Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Ing. arch. Alena M o r a v o v á

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Text Recognition in Images Using Recurrent Neural Networks

Guidelines:

1. Get familiar with the current state-of-the-art methods for text recognition in the context of text spotting (see the literature below).
2. Get familiar with the current framework for text spotting developed in CMP.
3. Adapt one of the state-of-the-art approaches for text recognition using recurrent neural networks (RNN) or develop your own.
4. Compare the method with currently the best approaches on standard datasets (e.g. ICDAR).

Bibliography/Sources:

[1] Jaderberg, M.; Simonyan, K.; Vedaldi, A. & Zisserman, A.: Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition, CoRR, 2014
[2] Jaderberg, M.; Vedaldi, A. & Zisserman, A.: Deep features for text spotting, ECCV, 2014
[3] http://www.robots.ox.ac.uk/~vgg/data/text/
[4] Shi, B.; Bai, X. & Yao, C.: An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition, CoRR, 2015
[5] Breuel, T.; Ul-Hasan, A.; Al-Azawi, M. & Shafait, F.: High-Performance OCR for Printed English and Fraktur Using LSTM Networks, ICDAR, 2013

Bachelor Project Supervisor: Mgr. Jan Šochman, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

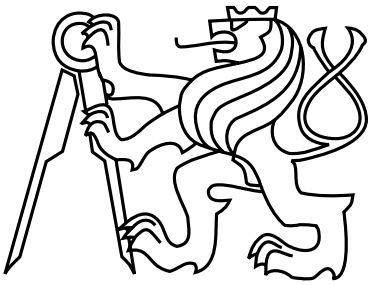
L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 20, 2016

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor Thesis

Text recognition in images using recurrent neural networks

Alena Moravová

Supervisor: Mgr. Jan Šochman, Ph.D.

Study Programme: Open Informatics, Undergraduate

Field of Study: Computer and Informatic Science

May 27, 2016

Aknowledgements

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

V Praze dne 27. 5. 2016 .....

Abstract

The image-based text recognition is a classical problem in computer vision. The problem usually consists of two stages: the text detection and text recognition. During the text detection stage the bounding boxes containing the text are selected in the input image. In the text recognition stage the label corresponding to the text in the bounding box is found. This thesis is concerned with the text recognition stage.

An existing model, the convolutional recurrent neural network [29], was used for the task. This model consists of three main components: the VGG Net [31], which is a convolutional feature extractor, the bidirectional LSTM [10] for sequence prediction, and the CTC algorithm [9] for final transcription of labels. These three parts are analyzed in the theoretical part of the thesis.

The practical part begins with the replication of the results for the original model. Three changes to the model are then proposed and evaluated on two standard datasets, ICDAR 2013 [20] and ICDAR 2015 [1], and compared to the original model. One change slightly improved the accuracy of the model and another one lead to the identification of small defects in the training dataset which might be addressed in the future work.

Abstrakt

Rozpoznávání textu patří mezi klasické problémy počítačového zpracování obrazu. Úloha se skládá ze dvou částí: detekce textu v obrazu a klasifikace detekovaného textu. Během detekce textu jsou nalezeny výseky obrazu obsahující text. Ve fázi klasifikace textu se čte text v daném výseku. Tato práce se zabývá úlohou klasifikace detekovaného textu.

Pro řešení úlohy byl použit existující model: konvoluční rekurentní neuronová síť [29]. Tento model se skládá ze tří hlavních komponent: z hluboké konvoluční sítě VGG Net [31], z obousměrné LSTM [10] a z CTC algoritmu [9] pro přepis do finálního textu. Tyto tři komponenty jsou analyzovány v teoretické části práce.

Praktická část začíná replikací původních výsledků [29]. Následně byly navrženy tři změny modelu, které byly otestovány na dvou datasetech, ICDAR 2013 [20] a ICDAR 2015 [1]. Jejich výsledky jsou porovnány s původním modelem. Jedna z navržených změn mírně vylepšila přesnost studovaného modelu a druhá vedla k objevení menších nedostatků na trénovací sadě, které mohou sloužit jako podklad pro budoucí práci.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Computer vision is a fast developing field of computer science which is concerned with the processing and understanding of high-dimensional data (images) and with the decision making based upon this data. Reading text in natural images is a classical problem in computer vision. As text can be found everywhere, the application of text-reading systems can be really extensive, from reading house numbers to helping navigate sightless people in the streets.

The sought text usually covers only a small fraction of the classified image, therefore it is important to filter out the rest of the image by creating bounding boxes in the image which contain the text. This stage is called text detection. The second part of the image-based text recognition is the actual reading of the sequence within the selected bounding boxes. This stage is called the text recognition. In this thesis we focus on the text recognition part of the problem, because the two parts are quite independent and various datasets exist, which contain images with already cropped text.

The recognition task remains still quite challenging as the text is often distorted in many ways, obfuscated by noise, or placed on different non-uniform backgrounds. Coping with such problems, which are always present in real-world images, needs a more general approach than for example reading text from scanned documents, which is solved by classical optical character recognition (OCR) systems [13]. In recent years the answer to this problem has become the convolutional neural networks (CNN). The CNN model is inspired by the hierarchical structure of the human vision [23]. CNNs are able to filter meaningful features from the input images, which can then be used for recognition of the text by a classification algorithm.

Text recognition can also be seen as the task of predicting a sequence of characters – a word in natural language. The occurrence of a character in a word is influenced by its surrounding context [25]. A novel approach to the text recognition is the usage of recur-

rent neural networks (RNNs), which are good at predicting sequences, as they use internal memory and thus can employ the contextual information.

In this thesis we build on top of one such approach proposed in [29] which combines convolutional neural networks with a recurrent neural network to obtain very good results on several widely used datasets. Those results, the generalizing approach to the task and the use of a recurrent neural network were the motivation for the choice to study the CRNN model.

1.2 Thesis contribution

The goal of this thesis is to understand and analyze the convolutional recurrent neural network (CRNN) [29] and the training dataset (Jaderberg synthtetic data [17]). Changes to the model should be proposed, implemented and evaluated on recent datasets.

The contributions of this thesis can be summarized as follows:

1. Verification of the results published in [29]

   The basic experiment was the replication of the results from the original model on the ICDAR 2013 dataset. The replication of the results was important for the confirmation of the correctness of the model and the setting of the training process.

2. Padding and down-scaling

   During the analysis of the algorithm some shortcomings in the model were discovered. The main problem was the scaling of all the training and testing images, which brought more distortion to the datasets and resulted into poorer performance on very short or long words.

   Based on this finding we propose to remove the scaling of the input data to a fixed size, which distorts the input. In the original model the scaling was introduced to overcome the differences in sizes of the input data. In the proposed experiment the input data was padded instead of scaling to the fixed size.

3. Case sensitivity

   Another experiment was the retraining of the model for case sensitive recognition. This enabled the comparison with the state-of-the-art results on the ICDAR 2015 dataset [1] and also revealed flaws in the training dataset.

4. Imposing blanks

   In this experiment prediction of blanks between characters was imposed. This change of the model was supposed to improve the prediction accuracy of the recurrent layer, but possibly due to the increase of the number of parameters in the model connected with the change, the results were not improved.

5. ICDAR 2015 dataset

   The results of evaluation of the model on ICDAR 2015 dataset are important for comparison with other state-of-the-art models, as the original model had not been tested on this dataset before. The results also indicated a possible direction of further research.

6. Differences between training and testing datasets

   The analysis of the training and testing datasets showed a difference in the distributions of word lengths and N-grams, which explains the observed discrepancy in accuracy of the model on the validation and testing datasets.

1.3 Related work

The problem of text recognition dates back before the history of computer science, however it wasn't until the development of early computers that the development could become faster. Since the early 1970s the task was dominated by the OCR techniques [13]. The traditional OCR systems perform very well on the recognition of text on a uniform background, e.g. in scanned documents, but those methods fail when applied on natural scene images, because they cannot cope with the noise, distortion, inconsistent lighting conditions, and variable fonts that influence the text in natural images.

The work of Epshtein et al. [7] belongs to the classical solutions of the text recognition tasks that perform well on the real world images. They employ the idea that characters are regions of similar stroke width (distance between the parallel edges), which they use for finding single characters that are then grouped into words.

Similar approach is implemented in the work by Neumann and Matas [27]. They are also using the notion of strokes representing the characters, for detection of the characters they use the extremal regions method [26].

In the recent years several solutions based on the feature extraction abilities of deep neural networks and especially the convolutional neural networks were proposed and they have since attracted a lot of attention thanks to their results on many present-day datasets.

One of the earliest works that solves both text detection and recognition in one complete architecture is a work by Bissacco et al. [4]. Their model uses a multistage approach. At first level the bounding boxes with potential text are detected. In the next stage those boxes are segmented into individual characters which are then fed into deep neural network for character classification.

The model proposed in End-to-End Text Recognition [33] with CNN window concentrates only on text recognition part. The model consists of a character detector and recognizer. The detector decides whether there is a character centered in the middle of the selected sliding window, and then the recognizer determines what character it is. The characters are

chosen from 62 predefined classes (26 lower and upper case letters, 10 digits). Both detector and recognizer are implemented as deep CNNs.

One of the earliest models for text recognition is the model proposed by Jaderberg et al. [18]. It is a CNN based architecture with conditional random field (CRF), which is a probabilistic method for structured predictions. The whole image is used as a single input, which is simultaneously fed into a CRF and CNN, the former being good at sequential modelling, and the latter searches for occurrence of N-Grams. The predictions from both algorithms are combined together and assigned as the label in the end.

Another recent model is the one proposed in [24]. The model is based on a CNN, which is extended with recurrent connections in the convolutional layers. According to the paper it has better feature extracting ability than a standard CNN. By combining the RNN model with the recursive CNN model, the authors achieved the accuracy of 90% on the ICDAR 2013 dataset.

To the present day the best accuracy achieving model is the one proposed by Jaderberg et al. [16]. The proposed model achieved the accuracy of 90.8% on the ICDAR 2013 and 98.6% on the ICDAR 2003 dataset. The authors formulate the image-based text recognition as a multi-class categorization problem. They first extract features from the input with a CNN and then use those features to classify into 90k categories (all English words) which makes the model very demanding on memory.

1.4 Structure of the thesis

The thesis is organized as follows. The next chapter introduces the fundamental algorithms underlying the studied model. Chapter 3 explains the architecture of the CRNN model and describes the fundamental parts of the model. Improvements proposed to the studied methods and their evaluation are presented in Chapter 4. In the last chapter the results of the experiments are summarized and ideas for future work are described.

Chapter 2

Fundamental algorithms

In this chapter the basis of the algorithms that were used in Convolutional recurrent neural network (CRNN) [29] are explained. As the name suggests the studied model is based upon convolutional neural networks and recurrent neural networks. Both of those are derived from the artificial neural networks model.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are a class of machine learning models which are inspired by biological neural systems and are used for classification, clustering, function approximation and other tasks. The NN model can be seen as a interconnected system of neurons that exchange information. The connections in the system have numeric weights on them, that influence the strength of the information exchanged between two neurons. The weights are updated based on the experience from the provided training data.

The basic computational unit of neural networks is the neuron (Fig. 2.1). The neuron receives weighted inputs and sums them into its inner activation. The output of the neuron is then calculated by passing its inner activation through a non-linear transfer function called activation function.

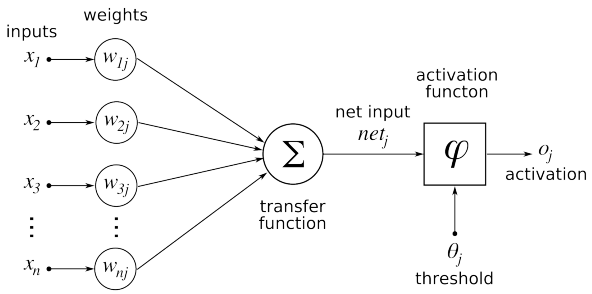


Figure 2.1: The basic computational unit of neural networks, the neuron[1].

[1] Courtesy of https://en.wikibooks.org/wiki/artificial_neural_networks/activation_functions

The inner activation a_j of j-th neuron is calculated by

$$a_j = \sum_{i=1}^{n} w_{ij} x_i + b \tag{2.1}$$

where w_{ij} is the weight between the i-th input and the j-th neuron, the x_i is the i-th input, and b is the bias. The inner activation is then used to calculate the output from the neuron by applying the activation function $y_j = f(a_j)$.

One simple model of neural network, that can well illustrate the principles of neural networks, can be seen in Fig. 2.2. It consists of three layers of neurons, the input, hidden and output layer. Only neurons in neighboring layers are connected and they form a fully connected directed graph. The output of the network is then calculated in the forward pass by following the direction of the edges, from the input layer to the output layer. In the training phase the error of the generated prediction is passed through the layer in the opposite direction.

The basic NN model is the feedforward model where the connections between units cannot form a directed cycle (Fig. 2.2).

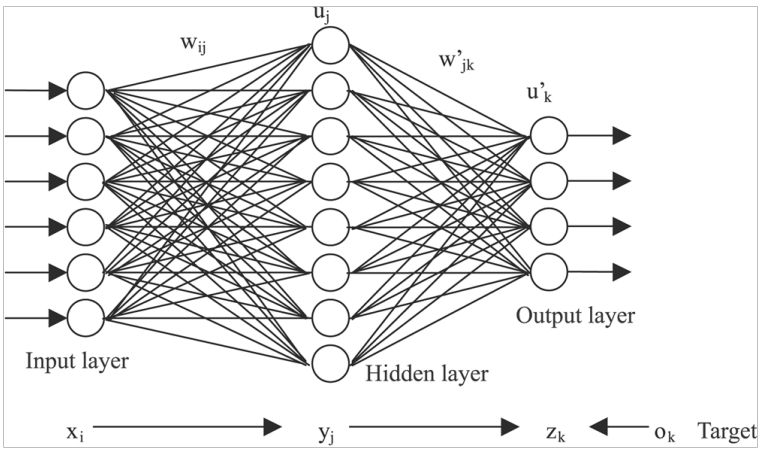


Figure 2.2: Example of a 3-layer neural network[2].

The main advantage of the neural networks lies in the universal approximation property [5], which states that a neural network with at least one hidden layer and finite number of neurons can approximate any function. Other positive characteristic is that it can be trained incrementally from the provided data, which assures that it can find patterns in the data and discriminate them from outliers and noise. Finally the neural network model can be evaluated in parallel, which enables the acceleration of the computation on GPU (Graphics Processing Units).

There are several issues concerning training of neural networks and using them for predictions. One of the main cons is the complexity of the structure of the network created during

[2]Courtesy of http://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they

the training process, which is hard to analyse. We are mostly left with the observation of the outcomes and statistical analysis of the provided results. Other problem is the need for long training time with low learning rate, without which there is a risk of divergence of the optimization process. Other issue comes from the fact, that the learning process is a gradient based method, which searches for local optima and thus the quality of the result depends on the initial conditions. There is a vivid research that is concerned with the overcoming of those poor local optima through regularization, good initialization etc.

The training of neural networks consists of three main phases: the network evaluation phase, the loss calculation and the backpropagation of the error towards the weights. During the evaluation the neural network's output is calculated. The received output is then compared with the required target and the error of the output is calculated by the loss function. The error is then backpropagated through the network to update weights, so as the error of the network would be minimized. A trained model can then be used in the testing phase, during which the forward pass calculates the output – prediction – of the model.

2.1.1 Network evaluation

During evaluation the network calculates the output layer by layer. All neurons in the following layer receive the outputs from all neurons in the previous layer. The output vector of the j-th neuron is calculated by

$$y_j = f(W_{ij}x_i + b_j) \tag{2.2}$$

where f is the activation function, W_{ij} is the weight matrix between the i-th and the j-th layer, x_i is the output vector from the i-th layer, and b_j is the bias vector for the j-th layer. The output vector y_j then used as input vector for the next layer. The output of the NN model is acquired by continuing this process through all layers and the model's prediction can be read from the last – output – layer.

2.1.2 Activation functions

Activation function is bringing non-linearity into the model, without which the network would be just a regular linear classifier. There are three frequently used activation functions: sigmoid, hyperbolic tangent (tanh) and rectified linear unit (ReLU). Sigmoid has the form

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.3}$$

It's outputs range from 0 to 1, which is convenient for estimation of probabilities. If we need also the negative outputs, the tanh activation function can be used

$$f(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.4}$$

The problem with both tanh and sigmoid functions is in their shape, which influences directly the update of the weights. The gradient of the sigmoid and tanh function vanishes if we increase or decrease x far from zero. This does not happen with the rectified linear unit (ReLU)

$$f(x) = \max(0, x), \tag{2.5}$$

a non-saturating activation function, that was introduced in [21].

Thanks to the shape of the function the value of the gradient does not decrease as the x increases, on the contrary it stays 1, and for negative x it is always 0. The use of the hard max function induces the sparsity of the network. The deep networks with ReLU activation functions can be trained efficiently even without pretraining [21].

2.1.3 Loss function

Loss function compares the prediction from the neural network with the desired output and returns the error of the prediction. The objective of the neural network training is the minimization of the loss on the training data. The basic loss function is the L_2 (Euclidean) norm

$$L(y, y') = ||y - y'||_2^2 = (y - y')^2 \tag{2.6}$$

or the L_1 norm (absolute loss)

$$L(y, y') = ||y - y'||_1 = |y - y'| \tag{2.7}$$

The error given by the loss function is then used for updating the weights in the network in the backward pass of the training.

2.1.4 Backpropagation

Backpropagation as learning algorithm for neural networks was first introduced by P. Werbos in 1982 [34], and to this day it is the most notable algorithm used for training neural networks via minimization of the loss function. The loss is optimized using the gradient method. The gradient is computed in the output layer and backpropagated through the network in order to calculate the change of the weights ΔW_{ij}. The ΔW_{ij} should lead towards the minimization of the loss E, which is achieved by calculating its partial derivative with respect to the weights W_{ij}:

$$\Delta W_{ij} = \frac{\partial E}{\partial W_{ij}}. \tag{2.8}$$

The weight deltas are then used to update the original weights:

$$W_{ij}(t+1) = W_{ij}(t) + \mu \Delta W_{ij}(t). \tag{2.9}$$

The weight deltas ΔW_{ij} (updates) are calculated by means of the chain rule

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \mathrm{a_j}} \frac{\partial \mathrm{a_j}}{\partial W_{ij}}. \tag{2.10}$$

The y_j is the output vector from j-th layer and the $\mathrm{a_j}$ are the inner activation of the neurons in the layer. The weight matrix W_{ij} belongs to a connection from layer i towards layer j. The E stands for the error of the prediction which in the case of L_2 norm is $E = (t - y_o)^2$, where t is the target and y_o is the output of the whole network.

To backpropagate the error the error vector δ_j of the neurons in j-th layer is needed. Those errors are again calculated layer by layer and describe how the error changes with the activation of the specific neuron in layer j

$$\delta_k = -\frac{\partial E}{\partial \mathrm{a_k}}, \tag{2.11}$$

$$\frac{\partial E}{\partial \mathrm{y}} = 2(t - y). \tag{2.12}$$

In the case of sigmoid activation function the derivative of the unit's output output w.r.t. the activation is

$$\frac{\partial y}{\partial \mathrm{a_k}} = \frac{e^{a_k}}{(1 + e^{a_k})^2}, \tag{2.13}$$

and the activation of the neuron is

$$\mathrm{a_k} = W_{jk} y_j, \tag{2.14}$$

and it's partial derivative w.r.t. weights is

$$\frac{\partial \mathrm{a_k}}{\partial W_{jk}} = y_j. \tag{2.15}$$

The gradients for hidden-to-output weights are then

$$\frac{\partial E}{\partial W_{jk}} = \delta_k y_j. \tag{2.16}$$

For the input-to-hidden weights the gradients are calculated equivalently to the hidden-to-output weight gradients. First the contribution E_j of the neurons in hidden layer j to the total error are calculated

$$E_j = W_{ij} \delta_k \tag{2.17}$$

and then those contributions are used to obtain the error vector δ_j equally to Eq. 2.13,

$$\delta_j = \frac{\partial E_j}{\partial \mathrm{a_j}}, \tag{2.18}$$

which is then used to calculate the error gradients w.r.t. the input-to-hidden weights by applying the same equation as Eq. 2.16 to the input weight matrix W_{ij}.

$$\frac{\partial E}{\partial W_{ij}} = \delta_j x_i, \tag{2.19}$$

where x_i is the input vector.

2.2 Deep learning

In the past years artificial neural networks became the leading algorithms for many computer vision tasks such as image classification, segmentation and other. This recent growth of interest in neural networks was triggered by the successes of deep learning, which enabled leveraging the vast amounts of available training data [6] and increasing computational power. Unlike the typical shallow neural networks (up to 3 layers) the deep architecture can efficiently train larger number of parameters and therefore approximate high complexity functions [3], which are necessary for solving difficult problems such as computer vision or natural language processing. The deep layered model can learn better representations of the input data, and subsequently be trained more efficiently.

The fundamental concept of deep learning algorithms is the idea that the input data have underlying representation, which is composed of various levels of abstraction from simple to more abstract concepts [2]. This notion is used by the deep learning algorithms to extract the important features from the data through cascading many nonlinear processing units. The layers are forming a hierarchy, where the higher level features are build upon the lower level features. By varying the number of layers, different level of abstraction can be achieved.

Convolutional neural networks

The convolutional neural networks (CNNs) are often used to enable training of deep learning algorithms. The CNN model is based on the traditional neural network model, however unlike the standard model of neural network, which is hard to train as deep architecture due to high amount of parameters. However, CNN models can be built very deep and still be efficiently trained through backpropagation. The efficiency of the CNN model comes from accounting for the local correlation present in images, which enables it to have much less parameters to train than a neural network model would have and still be able to find complex patterns in the presented data.

The architecture of CNN was inspired by the study of Hubel and Wiesel [14] from early 1960's. They described the overlapping receptive fields in animals' eyes and the existence of simple and complex cells within the visual cortex. Upon those observations they described the existence of a hierarchical arrangement, that would make the cells in the visual cortex process increasingly more complex patterns from spots and edges up to entire objects. The first commonly known work that followed upon the ideas of Hubel and Wiesel was LeCun's work from 1998 [23]. LeCun showed the exceptional characteristics of CNNs and the use of spatial correlation in pictures. The work showed the superior performance of machine learning based approach to feature extraction in comparison to hand-crafted selection of features. The model combined convolutional feature extractor with classifier atop, and it was trained jointly with backpropagation.

The basic ideas exploited by the CNN architecture are local connectivity and weight sharing. Local connectivity exploits the spatially local correlation of pixels in natural images

by always connecting neurons only to a small section of input from the preceding layer. By stacking several of those layers, the neurons in the higher layers become more global by reacting on larger section of the input image. They respond to more complex patterns than neurons in the lower layers as can be seen in Fig. 2.3. Second main contribution of the CNN model is the sharing of weights. It reduces the number of parameters by applying each filter which is also called kernel across the whole input. By applying one kernel on the input we receive a feature map, which together with other feature maps from other kernels forms the input to the next layer.

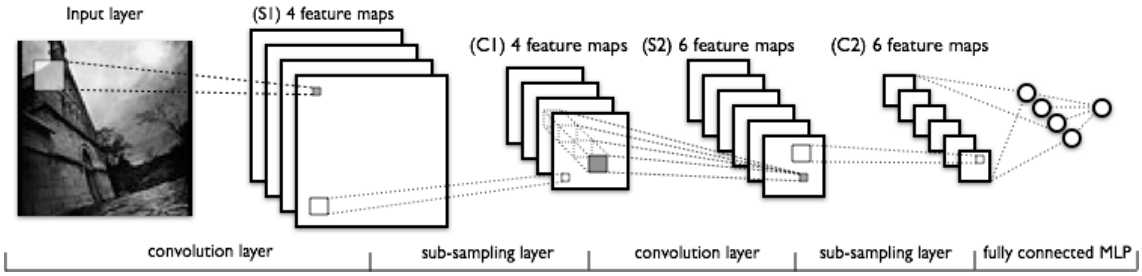


Figure 2.3: Depiction of CNN (LeNet) model. The figure shows the stacking of convolutional layers on top of each other which enlarges the receptive fields of the neurons.[4].

Since the year 2012, the CNN-based models are the most successful models at many vision task. The breakthrough was brought by the work of Krizhevsky et al. [21] who decreased the error rate at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [6] by about 10%. This major improvement in accuracy of the classification was achieved mainly thanks to two changes: the introduction of the non-saturating activation function ReLU and the usage of dropout (regularization method) to prevent overfitting, which enabled them to train a model consisting of 650.000 units with 60 million parameters.

A typical convolutional neural network consists of convolutional layers that create feature maps, alternated with the max-pooling layer for resizing of the processed data and sometimes also regularization techniques such as dropout, batch regularization etc..

Convolutional layer

Convolution is mathematical operation of two functions, it is marked as f * g. The basic equation for convolution of functions f and g is

$$(f * g)(t) \quad \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} f(t-\tau)\, g(\tau)\, d\tau. \tag{2.20}$$

In the case of convolutional neural networks the applied convolution is discrete

$$(f * g)[n] \quad \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m]\, g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]\, g[m]. \tag{2.21}$$

[4]Courtesy of http://deeplearning.net/tutorial/lenet.html

Specifically for convolutional layers the outputs of all units y_{ij}, where i and j are 2D coordinates of the input, in a layer l and kernel k are calculated by first calculating the inner activation of neuron x_{ij}^l:

$$x_{ij}^l = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} w_{ab}^k \; y_{(i+a)(j+b)}^{l-1} \tag{2.22}$$

where the filter is of size $m \times m$, y^{l-1} is the output from preceding layer, w_{ab} are weights in the applied filter. The non-linearity f is then applied to the activation from convolution.

$$y_{ij}^l = f(x_{ij}^l) \tag{2.23}$$

The weights applied in convolution are grouped in the kernels (Fig. 2.4). The same kernel is applied across the whole input, this process is also called weight sharing. The output of the application of one or more kernels is a convolutional layer. During training the weights of the convolution are updated by the selected optimization method, which results into the specialization of the kernel on a specific pattern which can occur anywhere in the picture, it is translation-invariant. The usual size of kernels ranges from 3x3 up to 7x7 input units.

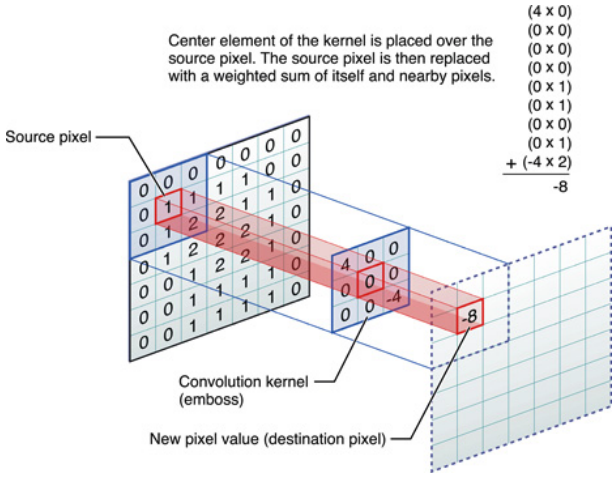


Figure 2.4: Kernel convolution[5].

The size of the output from the convolutional layer is determined by the number of kernels, the size of kernels, stride, and padding. Stride defines the size of the overlap of the receptive fields of neurons and thus the shift of the kernel, and padding is usually used to prevent loosing the information from the boundaries of the inputs and to prevent premature downsizing of the feature maps. The value of padding is typically 0.

Important notion in the convolutional neural networks is the receptive field of a neuron. It corresponds to a local region in the input data that influence the specific neuron. Between two successive layers the receptive field of a neuron is equivalent to the region in the previous layer in size of the applied kernel. By stacking more CNN layers on top of each other the

[5]Courtesy of http://deeplearning.net/tutorial/lenet.html

neurons in higher layers are sensitive to increasingly larger receptive field in the original input image.

Max-pooling layer

Another part of the convolutional neural network architecture is the max-pooling layer. It is a non-trainable layer that carries out a simple operation that results in the reduction of size of the model, it is usually applied in-between two successive convolutional layers. The idea of max-pooling comes from the fact, that the information in images is usually not restricted to a single pixel, but rather spread in groups of pixels, that all carry similar information. Therefore the size of the data can be reduced by leaving out some of those pixels without the loss of the important information. The max-pooling operation takes the maximal value from a window of size $k \times k$. The operation is applied across the whole input with a given stride.

Regularization methods

One of the problems of the basic deep neural networks was the overfitting of the models to the training data. The way to solve this problem is by using some regularization technique such as dropout, unit prunning, weight decay, or batch normalization. The dropout [32] is the most popular regularization method. When using dropout some randomly selected units are temporarily removed from the model in each training iteration. This way the model avoids learning rare dependencies in the training data that could lead to overfitting.

Another problem related to the training of deep networks is that the distribution of inputs in each layer is shifting based on the change of the previous layer. This problem can be overcome by batch normalization [15]. With this technique layer inputs are normalized over each training batch which is a subset of the training dataset which losses are summed together and backpropagated in one backward pass. The normalized value is then transformed by weights, which are trained along with other parameters of the model.

2.3 Recurrent neural networks

Recurrent neural networks (RNN) are a class of artificial neural networks that unlike feedforward neural networks have connections that form a directed cycle. The cyclic connections are called recurrent and they enable the RNN to hold the information from the past inputs for more than one time step. This enables the network to exhibit time-dependent behavior. This property makes RNNs useful for tasks involving prediction of sequences such as text recognition (printed and also handwritten), speech recognition, or natural language processing.

Standard recurrent neural network returns one output y_t at each time step

$$y_t = f(W_{hy}h_t + b_y) \tag{2.24}$$

where W_{hy} is the hidden-to-output weight matrix and h_t is the output of the hidden layer at time t:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{2.25}$$

where the W_{xh} and W_{hh} are the weight matrices, W_{xh} is the input-to-hidden layer weight matrix, W_{hh} is the hidden-to-hidden layer weight matrix, the b term denotes the bias vector, and f is the activation function.

The problem with the basic RNN architecture is that the model can't learn longer sequences because of the vanishing gradient problem [11]. The problem arises with the fact, that the errors computed in backpropagation are multiplied by each other every step of the propagation of error through time. If the gradient is small, the error diminishes towards zero, or on the contrary if it's very large the error goes beyond limits.

Long short-term memory network (LSTM)

The issue of the vanishing gradient was approached by Hochreiter and Schmidhuber [12] in an alternative computational unit called long short-term memory (LSTM) (Fig. 2.5. It's main distinction is the cell state within the LSTM unit which can remember an information for arbitrary long time. This way the LSTM can hold information theoretically throughout the whole presented history. LSTM units, however, don't suffer from the vanishing gradient problem because the activation function in the recurrency (in the cell state) is identity with the derivative of 1.0, which ensures that the gradient neither vanishes nor explodes. It remains constant and can be passed through the recurrent connection as many times as needed.

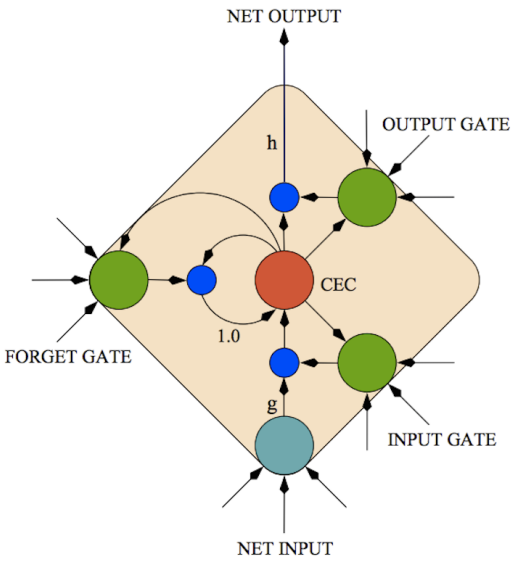


Figure 2.5: LSTM unit[6].

[6]Courtesy of http://haohanw.blogspot.cz/2015/01/lstm.html

The process of calculating the output from one LSTM unit is slightly more complex than in the case of the typical RNN neuron, however, it relies on the same principle. All LSTM units in hidden layer calculate their outputs h at time t. Those outputs are then used together with the new input data as the input to the units in time $t+1$. The equations for calculating h_t are as follows [10]:

$$\mathbf{i_t} = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.26}$$

$$\mathbf{f_t} = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.27}$$

$$\mathbf{c_t} = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.28}$$

$$\mathbf{o_t} = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.29}$$

$$\mathbf{h_t} = o_t \tanh(c_t) \tag{2.30}$$

where i, f, o are respectively the input, forget and output gate vectors. The gates are either effectively blocking the information flow, if their output is close to 0, or letting the information through, if their output is close to 1. More precisely the forget gate decides if the cell state will forget the information it is retaining, the input gate decides if new information will be let in and the output gate decides if the information will be sent out of the LSTM unit. The c is the cell activation vector and h the hidden vector (the vector used for sending recurrent information). The W_{xY} is the weight matrix between input units and any of the gates, the W_{hY} is the weight matrix of recurrent connections from activations of hidden units to any gate and the W_{cY} is the weight matrix of recurrent connections from cell activations to any gate. The activation functions used in the model are sigmoid σ and hyperbolic tangent tanh respectively.

Backpropagation through time (BPTT)

The most often used learning algorithm for recurrent networks is the backpropagation through time. It is a gradient-based technique for training recurrent neural networks. The basic idea works similarly to the backpropagation algorithm, however the error has to be propagated through the whole sequence back in history as can be seen in Eq. 2.6.

The BPTT algorithm begins with unfolding the recurrent neural network through time. Then the error is backpropagated not only through the feedforward connections, but also through the recurrent connections. The weights in the whole unfolded model are shared across the time, to achieve this, the gradients for input and recurrent weights are summed up at each time step and averaged into the final update for the weights.

The input, hidden and output weight matrix are in this section described as U, W, and V. Denoting y_t as the output of the network and E_t as the error of this output at time step t the basic equation for the weight update in BPTT is calculated as following.

The goal is to calculate the gradients with respect to the U, V, W weight matrices. For the output weight matrix V the gradients are calculated similarly to standard backpropagation. To calculate the weight deltas for the hidden matrix ΔW (Fig. 2.6) the chain rule is applied

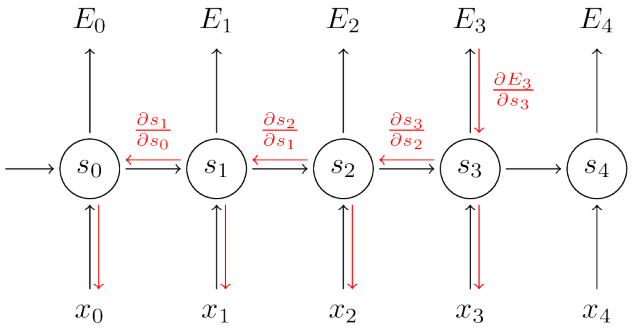


Figure 2.6: BPTT, error is backpropagated along the red arrows[7].

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial s_t}\frac{\partial s_t}{\partial W}. \tag{2.31}$$

The output vector s_t of hidden layer at time t depends on the outputs at previous time steps which also subsequently depend on the parameters of the hidden weight matrix W. This recursive behavior leads to a different chain rule

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial s_t}\frac{\partial s_t}{\partial s_k}\frac{\partial s_k}{\partial W} \tag{2.32}$$

where we sum up the contributions of activations at each time step to the gradient. As the matrix W is used in every time step up to the output at time step t, the gradients have to be backpropagated up to $t = 0$.

The hidden weight deltas ΔW are then equal to

$$\Delta W = \frac{\partial E_t}{\partial W}. \tag{2.33}$$

The weight deltas for the input matrix U are calculated in the same way as for the hidden matrix W.

[7] Courtesy of http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

Chapter 3

Convolutional recurrent neural network

For the proposal and implementation of extensions to the convolutional recurrent neural network (CRNN) [29], it was important to understand the architecture which will be described in this chapter.

The model consists of three main components: CNN model based on the very deep convolutional network for large-scale image recognition [31], also called the VGG Net, the bidirectional long short-term memory network (BLSTM) [10] and the connectionist temporal classifier (CTC) [9].

The VGG Net is used for extraction of features from the input images. The output of the VGG Net component is a sequence of feature vectors, where each of the feature vectors corresponds to a rectangular receptive field within the image. This sequence is then used in the BLSTM component to make predictions of characters based on the sequence. The predictions are then fed into the CTC layer that transcribes them into the final label.

3.1 VGG Net

Recurrent neural networks are suitable for predicting sequences, however the preprocessing of the input data is essential for their good performance. At present the best feature extraction models are the convolutional neural networks.

VGG Net is a deep convolutional neural network model which was introduced in 2015 by Simonyan and Zisserman [31]. They accomplished the state of the art results in image classification at that year. The main contribution of the model is the confirmation of the positive influence of the depth on the performance of the convolutional networks. The model consists of 16-19 layers. The same 3x3 convolution filters (kernels) are applied throughout the whole network. The VGG net is still widely used mainly due to its simplicity and easy implementation, but still a very good performance.

For the use in the text recognition the feature extraction model can have fewer layers, because much less classes have to be recognized in comparison to the object recognition.

In the case of the studied solution seven convolutional layers with five max-pooling layers were used to extract features from the input image. Inspired by the VGG Net the CRRN model uses simple 3x3 kernels with stride 1 in all of its convolutional layers. As opposed to the VGG Net the CRNN model doesn't include fully connected layers and a small change is made in the 3-rd and 4-th maxpooling layers, where different shape of pooling window to the original model is used. In the model a rectangular pooling window is adopted which is beneficial for recognition of some characters and also enables to extract longer feature sequence from the input. This change also leads to rectangular receptive fields at the end of the component.

The input to the VGG Net component is a 100×32 pixel image. It extracts 26 feature vectors from the image where each feature vector corresponds to a rectangular receptive field in the input image. Those 26 feature vectors are then combined into an input sequence and sent to the bidirectional long short-term memory network.

3.2 Bidirectional long short-term memory network

The bidirectional RNNs were introduced by Schuster in his work on speech recognition [28]. In the work of Graves et al. [10] the bidirectionality was applied to LSTMs. In comparison to standard LSTMs where the output y_t at time-step t depends only on the outputs at time-step $0, \ldots, t-1$ the bidirectional models take into account also all subsequent outputs of the sequence $t+1, \ldots, T-1$, where T is the length of the sequence. This is very helpful for all natural language related tasks such as speech or text recognition, because in natural language sequences characters depend not only on the characters that appeared prior to, but also after the given time-step.

Bidirectional LSTM (BLSTM) is built by adding one more set of hidden layers to the recurrent network, which goes backward in time. The two hidden layers (forward and backward) are completely separate. As can be seen in Fig. 3.1 they are only interacting in the output layer, where the activations from both of the hidden layers are used to compute the output from the network at every time step.

The output of BLSTM at time t, y_t, is a weighted sum of outputs of forward hidden layer $\vec{h}$ and backward hidden layer $\overleftarrow{h}$:

$$y_t = W_{\vec{h}y}\vec{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y \tag{3.1}$$

where $W_{\vec{h}y}$ and $W_{\overleftarrow{h}y}$ are weights for forward and backward hidden-to-output layers, respectively. Output of the forward $\vec{h}_t$ and backward $\overleftarrow{h}_t$ hidden layers are computed as:

$$\vec{h}_t = f(W_{x\vec{h}}x_t + W_{\vec{h}\vec{h}}\vec{h}_{t-1} + b_{\vec{h}}) \tag{3.2}$$

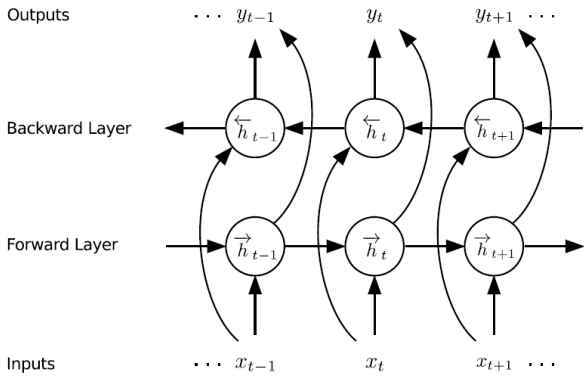


Figure 3.1: The bidirectional model computes the forward hidden sequence $\vec{h}$, the backward hidden sequence $\overleftarrow{h}$ and the output sequence y [2].

and

$$\overleftarrow{h}_t = f(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}) \tag{3.3}$$

where f is the activation function, $W_{x\vec{h}}$ and $W_{x\overleftarrow{h}}$ are weights for forward and backward input-to-hidden layers, respectively. x_t is the input to BLSTM at time t, $W_{\vec{h}\vec{h}}$ and $W_{\overleftarrow{h}\overleftarrow{h}}$ are weights for forward and backward hidden-to-hidden layers, respectively, $b_{\vec{h}}$ and $b_{\overleftarrow{h}}$ are the respective biases.

Another important aspect of the algorithm is the deep recurrent architecture, which is able to build increasingly higher leve l of representation of the provided data [12]. The deep LSTM model is created by building more hidden layers atop of each other. In the case of bidirectional model, each hidden layer receives input from the previous time step of the same layer and from the previous hidden layers of both directions in the current time step.

The deep BLSTM layer as implemented in [29] processes a sequence of 26 feature vectors of dimension 512 from the last layer of the convolutional network. Each of the vectors corresponds to a rectangular receptive field in the original image. The final BLSTM layer returns a soft-max prediction of size 26×37 (sequence length $\times$ number of possible symbols: 26 characters, 10 digits, and blank). The soft-max prediction is then decoded into the final label by the connectionist temporal classification algorithm.

3.3 Connectionist temporal classification layer

The bidirectional model of LSTM predicts sequences of a fixed length, but in the real world words vary in their length. This discrepancy is solved by the connectionist temporal classification (CTC) layer [9].

The key idea of CTC is that the label is not generated directly by the recurrent neural network, but instead a probability distribution over all possible characters is generated at

[2] Courtesy of http://qiita.com/t_signull/items/21b82be280b46f467d1b

every time step. In the training phase the conditional probability $p(l|x)$ of the target label l given the input x is calculated:

$$p(l|x) = \sum_{B^{-1}(l)} p(\pi|x) \tag{3.4}$$

Where B is a many-to-one mapping $B : L^T \rightarrow L^{\leq T}$, from the set of paths onto the set $L^{\leq T}$ of possible labellings, where L is the alphabet. The probability of path $\pi \in L'^T$ given the input x is calculated by:

$$p(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{3.5}$$

Where π_t is a symbol from alphabet L in the path π at time step t.

The conditional probability $p(l|x)$ is used for the calculation of log-likelihood in the objective function. The differentiation of the received error with respect to the output of the RNN layer is directly infeasible, therefore the forward-backward algorithm is used to backpropagate the errors to the outputs from the RNN layer. In the testing phase the raw RNN predictions are decoded into labels by means of maximum likelihood estimate.

3.3.1 Objective function

The training algorithm uses an objective function O based on maximum likelihood estimate to backpropagate the error towards the output of the BLSTM layer. The objective function O is defined as the negative log-probability of correctly labelling the training set S:

$$O = -\ln(\prod_{(x,z)\in S)} p(z|x)) = -\sum_{(x,z)\in S} \ln p(z|x) \tag{3.6}$$

To backpropagate the error through the whole network we need the partial derivatives of the outputs y_k^t from the recurrent network with respect to some training example (x, z),

$$\frac{\partial O}{\partial y_k^t} = -\frac{\partial \ln p(z|x)}{\partial y_k^t} = -\frac{1}{p(z|x)} \frac{\partial p(z|x)}{\partial y_k^t} \tag{3.7}$$

Directly differentiating according to Eq. 3.7 is infeasible, because there is no direct mapping from the RNN outputs to the label. For calculating of Eq. 3.7 the forward-backward algorithm is used.

3.3.2 CTC forward-backward algorithm

In the training phase we need to find the derivatives of the objective function with respect to the outputs from the BLSTM layer. It is infeasible to calculate them directly, therefore we use the CTC forward-backward algorithm [8], which is a dynamic programming algorithm. For every time step t of sequence z in the forward pass the probabilities of label prefixes are computed, in the backward pass the probabilities of suffixes are computed. The

two probabilities are then combined together to calculate the derivatives with respect to the RNN outputs, and then the standard backpropagation through time on the BLSTM network can be applied.

We first calculate the forward and backward probabilities for every prediction from BLSTM layer at every time step t in the forward and backward pass.

Forward pass

$$\alpha_t(s) = \sum_{\text{label}(\pi_{1:t})=\ell'_{1:s}} \prod_{j=1}^{t} y_j(\pi_j) \tag{3.8}$$

The variable $\alpha_t(s)$ is the summed probability of all paths π with prefix of length t, that carries the symbol s at time step t. The value of $\alpha_t(s)$ can be calculated recursively with the initialisation

$$\alpha_0(0) = 1, \tag{3.9}$$

$$\alpha_0(i) = 0 \quad (\text{for i} > 0). \tag{3.10}$$

Which means that the probability of blank prefix at $t = 0$ is 1 and the probability of any other prefix at $t = 0$ is 0. Then a simple rule can be applied to gradually calculate prefixes until the end of the sequence.

$$\alpha_t(s) = \begin{cases} y_t(\ell'(s)) \cdot (\alpha_{t-1}(s) + \alpha_{t-1}(s-1)), & \text{if } \ell'(s) = b \text{ or } \ell'(s-2) = \ell'(s) \\ y_t(\ell'(s)) \cdot (\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2)), & \text{otherwise} \end{cases}$$

This recursive rule is taking account of two types of transitions: transition between blank b and non-blank labels, and between any pair of distinct characters. The occurrence of blank labels is simplifying the work with the sequences, where the RNN is not forced to keep predicting some label until a new label is present and it also allows for two similar characters occurring next to each other.

Backward pass

In the backward pass the variable $\beta_t(s)$ is calculated. The equations for them are identical to the variable $\alpha_t(s)$ with the difference that the probabilities of suffixes, instead of prefixes, are recursively calculated with respect to the time step t and symbol s.

Now the two variables $\alpha_t(s)$ and $\beta_t(s)$ from the forward-backward algorithm can be used to calculate the Eq. 3.7.

We first need to calculate the probability of a sequence with a character s at time step t. It is a product of the probability $\alpha_t(s)$ that the prefix ends with s and of the probability

$\beta_t(s)$ that some suffix starts with s at time t. For any t the probability of label z given x is then

$$p(z|x) = \sum_{s=1}^{|z'|} \alpha_t(s)\beta_t(s) \tag{3.11}$$

where z' is the set of labellings that contain symbol s at time t.

To differentiate the Eq. 3.11 only the paths that go through label k at time t can be considered, since it is made sure that the outputs of the network do not influence each other. We then differentiate the Eq. 3.11 with respect to y_k^t to receive

$$\frac{\partial \alpha_t(s)\beta_t(s)}{\partial y_k^t} = \frac{\alpha_t(s)\beta_t(s)}{y_k^t} \tag{3.12}$$

if k occurs in z', otherwise $\frac{\partial \alpha_t(s)\beta_t(s)}{\partial y_k^t} = 0$. The same label can occur several times in one labelling, the set of positions where label k occurs in z is defined as $lab(z,k) = s : z'_s = k$. And if we substitute into (3.7) we get

$$\frac{\partial O}{\partial y_k^t} = -\frac{1}{p(z|x)y_k^t} \sum_{lab(z,k)} \alpha_t(s)\beta_t(s). \tag{3.13}$$

3.3.3 Transcription to labels

In the testing phase the transcription layer is used to convert the per-frame predictions from RNN into the label. To find the correct labelling the most probable label conditioned on the predictions is chosen. The CTC algorithm offers two main ways of decoding the predictions: the lexicon-free and lexicon-based transcription.

The lexicon-free transcription uses either the max path or the prefix search algorithm [9]. The max path algorithm finds the maximal probable sequence $l^* = B(argmax_\pi(\pi|y))$ by selecting the most probable character from the RNN predictions at each time step and then by removing the double occurrences of the same label and the dashes, which denote the blanks.

For lexicon-based decoding the label is the most probable word from a given lexicon. By using the forward pass of the CTC forward-backward algorithm the log-probability of the BLSTM prediction given every word from the lexicon is calculated and then the word from the lexicon with the highest obtained log-probability is selected.

Chapter 4

Practical part

The objective of this work was to study and possibly improve the CRNN algorithm [29]. The practical part of the thesis consists of two main parts:

1. Analysis of the data
2. Proposal and evaluation of improvements

4.1 Data analysis

Analysis of the datasets is an important step for the improvement of the model. First the training dataset is described, and then two test datasets follow: ICDAR 2013 and ICDAR 2015. The data analysis helped with the understanding of the structural limitations of the model. In the following section all analyzed widths of the images correspond to the sizes after proportional scaling to the height of 32 pixels which equals to the height of the input images to the CRNN model.

4.1.1 Training set

The network was trained solely on the synthetic dataset by Jaderberg et al. [17]. The dataset contains 9 millions training images, that were generated by a text engine. Although they were synthetically generated, the images look very realistic and the trained model gives good results when tested on real-world data.

The Jaderberg dataset consists of all 90000 English words (expanded Hunspell dictionary), which are rendered in several font types, put onto various backgrounds and rotated, translated and distorted to create very realistic scenarios for the images (Fig. 4.1). The labelling of the training images is generally case sensitive. The created dataset for the training was divided into a training and validation datasets with sizes of 8 millions and 1 million of images respectively.

Training dataset's statistics		
Width of images	# of images	%
<50	158771	17.8
<100	3238972	31.1
>100	5680269	63.6
>200	442806	5
>300	26297	0.3
>400	2543	0.03
>600	464	0.005
>800	377	0.004
>1600	155	0.001
all	8919241	100

Table 4.1: Width statistics of the training data from Jaderberg's dataset. All images were scaled to a fixed height (32 pixels)

The data statistics in Tab. 4.1 show that more than half of the images are wider than 100 pixels and about 5% are wider than 200 pixels. Some of the images have a very unrealistic width, of 600 pixels and more. As can be seen in Fig. 4.1 the extreme width of the images is caused by defects in the dataset.

In the original CRNN model all images were disproportionally scaled to the size of 32x100 pixels. The data analysis has shown that this could lead to severe distortion of the input data.

4.1.2 Testing set

The testing was done on two standard datasets for text recognition in real scenes, the "Focused Scene Text" challenge from ICDAR 2013 [20] and the "Incidental Scene Text" from ICDAR 2015 [1].

Test datasets statistics				
	ICDAR 2013		ICDAR 2015	
Width of images	# of images	[%]	# of images	[%]
<50	207	19	638	31
<100	634	58	1755	84
>100	461	42	322	16
>150	170	16	57	3
>200	62	6	10	0
>300	12	1	0	0
all	1095	100	2077	100

Table 4.2: Width statistics of the test data.

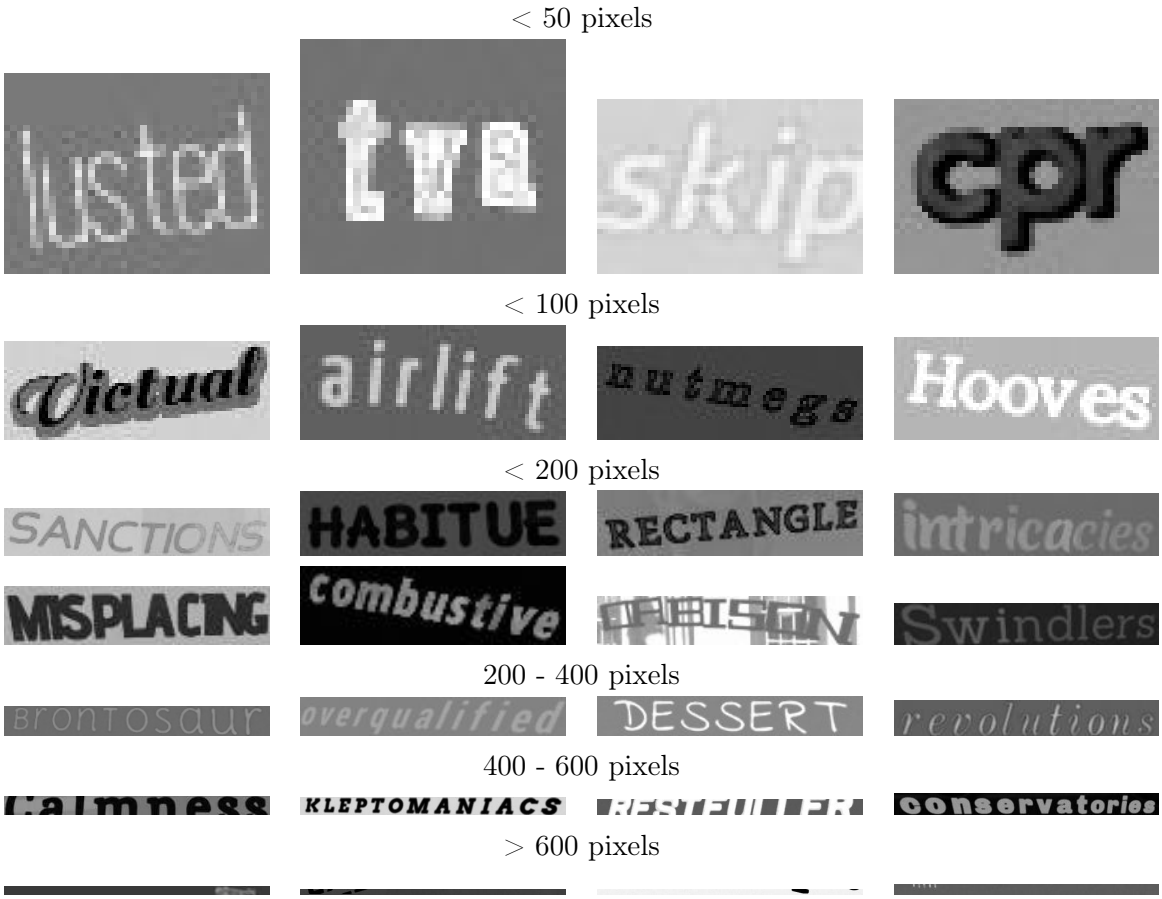


Figure 4.1: Examples of images from training set.

ICDAR 2013

The dataset is composed of images that are explicitly focused on the text content. It is corresponding to the scenario where the users directly focus their camera on the text. The ICDAR 2013 challenge features images that are reasonably well focused and mostly horizontally oriented. Examples of images from the ICDAR 2013 dataset can be seen in Fig. 4.2. The width statistics of the dataset can be seen in Tab. 4.2. The data shows that about 20% of the images are shorter than 50 pixels and that about 6% of them are longer than 200 pixels.

Similarly to other related works the tests in this thesis were run on a subset of the original ICDAR 2013 dataset by leaving out words shorter than 3 characters, which resulted in a test dataset of 897 images from the original 1095 images.

ICDAR 2015

The ICDAR 2015 dataset features a more complicated scenario. Unlike the ICDAR 2013 the images in ICDAR 2015 the images were taken without any specific action which would rectify the text positioning in the frame, or improve its appearance. Therefore the captured

Figure 4.2: Images from ICDAR 2013 Examples with different width.

text is more difficult to recognize correctly. Many text images are rotated by 90° (in both directions), which results in loose bounding boxes. Some images are extremely distorted. Examples of images from this dataset are shown in Fig. 4.3.

As can be seen in Tab. 4.2 more than 52% of the images in the dataset are between 50 and 100 pixels wide, about 30% are shorter than 50 pixels and only about 3% are wider than 150 pixels. These results indicate that the down-scaling is not a problem for ICDAR 2015. The up-scaling could create some problems to the model, but the distortion that is already present in the images poses a much bigger challenge.

4.2 Experiments

Several changes to the basic algorithm have been proposed in order to improve the accuracy of the algorithm.

Due to high memory demand and long computational time all training took place at the czech academic cloud center for computing Metacentrum Vo. Each model was trained for 300k iteration steps (3 days), the number of iterations was needed for the model to reach convergence. One iteration step is equivalent to the evaluation of one training batch.

The value for all presented results corresponds to the accuracy of the model that had the

Figure 4.3: Examples of images from ICDAR 2015 dataset.

best performance on the validation test in the first 300k iterations. Accuracy of a model is the percentage of correctly recognized words in the tested set.

4.2.1 Replication of the results on ICDAR 2013

The replication of the results from the CRNN paper [29] was important for verification, that the dataset was created correctly and that the training process was set up correctly.

In the Tab. 4.3 it can be seen that the result from [29] were obtained by the best achieving model from first 300k iterations. In the github repository the authors posted a model that achieved the accuracy of 88.7%, however, it is unclear, how was this result achieved.

Model	Accuracy [%]
Original model	86.7
Retrained model	87.05
Model from github [1]	88.70

Table 4.3: Comparison of results of the original model and the retrained model on ICDAR 2013. [29]

4.2.2 Padding and down-scaling

In the [29] every image is resized to a fixed size of 32×100 pixels. This leads to a distortion of the input image which consequently leads towards more complex situation for the feature extractor (convolutional neural network) and following also to the recurrent layer. In the scaled image the receptive field of an input vector to the recurrent layer carries different information according to the strength of the applied scaling. Although the authors of the model claimed that their main contribution is the model's ability to process variable text lengths, they did not really apply it and they rescaled disproportionally all training and testing images to a constant size.

Thus we modified the network and its training to remove the image scaling and instead we padded the image with a background color to the fixed size. The difference of the scaling and padding can be seen in Fig. 4.4. In principle by applying the padding we could handle both short and long words and keep the number of receptive fields that correspond to one character always same.

Due to practical limitations it is infeasible to implement only the padding of the images, because larger input sizes do not fit into the used GPU's memory. However, only 5% of the training images are wider than 200 pixels and only 0.3% of the images are wider than 300pixels. Therefore we use a combined solution: padding up to 200 pixels and scaling down above this limit. This way only a small portion of the images are actually scaled down and the applied distortion through the scaling is less extreme, which means the images in the training set correspond more to the test images.

Figure 4.4: Comparison between scaled and padded image.

As can be seen in Tab. 4.4 the proposed change in scaling of images only slightly improved the accuracy of the model. The difference in the performance is more visible in Tab. 4.5, where the tests were run on the whole dataset including images shorter than three characters. These short words are typically excluded from the evaluation as they are difficult to distinguish from noise and in [29] also lead to the extreme stretching of the image.

[1] The results from the model provided on Github repository. The model has been created by an unknown improvement.

Model	Accuracy [%]
Retrained model	87.05
Pad200 (mean)+scale down	87.51

Table 4.4: Results ICDAR 2013.

Model	Accuracy [%]
Retrained model	79.01
Pad200 (mean)+scale down	86.21

Table 4.5: Results ICDAR 2013 - all images, including words under 3 characters.

In Tab. 4.6 the results for padding with different backgrounds are compared. It can be observed, that the type of padding does not make a big difference in the accuracy of the model, however in the first 300k the mean padding seemed to achieve the best results. In the further experiments we chose to use the mean padding.

Training	Accuracy [%]
Pad200 (white)+scale down	87.00
Pad200 (black)+scale down	87.30
Pad200 (mean)+scale down	87.51

Table 4.6: Comparing the results on CDAR 2013 for different padding backgrounds.

4.2.3 Case sensitivity

In the next experiment the model was trained to recognize character cases. This was achieved by increasing the number of classes from 36 to 62. This also lead to a larger model with more parameters and slightly slower training. The change of the recognition towards case sensitive is an important step towards better understanding of the meaning of the provided text. The ICDAR 2015 challenge is also concentrated on the case sensitive recognition, so the results were important for the comparison towards other algorithms.

In the Tab. 4.7 the results of the testing on the ICDAR 2013 dataset can be observed. The results of the case sensitive model are naturally worse than for the case insensitive models. The model has to classify into more classes which makes the classification more complicated. It can be seen that the enlarged the model through the combination of padding and scaling results into more difficult model to train.

Model	Accuracy [%]
Case sensitive (scaling)	82.61
Case sensitive (pad200 (mean)+scale down)	81.45

Table 4.7: Results of the case sensitive model on ICDAR 2013.

The results in Tab. 4.8 show that the case sensitive model combined with padding is still better in the task of all recognition of images of all length.

Model	Accuracy [%]
Case sensitive (scaling)	77.34
Case sensitive (pad200 (mean)+scale down)	80.50

Table 4.8: Results of the case sensitive model on all images in ICDAR 2013.

Flawed labelling in the training dataset

The inability of the model to achieve similar results to the case insensitive models even after longer training period lead to a more detailed analysis of the training dataset. The analysis revealed that 25 out of 362 randomly picked training images were labelled with wrong cases. The reason for those errors is probably based on the used fonts out of which some may not support lower or upper cases or distinguish between cases in a standard way. Examples of those images can be seen in Tab. 4.9

rectangle	oafishness	disarrangement
		

Table 4.9: Examples of wrong case sensitive labelling in the training dataset

4.2.4 Imposing blanks

The experiment was based on the idea, that imposing of blanks between all letters could simplify the scenario for the RNN layer by making the information less noisy. The imposing of blanks was introduced by changes to the forward-backward algorithm [9], the algorithm rejected the probability of the sequences that included two distinct labels next to each other. This way it imposed that a blank would be inserted between those two. The number of receptive fields (length of sequence for the BLSTM layer) was doubled so that even the longest word in the training dataset (24 characters long) could be classified.

As shown in Tab. 4.10 the experiment didn't bring any improvement to the model, on the contrary it have underperformed the original model. The reason to the results could be that the model has enlarged too much without receiving enough additional information, and that by imposing additional restrictions on the paths through which the correct labelling had to be predicted, the RNN layer had to learn to return more detailed predictions.

Model	Accuracy [%]
Retrained model	87.05
Imposing dashes	85.65

Table 4.10: Results of the imposing blanks on ICDAR 2013.

4.2.5 Results for ICDAR 2015

Results (Tab. 4.13) from the evaluation of the proposed methods on ICDAR 2015 correspond to the analysis of the dataset, that the testing dataset differs widely from the Jaderbergs' training dataset.

The evaluation of the models on the ICDAR 2015 dataset is important for the comparison with other state-of-the-art models, because the results of the original model for this dataset have not been previously published. ICDAR 2015 is a case sensitive test, thus the case sensitive model was important for the comparison with other models.

As the images in ICDAR 2015 are often rotated even in 90 degrees (in both directions) a proposed improvement is to implement some preprocessing to the input data. Two types of simple preprocessing were tested on the basic retrained model: 90 degree rotation of the input images in direction based on the confidence of the prediction from the model and rotation by 45 degrees (also with the confidence selection). The confidence of the prediction is the probability of the selected label l. It is calculated by $p(l) = \prod_{t \in T} y_t^k$, where y_t^k is the probability of symbol k at time-step t. As can be seen in Tab. 4.11 the results of the two experiments are equivalent.

Model	Accuracy [%]
90 degrees rotation	53.68
45 degrees rotation	53.59

Table 4.11: Comparing the results of types of rotation on the basic model, disregarding the cases.

The 90 degrees rotation preprocessing was then added to the models from the original experiments and the models were tested in the ICDAR 2015 challenge. In the first experiment the case sensitive model was compared with the present-day best solution in the challenge, the results can be seen in Tab. 4.12. It can be seen that the model doesn't achieve the state-of-the-art results on ICDAR 2015, however, it has achieved better results than the solutions published in ICDAR 2013 (MAPS [22]).

Model	Accuracy [%]
SRC-B-TextProcessingLab	62.11
Case sensitive	43.81
Pad200 (mean)+scale down	43.19
MAPS	32.93

Table 4.12: Comparing the results of case sensitive models on ICDAR 2015 dataset.

In Tab. 4.13 it is shown that the proposed changes did not improve the accuracy of the model on the ICDAR 2015 challenge. It can be seen that the two tasks, the ICDAR 2013 and the ICDAR 2015, are quite different scenarios of the image-based text recognition, which corresponds to the analysis in Section 4.1.

Model	Accuracy [%]
SRC-B-TextProcessingLab	64.95
Retrained model	53.68
Pad200 (mean)+scale down	52.58
Imposing dashes	51.95

Table 4.13: Comparing the results of our methods on ICDAR 2015 dataset, disregarding the cases.

4.3 Training and testing data comparison

During the experiments in the previous section we observed some specific properties of the errors on the two datasets which will be further described and explained in this section.

ICDAR 2013

In all finished experiments on ICDAR 2013 dataset a discrepancy of about 5% in the performance on the validation and the test set was observed. Another issue was that by training for longer time the accuracy of the model on the validation set has been slowly improving, but on the testing set the accuracy started to decline at some point. This difference does not match the expectation that the accuracy on the validation set should correspond to the actual accuracy on the test set. There could be several reasons for the difference, like not totally realistic dataset, different distribution of lengths of words, different N-gram statistics, etc.

In the case of different length of words the recurrent layer would learn an incorrect probability of the number of characters in a word. The N-gram differences would also lead towards mistakes in the RNN layer. The RNN layer learns dependencies between following characters (features), depending on the last output the recurrent layer expects a certain output in the next step.

The difference in lengths of words can be seen in Fig. 4.5. It is shown, that the training dataset has the distribution of words shifted towards longer words.

The differences in the frequency of bigrams and trigrams can be seen in Fig. 4.6 and Fig. 4.7, respectively. The most frequent 100 bigrams and trigrams were compared, which account for 50% of all bigrams and trigrams in the datasets. The histograms show the percentage of different scales of the training to testing dataset frequency ratios in the whole dataset.

It can be seen that a discrepancy in frequencies between the two datasets exists. The reason for it could be that all English words were used to generate the training dataset with uniform frequency which is obviously not corresponding to their real frequency of occurrence in natural language and consequently also in the test images.

Those differences may be influencing the performance of the recurrent layer which is fitting to the distributions present in the training set and therefore through longer training

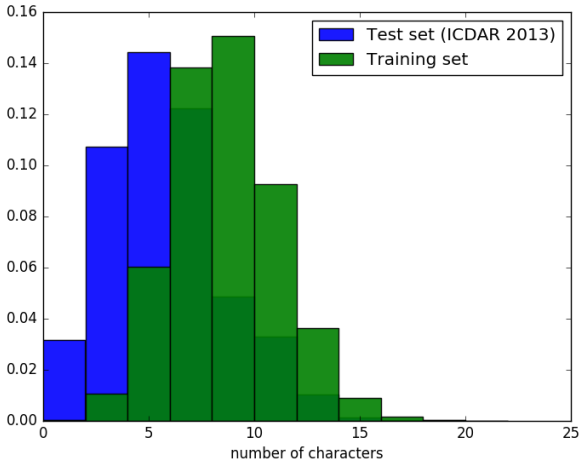


Figure 4.5: Histogram of length of words in training and testing dataset

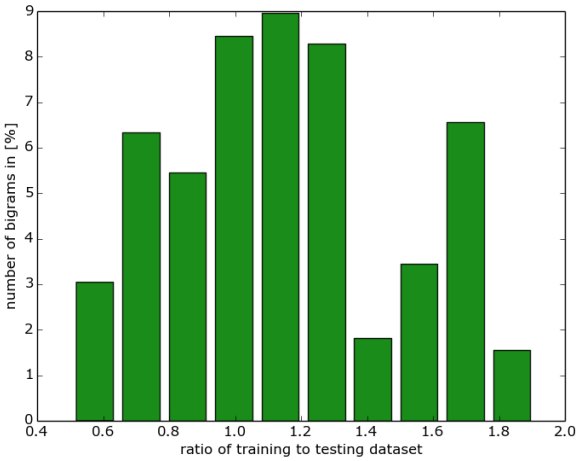


Figure 4.6: Histogram of bigram frequency ratios of training to testing dataset. The histogram was weighted by the percentage of compared bigrams in the whole dataset.

it can deteriorate it's performance. It is difficult to overcome this limitation as the real distribution of words in natural language would be difficult to obtain.

ICDAR 2015

The tests on ICDAR 2015 showed that the model is not capable of dealing with more complicated scenarios. The 'Incidental text' dataset contains many images with extreme rotation and inclination of text, therefore the test requires different qualities than the previous ICDAR 2013 dataset. The CRNN model has not achieved very good results on the ICDAR 2015 dataset. The reason for those poor results could be the insufficient capacity of the

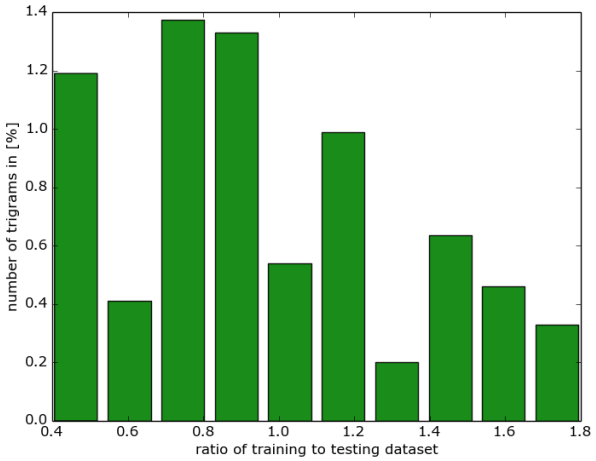


Figure 4.7: Histogram of trigram frequency ratios of training to testing dataset. The histogram was weighted by the percentage of compared trigrams in the whole dataset.

model and also the lack of distortion in the training dataset.

There is a big difference between the training data and ICDAR 2015 dataset in the amount of distortion of the text within the images. The Jaderberg synthetic dataset has been created with the focus on variance in fonts, sizes and different backgrounds, lighting conditions etc. Some distortion has also been applied to the images, however, the distortions are less extreme as in the ICDAR 2015 dataset. The different amount of rotation and distortion in general in testing and training datasets could lead towards incorrectly extracted features from the convolutional neural network.

The rotation of text in images in the Jaderberg dataset is determined by the angle $r = 5 \cdot N(0, 1) + 0)$ in degrees. The rotation of images is centered in the middle of the images. The value r shows that 50% of images are rotated by less then 7 degrees and only 20% of images are rotated by more than 13 degrees. This shows that the Jaderberg synthetic dataset doesn't provide with enough rotation in the training images. This indicates that the model trained on this dataset won't able to cope with more severe distortion in the testing images.

Chapter 5

Conclusion

The goal of this thesis was the analysis of the CRNN model [29], the design and evaluation of possible improvements to the model.

Three main parts of the CRNN model were described in the theoretical part: the VGG Net, the BLSTM and the CTC algorithm. We analyzed the possibilities for improvements of the studied model and proposed three changes to the model: retraining of case sensitive model, reduction of scaling through padding of the images and the imposing of blanks in the rnn made predictions.

For the evaluation of the proposed changes and comparison with the original model we replicated the results published in [29]. The accuracy of the retrained model even slightly surpassed the original results on ICDAR 2013 dataset. The model was also evaluated on the ICDAR 2015 dataset which enabled the comparison to the state-of-the-art solutions on the task, because the model had not been evaluated before on this dataset.

The evaluation of case sensitive model showed that after enlarging the model it was more difficult to achieve good results. The case sensitive model underperformed the state-of-the-art results on the ICDAR 2015 challenge. As described in Section 4.2.3 of the Practical part flaws in the case sensitive labelling of the training dataset are also decreasing the testing accuracy.

The results of the experiment with padding confirmed the drawbacks of the scaling approach in the original model. Padding and reduced scaling also largely improved the performance on the full ICDAR 2013 dataset including words shorter than 3 characters.

The experiment of imposing dashes did not improve the accuracy of the model. The reason for it could be that the model complexity was increased too much without receiving enough additional information and the restrictions imposed by the CTC layer made it harder for the RNN to give the correct prediction.

In Section 4.2.3 it has been shown that the synthetic dataset by Jaderberg et al. [17] is unsuitable for the training of the case sensitive model as the dataset includes many errors in the case sensitive labellings. In Section 4.3 the comparison of the training and testing dataset showed that the datasets have different distributions of word-lengths and N-grams

occurrences. This discrepancy might negatively influence the recurrent layer, which consequently learns a structure which differs from the natural language words.

The results on ICDAR 2015 indicated that the preprocessing of the input images will be principal to any good text recognition model in the future. As was shown in [30], the correcting transformation of the input images is improving the results of the recognition. The newly proposed differentiable module called the spatial transformer network (STN) by Jaderberg et al. [19] which can be jointly trained by a gradient descent method offers one direction where the preprocessing of the input data could lead.

It has been confirmed that the direction of the studied architecture - the combination of a feature extractor with a recurrent neural network achieves good results in the text recognition task. The findings from the thesis offer following directions for the future work: the improvement or replacement of the training set and the study and implementation of preprocessing of the input images to the model.

Bibliography

[1] *Proceedings of the 3rd International Workshop on Historical Document Imaging and Processing, HIP@ICDAR 2015, Nancy, France, August 22, 2015*. ACM, 2015.

[2] Y. Bengio, A. C. Courville, and P. Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[3] M. Bianchini and F. Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems*, 25(8):1553–1565, Aug 2014.

[4] A. Bissacco, M. Cummins, Y. Netzer, and H. Neven. Photoocr: Reading text in uncontrolled conditions. In *2013 IEEE International Conference on Computer Vision*, pages 785–792, Dec 2013.

[5] B. C. Csáji. Approximation with artificial neural networks. *MSc Thesis, Eötvös Loránd University (ELTE), Budapest, Hungary*, 2001.

[6] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255, June 2009.

[7] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2963–2970, June 2010.

[8] A. Graves. *Supervised sequence labelling with recurrent neural networks*. Studies in Computational intelligence. Springer, Heidelberg, New York, 2012.

[9] A. Graves, S. Fernández, and F. Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *In Proceedings of the International Conference on Machine Learning, ICML 2006*, pages 369–376, 2006.

[10] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

[11] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer and Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[13] R. Holley. How good can it get? analysing and improving ocr accuracy in large scale historic newspaper digitisation programs, Mar 2009.

[14] M. Hubel and T. N. Wiesel. *Brain and Visual Perception*. Oxford Univeristy Press, 2005.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[16] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Reading text in the wild with convolutional neural networks. *CoRR*, abs/1412.1842, 2014.

[17] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. *arXiv preprint arXiv:1406.2227*, 2014.

[18] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Deep structured output learning for unconstrained text recognition. In *International Conference on Learning Representations*, 2015.

[19] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.

[20] D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. G. i. Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazàn, and L. P. de las Heras. Icdar 2013 robust reading competition. In *Proceedings of the 2013 12th International Conference on Document Analysis and Recognition*, ICDAR '13, pages 1484–1493, Washington, DC, USA, 2013. IEEE Computer Society.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.

[22] D. Kumar, M. Prasad, and A. Ramakrishnan. Maps: midline analysis and propagation of segmentation. In *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*, page 15. ACM, 2012.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[24] C.-Y. Lee and S. Osindero. Recursive Recurrent Nets with Attention Modeling for OCR in the Wild. *ArXiv e-prints*, Mar. 2016.

[25] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

[26] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767, 2004.

[27] L. Neumann and J. Matas. Scene text localization and recognition with oriented stroke detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 97–104, 2013.

[28] M. Schuster. Bi-directional recurrent neural networks for speech recognition. Technical report, 1996.

[29] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *CoRR*, abs/1507.05717, 2015.

[30] B. Shi, X. Wang, P. Lv, C. Yao, and X. Bai. Robust scene text recognition with automatic rectification. *CoRR*, abs/1603.03915, 2016.

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

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

[34] P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

Appendix A

Contents of the CD

The root directory contains README which explains how to compile and use the CRNN model and the extensions.

Directory structure of the CD

```
.
├── bachelor_thesis (.pdf of the bachelor thesis)
├── crnn (The source code of the convolutional neural network model)
│   ├── data
│   ├── model
│   ├── src (The original source code)
│   ├── src-dashes (Implemented experiments)
│   ├── src-test-case-sensitive
│   ├── src-test-case-sensitive-padding-mean
│   ├── src-test-model
│   ├── src-test-pad-scale-200-black
│   ├── src-test-pad-scale-200-mean
│   ├── src-test-pad-scale-200-white
│   ├── third_party
│   └── tool (Scripts for the creating the datasets)
├── data_word_stats (Scripts for the dataset analysis)
│   ├── data_stats
│   └── word_stats
└── metacentrum (Scripts for running tasks at Metacentrum)
    ├── case_sensitive
    ├── create_dataset
    ├── imposing_dashes
    ├── model_check
    └── padding
```