computer graphics tübingen

Reading scene text in deep convolutional sequences

Jan Haug, Marius Hobbhahn, Roman Schulte

April 6, 2018

Winter Term 2017

Practical Course on Computer Graphics

Reading scene text in deep convolutional sequences

Jan Haug, Marius Hobbhahn, Roman Schulte

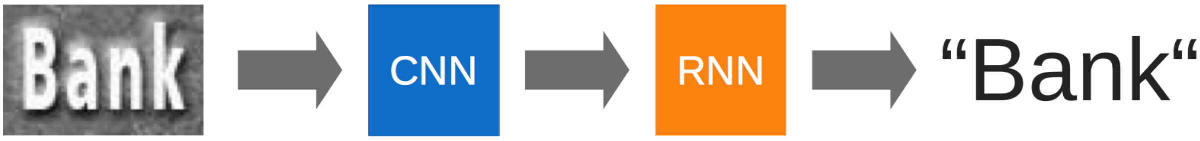

Figure 1: Basic concept of our network.

Abstract

The goal of our project was to implement a neural network which is able to identify text in real-world images. Our work is based on a paper from He et. al [HHQ+15]. The network consists of three parts: A CNN to detect major character features inside small slices of the image, a RNN to predict characters using the accumulated feature data and a CTC to reconstruct the final word. All parts were implemented using Python 3, the Tensorflow 1.6 API and some other supporting libraries. We used the IIIT5K data set to train and test our network.

1 Introduction

Natural scene recognition is one of the most important tasks in numerous automation processes. Autonomous cars have to be able to identify construction signs to follow their instructions, traffic agencies can identify license plates of traffic offenders and robots will need to read text and numbers in a natural environment. In 2015, He et. al [HHQ+15] proposed a model to read scene text in natural environments with a combination of a convolutional neural network (CNN) and recurrent neural network (RNN) which is not restricted to having words pre-defined in a dictionary or segmentation of individual letters (which poses a difficult task by itself), but instead views the input as a sequence and uses context information to decode a word as a whole, e.g. on an advertisement sign. One of the main challenges of scene recognition is the high variance in image quality and style of displayed signs. The same word can be represented in many different ways in an image by variation in lighting, colours, perspective or font. The approach chosen by the authors has some advantages over previous approaches (i.e. TODO), which fail to capture all representations of strings in a natural environment. It uses a combined model, capitalizing on both the high performance in character recognition of CNNs and also the ability of RNNs to process context information in sequences.

1. CNNs have become very accurate at character recognition and building high level representations. By moving a sliding window over a given picture, we are able to create character-level inputs without concerns about character segmentation, as this problem is handled by the RNN part of the model. The simplicity of a sliding window approach implies that some frames will not yield beneficial results, but the RNN is also able to process unclear outputs of the CNN; this only means that the sliding window should be moved in small steps, so no characters can be skipped by accident. On the other hand, this simplicity results in not only a small (i.e. fast) CNN, but also in a high versatility, as the inputs do not need to be preprocessed except for the trimming around a word and the re-scaling to a 32px height.
2. RNNs are very strong at processing sequences of inputs and accounting for context information, especially when the length of a given sequence is unknown at first. As words are nothing else than sequences of letters (with variable length), the application of an RNN seems very

intuitive. Due to this context information, the RNN can make decisions for a sign based on the previously identified signs, i.e. The letter 'a' might be more likely to occur after a 'b' than an 'y'. Since the RNN receives the activations of the penultimate layer of the CNN (which has more neurons), there should also be more information on letters in direct vicinity to the recognized one, since there will usually be more than one symbol in the sliding window. To use context information from previous and later signs we use a bidirectional approach, that independently applies the model forwards and backwards.

3. If property 1 and 2 are combined successfully, it is possible to process unknown words and arbitrary sequences of symbols. Since the training is done in a compositional way, it is independent of dictionaries or corpora of already known words and combinations of strings. Therefore it would be able to read the information on any given picture even if no meaning is attached to it. The only restriction is that the set of symbols needs to be specified as the CNN classifies each input as one of 26 letters or of the 10 different digits (i.e. the recognition of special letters like Umlauts would require slight modifications to the model and retraining it from scratch)

The main contribution of this paper was to improve the accuracy in scene text recognition on given benchmark test sets like the IIIT5K.Our contribution is the attempt to reproduce the work of the paper, the comparison of the results given the same benchmarks, and the documentation of possible caveats or other difficulties during this process, especially where the authors do not specify details on the implementation or its parameters.

2 Related Work

The paper in discussion was written by He et al. from 2015 [HHQ+15]. In this section we would like to address two types of related papers. Firstly, papers that have tried different approaches to solve the same task but with less accurate results. These will be mainly papers published before the original paper. Secondly, we explore how the presented methods was applied to other related tasks in later publications. Follow up studies and improvements will be provided in the discussion part of this paper.

Earlier approaches of text scene recognition like shown in [MAJ12], or [SWX+14] used combinations of structure guided character recognition, linguistic knowledge and Bayesian inference via decision trees. All of these broke the benchmarks on frequently used data sets such as IIIT5K or ICDAR but did not use learning algorithms based on neural networks. First approaches of leveraging the advantages of neural networks were done for example by [AP13], who used a classical CNN and hybrid hidden Markov models (HMMs), which could be seen as predecessors of the frequently used connectionist temporal classification (CTC) of today, which is also used in the final part of the model presented here. The very same model idea as described here, i.e. a combination of sliding window, CNN, RNN and CTC, was used to significantly improve benchmarks for other tasks of scene recognition. [GTLL16] for example used the method for the recognition of house numbers and achieved an accuracy of 91 percent, compared to 81 percent of previous approaches. [LS16] were able to lift the benchmark of license plate recognition on the Caltech cars data set from 95.5 to 97.56 and the recall from 84.8 to 95.24 percent. The difference between precision and recall in this paper not only implies superior recognition if the plate has been detected but also superior detection of license plates in the first place. Both of these papers show the strength of the combining CNN and RNN in a single model. The way [HHQ+15] differs from these approaches is through very high diversity of the image inputs: License plates and house numbers have much more structure and in the house number case, less different symbols on it. With the IIIT5K dataset, the network can make far fewer assumptions about its inputs, since there are a lot of different shapes within each class due to the vast amount of different fonts used.

3 Structure of the network

After applying a sliding window on a given image each frame is forwarded into a CNN. The sequence of resulting feature vectors is then sent through the RNN and results in a sequence of characters and numbers. An overview of the whole network is given in figure 2.

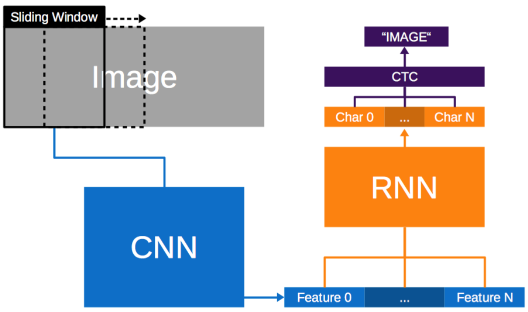

Figure 2: General structure of the network.

3.1 Data wrangling

The IIIT5K provides 5000 real-world images containing a single word. In general that is exactly the type of data our network should handle. But neither the CNN nor the RNN can be trained with that kind of input. The CNN needs 32×32 pixel images to learn single characters. These kind of data can be derived from the IIIT5K data set. Additionally to the labels the data set also provides some extra metadata about the data points. One of them are the bounding boxes of every character in the image. This information is used to create new training and validation data sets based on the content of IIIT5K. The RNN expects a sequence of $128D$ feature vectors. Therefore, we created new data sets containing the output of the already trained CNN for every image of the IIIT5K. Outside of the training our network can process any kind of image. When a image is passed to the network it will be preprocessed to match the requirements. This preprocessing includes: grayscaling, resizing and applying the sliding window.

3.2 Structure of the CNN

The CNN has 5 convolutional layers. In the first 3 we apply a 9x9 kernel on the inputs and use a max. group of 2. For the 4th layer a 8x8 kernel and a max group of 4 is used to reduce the dimensionality of the output vector to 128. Note that this output of a 128D feature vector is forwarded into the RNN. The 5th layer uses a 1x1 kernel and also uses a max group of 4 to further reduce dimensionality. This output is then forwarded through a fully connected layer with 36 outputs for the 26 possible lower case characters and 10 numbers. For activation we use the identity function. Note that the last convolutional and the fully connected layer are merely used for the training of the CNN but later not used for the classification in the RNN. A detailed description of the network can be seen in graphic 4.

3.3 Structure of the RNN

The recurrent part of the network receives a sequence of feature vectors, each of which is provided by the CNN. We use bidirectional stacked long-short-term-memory (LSTM) cell blocks with 128 inputs respectively as developed by [HS97] originally. Bidirectional means that the RNN essentially has 2 independent units. In one the sequence is put in forwards and the other backwards. The results are then concatenated afterwards and fed into a fully connected layer with 37 output classes. 26 for each character, 10 for numbers and 1 for undetectable sign or no character. This can be seen in figure 3.

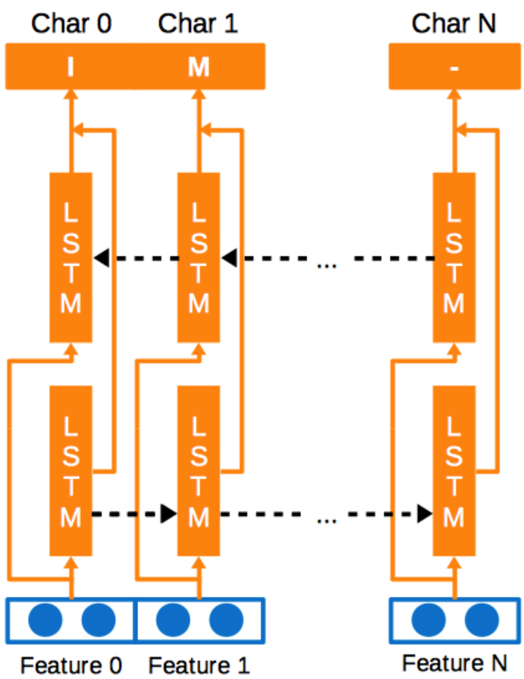

Figure 3: Structure of the RNN with a bidirectional approach with 128 inputs respectively.

At this point we still have redundancies or unclear information due to the sliding window. To solve this problem a connectionist temporal classification (CTC) is used. A detailed explanation

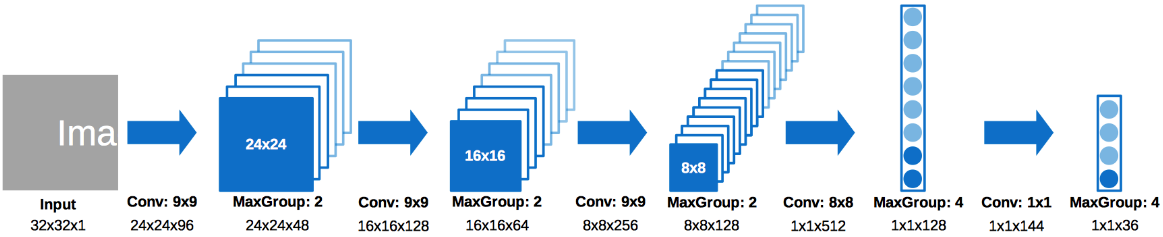

Figure 4: Structure of the CNN with 5 convolutional and 5 grouping layers.

would be outside of the scope of this paper and can be found in the original paper [GFGS06]. But for a working explanation consider it as a way to remove unlikely information given the labels. In sequential information the placement of non-character symbols or redundancies is always hard to deal with. The CTC approach therefore uses a hidden Markov model (HMM) to sample different possible outcomes as paths in a tree and choose the path with the highest likelihood. Consider an example where the sequence after the LSTM cell block is "iiii—mmmmaaagggee–", where '-' represents the non-character symbol. The CTC would then reduce the redundancies and remove non-character symbols until the most likely outcome, "image" is put out. The information learned by the CTC during the training are backpropagated through the network such that future classification can already access them. A graphical overview is provided in figure 5.

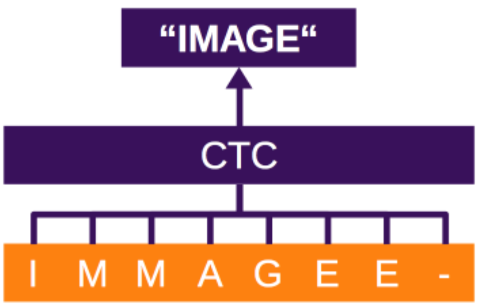

Figure 5: Simplified structure of the CTC.

4 Implementation details

The project is written in Python 3. It uses the Tensorflow 1.6 API to build and train the network graph. Other libraries we used are: Tensorpack which simplifies the usage of Tensorflow for maschine learning, OpenCV for image processing and Numpy as math framework.

4.1 Data wrangling

One of the major challenges in this project was to setup the data flow. As Python is a untyped languages it is easy to pass the right data in the wrong format. Sometimes this does not lead to a runtime error, but to useless training results.

Image preprocessing The CNN expects the input images to have a size of 32×32 pixels and only s single color channel. Most real world images will not match this requirements. Therefore we need to transform the input images and feed them in small bites to the CNN. The first step of the preprocessing merges all color channels of the input image into one. After that we resize the image to a height of 32 pixels, but keeping the aspect ratio. Now we can apply the sliding window and pass a small sub-image to the CNN. After each step the sliding windo is moved 8 pixels until it reaches the end of the image. Figure 6 shows a example for this step.

RNN training data One major problem in training the RNN was to provide the labels for a batch. The CTC expects the labels of one batch to share the same length. In reality this will fairly never happen. Therefore the labels need to expanded to have the same length. This is done by determine the maximal label length in each batch and then adopting the other labels. This is implemented by using a sparse tensor, where empty entries repre-

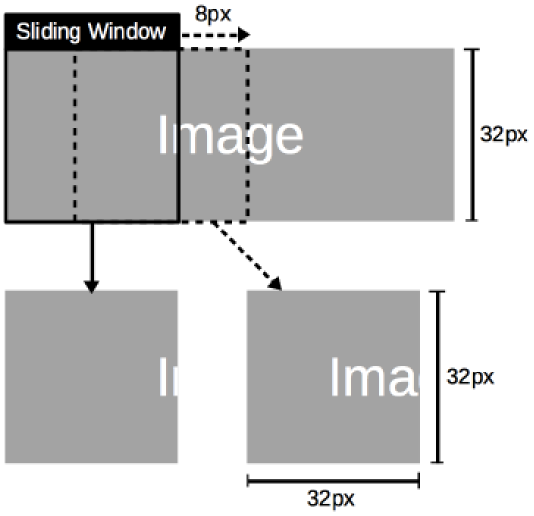

Figure 6: Graphical depiction of the sliding window.

sents the non-character symbol. Figure 7 shows a example with a batch size of 4.

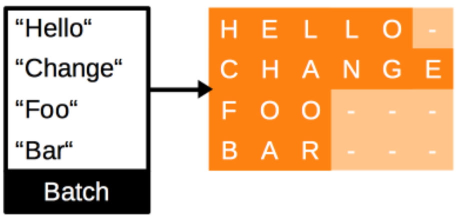

Figure 7: Implementation of label batches.

4.2 Implementation details of the CNN

The convolutional layer were implemented using the `conv2D()` function provided by the Tensorpack library and combined with a self-made `maxgroup()` function, since the tensorflow version had an error. The functionality of the maxgroup can be seen in figure 8. Instead of a softmax in the end, like in the original paper, we chose a fully connected layer with identity as activation function since it lead to a higher accuracy for the character recognition.

For the training we use a learning rate with exponential decay starting at 0.001 with decay rate of

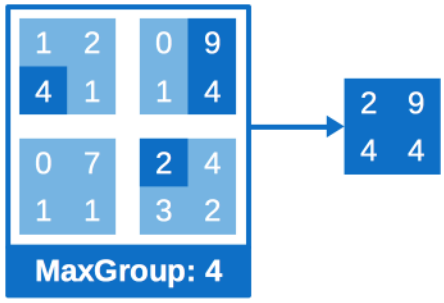

Figure 8: Example of a MaxGroup with group size 4.

0.99 and getting smaller every 5 epochs. As an optimizer we chose the `AdamOptimizer()` and used the prebuild `InferenceRunner()` from tensorpack to check for validation accuracy on a test set and prevent overfitting. Overall we trained for 1500 epochs and use the prebuild `SimpleTrainer()` from tensorpack.

4.3 Implementation details of the RNN

The RNN is implemented using the prebuild tensorflow functions `tf.nn.bidirectional_dynamic_rnn` with `tf.nn.BasicLSTMCell` for both directions. We use 128 units as described in the original paper and a tanh as activation function. For the fully connected layer on top we use `tf.contrib.layers.fully_connected` with identity as activation.
The CTC exists prebuild as well. For the loss we use the `tf.nn.ctc_loss` and `tf.nn.ctc_greedy_decoder` for the model. The greedy decoder is a variant of the `tf.nn.ctc_beam_search_decoder` function that is faster but less accurate. For our training the beam search decoder was too slow and in comparison did not add any visible improvements during the first couple of steps in training which lead us to using the greedy decoder.
For the training we use the Adam optimizer with a learning rate of 0.0001 and a momentum term of 0.9 over 1500 epochs. Since we divided our data set in test and train split we also use the `InferenceRunner()` class provided by the tensorpack library to prevent overfitting and check

for validation accuracy.

5 Experiments and Results

The test images solely came from the openly available IIIT5K data set. In the original study two other sets of images were additionally used but seemed to only contribute very small amounts of images. So only using the IIIT5K seemed sufficiently justifiable in this case. As the name suggests we initially have 5000 images of which a small amount was sorted out since they did not have 32 pixels in width. After segmentation the CNN trains on labeled 32×32 pictures and the RNN on around 2500 pictures with scene text due to the test train split. The experiments were conducted on the data sets described above using the training details listed in the respective implementation details section.
The CNN converges against an accuracy of 0.97 on the training set with a validation accuracy of 0.62. The process can be viewed in graphic 9.

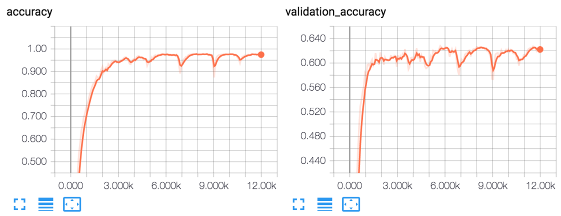

Figure 9: Training process for the CNN. Training accuracy on the left and validation accuracy on the right.

The validation accuracy seems rather low for a task like character recognition. This might be due to overfitting, very unclear images or just a suboptimal model. But note that the 5th and 6th layer of the CNN never get used at later stages anyways so only the accuracy of the 128D feature vectors matters. This cannot be determined directly but only through the loss of the RNN at later stages.
The RNN converges against an accuracy of 1 on the training set and against 0.66 for the validation. Note here that this is an increase compared to the 0.62 of the CNN solely, meaning the context of the word significantly added to identification of the letters. Additionally most words that are not correctly identified are only one letter off. For example the word ”wraps” was identified as ”wras” and ”chandigarh” became ”chandgarh”.The whole training process is seen in figure 10. Note that the validation error starts to get bigger after a certain period of time, indicating the point at which overfitting to the test data starts.

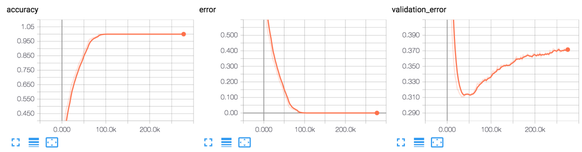

Figure 10: Training process for the RNN. Training accuracy on the left, error in the center and validation error on the right.

In comparison to the original, however, our results are rather unspectacular. Their accuracy was between 91,5 and 94 percent for the IIIT5K data set and even identified very ambiguous pictures.

6 Conclusion

The results that we achieved where slightly less impressive than those of the original publication. There are a couple of improvements that could be done in future research projects that we would like to outline. A first idea would be to train the network not in 2 separate steps but rather as one end-to-end process. This would likely result in better outcomes since the entire CNN would already get information from the RNN and CTC through backpropagation. A second idea is to use a bigger data set of images for training. The IIIT5K seems like a good benchmark but due to the test train split, overfitting might have occured which might be solved by significantly bigger datasets. A third possibility is to train the CNN and RNN on significantly lower sliding window frame sizes, i.e. 2 instead of 8
The paper we are replicating was published in summer 2015 which is already two and a half years back. In a field with such rapid changes like scene recognition, other projects have already further improved the methods that were revolutionary then. In the following we would like to have a closer look at some of the new ideas and improvements.
The first improvements were made by [YWZL17] who improved the sliding window by constructing it in a way that more closely resembles the

movements of saccades by biological eyes. They also added more layers to the CNN in the first step of the training. Lastly they improve the CTC algorithm to reduce the search space and make training with a beam decoder more effective. In the end they were able to improve accuracy from 94 to 98.9 and 91.5 to 96.7 percent on the IIIT5K dataset respectively. The second paper by [THH+16] is one conducted by partly the same authors that uses a similar approach to the original but contributed a new method which they call connectionist text proposal network (CTPN). The CTPN essentially is a convolutional network that already detects text structures within bigger image scenes. Due to this new technique they were not only able to find text within natural images, without cutting it out manually as in our paper, but also significantly improve nearly all benchmarks on the ICDAR data sets from 2011, 2013 and 2015. The last paper uses a slightly different approach on the just named ICDAR data sets. In [HHQY15] they only use a text-attentional convolutional neural networks for scene text detection without any recurrent part. They were able to break the benchmark of the ICDAR 2013 data set but were very quickly beaten afterwards by [THH+16] which uses an RNN. This last paper was mainly presented to show that there are were potent methods of classification that do not require a recurrent part out there.

References

[AP13] Ouais Alsharif and Joelle Pineau. End-to-end text recognition with hybrid HMM maxout models. *CoRR*, abs/1310.1811, 2013.

[GFGS06] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM.

[GTLL16] Qiang Guo, Dan Tu, Guohui Li, and Jun Lei. Memory matters: Convolutional recurrent neural network for scene text recognition. *CoRR*, abs/1601.01100, 2016.

[HHQ+15] P. He, W. Huang, Y. Qiao, C. Change Loy, and X. Tang. Reading Scene Text in Deep Convolutional Sequences. *ArXiv e-prints*, June 2015.

[HHQY15] Tong He, Weilin Huang, Yu Qiao, and Jian Yao. Text-attentional convolutional neural networks for scene text detection. *CoRR*, abs/1510.03283, 2015.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[LS16] Hui Li and Chunhua Shen. Reading car license plates using deep convolutional neural networks and lstms. *CoRR*, abs/1601.05610, 2016.

[MAJ12] A. Mishra, K. Alahari, and C. V. Jawahar. Scene text recognition using higher order language priors. In *British Machine Vision Conference*, 2012.

[SWX+14] C. Z. Shi, C. H. Wang, B. H. Xiao, S. Gao, and J. L. Hu. Scene text recognition using structure-guided character detection and linguistic knowledge. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(7):1235–1250, July 2014.

[THH+16] Zhi Tian, Weilin Huang, Tong He, Pan He, and Yu Qiao. Detecting text in natural image with connectionist text proposal network. *CoRR*, abs/1609.03605, 2016.

[YWZL17] Fei Yin, Yi-Chao Wu, Xu-Yao Zhang, and Cheng-Lin Liu. Scene text recognition with sliding convolutional character models. *CoRR*, abs/1709.01727, 2017.