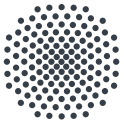


University of Stuttgart
Germany

Fraunhofer
IPA

End-to-End Scene Text Recognition based on Artificial Neural Networks

Markus Völk

May 2018

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in Engineering Cybernetics

Examiner

Prof. Dr.-Ing. Alexander Verl
Institute for Control Engineering of Machine Tools and Manufacturing Units
University of Stuttgart

Supervisor

Dipl.-Ing. Richard Bormann M.Sc.
Robot and Assistive Systems Department
Fraunhofer Institute for Manufacturing Engineering and Automation

Declaration

I hereby declare that this thesis and the related work submitted for partially fulfilling the requirements of the degree of Master of Science in Engineering Cybernetics at the University of Stuttgart are my own effort. It contains no material that has been accepted for the award of a degree at this or any other university. To the best of my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made in the text.
I further declare that I have performed this thesis according to the existing copyright policy and the rules of good scientific practice. In case this works contains contributions of someone else (e.g. pictures, drawings, text passages etc.), I have clearly identified the source of these contributions, and, if necessary, have obtained approval from the originator for making use of them in this thesis. I am aware that I have to bear the consequences in case I have contravened these duties.

Markus Völk
May 2018

Aknowledgement

I would like to thank my parents for their continuous support during my time at graduate school.
I would also like to express my gratitude to my supervisor and scientific advisor, Richard Bormann, for his guidance during the course of this thesis.
In addition, I would like to express my gratitude to Martin Hägele, head of the Robot and Assistive Systems department, who gave me his trust during my time at the IPA. Without him this work would not have been possible.
I would also thank Daniel Bargmann, who took the time for proofreading and pointing out last mistakes.

End-to-End Scene Text Recognition based on Artificial Neural Networks

Markus Völk

Abstract

The extraction of textual information from natural image data or simple scene text recognition is a challenging task with various applications. The difficulty of this task is caused by perspective distortions, environment disturbances and often poor image quality, making it an active research area that recently enjoyed rapid progress due to the use of Artificial Neural Networks (ANNs).
The goal of his work is to develop an end-to-end scene text recognition system that surpasses the current state-of-the-art in terms of accuracy and cost efficiency.
Therefore, an efficient two-stage end-to-end scene text recognition system, based on Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) is presented. The proposed system significantly outperforms prior approaches (increased f-measure by 8.3 percent points) while simultaneously reducing the model size of both, detection and recognition stage (47% less model parameters in the detection and 9% less in the recognition model). It also shows good real-time capabilities (~21.8 fps on detection and ~13.1 fps on end-to-end recognition). Both models can be trained from scratch in less than two days on a commercially available consumer-grade GPU. The improved performance is mainly the result of concepts such as Dense Blocks, Gated Recurrent Unit (GRU) and Focal Loss. The whole recognition pipeline was developed in Python with Keras and TensorFlow and its source code is freely available under MIT license.

Contents

Chapter 1

Introduction

In the field of machine learning, the availability of large, labeled datasets and affordable GPUs for efficient parallel computing has led to great advances in the last few years. In particular, the application of Artificial Neural Networks (ANNs) to practically all conceivable problems in Computer Vision produced impressive results. One specific application of ANN is text recognition. While the problem of recognizing text in a constrained environment like scanned documents can be considered as solved these days, the recognition of text in unconstrained environments like city landscapes, grocery stores or at workplaces is still an insufficiently solved and challenging problem. Due to an increasing number of use cases for scene text recognition systems, scene text recognition or text spotting in the wild has become subject to active research that has recently seen great progress.

Scene text recognition systems can provide valuable textual information for a wide range of application scenarios. Most of them are intended for augmented reality solutions on mobile devices, such as automatic real-time translation, user navigation, driving assistance, human-computer interaction or aid tools for visually impaired people. Other application scenarios involve scene understanding for navigation and task planing in self-driving cars and autonomous robot systems. Scene text recognition can be used to extract structured information from large sets of image data. Street-view-like images may be exploited for map services and geographic information systems and web images for image understanding and visual search engines. And as always for AI applications, there are surveillance scenarios, such as license plate recognition for criminal investigation.

However, scene text recognition is challenging for various reasons. Scene text instances show a high variability in scale, aspect ratio, font, language and style. They may also be arbitrarily oriented, partially occluded, perspectively distorted, exhibit curvature or interfere with the background. In addition to the difficult environmental conditions, the image quality is often quite poor. The images may have low resolution, contrast or saturation and may also suffer from reflections, from non-uniform illumination, blur, noise or compression artifacts. All those facts make scene text reading a problem that is not easy to solve.

The goal of this work is to provide an overview of recent approaches related to scene

text reading and to develop an end-to-end scene text reading system that hopefully surpasses the prior art approaches.
Thus, the main contribution of this work is twofold. First, this works provides an overview of recent concepts and ideas related to generic CNN-based object detection, scene text detection and recognition as well as approaches to end-to-end trainable scene text recognition. And second, an end-to-end scene text recognition pipeline is proposed that is composed of a detection and a recognition stage and significantly outperforms prior approaches in terms of accuracy while simultaneously considerably reducing the model size. The new detection model is 47% smaller than in the prior approach and can be trained from scratch, instead of relying on features obtained by pre-training a model for image classification. The new recognition model requires 9% less memory compared to a prior approach without sacrificing performance. The whole pipeline also shows good real-time performance, ~21.8 fps on the detection task and ~13.1 fps on the end-to-end recognition task.
The rest of this thesis is structured as follows. In Chapter 2 a theoretical foundation to ANNs is provided and several concepts, relevant for the following chapters, are introduced. Chapter 3 considers recent approaches and ideas related to general CNN-based object detection, which are often taken as the foundation for various text detection approaches. Next, in Chapter 4, state-of-the-art approaches to scene text detection and recognition are investigated and relevant ideas are collected. Chapter 5 describes the implementation of an end-to-end scene text recognition system and justifies the related design decisions. In the experiments Chapter 6, several models based on the implementation in Chapter 5 are trained and evaluated with regard to performance, speed, complexity and generalization. Qualitative results are discussed. The work ends with a final discussion and an outlook on further research in Chapter 7.

Chapter 2

Artificial Neural Networks

Humans and animals process information with neural networks, formed by trillions of nerve cells (neurons), which are connected via synapses to exchange electrical signals, the so-called action potential. Computer algorithms that mimic this biological structures are called Artificial Neural Network (ANN). Their development is primary driven by the desire to understand how the human brain works and the demand for computer systems that can deal with abstract and poorly defined problems, making them an active field of research.

ANNs or short networks in our context, have a long history dating back to the 1940's (McCulloch and Pitts 1990) when the ideas of cybernetics came up. Later in the 1950's and 60's people started to build learning machines based on the Hebbian theory[1] and by means of analog computers and vacuum tubes. This was also the time when the Perceptron (Rosenblatt 1958), a single-layer predecessor of today's networks, was invented. Later, the research on ANNs stagnated after Marvin Minsky, founder of the MIT AI Lab, and Seymour Papert, director of the lab, published their book (Minsky and Papert 1969) in which they expounded two main issues facing the perceptron model. First, the basic perceptrons are incapable of learning the XOR circuit, and second, they could not be easily extended to a multi-layer network due to the required amount of computation. The following period of reduced funding and interest in AI research is known as the first of several AI winters (Smith et al. 2006). In the 1980's the focus of the community was more on expert systems and later in the 1990's and 2000's, SVMs were the way to go. The true success story of ANNs began in the 2010's with the availability cheep computation power, large datasets and applications in computer vision and natural language processing. Today AI is practically defined by ANNs.

Before this chapter presents a theoretical foundation for neural networks, some textbooks may be recommended to the ambitious reader. Goodfellow et al. 2016 is freely available online and probably the most complete reference related to deep learning, Bishop 2006 is about machine learning in a more general way, and last but not least, Murphy 2012 provides a completely probabilistic approach to machine learning.

The rest of this chapter is now structured as follows. Section 2.1 introduces the

[1] The Hebbian rule in its shortest form says "Cells that fire together wire together."

basic framework of ANNs. Section 2.2 considers the training process and various related techniques in more detail. Section 2.3 addresses common feed forward layers and provides a mathematical description of them, followed by Section 2.4 where the same is done for recurrent layers. Section 2.5 considers various task dependent loss functions required to train ANNs. Section 2.6 is about neural network architectures and the final Section 2.9 gives an overview of open source software for implementing them.

2.1 Basic Framework

On a cellular level, a neuron receives input signals on its synapses, performs some internal pattern matching and produces an input dependent output signal, which is then transmitted over its axon to other neurons. The most popular approach to mathematically model this behavior is to do a linear combination $\sum_i w_i x_i$ of the input signals x_i, add some bias term b and throw it through a nonlinear activation function for thresholding. The analogy between the biological neuron and the mathematical model is depicted in Fig. 2.1. The process of finding suitable parameters (weights w_i and bias b) that enable the neuron to produce a desired output is called training or learning.

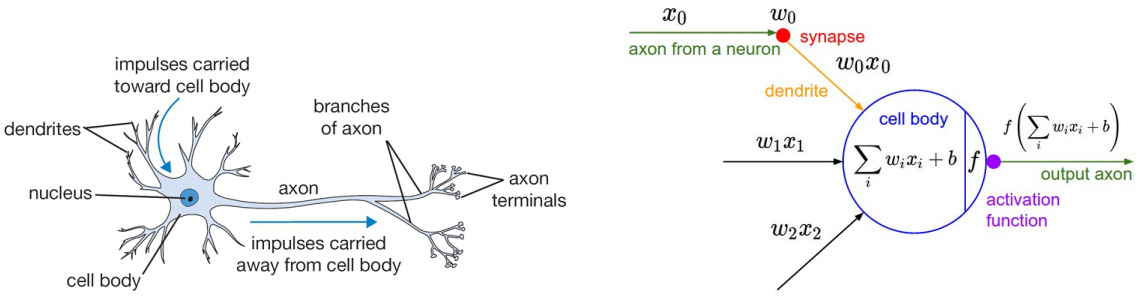


Fig. 2.1: Drawing of a biological neuron (left) and its mathematical model (right)[2].

Considering the topological structure of ANNs, one can distinguish between Feed-Forward Neural Networks and Recurrent Neural Networks. In FFNNs the signals can only flow in one direction, from input to output, whereas RNNs contain internal feedback connections allowing the signals to flow in both directions.
Most ANNs that are used today, are organized in a layer-wise fashion, similar to the FFNN in Figure 2.2. They contain an input layer, one or more hidden layers and an output layer. Each layer, except of the input layer, contains a certain number ob units (neurons) with nonlinear activation functions. In other words, the input layer is passive and does not modify the signals. Further one may distinguish networks between Shallow and Deep Neural Networks (DNNs), where 'shallow' is a term used

[2]Source *CS231n Convolutional Neural Networks for Visual Recognition* 2017

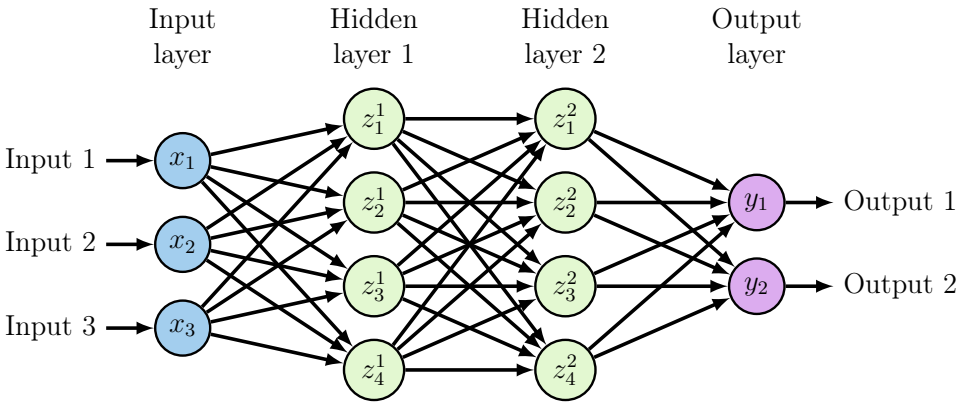


Fig. 2.2: Feed-Forward Neural Network with two hidden layers; subscript indicates the unit index in the layer, superscript the index of a hidden layer.

for networks with one or just a few hidden layers and 'deep' in contrast is used to refer on networks with many layers. Deep in this sense is also the term that gives its name to the Deep Learning (DL) paradigm that can be witnessed today.
In nearly all current applications of ANNs, ANNs are used as function approximators and the training is done in a supervised learning setup. Supervised learning is the case, in which only a limited set of N labeled samples $(X_n, Y_n)_{n=1}^N$ is available for training[3]. The relation between the observation X_n and the label Y_n is then defined by the unknown target function f, which we want to approximate.

$$Y_n := f(X_n) \tag{2.1}$$

Finding an approximator $\hat{f}$ with correct behavior is usually an ill-posed problem with many solutions where each solution may perform different on samples that are not contained in the training set.
To tackle the search for a good approximator, one may utilize a parametric function $\hat{f}_\theta$, where θ are the free parameters controlling the behavior of $\hat{f}_\theta$. If $\hat{f}_\theta$ represents an ANN that maps the network input $x \in \mathbb{R}^D$ to the network output $\hat{y} \in \mathbb{R}^C$, then the trainable parameters of the network, in particular the weights and biases of all the units, are the parameters θ. If we further assume that $\hat{f}_\theta$ is differentiable with respect to θ and introduce an also differentiable error measure, the so called Loss function $\mathcal{L}(Y, \hat{f}_\theta(X))$, which compares the approximation with the label, then the training of the network can be treated as an in general highly nonlinear and nonconvex optimization problem that can be solved by Gradient Descent (GD).
Expressing the gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ in an analytic form is, especially for DNNs, a practically infeasible undertaking. Therefore, a more efficient concept called backpropagation (Rumelhart et al. 1986) was developed. It is basically a divide and conquer strategy for a progressive computation of the gradient. To understand this idea, it is helpful

[3]Unsupervised learning in contrast is the case when no labels are available and the system has to learn the underlying pattern from the data alone.

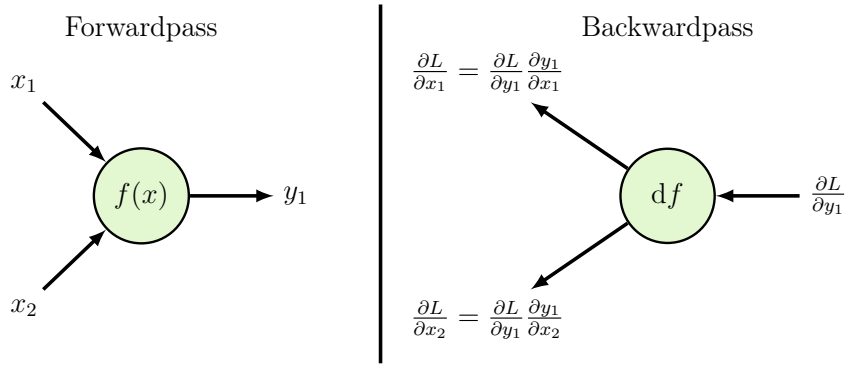


Fig. 2.3: The chain rule applied to one single unit.

to interpret a network an acyclic computation graph. By applying the chain rule on each node of the graph, the gradients with respect to the signals entering the node can be expressed by the gradients with respect to the signals leaving the node and the gradient of the unit itself (Figure 2.3). By recursively computing and caching the gradients of the signals in the network, all the elements of $\frac{\partial \mathcal{L}}{\partial \theta}$ can be computed without explicitly writing down the analytic expressions. This concept is also known as automatic differentiation in reverse-mode (Griewank 2012) and a survey on its application in machine learning can be found in Baydin et al. 2015. The gradient that is backpropagated is sometimes referred to as the error signal, which is backproagated and used to adjust the parameters in the network in such a way that the error decreases. For the rest of this work, the author tries to avoid the excessive derivation of gradient expressions and assumes that automatic differentiation is performed by one of the software solutions listed in Section 2.9.
The convergence to a global optimal θ^* of a highly nonlinear and nonconvex function $\hat{f}(\theta)$ is in general not guaranteed by GD and the algorithms usually end up in a local optimum. Parameters that are not contained in the trainable parameters θ, but influence the structure or training of the network, are called hyperparameters. Neural networks in general are parallel computing devices and a CPU architecture is for sure not the best place for implementing them. Also, one must be careful with the analogies to the biology. Nature is much more complex and it seems that no biological equivalent can be found for simple mathematical concepts such as backpropagation.

2.2 Training

As mentioned in Section 2.1, training of ANN is most often done with backpropagation. The purpose of this section is now to look at the training process in more detail and explain various training-related techniques.
To clarify the basic terminology, an iteration contains all the steps required to per-

form a parameter update by means of GD. A batch is a set of training samples that is used for one iteration and the number of samples in a batch is called the batch size. One cycle through all samples in the training set is called an epoch. The whole training process is summarized by the pseudo code in Algorithm 2.1.

```
initialize trainable parameters
repeat
    randomize order of training samples
    for each batch in epoch
        for each sample in batch
            forward propagate input by iteratively computing the output of layers 1 to L
            get prediction and compute gradient of loss function with respect to the output
            accumulate loss gradient over batch
        end for
        back propagate the accumulated gradient from layer L to 1
        update parameters for all units by GD
    end for
until stopping criteria is reached
```

Algorithm 2.1: Training Algorithm based on Backpropagation and Gradient Descent

2.2.1 Initialization

The solution of a large non-convex optimization problem depends significantly on the initial values of the trainable parameters.
The naive approach of initializing with zeroes works well for biases, but using the same strategy for the weights in combination with tanh activation functions (Section 2.3.2) results in a saddle point and no learning progress. Setting all weights to the same constant value also fails because they would all behave in the same way.
So the pragmatic solution to break these symmetries and make training possible is random initialization. Today's most popular initialization method, the Xavier Initialization, was introduced by Glorot and Bengio 2010. It initializes the weights randomly by drawing the values from a Gaussian distribution with a variance determined by $\text{var}(w_i) = 1/n_{\text{in}}$, where the w_i is one of the units weights and n_{in} is the number of inputs for the unit. This method brings the weights for the tanh activations to a reasonable scale and also partially solves the vanishing gradient problem that occurs when the input to an activation is too large and the tanh saturates. It also addresses the opposite of the vanishing gradient problem, the case in which the inputs of the tanh activation functions are to close to zero, where the tanh activations are practically linear and the network looses its non-linear properties. For ReLU

units (Section 2.3.2), He et al. 2015b used $\text{var}(w_i) = 2/n_{\text{in}}$ instead $\text{var}(w_i) = 1/n_{\text{in}}$, which is known as He Initialization. The idea in both cases is to keep the variance of the layer input the same as these of the layer output. The factor 2 makes sense, because in the ReLU case half of the inputs are zero. Xavier Initialization sometimes appears with $\text{var}(w_i) = 2/(n_{\text{in}} + n_{\text{out}})$ to tradeoff these criterion for the forward and backward pass.

2.2.2 Batch Size

As mentioned above, weight updates are performed after each batch of training samples, and the number of samples used for an update is a hyperparameter called the batch size M. Depending on the choice of the batch size, one can distinguish the following training methods.

Batch Gradient Descent considers the full batch of all training samples ($M = N$) and the averaged gradient is used. This guarantees the convergence to a local (global if convex) optimum, but using the full training set at each iteration is computationally expensive, especially for large datasets and networks with a high degree of freedom.

Stochastic Gradient Descent performs a parameter update after each single training sample ($M = 1$). This requires much less computation, but results in fluctuations around the optimum and the exact optimum will never be reached. Since the Law of Large Number does not apply, the convergence of Stochastic Gradient Descent (SGD) is rather slow for datasets with high variance, but it is computationally efficient and can be used for online learning.

Mini-Batch Gradient Descent (Li et al. 2014) takes a small subset of the training set to average the gradient for a parameter update. This has less computation cost than using the full batch and simultaneously better convergence behavior then SGD.
Depending on the data and the model, there is an optimal batch size between 1 and N. In practice relatively small values such as 32, 64 or 128 are used and for large networks used for computer vision task, the batch size is often limited by the memory size of the GPU and is chosen as big as possible. The noise added by the stochastic nature of the method is a kind of implicit regularization (Section 2.2.6) and can help to escape from local optimum.
In most cases in which people talk about SGD today, they actually mean mini-batch gradient descent.

2.2.3 Update Rules

The optimization of larger networks is only feasible due to mini batch training and efficient optimization algorithms. Since second order algorithms, such as Newton's method, require the costly evaluation of the Hessian matrix, in practice almost exclusively first order algorithms based on Gradient Descent are used.
The basic rule for the parameter update in GD is given by (2.2), where t refers to the time before and $t+1$ to the time after the parameter update.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta) \tag{2.2}$$

The hyperparameter η determines the magnitude of the parameter change and is called the learning rate or step-size. A large learning rate will lead to a large decrease of the loss in the beginning, but later causes the insufficient decrease of the loss or unstable convergence behavior. For this reason, a good learning rate is large at the beginning and decreases when the minimum of the loss function is approached.
The choice of a good learning rate turns out to be a rather tricky undertaking. Simple approaches to control the learning rate are, changing it manually, automatically depending on the number of batches or epochs (learning rate decay or scheduler) or dependent on a performance measure like training/validation loss or accuracy.
Due to the desire for better convergence behavior and adaptive learning rates, more advanced versions of the basic update rule (2.2) were conceived in the last few years. Some impotent ones of them will be now explained, but a larger overview is provided by Ruder 2016.

Momentum (Qian 1999) utilizes the update term from the previous update step in the current step. This damps oscillation and speeds up convergence in cases where the surface of the loss function is in one dimension much steeper than in the other.

$$\begin{aligned} v_{t+1} &= \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta) \\ \theta_{t+1} &= \theta_t - v_{t+1} \end{aligned} \tag{2.3}$$

The hyperparameter γ controls the influence of the previous update values and is simply termed momentum (because of the physical interpretation). A typical value for γ is 0.9.

Nesterov momentum or Nesterov's Accelerated Gradient (NAG) (Botev et al. 2016; Nesterov 1983) is similar to the momentum method, but does some look ahead by evaluating the gradient at the position where it would end up with one step in the current direction. This minor change allows to slow down earlier and reduce

overshooting.

$$\begin{aligned} v_{t+1} &= \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta + \gamma v_t) \\ \theta_{t+1} &= \theta_t - v_{t+1} \end{aligned} \tag{2.4}$$

Experiments by Sutskever et al. 2013 have shown that NAG comes with increasing stability and sometimes performs better compared to classical momentum.

Adagrad (Duchi et al. 2011) adjusts the learning rate individually for each parameter by involving the squared sum of all past gradients. It avoids to manually tune the learning rate and works well for space data.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t \tag{2.5}$$

For the sake of brevity, $g_t = \nabla_\theta \mathcal{L}(\theta)$ evaluated at time t and G_t is a diagonal matrix with elements $G_{t,i,i} = \sum_{\tau=1}^{t} g_{\tau,i}^2$. ϵ is a small constant that avoids division by zero. The main weakness of Adagrad is that with steadily increasing G_t, the learning rate gets infinitesimal small and the learning process stops.

Adadelta (Zeiler 2012) avoids the steadily increasing denominator in Adagrad by only taking an exponentially decaying average of the previous gradients.

$$\begin{aligned} \mathrm{E}[g^2]_t &= \gamma \, \mathrm{E}[g^2]_{t-1} + (1-\gamma) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\mathrm{E}[g^2]_t + \epsilon}} g_t \end{aligned} \tag{2.6}$$

The hyperparameter $\gamma \in [0, 1]$ is chosen in a similar way as for the momentum method above and the vector operation are element-wise (e.g. $g_t^2 = g_t \odot g_t$). The authors of the original publication recognized that hypothetically introduced units in (2.6) are not consistent. This insight led to an update rule that completely eliminates the default learning rate.

$$\begin{aligned} \mathrm{E}[\Delta\theta^2]_t &= \gamma \, \mathrm{E}[\Delta\theta^2]_{t-1} + (1-\gamma) \Delta\theta_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\sqrt{\mathrm{E}[\Delta\theta^2]_{t-1} + \epsilon}}{\sqrt{\mathrm{E}[g^2]_t + \epsilon}} g_t \end{aligned} \tag{2.7}$$

with the initial values $\mathrm{E}[g^2] = 0$ and $\mathrm{E}[\Delta\theta^2] = 0$.

RMSprop is a frequently used method mentioned in the lecture of Hinton 2012a. It is actually a special case of Adadelta with $\gamma = 0.9$.

$$\begin{aligned} \mathrm{E}[g^2]_t &= 0.9 \, \mathrm{E}[g^2]_{t-1} + 0.1 g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\mathrm{E}[g^2]_t + \epsilon}} g_t \end{aligned} \tag{2.8}$$

Adam (Kingma and Ba 2014) stands for Adaptive Moment Estimation and utilizes exponentially decayed estimates of gradient and squared gradient. It is also a more probabilistic interpretation of the problem where the estimate of the gradient corresponds to the mean m_t and the estimate of the squared gradient corresponds to the uncentered variance v_t.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{2.9}$$

Initialization values $m_0 = 0$ and $v_0 = 0$ cause that the estimates are biased towards zero. To compensate for this a correction is applied.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{2.10}$$

The Adam update rule is then similar to Adadelta.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{2.11}$$

Typical values for the hyperparameters are $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

2.2.4 Generalization and Overfitting

Generalization is the ability of a model to perform well on samples that are not contained in the training set.
Overfitting is a phenomenon, that is closely related to generalization. It happens when the model fits the training data in such a way that it just memorizes all the individual training samples or it approximates the noise in the training data instead of learning the underlying correlations in the data distribution. The major causes for overfitting are a too small training set that does not sufficiently represent the true data distribution, and a too high degree of freedom in the model parameters, so that the model has the capacity to approximate the noise in the data.
With increasing overfitting, a model looses is ability to generalize. Therefore, the goal for obtaining a good model is to avoid overfitting and maximize its capability to generalize.
To get an estimation how well a model generalizes to data that are not contained in the training set, a second, normally smaller, test set is used on which the model is evaluated. Since the test set is not always available and it should not be involved in the design and training of the model, a third set, the so called validation set, is kept out of all available training samples and is used for model validation during the

training. The set of all available samples is normally split into two disjoint subsets, training set (80%) and validation set (20%).
Validation as part of the training process is done by evaluating the loss function on the validation set after each iteration or epoch and can be used for selecting hyperparameters. The validation loss will always be higher than the training loss, as long as the training and the validation data originate from the same data distribution. The gap between the training and the validation loss can be interpreted as a measure of overfitting. The larger the gap, the larger the overfitting and the smaller the models generalizability.

2.2.5 Stopping Criterion

A good question is, when stop training? Too few training iterations reduce the error insufficiently and too many can lead to overfitting the training data.
The pragmatic solution to this problem is to stop when the model starts overfitting, which can be determined by observing the validation loss during training. Stopping the training process after the validation loss stops improving, is called Early Stopping.
It is also common practice to train for a fixed number of iterations while regularly saving model snapshots and evaluating them afterwards.
The optimal training time is actually a hyperparameter with regularization effect.

2.2.6 Regularization

Regularization considers various methods for reducing overfitting.
The probably most common form of regularization is called Weight Decay or L1/L2 Regularization. It adds an extra term to the loss function that penalizes large weights and thus constrains the degrees of freedom that would be required to approximate the noise in the training data. It is also beneficial that it forces the network to learn small weights because large weights slow down the convergence process.
For L2 Regularization the penalizing term is given by $\frac{\lambda}{2n}\sum w^2$, where the sum is over all n weights w in the network. The hyperparameter λ controls the influence of the weight decay and the factor 2 is used for getting a term that leads to $\eta\lambda w$ in the parameter update (2.2). The L2 penalty is the L2-norm on the weights and can also be interpreted as a zero mean Gaussian prior on weights (Murphy 2012, Section 16.5.6). An interesting property of the L2 regularization is that, if the input signals of a unit are too strongly correlated, it will weight both of them the half rather than giving the full weight to one of them.

The L1 Regularization is constructed similar to the L2-regularization. By using the L1-norm instead of the L2-norm, one gets $\frac{\lambda}{n} \sum |w|$ as the penalty term. The L1 penalty brings many weights exactly to zero and therefore enforces the learning of sparse weight matrices. Sparse weights are virtually invariant to the measurement noise and can be helpful for the interpretation of the signals in the network.
In practice, the L2 regularization leads usually to better performance than the L1 regularization. The combination of L1 and L2 regularization is known as elastic net regularization (Zou and Hastie 2005).
Another simple regularization method is the application of hard constraints on the weights. This can be done by clamping the norm of the weight vector for each unit by a constant value ($\|w\|_2 < c$), which preserves the weights from exploding and keeps activation functions as tanh out of saturation. Another advantage of this method is that the weight updates are always bound, even if the learning rate was chosen much too high.
The regularization of biases is not common practice and leads to worse results (Hinton 2012b).
More advanced regularization techniques, that introduce randomness in the training process are Dropout (Section 2.3.6), DropConnect (Wan et al. 2013), Stochastic or Fractional Pooling (Graham 2014; Zeiler and Fergus 2013a) or simply the adding of noise in the forward pass (to the input, weights or activations).
Other efficient methods to conquer overfitting are Early Stopping (Section 2.2.5) or the artificial increasing of the training set, which is referred to as data augmentation. Data augmentation is often used for image data, where spatial transformations (shifting, scaling, rotating, flipping, etc.) or channel manipulations (brightness, contrast, saturation, color, etc.) are applied.
Furthermore, every kind of prior knowledge that helps the model to better generalize can be interpreted as regularization. For instance architectural constraints (number of units, number of layers, layer type etc.), supervised (Section 2.7) or unsupervised pre-training etc.
Many authors make a distinction between explicit and implicit regularization methods. The definition of these terms are not always clear. Sometimes explicit means, that regularization was the primary motivation for the concept, and implicit, that the regularization is more or less a side effect of the original idea. Another definition refers to implicit, when the method is only applied during training, and explicit, when it is done by construction and persist after training. Which of the definition is used is often apparent from the context.

2.3 Feed-Forward Layer

To provide a mathematical description of the various layer types that are used in this work, consider the output $z^l \in \mathbb{R}^{D^l}$ of a single layer in a FFNN. The superscript $l \in \{1, \ldots, L\}$ indicates the layer index and D^l refers to the length of the layer's output vector. Since the input to a layer is the output of the previous layer, the layer itself can be described by a mapping $f_\theta^l : \mathbb{R}^{D^{l-1}} \to \mathbb{R}^{D^l}$ or more precisely, the layer output can be expressed as

$$z^l(z^{l-1}(x)) = f_\theta^l(z^{l-1}(x)) \tag{2.12}$$

The output of the first (input) layer is then by definition the network input

$$z^0(x) := x \tag{2.13}$$

and the output of all further layers is the concatenation of the preceding layers.

$$z^l(x) = (f_{\theta^l}^l \circ \cdots \circ f_{\theta^1}^1)(x) \tag{2.14}$$

The network output $y(x)$ is then given as the output of the last layer $z^L(x)$ and the set of all trainable parameters in the network is the union of the parameters defined in the individual layers $\theta = \{\theta^L, \ldots, \theta^1\}$. In principle, the layer function f_θ^l can be any composition of mathematical operations, as long as they are differentiable and allow backpropagation.
This notation is applied for all subsequent layer descriptions, except for the convolutional and pooling layers, where it is slightly changed. It must also be mentioned that, in contrast to the simplified illustrations (Figure 2.2 and 2.5), the activation functions are now regarded as separate layers.
The rest of this section discusses the basic linear layer and nonlinear activation, followed by more computer vision related layers for convolution and pooling, softmax for classification and training related layers for Dropout and Batch Normalization.

2.3.1 Fully-connected

A Fully Connected or Dense Layer performs a linear transformation on its inputs and is therefore also known as Linear Layer. The layers of the MLP[4] in Figure 2.2 are all of type FC. The description of a fully connected layer is straightforward and given by (2.15). Where $W \in \mathbb{R}^{D^l \times D^{l-1}}$ are the weights and $b \in \mathbb{R}^{D^l}$ are the biases of

[4] A Multilayer Perceptron (MLP) is a FFNN that consists only of FC-layers and has at least one hidden layer.

the layer.

$$f_W^l(z^{l-1}(x)) = W^l z^{l-1}(x) + b^l \tag{2.15}$$

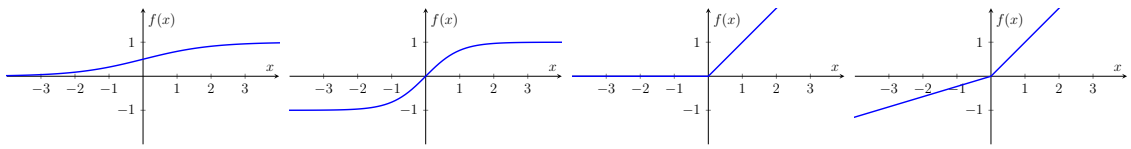
Weights and biases are the trainable parameters $\theta^l = \{W^l, b^l\}$ of a FC layer.
Since all units in a FC layer are connected to all inputs of the layer, the degree of freedom in the layer explodes combinatorially with the number of units in layer l and $l-1$. The resulting high number of $D^l D^{l-1} + D^l$ parameters make FC layers infeasible for large networks.
FC layers are either used for linear regression or whenever global connections are required. Convolutional layers (Section 2.3.3) in contrast have local connections and consider only local information.

2.3.2 Nonlinear Activation

To enable a ANNs to model nonlinear relationships, it requires nonlinear elements[5]. These nonlinearities are provided by the element-wise operations of activation functions, as formulated in (2.16).

$$f_\sigma^l(z^{l-1}(x))_d = \sigma(z_d^{l-1}(x)) \tag{2.16}$$

Due to the element-wise operation, the input dimension of an Activation Layer is equivalent to its output dimension ($D^l = D^{l-1}$). The index $d = 1, \ldots, D^l$ indicate the individual units in the layer and $\sigma : \mathbb{R} \to \mathbb{R}$ represents the activation function.
There are many possible, application-dependent candidates for activation functions. In the following a description of several important ones will be presented. They all share two desirable properties. First, they are piecewise differentiable, so that backpropagation can be applied, and second, they are monotonic increasing to avoid chaotic behavior during training.
A known problem with some activation functions is that their gradient vanishes if they are piece-wise constant or near-constant. A direct consequence of the chain rule in this case is that no error signal can be propagated back to previous layers that could be used for parameter update.



(a) Sigmoid (b) tanh (c) ReLU (d) Leaky ReLU

Fig. 2.4: Activation Functions

[5]It can easily be shown that the output of a concatenation of linear layers always depends linearly on the input.

Sigmoid or Logistic activation function $\text{sigm} : \mathbb{R} \to [0, 1]$ (Figure 2.4a)

$$\text{sigm}(u) = \frac{1}{1 + e^{-u}} \tag{2.17}$$

was originally motivated by its biological interpretation as the firing rate of a neuron. Today, it is primary used as output for binary classification, logistic regression or whenever a continues value between 0 and 1 is modeled. Sigmoid activations have some major drawbacks regarding DNNs. The gradient vanishes for large positive and negative input values, and since their output is not zero-centered, it will always be positive as well as the input to the subsequent layer.

Hyperbolic Tangent activation function $\tanh : \mathbb{R} \to [-1, 1]$ (Figure 2.4b)

$$\tanh(u) = \frac{1 - e^{-2u}}{1 + e^{-2u}} \tag{2.18}$$

is similar to the sigmoid activation function, it still suffers from vanishing gradients, but is centered at zero. The relation between the two is quite obvious and given by $\tanh(u) = 2\,\text{sigm}(2u) - 1$.

ReLU (Nair and Hinton 2010) is quasi the default activation function for DNNs today. The Rectified Linear Unit (ReLU) (2.19) maps any value in $\mathbb{R}$ to $\mathbb{R}^+$. Its graph is shown in Figure 2.4c.

$$\text{relu}(u) = \max(u, 0) \tag{2.19}$$

ReLU, compared to sigm and tanh, greatly accelerates the convergence of SGD (Krizhevsky et al. 2012). This is done by avoiding the vanishing gradient problem for positive input values and is computationally much cheaper. It can be implemented by simply thresholding a matrix of activations at zero and it does not require an normalization of the input (Krizhevsky et al. 2012). Unfortunately, ReLUs can "die" out. This happens when the weights are updated in such a way that the unit will never activate again. In practice, ReLU networks tend to learn sparse feature. Due to SGD, the chance is high that the gradient is none-zero for some input. Therefore, dying ReLUs are not a real problem as long as the learning rate is not chosen too high.

Leaky ReLU (Maas et al. 2013) provides a solution to the "dying ReLU" problem. The Leaky ReLU function (2.20) has a small slope μ for $u < 0$, as it can be seen in Figure 2.4d.

$$\text{leakyrelu}(u) = \max(u, 0) + \min(\mu u, 0), \quad \mu > 0 \tag{2.20}$$

By setting $\mu = 0$ the Leaky ReLU function is equal to the normal ReLU function. Xu et al. 2015a compared ReLU with Leaky ReLU and found that Leaky ReLU does slightly better.
There are plenty of other activation functions building on the concept of ReLU. The probably most notable are PReLU (He et al. 2015b), which uses trainable parameters for Leaky ReLU, and Maxout (Goodfellow et al. 2013b) as a generalization of ReLU and Leaky ReLU.

2.3.3 Convolutional

Convolution as a linear operator is long known and described by (2.21) (see Dominguez 2015 for history of convolution operator).

$$f(x) * g(x) = \int_{-\inf}^{\inf} f(s)g(x-s)\,\mathrm{d}s \tag{2.21}$$

It can be understood as measure of the similarity between two functions f and g, or as the weighting of f with g, g is called the kernel or filter of the convolution and it is said that f is convolved with g.
The discrete form of convolution

$$f(x) * g(x) = \sum_{i=-N}^{N} f(i)g(x-i) \tag{2.22}$$

is one of the most often used operations in signal processing and may be familiar from image manipulation software like GIMP (*GNU Image Manipulation Program* 2018, Section 8.2), where different convolution kernels are used to produce different effects like sharpening, blur or edged detection. The definition of convolution is strongly related to the definition of cross-correlation, by the fact that one obtains the cross-correlation when changing the minus sign in (2.21)/(2.22) into a plus sign. The idea behind Convolution Layers is to learn convolution kernels for efficient pattern recognition (a.k.a. template matching). In Figure 2.5, a simple version of a Convolutional Neural Network (CNN) is shown to illustrate some important properties of convolution layers. The first thing that strikes out in Figure 2.5 is that the three hidden layer are convolutional layers and differ from the FC layer in Figure 2.2. They are only locally connected and all the units in one layer share the same weights (same coloring indicates same weight). The weights of convolutional layers are also termed feature. The subset of input units that can influence a unit within a convolutional layer is called the receptive field (analogous to a biological vision system). For instance, unit z_3^2 can only receive signals from the input units x_4, x_5, x_6, x_7 and x_8, so they represent the receptive field of z_3^2. The number of

different weights in a convolutional layer is called the kernel size. In addition to the kernel size, there are two more hyperparameters for convolutional layers, padding and stride. Padding considers the behavior of the first and last units in the layer. There are two types of padding, valid (layer 3 in Figure 2.5) and same (layer 4). Valid means that no padding is performed, while same means, that zero padding, which is also commonly used in signal processing, is utilized to keep the layers in- and output dimension the same. Stride is the number of pixels the filter is shifted spatially (horizontally or vertically) on the layer input during convolution. For instance, layer 1 has stride 1, whereas layer 2 has stride 2.

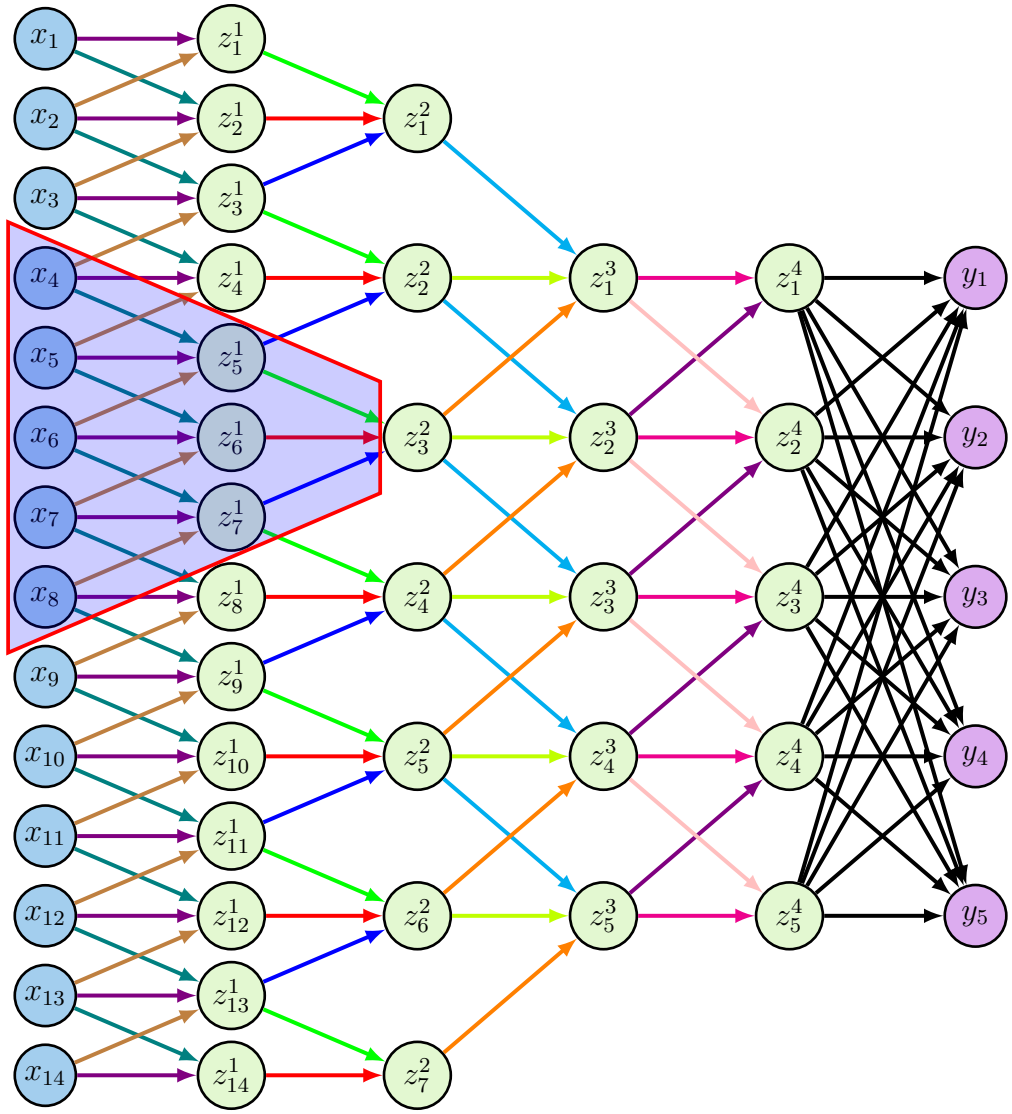


Fig. 2.5: 1D Convolutional Neural Network; subscript indicates the index of the unit in a layer, superscript the index of a hidden layer. The blue area with red boundary contains the units that are located in the receptive field of unit z_3^2.

Most tasks in computer vision demand for processing two-dimensional images with 3 channels (RGB) or more (stereo, depth, infrared, alpha, etc.). This leads to a network input of shape (I, J, C), where I is the height, J the width and C are

the channels of the image. Convolution is then performed in two dimensions and the need to learn more than one feature per layer results in a layer output of shape (I^l, J^l, C^l), where I^l and J^l are the spatial dimensions and C^l is the depth or number of features.

For a formal description, such a convolution layer can be considered as a mapping $f_C : \mathbb{R}^{I^{l-1} \times J^{l-1} \times C^{l-1}} \to \mathbb{R}^{I^l \times J^l \times C^l}$ with (M^l, N^l) as the kernel size and C^l as the parameter for defining the number of features in the layer.

As long as the kernel is larger than 1×1, the spatial output size will be smaller than the input size. To keep the dimensions the same (zero padding), $\lfloor (M^l - 1)/2 \rfloor$ rows before the first row, $\lfloor M^l/2 \rfloor$ after the last row, $\lfloor (N^l - 1)/2 \rfloor$ columns before the first column and $\lfloor N^l/2 \rfloor$ after the last column have to be inserted.

For sake of brevity, (2.23) describes the output of a convolution layer with no padding and stride $(1, 1)$. The height and width of the output are then given by $I^l = I^{l-1} - M^l + 1$ and $J^l = J^{l-1} - N^l + 1$.

$$f_C(z^{l-1}(x))_{i^l,j^l,c^l} = \sum_{i=1}^{M^l} \sum_{j=1}^{N^l} \sum_{c=1}^{C^{l-1}} W^l_{i,j,c,c^l} z^{l-1}(x)_{i^l+i,j^l+j,c} + b^l_{c^l} \tag{2.23}$$

(2.23) is applied for each spatial location $i^l = 1, \ldots, I^l$, $j^l = 1, \ldots, J^l$ and every channel $c^l = 1, \ldots, C^l$. One output element is the weighted sum over all the elements in a location specific input volume of shape (M^l, N^l, C^{l-1}) plus a bias term. The convolution kernels are then a 4th-order tensor $W^l \in \mathbb{R}^{M^l \times N^l \times C^{l-1} \times C^l}$ and the bias $b^l \in \mathbb{R}^{C^l}$ is a vector containing one element per feature. The weights and biases are the trainable parameters $\theta^l = \{W^l, b^l\}$ of the layer.

Compared to the $(I^{l-1} J^{l-1} C^{l-1} + 1) C^l I^l J^l$ trainable parameters of a fully connected layer, the utilization of convolution reduces these number to $(M^l N^l C^{l-1} + 1) C^l$, and most important, the number of parameters is independent of the spatial dimensions of the in- and output. An interesting insight is it, that a convolutional layer can therefore be trained on a comparable small spatial input and later evaluated on a much larger input.

The 1D convolutional layers in Figure 2.5 are special cases of (2.23). For instance, layer 3 would have an input shape $(3, 1, 1)$, kernel size $(3, 1)$ and 1 feature. It can also be noted that (2.23) can always be reshaped in such a way that the in- and output are vectors (Balestriero and Baraniuk 2017).

The special case of an 1×1 convolution layer (Lin et al. 2013) is often used for adapting the depth (number of features) of the output, or in other words, the features are linearly combined without changing the spatial information. Depth is also the primary parameter to adjust the complexity of the network, higher depth means more representational capacity. The 1×1 convolution can also be used for feature fusion (Lin et al. 2016), when the feature maps originate from different sources, and it plays an important role in network compression (Kim et al. 2015). It should also

be mentioned, that a convolutional layer with kernel size equals the input size can be interpreted as a fully connected layer in the depth dimension (Long et al. 2014). The concept of local connections and weights sharing allow Deep Convolutional Neural Networks to efficiently learn rich feature hierarchies, but the idea of convolution is also a kind of prior knowledge that is introduced in the network topology and assume that the patterns in the data also have a hierarchical structure. Normal CNNs only provide translational invariance[6] and are therefore notoriously weak when it comes to handling other types of variances, like scale or rotation.
CNNs are biologically inspired by the experiments of Hubel and Wiesel 1962, 1968, in which they found that cells in the mammalian visual cortex are sensitive to small sub-regions in the visual field, called the receptive field. These cells tile the input with sub-regions and function as local filters that can exploit spatial correlations. LeCun et al. 1998; LeCun et al. 1990 firstly introduced a CNN that mimics this biological structure and applied it successfully to handwritten character recognition. More recent work (Bronstein et al. 2017) extends the concept of CNN to non-euclidean data like graphs and manifolds and Deformable Convolution (Dai et al. 2017) is a promising approach that enables CNN to learn geometric transformations. For the basic theory on CNNs, Dumoulin and Visin 2016 provides a detailed guide on convolution arithmetic and the lecture slides from *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition* 2018 may be considered as a compact introduction.

2.3.4 Pooling

Pooling is common practice in signal processing, where it is often termed sub- or down-sampling. A Pooling Layer is similar to a convolutional layer and normally applied after one or more convolutional layers. It has local connections and the hyperparameters, kernel size, padding and stride are the same as for convolutional layer, but instead of learning weights, a fixed pooling policy is used to calculate the layer output. The most commonly used pooling policies are Max-Pooling and Average-Pooling (illustrated in Figure 2.6).
For the mathematical description of these two cases, the same notation as in Section 2.3.3 is adopted, (M^l, N^l) is the kernel size, (s, s) the stride and the output size (I^l, J^l, C^l) is then given with $I^l = \lfloor (I^{l-1} - M^l)/s \rfloor + 1$, $J^l = \lfloor (J^{l-1} - N^l)/s \rfloor + 1$ and $C^l = C^{l-1}$. (2.24)/(2.25) is then applied for each spatial location $i^l = 1, \ldots, I^l$, $j^l = 1, \ldots, J^l$ and every channel $c^l = 1, \ldots, C^l$.

[6] Translational invariance in this context actually means equivariance, but it is a commonly used term. Translation invariance actually means that the system produces exactly the same response, regardless of how its input is shifted. Equivariance actually means that the system works equally well across positions, but its response are shifts with the position.

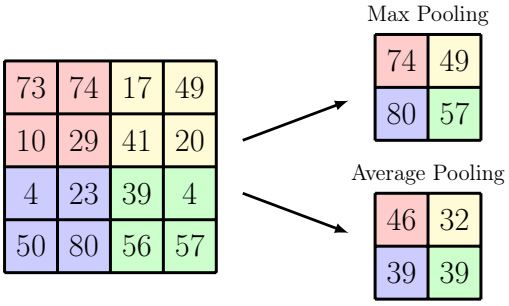


Fig. 2.6: Pooling policies with kernel size $(2, 2)$ and stride $(2, 2)$, input size $(4, 4)$

Max-Pooling layer takes only the highest activations from input and so extracts only the most important features.

$$f_P(z^{l-1}(x))_{i^l,j^l,c^l} = \max_{\substack{i=1,\dots,M^l \\ j=1,\dots,N^l}} z^{l-1}(x)_{i^l s+i, j^l s+j, c^l} \tag{2.24}$$

Average-Pooling or Mean-Pooling layer averages the activations of the previous layer and so extracts features more smoothly.

$$f_P(z^{l-1}(x))_{i^l,j^l,c^l} = \frac{1}{M^l N^l} \sum_{i=1}^{M^l} \sum_{j=1}^{N^l} z^{l-1}(x)_{i^l s+i, j^l s+j, c^l} \tag{2.25}$$

Because average-pooling takes all features into account, whether they are important or not, it is less often used for tasks such as classification and object detection. Pooling layers exploit, that the exact location of a feature is not as important as its relative location to other features. Their usage is beneficial in tree ways, first, they reduce the spatial dimension and lead to less computation in the upcoming layers, second, they reduce information redundancy present in the input data, and third, they make the system invariant to small translations. The pooling operations in Figure 2.6 reduce the number of parameters by 75%.

2.3.5 Softmax

For problems like multi-class classification or action prediction in reinforcement learning[7], it makes sense to have a network output that can be interpreted as a probability distribution. This is where the Softmax Layer comes into play. The softmax function is a mapping $\mathbb{R}^{D^l} \to [0, 1]^{D^l}$, where the output values sum up to 1. The softmax layer (2.26) is defined by the softmax function and computes the

[7]Reinforcement learning builds on the idea of an software agent that performs actions in an environment and has the objective to maximize an accumulated long-term reward.

normalized exponential function for all the units in the layer.

$$f^l(z^{l-1}(x))_d = \frac{e^{z_d^{l-1}(x)}}{\sum_{i=1}^{D^l} e^{z_i^{l-1}(x)}} \tag{2.26}$$

The output dimension of the softmax layer is equivalent to its input dimension ($D^l = D^{l-1}$). The subscript $d = 1, \ldots, D^l$ indicates the d-th unit in the layer.

2.3.6 Dropout

The use of dropout (Srivastava et al. 2014) provides an effective way to conquer the overfitting of large networks. The key idea behind dropout is to randomly disconnect (drop) units while training, this prevents the units from too much co-adaption and leads to a fast to evaluate ensemble of many smaller models at test time. A mathematical formulation of a dropout layer is straightforward.

$$f_D^l(z^{l-1}(x)) = r^l \odot z^{l-1}(x), \qquad r_d^l \sim \text{Bernoulli}(p) \tag{2.27}$$

Where the vector r^l contains D^l independent Bernoulli variables, each with probability p. They are sampled at training time and set to $\mathbb{1}$ at test time. The variable p, as a hyperparameter, is also known as the dropout rate or ratio and has a typical value between 0.5 and 0.8. Higher dropout rate reduces overfitting, but also increases the convergence time.

2.3.7 Batch Normalization

When training a Deep Neural Network, the input distribution of each layer changes at each iteration. This phenomenon is known as internal covariance shift (Ioffe and Szegedy 2015). It slows down training by requiring small learning rates and carefully chosen initialization parameters to keep the nonlinearities out of saturation. Batch Normalization (BN) solves this issue by normalizing the layer input at each iteration (per batch). It works by shifting the layer input to zero-mean and scaling it to the unit variance.
BN layers are typically applied between linear layers and their nonlinear activations. Hence, their output dimension is equal to their input dimension ($D^l = D^{l-1}$).
The formal description of a BN layer is given in Algorithm 2.2. The parameters $\gamma \in \mathbb{R}^{D^l}$ and $\beta \in \mathbb{R}^{D^l}$ are trainable and learned during training. The constant $\epsilon \approx$ 1e-5 is introduced for numerical reasons and avoids dividing by zero.

for each iteration
  do forward propagation of M samples and collect the output of layer $l-1$ $\mathcal{B}^{l-1} = \{Z_1^{l-1}, \ldots, Z_M^{l-1}\}$
  for d **in** $1, \ldots, D^l$
    $\mu_B = \frac{1}{M} \sum_{m=1}^{M} Z_{m,d}^{l-1}$
    $\sigma_B^2 = \frac{1}{M} \sum_{m=1}^{M} (Z_{m,d}^{l-1} - \mu_B)^2$
    $\hat{z}_d^{l-1} = \frac{z_d^{l-1} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
    $z_d^l = \gamma_d \hat{z}_d^{l-1} + \beta_d$
  end for
  update exponential moving average of mean and variance
  do back propagation
end for

Algorithm 2.2: Batch Normalization

A per batch based normalization is not desirable during inference and so the mean and variance over all N training samples can be used, but it is much easier to calculate moving averages during training and used them instead.
BN became quickly popular, because it increases convergence speed and makes the training of deeper networks easier. It also comes with some regularization effect and makes the use of Dropout less important, but they both work well in combination and usually do not require additional L2 regularization.

2.3.8 Skip Connections

Skip connections can be considered as a bypass connection between a layer and its output. Given a layer $f_{\theta^l}^l$ and the input $z^{l-1}(x)$, the skip-connection in form of a layer is formulated in (2.28).

$$f_S^l(z^{l-1}(x), f_{\theta^l}^l) = z^{l-1}(x) + f_{\theta^l}^l(z^{l-1}(x)) \tag{2.28}$$

During backpropation, skip connections provide a stronger error signal and therefore address the vanishing gradient problem. They allow the signals to bypass units or pathways that have learned something wrong and can be seen as a form of implicit regularization.

2.4 Recurrent Layer

When it comes to sequence data, like audio, video, text or DNA, classical FFNNs have the handicap, that they possess no persistent memory and hence can not handle

temporal patterns in a sequence of samples.
The first idea that comes in mind to address this problem is to take the output of a network from the previous time step and feed it additionally into the network at the current time step. The resulting network structures are called Recurrent Neural Networks (RNNs) and known since the 80s (Elman 1990; Rumelhart et al. 1986; Werbos 1988).
In general, a RNNs can be described in form of a function $h_t = F(h_{t-1}, x_t, \theta)$. Where x denotes the input vector, h_t the output vector at time step t and θ the trainable parameters.
Training of RNNs can be done via Backpropagation Through Time (BPTT), which is done in practice by unfolding the network (Figure 2.7) and subsequently training the unfolded network as it would be done for a normal FFNN, but with the except, that the entire epoch has to be fed into the unfolded layer. In order to keep training feasible, usually only a limited number of time steps is considered during the training process, what is known as truncated Backpropagation Through Time (BPTT) and typically done per batch.

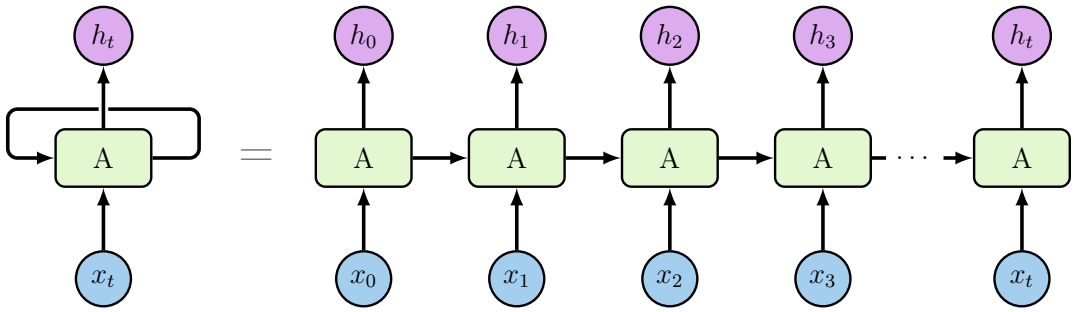


Fig. 2.7: Unfolding of a one layer Recurrent Neural Network[8]

To naturally integrate this idea into the framework described in Section 2.3, it is now formulated in form of recurrent units in recurrent layers.
The rest of this section describes the basic recurrent unit or cell (RNN) and some more advanced recurrent cells (LSTM, GRU) in detail.

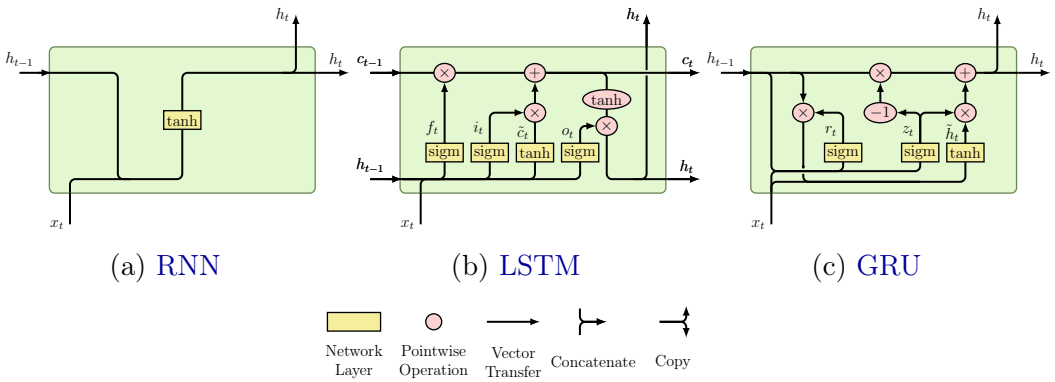
2.4.1 RNN

The basic recurrent cell is depicted in Figure 2.8a and formally described with (2.29)[9]. Where h_t is the so called hidden state, the signal that is recursively fed back at the next time step. The output of the RNN cell is simply the hidden state.

$$h_t = \tanh(W[x_t, h_{t-1}] + b) \tag{2.29}$$

[8]Figures heavily inspired by *Understanding LSTM Networks – colah's blog* 2017
[9]The square brackets mean the concatenation of the vectors inside.

Even though RNNs are theoretically able to learn long-term dependencies, they seem to bee incapable to learn them in practice. This may be no surprise since by unfolding (Figure 2.7) a RNN shows up as a DNN in time suffering from the vanishing gradient problem as normal FFNNs.



(a) RNN (b) LSTM (c) GRU

Fig. 2.8: Popular Recurrent Layers [10]

2.4.2 LSTM

To enable RNNs to learn long term dependencies, Hochreiter and Schmidhuber 1997 introduced the Long Short-Term Memory (LSTM) cell, as shown in Figure 2.8b. The basic idea behind LSTM is to have a cell state that is passed from one time step to the other and some trainable gates, that add and remove information from the cell state. A gate is actually a linear layer with a sigmoid activation function and an element-wise multiplication with a signal. Since the value of the sigm function is always in the range between $[0, 1]$, the signal is completely suppressed for a value of 0 or can pass without any modification for a value of 1. In the original LSTM, there are three gates, the input i_t, the output o_t and the forget gate f_t. (2.30) represents the description of a LSTM cell, where x_t is the input, h_t is the output and c_t the cell state.

$$\begin{aligned} f_t &= \sigma(W_f[x_t, h_{t-1}] + b_f) \\ i_t &= \sigma(W_i[x_t, h_{t-1}] + b_i) \\ o_t &= \sigma(W_o[x_t, h_{t-1}] + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_c[x_t, h_{t-1}] + b_c) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{2.30}$$

The name LSTM comes because, besides the actual input, both the long-term state and the output from the previous time step are fed into the cell. There exist a bunch of LSTM variants, for instance, Gers and Schmidhuber 2000 added peephole connections to let the gates access the cell state. (Greff et al. 2015) compared the most

[10] Figures heavily inspired by *Understanding LSTM Networks – colah's blog* 2017

popular variants of LSTM and found that they all perform with comparable results. Jozefowicz et al. 2015 performed an extensive evolutionary architecture search, compared over thousand different recurrent cells and found some that perform better on certain tasks. LSTM in general does not solve the vanishing gradient problem, it circumvents it by adding the cell state as a memory element.

2.4.3 GRU

Gated Recurrent Unit (GRU) (Cho et al. 2014) is the most popular variant of LSTM. It simplifies the LSTM cell by combining the forget and input gates into a single update gate and merging the cell and hidden state into a single signal. This results in fewer trainable parameters per cell, with same or even better performance compared to the LSTM cell. The description of a GRU cell is given in (2.31), z_t is the update gate and r_t is called the reset gate.

$$\begin{aligned} z_t &= \sigma(W_z[x_t, h_{t-1}]) \\ r_t &= \sigma(W_r[x_t, h_{t-1}]) \\ \tilde{h}_t &= \tanh(W[x_t, r_t \odot h_{t-1}]) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \end{aligned} \tag{2.31}$$

2.4.4 Bidirectional RNN

LSTM can only preserve information from the past, not from the future. To also consider information from the future, the sequence can be processed by two separate recurrent layers, one does this from the beginning to the end of the sequence and the other does it in reverse order, from the end to the beginning. Therefore, this concept is called bidirectional RNN/LSTM/GRU (Graves and Schmidhuber 2005; Schuster and Paliwal 1997). The hidden state of the two layers is then merged by either addition, multiplication or most often by concatenation. The second layer provides more context and typically leads to better results, but it also doubles the number of trainable parameters.
Recurrent layers have been extended to multiple dimensions (Graves et al. 2007) and can be used for hierarchical subsampling (Graves 2012, Chapter 9). LSTM has been combined with convolution (Bai et al. 2018). In context of system identification and control theory, the concept of LSTM has also been extended to learn probabilistic State Space Models (Doerr et al. 2018) and inspired more sophisticated mechanisms for memory management (Graves et al. 2016).
Recent experiments by Bai et al. 2018 show that relatively simple CNN architectures outperform LSTM on a diverse range of tasks.

2.5 Loss Functions

As mentioned in the previous section, an error measure called the loss function is required to define the training objective. The loss in general is a mapping $\mathcal{L} : \mathbb{R}^{N\times C} \times \mathbb{R}^{N\times C} \to \mathbb{R}$ which maps the network outputs $\hat{y}$ and corresponding labels y of all N samples to a real value.
To avoid the confusion in advance, the notation here considers the case of a full batch update, if SGD is used N has to be replaced by the batch size M.
The requirements on a loss function are, that it has to be differentiable, so Gradient Descent (GD) can be performed, and it should be convex with respect to the network output, so it does not make the problem non-convex if the network itself is convex[11].
Depending on the application, a loss is divided by N.
The rest of this section introduces some basic loss functions for classification and regression, as well as a more advanced concept for sequential data called the Connectionist Temporal Classification (CTC).

2.5.1 L2 Loss

The L2 Loss

$$\mathcal{L} = \sum_{i=1}^{N} (y_n - \hat{y}_n)^2 \tag{2.32}$$

is the square of the L2 Norm of the difference between actual and desired value. It is the Mean Square Error (MSE) not divided by N, has nice properties regarding gradient calculation and is mostly used for linear regression.
Further, it can be shown, that minimizing MSE is equivalent to minimizing the negative log-likelihood, or more general, the cross entropy between the empirical data distribution and a Gaussian model (Goodfellow et al. 2016, Section 5.5.1).

2.5.2 L1 Loss

The L1 Loss

$$\mathcal{L} = \sum_{n=1}^{N} |y_n - \hat{y}_n| \tag{2.33}$$

is the sum of absolute values of the difference between actual and desired value. It is the Mean Average Error (MAE) not divided by N and, compared to the L2 Loss, less prone to outliers. Regarding gradient calculation, the L1 Loss has the disadvantage that it is not a smooth function of the parameters.

[11] The composition of two convex function is also convex. In this case the concatenation of the network and the loss function is convex if they are both convex.

2.5.3 Smooth L1 Loss

The Smooth L1 Loss (Girshick 2015)

$$\mathcal{L} = \sum_{n=1}^{N} \text{smooth}_{L1}(y_n - \hat{y}_n), \qquad \text{smooth}_{L1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| \leq 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{2.34}$$

is the combination of L1 and L2 Loss, which is differentiable at zero and less sensitive to outliers.
Further, the Smooth L1 Loss is a special case of the Huber Loss (Huber 1964).

2.5.4 Hinge Loss

The Hinge Loss is used for maximum-margin classification (Bishop 2006, Section 7.1), most notably for Support Vector Machines (SVMs). It is defined by

$$\mathcal{L} = \max(0, 1 - y\hat{y}) \tag{2.35}$$

Where $y \in \pm 1$ is the target class and $\hat{y} \in \mathbb{R}$ is the classifier output (not the predicted class label). It can be seen in (2.35) that when $\hat{y}$ and y have the same class, meaning that the prediction is correct and $|\hat{y}| \geq 1$, the Hinge Loss is 0, but when they have opposite sign it increases linearly with $\hat{y}$ (one-sided error).
In context of our NN framework, a linear classifier in form of a SVM consists of a FC layer with linear activation that is trained with the Hinge Loss.

2.5.5 Cross Entropy Loss

For solving various classification problems, a family of appropriate loss functions has been formulated. They share in common, that they try to maximize the likelihood of the correct class. Maximum Likelihood Estimation (MLE) essentially boils down to maximizing the Cross Entropy (CE) between the empirical data distribution of the labels p and the predicted model distribution q (Bishop 2006, Section 5.2). This is why they are called Cross Entropy Loss or sometimes, due its logarithmic structure, also Log Loss.
Further, minimizing the CE is also equivalent to minimizing the Kullback Leibler divergence (KL divergence) between p and q[12].
Consider the case of a binary classifier with labels $y_n \in \{0, 1\}$ and a sigmoid output layer containing one single unit. In this case, the output of the network can be

[12]Given the relation between CE and KL divergence, $H(p, q) = H(p) + D_{KL}(p, q)$, and since the data distribution is independent of the model parameter, $H(p)$ is just an irrelevant constant in view of optimization.

interpreted as conditional probability $\mathrm{P}(y = 0|x) = y(x, \theta)$, $P(y = 1|x) = 1 - y(x, \theta)$ and formulated as Bernoulli distribution

$$\mathrm{P}(y|x, \theta) = \hat{y}(x, \theta)^y (1 - \hat{y}(x, \theta))^{1-y} \tag{2.36}$$

The CE Loss is then given by the negative log likelihood of this Bernoulli distribution

$$\mathcal{L} = -\sum_{n=1}^{N} (y_n \log(\hat{y}_n) + (1 - y_n) \log(1 - \hat{y_n})) \tag{2.37}$$

Since y can only be 0 or 1, (2.37) can also be written as

$$\mathcal{L} = -\sum_{n=1}^{N} \log(p_n), \qquad p_n = \begin{cases} \hat{y}_n, & \text{if } y_n = 1 \\ 1 - \hat{y}_n, & \text{otherwise} \end{cases} \tag{2.38}$$

In the case of K separate binary classifiers with labels $y_{nk} \in \{0, 1\}$ and a sigmoid output layer containing K units, (2.37)[13] extends to

$$\mathcal{L} = -\sum_{n=1}^{N} \sum_{k=1}^{K} (y_{nk} \log(\hat{y}_{nk}) + (1 - y_{nk}) \log(1 - \hat{y}_{nk})) \tag{2.39}$$

and is usually named Binary Cross Entropy Loss or Logistic Regression Loss.
In case of a multi-class classifier, where we have K mutually excluding classes, the labels $y_{nk} \in \{0, 1\}$ have a one-hot coding scheme (1-of-K) and a softmax output layer with K units. In this case the network output can be interpreted as $\mathrm{P}(y_k = 1|x) = y_k(x, \theta)$ and formulated as a multinomial distribution with one observation. The resulting expression for the CE Loss is given by

$$\mathcal{L} = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \log(\hat{y}_{nk}) \tag{2.40}$$

Other common names for this version of the CE Loss are Categorical Cross Entropy Loss, Multinomial Logistic Regression Loss or Softmax Loss.

2.5.6 Focal Loss

In terms of (2.38), the Focal Loss (Lin et al. 2017), here for the binary case,

$$\mathcal{L} = -\sum_{n=1}^{N} \alpha_n (1 - p_n)^\gamma \log(p_n) \tag{2.41}$$

[13] y_{nk} is abbreviated and denotes $y_k(x_n, \theta)$

is a dynamically scaled version of the Cross Entropy Loss. It was developed in context of object detection to address the large imbalance between the classes (e.g. the background class may appear 100 or 10000 times more often than the object class). The idea is to weight down easy samples and focus more on hard samples, whereby the hard samples can be interpreted as the outliers. The focusing parameter $\gamma \geq 1$ is a tunable hyperparameter, but a value of 2 seems to work fine in most cases. α_n is a class specific weighting factor that is set to $\alpha \in [0, 1]$ for the first and $1 - \alpha$ for the second class. In practice α may be set to the inverse of the class frequency or gets chosen by cross validation.

2.5.7 CTC Loss

The Connectionist Temporal Classification (CTC) is an approach to solve the Dynamic Time Warping (DTW) problem, that occurs when a RNN has to map a sequence with unsegmented input data of length T to a label sequence of variable length $U \leq T$ (e.g. for speech or handwritten text recognition). It was first introduced by Graves et al. 2006 and, to make the reference clear, prior approaches to this problem were often based on HMMs (Rabiner 1989) or combinations of a HMM and a NN.
For the formal description of the CTC, consider a set of labels L (e.g. all 26 English alphanumeric characters). Then an additional label, the 'blank', is introduced to indicate the boundaries for segmenting the sequence and representing the case when no label is present. The extended alphabet is then given by $L' = L \cup \{\text{blank}\}$ and the network output for each of the T time-steps can be chosen as a softmax distribution (Section 2.3.5) over $K = |L'|$ classes. An element of the output y_k^t is then the probability of observing label k at time-step t. The complete output y is then a distribution over the set L'^T, representing the probability of all possible ways of aligning all possible label sequences with the input sequence. To distinguish them from the label sequence l, the elements of L'^T are now referred to as paths π.
With the assumption, that the output probabilities at different time-steps are conditionally independent, the conditional distribution over $\pi \in L'^T$ is given by

$$\mathrm{P}(\pi|y) = \prod_{t=1}^{T} y_{\pi^t}^t \tag{2.42}$$

The next step is to define a many-to-many mapping $\mathcal{B} : L'^T \to L^{\leq T}$ that transform a path into a label sequence. Therefore, $\mathcal{B}$ merges repeated continuous labels to a single one and removes the 'blank' labels (e.g. "__aa_b_c_cc_dd_" to "abccd", where '_' represents the blank). The conditional probability of a given labeling $l \in L^{\leq T}$ is then given by the sum of all paths mapped into it (marginalize over the

set of all possible alignments).

$$\mathrm{P}(l|y) = \sum_{\pi \in \mathcal{B}^{-1}(l)} \mathrm{P}(\pi|y) \tag{2.43}$$

For training the mode, the negative log likelihood loss can be defined by

$$\mathcal{L} = -\ln \prod_{n=1}^{N} \mathrm{P}(l_n|\hat{y}_n(x,\theta)) = -\sum_{n=1}^{N} \ln \mathrm{P}(l_n|\hat{y}_n(x,\theta)) \tag{2.44}$$

The problem now is that the number of possible paths grows exponentially with the sequence length T, making the calculation of the conditional probability in (2.43) practically infeasible. Fortunately, this problem can be solved with dynamic programming[14]. The algorithm to accomplish this is known as the forward-backward algorithm from the theory of HMMs, where it is used for inference.
After applying the forward-backward algorithm, as described in Graves 2012, Section 7.3, the loss function (2.44) can be formulated by

$$\mathcal{L} = -\sum_{n=1}^{N} \ln \sum_{u=1}^{U'_n} \alpha(t,u)\beta(t,u) \tag{2.45}$$

where $\alpha(t,u)$ and $\beta(t,u)$ are the forward and backward variables of the forward-backward algorithm at time-step t. To allow blanks in the paths, blanks were added at the beginning, end and between the label sequence, resulting a modified label sequence l' with length $U' = 2U + 1$. The CTC-Loss (2.45) is differentiable with respect to the network parameters and used for training the network with BPTT (see Graves 2012, Section 7.4).
In the inference case, the easiest way to get the final label sequence $\hat{l}$ is by best path decoding (Graves et al. 2006), which assumes that the most probable path π^* corresponds to the most probable class at each time-step.

$$\pi^* = \underset{\pi \in L'^T}{\operatorname{argmax}}\, \mathrm{P}(\pi|\hat{y}) \tag{2.46}$$

$$\hat{l} = \mathcal{B}(\pi^*) \tag{2.47}$$

However best path decoding does not guarantee to find the optimal output sequence and more advanced decoding methods such as Prefix Search Decoding or Constrained Decoding via beam search and a language model (Graves 2012, Section 7.5) were developed.
Even if the name of the CTC indicates its special design for temporal sequence classification tasks, it does not necessarily require a RNN architecture and can also

[14]The trick of setting up a recursion reduces the time complexity from $\mathcal{O}(TL'^T)$ to $\mathcal{O}(TL'^2)$.

be used with other architectures networks such a CNNs (Gao et al. 2017). The collapsing of the label sequence allows the CTC to handle unsegmented sequences, but it loses the information about the exact location where the label appeared in the input sequence, and thus may be unsuitable for certain tasks. A nice overview on the CTC and its relation to HMMs and Encoder-Decoder Models can be found in Hannun 2017.

2.5.8 Multi-Task Loss

Sometimes it may be desirable to fulfill multiple objectives at the same time. The easiest way to accomplish this and tradeoff between the objectives is a linear combination of multiple loss functions, as described in (2.48).

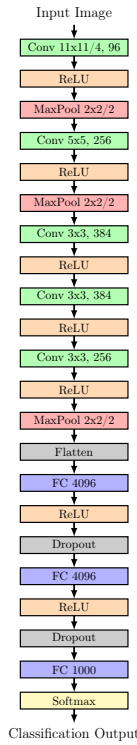
$$\mathcal{L} = \lambda_1 \mathcal{L}_1 + \lambda_2 \mathcal{L}_2 + \dots \tag{2.48}$$

The constants $\lambda_1, \lambda_2, \dots$ are hyperparameters to control the balance among the different loss terms. It is also common that this type of multi-task loss appears in form of a convex combination, where $\lambda_i \geq 0$ and $\sum \lambda_i = 1$.

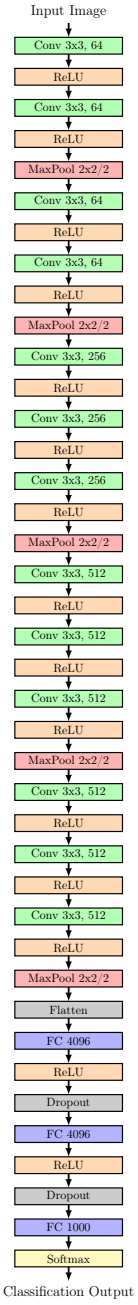
2.6 Network Architectures

Depth, in sense of the number of layers in a FFNNs, is essential for learning rich feature hierarchies and making the computation of the output compositional[15].
This insight, combined with parallel computing on consumer GPUs and various efficient training techniques (Section 2.2) led to the development of increasingly deeper network architectures and a major success in almost all areas of machine learning. Especially the application of Deep Convolutional Neural Network (DCNN) caused rapid progress in the field of computer vision as well as a huge "Deep-Learning Hype" in the tech industry.
The aim of this section is to introduce some historically important and often referenced network architectures and their underlying ideas. A more quantitative comparison of different architectures can be found in Canziani et al. 2016.
In general, the traditional feed forward convolution architecture for classification is a stack of some `Conv->ReLU->Conv->ReLU->MaxPool->` blocks, followed by some `FC->ReLU->` and a final `FC->Softmax->`. Without the softmax block at the end, this basic architecture can also be used for regression tasks. The part after the convolutional blocks is often referred to as classification or regression head.

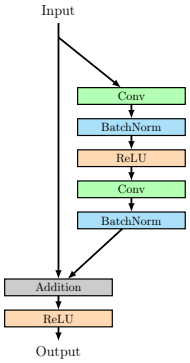
[15] Even though the universal approximation theorem (Cybenko 1989; Hornik 1991) states, in theory, that a single hidden layer with enough units can approximate any possible function with arbitrary small error.



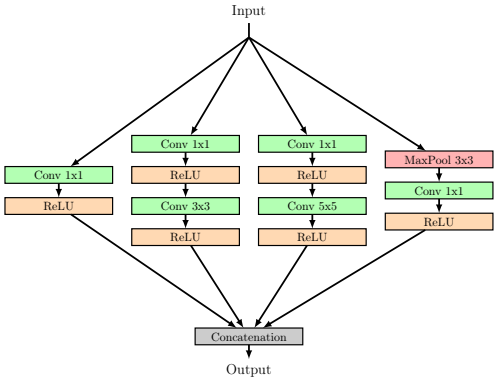
(a) AlexNet



(b) VGG-16



(c) Residual Block



(d) Inception Block

Fig. 2.9: Network Architectures

LeNet (LeCun et al. 1998) was the first successful application of CNN in the 90s. The architecture used two convolutional layers each followed by an average pooling layer and two final FC layers. The activation functions were sigmoid and tanh. RBFs were used to produce an error measure. The model was trained on the MNIST[16] dataset and used for reading zip codes, handwritten digits, etc.

[16]MNIST dataset of handwritten digits (60k for training, 10k for testing) http://yann.lecun.com/exdb/mnist

AlexNet (Krizhevsky et al. 2012) was the first deep architecture. It received much attention after winning the ILSVRC challenge 2012 with a large margin (top 5 error of 16% compared to the runner-up with 26%). The architecture of AlexNet (shown in Figure 2.9a) was similar to LeNet, but used ReLU as activations, Dropout for regularization and also the input sizes (227x227) as well as the learned features were much larger. AlexNet was also the first architecture that stacked more than one convolutional layer directly on top of each other. The implementation was done for GPU, what came with an enormous training speedup (compared to CPU). The original architecture was split into two sub-networks, which allowed the model to be trained on 2 GPUs and reduced the top-1 and top-5 error rate, compared to the 1 GPU version with the half of the parameters, by further 1.7 respectively 1.2 percent points. The model was trained with data augmentation on the ImageNet dataset for 5 to 6 days on two NVIDIA GTX 580 3GB GPUs.

ZFNet (Zeiler and Fergus 2013b) was the winner of ILSVRC challenge 2013. ZFNet (short for Zeiler & Fergus Net) improvement AlexNet by tweaking some hyperparameters, in particular by reducing the filter size and stride of the first layer from 11x11/4 to 7x7/2 and increased the number of filters in the last three convolutional layers from 384, 384, 256 to 512, 1024, 512. This reduced the number of trainable parameters and improved the overall recognition accuracy.

VGGNet (Simonyan and Zisserman 2014) was introduced by Visual Graphics Group at Oxford, hence VGG. It was the runner-up in the ILSVRC challenge 2014 and showed that dept is critical for performance. The architecture is quite straightforward and contains only blocks of 3x3 convolution and 2x2 max pooling layers followed by a classification head. VGGNet models are in general hard to train and take relatively long to evaluate due to the many parameters (140M). Most of them are in the first FC layer, but it was later found out that the layer can be removed without loosing the performance of the architecture. In the original paper, the authors made experiments with different depths, VGG-16 (Figure 2.9b) for instance contains 16 convolutional and FC layers. They used pre-trained parameters of shallower variants to initialize the deeper ones. VGG-16/19 models trained on ImageNet are still often used for transfer learning.

Inception (Szegedy et al. 2014) is a family of architectures with similar structure, starting with the 22-layer GoogLeNet as winner of the ILSVRC challenge 2014. The biggest innovation in the Inception architecture was the introduction of Inception Blocks, as shown in Figure 2.9d. The idea behind Inception Blocks is to do multiple convolutions with different kernel sizes in parallel and concatenate their outputs. To make that work properly, the convolutional and pooling layers must have padding

same and 1x1 convolutions are required to keep the number of features and the associated computation cost low. The architecture itself is basically a stack of inception blocks and max pooling layers, where some topmost layers have their own classification heads that contribute additional error signals and help the model to converge faster. Also, average pooling before the FC layers in the head seem not to affect the performance. All these changes together reduce the number of parameters in the network dramatically (from 60M in AlexNet to 4M) and, compared to VGGNet, make the model much easier to train. Later publications (Szegedy et al. 2016, 2015) improved the architecture and developed several variants of the original inception block.

ResNet (He et al. 2015a) was the winner of ILSVRC competition 2015. It utilizes skip connections in form of Residual Blocks, shown in Figure 2.9c, and make heavy use of batch normalization. The whole architecture is a stack of residual blocks with appropriate stride and pooling. The 152-layer ResNet architecture has only one FC layer at the end and lower complexity then VGG-16/19. Its name, ResNet, comes from the original idea, that it is hard to learn a mapping directly, so take the identity (skip connection) and only try to learn the "residual". Later, the authors studied variants of the original residual block (He et al. 2016) and successfully trained a 1001-layer network on CIFAR-10 (4.62% error rate). ResNet models trained on ImageNet are the default choice for transfer learning today.

DenseNet (Huang et al. 2016a) is a feed forward convolutional architecture that excessively makes use of skip connections. Therefore, Dense Blocks were introduced as shown in Figure 5.2 (gray box). They contain multiple `Conv->BN->ReLU->` blocks with all possible connections to skip them. 1x1 convolutions are used to keep the number of features feasible. The complexity of the blocks is primarily controlled by a single hyperparameter called the growth rate. More precisely, the growth rate is the increase in the number of features after each convolutional layer in a Dense Block. The skip connections allow the DenseNet models to reuse features from other layers and overcome the vanishing gradient problem and thus, the models can be deep, have few parameters and fast convergence. In comparison to ResNet on CIFAR-10, DenseNet showed better results with less parameters. The main disadvantage of DenseNet compared to other architectures is that it requires an enormous amount of memory, especially during training. A more detailed discussion of this problem is contained in the experiments Chapter 6.

Xception (Chollet 2016) stands for "extreme inception". It literally takes the concept of Inception Blocks to the extreme by introducing depth-wise separable convolutions. Xception showed slightly better results compared to Inception v3 and

the idea, that cross-channels correlations and spatial correlations in the feature maps of CNNs can be entirely decoupled, is now known as the "inception hypothesis". The concept of depth-wise separable convolution has been quickly adopted as basic building block in many other architectures (Howard et al. 2017; Iandola et al. 2016; Zhang et al. 2017).

MobileNet (Howard et al. 2017) builds on depth-wise separable convolutions as introduced in Xception, resulting in much smaller models with fewer parameters. The architecture is also designed to easily trade off between accuracy and computation cost, making it a suitable choice for applications on mobile devices. The MobileNet architecture has been further improved and was applied to object detection and semantic segmentation (Sandler et al. 2018).

2.7 Transfer Learning

The training of large models in a supervised fashion is time-consuming and requires a large amount of labeled data, which is often not available for real world applications. That's where transfer learning comes into play.
Its basic idea is to train a model on a task $\mathcal{T}_S$ for which a large training set $\mathcal{D}_S$ is available and then reuse the model for further training on a second task $\mathcal{T}_T$ for which only a much smaller training set $\mathcal{D}_T$ is available. The subscript S and T stand for source and target, thus the knowledge is transferred from the source to the target domain.
In the context of computer vision and DCNN, there are practically two common scenarios of how transfer learning is performed. They both start with a model that is trained on a large dataset $\mathcal{D}_S$ like ImageNet[17] (Russakovsky et al. 2014) for performing a classification task $\mathcal{T}_S$. This pre-trained model functions as feature extractor, which has learned a rich hierarchy of features, starting from low-level feature (e.g. edges or color blobs) in its first convolutional layer to high-level features (e.g. cat faces or cars) in the last convolutional layer, before FC and a final softmax layer are attached for classification. The next step is to remove the classification head (FC and softmax) and add a new one that should conduct classification or regression for $\mathcal{T}_T$.
The first scenario that can be applied is to keep the parameters of the convolutional layers fixed initialize the head randomly and thus train only the new head on $\mathcal{D}_T$. This can also be done by considering the feature extractor and the classifier/regressor (in the simplest case a SVM) as separate models and using the feature extractor as a static function.

[17] The ImageNet dataset contains more than 14M images and 1k categories.

The second scenario is fine-tuning. Therefore, the new head is also randomly initialized, but none or only the first convolutional layers are fixed. The training on $\mathcal{D}_T$ is then performed with a relatively small learning rate, allowing the network to adjust the feature maps to the new task without destroying too much of the pre-trained features. How well fine-tuning works, depends primarily on how similar the two tasks are and how well $\mathcal{D}_T$ represents the true data distribution. If $\mathcal{D}_T$ is too small and fine-tuning leads to large overfitting, then the first scenario is the better choice. Transfer learning in general is its own research area, that is closely related to Multi-Task, Few-Shot and Semi-Supervised Learning.

2.8 Interpretability

In general, a model in form of a function approximator trained via supervised learning does not guarantee that it reflects causal relationships and the training objective may be a weak surrogate for the real-world engineering goal.
The question that arises now is that of interpretability, one might want to see what is going on in a model and understand when and why it makes mistakes.
Unfortunately, there is no formal technical definition of interpretability and the topic involves ethical and philosophical questions (Lipton 2016). Often, however, a distinction is made between transparent and black-box models. Transparent models are typically simple models that can be simulated by a human brain, and black-box models are typically models that can not match or surpass our abilities on complex tasks.
ANNs are often considered as black-box models due to the fact that they are high-dimensional constructs that are not directly visually accessible to humans, which is not entirely the case, especially for CNNs with their hierarchical feature structure. Good visualization concepts have been devised, which will now be considered in more detail.
A few simple visualization techniques exist that can be applied to ANNs in general. For instance, computing the loss for all samples in the test set to see which are the easy ones and which are the hard ones, or recording the activation of a certain unit to see which samples it responds to. To get an idea of what a certain layer has learned, one may record the activation of the layer for all samples and apply k-nearest neighbors based on the proximity in the space learned by the model. Another method for high-dimensional data in general is t-SNE (Maaten and Hinton 2008), it visualizes high-dimensional data in a 2D plane while preserving some similarity metric.
Most visualization methods for CNNs either render gradients as an image (Simonyan et al. 2013), alter the input image via GD to strengthen the activation of a certain unit selected from a hidden layer (Mordvintsev et al. 2015) or invert the convolution

operation of the network (Bojarski et al. 2016; Dosovitskiy and Brox 2016; Zeiler and Fergus 2013b; Zeiler et al. 2010).
Directly visualizing feature maps may be interesting for small networks, but is unsuitable for large ones. However, attention mechanisms (Xu et al. 2015b) usually provide saliency maps that can be displayed to see on which parts of an image the model is focusing on.
Also, adversarial samples (Nguyen et al. 2015) reveal a lot about a model, especially its weaknesses.
Last but not least, theoretical approaches, based on the idea of information bottlenecks (Shwartz-Ziv and Tishby 2017) or local linearisation (Balestriero and Baraniuk 2017) may give valuable insights.

2.9 Deep Learning Frameworks

The success story of DNN goes along with the development of powerful open source DL frameworks. They are practically all written in C/C++ and utilize CUDA[18]. Most of them are available for the major OSs (i.e. Linux, MacOS, Window) and provide Python bindings to leverage the scientific Python stack with its libraries (e.g. NumPy, SciPy, Pandas, SymPy etc.). They all support convolutional layers as well as recurrent layers and implement a version of Automatic Differentiation (Baydin et al. 2015) to perform backpropagation. In this section, some popular and historically influential frameworks are presented, but it has to be mentioned that the list is far from being complete and permanently new ones are coming up.

Caffe[19] (Jia et al. 2014) is well-known and was one of the first DL libraries. It was developed from the Berkeley Vision and Learning Center (BVLC) as a faster C/C++ port of a Matlab based convolutional network implementation. Caffe's strengths were, that it came with implementations of common layer functions (e.g. Conv, FC...), out of the box GPU support and interfaces for Python and MATLAB. With the Caffe Model Zoo, many pre-trained models are available for fine-tuning, and it is not uncommon to port them to other frameworks. One of Caffe's downsides is that it has relatively many dependencies on other libraries and it is not always straightforward to setup. It also comes with rather poor documentation. Since, Caffe was intended for computer vision, it is not a general purpose framework and the implementation of new features have to be done in C++ and CUDA. It also lacks of auto-differentiation and one has to define the full forward, backward and gradient update procedure for a new layer. For new projects building on Caffe, it is common practice to fork the original source code and modify it according the needs,

[18] CUDA is Nvidia's general purpose computing platform and API for CUDA-enabled GPUs.
[19] `http://caffe.berkeleyvision.org`

resulting in more then 14k forks on GitHub. In 2017, Facebook released Caffe2[20] as successor of Caffe. It is intended for production environments and claims to be more scalable and light-weight.

Theano[21] (Bastien et al. 2012) was the first widely used and first with proper auto-differentiation. It is written in Python and more a low level library for symbolic computation. It can handle multidimensional arrays like NumPy and applies the concept of a computation graph, that has to be defined, gets optimized, compiled and evaluated as mathematical expressions. The compiler approach makes the models sometimes hard to debug and it has only single GPU support, but many other high level deep learning frameworks, such as Keras, Lasagne and Blocks build on top of it. In September 2017 Yoshua Bengio announced that the development of Theano will stop after version 1.0 and in favor of other frameworks. It was primary intended and used for research.

Torch[22] (Collobert et al. 2002) is like Theano, it has its origin in academia and gained a large open source community. It was originally started at IDIAP (Switzerland), continued at NEC Laboratories America, later adopted by New York University and DeepMind[23][24]. Today it is maintained by Google, Facebook and Twitter. The goal of Torch is it to be portable, fast, extensible and easy to use. Therefore it is written in C, uses cuDNN, Nvidia's library to optimize CUDA for ANN, and provides an interface to the scripting language Lua via LuaJIT. Torch supports neural networks as well as energy based models, can import Caffe models and was ported to embedded systems (iOS, Android and FPGA).

Tensorflow[25] (Abadi et al. 2016) is powered by Google and was the first DL library intended for engineering. It is similar to Theano and comes with bindings for Python, C++, Haskell, Java, Go, and Rust, but Python is by far the most used. The main focus of Tensorflow is on scalability and it supports multiple CPUs, GPUs and TPU, an ASIC developed by Google specific for ANNs. Like Theano, Tensorflow follows the compiler approach and has a static computation graph. Tensorflow uses cuDNN and multiple high level Tensorflow wrappers exits (TFlearn, Keras, TF-Slim, TensorLayer). To make model debugging easier, a suit of visualization tools, called TensorBoard, is available. Since version 1.6 Tensorflow is shipped with Tensorflow Lite, a more light-weight version of the library intended for performing

[20] https://caffe2.ai
[21] http://deeplearning.net/software/theano
[22] http://torch.ch
[23] https://github.com/torch/torch7/blob/master/COPYRIGHT.txt
[24] In 2014, DeepMind was acquired by Google.
[25] https://www.tensorflow.org

inference on embedded devices. Tensorflow is probably the framework with the largest community these today.

Deeplearning4j[26] was the first commercial oriented open source DL library and is supported by the startup Skymind. Deeplearning4j's goal is it to bring distributed deep learning to production environments. Therefore, Deeplearning4j was written for Java and other JVM languages. The JVM lets it run on various platforms and opens it to the Java ecosystem. It can also get integrated on top of Big Data tools such as Apache Hadoop and Apache Spark. In addition to neural networks, deep belief nets and many other types of models are available. The models from most major frameworks can be imported and a Python API is provided via Keras.

MXNet[27] is developed by the Apache Software Foundation and gets more and more popularity. It is probably the framework with the largest language-support (officially: C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl, Wolfram Language). It scales very well, works with multiple GPUs and suits for enterprise. Amazon has chosen MXNet as its reference library for DL. Amazon, Microsoft and Facebook developed the Open Neural Network Exchange (ONNX) format[28] for deep learning models, which got early adopted by MXNet. In addition to industry, MXNet also receives strong support from research institutes, such as MIT or the University of Washington. NXNet supports both imperative and symbolic programming, and it provides an efficient way to deploy trained models for inference on mobile (Android and iOS) and IoT devices.

Microsoft Cognitive Toolkit,[29] previously known as Computational Computational Network Toolkit (CNTK) is open source and used in various commercial Microsoft products such as Skype, Cortana, Bing and Xbox. It was first released in January 2016[30] and is available for Linux and Windows OSs. The focus of CNTK is on distributed, multi-GPU computation for DL and it is optimized for efficient resource utilization. Definition, training and evaluation of models can be done in Python, C++ or BrainScript, CNTK's own model description language. Inference can only also be done in C# and Java. Models for Reinforcement Learning or Generative Adversarial Networks (GANs) can be implemented relatively easy in CNTK.

[26] https://deeplearning4j.org
[27] https://mxnet.apache.org
[28] https://github.com/onnx/onnx
[29] https://www.microsoft.com/en-us/cognitive-toolkit
[30] https://github.com/Microsoft/CNTK
[31] https://keras.io

Keras[31] is a popular high level DL library for Python. It is primarily developed by François Chollet[32], to simplify the creation and evaluation of neural networks, what can be quiet complex when it is done with low level libraries. Keras acts as a wrapper around other DL libraries. Backends are currently available for Tensorflow, Theano, CNTK and MXNet. Keras provide basic building blocks such as layers, loss functions, optimizers and tools to make it easier to work with image and text data. It is lightweight, easy to use and comes with good documentation. With Tensorflow version 1.4, Keras has been integrated as official high level API in Tensorflow.

PyTorch[33] (Paszke et al. 2017) is Torch with a loose Python port of its Lua interface. As a framework PyTorch is fairly new and gets increasingly popular due to its concept of a dynamic computation graph. A dynamic computation graph is actually not a new idea (Chainer[34]), but compared to the compiler approach of Theano and Tensorflow, it can be seen as the JIT approach that allows imperative programing. Therefore, PyTorch has a similar API as NumPy, combined with AutoGrad and GPU acceleration. Tensors can be easily moved between CPU and GPU, and computation is performed instantly, making debugging much easier and PyTorch a good choice for research. It is developed by Facebook, but also used by many other tech companies such as Twitter and Nvidia, and Uber's probabilistic programming library Pyro[35] is build on top of PyTorch.

[32] François Chollet is also the creator of the Xception architecture.
[33] http://pytorch.org
[34] https://chainer.org
[35] http://pyro.ai

Chapter 3

Object Detection

Most state-of-the-art scene text recognition systems rely on efficient CNN-based approaches to generic object detection. Therefore, this section will give a brief overview on recent developments in object detection.

3.1 Terminology

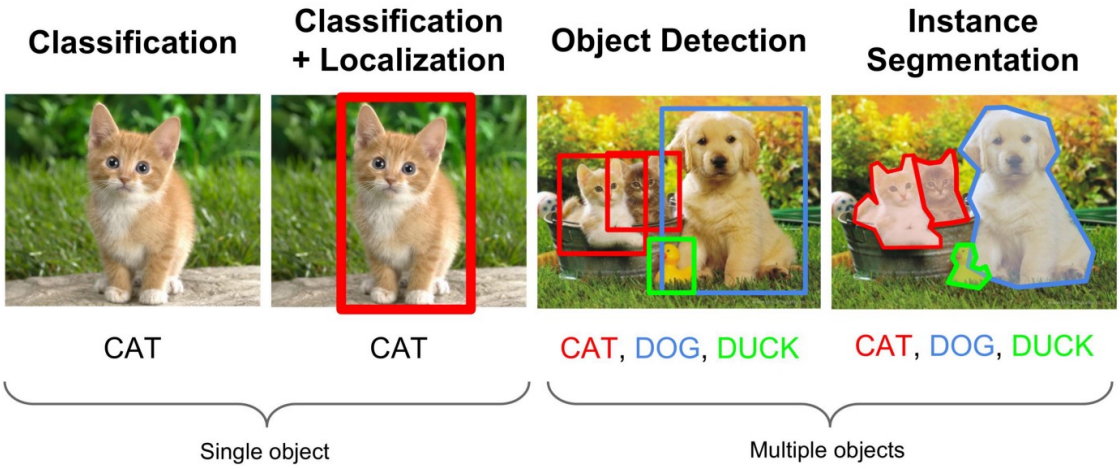


Fig. 3.1: Comparison between image classification, object detection and instance segmentation[1]

To clarify the basic terminology, *image classification* is the most basic task, where the model sees images, each contains one object, and has to output the correct one of several predefined classes (e.g. 'cat' or 'dog'). *Object localization* is about estimating the position of a object within an given image. The combination of classification and localization is called *object detection*. The fact that multiple instances of each class can appear in the same image makes detection a much harder problem compared to classification and localization on their own. *Segmentation* is the task of partitioning a image into several parts. Each pixel gets classified without understanding what the parts represent. For *semantic segmentation*, the image is partitioned into semantically meaningful parts and each part is classified. The problem with semantic

[1]Image source Ouaknine 2018

segmentation is that it can not distinguish among multiple objects of the same class. To solve this problem, it requires what is called *instance segmentation*.

3.2 Datasets for Object Detection

The supervised training of large CNNs for object detection requires a huge amount of labeled training data, but annotating large amounts of image data with bounding boxes or even masks for instance segmentation is a rater expensive undertaking. Private companies normally do not want to freely give away their investment, and universities do not have that many resources. Consequently, training datasets for object detection were always rare.
This section gives an brief overview on freely available and often used datasets.

Datasets	**Year**	**Classes**	**Images**	**Objects/Image**
Pascal VOC	2007	20	10k	2.5
Pascal VOC	2012	20	12k	2.4
Oxford-IIIT Pet	2012	37	7k	1.0
ImageNet Detection	2014	200	450k	1.1
MS-COCO	2014	80	120k	7.2
KITTI Vision	2014	3	7k	

Tab. 3.1: Datasets for object detection, the number of images refer to the training plus validation set

The Pascal VOC dataset (Everingham et al. 2010)[2] was created for the VOC (Visual Object Classes) challenges and is still one of the most often used datasets for training and benchmarking. It contains 20 different object classes of persons, animals, vehicle and everyday objects. The version released in 2012 contains 11,530 images for training and validation, including all images from the 2007 version. All images come with bounding box annotations and segmentation maps.
The MS-COCO dataset (Lin et al. 2014)[3] was annotated at Microsoft and its name stands for Common Object in Context. It contains 80 classes, including the 20 Pascal VOC classes and also provides bounding boxes and segmentation maps. COCO is roughly 10 times larger then Pascal VOC, but the objects are also more difficult to detect.
ImageNet (Russakovsky et al. 2014) is most known as the large dataset with 1.4 million images and objects in 1,000 categories, that is used in the ILSVRC[4] Classifi-

[2] http://host.robots.ox.ac.uk/pascal/VOC
[3] https://cocodataset.org
[4] ImageNet Large Scale Visual Recognition Competition (ILSVRC)

cation Challenge[5]. 450,000 of these 1.4 million images were annotated with bounding boxes for objects in 200 different categories to serve as training data for the ILSVRC Detection Challenge.
The Oxford-IIIT Pet dataset (Parkhi et al. 2012)[6] contains 7349 image with one of 37 cat and dog breeds per image. Each image has ground truth in form of breed annotation, head bounding box and a trimap segmentation of the pet.
The KITTI Vision dataset (Geiger et al. 2012)[7] is an effort related to autonomous driving, but it also contains 7.000 images with bounding box annotation for 3 classes ('pedestrian', 'car', 'cyclist').
Due to the rather small size of detection datasets, transfer learning (Section 2.7) is established in practice to exploit trained classification models for object detection.

3.3 Evaluation Metrics

Evaluation metrics are used to assess the performance of object detection systems. When object detection is considered on the basis of bounding boxes, the *Jaccard Index* or Intersection over Union (IoU), as defined in (3.1), is the primary measure on how many pixels of one bounding box A overlap with those of a second bounding box B.

$$\mathrm{IoU}(A, B) = \frac{A \cap B}{A \cup B} \tag{3.1}$$

A detected bounding box is correctly detected and considered as True Positive (TP) if the detected box has a sufficiently large overlap (e.g. IoU > 0.5) with the ground truth bounding box and it was correctly classified. It is considered as False Positive (FP) if it was detected, but no matching ground truth box exist. A ground truth box that does not get detected at all is considered as False Negative (FN). Multiple detections of the same object are taken as FP and only the detection with the highest confidence is taken as TP.
Two metrics that are derived from this concept are *precision* and *recall*.

$$\mathrm{precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{3.2}$$

$$\mathrm{recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3.3}$$

Where TP, FP and FN stand for the number of the corresponding cases. Precision is a measure for how few false detections are made and recall is a measure for how many actual objects are correctly detected. For the perfect detector, one may desire the highest possible precision and recall of 1.0, but every real detection system has

[5] http://www.image-net.org/challenges/LSVRC
[6] http://www.robots.ox.ac.uk/~vgg/data/pets
[7] http://www.cvlibs.net/datasets/kitti/

to make a tradeoff between precision and recall.
A metric that takes this tradeoff into account is the *F-measure* or *F1-score* (3.4). It is the harmonic mean of precision and recall.

$$\text{f-measure} = 2\,\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3.4}$$

For the most detection approaches, the tradeoff between precision and recall can be made by rejecting detections under a certain confidence threshold and the precision-recall curve can be plotted for each class.
A more sophisticated performance metric that will now be introduced is the Average Precision (AP). Theoretically, it is the the integral $\int_0^1 p(r)dr$ under the precision-recall curve, that is typically approximated by

$$\text{AP} = \sum_{k=1}^{K} p(k)\Delta r(k) \tag{3.5}$$

Where K is the total number of detections, $p(k)$ the precision at a cutoff of k detections (the k detections with the highest confidence) and $\Delta r(k)$ is the change in recall between cutoff $k-1$ and k. This definition is not always consistent in publications and some authors use interpolated version of the AP. If $\max_{\tilde{k}\geq k} p(\tilde{k})$ is used instead of $p(k)$, which results in higher AP values, it is called the interpolated AP. Another variant where the precision is sampled at 11 fix points $0.0, 0.1, \ldots, 0.9, 1.0$ is known as 11-point interpolated AP.
However, the most commonly used metric for detection is the mean Average Precision (mAP). It is obtained by separately computing the AP for each class and averaging them.

$$\text{mAP} = \frac{1}{|\text{classes}|} \sum_{c \in \text{classes}} \text{AP}(c) \tag{3.6}$$

The value of the mAP is then typically denoted with "mAP@0.5" where "0.5" means that the evaluation was done with an IoU threshold of 0.5. The computation is performed with all samples in the test set.
Some benchmarks, as in the evaluation metrics for MS-COCO, also average over multiple IoU thresholds. This will be indicated, for instance, with "mAP@[.50:.05:.95]", where 10 IoU thresholds from 0.5 to 0.95 were used.
Also, a two-dimensional *confusion matrix* may give useful insights on how good a classifier works and which classes are frequently misclassified. One dimension of the matrix represents the actual (ground truth) classes and the other the predicted classes. The elements of the matrix are the number of ground truth elements of one class detected as object of a certain class. In case of a perfect classifier, the confusion matrix would be diagonal.

3.4 CNN based Object Detectors

From a historical point of view, object detection is an old problem, that is classically tackled with a sliding window approach. A fixed sized pattern or feature is compared with the input image at each possible location, and after getting all the predictions, some of them are discarded and some are merged. This is simple but inefficient, especially for a high number of features and different window sizes. Another issue with these approaches is that the features typically have to be engineered by hand. The most notable of these efforts were, the Viola-Jones framework (Viola and Jones 2001), the combination of HOG feature pyramid with SVMs (Dalal and Triggs 2005) and Deformable Part Modelss (DPMs) (Felzenszwalb et al. 2013). Viola-Jones uses cascades of Haar-like features and AdaBoost to get an efficient ensemble of many binary classifiers. It was also the first real-time face detector with application in point-and-shoot cameras. The combination of a HOG feature pyramid with SVMs as classifiers is superior to Viola-Jones in terms of AP, but also much slower. DPM are the approach that came closest to today's CNN-based detectors (Girshick et al. 2014). They consider each object as a deformed version of an ideal template and use dynamic programming to optimize a graph that represents the relations between the features. The problem with DPMs is that they still depend on HOG features. CNNs revolutionized the sliding window approach due to two main properties. First, they make it possible to learn the features, and second, they have an intrinsic capability to share and parallelize the computation. However, all the most successful approaches that are currently available can be understood as extensions of image classification models. Some relevant ones will now be discussed in more detail.

3.4.1 OverFeat

Krizhevsky et al. 2012 won with AlexNet not only the ILSVRC classification but also the ILSVRC localization challenge 2012. However, since they never published anything regarding localization, OverFeat (Sermanet et al. 2013) was the first publication related to efficient object detection with CNN. The architecture of OverFeat was a optimized version of AlexNet (bigger model, more layers, smaller stride, 145M parameters). The authors realized that the last FC layers of a classification network can be converted into convolutional layers and that the resulting fully convolutional network can handle inputs of different sizes. It also allows to perform the classification on different input locations while sharing the computation of overlapping regions. The model was pre-trained for classification only and fine-tuned for classification and regression in one model. Due to its fully convolutional architecture, the model can be evaluated at multiple scales. The bounding boxes of all locations and scales are accumulated instead of suppressing them, resulting in an increased detec-

tion performance. OverFeat was the winner of the ILSVRC localization challenge 2013 and later gave rise to many other papers on learning bounding boxes.

3.4.2 R-CNN

R-CNN (Girshick et al. 2013) and its successors are one of the most impactful advancements in computer vision[8]. The name R-CNN stands for Region and Convolutional Neural Network, and that is literally what it does. It uses a region proposal method, called Selective Search[9] (Uijlings et al. 2013), that generates 2000 different regions with a high probability of containing an object. The image regions are then 'warped' into images of fixed size and fed into a CNN (AlexNet in this case). The so extracted feature vector is then used as input for a set of linear SVMs (Section 2.5.4) that are trained for the individual classes. To obtain more accurate coordinates, the same feature vector is also fed into a bounding box regressor. After performing the classification and regression for each region, Non Maximum Suppression (NMS)[10] is used to suppress bounding boxes that have a significant overlap with each other. R-CNN was intuitive but also very slow and took a lot of disk space.

Fig. 3.2: Selective Search bottom-up segmentation starts with tiny segments and incrementally merges (from left to right) them by similarity to obtain region proposals of different size (Uijlings et al. 2013).

[8] According to Google Scholar, the original paper has been cited more than 5000 times.

[9] Selective Search does bottom-up segmentation and merges the segments greedily by utilizing some similarity measures. The segments are then converted to boxes.

[10] Bounding boxes of the same class and with IoU > 0.5 are discarded so that only the instances with the highest confidence will remain.

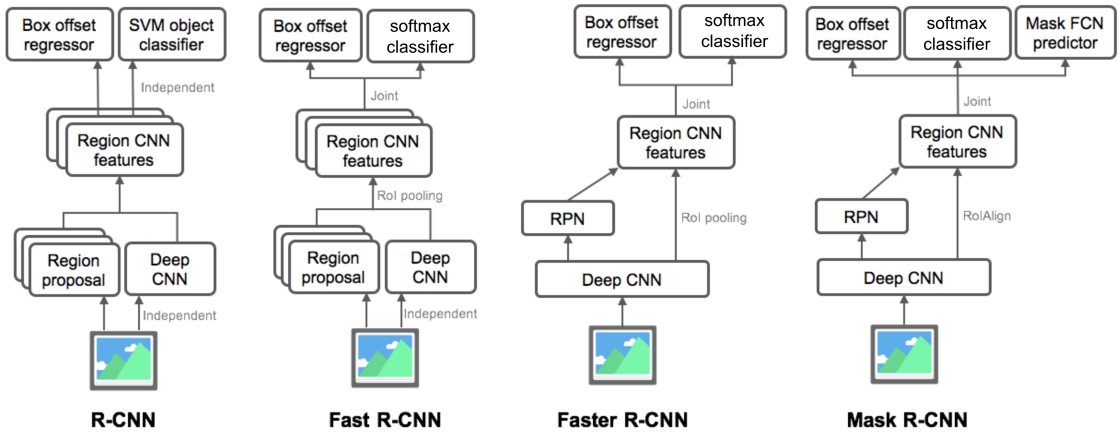


Fig. 3.3: Evolution of the R-CNN detector family (Weng 2017)

3.4.3 Fast R-CNN

Fast R-CNN (Girshick 2015) solved the speed issue of R-CNN by swapping the order of generating region proposals and running the CNN. Therefore, the region proposals are generated from the last feature map, rather then from the original image. The authors introduced a Region of Interest (RoI) Pooling layer after the last convolution, which performs the warping and cropping of the RoIs into an fixed size input that can be handled by the subsequent FC layers. They also replaced the SVMs with a single softmax layer that utilizes the NN for classification instead of training separate SVM models for each class. Since the RoI Pooling layer is a differentiable operation, the whole FFNN can be trained with a multi-task loss for RoI classification and bounding box regression in an end-to-end manner. The CNN computation is shared between all the RoIs and only one forward/backward pass is required per image. The proposals/bounding boxes are projected on the feature maps (intuitive by consideration of the receptive fields) and thus the down sampling from a large input resolution to a rather small feature map leads to quantization errors and misalignments of the bounding boxes in the input image. A bigger issue with Fast R-CNN is that it still requires Selective Search, which is now the bottleneck in the inference pipeline. The RoI Pooling layer is a special case of Spatial Pyramid Pooling layer (He et al. 2014).

3.4.4 Faster R-CNN

Faster R-CNN (Ren et al. 2015) has the same architecture as Fast R-CNN, but it gets rid of Selective Search by replacing it with a CNN, the so called Region Proposal Network (RPN). The RPN is attached to the last convolutional layer and outputs 'objectness' scores and bounding box regression for multiple anchor boxes at each location of a so called anchor feature map that is obtained by simply applying a 3x3

convolution on the input of the RPN. Anchor boxes (sometimes also default or prior boxes) are a prior in form of manually defined reference boxes with different scales and aspect ratios that help the bounding box regression easier fit to the objects. The output of the RPN is then filtered by the objectness score, overlapping boxes are eliminated by NMS and the remaining bounding boxes are used as region proposals for the RoI Pooling layer. The RPN serves as attention mechanism for the detection, but since the derivatives of the RoI pooling layer w.r.t. the bounding box coordinates are not easy to get, the whole system is a bit tricky to train. One way to handle this is to start with a model pre-trained on ImageNet and then alternately train the RPN, use the region proposals to train the detector, train the RPN and so on. The mini-batch arise from a single image by randomly sampling 256 anchors. To prevent the negative samples from dominating the positive ones, the ratio between the positive and negative is kept close to one. Faster R-CNN takes 0.2 seconds per image (compared to R-CNN with 60 sec/img and Fast R-CNN with 2 sec/img) and has become the default choice for most detection tasks today.

3.4.5 Mask R-CNN

Mask R-CNN (He et al. 2017) makes instance segmentation possible. It essentially takes the architecture of Faster R-CNN and combines it with a Fully Convolutional Network for semantic segmentation (Long et al. 2014)[11]. Therefore, a third branch, that performs semantic segmentation for each RoI, was added next to the classification and bounding box regression branch. The authors also addressed the issue with the quantization errors, that results from the RoI Pooling layer and replaced it with a modified version that they call RoI Align. It takes region proposals in form of floating numbers instead of integers and applies bilinear interpolation to avoid the information loss. A third major change in Mask R-CNN was the decoupling of the class prediction. Instead of the softmax layer, they used a binary classifier for each individual class. The advantage is now, that the different classes are no longer in competition with each other (e.g. the instances of class 'woman' are also contained in class 'person'). Both, the change from RoI Pooling to RoI Align and the change from softmax to sigmoid, increased the AP significantly. A further increase in terms of AP came due to the change of the feature extractor from VVG-16 to ResNet-101/ResNeXt-101 (Xie et al. 2016).

[11] FCN takes a image classification model, attaches a deconvolution or transposed convolution layer (Dumoulin and Visin 2016, Chapter 4) and trains it for classifying each pixel of the input image.

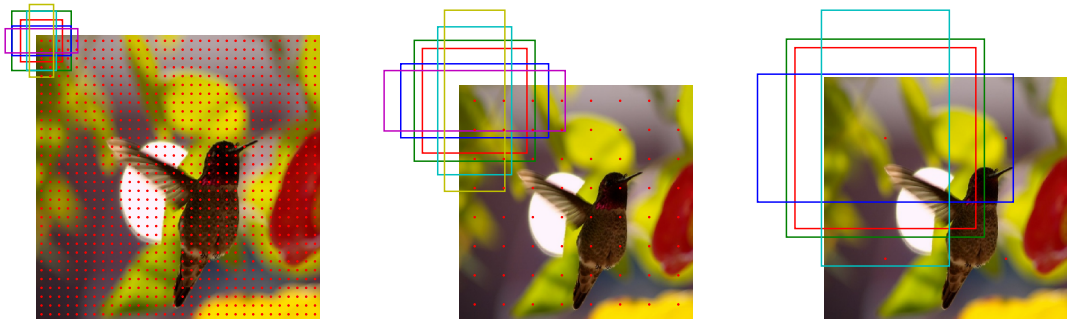


Fig. 3.4: Prior boxes defined on different feature maps in the SSD architecture; The prior, anchor or default boxes are defined at each of the locations (red dots), but for clarity, only the boxes at the first location are drawn.

3.4.6 R-FCN

R-FCN (Dai et al. 2016) builds on Faster R-CNN. It maximizes the sharing of the computation by getting rid of the RoI-wise sub-networks and making the whole network fully convolutional. The authors do this by introducing position-sensitive score maps and attaching a Position-sensitive RoI Pooling layer on top of them. Each RoI is divided into a $k \times k$ grid and elements are considered as 'bins' or sub-regions. For the classification part, $(C+1)k^2$ score maps are predicted from the last feature map (C classes plus the background class). Each score map is associated with one sub-region of a class, or in other words, they are convolutional feature maps that will be trained to recognize certain parts of an object class (e.g. the top-right of a cat face). The Position-sensitive RoI Pooling layer accumulates the scores from the subregion on the associated score map and divides it by number of pixels in the sub-region (average pooling, but max pooling is also possible). The scores of all bins in the RoI are then averaged per class and a softmax is applied to train it with the CE loss. The regression part is handled in a similar way, $4k^2$ score maps are predicted, average pooling for each bin is performed and the average over the bins results in 4 values for fitting the bounding box. The concept of sub-regions is able to address the dilemma of increasing translation invariance for image classification vs minimizing translation variance for object detection. Location variance by proposing different object regions and location invariance by having each region proposal refer to the same bank of score maps. The architecture also uses ResNet as feature extractor. During training, Online Hard Example Mining (OHEM)[12] (Shrivastava et al. 2016) is used and in the test case NMS is applied. One issue with R-FCN is that the number of score maps increases with the number of classes, this is addressed by R-FCN-3000 (Singh et al. 2017). Later, the authors of R-FCN published about Deformable Convolution and combined the idea of sub-regions with that of Deformable RoI

[12]For OHEM the per-RoI classification loss is calculated and only the samples with the highest loss are used for backpropagation.

Pooling (Dai et al. 2017).

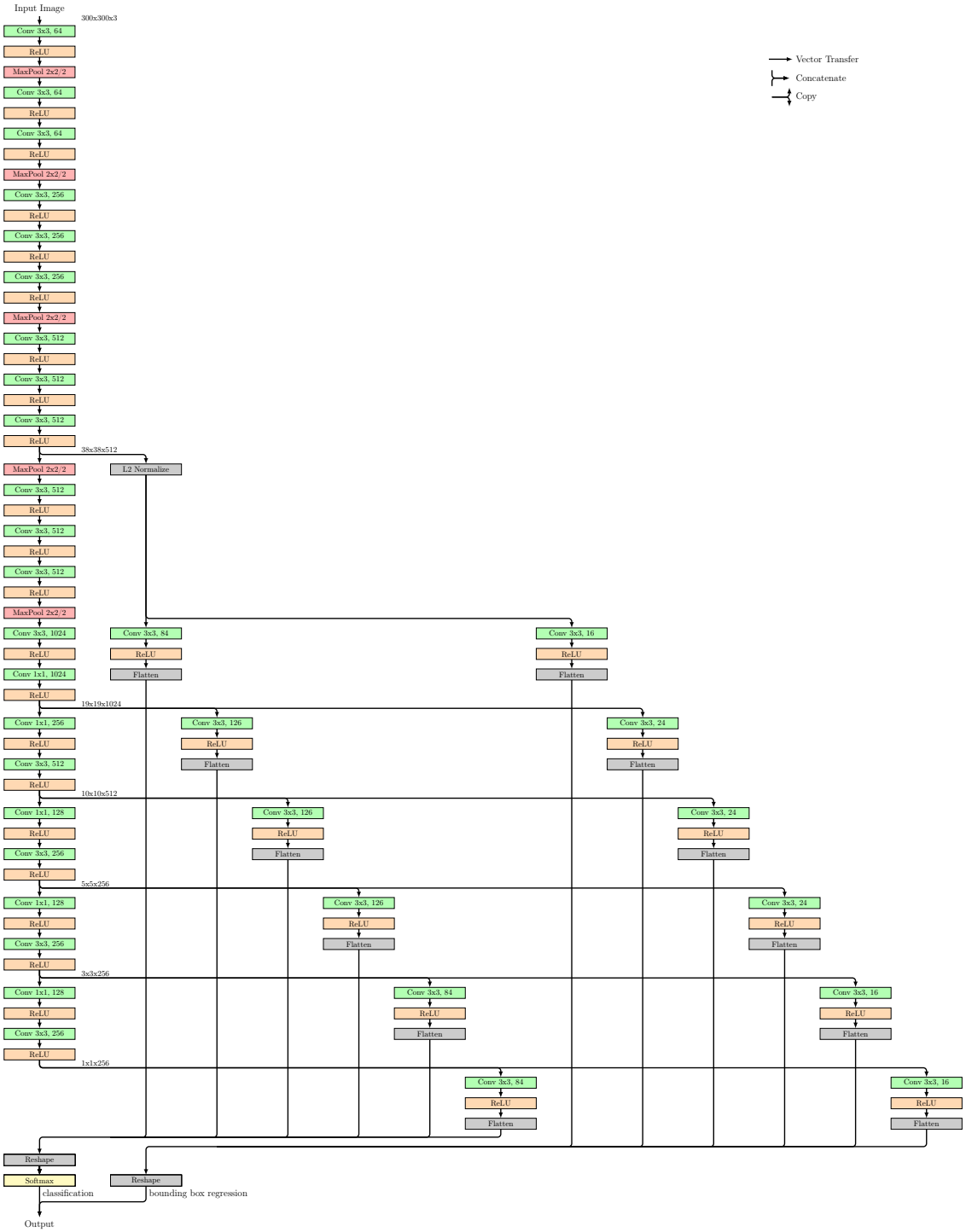


Fig. 3.5: The SSD architecture with its backbone or feature extractor on the left and its prediction paths to the right

3.4.7 SSD

SSD (Liu et al. 2016a) stands for Single Shot MultiBox Detector and is an approach that does not require region proposals at all. It does a dense prediction of bounding boxes over the entire image which then get filtered by NMS. The architecture, as shown in Figure 3.5, builds on a pre-trained VGG-16 model, where the first FC layer is converted into a convolutional layer and the rest of the classification head is replaced by a stack of new convolutions. A set of prior boxes with different aspect ratio and scale is defined for each spatial location on several feature maps (e.g. for feature map size 32x32, 8x8, 2x2 as drawn in Figure 3.4). This is the same idea as with the anchor boxes in Faster R-CNN and rather intuitive since the elements of the feature map at one spacial location are all connected to the input pixels in the receptive of this units. Classification and offset regression for all default boxes is done by convolutions over the feature maps (see Figure 3.5). To obtain a model that is robust to variations in object size and shape, the pre-trained model is finetuned with data augmentation. Training requires local ground truth. Thus a ground truth bounding box is associated with a prior box if their IoU is larger than 0.5. Prior boxes without associated ground truth are of class 'background'. The loss is a multi-task loss $\mathcal{L} = \mathcal{L}_{cls} + \lambda\mathcal{L}_{reg}$. Where the first term $\mathcal{L}_{cls}$ is the sum over the CE losses for classifying the boxes selected by hard negative mining[13] with neg/pos ratio not larger then 3. The second term $\mathcal{L}_{reg}$ is the sum over the Smooth L1 Losses for the bounding box regression of the true positive samples. At test time, the output of the network is the aggregation of all the predictions from the different feature maps and is subsequently filtered by confidence threshold and NMS. The benefits of SSD come from its simple and fully convolutional architecture. It is trained end-to-end, it has a high detection speed with comparable AP, and it can process inputs with different sizes. One issue with SSD is that it relies on a pre-trained model with features that are not specific for the detection task. DSOD (Shen et al. 2017) adresses this by replacing the VGG-16 backbone with DenseNet (Section 2.6). This results in a model that can be trained from scratch, has much less parameters and achieve a higher AP. Another issue is that SSD has problems with detecting small objects. This is most likely caused by insufficient contextual information. Approaches that try to involve more context (Cao et al. 2017; Chen et al. 2017; Fu et al. 2017; Lee et al. 2017; Li and Zhou 2017) typically improve the AP on PASCAL VOC by less then 2 percent points. BlitzNet (Dvornik et al. 2017) extends SSD with semantic segmentation.

[13] Hard negative mining is similar to OHEM, it prevents the negative class ('background') from dominating the positive ones by only selecting the negative samples with the highest classification loss.

3.4.8 YOLO

You Only Look Once (YOLO) is the first of a series of single shot detectors and was the inspiration for SSD. It has a rather simple architecture and is mainly optimized for speed. Since an extensive discussion of YOLO would not introduce that many new ideas at this place, the reader is referred to the original papers (Redmon et al. 2015; Redmon and Farhadi 2016, 2018).
The TensorFlow detection API[14] provides implementations of various detectors (e.g. SSD with MobileNet or Inception v2, R-FCN with ResNet-101, Faster R-CNN with ResNet-101 or Inception ResNet v2) that can be easily applied for practical use. A detailed comparison of Faster R-CNN, R-FCN and SSD was elaborated by Huang et al. 2016b. Also, the object detectors presented here can be modified relatively easily for custom applications. Some of them are human pose estimation (Güler et al. 2018), Face detection (Wang et al. 2017b), 3D bounding box estimation (Mousavian et al. 2016; Simon et al. 2018) and of course text detection, which will be extensively discussed in the next section.

[14] https://github.com/tensorflow/models/tree/master/research/object_detection

Chapter 4

ANN-based Text Spotting

Text in general is an object or pattern produced by humans as explicit carrier of high level semantics. It differs from generic objects, which generally have a well-defined boundary and center, allowing it to infer the whole object from only a part of it. Moreover, it is a sequence with variable length and tremendous geometrical variations.
Text in natural scene images typically suffers from poor image quality and the text instances vary heavily in scale, font and style. Scene text may also be arbitrarily oriented, perspectively distorted or curved.
All these properties make scene text reading, a.k.a. text spotting, photo OCR or text recognition in the wild, a unique and challenging task that requires the exploitation of low-level geometric and high-level semantic features.
Approaches to the scene text reading problem usually consider it as at least two sub-problems, text detection and text recognition.
Classical approaches prior to DLs typically had many intermediate stages, required tedious parameter tuning, manually designed features and are highly application specific.
Current state-of-the art ANN based approaches are far superior to these former methods due to learned features and shared computation. They typically have no intermediate stages, except final NMS or decoding, and are characterized by being faster, more robust and more accurate.
Surveys on text detection and recognition can be found in Ye and Doermann 2015 and Zhu et al. 2016. However, this chapter will cover more recent ideas which are not considered in these surveys. Section 4.1 will provide an overview of text related datasets. Section 4.2 and 4.3 consider the text detection and recognition on their own. The final Section 4.4 is about more holistic approaches that are end-to-end trainable.

4.1 Datasets for Text Detection and Recognition

The supervised training and evaluation of models for scene text detection and recognition systems require large datasets.
Most of the available real-world datasets were provided in context of the International ICDAR[1] Robust Reading Competition or are annotated Google Street View images. However, the real-word datasets are not that large and annotating is expensive. For this reason, synthetic data engines were developed for generating larger training datasets with scene images containing highly realistic text instances. Synthetic data may seem strange, but it works well for training and is a rather natural approach, since text is a object produced exclusively by humans. Annotation is typically done via bounding boxes at character, word or line-level and the transcribed character string. Datasets related to special cases, such as curved text, may also provide segmentation maps to indicate the text areas.
The rest of this section provides a short overview on frequently used text-related datasets.

ICDAR 2003 (Lucas et al. 2003) contains 251 full scene images with horizontal text and 860 images of cropped words.
Wang et al. 2011 removed word images with non-alphanumeric characters or less then three characters and defined a 50-word lexicon for each. In addition a 50k word lexicon is available that consists of all lexicon words and words from the Hunspell spell-checking dictionary [2].

ICDAR 2015 FST or ICDAR 2013[3] (Karatzas et al. 2013) contains 229 training and 233 testing images of Focused Scene Text. It inherits most of its samples from the ICDAR 2003 dataset. All of them are real-world images showing text on sign boards, books, posters or other objects (Figure A.1). The text instances are English and horizontally aligned. The annotations are axis-aligned bounding boxes and result in 1015 cropped word images. The dataset is widely adopted for text detector performance and will be referred to as ICDAR 2013.

ICDAR 2015 IST[4] (Karatzas et al. 2015) was provided for the ICDAR 2015 challenge 4 on Incidental Scene Text. It contains 1,000 training and 500 test images. The images were acquired by using Google Glass without taking care of viewpoint, positioning or image quality. Text instances may appear in any orientation and with small size or low resolution (Figure A.2), making it much more difficult than

[1] Conference on Document Analysis and Recognition (ICDAR)
[2] http://hunspell.github.io
[3] http://rrc.cvc.uab.es/?ch=2&com=downloads
[4] http://rrc.cvc.uab.es/?ch=4&com=downloads

ICDAR 2013. The complete dataset has 7,548 text instances with annotation in form of quadrilaterals. It is often used for benchmarking and will be now referred to as ICDAR 2015.

ICDAR 2017 MLT[5] (Nayef et al. 2017) is intended for benchmarking the task of Multi-Lingual scene Text detection and identification. It contains 7,200 training, 1,800 validation and 9,000 testing natural scene images with text in 6 different scripts (Latin, Arabic, Bengala, Hangul, Hiragana, Katakana and Symbols). Annotations are provided in form of quadrilaterals, script class and transcription (UTF-8 text).

COCO-Text[6] (Veit et al. 2016) was created by annotating images from the MS COCO dataset (Section 3.2). It contains 63,686 images (Figure A.3) with 173,589 labeled text regions and is thus two magnitudes larger than other scene text datasets. Each text instance is annotated with an axis-aligned bounding box and three attributes, machine printed or handwritten text, legible or illegible text and English or non-English script. For legible text, the transcriptions is given. The authors have chosen 20,000 samples as validation/test set and the rest for training. COCO-Text is also used in the ICDAR 2017 Robust Reading Challenge on Text Localization, Cropped Word Recognition and End-To-End Recognition. It is much more challenging dataset than ICDAR 2015 IST.

Synthetic Word[7] (Jaderberg et al. 2014) is a dataset of 9M synthetic word images covering 90k English words. The font for each word was randomly chosen from a set of 1,400 different font types and their borders/shadows were rendered with random width. The basic coloring was chosen from color samples obtained from k-means clustering on natural images and image patches randomly sampled from the ICDAR 2003 training dataset were used as background. Perspective transformations were used to simulated projective distortion. Noise and blurring effects as well as JPEG compression artifacts were added to make the images more realistic. All images have a fixed height of 32 pixels and variable width.

SynthText[8] (Gupta et al. 2016) contains 800k training images generated by using a synthetic text engine. They were synthesized by blending natural images with artificially rendered text. Text with random font type, size and color is perspectively placed at regions with uniform color and texture, taking into account the 3D scene geometry. Each image has about ten word instances annotated with oriented character and word bounding boxes plus transcription. Example images are attached in

[5] http://rrc.cvc.uab.es/?ch=8&com=downloads
[6] https://vision.cornell.edu/se3/coco-text
[7] http://www.robots.ox.ac.uk/~vgg/data/text
[8] http://www.robots.ox.ac.uk/~vgg/data/scenetext

Figure A.4.

MSRA-TD500[9] (Cong Yao et al. 2012) was the first popular dataset with focus on oriented text. It contains 500 high-resolution natural images, 300 for training and 200 for testing, which were taken from indoor (office and mall) and outdoor (street) scenes using a pocket camera. The text instances are either Chinese or English and are annotated as rotated bounding boxes at line-level. Compared with ICDAR 2003, MSRA-TD500 is more challenging due to higher variability in text and more complex backgrounds.

Google FSNS[10] (Smith et al. 2017) consisting of more than a million images of French Street Name Signs cropped from Google Street View images. Each image contains four views of the same street name sign and is annotated with a ground truth text as it would appear on a map. The text instances at each sign can span up to three lines and may suffer from blur, occlusion or low resolution. FSNS is also used in the ICDAR 2017 Robust Reading Challenge on End-To-End Recognition.

SVT[11] (Wang et al. 2011) stands for Street View Text. The dataset has 249 images with 647 word instances collected from Google Street View. The images may suffer from low resolution and unfavorable lighting conditions. Each image is associated with a 50-word lexicon defined by Wang et al. 2011.

SVT-Perspective (Phan et al. 2013) contains 639 cropped testing images picked from side-view snapshots in Google Street View which may have severe perspective distortions.

CUTE80[12] (Risnumawan et al. 2014) is intended for evaluating the performance of curved text recognition. It contains 80 natural images with curved text, resulting in 288 cropped word images.

IIIT 5k-word[13] (Mishra et al. 2012) consists of 5k cropped word images harvested from Google image search. They were obtained by querying words like billboards, signboard, house numbers, house name plates or movie posters. The dataset is split into 2k training and 3k test images. Each sample comes with ground truth text, character-level bounding box, categorization into easy or hard and two lexicons, one

[9] http://www.iapr-tc11.org/mediawiki/index.php/MSRA_Text_Detection_500_Database_(MSRA-TD500)
[10] http://rrc.cvc.uab.es/?ch=6&com=downloads
[11] http://www.iapr-tc11.org/mediawiki/index.php/The_Street_View_Text_Dataset
[12] http://cs-chan.com/downloads_CUTE80_dataset.html
[13] http://cvit.iiit.ac.in/projects/SceneTextUnderstanding/IIIT5K.html

with 50-word and one with 1k-word. In addition, a overall lexicon with 500k words is provided.

Total-Text[14] (Ch'ng and Chan 2017) is a relatively new dataset. It contains 1555 images with text instances that are either horizontal, multi-oriented or curved. The complete set is split into 1255 images for training and 300 images for testing and text instances occur with 7.27 per image rather often. All of them come with ground truth text, orientation type ('curved', 'horizontal', 'multi-oriented' or 'dont care'), polygon shaped bounding regions and segmentation masks at character and region level.

4.2 Text Detection

Given an image, the objective of text detection is to obtain precise bounding boxes or segmentation maps for all text instances in the image.
Bounding boxes are used to represent ground truth or detection results at character, word or line level. Single characters are often considered by bottom-up approaches that try to build connected components. Lines are primarily a suitable representation for document processing. However, most of today's scene text detection systems and benchmarks rely on word-level bounding boxes. Methods that can only detect horizontal text typically use axis-aligned bounding boxes which are either parametrized by their center and size (x, y, w, h) or by two coordinates $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$. Other methods that can detect arbitrary oriented text work with rotated bounding boxes which may be parametrized by the center, size and orientation angle (x, y, w, h, θ), two base coordinates and the height (x_1, y_1, x_2, y_2, h) or by the distance of the edges to a relative coordinate and the rotation angle (t, b, l, r, θ). Another common representation are quadrilaterals in form of four corner points $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$ and may be a better representation for perspective text instances. Methods that produce segmentation maps typically post-process them to obtain bounding boxes that can be used for benchmarking.
The metrics used to measure the performance of text detection systems are the same as those for general object detectors (Section 3.3), but most often, precision (3.2), recall (3.3) and f-measure (3.4) are the values that are compared.
In general, defining the problem by fitting word-level bounding boxes has its limitations when it comes to languages which have no word representations (e.g. Chinese or Thai). Moreover, it is also questionable how informative it is to measure the accuracy of the detector by how well it approximating the way human annotators create bounding boxes around text, and these without considering how well a subsequent

[14] https://github.com/cs-chan/Total-Text-Dataset

recognition stage would recognize the text.
Pre-DL approaches to the text detection problem typically work locally by seeking text fragments via extremal region extraction and edge detection, followed by character, word or line-level candidate aggregation and false positive elimination. The most of these methods are based on Stroke Width Transform (SWT) (Epshtein et al. 2010) or Maximally Stable Extremal Regions (MSER) (Neumann and Matas 2010), but also HOG features were a popular tool at these time and have been proposed in combination with the sliding window approach (Wang et al. 2011).
Starting in 2012, the first DL-based approaches to the text detection got the attention from the community. Wang et al. 2012 used a CNN in sliding window fashion to detect text instances. Goodfellow et al. 2013a read house numbers in Google Street View images.
All the most successful text detection methods from today are either modifications of box-based object detectors like Faster R-CNN (Section 3.4.4) and SSD (Section 3.4.7), rely on segmentation as proposed in FCN (Long et al. 2014) or are combinations of these approaches. Therefore, the subsequent sections (Section 4.2.1, 4.2.2 and 4.2.3) are about recent text detection concepts related to region proposal networks, single shot detectors and segmentation. A more quantitative comparison of these approaches can be found in Table 4.1.

4.2.1 Region Proposal-based

Region proposals and RoI-pooling are used by general object detectors (Section 3.4.2 - 3.4.5) to regress and classify axis-aligned bounding boxes. For text detection, it is quite intuitive to try to modify these mechanisms to make them better fit arbitrary oriented text instances with large and varying aspect ratio. Different approaches related to this idea have been proposed.
Jaderberg et al. 2016 were inspired by R-CNN. They use Edge Boxes (Zitnick and Dollár 2014) as region proposal method and Aggregate Channel Feature (Dollár et al. 2014) to predict horizontal word-level bounding boxes. As in R-CNN, a regression head is used to refine the proposals, but the classification is done via a random forest classifier (Bosch et al. 2007) and proposals are filtered via NMS.
Tian et al. 2016 proposed a method for horizontal text line detection, the Connectionist Text Proposal Network (CTPN). The idea emerged from the RPN in Faster R-CNN. Anchors width different height are defined at each location on the last convolutional layer of a VGG-16 model. A BLSTM is attached and the anchors in each row are considered as a sequence. For each location or time-step in the sequence, text/non-text scores, y-coordinates and side-refinements are predicted via FC layers. The x-coordinates and widths of the anchors are fixed. The so obtained proposals are subsequently filtered by threshold and NMS. Positive proposals are

connected based on their horizontal distance and vertical overlap. The resulting connected proposals are then merged into word-level bounding boxes and the estimated values for the side-refinement are used to improve the left and right boundary. Zhu et al. 2017 took up the idea of the vertical anchor mechanism and applied it with a ResNet backbone.

Ma et al. 2017 modified Faster R-CNN to work with oriented region proposals and produce text candidates in form of oriented bounding boxes. This is done by introducing anchor boxes in 6 different orientations. Consequently the RPN and the RoI-Pooling layer have to be changed into variants that can handle the rotation. The RRPN now generates proposals with 5 geometric parameters (x, y, w, h, θ) and the RRoI-Pooling layer pools the RRoIs to a fixed size output. Also, the calculation of IoUs and the procedure for NMS are less trivial. Nevertheless, text candidates with small or extremely large aspect ratio are still an issue.

Jiang et al. 2017 proposed R^{2}CNN, which is another variant of Faster R-CNN that can better cope with arbitrarily oriented text. Therefore, it first uses a RPN to predict axis-aligned bounding boxes and subsequently pools the RoIs into three different aspect ratios. These pooled feature maps are then concatenated and fed into the first of two FC layers. The output of the second FC layer is then used for text/non-text classification, axis-aligned bounding box refinement and, most important, parameter estimation for an oriented bounding box. For IoUs calculation and NMS, the method proposed by Ma et al. 2017 is adopted.

Model	**Precision**	**Recall**	**F-measure**	**FPS**
Yao et al. 2016	72.3	58.7	64.8	1.6
R^{2}CNN	86.6	79.7	83.0	—
RRPN	84.0	77.0	80.3	—
CTPN VGG16	74.2	51.6	60.9	7.1
EAST PVANET2x MS	83.3	78.3	80.7	—
EAST PVANET2x	83.6	73.5	78.2	13.2
EAST VGG16	80.5	72.8	76.5	6.5
SegLink VGG16	73.1	76.8	74.9	—
TextBoxes++ VGG16	87.2	76.7	81.6	—
TextBoxes++ VGG16 MS	**87.8**	78.5	82.9	—
PixelLink VGG16 2s	85.5	**82.0**	**83.7**	3.0
PixelLink VGG16 4s	82.9	81.7	82.3	7.3

Tab. 4.1: Comparison of text detection approaches on the ICDAR 2015 dataset; values are taken from the original papers, 'MS' stands for multi-scale testing, '2s'/'4s' stands for different Conv layer from where the prediction is made, frame rate was measured on Titan X GPU, except Yao et al. 2016 on K40m.

4.2.2 Single Shot Detector-based

SSD and other single shot detectors have the advantage that they allow practically arbitrary local predictions, as long as they are correlated with the content of the receptive fields on which the prior boxes are defined. The intention to utilize this for text detection led to various successful approaches in recent years. Some of them are now presented to the reader.

Liao et al. 2016 proposed TextBoxes, an efficient detector for horizontal text. It is actually the SSD (Figure 3.5) architecture with three minor changes. First, more text specific default boxes with much larger aspect ratios (1,2,3,5,7 and 10) are used. Second, to better cover the input image with default boxes, a second identical set of default boxes is added with a vertical displacement. And third, irregular 1x5 convolution kernels are used within the prediction branches, which leads to wider receptive fields and helps the model to better capture the horizontal text areas. TextBoxes, as the first SSD-based approach, showed good results and is often cited.

Shi et al. 2017 utilized SSD in a connected component approach that they call SegLink. The idea behind SegLink is to just detect segments of text (corresponding to objects in SSD) and additionally predict confidences for their relation to neighboring segments. The segments are considered as rotated bounding boxes and do not necessarily have to be characters or words. One default box, representing a segment, is defined per location and results in 31 predicted values. These are 2 for the segment confidence, 16 for the confidence of 8 within-layer links, 8 for the confidence of 4 cross-layer links and 5 for the offsets of the bounding box[15]. For the cross-layer links to work properly, there is a further constraint on the source feature maps on which the default boxes are defined. They must have exactly the half spatial dimension as the previous one. The detected segments and links are then considered as graph, segments as vertices and links as edges. After filtering segments and links by thresholding, the remaining connected components are combined to oriented bounding boxes.

Liao et al. 2018 published TextBoxes++, which equips their TextBoxes for arbitrarily oriented text. Therefore, it does not only offset regression for axis-aligned bounding boxes, but also for rotated bounding boxes and quadrilaterals. To better fit to arbitrary oriented text instances, the aspect ratios of the default boxes are changed to (1,2,3,5,1/2,1/3,1/5), the size of the irregular convolution kernels is set to 3x5, and to speed up NMS, the NMS is first performed on the axis-aligned bounding boxes and then, only if it is necessary, on the computationally more expensive oriented bounding boxes/quadrilaterals. The authors mentioned that the simultaneous regression of axis-aligned bounding boxes, rotated bounding boxes and

[15] All confidences are modeled via Softmax Layers (Section 2.3.5) with two classes (positive and negative) and thus lead to two values each.

quadrilaterals has a positive effect on the training process and makes it more stable. They also proposed a simple but efficient way to eliminate false-positive detections. Therefore, they applied the TextBoxes++ model in with CRNN (Shi et al. 2015) for recognition and used a combined value of detection and recognition score to suppress candidates that are unlikely to contain meaningful words.
Zhou et al. 2017 proposed EAST, which outperformed the previous state-of-the-art on the ICDAR 2015 challenge by an f-measure increase of 16 percent. It is inspired by DenseBox (Huang et al. 2015), which predicts scores and bounding boxes for each pixel in the input image, but gets most of its performance gain from the PVANet[16] backbone architecture (Hong et al. 2016). For the pixel-wise predictions, an upsampling branch like in U-Net (Ronneberger et al. 2015) is used to gradually merge feature maps from multiple convolution blocks and exploit more contextual information. The predictions contain the confidence scores, rotated bounding boxes and quadrilaterals. Instead of hard negative mining, the class balanced CE loss from HED (Xie and Tu 2015) is used and loss functions for regression are chosen in such a way that they better cope with the high aspect ratio of the bounding boxes. For the rotated bounding boxes, the IoU loss as in UnitBox (Yu et al. 2016) is utilized in combination with the rotation loss in (4.1). For the quadrilaterals, the Smooth L1 loss is used and normalized by the shortest edge length of the quadrilateral.

$$\mathcal{L} = 1 - \cos(\hat{\theta} - \theta) \tag{4.1}$$

4.2.3 Segmentation-based

Text instances in scene text images usually lie very close to each other or even overlap, making it difficult, if not impossible, to separate them by methods that regress rectangular bounding boxes. In addition, curved and perspective text is another major issue for box-based detectors and a subsequent recognition stage. In such cases, segmentation methods represent a more promising approach. They typically produce more accurate pixel-wise predictions, but also require more computation and an additional post-processing step to get from a semantic or instance segmentation map to proposals in form of bounding boxes.
Yao et al. 2016 tackled the text detection problem in form of semantic segmentation and were inspired by HED (Xie and Tu 2015), an efficient CNN-based edge detector. This is not surprising since text in general correlates strongly with edges. HED does upsampling as in FCN (Long et al. 2014) and fuses features from different feature maps, which involves more contextual information and provides deep supervision (Lee et al. 2014) during training. For text detection, the model produces three

[16] PVANet is an architecture with residual connections and inception blocks designed for object detection.

dense prediction maps, one for the text regions, one for the individual characters and one for the linking orientation of the characters. Candidates for text regions and characters are then obtained via adaptive thresholding on the prediction maps. Character candidates within a region are then considered as vertices of a graph, where the edges represent the geometric (spatial and orientation) similarity of the characters. Finally, graph partitioning is used to split groups of character candidates into separate text lines. The advantage of this segmentation approach is that it can also handle curved and arbitrary oriented text.
Deng et al. 2018 proposed PixelLink, an instance segmentation approach that is actually inspired by SegLink. The idea is straightforward, instead of detecting segments and predicting their linkage to their neighbors, consider a map with same size as the input image and do classification and linking estimation for each individual pixel. The network output has then 18 values per pixel, 2 for the text/non-text confidence and 16 for the linking confidence with respect to the 8 neighboring pixels. Only links within the same instance are positive, otherwise they are negative. Pixels predicted positive are connected via positive links are joint into connected components and to obtain a minimal bounding boxes, the authors used the OpenCV (*The OpenCV Reference Manual* 2014) method *minAreaRect*. The architecture of PixelLink is quiet simple, a fully convolutional VGG-16 model as backbone, an upsampling branch with feature fusion from each of the conv stages in the backbone and a 1x1 convolution with 18 kernels for the predictions. PixelLink with VGG-16 backbone outperforms EAST with PVANet and multi-scale prediction on the ICDAR 2015 benchmark.

4.3 Text Recognition

The objective of text recognition is to take a cropped word image and generate the transcription of the word present in the image.
Since humans read text by processing it as a sequence, it is quite natural to consider text recognition as a sequence-to-sequence learning problem. Practically all recent and successful approaches do so and process text images either with RNNs or CNNs. For this reason, Section 4.3.1 will introduce some of these concepts.
Another issue inherited from box-based text detectors is that the cropped word images often contain curved or perspectively distorted text, making it difficult to handle them as proper sequence. To address this, more advanced methods have been proposed to bring the content of the image in a more sequence-like form. Section 4.3.2 will address this in more detail.
Section 4.3.3 will provide some common recognition metrics that can be used to measure and compare the performance of text detection systems.

4.3.1 Sequence-based Recognition Methods

Jaderberg et al. 2014 not only published their synthetic word dataset, but they also made experiments with CNNs for word recognition. The architecture (`3x(Conv->MaxPool)->Conv->FC->FC->Softmax`) was rather standard and had a fixed input of 32x100 pixels. The images were resized to the input size without preserving their aspect ratio. Based on this architecture, they came up with three different variants on how the network output is defined and the actual classification is performed.
The first variant is pure word classification with one class for each word in a dictionary. In this case the first FC has 4096 and the second FC layer has as many units as words in the dictionary (90k classes). Training is done with the Categorical CE Loss, starting with 5k classes and gradually increasing them after partial convergence. The large number of classes is quite feasible, but only words that are contained in the dictionary can be recognized.
The second variant was the naive sequence encoding approach with independent classifiers and without further language assumptions. One classifier for each character in a fixed size sequence. Instead of separately predicting the sequence length, as it was done by Goodfellow et al. 2013a, they introduced a null character to identify the end of the sequence. The last fully FC layer has then a dimension of 23 for the maximal sequence length times 27 for the number of characters in the alphabet. Consequently, the training is done with 23 softmax distributions and Categorical CE Loss.
The third and last variant encodes the output as a bag-of-N-grams[17]. When $|G_N|$ is the number of all N-grams in the training set, the network output is encoded to a $|G_N|$-dimensional binary vector for the gram occurrences. This vector is typically very sparse and for $N = 4$ there are only 7 collisions when encoding the words from the 90k-word dictionary. The last FC layer has G_N units with sigmoid activation, to represent the probability for the presence of the individual N-grams. Training is done with G_N Binary CE Loss functions and since some N-grams occur much more frequently then others, the gradients are scaled by the inverse of the frequency in the training set.
All three models achieved state-of-the-art results on ICDAR 2003, ICDAR 2013, etc., which is most likely due to the large amount of synthetic training data.
Shi et al. 2015 proposed Convolutional Recurrent Neural Network (CRNN), a simple but efficient architecture (Figure 4.1) for text recognition that uses a stack of convolutional layers to get from an input image, with theoretically arbitrary width, to a sequence of feature vectors which subsequently serves as the input to the first of two BLSTM layers (Section 2.4.4). Each feature vector represents a local region in

[17] N-grams are all possible sub-strings up to length N, e.g. for $N = 3$ G_3('word') = {'w', 'o', 'r', 'd', 'wo', 'or', 'rd', 'wor', 'ord'}.

the input image. The BLSTMs function as a mechanism that holds contextual information and thus help the model to easier distinguish between ambiguous characters. The whole model comes with relatively few parameters and is trained via CTC (Section 2.5.7) on the synthetic word dataset (Jaderberg et al. 2014). Tanks to the CTC, the transcription of the output can be done in a lexicon-based or lexicon-free manner. The authors also applied the method successfully to the problem of musical score recognition. CRNN and its architecture are rather basic and often used for recognition tasks (Jain et al. 2017; Liao et al. 2018). Also, the CTC is often the default approach to the sequence-to-sequence learning problem. Figure 4.2 shows typical model in- and output of a text recognition model trained with the CTC.

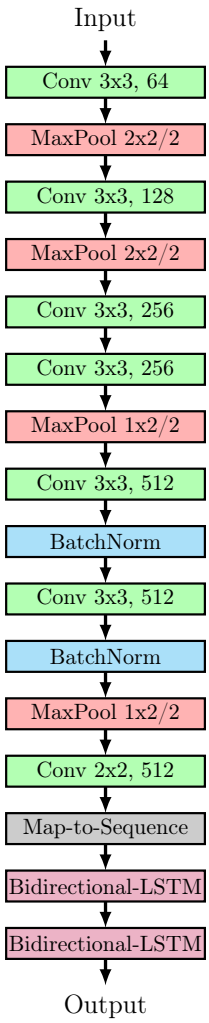


Fig. 4.1: CRNN Architecture (for clarity, the ReLU after each Conv layer are not displayed)

Lee and Osindero 2016 proposed R^2AM, another method that extracts features via convolutional layers and learns a character level language model via recurrent layers. For the CNN part they used Recursive Convolution, where the weights are shared across multiple layers (Liang and Hu 2015), resulting in a much deeper model with fewer parameters. In the RNN part of the architecture they leveraged a sequential attention mechanism, which originated from applications in machine translation (Bahdanau et al. 2014; Chorowski et al. 2015). Attention in general enables the model to focus on relevant features and provides a potential way of interpretability (Xu et al. 2015b). The authors of R^2AM experimentally evaluated different variants of how the image features are fed into the RNN, including versions with stacked RNN and with the attention mechanism. In all experiments, the RNN learned a character level language model that is not constrained by a dictionary or predefined N-grams. Both, Recursive CNN and attention mechanism outperformed the current state-of-the-art on their own.

Gao et al. 2017 proposed an attention convolutional network for scene text recognition, where they replaced the recurrent layers with a stack of convolution layers and employed residual attention modules as they were proposed by Wang et al. 2017a. These attention modules come with two branches, one which generates a soft attention map that captures high-level semantic information in a bottom-up top-down fashion, and a second residual branch on which the salient map is subsequently applied. The attention mechanism helps the model to focus on contextually useful

information and suppress background noise. Additionally, dense blocks (Huang et al. 2016a) were adopted to improve the information flow in the network. Training is done classically via CTC. The authors have shown slightly higher recognition accuracies, compare to Convolutional Recurrent Neural Network (CRNN), but with fewer model parameters, less training time and much higher recognition speed (9 times faster).

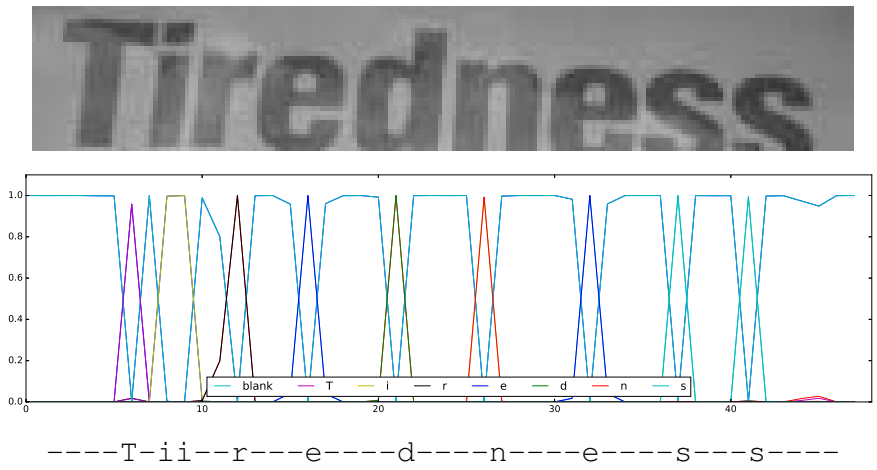


Fig. 4.2: CTC based text recognition (Bušta et al. 2017); from top to bottom, CNN input, character probabilities, output string (before removing blanks and duplicates)

4.3.2 Rectification Methods

As mentioned above, irregular word images which contain curved, perspective, distorted, multi-oriented or loosely bounded text are difficult to recognize. In the following, two interesting approaches to solve this problem will be considered.
Shi et al. 2016 build on Spatial Transformer Networks (STNs) (Jaderberg et al. 2015) [18] and learn a Thin-Plate Spline (TPS) transformation (Zagorchev and Goshtasby 2006) to rectify the input image. Therefore, their architecture mainly consists of two subnetworks, a Localization Network and a Sequence Recognition Network.
The Localization Network is a CNN that directly regresses the xy-coordinates of K fiducial points on the input image. These points are used in the grid generator to calculate the parameters of the TPS transformation and to generate the sampling point coordinates on the input image (Figure 4.3). The Recognition Network has a sequence-to-sequence learning structure with an encoder (CNN + BLSTM) and a decoder (GRU + attention mechanism). The attention mechanism is the same as in R^{2}AM and learns for each time-step a weighted sum of the BLSTM output, considered as the hidden state. The training is similar to applying a CTC and does

[18] STNs learn the parameters of a spatial transformation and sample their output from their input feature map.

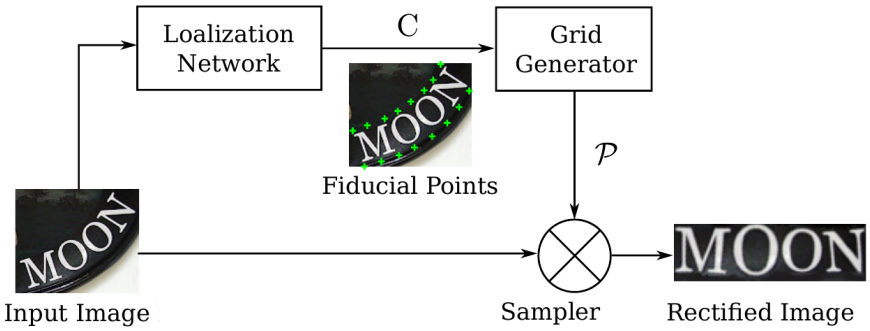


Fig. 4.3: Structure of rectification STN; fiducial points C, sampling grid $\mathcal{P}$ (Shi et al. 2016).

not require extra annotations for the fiducial points. Recognition is lexicon-based with prefix tree and beam search.

Yang et al. 2017 took the sequential attention mechanism as used in R^{2}AM and generalized it to perform 2D feature selection. Their model consists of three sub-networks, a feature extractor, a character detector and a RNN for the recognition part. The FCN-based (Long et al. 2014) character detector does pixel-wise text/non-text classification and functions as auxiliary task for learning a high-level visual representation in the feature extractor. The RNN with attention mechanism receives this high-level representation as inputs and predicts the character sequence. The model is trained on a synthetically generated dataset with character-level annotations and a multi-task loss function consisting of three terms, a pixel-wise Binary CE for the detection, Categorical CEs for the recognition and an attention alignment term to guide the attention mechanism. The ground truth for the detection task is a binary map and can be computed from the character-level bounding boxes. Similarly with the ground truth for the attention alignment which is considered as a truncated Gaussian distribution that can be calculated from a single character-level bounding box at each time-step. For the attention alignment loss, the authors propose various metrics for minimizing the difference between the ground truth and the predicted distribution. These can be an element-wise L1 or L2 loss, the KL divergence or a approximation of the Wasserstein distance[19]. The approach significantly outperformed (Shi et al. 2016) on SVT-Perspective and CUTE80.

4.3.3 Metrics for Text Recognition

The most common measure known from OCR is the Edit Distance. Formally, given two character strings a and b, the Edit Distance $\mathrm{ed}(a, b)$ is then the minimum weight-

[19]The Wasserstein (WD) or Earth Mover's distance indicates the minimum mass that needs to be transported in order to transform one distribution into another. As a loss function, the WD results in a more stable training behavior and got recent attention due to its use in adversarial training (Arjovsky et al. 2017).

series of edit operations to transform string a into string b, but usually, a special case, the Levenshtein Distance is meant. In case of the Levenshtein Distance, the edit operations are Insertions, Deletions and Substitutions and all operations have unit cost/weight[20]. In other words, the Levenshtein Distance is the number of Insertions, Deletions and Substitutions to transform string a into string b. Due to the same cost, the Levenshtein Distance distance is symmetric ($\text{ed}(a,b) = \text{ed}(b,a)$). The edit distance is usually calculated per word, and then the mean edit distance over all test samples is compared.
Other common measures are Word and Character Recognition Rate. The word recognition rate is simply the ratio of correctly recognized words to the total number of words.

$$\text{WRR} = \frac{|\text{CorrectlyRecognizedWords}|}{|\text{Words}|} \tag{4.2}$$

The character recognition rate is similar, but on character level and can be expressed by utilizing the edit distance.

$$\text{CRR} = \frac{\sum_1^{|\text{Words}|} |\text{GT}| - \text{ed}(\text{GT}, \text{RT})}{\sum_1^{|\text{Words}|} |\text{GT}|} \tag{4.3}$$

Where word-level recognition is considered and 'Words' is the set of test samples. For each word, 'GT' refers to Ground Truth and 'RT' to Recognized Text string.

4.4 End-to-End trainable Text Recognition

Instead of ad-hoc combining the state-of-the-art methods for text detection and recognition, a few recent approaches appeared which train a holistic model for scene text spotting end-to-end. The benefits of an end-to-end trainable system are obvious, no intermediate stages or hyperparameter tuning is required, the computation and the learned features can be shared between the detection and recognition, and more contextual information can be included to improve the overall performance.
All the approaches that will be investigated below are more or less difficult to train and either build on the idea of STNs (Jaderberg et al. 2015) or RPN and RoI Pooling (Ren et al. 2015).
Li et al. 2017 proposed an end-to-end trainable architecture for spotting horizontal text. Rectangular anchor boxes are defined on the last convolutional layer of a truncated VGG-16 model and a RPN with rectangular convolution kernels (Liao et al. 2016) is used to generate text region proposals at word-level. A Region Feature Encoder re-samples the RoIs from the convolutional feature map into tensors of variable width and encodes them via a LSTM layer into a fixed length repre-

[20] Substituting a character with itself has a cost of 0.

sentation[21]. Subsequently, a Text Detection Network with FC layers estimates the textness score and bounding box refinement for each proposal. After filtering the proposals via threshold and NMS, the refined RoIs are re-sampled again and encoded by the Region Feature Encoder. The resulting fixed length representations are then decoded in a Text Recognition Network into words. This is done via LSTM and a soft attention mechanism as in Lee and Osindero 2016; Shi et al. 2016. The convolutional feature extractor only needs to be evaluated once for an image. For the training process, a curriculum learning strategy (Bengio et al. 2009) is applied. Starting with a trained VGG-16 model, only the detection network is trained on images with monochrome background and words from the Syntehtic Word engine. In the next step, the recognition network is added and both are trained on the synthetic data and two more steps follow, one with the SynthText dataset an on with real-world data from ICDAR 2015, SVT and AddF2k.

The authors of Deep TextSpotter (Bušta et al. 2017) have chosen YOLOv2 (Section 3.4.8) as the foundation for their work. The YOLOv2 model is made fully convolutional by removing the FC layers, which allows the model to process high resolution scene images and leads to higher accuracy. A RPN is used to generate proposals in form of rotated rectangles. The scales and aspect ratios of the anchor boxes are determined by k-means clustering on the aggregated training set (Redmon and Farhadi 2016) and the proposals are filtered by confidence, but no NMS is applied. A STN is then learned to transforms the regions with different size and orientation into a tensor with fixed height and variable width while preserving the aspect ratio. The subsequent recognition network is fully convolutional (so it can handle the variable width) and trained with CTC. At test time, empty detections are rejected and finally NMS is applied based on the recognition score. The training process benefits from pre-training each of the two subnetworks on their own. The detector network is initialized with weights from ImageNet and pre-trained on SynthText. The recognition network is randomly initialized and pre-trained on the Synthetic Word dataset. Finally, the whole model is trained on a combination of SynthText, Synthetic Word and ICDAR 2013.

Bartz et al. 2017 came up with an end-to-end text spotting system, called SEE (short for Semi-supervised End-to-End), that does not require bounding box annotations and can be trained solely by classifying the word sequences. The detector network in SEE consists of a ResNet architecture with two attached LSTM layers for sequentially detecting one word candidate per time-step. These word candidates are estimated as the parameters of affine transformation matrices, which are then used to bilinearly sample the text area from the image feature map into the input of the recognition network. The recognition network is also based on a ResNet archi-

[21] The input of the LSTM is considered as sequence of variable length (width in this case) and fed into the LSTM. At the last time-step, the hidden state is returned as a holistic fixed-length representation of the sequence.

tecture that comes with an BLSTM on top of it. The hidden state of the BLSTM at each time-step is then classified[22] via `FC->Softmax->` and the bounding boxes are derived from the affine transformation matrices. ResNet was chosen by the authors because the skip connections can carry a strong error signal to the detector network, which is the only error signal that the detector receives. Training requires a curriculum learning strategy as well as several regularizations on the transformation matrices. Therefore, a dropout (Section 2.3.6) mechanism is used to prevent the network from learning matrices which perform excessive rotations and regularization terms concerning the aspect ratio and orientation of the sampling grid are added to the loss. The authors have shown, with comparable results on the French Street Name Signs dataset, that it's possible to train an end-to-end text spotting system without bounding box annotation. However, the system is not state-of-the-art and certainly has the potential to be improved.
Liu et al. 2018 proposed Fast Oriented Text Spotting (FOTS). Its feature extractor is also based on ResNet. To include more contextual information, high- and low-level features from multiple feature maps are fused (Lin et al. 2016) in a upsampling branch (from 1/32 to 1/4 of the input size). The upsampled feature map is then used in a RPN for the dense pixel-wise prediction of rotated bounding boxes and their textness score. The proposals are then filtered by thresholding and NMS. RoIRotate, inspired by RoIAlign in Mask R-CNN (Section 3.4.5), re-samples the proposed RoIs into axis-aligned feature maps with variable width. The re-sampling is done via affine transformation and bilinear interpolation. The necessary parameters are either calculated from estimated or ground truth bounding boxes (that is different from STNs). The subsequent recognition branch is rather basic, `3x(Conv->Conv->MaxPool)->BLSTM->FC->CTC`. The loss function is a multi-task loss, where the detection part utilizes CE loss for classification and IoU loss with rotation angle loss for bounding box regression (same in EAST (Zhou et al. 2017)). At training time, the model is initialized with pre-trained weights from ImageNet and trained on SynthText before it gets fine-tuned on real-world data. The transformation parameters used during training are calculated from the ground truth bounding boxes. Data augmentation and OHEM are applied. The approach is elegant, relative easy to train, fast (22.6 fps on NVIDIA Titan Xp) and outperformed the previous state-of-the-art on ICDAR 2015 by more then 5 percent points.

[22] The classes are the characters of the alphabet plus an additional 'blank' (see Section 2.5.7 for more details).

Chapter 5

Implementation

The goal of this work is to implement and eventually improve a state-of-the-art end-to-end scene text recognition system.
Therefore, a preexisting open source implementation of SSD[1] was updated and heavily refactored. Starting with this code basis, TextBoxes for detecting horizontal text and SegLink for oriented text were implemented. Afterwards, a recognition stage similar to CRNN was developed. For applications on a mobile robot platform, a ROS[2] node was implemented, which receives a camera image and performs the detection (and recognition) task in real-time. Support for the following datasets have been added: PASCAL VOC, MS COCO, ICDAR 2013, ICDAR 2015, SynthText, MSRA TD500, SVT and COCO Text. The whole code was written in Python, using Keras, Tensorflow and OpenCV. It is publicly available under MIT license[3].

5.1 Design Decisions

The choice to start with SegLink had multiple reasons. First, it was the best approach listed on the ICDAR 2015 challenge, for which a publication was available at this time. Second, SSD as object detector is quite versatile, has good speed-performance trade-off and manageable complexity. CRNN was chosen to build an end-to-end recognition pipeline. It has shown good results and is rather elegant due to its architectural simplicity. Another major reason for the decisions was that the author wanted to get familiar with concepts like CNN, LSTM and CTC. Python as programming language was chosen because of its large ecosystem related to scientific computing and machine learning in particular. Keras as DL framework was chosen since, on the one hand, it provides a high-level API to keep unnecessary programming overhead at a minimum, and on the other hand, it allows writing necessary code like custom layers and loss function directly with TensorFlow.

[1] https://github.com/rykov8/ssd_keras by Andrey Rykov
[2] http://www.ros.org
[3] https://github.com/mvoelk/ssd_detectors

5.2 Detection

This section tries to provide more detailed insights into the detection stage.
After updating the existing SSD code to Keras 2.0 and making the model compatible with the original architecture, a conversion script was written to import pre-trained weights from the original SSD Caffe implementation[4]. This was done since SSD can't be trained from scratch and starting with a pre-trained SSD model should converge faster than initializing with VGG-16 weights trained on ImageNet. Also, the concept of PriorBox Layer was dropped and replaced by a class called PriorMap. PriorBox Layers originate from Caffe implementation and have been recreated in the Keras implementation. They are responsible for the computation of the prior boxes defined on each location of a feature map. Their replacement with the PriorMap class was done because it makes less sense to calculate the prior boxes in NumPy, push them to the GPU where they get returned unchanged and processed in NumPy again. This is neither necessary since no backpropagation goes through them, nor intuitive and often leads to confusion. As long as the spatial dimension of the feature maps remains constant, it is also sufficient to calculate the prior boxes only once at beginning. The development of an existing procedure for data augmentation was continued in form of an InputGenerator class and another InputGenerator class was written for cropping word instances from training images.
The implementation of TextBoxes was straightforward, simply changing the kernel size and adjusting the default boxes. A computationally more efficient algorithm for NMS was adopted from Girshick et al. 2012.
The SegLink (Section 4.2.2) implementation was a bit more time-consuming. A SSD model with input sizes 512x512 analog to pre-trained models in the Caffe version was added. It has prior boxes defined on feature maps that scale their size in powers of two (64x64, 32x32, 16x16, 8x8, 4x4, 2x2, 1x1). At each location of these feature maps a default box is defined and local predictions are performed in a convolutional manner (analog to Figure 3.5). For each default box or segment $s = (x_s, y_s, w_s, h_s, \theta_s)$, the 31 values, 2 for segment confidence, 5 for segment offsets, 16 for within-layer link confidence and 8 for cross layer to the next larger feature map are estimated. The linking is illustrated in Figure B.1. Offset $(\Delta x_s, \Delta y_s, \Delta w_s, \Delta h_s, \Delta\theta_s)$ are related to the segment geometry as described in (5.1), where the factor $a_l = \gamma\frac{w_I}{w_l}$ is used to scale the segments with regard to the size of the receptive field. w_I is the width of the input image, w_l the width of the feature map and γ is a empirically

[4] https://github.com/weiliu89/caffe/tree/ssd by Wei Liu

determined constant set to 1.5.

$$\begin{aligned} x_s &= a_l \Delta x_s + x_a \\ y_s &= a_l \Delta y_s + y_a \\ w_s &= a_l \exp(\Delta w_s) \\ h_s &= a_l \exp(\Delta h_s) \\ \theta_s &= \Delta \theta_s \end{aligned} \tag{5.1}$$

The center of the prior (or anchor) box in the input image (x_a, y_a) is computed from the location on the feature map (x, y) via (5.2) and the exp function is used to make differences (e.g. 1 pixel) for smaller bounding boxes more significant in the loss than for large ones.

$$\begin{aligned} x_a &= \frac{w_I}{w_l}\left(x + \frac{1}{2}\right) \\ y_a &= \frac{w_I}{w_l}\left(y + \frac{1}{2}\right) \end{aligned} \tag{5.2}$$

For predicting bounding boxes, the detected segments and links are filtered by confidence thresholds and considered as a graph (segments as vertices and links as edges). A depth first search is used to identify the groups of connected segments. The segments of each group are combined into rotated bounding boxes $b = (x_b, y_b, w_b, h_b, \theta_b)$, as illustrated in Figure B.4. Starting with a group $\mathcal{B} = \{s^i\}_{i=1}^{|\mathcal{B}|}$ of $|\mathcal{B}|$ segments s^i, he bounding box orientation is the average of the segments orientations.

$$\theta_b = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \theta_s^i \tag{5.3}$$

Next, a straight line $y = \tan(\theta_b)x + b$ is fitted to minimize its distance to the centers of the segments in $\mathcal{B}$. All the segment centers are then projected onto the line and the extreme points (the outermost projected points on the line) (x_p, y_p) and (x_q, y_q) are determined. The remaining parameters of the bounding box are calculated as follows.

$$\begin{aligned} x_b &= \frac{1}{2}(x_p + x_q) \\ x_y &= \frac{1}{2}(y_p + y_q) \\ w_b &= \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} + \frac{1}{2}(w_p + w_q) \\ h_b &= \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} h_s^i \end{aligned} \tag{5.4}$$

For calculating local ground truth for the links and segments, a prior box is assigned to a ground truth bounding box if its center is inside the ground truth box and fulfills (5.5). If the center of a default box is inside multiple ground truth boxes, it

is assigned to that with the smallest distance. All other segments are negative. h_a is the height of the ground truth bounding box.

$$\max\left(\frac{a_l}{h_g}, \frac{h_g}{a_l}\right) \leq 1.5 \tag{5.5}$$

Links are positive if both adjacent segments are positive and assigned to the same ground truth box, otherwise they are negative. This assignment procedure is also illustrated in Figure B.2. The geometry of the local ground truth is calculated as depicted in Figure 5.1. For training, a Multi-Task Loss (2.48) with three terms (CE Loss (2.40) for segment and link confidence, L1 Loss (2.34) for segment offsets) and a procedure for OHNM of both segments and links was implemented directly with TensorFlow. For more details on SegLink see Shi et al. 2017.

Furthermore, variants of the SSD and SegLink loss were developed that replace the CE Loss and OHNM strategy by the Focal Locss (2.41). They are more elegant and focus training automatically on hard samples by dynamically weighting them based on their confidence.

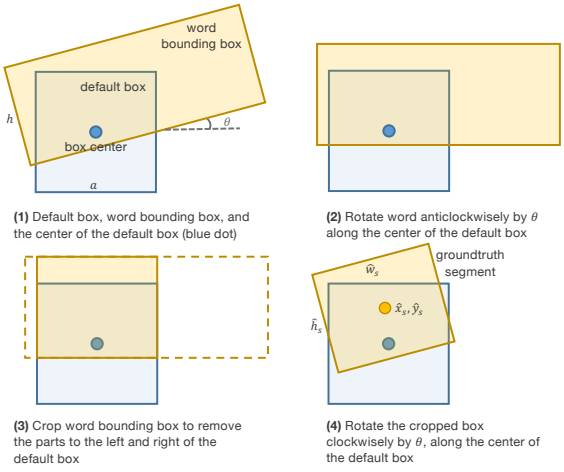


Fig. 5.1: Ground truth segment calculation; given default box and ground truth bounding box (Shi et al. 2017)

To be able to train SegLink models from scratch and not rely on pre-trained VGG-16 models, a DenseNet-based backbone architecture inspired by Shen et al. 2017 was added. The resulting models have fewer trainable parameters and should contain more text-specific features. The architecture (for 512x512 input) is fully convolutional, contains 4 Dense Blocks with growth rate 48 and has a total depth of 83 layers. Figure 5.2 shows these architecture in more detail.

5.3 Recognition

The recognition stage is less complex than the detector, but it still deserves some words.

For reading the text contained in a cropped word image, a network architecture similar to that of CRNN Shi et al. 2015, shown in Figure 4.1, was chosen. The CNN part learns text-specific features and outputs a sequence of feature vectors. The subsequent RNN part processes this sequence and learns an implicit language model. Decoding to the RNN output and training of the model is done via CTC.

Deviating from CRNN, the BLSTM layers were replaced by Bidirectional GRU

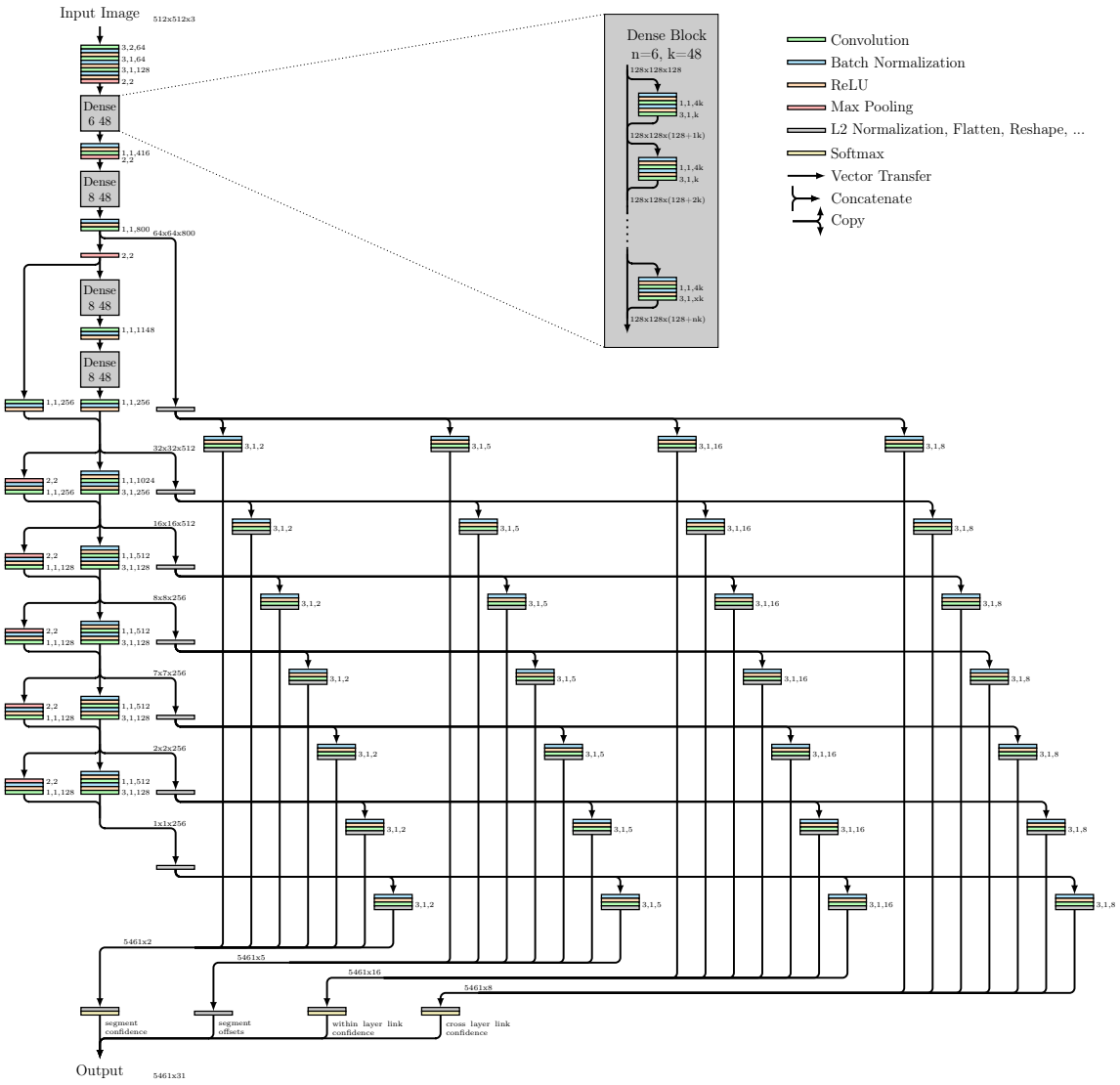


Fig. 5.2: Architecture of the SegLink-DN model; backbone with Dense Blocks for feature extraction on the left and the prediction paths on the right. The layer coloring and the structure of the prediction paths are analog to the SSD architecture in Figure 3.5. The notation related to convolution and pooling layer is 'kernel size, stride, filters' and 'pooling size, stride' respectively. Padding is always 'same'. (The figure is best viewed in PDF version.)

layers, which come with fewer trainable parameters without sacrificing performance. The input of the recognition network is a gray-scale word image with fixed height of 32 pixels and theoretically variable width. However, the width has been set to 256 pixels and the images are zero-padded/truncated to preserve the aspect ratio. Keras requires that all samples in a batch have the same size and a fixed width was just the simplest solution to this limitation. The alphabet was extended and now contains lower-case, upper-case, digits and ' `+-*.,:!?%&$~/()[]<>"\'@#_`' (practically all characters contained in the SynthText dataset, '_' indicates the blank).

Chapter 6

Experiments

For the following experiments the SynthText dataset was split into two subsets, one for training (90%, 772874 samples) and one for test/validation (10%, 85875 samples). SynthText was chosen due to its large size. Cross-validation[1] was omitted due time constraints and the large size of the dataset. All the experiments were carried out on a NVIDIA GeForce GTX 1080 Ti with 11 GB RAM.

6.1 SegLink

To get a base line, a SegLink model was trained. The backbone of the network was initialized with weights from a pre-trained SSD512 model[2] converted from Caffe and the prediction paths[3] randomly using Xavier Initialization method (Section 2.2.1). Training was done with the standard SegLink Loss and OHNM. The terms in the loss (for segment, link confidence and offset regression) were all equally weighted and the negative to positive ratio was chosen as 3.0. Adam (Section 2.2.3) was used as optimizer with an initial learning rate of 1e-3, β_1 of 0.9 and β_2 of 0.999. The batch size was 24 and no data augmentation was applied. The training process initially converged and diverged after the second epoch. This behavior was caused by a numerical issue in Adam and disappeared after increasing the epsilon parameter in Adam from 1e-8 to 1e-3. Training was repeated.
Threshold values for filtering segments and links were obtained via grid search. Therefore, the f-measure on 64 samples from the validation set was calculated for segment-link threshold combinations in steps of 0.1 (Figure 6.1a). 64 samples are probably a too small set for estimating reliable threshold values, but the computation of the grid search is relatively fast.
The resulting model shows an f-measure of 82.8% on 1024 samples[4] from the val-

[1] For k-fold cross-validation, a dataset is split into k equally sized subsets. Then one of them is held out for validation and the rest of them are used for training a model. After repeating this for all k subsets, the performance measures are averaged over the k cases.

[2] The pre-trained SSD model was initialized with VGG-16 weights trained on ImageNet and fine-tuned on Pascal VOC.

[3] The prediction paths of the SegLink architecture have the same structure as those in Figure 3.5.

[4] The size of the validation set was chosen so small due to the time required for evaluation.

idation set. This value can not be directly compared with the 75% shown by the SegLink authors on the ICDAR 2015 test set, but it indicates that the implementation is working. The original SegLink was trained on SynthText and fine-tuned on the ICDAR 2015 training set. Since ICDAR 2015 is more difficult compared to SynthText and the test set in our case is from the same distribution as the training set, which is not the case when fine-tuned and evaluated on ICDAR 2015, makes an f-measure increase of 8 percent points quite plausible. For the rest of the work, this model is referred to as 'SegLink' and further details regarding the model are summarized in Table 6.1.

Model	SegLink	SegLink-DN	SegLink-DN-FL
Parameters	~24M	**~13M**	**~13M**
Memory usage	**420 MB**	1.50 GB	1.50 GB
Batch size	24	6	6
Epochs	2	2	2
Iterations	~24k	~257k	~257k
Segment threshold	0.67	0.57	0.55
Link threshold	0.55	0.22	0.35
Precision	84.6	89.6	**93.1**
Recall	81.2	**89.6**	89.1
F-measure	82.8	89.6[5]	**91.1**
Prediction time	**27.8 ms**	44.1 ms	44.1 ms
Prediction + decoding time	**32.7 ms**	45.8 ms	45.8 ms
Frame rate	**30.6 fps**	21.8 fps	21.8 fps

Tab. 6.1: Comparison of text detection models; training and validation was done on the SynthText dataset split by 0.9, precision, recall and f-measure were calculated on 1024 samples from the validation set and are given in percent, segment and link threshold were estimated by examination of plots as shown in Figure 6.1

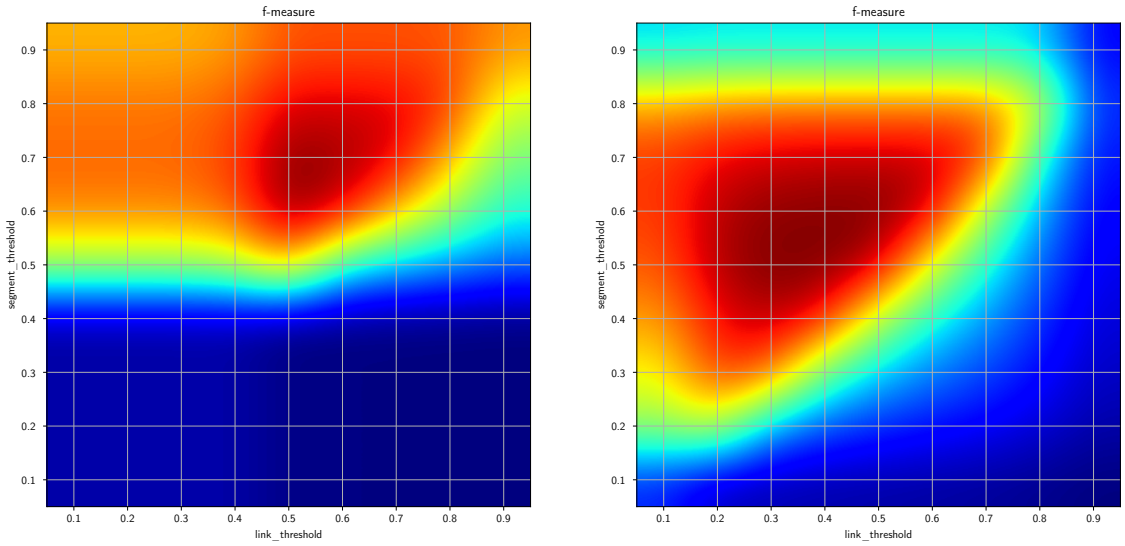
.

6.2 SegLink with DenseNet Backbone

In the next step a SegLink model with DenseNet backbone was trained from scratch. The complete model was randomly initialized and the training parameters were kept the same as those for the SegLink model, except the batch size had to be reduced drastically.

[5]Yes, the values are the same (precision=0.896121, recall=0.895801).

This reduction in batch size is caused by the many skip connections in the Dense Blocks that require much more gradient data to be memorized during backpropagation. Compared to the SegLink model, the memory consumption for a batch of size 1 increased from 420 MB to 1.50 GB, which leads with 11 GB memory to a maximum batch size of 6. A virtual batch size for which the gradient of multiple physical batches gets accumulated before the backward pass is performed would eliminate this limitation, but this is not (yet) implemented in Keras and also not within the scope of this work.
As expected, the training process was highly stochastic due to the small batch size, but also has shown better results. The models obtained after each epoch were evaluated as in the previous experiment and the model after the second epoch yielded the highest f-measure of 89.6%. This is a increase by 6.8 percent points, whereby the model was trained from scratch and the number of parameters has been reduced to almost half. More details on the model, which is now referred to as 'SegLink-DN', can be found in Table 6.1.



(a) Grid search SegLink model

(b) Grid search SegLink-DN-FL model

Fig. 6.1: Gird search for estimating segment and link thresholds; heat-map via bicubic interpolation.

6.3 SegLink with DenseNet Backbone and Focal Loss

To see if the performance of the SegLink-DN model can be further increased by more efficiently focusing on hard samples, another model using the Focal Loss instead of CE Loss and OHNM was trained, which is in the following referred to as 'SegLink-DN-FL'.
The focusing parameter γ of the focal loss was set to 2.0 for both segments and links.

The weighting of the different loss terms have been adjusted to better scale the focal loss ($\lambda_{\text{segments}}$=100.0, λ_{offsets}=1.0 and λ_{links}=100.0). All other training parameters were adopted from the SegLink-DN experiment. The training loss over the first two epochs is shown in Figure 6.2. It can be clearly seen how stochastic the training process is due to the small batch size. Also, a sudden drop in of the loss can be observed at ~3600 iterations, which is most likely caused by a local optimum in the segment classification (`seg_conf_loss` in plots).

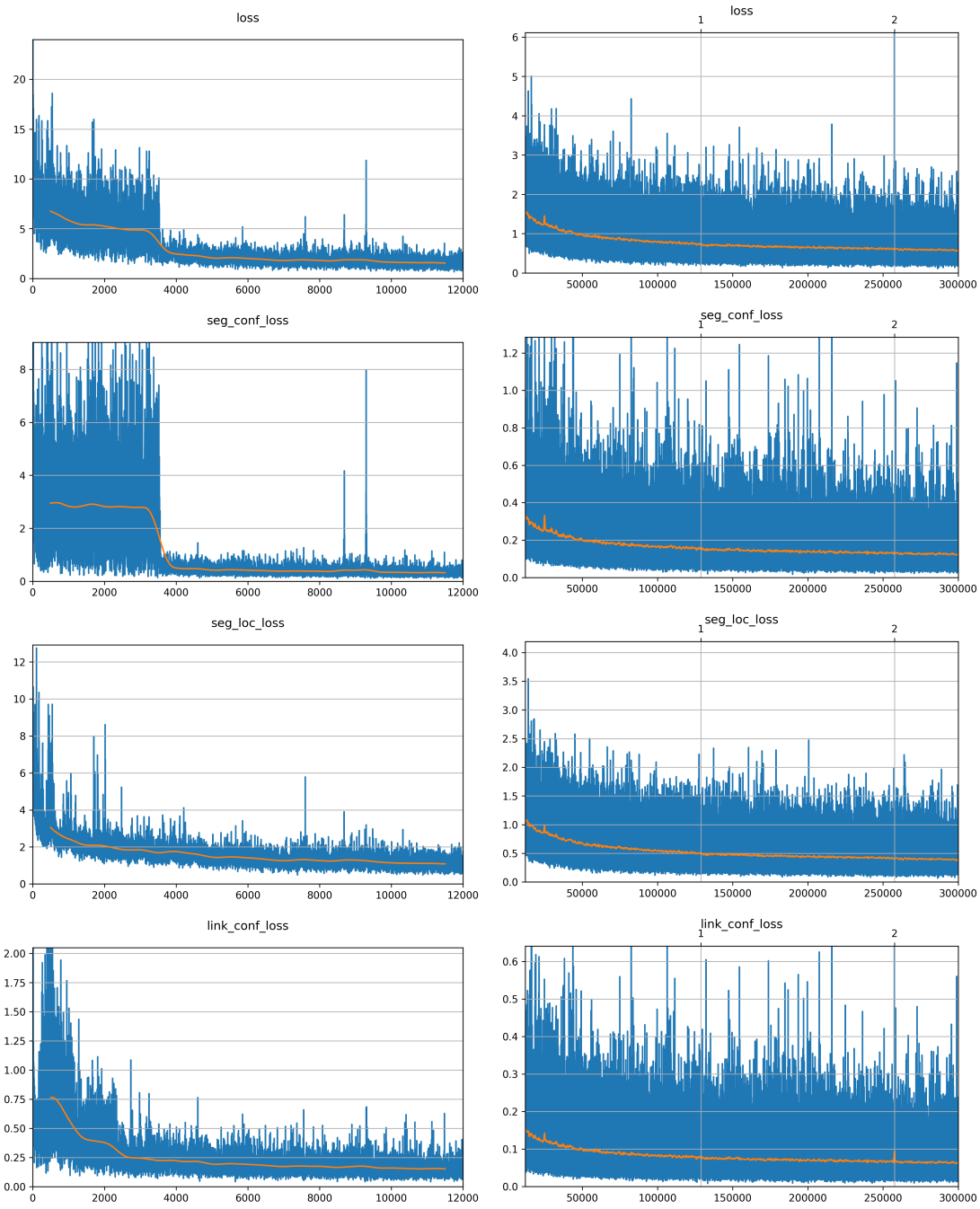


Fig. 6.2: Training loss SegLink-DN-FL model; bottom axis indicates iterations, top axis epochs. The right plots are the continuation of the left with appropriate scale. The orange line is a moving average with window size of 1000 iterations.

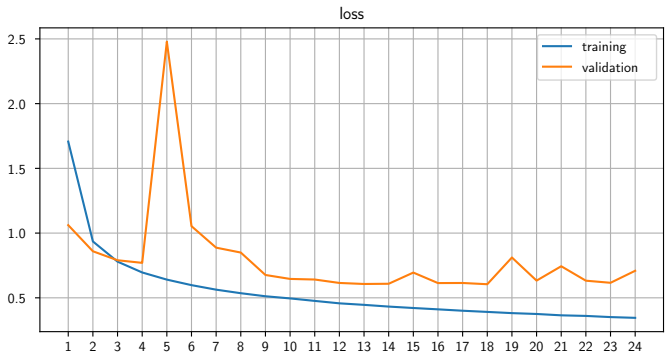


Fig. 6.3: Training and validation loss of a SegLink-DN-FL model after each epoch; the training loss is averaged during training and thus initially greater than the validation loss.

The grid search for finding good values for segment and link thresholds was performed as in the experiments above and is shown in Figure 6.1b. It looks more natural compared to that of the SegLink in Figure 6.1a, which was initialized with ImageNet feature and trained with OHNM.
In terms of f-measure, the focal loss improved the SegLink-DN model by 1.5 percent points, resulting in a total increase of f-measure, compared to the SegLink model, of 8.3 percent points. The gain in f-measure is caused by a high precision of 93.1%, but further experiments would be required to find more optimal values for γ and λ.
The deeper architecture of the SegLink-DN model has its price. Since the forward pass takes longer, the frame rate for real-time detection drops by almost a third compared to the SegLink model.
Since the SynthText dataset is relatively large, taking model snapshots and doing validation after each epoch is not the best practice. Therefore, another SegLink-DN-FL model was trained on a quarter of the training/validation set. The training and validation loss is plotted in Figure 6.3. It shows little overfitting, but the validation loss peaks after the 5th epoch, which should be further investigated. The model obtained after the 12th epoch exhibits with 92.2% an even higher f-measure on the same validation set as the model listed in Table 6.1. Figure 6.4 shows the precision, recall and f-measure evaluated on the local predictions for segments and links over the epochs. The training plots show plausibly that the model first learns the segments and then the linking. The high precision and gradually increasing recall also indicate that it first learns to detect the easy instances and later the hard ones. The validation plots give a first hint, that the peak in the validation loss is caused by the linking.
It may be a better idea and should be considered for further experiments that validation can be done regularly after a certain number of iterations. Also, early stopping (Section 2.2.5) should be taken into consideration.
In addition, experiments in which all ReLU activations were replaced by Leaky ReLU

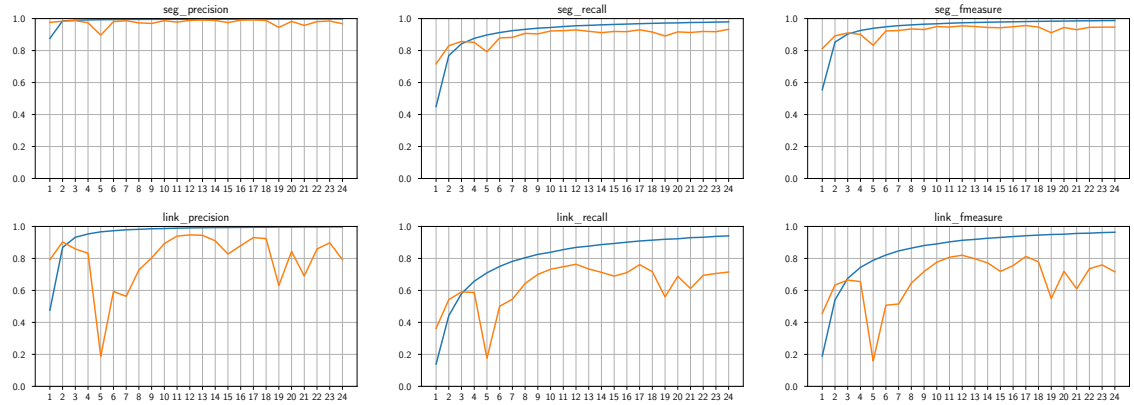


Fig. 6.4: Precision, recall and f-measure during training of a SegLink-DN-FL model; evaluated on the top 100 local predictions for segments and links.

were carried out, but they showed no significant improvement in neither convergence speed nor performance.

To get a more qualitative impression of detection stage, example detections from the SynthText validation set (Figure C.1) as well as from real-world images (Figure C.2 and C.3) are attached in Appendix C.

The sample images also reveal some major issues related to the SegLink approach and the training data.

Textures that have a similar structure to letters get falsely detected as text instances (e.g. '//////' or 'HHHHHHHH'). A possible remedy could be the inclusion of more contextual information. This can either be done by smaller default boxes, so that the corresponding receptive field would contain more of the surrounding, or by an up-sampling path in the architecture as in EAST (Zhou et al. 2017).

Another problem that arises from the construction of the bounding boxes is caused by false positive links between nearby segment groups. This case can be observed in the debugging images (Figure B.3, B.4 and B.5) on the bottom left. The final bounding box is constructed by averaging the orientation and height of the segments in a group and placing it on the line best fitting the segment positions. This resulting in one, instead of two, bounding box of appropriate height, but between the two word instances. A possible solution to this could be a post-processing of the grouping, which disconnects all segments in a group that have a distance to each other, orthogonal to the line, that is larger than a certain value (e.g. 0.7 of the averaged height).

Since the input resolution for the experiments was only 512x512, the detection of small text instances, as in the real-world images (Figure C.2 and C.3), is a serious problem, that can be easily solved by increasing the input of the network without training it again (the architecture is fully convolutional). This has also to be done to get decent results on the ICDAR 2015 test set. It may also be desirable to have an input resolution that is non-square and more natural (e.g. 4:3, 16:9). In this case

training a new model would be recommended, instead of using the stretched filters. The new model would also require a larger input size during training to maintain the cross-layer linking, but a larger input size would result in a higher GPU memory consumption and a further reduction of the batch size. A solution to this problem may be the removal of the last feature map, so that the cross-layer linking could be maintained with a lower input size.
The detection on the real-world images also completely fails on heavily curved and vertical text instances, simply for the reason that similar instances do not occur in the training data. The vertical instances may be addressed by data augmentation (rotated images) and for the curved text, it should be possible to modify the SynthText synthetic text engine to produce appropriate samples with curved ground truth.
The detector also recognizes symbols that are not contained in the training set. A possible way to suppress them would be based on the score of a subsequent recognition stage, as it was done in TextBoxes++ (Liao et al. 2018).

6.4 CRNN with GRU

The modified CRNN architecture as described in Section 5.3 was trained on cropped word images from the SynthText training set and the resulting model is in the following referred to as 'CRNN-GRU' and will be compared with its LSTM baseline 'CRNN-LSTM'.
The input images were converted to gray-scale for both models. Initialization was done randomly via Xavier method and SGD was used as optimizer. The optimization parameters were chosen rather intuitive: initial learning rate of 0.01, learning rate decay of 1e-6, which is actually too low, Nesterov Momentum of 0.9 and a batch size of 128. The training process showed initial instability. To fix this, the norm of the gradient was clipped with a constant value of 5. Afterwards, the training converged and was continued for 400k iterations. Best path decoding (Section 2.5.7) was used to decode the output sequence.
Evaluation was done on the first 100k cropped word images from the validation set. The calculated values for mean edit distance and recognition rates are listed in Table 6.2.
Qualitative recognition results are attached in Appendix C Figure C.4.
For the subjectively low word recognition rate, two primary causes could be identified. The first is that some samples in the SynthText dataset are quite hard to read even for humans, especially when converted into gray-scale. The second is that the SynthText dataset contains some samples in uppercase letters that are labeled with lowercase. This affects the test results negatively in the training case by learning this pattern and in the test case by not matching it.

Model	CRNN-LSTM	CRNN-GRU
Parameters	8745k	**7957k**
Memory usage	41.17 MB	**38.17 MB**
Batch Size	128	128
Iterations	300k	300k
Alphabet Length	86	86
Mean Edit Distance	**0.346**	0.350
Mean Normalized Edit Distance	**0.084**	0.085
Character Recognition Rate	**0.913**	0.912
Word Recognition Rate	**0.853**	0.851
Prediction time	38.2 ms	**30.4 ms**
Prediction + decoding time	38.5 ms	**30.6 ms**

Tab. 6.2: Text recognition models; training and validation was done on cropped word images from the SynthText dataset split by 0.9, edit distance and recognition rates were calculated on the first 100k samples from the validation set, prediction time was measured on a batch of 16 text instances.

6.5 End-To-End Recognition

An end-to-end scene text recognition pipeline was built by concatenating the SegLink-DN-FL model and the CRNN-GRU model.
Therefore, the bounding boxes predicted by the SegLink-DN-FL model are used for cropping their content from the input image, which is then converted to gray-scale and fed as cropped word images into the CRNN-GRU model.
Qualitative end-to-end recognition examples on the SynthText validation subset are shown in Figure C.5.
The most frequent cause for errors in this setup is the failure of the detector. If the detection stage predicts bounding boxes which contain none or partial text instances, then the recognition stage has no chance to predict anything useful. One of the end-to-end trainable text spotting systems mentioned in Section 4.4 should be a promising approach to counteract this issue.
The proposed end-to-end recognition pipeline was also tested on the real-world images, as shown in Figure C.6, where it revealed all the problems mentioned earlier and some further limitations. The recognition stage is unable to recognize German umlauts because they are not included in the training data and the alphabet. In addition, long words and failed word separation lead to large bounding boxes that are too wide for the 256 pixel wide input of the CRNN-GRU model.

Chapter 7

Conclusion and Outlook

In this work, a basic introduction to Feed-Forward Neural Networks and Recurrent Neural Networks was given, followed by an overview on state-of-the-art concepts on Convolutional Neural Networks-based object detection. After that, various recent concepts and ideas related to text detection and recognition were discussed.
The main contribution of this work is the implementation of end-to-end scene text recognition pipeline as a composition of a detection and a recognition stage.
For the detection of arbitrary oriented text instances, a prior approach based on generic object detection was implemented and further improved. Therefore, a network architecture based on DenseNet and the recently published concept of Focal Loss were adopted. In the subsequent experiments, a model was trained from scratch, which, unlike the prior approach, does not depend on a pre-trained VGG-16 model and has merely half as many model parameters. In addition, the new model increased the f-measure by 8.3 percentage points.
A prior approach was also adopted for the recognition state, which considers the text recognition task as a sequence-to-sequence learning problem. The architecture of the implemented recognition network consists of convolutional and recurrent layers, where the recurrent LSTM layers were replaced by GRU. This minor change results in a model that has 9% fewer parameters compared to the original LSTM variant and these without sacrificing performance.
The system has also shown good real-time performance, ~21.8 fps on the detection task and ~13.1 fps on the end-to-end recognition. One has to mention, that the prior detection stage runs at 30.6 fps and that this reduction in frame rate is caused by the much deeper network architecture with 83 instead of 25 layers.
Due to the rapid progress in the field and new superior approaches that appeared during the time of this work, the proposed approach is no longer state-of-the-art. Anyway, there are a lot of possibilities for further research related to the presented approach and newer ideas.
The results of the detection stage could be improved by involving more contextual information. This can be done by introducing an upsampling path in the architecture (Deng et al. 2018; Zhou et al. 2017). Also, for real-time detection, temporal contextual information could be considered. The architecture could be improved

by using Recursive Convolution (Liang and Hu 2015) for better feature recycling or Deformable Convolution (Dai et al. 2017) for learning rotation invariant features. Further experiments with different growth rates and different backbone architectures (ResNet etc.) would be interesting to see. A careful tuning of the hyperparameters would probably lead to better results, but even simpler approaches (Liao et al. 2018) combined with learned non-maximum suppression (Hosang et al. 2017) may have the potential to outperform the presented approach.

The recognition stage could be speeded up by replacing the RNN based sequence-to-sequence learning mechanism with one based on CNN (Gao et al. 2017). Experiments with beam search and a language model for Connectionist Temporal Classification would be interesting.

Another issue that can be tackled are the irregular text instances, which contain curved or perspectively distorted text. They could either be addressed by an attention mechanism (Yang et al. 2017) or by means of STNs (Liu et al. 2016b; Shi et al. 2016).

Various attention mechanisms (Bahdanau et al. 2014; Wang et al. 2017a; Xu et al. 2015b) could also improve the whole pipeline, either in the detection or in the recognition stage. An end-to-end trainable solution (Section 4.4), either based on Spatial Transformer Networks or RoI Pooling layers would also be favorable.

Weakly supervised[1] approaches (Hu et al. 2017; Tian et al. 2017), where the later mentioned also consider non-sequential math expressions and script identification in a multi-language setup (Patel et al. 2018), would also be interesting topics that deserve further attention.

From a broader perspective, the supervised learning approach has its limitations due to the massive amount of labeled data required. Recent findings related to semi-supervised learning and generative models such as GANs (Arjovsky et al. 2017; Fisher et al. 2018; Goodfellow et al. 2014) may be a remedy for this problem. GANs are not only a good idea for improving the quality of scene images (Kupyn et al. 2017), but may also be used in a semi-supervised learning scenario (Adiwardana et al. 2017; Souly et al. 2017). Also, reconstruction as a general proxy task for learning better features (Balestriero et al. 2018, 2017), or text recognition as one of multiple tasks (Kirkpatrick et al. 2017) in a more general vision system may be an interesting direction for further research.

One final remark on the implementation of the networks. It was all done in Keras and TensorFlow, but a framework like PyTorch, which allows imperative programming, would have been a better decision, especially due to the better debugging possibilities.

In conclusion, one can say that scene text recognition is a nice application case if one

[1] Weakly supervised learning is the case in which the samples are only partially labeled (e.g. in object detection if only a few objects have bounding boxes annotations).

wants to get a comprehensive background in the field of artificial neural networks.

Bibliography

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (Mar. 2016). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *arXiv:1603.04467 [cs]*. URL: http://arxiv.org/abs/1603.04467.

Adiwardana, D. D. F., A. Matsukawa, and J. Whang (2017). *Using Generative Models for Semi-Supervised Learning*, p. 7.

Arjovsky, M., S. Chintala, and L. Bottou (Jan. 2017). "Wasserstein GAN". In: *arXiv:1701.07875 [cs, stat]*. URL: http://arxiv.org/abs/1701.07875.

Bahdanau, D., K. Cho, and Y. Bengio (Sept. 2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv:1409.0473 [cs, stat]*. URL: http://arxiv.org/abs/1409.0473.

Bai, S., J. Z. Kolter, and V. Koltun (Mar. 2018). "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In: *arXiv:1803.01271 [cs]*. URL: http://arxiv.org/abs/1803.01271.

Balestriero, R. and R. G. Baraniuk (2017). *A Spline Theory of Deep Learning*. URL: http://www.ece.rice.edu/~richb/dox/files/splineNet/Template_Matching.pdf (visited on 01/25/2018).

Balestriero, R., H. Glotin, and R. Baraniuk (Feb. 2018). "Semi-Supervised Learning Enabled by Multiscale Deep Neural Network Inversion". In: *arXiv:1802.10172 [cs, stat]*. URL: http://arxiv.org/abs/1802.10172.

Balestriero, R., V. Roger, H. G. Glotin, and R. G. Baraniuk (Nov. 2017). "Semi-Supervised Learning via New Deep Network Inversion". In: *arXiv:1711.04313 [cs, stat]*. URL: http://arxiv.org/abs/1711.04313.

Bartz, C., H. Yang, and C. Meinel (Dec. 2017). "SEE: Towards Semi-Supervised End-to-End Scene Text Recognition". In: *arXiv:1712.05404 [cs]*. URL: http://arxiv.org/abs/1712.05404.

Bastien, F., P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio (Nov. 2012). "Theano: new

features and speed improvements". In: *arXiv:1211.5590 [cs]*. URL: http://arxiv.org/abs/1211.5590.

Baydin, A. G., B. A. Pearlmutter, A. A. Radul, and J. M. Siskind (Feb. 2015). "Automatic differentiation in machine learning: a survey". In: *arXiv:1502.05767 [cs]*. URL: http://arxiv.org/abs/1502.05767.

Bengio, Y., J. Louradour, R. Collobert, and J. Weston (2009). "Curriculum learning". In: ACM Press, pp. 1–8. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553380.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 0-387-31073-8.

Bojarski, M., A. Choromanska, K. Choromanski, B. Firner, L. Jackel, U. Muller, and K. Zieba (Nov. 2016). "VisualBackProp: efficient visualization of CNNs". In: *arXiv:1611.05418 [cs]*. URL: http://arxiv.org/abs/1611.05418.

Bosch, A., A. Zisserman, and X. Munoz (Oct. 2007). "Image Classification using Random Forests and Ferns". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1–8. DOI: 10.1109/ICCV.2007.4409066. URL: http://ieeexplore.ieee.org/document/4409066.

Botev, A., G. Lever, and D. Barber (July 2016). "Nesterov's Accelerated Gradient and Momentum as approximations to Regularised Update Descent". In: *arXiv:1607.01981 [cs, stat]*. URL: http://arxiv.org/abs/1607.01981.

Bronstein, M. M., J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst (July 2017). "Geometric deep learning: going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4, pp. 18–42. ISSN: 1053-5888. DOI: 10.1109/MSP.2017.2693418. URL: http://arxiv.org/abs/1611.08097.

Bušta, M., L. Neumann, and J. Matas (Oct. 2017). "Deep TextSpotter: An End-to-End Trainable Scene Text Localization and Recognition Framework". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2223–2231. DOI: 10.1109/ICCV.2017.242. URL: http://ieeexplore.ieee.org/document/8237504.

Canziani, A., A. Paszke, and E. Culurciello (May 2016). "An Analysis of Deep Neural Network Models for Practical Applications". In: *arXiv:1605.07678 [cs]*. URL: http://arxiv.org/abs/1605.07678.

Cao, G., X. Xie, W. Yang, Q. Liao, G. Shi, and J. Wu (Sept. 2017). "Feature-Fused SSD: Fast Detection for Small Objects". In: *arXiv:1709.05054 [cs]*. URL: http://arxiv.org/abs/1709.05054.

Ch'ng, C. K. and C. S. Chan (Oct. 2017). "Total-Text: A Comprehensive Dataset for Scene Text Detection and Recognition". In: *arXiv:1710.10400 [cs]*. URL: http://arxiv.org/abs/1710.10400.

Chen, Y., J. Li, B. Zhou, J. Feng, and S. Yan (Dec. 2017). "Weaving Multi-scale Context for Single Shot Detector". In: *arXiv:1712.03149 [cs]*. URL: http://arxiv.org/abs/1712.03149.

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*. URL: http://arxiv.org/abs/1406.1078.

Chollet, F. (Oct. 2016). "Xception: Deep Learning with Depthwise Separable Convolutions". In: *arXiv:1610.02357 [cs]*. URL: http://arxiv.org/abs/1610.02357.

Chorowski, J., D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio (June 2015). "Attention-Based Models for Speech Recognition". In: *arXiv:1506.07503 [cs, stat]*. URL: http://arxiv.org/abs/1506.07503.

Collobert, R., S. Bengio, and J. Mariéthoz (2002). "Torch: A Modular Machine Learning Software Library". In:

Cong Yao, Xiang Bai, Wenyu Liu, Yi Ma, and Zhuowen Tu (June 2012). "Detecting texts of arbitrary orientations in natural images". In: IEEE, pp. 1083–1090. ISBN: 978-1-4673-1228-8 978-1-4673-1226-4 978-1-4673-1227-1. DOI: 10.1109/CVPR.2012.6247787. URL: http://ieeexplore.ieee.org/document/6247787.

CS231n Convolutional Neural Networks for Visual Recognition (2017). URL: http://cs231n.github.io (visited on 05/06/2017).

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4, pp. 303–314.

Dai, J., Y. Li, K. He, and J. Sun (May 2016). "R-FCN: Object Detection via Region-based Fully Convolutional Networks". In: *arXiv:1605.06409 [cs]*. URL: http://arxiv.org/abs/1605.06409.

Dai, J., H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei (Mar. 2017). "Deformable Convolutional Networks". In: *arXiv:1703.06211 [cs]*. URL: http://arxiv.org/abs/1703.06211.

Dalal, N. and B. Triggs (June 2005). "Histograms of oriented gradients for human detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1, pp. 886–893. DOI: 10.1109/CVPR.2005.177. URL: http://ieeexplore.ieee.org/document/1467360.

Deng, D., H. Liu, X. Li, and D. Cai (Jan. 2018). "PixelLink: Detecting Scene Text via Instance Segmentation". In: *arXiv:1801.01315 [cs]*. URL: http://arxiv.org/abs/1801.01315.

Doerr, A., C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe (Jan. 2018). "Probabilistic Recurrent State-Space Models". In: *arXiv:1801.10395 [stat]*. URL: http://arxiv.org/abs/1801.10395.

Dollár, P., R. Appel, S. Belongie, and P. Perona (Aug. 2014). "Fast Feature Pyramids for Object Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.8, pp. 1532–1545. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2014.2300479. URL: http://ieeexplore.ieee.org/document/6714453.

Dominguez, A. (Jan. 2015). "A History of the Convolution Operation [Retrospectroscope]". In: *IEEE Pulse* 6.1, pp. 38–49. ISSN: 2154-2287. DOI: 10.1109/MPUL.2014.2366903. URL: http://ieeexplore.ieee.org/document/7015692.

Dosovitskiy, A. and T. Brox (2016). "Inverting visual representations with convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4829–4837. URL: http://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Dosovitskiy_Inverting_Visual_Representations_CVPR_2016_paper.html.

Duchi, J., E. Hazan, and Y. Singer (Jul 2011). "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12, pp. 2121–2159.

Dumoulin, V. and F. Visin (Mar. 2016). "A guide to convolution arithmetic for deep learning". In: *arXiv:1603.07285 [cs, stat]*. URL: http://arxiv.org/abs/1603.07285.

Dvornik, N., K. Shmelkov, J. Mairal, and C. Schmid (Aug. 2017). "BlitzNet: A Real-Time Deep Network for Scene Understanding". In: *arXiv:1708.02813 [cs]*. URL: http://arxiv.org/abs/1708.02813.

Elman, J. L. (1990). "Finding structure in time". In: *Cognitive Science* 14.2, pp. 179–211.

Epshtein, B., E. Ofek, and Y. Wexler (June 2010). "Detecting Text in Natural Scenes with Stroke Width Transform". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2963–2970. DOI: 10.1109/CVPR.2010.5540041. URL: http://ieeexplore.ieee.org/document/5540041.

Everingham, M., L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman (2010). "The pascal visual object classes (voc) challenge". In: *International Journal of Computer Vision* 88.2, pp. 303–338. URL: http://link.springer.com/article/10.1007/s11263-009-0275-4.

Felzenszwalb, P., R. Girshick, D. McAllester, and D. Ramanan (Sept. 2013). "Visual object detection with deformable part models". In: *Communications of the ACM* 56.9, p. 97. ISSN: 00010782. DOI: 10.1145/2500468.2494532.

Fisher, C. K., A. M. Smith, and J. R. Walsh (Apr. 2018). "Boltzmann Encoded Adversarial Machines". In: *arXiv:1804.08682 [cs, stat]*. URL: http://arxiv.org/abs/1804.08682.

Fu, C.-Y., W. Liu, A. Ranga, A. Tyagi, and A. C. Berg (Jan. 2017). "DSSD : Deconvolutional Single Shot Detector". In: *arXiv:1701.06659 [cs]*. URL: http://arxiv.org/abs/1701.06659.

Gao, Y., Y. Chen, J. Wang, and H. Lu (Sept. 2017). "Reading Scene Text with Attention Convolutional Sequence Modeling". In: *arXiv:1709.04303 [cs]*. URL: http://arxiv.org/abs/1709.04303.

Geiger, A., P. Lenz, and R. Urtasun (June 2012). "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 3354–3361. ISBN: 978-1-4673-1228-8 978-1-4673-1226-4 978-1-4673-1227-1. DOI: 10.1109/CVPR.2012.6248074. URL: http://ieeexplore.ieee.org/document/6248074.

Gers, F. A. and J. Schmidhuber (2000). "Recurrent Nets that Time and Count". In: *Proceedings of the International Joint Conference on Neural Network (IJCNN)*. Como, Italy.

Girshick, R. B., P. F. Felzenszwalb, and D. McAllester (2012). *Discriminatively Trained Deformable Part Models, Release 5*. URL: http://people.cs.uchicago.edu/~rbg/latent-release5.

Girshick, R. (Apr. 2015). "Fast R-CNN". In: *arXiv:1504.08083 [cs]*. URL: http://arxiv.org/abs/1504.08083.

Girshick, R., J. Donahue, T. Darrell, and J. Malik (Nov. 2013). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *arXiv:1311.2524 [cs]*. URL: http://arxiv.org/abs/1311.2524.

Girshick, R., F. Iandola, T. Darrell, and J. Malik (Sept. 2014). "Deformable Part Models are Convolutional Neural Networks". In: *arXiv:1409.5403 [cs]*. URL: http://arxiv.org/abs/1409.5403.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

GNU Image Manipulation Program (2018). URL: http://docs.gimp.org/en (visited on 02/22/2018).

Goodfellow, I. J., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press. ISBN: 978-0-262-03561-3. URL: http://www.deeplearningbook.org.

Goodfellow, I. J., Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet (Dec. 2013a). "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". In: *arXiv:1312.6082 [cs]*. URL: http://arxiv.org/abs/1312.6082.

Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (June 2014). "Generative Adversarial Networks". In: *arXiv:1406.2661 [cs, stat]*. URL: http://arxiv.org/abs/1406.2661.

Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (Feb. 2013b). "Maxout Networks". In: *arXiv:1302.4389 [cs, stat]*. URL: http://arxiv.org/abs/1302.4389.

Graham, B. (Dec. 2014). "Fractional Max-Pooling". In: *arXiv:1412.6071 [cs]*. URL: http://arxiv.org/abs/1412.6071.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Vol. 385. Studies in Computational Intelligence. Springer. ISBN: 978-3-642-24796-5. DOI: 10.1007/978-3-642-24797-2.

Graves, A., S. Fernández, F. Gomez, and J. Schmidhuber (2006). "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks". In: *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. ACM, pp. 369–376.

Graves, A., S. Fernandez, and J. Schmidhuber (May 2007). "Multi-Dimensional Recurrent Neural Networks". In: *arXiv:0705.2011 [cs]*. URL: http://arxiv.org/abs/0705.2011.

Graves, A. and J. Schmidhuber (2005). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural Networks* 18.5, pp. 602–610.

Graves, A., G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis (Oct. 2016). "Hybrid Computing using a Neural Network with Dynamic External Memory". In: *Nature* 538.7626, pp. 471–476. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature20101. URL: http://www.nature.com/doifinder/10.1038/nature20101.

Greff, K., R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber (Mar. 2015). "LSTM: A Search Space Odyssey". In: *arXiv:1503.04069 [cs]*. URL: http://arxiv.org/abs/1503.04069.

Griewank, A. (2012). "Who Invented the Reverse Mode of Differentiation?" In: *Documenta Mathematica, Extra Volume ISMP*, pp. 389–400.

Güler, R. A., N. Neverova, and I. Kokkinos (Feb. 2018). "DensePose: Dense Human Pose Estimation In The Wild". In: *arXiv:1802.00434 [cs]*. URL: http://arxiv.org/abs/1802.00434.

Gupta, A., A. Vedaldi, and A. Zisserman (Apr. 2016). "Synthetic Data for Text Localisation in Natural Images". In: *arXiv:1604.06646 [cs]*. URL: http://arxiv.org/abs/1604.06646.

Hannun, A. (Nov. 2017). "Sequence Modeling with CTC". In: *Distill* 2.11, e8. ISSN: 2476-0757. DOI: 10.23915/distill.00008. URL: http://distill.pub/2017/ctc.

He, K., G. Gkioxari, P. Dollár, and R. Girshick (Mar. 2017). "Mask R-CNN". In: *arXiv:1703.06870 [cs]*. URL: http://arxiv.org/abs/1703.06870.

He, K., X. Zhang, S. Ren, and J. Sun (2014). "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *arXiv:1406.4729 [cs]* 8691, pp. 346–361. DOI: 10.1007/978-3-319-10578-9_23. URL: http://arxiv.org/abs/1406.4729.

He, K., X. Zhang, S. Ren, and J. Sun (Dec. 2015a). "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]*. URL: http://arxiv.org/abs/1512.03385.

He, K., X. Zhang, S. Ren, and J. Sun (Feb. 2015b). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *arXiv:1502.01852 [cs]*. URL: http://arxiv.org/abs/1502.01852.

He, K., X. Zhang, S. Ren, and J. Sun (Mar. 2016). "Identity Mappings in Deep Residual Networks". In: *arXiv:1603.05027 [cs]*. URL: http://arxiv.org/abs/1603.05027.

Hinton, G. (2012a). *Rmsprop: Divide the gradient by a running average of its recent magnitude*. Coursera: Neural Networks for Machine Learning. URL: http://www.coursera.org/learn/neural-networks/lecture/YQHki/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude.

Hinton, G. E. (2012b). "A Practical Guide to Training Restricted Boltzmann Machines". In: *Neural Networks: Tricks of the Trade*. Ed. by G. Montavon, G. B. Orr, and K.-R. Müller. Vol. 7700. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 599–619. ISBN: 978-3-642-35288-1 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_32. URL: http://link.springer.com/10.1007/978-3-642-35289-8_32.

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

Hong, S., B. Roh, K.-H. Kim, Y. Cheon, and M. Park (Nov. 2016). "PVANet: Lightweight Deep Neural Networks for Real-time Object Detection". In: *arXiv:1611.08588 [cs]*. URL: http://arxiv.org/abs/1611.08588.

Hornik, K. (1991). "Approximation Capabilities of Muitilayer Feedforward Networks". In: p. 7.

Hosang, J., R. Benenson, and B. Schiele (May 2017). "Learning non-maximum suppression". In: *arXiv:1705.02950 [cs]*. URL: http://arxiv.org/abs/1705.02950.

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (Apr. 2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv:1704.04861 [cs]*. URL: http://arxiv.org/abs/1704.04861.

Hu, H., C. Zhang, Y. Luo, Y. Wang, J. Han, and E. Ding (Aug. 2017). "WordSup: Exploiting Word Annotations for Character based Text Detection". In: *arXiv:1708.06720 [cs]*. URL: http://arxiv.org/abs/1708.06720.

Huang, G., Z. Liu, K. Q. Weinberger, and L. van der Maaten (Aug. 2016a). "Densely Connected Convolutional Networks". In: *arXiv:1608.06993 [cs]*. URL: http://arxiv.org/abs/1608.06993.

Huang, J., V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy (Nov. 2016b). "Speed/accuracy trade-offs for modern convolutional object detectors". In: *arXiv:1611.10012 [cs]*. URL: http://arxiv.org/abs/1611.10012.

Huang, L., Y. Yang, Y. Deng, and Y. Yu (Sept. 2015). "DenseBox: Unifying Landmark Localization with End to End Object Detection". In: *arXiv:1509.04874 [cs]*. URL: http://arxiv.org/abs/1509.04874.

Hubel, D. H. and T. N. Wiesel (1962). "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of Physiology* 160.1, pp. 106–154.

Hubel, D. H. and T. N. Wiesel (1968). "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of Physiology* 195.1, pp. 215–243.

Huber, P. J. (Mar. 1964). "Robust estimation of a location parameter". In: *Annals of Mathematical Statistics* 35.1, pp. 73–101. ISSN: 0003-4851. DOI: 10.1214/aoms/1177703732.

Iandola, F. N., S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer (Feb. 2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". In: *arXiv:1602.07360 [cs]*. URL: http://arxiv.org/abs/1602.07360.

Ioffe, S. and C. Szegedy (Feb. 2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]*. URL: http://arxiv.org/abs/1502.03167.

Jaderberg, M., K. Simonyan, A. Vedaldi, and A. Zisserman (June 2014). "Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition". In: *arXiv:1406.2227 [cs]*. URL: http://arxiv.org/abs/1406.2227.

Jaderberg, M., K. Simonyan, A. Vedaldi, and A. Zisserman (Jan. 2016). "Reading Text in the Wild with Convolutional Neural Networks". In: *International Journal of Computer Vision* 116.1, pp. 1–20. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-015-0823-z. URL: http://link.springer.com/10.1007/s11263-015-0823-z.

Jaderberg, M., K. Simonyan, A. Zisserman, and K. Kavukcuoglu (June 2015). "Spatial Transformer Networks". In: *arXiv:1506.02025 [cs]*. URL: http://arxiv.org/abs/1506.02025.

Jain, M., M. Minesh, and C. V. Jawahar (2017). "Unconstrained Scene Text and Video Text Recognition for Arabic Script". In: *1st International Workshop on Arabic Script Analysis and Recognition, Nancy, France*.

Jia, Y., E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (June 2014). "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv:1408.5093 [cs]*. URL: http://arxiv.org/abs/1408.5093.

Jiang, Y., X. Zhu, X. Wang, S. Yang, W. Li, H. Wang, P. Fu, and Z. Luo (June 2017). "R2CNN: Rotational Region CNN for Orientation Robust Scene Text Detection". In: *arXiv:1706.09579 [cs]*. URL: http://arxiv.org/abs/1706.09579.

Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). "An empirical exploration of recurrent network architectures". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 2342–2350.

Karatzas, D., 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 (Aug. 2015). "ICDAR 2015 competition on Robust Reading". In: *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1156–1160. DOI: 10.1109/ICDAR.2015.7333942.

Karatzas, D., F. Shafait, S. Uchida, M. Iwamura, L. G. i. Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazan, and L. P. de las Heras (Aug. 2013). "ICDAR 2013 Robust Reading Competition". In: IEEE, pp. 1484–1493. ISBN: 978-0-7695-4999-6. DOI: 10.1109/ICDAR.2013.221. URL: http://ieeexplore.ieee.org/document/6628859.

Kim, Y.-D., E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin (Nov. 2015). "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications". In: *arXiv:1511.06530 [cs]*. URL: http://arxiv.org/abs/1511.06530.

Kingma, D. and J. Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*. URL: http://arxiv.org/abs/1412.6980.

Kirkpatrick, J., R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell (Mar. 2017). "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences*, p. 201611835. ISSN: 0027-8424, 1091-6490. DOI:

10.1073/pnas.1611835114. URL: http://www.pnas.org/content/early/2017/03/13/1611835114.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

Kupyn, O., V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas (Nov. 2017). "DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks". In: *arXiv:1711.07064 [cs]*. URL: http://arxiv.org/abs/1711.07064.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (Nov. 1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791. URL: http://ieeexplore.ieee.org/document/726791.

LeCun, Y., B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel (1990). "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems 2*. Ed. by D. S. Touretzky. Morgan-Kaufmann, pp. 396–404. URL: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf.

Lee, C.-Y. and S. Osindero (Mar. 2016). "Recursive Recurrent Nets with Attention Modeling for OCR in the Wild". In: *arXiv:1603.03101 [cs]*. URL: http://arxiv.org/abs/1603.03101.

Lee, C.-Y., S. Xie, P. Gallagher, Z. Zhang, and Z. Tu (Sept. 2014). "Deeply-Supervised Nets". In: *arXiv:1409.5185 [cs, stat]*. URL: http://arxiv.org/abs/1409.5185.

Lee, K., J. Choi, J. Jeong, and N. Kwak (July 2017). "Residual Features and Unified Prediction Network for Single Stage Detection". In: *arXiv:1707.05031 [cs]*. URL: http://arxiv.org/abs/1707.05031.

Li, H., P. Wang, and C. Shen (July 2017). "Towards End-to-end Text Spotting with Convolutional Recurrent Neural Networks". In: *arXiv:1707.03985 [cs]*. URL: http://arxiv.org/abs/1707.03985.

Li, M., T. Zhang, Y. Chen, and A. J. Smola (2014). "Efficient mini-batch training for stochastic optimization". In: ACM Press, pp. 661–670. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623612.

Li, Z. and F. Zhou (Dec. 2017). "FSSD: Feature Fusion Single Shot Multibox Detector". In: *arXiv:1712.00960 [cs]*. URL: http://arxiv.org/abs/1712.00960.

Liang, M. and X. Hu (June 2015). "Recurrent convolutional neural network for object recognition". In: IEEE, pp. 3367–3375. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298958. URL: http://ieeexplore.ieee.org/document/7298958.

Liao, M., B. Shi, and X. Bai (Jan. 2018). "TextBoxes++: A Single-Shot Oriented Scene Text Detector". In: *arXiv:1801.02765 [cs]*. URL: http://arxiv.org/abs/1801.02765.

Liao, M., B. Shi, X. Bai, X. Wang, and W. Liu (2016). "TextBoxes: A Fast Text Detector with a Single Deep Neural Network". In: *arXiv preprint arXiv:1611.06779*. URL: http://arxiv.org/abs/1611.06779.

Lin, M., Q. Chen, and S. Yan (Dec. 2013). "Network In Network". In: *arXiv:1312.4400 [cs]*. URL: http://arxiv.org/abs/1312.4400.

Lin, T.-Y., P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie (Dec. 2016). "Feature Pyramid Networks for Object Detection". In: *arXiv:1612.03144 [cs]*. URL: http://arxiv.org/abs/1612.03144.

Lin, T.-Y., P. Goyal, R. Girshick, K. He, and P. Dollár (Aug. 2017). "Focal Loss for Dense Object Detection". In: *arXiv:1708.02002 [cs]*. URL: http://arxiv.org/abs/1708.02002.

Lin, T.-Y., M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár (May 2014). "Microsoft COCO: Common Objects in Context". In: *arXiv:1405.0312 [cs]*. URL: http://arxiv.org/abs/1405.0312.

Lipton, Z. C. (June 2016). "The Mythos of Model Interpretability". In: *arXiv:1606.03490 [cs, stat]*. URL: http://arxiv.org/abs/1606.03490.

Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg (2016a). "SSD: Single Shot MultiBox Detector". In: *arXiv:1512.02325 [cs]* 9905, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. URL: http://arxiv.org/abs/1512.02325.

Liu, W., C. Chen, K.-Y. K. Wong, Z. Su, and J. Han (Sept. 2016b). "STAR-Net: A SpaTial Attention Residue Network for Scene Text Recognition". In: *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, pp. 43.1–43.13. ISBN: 1-901725-59-6. DOI: 10.5244/C.30.43.

Liu, X., D. Liang, S. Yan, D. Chen, Y. Qiao, and J. Yan (Jan. 2018). "FOTS: Fast Oriented Text Spotting with a Unified Network". In: *arXiv:1801.01671 [cs]*. URL: http://arxiv.org/abs/1801.01671.

Long, J., E. Shelhamer, and T. Darrell (Nov. 2014). "Fully Convolutional Networks for Semantic Segmentation". In: *arXiv:1411.4038 [cs]*. URL: http://arxiv.org/abs/1411.4038.

Lucas, S. M., A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young (Aug. 2003). "ICDAR 2003 Robust Reading Competitions". In: *Proceedings of the*

International Conference on Document Analysis and Recognition (ICDAR), pp. 682–687. DOI: 10.1109/ICDAR.2003.1227749.

Ma, J., W. Shao, H. Ye, L. Wang, H. Wang, Y. Zheng, and X. Xue (2017). "Arbitrary-Oriented Scene Text Detection via Rotation Proposals". In: *IEEE Transactions on Multimedia*. ISSN: 1520-9210, 1941-0077. DOI: 10.1109/TMM.2018.2818020. URL: http://arxiv.org/abs/1703.01086.

Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, p. 6.

Maaten, L. v. d. and G. Hinton (Nov 2008). "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9, pp. 2579–2605.

McCulloch, W. and W. Pitts (1990). "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biology* 52.1, pp. 99–115. ISSN: 00928240. DOI: 10.1016/S0092-8240(05)80006-0.

Minsky, M. and S. Papert (1969). *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press.

Mishra, A., K. Alahari, and C. Jawahar (2012). "Scene Text Recognition using Higher Order Language Priors". In: *Proceedings of the British Machine Vision Conference (BMVC)*. Surrey, United Kingdom: BMVA Press. DOI: 10.5244/C.26.127.

Mordvintsev, A., C. Olah, and M. Tyka (2015). *Inceptionism: Going Deeper into Neural Networks*. Research Blog. URL: http://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html (visited on 03/27/2018).

Mousavian, A., D. Anguelov, J. Flynn, and J. Kosecka (Dec. 2016). "3D Bounding Box Estimation Using Deep Learning and Geometry". In: *arXiv:1612.00496 [cs]*. URL: http://arxiv.org/abs/1612.00496.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press. ISBN: 978-0-262-01802-9.

Nair, V. and G. E. Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. Omnipress, pp. 807–814. ISBN: 978-1-60558-907-7.

Nayef, N., F. Yin, I. Bizid, H. Choi, Y. Feng, D. Karatzas, Z. Luo, U. Pal, C. Rigaud, J. Chazalon, W. Khlif, M. M. Luqman, J. C. Burie, C. l. Liu, and J. M. Ogier (Nov. 2017). "ICDAR2017 Robust Reading Challenge on Multi-Lingual Scene Text Detection and Script Identification - RRC-MLT". In: *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 01, pp. 1454–1459. DOI: 10.1109/ICDAR.2017.237.

Nesterov, Y. (1983). "A Method of Solving a Convex Programming Problem with Convergence Rate O(kˆ(-2))". In: *Soviet Mathematics Doklady* 27.2,

pp. 372–376. URL: http://www.cis.pku.edu.cn/faculty/vision/zlin/1983-A%20Method%20of%20Solving%20a%20Convex%20Programming%20Problem%20with%20Convergence%20Rate%20O(k%5E(-2))_Nesterov.pdf.

Neumann, L. and J. Matas (2010). "A Method for Text Localization and Recognition in Real-world Images". In: *Proceedings of the 10th Asian Conference on Computer Vision - Volume Part III*. ACCV. Berlin, Heidelberg: Springer-Verlag, pp. 770–783. ISBN: 978-3-642-19317-0.

Nguyen, A., J. Yosinski, and J. Clune (2015). "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 427–436. URL: http://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Nguyen_Deep_Neural_Networks_2015_CVPR_paper.html.

Ouaknine, A. (Feb. 2018). *Review of Deep Learning Algorithms for Object Detection*. Medium. URL: http://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852 (visited on 04/07/2018).

Parkhi, O. M., A. Vedaldi, A. Zisserman, and C. V. Jawahar (2012). "Cats and Dogs". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: http://ieeexplore.ieee.org/document/6248092.

Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer (Oct. 2017). "Automatic differentiation in PyTorch". In: URL: http://openreview.net/forum?id=BJJsrmfCZ.

Patel, Y., M. Bušta, and J. Matas (Jan. 2018). "E2E-MLT - an Unconstrained End-to-End Method for Multi-Language Scene Text". In: *arXiv:1801.09919 [cs]*. URL: http://arxiv.org/abs/1801.09919.

Phan, T. Q., P. Shivakumara, S. Tian, and C. L. Tan (Dec. 2013). "Recognizing Text with Perspective Distortion in Natural Scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 569–576. DOI: 10.1109/ICCV.2013.76. URL: http://ieeexplore.ieee.org/document/6751180.

Qian, N. (1999). "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1, pp. 145–151.

Rabiner, L. R. (Feb. 1989). "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2, pp. 257–286. ISSN: 0018-9219. DOI: 10.1109/5.18626.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (June 2015). "You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv:1506.02640 [cs]*. URL: http://arxiv.org/abs/1506.02640.

Redmon, J. and A. Farhadi (Dec. 2016). "YOLO9000: Better, Faster, Stronger". In: *arXiv:1612.08242 [cs]*. URL: http://arxiv.org/abs/1612.08242.

Redmon, J. and A. Farhadi (Apr. 2018). "YOLOv3: An Incremental Improvement". In: *arXiv:1804.02767 [cs]*. URL: http://arxiv.org/abs/1804.02767.

Ren, S., K. He, R. Girshick, and J. Sun (June 2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *arXiv:1506.01497 [cs]*. URL: http://arxiv.org/abs/1506.01497.

Risnumawan, A., P. Shivakumara, C. S. Chan, and C. L. Tan (Dec. 2014). "A robust arbitrary text detection system for natural scene images". In: *Expert Systems with Applications* 41.18, pp. 8027–8048. ISSN: 09574174. DOI: 10.1016/j.eswa.2014.07.008.

Ronneberger, O., P. Fischer, and T. Brox (May 2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *arXiv:1505.04597 [cs]*. URL: http://arxiv.org/abs/1505.04597.

Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review*, pp. 65–386.

Ruder, S. (Sept. 2016). "An overview of gradient descent optimization algorithms". In: *arXiv:1609.04747 [cs]*. URL: http://arxiv.org/abs/1609.04747.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (Oct. 1986). "Learning representations by back-propagating errors". In: *Nature* 323, pp. 533–536. DOI: 10.1038/323533a0.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (Sept. 2014). "ImageNet Large Scale Visual Recognition Challenge". In: *arXiv:1409.0575 [cs]*. URL: http://arxiv.org/abs/1409.0575.

Sandler, M., A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen (Jan. 2018). "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation". In: *arXiv:1801.04381 [cs]*. URL: http://arxiv.org/abs/1801.04381.

Schuster, M. and K. K. Paliwal (1997). "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681. URL: http://ieeexplore.ieee.org/document/650093.

Sermanet, P., D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun (Dec. 2013). "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *arXiv:1312.6229 [cs]*. URL: http://arxiv.org/abs/1312.6229.

Shen, Z., Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue (Aug. 2017). "DSOD: Learning Deeply Supervised Object Detectors from Scratch". In: *arXiv:1708.01241 [cs]*. URL: http://arxiv.org/abs/1708.01241.

Shi, B., X. Bai, and S. Belongie (Mar. 2017). "Detecting Oriented Text in Natural Images by Linking Segments". In: *arXiv:1703.06520 [cs]*. URL: http://arxiv.org/abs/1703.06520.

Shi, B., X. Bai, and C. Yao (July 2015). "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition". In: *arXiv:1507.05717 [cs]*. URL: http://arxiv.org/abs/1507.05717.

Shi, B., X. Wang, P. Lyu, C. Yao, and X. Bai (Mar. 2016). "Robust Scene Text Recognition with Automatic Rectification". In: *arXiv:1603.03915 [cs]*. URL: http://arxiv.org/abs/1603.03915.

Shrivastava, A., A. Gupta, and R. Girshick (Apr. 2016). "Training Region-based Object Detectors with Online Hard Example Mining". In: *arXiv:1604.03540 [cs]*. URL: http://arxiv.org/abs/1604.03540.

Shwartz-Ziv, R. and N. Tishby (Mar. 2017). "Opening the Black Box of Deep Neural Networks via Information". In: *arXiv:1703.00810 [cs]*. URL: http://arxiv.org/abs/1703.00810.

Simon, M., S. Milz, K. Amende, and H.-M. Gross (Mar. 2018). "Complex-YOLO: Real-time 3D Object Detection on Point Clouds". In: *arXiv:1803.06199 [cs]*. URL: http://arxiv.org/abs/1803.06199.

Simonyan, K., A. Vedaldi, and A. Zisserman (Dec. 2013). "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *arXiv:1312.6034 [cs]*. URL: http://arxiv.org/abs/1312.6034.

Simonyan, K. and A. Zisserman (Sept. 2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv:1409.1556 [cs]*. URL: http://arxiv.org/abs/1409.1556.

Singh, B., H. Li, A. Sharma, and L. S. Davis (Dec. 2017). "R-FCN-3000 at 30fps: Decoupling Detection and Classification". In: *arXiv:1712.01802 [cs]*. URL: http://arxiv.org/abs/1712.01802.

Smith, C., B. McGuire, T. Huang, and G. Yang (2006). "The History of Artificial Intelligence". In: *University of Washington, History of Computing CSEP 590A*, p. 27.

Smith, R., C. Gu, D.-S. Lee, H. Hu, R. Unnikrishnan, J. Ibarz, S. Arnoud, and S. Lin (Feb. 2017). "End-to-End Interpretation of the French Street Name Signs Dataset". In: *arXiv:1702.03970 [cs]*. URL: http://arxiv.org/abs/1702.03970.

Souly, N., C. Spampinato, and M. Shah (Oct. 2017). "Semi Supervised Semantic Segmentation Using Generative Adversarial Network". In: IEEE, pp. 5689–5697. ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.606. URL: http://ieeexplore.ieee.org/document/8237868.

Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting."

In: *Journal of Machine Learning Research* 15.1, pp. 1929–1958. URL: http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

Stanford University CS231n: Convolutional Neural Networks for Visual Recognition (2018). URL: http://cs231n.stanford.edu (visited on 02/27/2018).

Sutskever, I., J. Martens, G. Dahl, and G. Hinton (June 2013). "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research. Atlanta, Georgia, USA: PMLR, pp. 1139–1147. URL: http://proceedings.mlr.press/v28/sutskever13.html.

Szegedy, C., S. Ioffe, V. Vanhoucke, and A. Alemi (Feb. 2016). "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *arXiv:1602.07261 [cs]*. URL: http://arxiv.org/abs/1602.07261.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (Sept. 2014). "Going Deeper with Convolutions". In: *arXiv:1409.4842 [cs]*. URL: http://arxiv.org/abs/1409.4842.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (Dec. 2015). "Rethinking the Inception Architecture for Computer Vision". In: *arXiv:1512.00567 [cs]*. URL: http://arxiv.org/abs/1512.00567.

The OpenCV Reference Manual (Apr. 2014). 2.4.9.0. Itseez.

Tian, S., S. Lu, and C. Li (Oct. 2017). "WeText: Scene Text Detection under Weak Supervision". In: *arXiv:1710.04826 [cs]*. URL: http://arxiv.org/abs/1710.04826.

Tian, Z., W. Huang, T. He, P. He, and Y. Qiao (Sept. 2016). "Detecting Text in Natural Image with Connectionist Text Proposal Network". In: *arXiv:1609.03605 [cs]*. URL: http://arxiv.org/abs/1609.03605.

Uijlings, J. R., K. E. Van De Sande, T. Gevers, and A. W. Smeulders (2013). "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104.2, pp. 154–171. URL: http://link.springer.com/article/10.1007/s11263-013-0620-5.

Understanding LSTM Networks – colah's blog (2017). URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 11/30/2017).

Veit, A., T. Matera, L. Neumann, J. Matas, and S. Belongie (Jan. 2016). "COCO-Text: Dataset and Benchmark for Text Detection and Recognition in Natural Images". In: *arXiv:1601.07140 [cs]*. URL: http://arxiv.org/abs/1601.07140.

Viola, P. and M. Jones (Dec. 2001). "Rapid Object Detection using a Boosted Cascade of Simple Features". In: *Proceedings of the IEEE Conference on*

Computer Vision and Pattern Recognition (CVPR). Vol. 1, pp. I–I. DOI: 10.1109/CVPR.2001.990517. URL: http://ieeexplore.ieee.org/document/990517.

Wan, L., M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus (2013). "Regularization of neural networks using dropconnect". In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pp. 1058–1066.

Wang, F., M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang (Apr. 2017a). "Residual Attention Network for Image Classification". In: *arXiv:1704.06904 [cs]*. URL: http://arxiv.org/abs/1704.06904.

Wang, H., Z. Li, X. Ji, and Y. Wang (June 2017b). "Face R-CNN". In: *arXiv:1706.01061 [cs]*. URL: http://arxiv.org/abs/1706.01061.

Wang, K., B. Babenko, and S. Belongie (Nov. 2011). "End-to-End Scene text Recognition". In: *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 1457–1464. DOI: 10.1109/ICCV.2011.6126402.

Wang, T., D. J. Wu, A. Coates, and A. Y. Ng (Nov. 2012). "End-To-End Text Recognition with Convolutional Neural Networks". In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*, pp. 3304–3308.

Weng, L. (Dec. 2017). *Object Recognition for Dummies Part 3: R-CNN and Fast/Faster/Mask R-CNN and YOLO*. URL: http://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html (visited on 04/13/2018).

Werbos, P. J. (1988). "Generalization of backpropagation with application to a recurrent gas market model". In: *Neural Networks* 1.4, pp. 339–356. ISSN: 0893-6080. DOI: 10.1016/0893-6080(88)90007-X. URL: http://www.sciencedirect.com/science/article/pii/089360808890007X.

Xie, S., R. Girshick, P. Dollár, Z. Tu, and K. He (Nov. 2016). "Aggregated Residual Transformations for Deep Neural Networks". In: *arXiv:1611.05431 [cs]*. URL: http://arxiv.org/abs/1611.05431.

Xie, S. and Z. Tu (2015). "Holistically-Nested Edge Detection". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1395–1403. URL: http://www.cv-foundation.org/openaccess/content_iccv_2015/html/Xie_Holistically-Nested_Edge_Detection_ICCV_2015_paper.html.

Xu, B., N. Wang, T. Chen, and M. Li (May 2015a). "Empirical Evaluation of Rectified Activations in Convolutional Network". In: *arXiv:1505.00853 [cs, stat]*. URL: http://arxiv.org/abs/1505.00853.

Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio (Feb. 2015b). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *arXiv:1502.03044 [cs]*. URL: http://arxiv.org/abs/1502.03044.

Yang, X., D. He, Z. Zhou, D. Kifer, and C. L. Giles (2017). "Learning to Read Irregular Text with Attention Mechanisms". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3280–3286. DOI: 10.24963/ijcai.2017/458.

Yao, C., X. Bai, N. Sang, X. Zhou, S. Zhou, and Z. Cao (2016). "Scene Text Detection via Holistic, Multi-Channel Prediction". In: *arXiv preprint arXiv:1606.09002*. URL: http://arxiv.org/abs/1606.09002.

Ye, Q. and D. Doermann (July 2015). "Text Detection and Recognition in Imagery: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.7, pp. 1480–1500. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2014.2366765. URL: http://ieeexplore.ieee.org/document/6945320.

Yu, J., Y. Jiang, Z. Wang, Z. Cao, and T. Huang (2016). "UnitBox: An Advanced Object Detection Network". In: *arXiv:1608.01471 [cs]*, pp. 516–520. DOI: 10.1145/2964284.2967274. URL: http://arxiv.org/abs/1608.01471.

Zagorchev, L. and A. Goshtasby (Mar. 2006). "A Comparative Study of Transformation Functions for Nonrigid Image Registration". In: *IEEE Transactions on Image Processing* 15.3, pp. 529–538. ISSN: 1057-7149. DOI: 10.1109/TIP.2005.863114. URL: http://ieeexplore.ieee.org/document/1593658.

Zeiler, M. D. (Dec. 2012). "ADADELTA: An Adaptive Learning Rate Method". In: *arXiv:1212.5701 [cs]*. URL: http://arxiv.org/abs/1212.5701.

Zeiler, M. D. and R. Fergus (Jan. 2013a). "Stochastic Pooling for Regularization of Deep Convolutional Neural Networks". In: *arXiv:1301.3557 [cs, stat]*. URL: http://arxiv.org/abs/1301.3557.

Zeiler, M. D. and R. Fergus (Nov. 2013b). "Visualizing and Understanding Convolutional Networks". In: *arXiv:1311.2901 [cs]*. URL: http://arxiv.org/abs/1311.2901.

Zeiler, M. D., D. Krishnan, G. W. Taylor, and R. Fergus (June 2010). "Deconvolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 2528–2535. ISBN: 978-1-4244-6984-0. DOI: 10.1109/CVPR.2010.5539957. URL: http://ieeexplore.ieee.org/document/5539957.

Zhang, X., X. Zhou, M. Lin, and J. Sun (July 2017). "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: *arXiv:1707.01083 [cs]*. URL: http://arxiv.org/abs/1707.01083.

Zhou, X., C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang (Apr. 2017). "EAST: An Efficient and Accurate Scene Text Detector". In: *arXiv:1704.03155 [cs]*. URL: http://arxiv.org/abs/1704.03155.

Zhu, X., Y. Jiang, S. Yang, X. Wang, W. Li, P. Fu, H. Wang, and Z. Luo (Nov. 2017). "Deep Residual Text Detection Network for Scene Text". In: *arXiv:1711.04147 [cs]*. URL: http://arxiv.org/abs/1711.04147.

Zhu, Y., C. Yao, and X. Bai (Feb. 2016). "Scene text detection and recognition: recent advances and future trends". In: *Frontiers of Computer Science* 10.1, pp. 19–36. ISSN: 2095-2228, 2095-2236. DOI: 10.1007/s11704-015-4488-0. URL: http://link.springer.com/10.1007/s11704-015-4488-0.

Zitnick, L. and P. Dollár (Sept. 2014). "Edge Boxes: Locating Object Proposals from Edges". In: *ECCV*. European Conference on Computer Vision. URL: http://www.microsoft.com/en-us/research/publication/edge-boxes-locating-object-proposals-from-edges.

Zou, H. and T. Hastie (2005). "Regularization and variable selection via the elastic net". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2, pp. 301–320.

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BLSTM	Bidirectional Long Short-Term Memory
BN	Batch Normalization
BPTT	Backpropagation Through Time
CE	Cross Entropy
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRNN	Convolutional Recurrent Neural Network
CTC	Connectionist Temporal Classification
DCNN	Deep Convolutional Neural Network
DL	Deep Learning
DNA	Deoxyribonucleic acid
DNN	Deep Neural Network
DPM	Deformable Part Models
CNTK	Computational Network Toolkit
FC	Fully Connected
FCN	Fully Convolutional Network
FFNN	Feed-Forward Neural Network
FPGA	Field-Programmable Gate Array
GAN	Generative Adversarial Network
GD	Gradient Descent
GIMP	GNU Image Manipulation Program
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
HED	Holistically-Nested Edge Detection
HOG	Histogram of Oriented Gradients

IoU Intersection over Union
IoT Internet of Things
JVM Java Virtual Machine
JIT just-in-time
KL Kullback Leibler
LSTM Long Short-Term Memory
mAP mean Average Precision
MLP Multilayer Perceptron
MAE Mean Average Error
MSER Maximally Stable Extremal Regions
MLE Maximum Likelihood Estimation
MSE Mean Square Error
NMS Non Maximum Suppression
NN Neural Network
OCR Optical Character Recognition
OHEM Online Hard Example Mining
OHNM Online Hard Negative Mining
ONNX Open Neural Network Exchange
OS Operating System
PReLU Parametric Rectified Linear Unit
RBF Radial Basis Function
R-CNN Region-based Convolutional Neural Netwoks
ReLU Rectified Linear Unit
R-FCN Region-based Fully Convolutional Network
RGB Red, Green, Blue
RNN Recurrent Neural Network
RoI Region of Interest
RPN Region Proposal Network
ROS Robot Operating System
SGD Stochastic Gradient Descent
SSD Single Shot MultiBox Detector
TPS Thin-Plate Spline
STN Spatial Transformer Network
SVM Support Vector Machine
TPU Tensor Processing Unit
YOLO You Only Look Once

List of Figures

List of Tables

List of Algorithms

Appendix A

Dataset Examples

Fig. A.1: Example images from the ICDAR 2013 / ICDAR 2015 Focused Scene Text dataset. All images have axis-aligned bounding box annotation.

Fig. A.2: Example images from the ICDAR 2015 Incidental Scene Text dataset. The bounding box annotations are provided as quadrilaterals and most of the text instances are perspectively distorted, blurred or suffer from unfavorable lighting conditions.

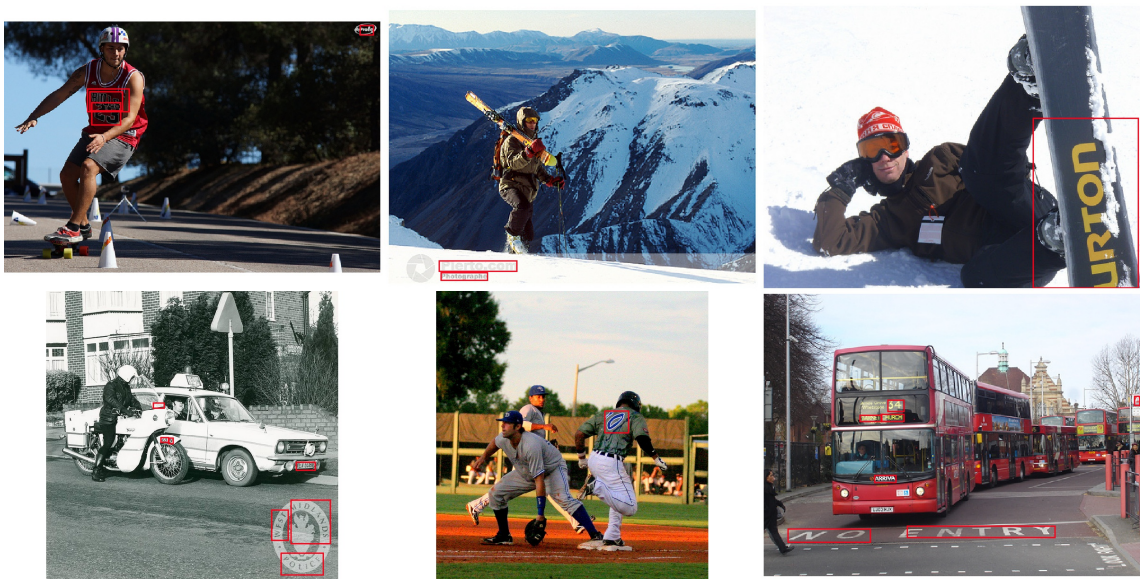


Fig. A.3: Example images from the COCO-Text dataset. The images were not selected with text in mind and the axis-aligned bounding boxes my not be optimal for curved or oriented text.

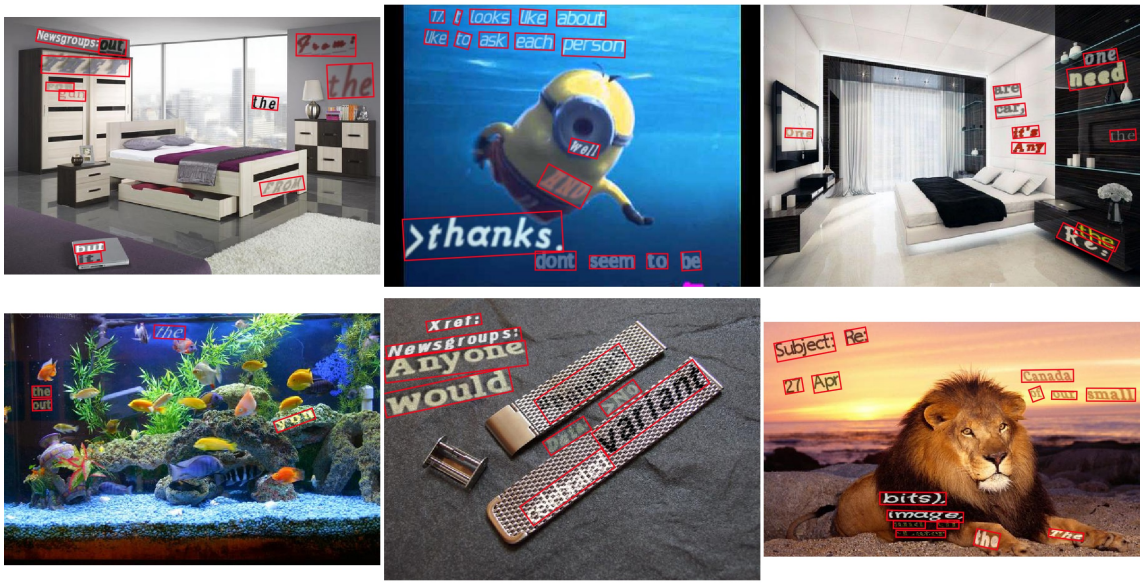


Fig. A.4: Example images from the SynthText dataset. Text instances and annotations are in form of oriented bounding boxes and were generated by a synthetic text engine.

Appendix B

SegLink Illustrations

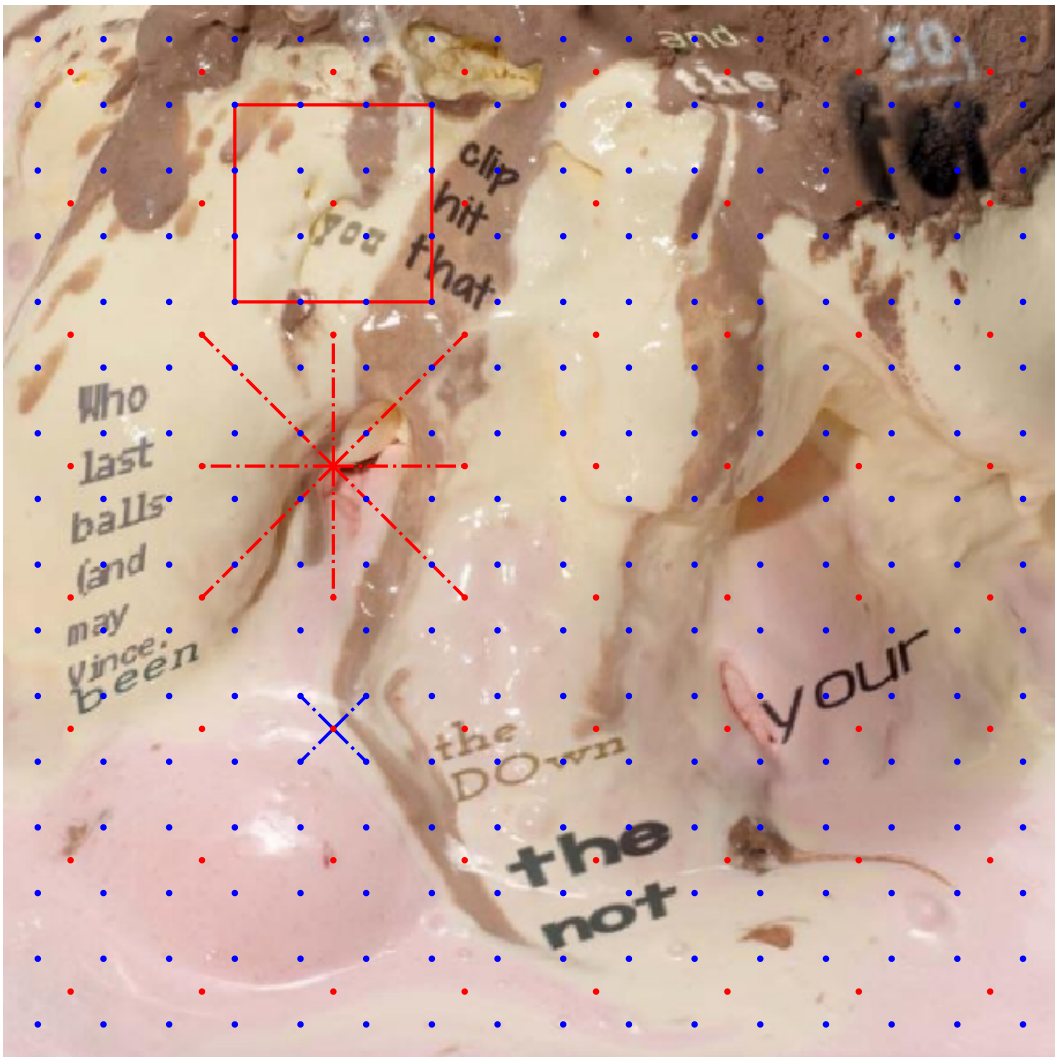


Fig. B.1: Local definitions in SegLink; locations in the input image corresponding to the 8x8 feature map as red and locations corresponding to the 16x16 feature map as blue dots. One default box (red box), within layer links (red dashed lines) and cross layer links (blue dashed lines), each defined for one single location of the 8x8 feature map. They are defined for all 64 locations, but for the sake of clarity, they are each drawn for one location only.

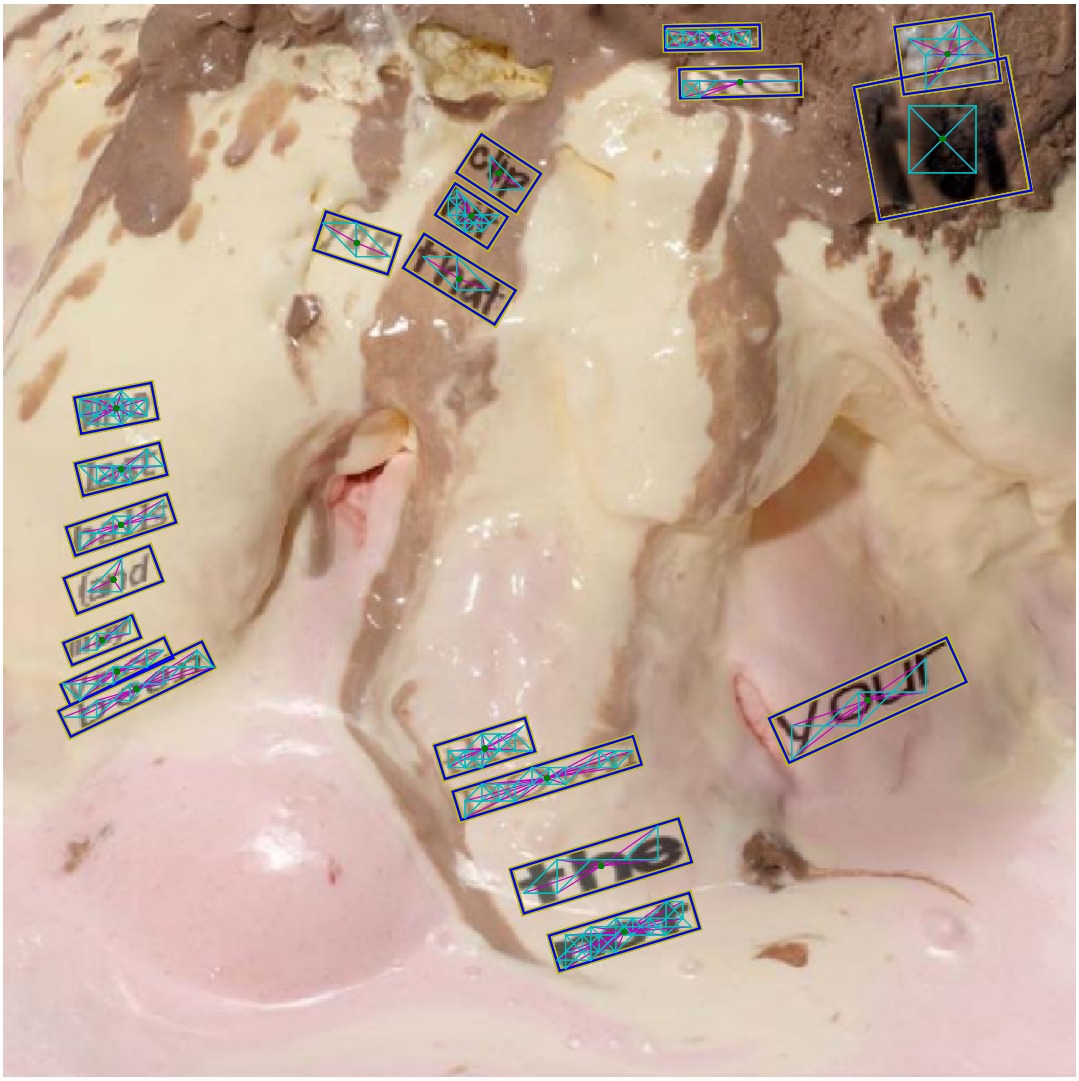


Fig. B.2: Local ground truth assignment in SegLink; ground truth bounding boxes are colored in yellow/blue with their center in green. Positive ground truth links are shown in cyan and the assignment of reference locations to ground truth boxes magenta.

Fig. B.3: Local predictions in SegLink; positive segments are shown in black, their centers in magenta and their reference location in green. Positive within-layer links in yellow and positive cross-layer links orange.

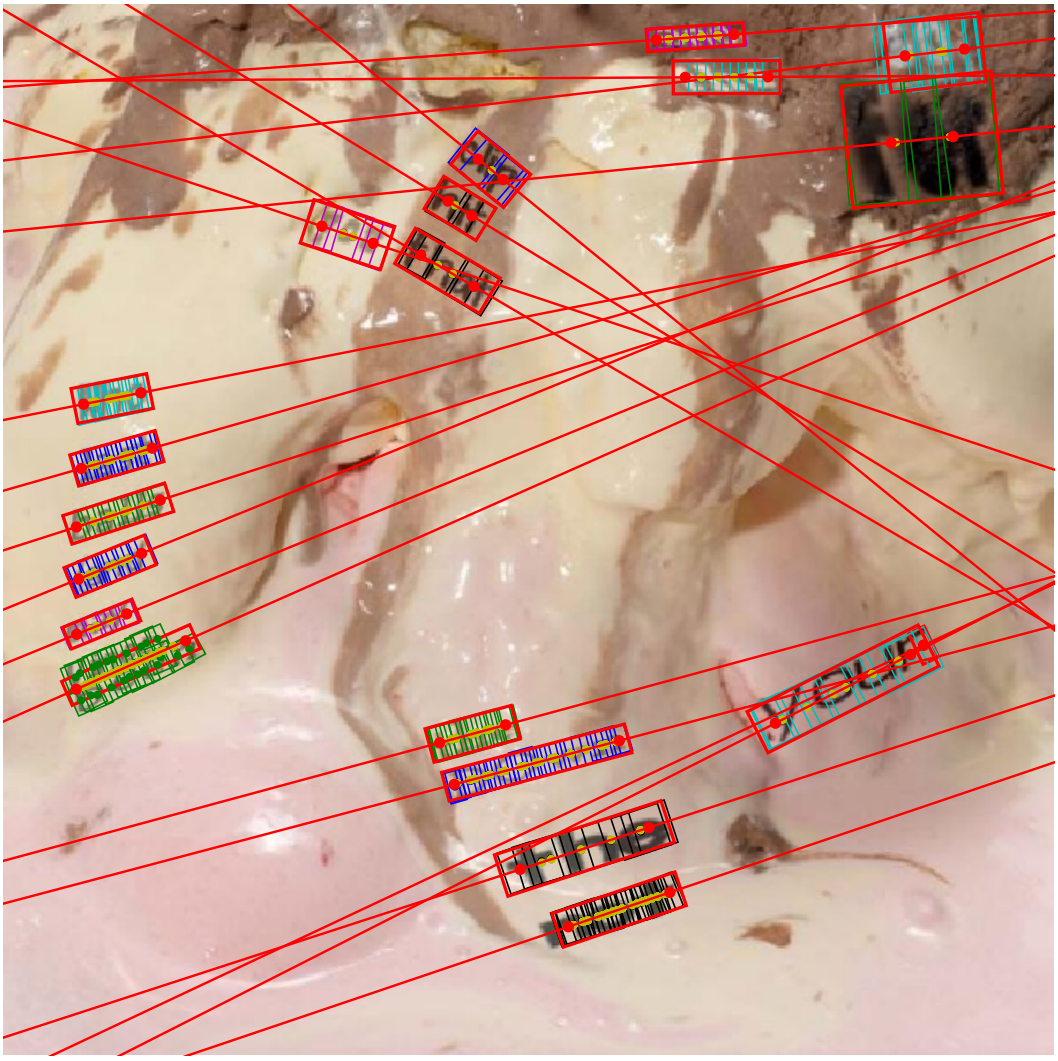


Fig. B.4: Bounding box construction in SegLink; segments within the same group have the same color. The red line for each group is the one with minimum distance to the segment centers. The yellow dots are their projections on the lines and the red dots are the extreme points. The large red boxes are the combined bounding boxes.

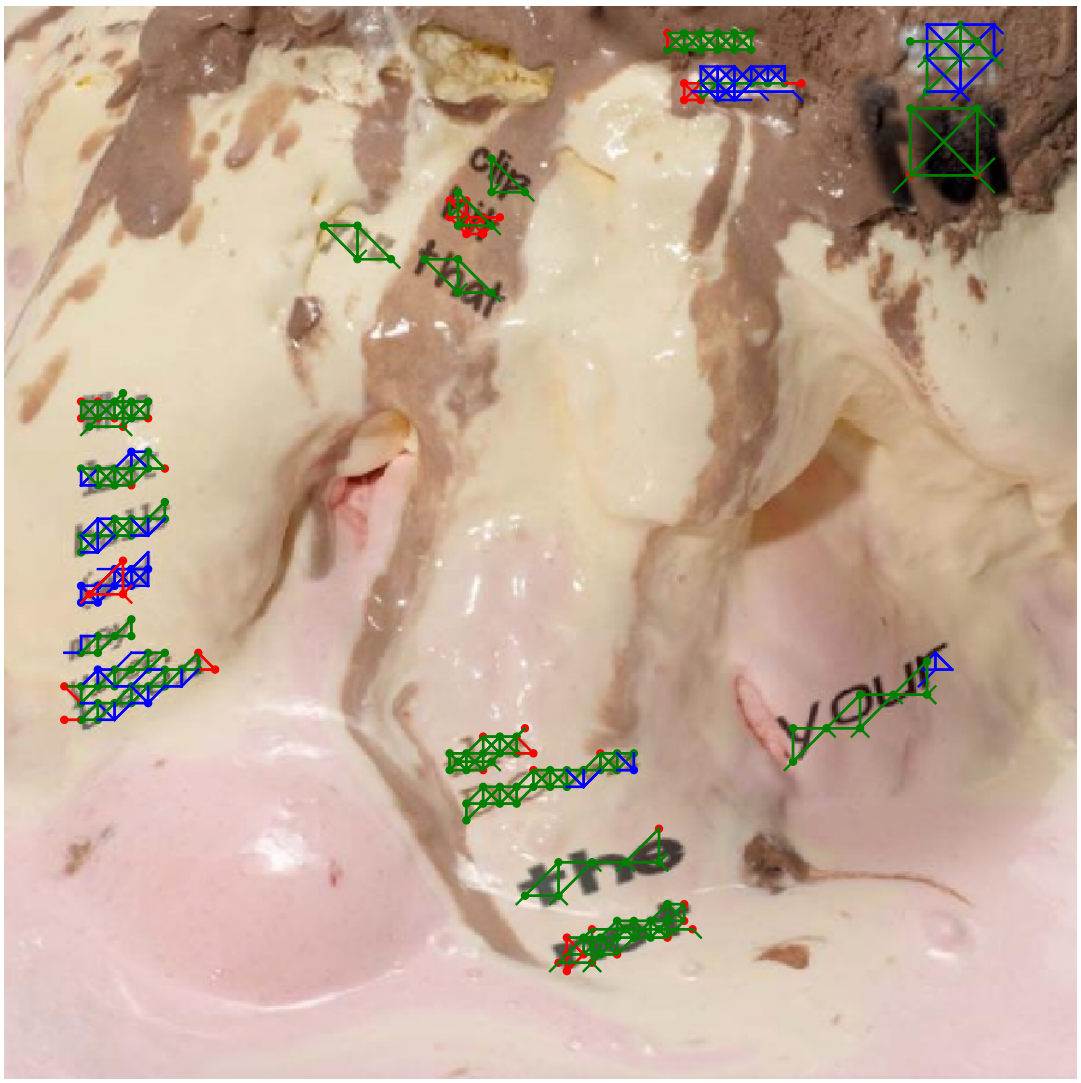


Fig. B.5: Visual evaluation of local predictions in SegLink; true positive segments and links are colored in green, false positives in blue and false negatives in red.

Appendix C

Experimental Results

Fig. C.1: Detection examples using the SegLink-DN-FL model on SynthText validation subset; ground truth bounding box in yellow (distorted original box due to 512x512 input) and blue (rectangular correction), center in green and detected bounding box in red.

Fig. C.2: Real-world detection examples using the SegLink-DN-FL model

Fig. C.3: Real-world detection examples using the SegLink-DN-FL model continued

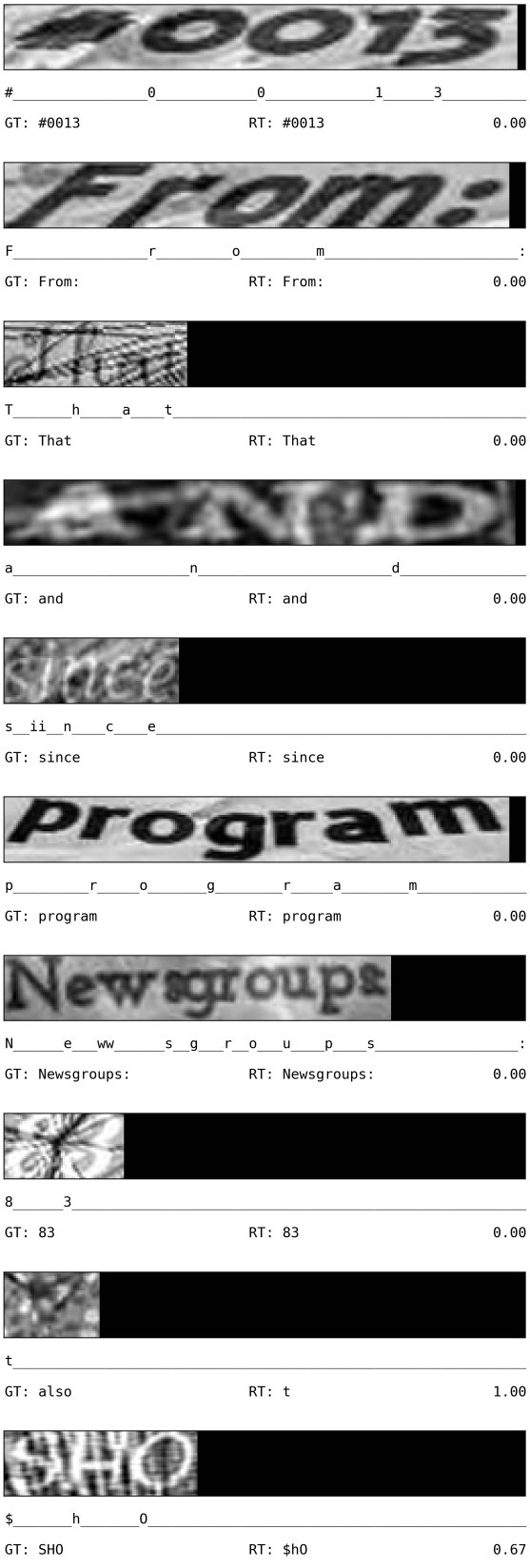


Fig. C.4: Recognition examples CRNN-GRU model on SynthText validation subset; from top to bottom: input image, sequence classification, ground truth (GT), recognized text (RT) and normalized edit distance.

Fig. C.5: End-to-End SegLink-DN-FL + CRNN-GRU recognition examples on SynthText validation subset

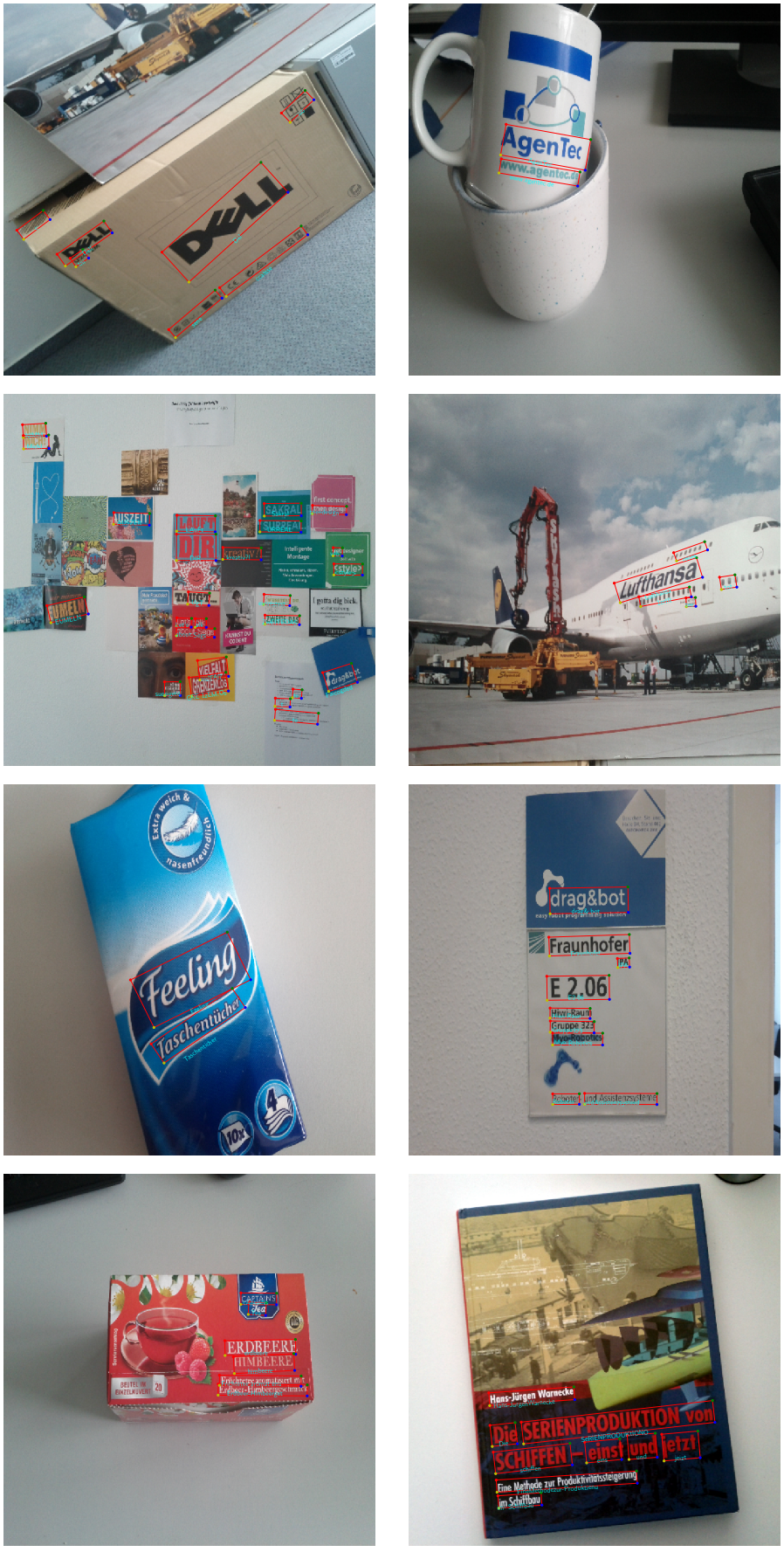


Fig. C.6: End-to-End SegLink-DN-FL + CRNN-GRU recognition examples on real-world images