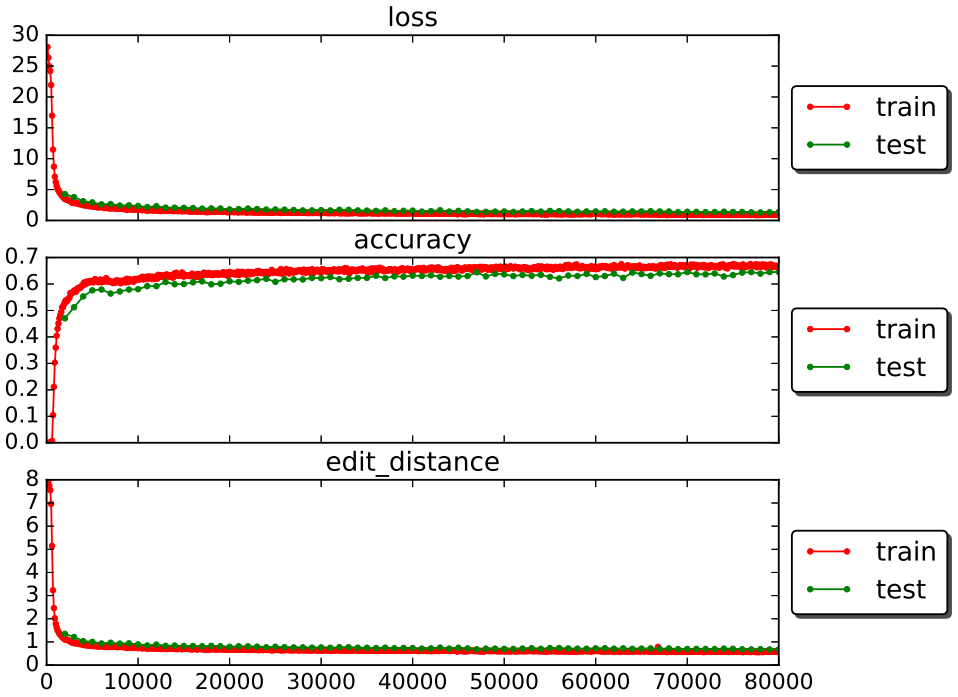
The dataset with the longest history is the ICDAR word recognition dataset from the robust reading competition which has been held since 2003 [40, 50, 32, 31]. The word recognition dataset contains images of cropped text lines. Over the course of the years this dataset has been adapted by exchanging and adding new images where it is more difficult to recognize the text. In our evaluation on the ICDAR dataset we used the ICDAR dataset for task 3 of the robust reading competition from 2015. The benchmark dataset contains 1095 text line images and the corresponding ground truth.

The second dataset we are going to use for evaluating the performance of the different approaches is the Street View Text (SVT) dataset introduced in 2010 by Wang et al. [59, 58]. This dataset contains 250 images from which 514 cropped text line images have been extracted. The images used in this dataset have been harvested from Google Street View. This dataset also provides a lexicon of 50 words per images, which can be used to further improve the recognition results.

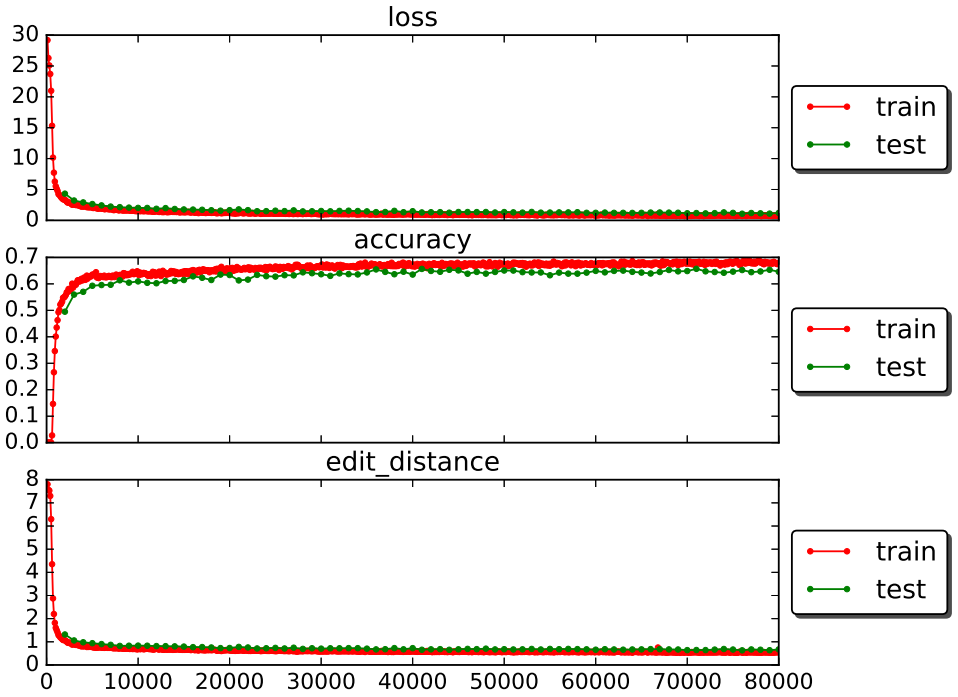
The third dataset we are using is the IIIT5K-word dataset by Mishra et al. [42]. This dataset contains 740 images from which 3000 cropped text line images have been extracted. The images have been harvested from Google image search by querying for words like billboards or signboards. This dataset also contains a small lexicon of 50 words per image and also larger lexicons with a size of 1000 words and 500 000 words.

For our evaluation we performed predictions on the text line images without any lexicon, with a small lexicons containing 50 words (if supplied by the dataset) and also with the standard OpenOffice dictionary for US American English[2] and its corresponding affix file, to which we added the words from each dataset (denoted

[2] http://extensions.openoffice.org/en/project/us-english-spell-checking-dictionary

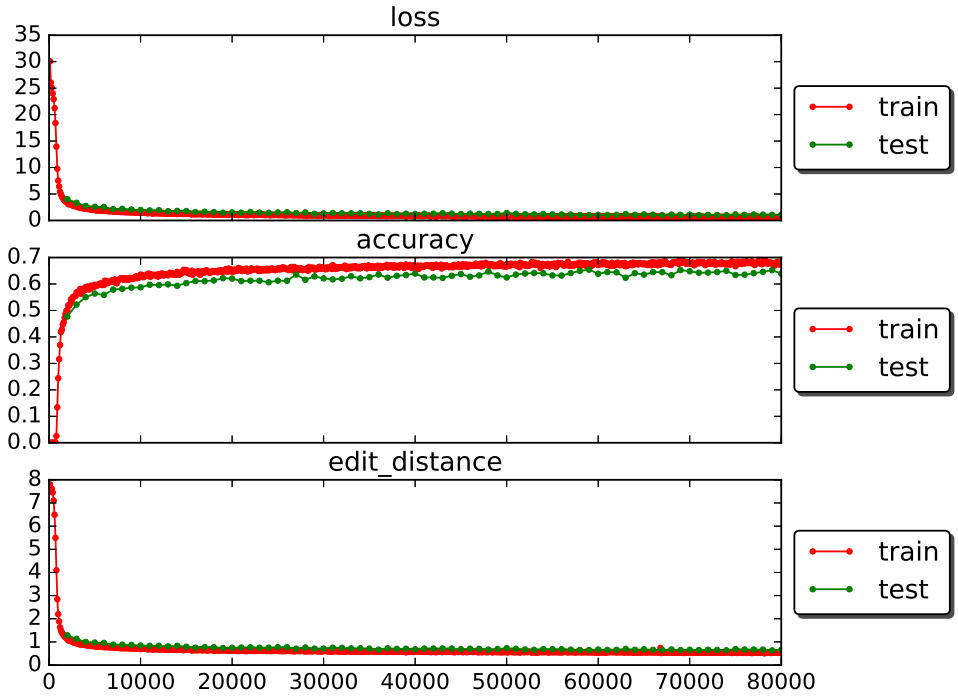


(a) training with one LSTM on top

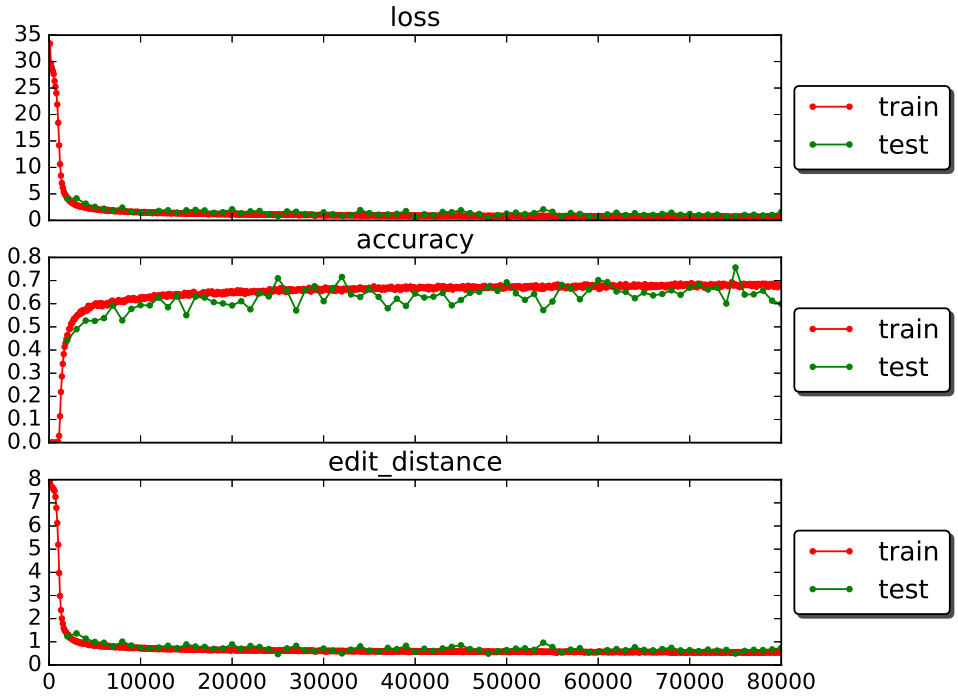


(b) training with single BLSTM on top

Figure 6.2.: Training progress of experiments with feature extraction approach



(c) training with two BLSTMs on top



(d) training with three BLSTMs on top

Figure 6.2.: Training progress of experiments with feature extraction approach

as *60K*). Where applicable we also report the result of other approaches or the original reported results by the authors of the paper where the approaches are based on.

6.4. Evaluation on Benchmark Datasets

We evaluate all models that are the results of the experiments we conducted and described in section 6.2.1 and section 6.2.2 on the datasets described in the last section. We compare the results of our evaluation with results reported by other researchers on the given datasets if the results posted by the researchers are directly comparable to the results we obtained. The recognition approach of Jaderberg et al. [24] for instance can not be compared to our approach on the ICDAR 2015 dataset as they did not use all images of the test set for their evaluation and did not support all types of characters contained in the images.

For a more compact layout of the evaluation table we provided a short name for each experiment which can be found at the beginning of the description of each experiment. The results of our evaluation can be found in table 6.1. Datasets like IIIT5K and SVT are mostly used by researchers to report the results of their experiments with scene text recognition. The ICDAR 2015 dataset however is also used by large companies like Google, Baidu and Megvii to showcase their recognition accuracies[3]. In table 6.2 we show that the result of our best method is still competitive to the results reported by large companies, regarding the fact that we only used synthetic data for training the network and are able to achieve real-time performance with our network (see section 6.5 for more information about our real time experiments).

The evaluation results clearly show that all approaches using LSTMs are superior to the approach using a CNN. The results of experiment *CNN-1* compared to experiment *CNN-2* show that a text recognition model using a CNN can heavily benefit from fine-tuning with more and more training data (9 % increase of accuracy on the ICDAR 2015 dataset). We can see the same behavior for our *CNN-s* experiments where we added an image-feature to sequence LSTM on top of the convolutional feature extractor. During fine-tuning we were able to increase the accuracy of this network by 13 % on the ICDAR benchmark dataset. We also find that the model of the experiment *CNN-S-2* provides the best results on nearly all benchmark datasets. This indicates that fine-tuning a model with more data can provide impressive performance gains.

All LSTM based experiments have shown to be indeed very good in learning relations of characters to each others, thus increasing the overall accuracy reached in our experiments. We find that the feature extraction approach is superior to the sliding window approach. We argue that this is due to the fact that the need for creating a sliding window with a fixed width might not capture the relevant image parts in the sliding windows, making it more difficult to correctly recognize the

[3] ICDAR 2015 Result table - http://rrc.cvc.uab.es/?ch=2&com=evaluation last access: 11/07/2016

Benchmark	IIIT5K		SVT		ICDAR 2015	
Lexicon size	50	None	50	None	60K	None
Wang et al. [60]	-	-	-	70.0	-	-
Mishra et al. [42]	64.1	-	73.2	-	-	-
Jaderberg et al. [28]	-	-	86.1	-	-	-
Bissacco et al. [4]	-	-	90.4	78.0*	-	85.3*
Jaderberg et al. [24]	95.0	-	93.5	68.0	-	-
Yao et al. [63]	-	-	-	-	-	86.0*
He et al. [16]	94.0	-	93.5	-	-	-
Shi et al. [51]	97.6	78.2	96.4	80.8	-	-
CNN - 1	97.2	71.9	93.2	65.7	79.4	73.1
CNN - 2	94.6	58.1	88.6	57.8	73.3	64.7
CNN-S-1	90.4	60.3	92.1	66.3	66.8	63.1
CNN-S-2	96.6	**73.2**	**95.3**	**75.9**	**82.6**	**76.3**
SL-Maxout-1-BLSTM	69.1	16.2	59.0	10.2	26.2	15.5
SL-VGG-1-BLSTM	97.2	64.2	90.4	57.4	77.8	68.6
SL-Maxout-2-BLSTM	90.0	46.9	76.6	34.0	54.2	44.7
SL-VGG-2-BLSTM	96.8	65.5	88.9	57.7	79.4	69.8
FE-1-LSTM	**98.3**	65.9	94.9	60.7	78.8	69.2
FE-1-BLSTM	97.3	65.5	94.9	63.9	78.2	69.9
FE-2-BLSTM	98.0	66.6	94.1	61.9	78.8	71.2
FE-3-BLSTM	**98.3**	68.6	94.5	61.8	80.6	72.4

Table 6.1.: Evaluation results on public benchmark datasets. *(Top)* Results reported by other researchers, - indicates that no results were reported or are not comparable to the results reported by us. *(Bottom)* Results of our experiments. Bold number indicate our best result and underlined numbers indicate our second best result
(* [4] and [63] are not lexicon free in the strict sense, as their outputs are constrained to large dictionaries used in a post processing step.)

Description	Result
SRC-B-TextProcessingLab	88.95 %
BAIDU-IDL	87.20 %
Megvii-Image++ [63]	86.03 %
PhotoOCR [4]	85.30 %
CNN-S-2 (our result)	82.60 %
NESP [35]	64.84 %
MAPS [36]	63.29 %
PLT [34]	63.11 %
PicRead	61.92 %
PIONEER	55.71 %
Feilds's Method	52.33 %
Baseline	46.58 %
TextSpotter	28.13 %

Table 6.2.: Reported accuracy results on the ICDAR 2015 Competition on robust reading (Challenge 2, Task 3). The baseline has been set using a commercially available OCR system (ABBYY FineReader[4]).

text contained in the image. We furthermore find that the original sliding window approach by He et al. that uses a maxout network as feature extractor and only one BLSTM on top of that feature extractor did not perform as well as described in the according publication. This might be due to the fact that we did not perform a pre-training of the CNN part of the model as described in the paper. We do find that exchanging the maxout feature extractor with a feature extractor based on the VGGNet is superior and also exceeds the accuracy reported by He et al. on the IIIT5K dataset with a lexicon of 50 words. Adding a second BLSTM layer on top of the first BLSTM layer provides a great benefit for the network based on a maxout feature extractor which indicates that the second BLSTM helps the network to extract more useful information from the features extracted by the convolutional part of the network.

All experiments with the feature extraction approach show good results and their results are not very far from each other when recognized words have been constrained to a lexicon. In the unconstrained case we find that adding more depth to the network increases the recognition accuracy slightly. The best performing model was the feature extraction model with 3 BLSTMs on top of the feature extractor. This model nearly reaches the unconstrained recognition accuracies of the fine-tuned CNN model from experiment *CNN-1* showing that using LSTMs leads to a superior performance in text recognition. This model also nearly reaches the recognition accuracy of the fine-tuned *CNN-S-2* model, showing that this model architecture will be superior to all other architectures if fine-tuned accordingly. Furthermore this model exceeds the recognition accuracies on the IIIT5K dataset with a lexicon of 50 words reported by other researchers, although the recognition accuracy with no lexicon is by far not as good as reported by other researchers.

Figure 6.3.: Sample of a challenging image for the sliding window approach. The model correctly recognizes the word *GREEK*, but also recognizes the characters between the capital letters of the word *GREEK*. Approaches like the feature extraction approach do not get that confused when confronted with such an image.

We argue that this shows that this model is already generating good recognition results that contain only slight mistakes for words in the given dataset.

Weaknesses of approaches

Besides their already good performance on the benchmark datasets each approach has its weaknesses that can be addressed for further improvement by fine-tuning the already generated models.

The sliding window method seems to work very well on a large amount of different tasks. But we encountered some problems that are immanent to this method. The first problem is that the sliding window model does not work very well if the space between single characters is very large and the sub window does not contain a letter at all or only very few parts of two letters. In this case it may happen that the model recognizes characters that are not present in the image. This problem also exists if the model has to recognize text from a text line where two different text lines are mixed, the model will then recognize text from both text lines in a single recognition step (see figure 6.3 for an example of such an image). We also observed that very noisy backgrounds also lead to wrong recognition results of the model. Here the model predicts extra characters between other characters.

The feature extraction method is more robust to spaces between characters and noisy backgrounds but suffers from correctly recognizing double letters like *ll* or *oo*. This happens because the CTC path generated by the network does not have a strong peak for the no letter class in this case. Figure 6.4 illustrates this problem by showing the activation of the output layer of the last LSTM for the image that contains the word *GREEK* that we showed in figure 6.3. We think that we can mitigate this problem by adding more samples with double letters to our training data.

The CNN approach is not good when dealing with words longer than eleven characters. The reason for this might be that the mean length of english words is at around eight characters and that our training set does not contain enough long words. This approach also fails on very short words which is very likely based on the same problem.

The results of the CNN Sequence approach show that the characters seen at the beginning of a word heavily influence the predictions for characters at the end of a

[4] https://www.abbyy.com/finereader/

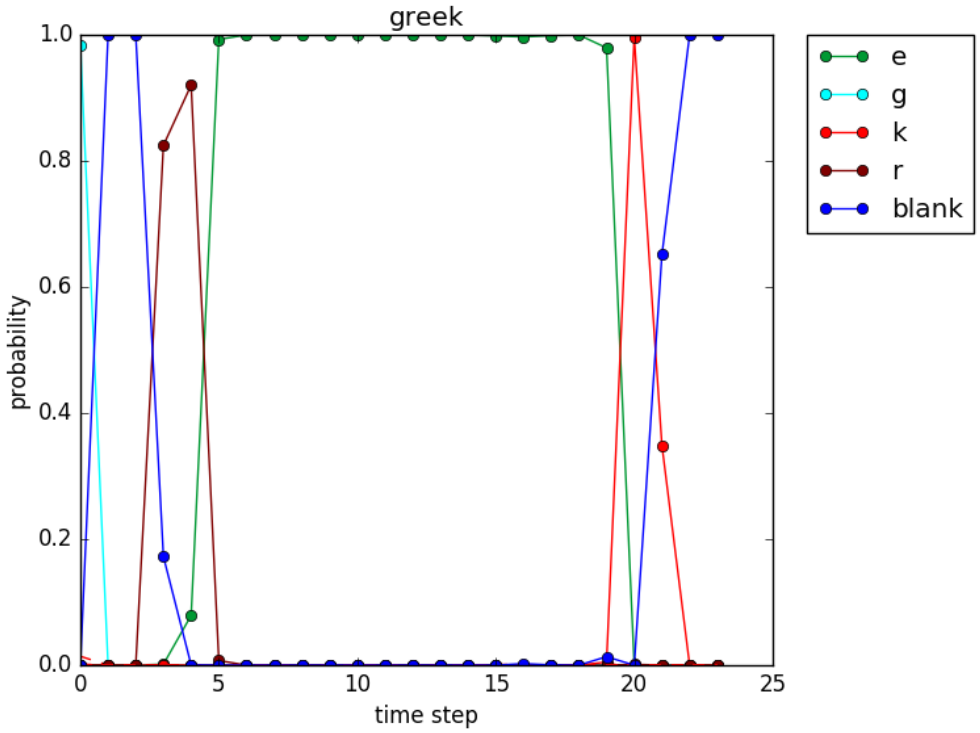


Figure 6.4.: Output of the last LSTM for each time step when trying to recognize the text of the image shown in figure 6.3. It is possible to see that the blank label is not at all predicted in between the two e characters that are contained in the image. This leads to the observed problem that the feature extraction models are not able to correctly recognize words with double letters.

word. This is due to the fact that the input to the LSTM at each time step consists of the last predicted character and the image features extracted from the image. This is different to the input of the feature extraction approach where the input to the LSTM for each time step is only a part of the extracted feature vector. It might be possible to break the dependency on only the last predicted letters by using a BLSTM instead of a LSTM for the CNN Sequence approach.

6.5. Evaluation of Real-Time capability

Besides an evaluation of accuracy for each experiment we also performed an evaluation based on execution speed. In this evaluation we wanted to see how fast each model is able to generate its prediction given a live scene that always consists of the same view for the camera plugged into the benchmark pc. The view of the camera used to evaluate all models is shown in figure 6.5. For this experiment we created a tool that is able to read text from live images. This tool performs text detection and text recognition on a live scene captured by a webcam. The text detection stage is based on the text detection stage introduced by Yang et al. [61].

Figure 6.5.: Live Demo Scene for evaluating real time performance of generated models

In the first step text candidates are produced using MSERs, these text candidates are then verified using a text detection model based on a convolutional feature extractor.

We conducted the evaluation in the following way:

1. loading of model and first iterations of model until execution time is stable
2. recording of execution time of model for 100 forward passes
3. calculation of mean and standard deviation of recorded execution times

We performed this evaluation on a machine with the following hardware specification: *CPU:* Intel(R) Xeon(R) E5-1607, *RAM:* 8 GB, *GPU:* NVIDIA GTX 780 with 3 GB of graphic memory.

We found that the model solely based on a CNN feature extractor is by far the fastest model. This is due to the fact that the models based on LSTMs perform more calculations that can not be parallelized. The second fastest models where our models that are based on the feature extraction approach. Using more than a single LSTM on top of the feature extractor leads to a significant increase of execution time. This is again due to the fact that is not easy to parallelize the execution of a BLSTM. Our implementation of a BLSTM also contains a lot of python code that has to run on a CPU which might involve a lot of memory copying from the GPU to the CPU, making the use of BLSTMs slow down the execution speed of the network. The models based on the sliding window approach are very slow

Model	Time	Standard Deviation
CNN-1	38.0 *ms*	1.7 *ms*
FE-1-LSTM	62.8 *ms*	1.7 *ms*
FE-1-BLSTM	94.4 *ms*	2.2 *ms*
FE-2-BLSTM	146.6 *ms*	3.2 *ms*
FE-3-BLSTM	205.8 *ms*	3.7 *ms*
SL-Maxout-1-BLSTM	288.5 *ms*	5.0 *ms*
SL-VGG-1-BLSTM	271.2 *ms*	6.6 *ms*
SL-Maxout-2-BLSTM	352.8 *ms*	5.2 *ms*
SL-VGG-2-BLSTM	338.6 *ms*	6.8 *ms*

Table 6.3.: Real-Time evaluation results for all of our experiments with model trained from scratch

because they have to run the convolutional feature extractor several times before the recurrent part of the network can be used. The models relying on a maxout feature extractor are slower than the VGG like feature extractors because they use very large convolutional kernels (9x9 kernel compared to 3x3 kernels) that slow down the execution speed. We did not evaluate the execution speed of the *CNN-S* models because we were not able to use the models generated with Caffe in our real-time evaluation tool based on Chainer.

7. Conclusion and Future Work

7.1. Conclusion

In this work we have studied the problem of scene text recognition as a means for information extraction from multimedia data such as photos and videos. Scene text recognition hereby differs from traditional text recognition of printed text in documents as texts in scene images come in various fonts, with various backgrounds, often with low resolution or with challenging contrasts making it difficult for traditional print OCR engines to recognize the text contained in a text line extracted from a real world image.

The contributions of this work can be concluded as follows:

1. We developed a data generation tool that is capable of producing large amounts of synthetic training data for the training of scene text recognition systems using DNNs. The developed data generation tool is highly configurable and can render texts in arbitrary sizes, colors, fonts, rotations, perspective distortions, with a wide range of reflection effects and backgrounds. The synthetic data produced by our data engine is thereby virtually indistinguishable from real-world data (see figure 3.3 for a comparison of real-world data and generated data), making it very well suited as replacement for real world training data that is nearly impossible to acquire in the quantities necessary to successfully train a DNN.
2. We have used the data generated by the data generation tool to train and evaluate different deep learning approaches to scene text recognition. We conducted several experiments and with different deep learning approaches to see the impact a changed network structure has on a deep model.
   We conducted experiments with a network layout based only on CNNs, a CNN Sequence model based on LSTMs, that uses features extracted by a convolutional feature extractor to generate a sequence of characters that represent the word contained in the given input image. We furthermore conducted experiments with other types of RNNs that utilize convolutional feature extractors and deep BLSTMs. We found that using RNNs leads to a significant increase in recognition accuracy. The recognition results produced by networks with RNNs have also shown to be more stable than recognition results produced by a CNN only network.
3. We showed that fine-tuning an initially trained model can further increase the performance of that model on its task. In our experiments we found an increase of recognition accuracy on the test dataset of up to 13 %. This finding shows

that creating a good performing deep neural network still requires a lot of engineering effort.

4. We included our evaluated scene text recognition approaches in a tool that can perform end-to-end scene text recognition in real-time using the models we created during our experiments with deep learning approaches for scene text recognition. While evaluating the recognition speed of the models created by each experiment we found that approaches utilizing a sliding window approach for feature extraction are not very well suited for real-time scene text recognition.

In chapter 2 we first reviewed the history of OCR and introduced the general workflow of a typical print OCR system. We then showed why there is a need for a different approach on OCR for scene text recognition. We then introduced the fundamentals of DNNs and presented related work in the field of scene text recognition systems.

In chapter 3 we showcased our data generation tool that we created to generate tailor-made training data for the creation of our deep neural models that we described in chapter 4 and chapter 5.

In chapter 6 we evaluated the models we described earlier and showed that DNNs using RNNs outperform CNN based approaches on scene text recognition.

7.2. Future Work

Current research is mostly focused on creating scene text recognition systems for latin script. Scripts like chinese script with its large amount of different symbols are out of scope. There are claims [51] that the developed systems may be directly applicable to other types of script. We want to see whether these assumptions hold by conducting experiments with our feature extraction model using samples that contain chinese script.

For now we have only focused on examining still images for the purpose of scene text recognition. We think that we are loosing a lot of information if we want to recognize text in a video sequence using only still images. We think that using multiple frames that contain the same text line as input to a text recognition network that operates on a sequence of input images could further improve the achieved recognition accuracy.

Currently text recognition can be seen as an isolated part of an end-to-end scene text recognition system. We plan to create a deep neural network that can perform every step of an end-to-end system (text detection, bounding box regression, text recognition, post processing) in one step and in real-time. We think that we can utilize current research advances in real-time object detection and recognition to achieve this goal.

Bibliography

[1] Jon Almazan, Albert Gordo, Alicia Fornes, and Ernest Valveny. "Word Spotting and Recognition with Embedded Attributes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.12 (2014), pp. 1–1. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2014.2339814.

[2] Ian Goodfellow Yoshua Bengio and Aaron Courville. "Deep Learning". 2016.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, Université De Montréal, and Montréal Québec. "Greedy layer-wise training of deep networks". In: *In NIPS*. MIT Press, 2007.

[4] Alessandro Bissacco, Mark Cummins, Yuval Netzer, and Hartmut Neven. "PhotoOCR: Reading Text in Uncontrolled Conditions". In: 2013, pp. 785–792.

[5] James F. Blinn. "A Generalization of Algebraic Surface Drawing". In: *ACM Trans. Graph.* 1.3 (1982), pp. 235–256. ISSN: 0730-0301. DOI: 10.1145/357306.357310.

[6] Augustin Cauchy. "M{é}thode g{é}n{é}rale pour la r{é}solution des systemes d'{é}quations simultan{é}es". In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538.

[7] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. "Handwritten digit recognition: applications of neural network chips and automatic learning". In: *IEEE Communications Magazine* 27.11 (1989), pp. 41–46. ISSN: 0163-6804. DOI: 10.1109/35.41400.

[8] G. Cybenko. "Approximation by superpositions of a sigmoidal function". en. In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314. ISSN: 0932-4194, 1435-568X. DOI: 10.1007/BF02551274.

[9] E. E. Fournier d'Albe. "On a Type-Reading Optophone". en. In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 90.619 (1914), pp. 373–375. ISSN: 1364-5021, 1471-2946. DOI: 10.1098/rspa.1914.0061.

[10] Line Eikvil. *Optical character recognition*. 1993.

[11] B. Epshtein, E. Ofek, and Y. Wexler. "Detecting text in natural scenes with stroke width transform". In: *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010, pp. 2963–2970. DOI: 10.1109/CVPR.2010.5540041.

[12] Xavier Glorot, Yoshua Bengio, and Antoine Bordes. "Deep sparse rectifier neural networks". In: *International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323.

[13] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". In: *arXiv:1312.6082 [cs]* (2013).

[14] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout Networks". In: *arXiv:1302.4389 [cs, stat]* (2013).

[15] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. New York, NY, USA: ACM, 2006, pp. 369–376. ISBN: 1595933832. DOI: 10.1145/1143844.1143891.

[16] Pan He, Weilin Huang, Yu Qiao, Chen Change Loy, and Xiaoou Tang. "Reading Scene Text in Deep Convolutional Sequences". In: *arXiv:1506.04395 [cs]* (2015).

[17] G.E. Hinton and T.J. Sejnowski. "Learning and Relearning in Boltzmann Machines". In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1 (1968), pp. 282–317.

[18] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Comput.* 18.7 (2006), pp. 1527–1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527.

[19] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[20] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jurgen Schmidhuber, and Corso Elvezia. "Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies". In: (2001).

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.

[22] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]* (2015).

[23] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Structured Output Learning for Unconstrained Text Recognition". In: *arXiv:1412.5903 [cs]* (2014).

[24] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Structured Output Learning for Unconstrained Text Recognition". In: *arXiv:1412.5903 [cs]* (2014).

[25] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Reading Text in the Wild with Convolutional Neural Networks". en. In: *International Journal of Computer Vision* 116.1 (2015), pp. 1–20. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-015-0823-z.

[26] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition". In: *arXiv:1406.2227 [cs]* (2014).

[27] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Deep Features for Text Spotting". en. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars. Lecture Notes in Computer Science 8692. DOI: 10.1007/978-3-319-10593-2_34. Springer International Publishing, Sept. 2014, pp. 512–528. ISBN: 978-3-319-10592-5 978-3-319-10593-2.

[28] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Deep Features for Text Spotting". en. In: *Computer Vision - ECCV 2014*. Ed. by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars. Lecture Notes in Computer Science 8692. Springer International Publishing, 2014, pp. 512–528. ISBN: 978-3-319-10592-5 978-3-319-10593-2.

[29] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22Nd ACM International Conference on Multimedia*. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678. ISBN: 9781450330633. DOI: 10.1145/2647868.2654889.

[30] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001.

[31] D. Karatzas, L. Gomez-Bigorda, A. Nicolaou, S. Ghosh, A. Bagdanov, M. Iwamura, J. Matas, L. Neumann, V. R. Chandrasekhar, S. Lu, F. Shafait, S. Uchida, and E. Valveny. "ICDAR 2015 competition on Robust Reading". In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. 2015, pp. 1156–1160. DOI: 10.1109/ICDAR.2015.7333942.

[32] Dimosthenis Karatzas, Faisal Shafait, Seiichi Uchida, Masakazu Iwamura, Lluis Gomez i Bigorda, Sergi Robles Mestre, Joan Mas, David Fernandez Mota, Jon Almazan Almazan, and Lluis Pere de las Heras. "ICDAR 2013 robust reading competition". In: *2013 12th International Conference on Document Analysis and Recognition*. IEEE, 2013, pp. 1484–1493.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.

[34] D. Kumar and A. G. Ramakrishnan. "Power-law transformation for enhanced recognition of born-digital word images". In: *2012 International Conference on Signal Processing and Communications (SPCOM)*. 2012, pp. 1–5. DOI: 10.1109/SPCOM.2012.6290009.

[35] Deepak Kumar, M. N. Anil Prasad, and A. G. Ramakrishnan. "NESP: Nonlinear enhancement and selection of plane for optimal segmentation and recognition of scene word images". In: vol. 8658. 2013, DOI: 10.1117/12.2008519.

[36] Deepak Kumar, M. N. Anil Prasad, and A. G. Ramakrishnan. "MAPS: Midline Analysis and Propagation of Segmentation". In: *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*. ICVGIP '12. New York, NY, USA: ACM, 2012, 15:1–15:7. ISBN: 9781450316606. DOI: 10.1145/2425333.2425348.

[37] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. "Efficient BackProp". en. In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Lecture Notes in Computer Science 7700. Springer Berlin Heidelberg, 2012, pp. 9–48. ISBN: 978-3-642-35288-1 978-3-642-35289-8.

[38] Vladimir I Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals". In: *Soviet physics doklady*. Vol. 10. 1966, p. 707.

[39] Tsungnan Lin, B. G. Horne, P. Tino, and C. L. Giles. "Learning long-term dependencies in NARX recurrent neural networks". In: *IEEE Transactions on Neural Networks* 7.6 (1996), pp. 1329–1338. ISSN: 1045-9227. DOI: 10.1109/72.548162.

[40] Simon M. Lucas, Alex Panaretos, Luis Sosa, Anthony Tang, Shirley Wong, Robert Young, Kazuki Ashida, Hiroki Nagai, Masayuki Okamoto, Hiroaki Yamamoto, Hidetoshi Miyao, JunMin Zhu, WuWen Ou, Christian Wolf, Jean-Michel Jolion, Leon Todoran, Marcel Worring, and Xiaofan Lin. "ICDAR 2003 robust reading competitions: entries, results, and future directions". en. In: *International Journal of Document Analysis and Recognition (IJDAR)* 7.2-3 (2005), pp. 105–122. ISSN: 1433-2833, 1433-2825. DOI: 10.1007/s10032-004-0134-3.

[41] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". en. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02478259.

[42] Anand Mishra, Karteek Alahari, and Cv Jawahar. "Scene Text Recognition using Higher Order Language Priors". en. In: British Machine Vision Association, 2012, pp. 127.1–127.11. ISBN: 1901725464. DOI: 10.5244/C.26.127.

[43] Richard S. Morgan. "Optical readers: 1970". en. In: *Computers and the Humanities* 5.2 (1970), pp. 75–78. ISSN: 0010-4817, 1572-8412. DOI: 10.1007/BF02402284.

[44] L. Neumann and J. Matas. "Real-time scene text localization and recognition". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012, pp. 3538–3545. DOI: 10.1109/CVPR.2012.6248097.

[45] Lukas Neumann and Jiri Matas. "A Method for Text Localization and Recognition in Real-World Images". en. In: *Computer Vision - ACCV 2010*. Ed. by Ron Kimmel, Reinhard Klette, and Akihiro Sugimoto. Lecture Notes in Computer Science 6494. Springer Berlin Heidelberg, 2010, pp. 770–783. ISBN: 978-3-642-19317-0 978-3-642-19318-7.

[46] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. "Scalable Parallel Programming with CUDA". In: *Queue* 6.2 (2008), pp. 40–53. ISSN: 1542-7730. DOI: 10.1145/1365490.1365500.

[47] Christopher Olah. *Understanding LSTM Networks*. 2015.

[48] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471(Electronic);0033-295X(Print). DOI: 10.1037/h0042519.

[49] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". en. In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[50] A. Shahab, F. Shafait, and A. Dengel. "ICDAR 2011 Robust Reading Competition Challenge 2: Reading Text in Scene Images". In: *2011 International Conference on Document Analysis and Recognition*. 2011, pp. 1491–1496. DOI: 10.1109/ICDAR.2011.296.

[51] Baoguang Shi, Xiang Bai, and Cong Yao. "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition". In: *arXiv:1507.05717 [cs]* (2015).

[52] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv:1409.1556 [cs]* (2014).

[53] "Statistical machine". 1838389. Dec. 1931.

[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions". In: *arXiv:1409.4842 [cs]* (2014).

[55] Simon J. Thorpe and Michèle Fabre-Thorpe. "Seeking Categories in the Brain". en. In: *Science* 291.5502 (2001), pp. 260–263. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1058249.

[56] Seiya Tokui, Kenta Oono, Shohei Hido, CA San Mateo, and Justin Clayton. *Chainer: a Next-Generation Open Source Framework for Deep Learning*.

[57] Cheng Wang, Haojin Yang, Christian Bartz, and Christoph Meinel. "Image Captioning with Deep Bidirectional LSTMs". In: *arXiv:1604.00790 [cs]* (2016).

[58] Kai Wang, B. Babenko, and S. Belongie. "End-to-end scene text recognition". In: *2011 International Conference on Computer Vision*. 2011, pp. 1457–1464. DOI: 10.1109/ICCV.2011.6126402.

[59] Kai Wang and Serge Belongie. "Word Spotting in the Wild". en. In: *Computer Vision - ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Lecture Notes in Computer Science 6311. Springer Berlin Heidelberg, 2010, pp. 591–604. ISBN: 978-3-642-15548-2 978-3-642-15549-9.

[60] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng. "End-to-end text recognition with convolutional neural networks". In: *2012 21st International Conference on Pattern Recognition (ICPR)*. 2012, pp. 3304–3308.

[61] Haojin Yang, Cheng Wang, Xiaoyin Che, Sheng Luo, and Christoph Meinel. "An Improved System For Real-Time Scene Text Recognition". In: *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. ICMR '15. New York, NY, USA: ACM, 2015, pp. 657–660. ISBN: 9781450332743. DOI: 10.1145/2671188.2749352.

[62] Cong Yao, Xiang Bai, Baoguang Shi, and Wenyu Liu. "Strokelets: A Learned Multi-Scale Representation for Scene Text Recognition". In: 2014, pp. 4042–4049.

[63] Cong Yao, Jianan Wu, Xinyu Zhou, Chi Zhang, Shuchang Zhou, Zhimin Cao, and Qi Yin. "Incidental Scene Text Understanding: Recent Progresses on ICDAR 2015 Robust Reading Competition Challenge 4". In: *arXiv:1511.09207 [cs]* (2015).

[64] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *arXiv:1212.5701 [cs]* (2012).

Appendix A.

Listings

```
class BLSTMLayer(chainer.Chain):
  def __init__(self, n_units=512, input_units=None):
    super(BLSTMLayer, self).__init__(
      forward_lstm=L.LSTM(input_units if input_units is not None else n_units, n_units),
      backward_lstm=L.LSTM(input_units if input_units is not None else n_units, n_units),
    )
    self._train = True
    self.n_units = n_units
    self.reset_state()

  @property
  def train(self):
    return self._train

  @train.setter
  def train(self, value):
    self._train = value

  def reset_state(self):
    self.forward_lstm.reset_state()
    self.backward_lstm.reset_state()

  def __call__(self, feature_vectors):
    sequence = []
    backward_sequence = []

    for feature_vector, backward_feature_vector in zip(feature_vectors, reversed(feature_vectors)):
      sequence.append(self.forward_lstm(feature_vector))
      backward_sequence.append(self.backward_lstm(backward_feature_vector))

    return sequence, backward_sequence
```

Listing A.1: Implementation of a BLSTM layer in Chainer

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst sowie keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe.

Potsdam, den 1. August 2016

Christian Bartz