Contents

List of Figures

List of Tables

Text Extraction from Infographics Using Neural Networks

Morten Jessen
Arbeitsgruppe Knowledge Discovery
Institut für Informatik
Christian-Albrechts-Universität zu Kiel
stu85924@mail.uni-kiel.de

ABSTRACT

Most of the recent text extraction methods [11] use pipelines based on Connected Component Labelling (CCL) to extract text from infographics. We propose a two-step neural network based pipeline. We localize the text using the foundation of Faster R-CNN [67], where we combined a Residual Network [39] with the Region Proposal Network (RPN). We use five infographic datasets and successfully implement simple data augmentation techniques to increase the size of the training datasets and improve the results of our pipeline. In a second step we use Tesseract 4.0 [7] in combination with data pre-processing on the predicted bounding boxes to extract the text from the infographics. We use the Average Precision (AP) measure to assess our localization results and adopt the precision, recall, f1, and Levenshtein Distance measures from Böschen et al. [11] to allow direct comparisons between the text extraction results. Additionally, we use Gestalt Pattern Matching (GPM) to gain insights into how similar the predicted text and the gold standard are. The comparisons show, that our pipeline achieves better results than the best pipeline presented in [11].

CCS CONCEPTS

• **Applied computing → Optical character recognition**;

KEYWORDS

Deep Learning, Neural Networks, Text Extraction, Scholarly Figures

1 INTRODUCTIONS

Infographics [12, 44] are the visual representation of information or data and are created with the intention of providing a quick and clear way to provide knowledge to a reader. They are graphical elements, that are used to illustrate complicated constructs or complex relationships and make them easier to understand. These summaries of information are often not repeated directly in the text, which makes extracting information from them even more important for the fully automated analysis of scientific literature [16]. The special difficulties associated with the task of text extraction are the complex distinction between graphic elements – like axes, markings inside a graph, etc. – and text with different fonts, letter sizes, resolution and orientation [11]. Additionally, there are very few annotated infographics available for training, which is often one the main hindrances when using neural networks.

The task of extracting text from images like infographics can be split into multiple steps. The first step finds Regions of Interest (RoIs) – areas of the input image that might contain text – which are then classified as containing text or not containing text. After this, there might be multiple pre- or post-processing steps which prepare the detected regions for an optical character recognition (OCR) engine. One way to do this, is to use a multi-step extraction pipeline [14] that uses clustering methods for region extraction and classification and one of the popular OCR engines to extract the text. We chose to split the task into two subtasks: Localization of text in images and Extraction of text from the localized bounding boxes.

The rise in popularity of neural networks provided very effective solutions to many object detection, segmentation, classification and localization tasks. Especially Fully Convolutional Neural Networks (FCNs) are very good at solving image processing tasks and have surpassed other machine learning algorithms in many – if not most – current problems like semantic segmentation, character/facial recognition, etc. [18, 56, 59, 69, 86]. But one of the main problems of solving new problems with FCNs – and neural networks in general – is the lack of annotated training images. Generally, the more training data available the better the model is going to be and most neural networks are trained with thousands of images. But for many problems the available datasets are very small and contain fewer than 100 images. Broadly speaking there are two reasons for this lack of training data. The first is the lack of annotations. Most annotations have to be created by hand and this task is very time consuming which often results in very few annotated images especially for projects that are not well funded. The second is the lack of training images which is common in datasets from the medical sector – where the available data is limited by privacy issues [8] or the complexity and cost of data collection [72] – or in specialized tasks like the classification of deep sea creatures where the opportunities to capture additional images are limited.

The recent successes of FCNs in object detection tasks raises the question whether or not they will perform similarly well for text localization tasks. To address this question, we choose a Residual Network (ResNet) [39] due to its recent successes [19, 40] and determine its usefulness for the localization of text in infographics. We test different versions of the classic ResNet to determine the optimal architecture for our problem.

We also address the question of how to deal with the problem of small datasets. The five main datasets used in this thesis contain only 633 images in total, which is low for training a complex neural network. We adopt multiple approaches to solve this problem in an attempt to find the optimal solution for text extraction from our datasets. First, we pre-train our neural network on one of the available bigger text-datasets. The main problem with this approach is, that the big datasets do contain fundamentally different images – either natural images or nothing but text, which

might not help our training process. Another approach is to use similar (small) datasets to pad our own datasets during training. Lastly, we apply different data augmentation methods to increase our training set without adding additional new data. Especially the data augmentation approach shows very promising results.

After the localization step has created bounding box predictions we use the Long Short Term Memory (LSTM)-based OCR Engine Tesseract 4.0 [7] to extract text from these boxes.

2 RELATED WORK

Text extraction from images has many different applications and there are many different approaches to solve this problem. Böschen et al. [11] conducted a comprehensive survey of the different approaches and locates them in a text extraction pipeline as shown in Fig. 1.

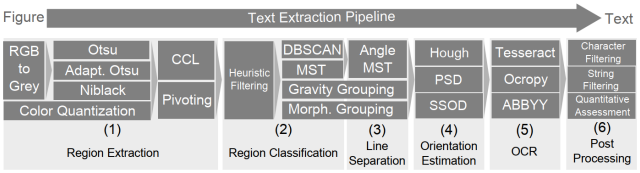


Figure 1: An overview of the text extraction pipeline of Böschen et al. [11].

Böschen et al. [11] implemented their own pipeline, called TX pipeline. It starts with an adaptive binarization [61] and then applies Connected Component Labelling (CCL) to extract regions. After processing the results with heuristic rules the regions are clustered using DBSCAN to differentiate text from graphical elements. Single lines of text are located using a minimum spanning tree from which the text is extracted using an OCR engine.

Other research aims to improve text extraction from natural images to automate updates for digital maps [80], or to translate image content to Braille language allow visually impaired to read the information contained in said images [48].

The technical approaches to solve these tasks are as diverse as the applications. Some of the approaches [28, 44, 45] are based on Connected Component Labelling (CCL) [70] which identifies connected regions inside of the image, and allows to identify text in images. Other approaches combine CCL with other methods to improve the results. Jayant et al. [48] use Support Vector Machines (SVM) to differentiate text elements from graphical elements. Lu et al. [58] use a Hough transformation on an edge image to create a feature vector which can be classified by CCL.

Our approach is to use neural networks to extract text from infographics. The work is based on the Faster R-CNN [66] architecture. We adopt the proposed Region Proposal Network, but use some of the improvements made by He et al. [40] for Mask R-CNN and the provided implementation. The results of the ICDAR 2017 [3] DeTEXT challenge show that TencentAiLab[1] uses a similar Faster R-CNN architecture for text extraction from infographics as well, but no paper has been published yet.

We split the pipeline into two parts: text localization (step one to three) and text extraction (step four and five). This approach has previously been used by Chen [20] and Jaderberg [47] when extracting text from natural images. Jaderberg also used shared convolutional layers to allow their network to do text localization and character classification within the same network architecture. This paper's focus is to improve the localization results of the first part of the pipeline using neural networks .

3 METHODS

As mentioned above, we are splitting the task of extracting text from infographics into two sub-tasks. First, we use a neural network to predict bounding boxes for text elements in the images. Second, we use an OCR engine to extract the text from the predicted regions. The localization task is split into two parts as well. We combine a Region Proposal Network (RPN) with a classification network that share their first convolutional layers as is shown in Fig. 2. The output of this block is used by the RPN to predict regions containing text which are then – combined with output of the convolutional block – classified into text or non-text.

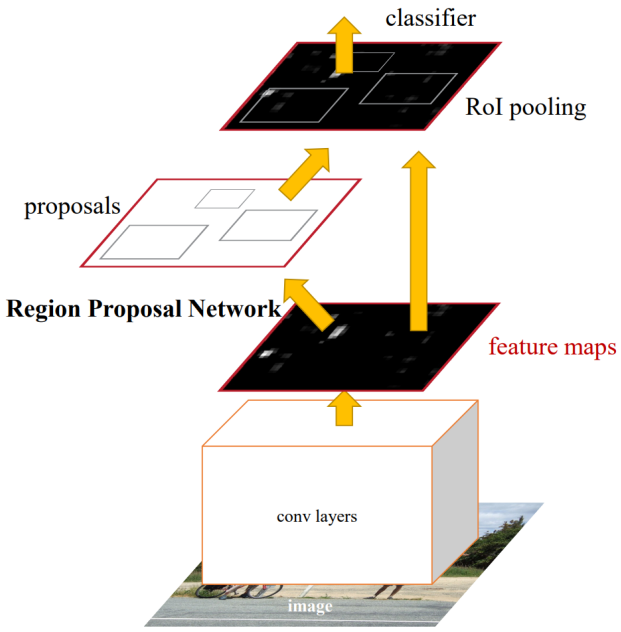


Figure 2: The Faster R-CNN Object Detection Network [67]. The Region Proposal Network is integrated between the Convolutional Layers and the Classifier. Both, classifier and RPN, use the CNN's output feature map. The original FR-CNN uses RoIPool to prepare the output of the RPN for the classifier.

In this section, we introduce the general concepts of the neural networks that we use and explain the role they play in the text extraction task. First we give a general introduction to Fully Convolutional Networks (FCN) and then explain how the localization network works. At the end of this section, we introduce our chosen OCR Engine.

[1] https://ai.tencent.com

3.1 Fully Convolutional Networks (FCN)

Convolutional Neural Networks (CNNs) are specialized neural networks for grid-like data like pictures – which are essentially 2- or 3-dimensional grids – that are able to learn features. The convolution operation calculates the output for a certain activation function by summing up the weighted values in the surrounding area of the previous layer. This area is called the receptive field of that activation function, because the calculated value is affected by these inputs. This process allows the lower layers to grasp local information while higher layers can use the condensed output of these lower layers to capture global information. This also reduces the amount of parameters needed by a network – which in turn decreases the computational complexity [35].

Like all other layers, convolutional layers are used to produce a new output map from an input map. In fully connected layers, this is done by having a learned weight for every input feature for every activation unit. This means that the weight matrix for each output feature has to be the same size as the input for each activation unit in every layer. For deep networks, this is computationally very expensive especially for higher resolution output maps. In convolution operations this weight-matrix – called kernel – is much smaller and is reused for the calculation of every element of the output layer [41]. This reduces the amount of parameters greatly while keeping the ability to use surrounding features to calculate an output feature. The kernel's values are usually learned during the training process.

The reduced size of the kernel allows using a lot of different kernels in the same layer to create multiple feature maps while keeping the computational complexity manageable. The use of many maps is infeasible in fully connected solutions with big kernels, due to the amount of parameters introduced by those layers [2]. The conceptual idea of using multiple feature maps is that different kernels find different features [51]. While lower layer kernels may find horizontal, vertical, or diagonal edges higher layer ones might learn to combine those edges to general object shapes or find class-specific features [51].

The last layer of CNNs can either be another convolutional layer or a fully connected layer that uses all the information contained in the previous output map to ascertain the class affiliation of each pixel. In fully convolutional networks, this last layer is a convolutional layer as well [56]. This allows the network to use input images of any size[2] and avoid the implied loss of spatial information due to the connection to all input values at once. Convolution, with its fixed receptive field, is better at using the relevant spatial data in a restricted area to make better segmentation decisions. Lastly, like explained above, the convolutional layer is much less computationally complex than the fully connected layer. The disadvantage of this procedure is that every feature on the output layer has a limited receptive field – meaning that the features of previous layers which affect its activation are limited to a certain area [86]. The difficulty is to find a balanced approach that increases the size of the receptive field on the one hand – providing global context – while not losing local information on the other hand.

[2]The input size for a specific model is still going to be fixed, but the same network could be trained with different sized input images without changing the architecture.

The text localization part of our pipeline is realized using two fully convolutional network architectures – one for classification, one to predict regions – which share some of their convolutional layers.

3.2 Text Localization

One of the improvements proposed for Mask R-CNN [40] is to replace the Zeiler and Fergus model [87] and Simonyan and Zisserman model [73] used in Faster R-CNN with a Residual Network [39].

The problem addressed by ResNet is the degradation problem. Based on the observation that a layer can assume the identity function – which would make it mathematically negligible – it would be reasonable to assume that adding additional layers to a network could either reduce the training error or not change it at all. But experiments performed by He et al. [39] show that this is not the case. Instead with the increase of the network's depth, the accuracy of the network first gets saturated and then – unexpectedly – drops rapidly. To avoid this behaviour, additional paths through the model are added which allow easy approximation of the identify function, if necessary. The basic building block is visualized in Figure 3.

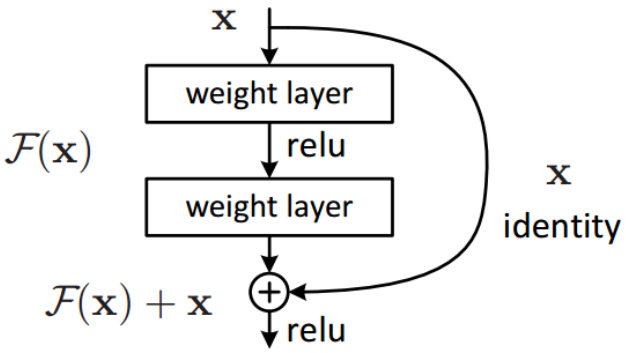


Figure 3: Building block for the ResNet. [39].

Instead of approximating a single function $H(x)$, the normal path through the layers appoximates $H(x) - x$. An additional circumventing path is added which is implemented as an identity function, which results in a final output of the unit of $F(x) = (H(x) - x) + x$. With this changed architecture it is very easy for the algorithm to approximate the identity function by setting the weights inside the traditional path to zero. Tests by He et al. [39] have shown that this method works and that ResNet can gain accuracy by increasing the depth of the network up to $\sim 1,000$ layers.

One of the very convenient features of the ResNet is that it consists of four different building blocks, which allows to easily construct networks of varying depth by simply stacking building blocks of the same shape on top of each other.

The initial input data is rescaled by a 7x7 convolutional layer and a stride of 2 with max-pooling to a size of 112px. The following path can generally be divided into four major parts. From one block type to the next the amount of kernels is doubled (256, 512, 1024,

2048) and the input size is halved (56, 28, 14, 7). The additional circumventing paths can be seen in the graphic as well.

We also considered an updated variant of ResNet called ResNeXt [81]. The novelty of this design is the addition of aggregated residual transformations to the original architecture. Xie et al. [81] present an architecture that uses the strategy of repeating layers on which VGG [73] and ResNet have been built and use it to implement a split-transform-aggregate strategy. For ResNeXt, the ResNet path with width 64 is split into 32 paths with width 4, the number of paths is called *cardinality*. All paths contain the same topology. This addition adds an additional way, next to *depth* and *width*, to change the general architecture of the network without adding much complexity. Xie et al. [81] show that this approach can help to decrease the validation error and generally improves the results.

3.2.1 Region Proposals. Our region proposal method is based on Faster R-CNN [66], which improves the original Fast R-CNN [30] by replacing the region proposal bottleneck caused by methods like Selective Search [76] and Edge Boxes [89]. Like Ren et al. [66], we use a Region Proposal Network (RPN) in combination with a Feature Pyramid Network (FPN). Computing the region proposals with a deep network has the advantages of being able to reuse parts of the object detection network and take advantage of GPUs [66]. The sharing of convolutional layers at test-time allows to compute proposals in $\sim 10ms$ per image – compared to $\sim 2s$ for Selective Search and $\sim 0.2s$ for Edge box. Instead of using a combined loss function, we alternate between fine-tuning for region proposal and object detection. The RPN uses a sliding window approach and slides a small network over the last shared convolutional layer.

We also replaced RoIPool [30] with RoIAlign [40] as is proposed for Mask R-CNN [40]. RoIAlign removes the quantization performed by RoIPool. RoIPool quantizes a floating-number RoI – see Figure 4 – to a discrete number area of the feature map and then subdivides the area into spatial bins – quantizing again. RoIAlign avoids harsh quantization by using bilinear interpolation to compute the exact values of the input features at four regularly sampled locations in each RoI bin and aggregate the result (using maximum or average pooling) [40].

3.3 Text Extraction

We use a multi-step text extraction process that stops as soon as the confidence in the prediction – provided by the OCR engine – is $\geq$ 96%. We start with the originally predicted bounding box with an added white border and then, if our confidence limit is not met, continue with the pre-processed and modified images as input for Tesseract. We resize the image, use thresholding mixed with gaussian blur to improve text segmentation in image and rotate the image. While thresholding and resizing are only used to improve extraction results, rotation is necessary for our pipeline, because our text localization network does not detect the text's rotation. The rotation is split into multiple steps: First we use only multiples of 90°and halve the steps until we reach 15°steps. Finally, if after all these steps the confidence is still < 90% we repeat the process, but set Tesseract to Single-Character mode. We do this, because our experiments show us, that most of the previously unreadable text bounding boxed contained single characters, which can not be read in Tesseract's standard mode. We use the OCR Engine Tesseract 4.0

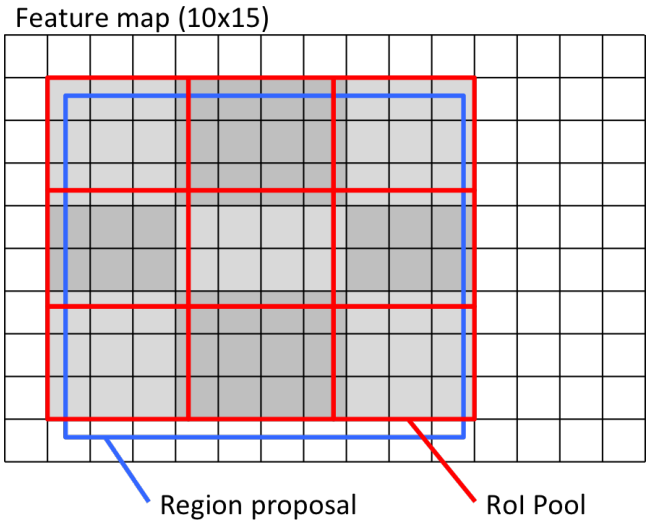


Figure 4: RoIPool's quantization of the region proposal. [5].

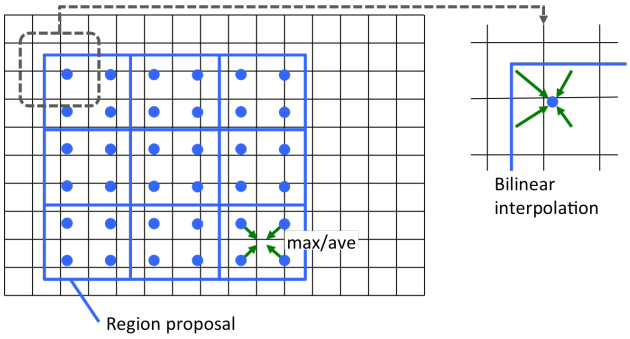


Figure 5: RoIAlign's method of not quantizing the region proposal. [5].

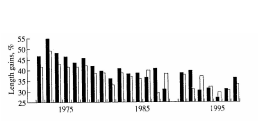
[7], which is pre-trained for English language, to extract text from the bounding boxes that we detect in the infographics.

4 EXPERIMENTAL SETUP

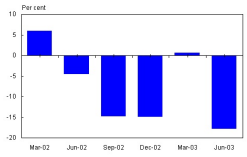
In this chapter, we introduce the configurations that we use for our experiments. For our localization optimization, we use two images, chosen uniformly at random, per gradient descent step. The images are resized to their maximum possible size, which is restricted by GPU memory. When resizing to $x \in \mathbb{N}$ pixel we resize the *shortest* side of the image to be x pixel long. We keep the aspect ratio of the image and cap the longest side at $\sim 2 \cdot x$ pixel, thus the shortest side may actualy be shorter than x pixel. We sample 512 Regions of Interest (RoIs) per image with a ratio of 1:3 [31] positive to negative. A RoI is positive, when it contains foreground object class examples and negative when it contains background object class examples.

4.1 Datasets

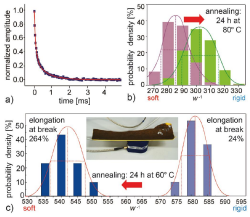
We use five relatively small datasets containing infographics for our training and test process. CHIME-R is a set of 115 real images from



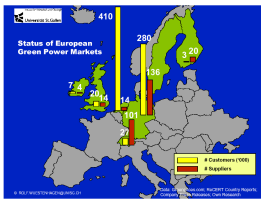
(a) CHIME-R Example.



(b) CHIME-S Example.



(c) DeGruyter Example.



(d) EconBiz Example.

Figure 6: Example Images for CHIME-R, CHIME-S, DeGruyter and EconBiz.

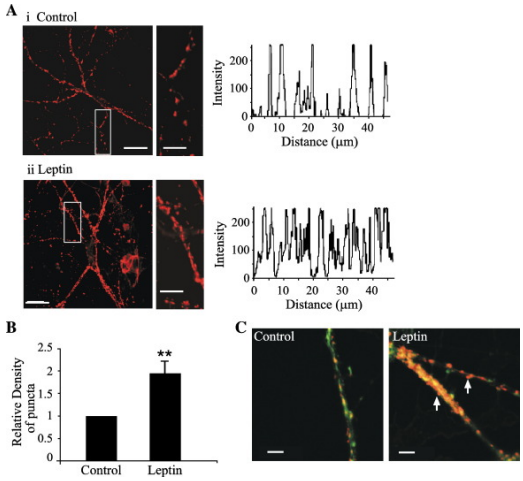


Figure 7: Example Image for DeTEXT.

the Internet or scanned from paper. The gold standard was created by Yang Li [84]. CHIME-S is a set of 85 synthetically generated images, which are very similar to the CHIME-R dataset. The gold standard was created by Zhao Jiuzhou [49]. The DeGruyter dataset contains the ground truth information of 120 figures from academic books provided by DeGruyter. And the EconBiz dataset contains 121 randomly extracted scholarly figures from a corpus containing 288, 000 open access publications in the economics domain. The fifth dataset that we use is the DeTEXT dataset [85]. This dataset contains the ground truth information of 192 biomedical images and is part of the ICDAR robust reading competition [3].

CHIME-R and CHIME-S contain only images of bar, pie, and line charts. The infographics in DeGruyter, EconBiz and DeTEXT are more complex and versatile. All three contain bar, pie, line, scatter, flow/process charts and histograms. They also add real pictures or abstract visualizations to these charts or have multiple different kind of charts in one image. EconBiz contains multiple map based infographics and DeTEXT includes many medical image – real and abstract – based infographics and one map based image as well. DeGruyter, EconBiz and DeTEXT combine different types of chart or image in one infographic. Example images for all five datasets can be seen in Figure 6 and 7. Additionally, we experimented with pre-training our network on bigger datasets to remedy the lack of available training data. We used more general text-based datasets to fulfill the pre-training role for our network, because there are no big, annotated infographic datasets. Pre-training on big datasets is a very common method for neural network training and visual tasks. The idea behind it is to prepare the network to detect general image structures which can be fine-tuned with the actual dataset later.

COCO-Text is based on the MS-COCO Dataset [52], which is a general object detection dataset and consists of more than 200, 000 labelled natural images with more than 1, 500, 000 individual object annotations in 80 object categories. We used version 1.3 of COCO Text which uses the MS-COCO images but adds bounding box annotations for text occurrences in these images. The annotations consist of 145, 000 text annotations for 63, 686 images which results in an average of 2.28 text annotations per image. We only used a smaller subset of those images namely the ones that contained text that is English, machine written and legible. This reduces the dataset to 13, 966 training and 3, 271 test images with 63, 545 and 14, 644 text annotations respectively.

We also used a dataset which is designed for text recognition tasks. The Total-Text-Dataset [24] contains photographs like MS-COCO, but is designed for text extraction. It contains 1, 555 images with an average of 7.37 annotations per image – compared to 2.73 for COCO Text. The annotated text is multi-oriented and sometimes curved, which is not the case in COCO-Text.

4.2 Metrics

We decided to use the Average Precision (AP) metric for our experiments, which has been used in similar object detection tasks and challenges [3, 32]. We use AP, AP50 and AP75. AP50 counts a detected instance as correct if it has at least 50% Intersection-over-Union (IoU) with the ground truth mask it has been matched with [29]. AP75 does the same, but with 75%. AP is a summary metric that combines ten equally spaced IoU thresholds from 50% to 95% (0.5, 0.55, 0.60,...,0.90,0.95). Figure 8 shows random examples for IoU values of 0.5, 0.7, 0.9. While an IoU of 0.5 is sufficient in most cases to capture the text, an IoU of $\geq$ 0.7 ensures that the whole text is captured . Additionally, we are evaluating our text extraction

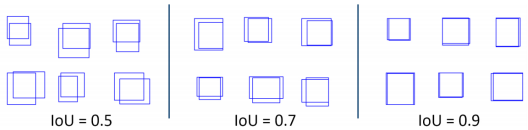


Figure 8: Examples for different Intersection over Union (IoU) values. [89].

results with the measures used by Böschen et al. [11], to allow us to compare their results to ours. They use precision, recall and

F1-measure to evaluate the localization part and the Levenshtein distance to evaluate the text extraction of their pipeline. Precision calculates the proportion of true positives to all positives, i. e. precision denotes which percentage of the predictions are correct. Recall compares the true positives to all relevant elements – true positives and false negatives – and gives the percentage of how many of the ground truth bounding boxes have been found. F1-measure is a combination of Precision and Recall to provide a single measurement to evaluate a system. Levenshtein distance calculates the single-character-edit-distance between two strings. Single-character-edits are insertion, deletion and replacement. Böschen et al. use a local and a global variant. The local Levenshtein distance (LSD) calculates the distance between one text instance and the corresponding prediction. The global Levenshtein distance (LSG) sorts all predicted text alphabetically and combines it into one global prediction and compares this to a gold standard of the same kind.

Additionally, we decided to use Gestalt Pattern Matching (GPM) to evaluate our extraction results. Levenshtein distance provides the minimal edit sequences from one string to another, which does not represent a subjective impression of similarity. Gestalt Pattern Matching finds the longest contiguous substring that is contained in both strings, and repeats this process recursively to the left and right of the matching string. This method values strings that *look similar* to each other higher. GPM measures the correctness of the word in relation to the total length of the prediction and gold standard, which gives a better insight into how correct a prediction is. It is important to note, that the Levenshtein distance does not account for word length at all, even though a LSD of 3 is obviously worse for a three letter word, than for a 20 letter word. As a rule of thumb values ≥ 0.6 [4] means the strings are close matches.

4.3 Pre-Processing

The biggest problem during localization, next to graphical elements – like circles or triangles – being mistaken as text elements is the detection of white text on a colorful background – see Figure 9. To alleviate this problem, we decided to add pre-processing in front of our pipeline, and convert the training and test images to grayscale. Using the findings of Kanan and Conttrell [50] we chose to use the mean of the RGB channels method.

4.4 Experiments

To ensure a fair test environment, we use the same datasets and same parameters like training length and learning rate during our experiments. We split the datasets into training- and test-set with a ratio of 4 : 1. For our final tests, we perform 5-fold cross validation: We repeat the training and test steps five times by selecting a different fold as test data in each pass. Finally, we compare the average of our 5-fold cross validation extraction results to Böschen et al.'s [11] results to determine the effectiveness of our pipeline.

For our text localization network, we chose a learning rate of 0.0075, used 512 RoI proposals in a Residual Network with 101 layers. We trained our network for 250,000 iterations, which is equal to ~ 158 epochs for the fully augmented dataset.

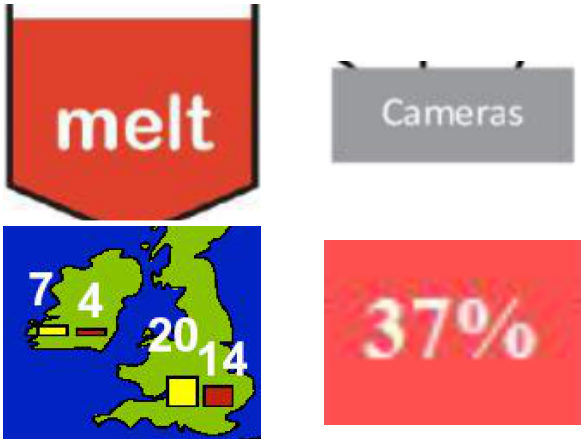


Figure 9: Examples for text elements that are challenging to localize. The common features of these images are that they are on differently colored backgrounds and the text elements are brighter than the background. To solve the first problem, we decided to convert all our images to Grayscale. To alleviate the second problem, we introduced inverse images in our text extraction process.

5 RESULTS

Before we conducted the final experiments, we determined the effectiveness of different configurations, data augmentation methods and the generalization capabilities of our pipeline.

5.1 Network Architecture

After choosing ResNet [39] as our base model, we experimented with networks with the most common configurations of 50 and 101 layers. While it is possible to create ResNets with as low as 18 or as high as 1,000 layers, both seem to be unfitting for our specific problem. Decreasing the network's depth to less than 50 layers provides only a marginal performance boost, while our very limited training data seems to be insufficient to effectively train a network with more than 100 layers.

Additionally, we have tested an advanced and newer variant of the ResNet called ResNeXt [81] with the same depth configurations. We have decided to use a cardinality of 32 and 64 and bottleneck width of 8d and 4d respectively in accordance with the findings of Xie et al.[81].

Please note, that due to GPU memory restrictions we had to use different input image resolutions for different network architectures. To accommodate this, we choose to test the network variants with the same resolution restrictions and with their particular maximum resolution. While the test with restricted resolutions might not be crucial for our experiments, it might be interesting to see the performance differences between the network with the same resolution restrictions.

The results with all networks being restricted to 600 pixel is shown in Table 2. The best networks are the ones using the original ResNet50/101 variant. With lower resolution images, the 50 layer version slightly outperforms the deeper one in the more difficult AP

	AP	AP50	AP75
ResNet50 (800px)	53.15%	**91.79%**	54.65%
ResNet101 (700px)	**53.94%**	91.58%	**58.51%**
ResNeXt50 32x8d (750px)	51.26%	89.44%	53.14%
ResNeXt50 64x4d (750px)	51.50%	89.38%	53.66%
ResNeXt101 32x8d (600px)	49.70%	88.60%	50.41%
ResNeXt101 64x4d (600px)	50.73%	88.84%	51.05%

Table 1: Testing different neural network architectures for our pipeline. All networks used their respective maximum input image resolution on our hardware. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz. We used AP, AP50 and AP75 measures to compare the results.

	AP	AP50	AP75
ResNet50	**52.49%**	89.81%	**54.11%**
ResNet101	51.46%	**90.19%**	52.58%
ResNeXt50 32x8d	49.21%	88.11%	48.02%
ResNeXt50 64x4d	49.42%	87.86%	49.16%
ResNeXt101 32x8d	49.70%	88.60%	50.41%
ResNeXt101 64x4d	50.73%	88.84%	51.05%

Table 2: Testing the effectiveness of different neural network architectures for text detection. Input images have been fixed to be 600px wide. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz. We use AP, AP50 and AP75 measures to compare the results.

and AP75 metrics, while ResNet101 is slightly better when using AP50. The ResNeXt networks performed worse in all experiments.

We tested the networks by setting their individual input image resolution to the maximum value – only restricted by GPU memory. The results can be seen in Table 1. Overall, the classic ResNet variants are still providing the best results. But ResNet101 leads in the harder AP and AP75 metrics while ResNet50 delivered the best AP50 results. The most important improvement is in the AP75 metric, where ResNet101 gained 6% and has proven itself as the superior network architecture for this task. It is also noteworthy, that ResNet101 has to be restricted to 700 pixel images, while ResNet50's input can be increased to 800 pixel, however ResNet101 does gain more accuracy by increasing the resolution by 100 pixel than ResNet50 by an 200 pixel increase.

Lastly, we tested the effect of using augmented data on all different networks. Out initial tests show that ResNet50 surpasses ResNet101 in those experiments, if the training duration is relatively short – up to $\sim 100,000$ iterations – which might be interesting when the training-speed is important. The results – after increasing the training to $250,000$ iterations – is shown in Table 3. While the differences are relatively small, ResNet101 provides the best results in all three metrics, which is why we chose to use it in all our following experiments.

5.2 Data Augmentation

One of the biggest problems we faced was the general lack of annotated training data for our networks. Our datasets combined

	AP	AP75	AP50
Results without Augmentation	53.94%	58.51%	91.79%
ResNet50	64.35%	74.29%	93.95%
ResNet101	**64.89%**	**74.82%**	**94.98%**
ResNeXt50 32x8	62.70%	71.40%	93.56%
ResNeXt50 64x4	61.22%	70.92%	93.49%
ResNeXt101 32x8	61.93%	70.08%	94.49%
ResNeXt101 64x4	61.01%	69.73%	94.85%

Table 3: Comparing the effect of our augmentation methods with different neural network architectures. All networks have been trained for $250,000$ iterations. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz. We compared the results using AP, AP50 and AP75 measures.

contain only 633 images which we had to split into training and testing datasets. This left us with 507 images for training and 126 images for testing the models.

We used different approaches to improve our training results. First, we pre-trained our network on COCO-Text [77] and TTD [25]. While we assumed that pre-training the network would increase the general results, the low average of annotations per image – compared to our datasets – and the nature of the images – natural images – suggest that there are big differences in the detection tasks and the learned structures might not transfer between the tasks. Our test results are shown in Table 5 and demonstrate that the pre-training is effective. Both TTD and COCO-Text improve the test results, with COCO-Text providing the best results by improving AP, AP75 and AP50 by $\sim 5\%$, $\sim 9\%$ and $\sim 2\%$ respectively. Our next approach to improve the effect of the training data, was to use data augmentation techniques. We used different methods to increase our training dataset size and verified the effect of them one by one. In our final augmentation composition, we added rotations – $90°$, $180°$, $270°$– and random scaling of the input images to 400, 600 or 700 pixel. We also added horizontally or vertically flipped images, translated the images using two different translation matrices, and added Gaussian or salt and pepper noise to our data in order to increase general training results and robustness.

The results can be seen in Table 6 and indicate that these comparatively simple augmentation techniques are very effective for our problem. AP increases by $\sim 12\%$, AP50 increases from a good 90.34% to a convincing 92.88% and the results for the AP75 metric increases by $\sim 15\%$ to a final value of 67.57%. This indicates that the addition of augmented data does not only increase the general detection rate, but is even more useful to increase the detection quality. We also examined the effect of using multiple datasets or only one dataset for training purposes. This gives a general indication of likeliness between the different datasets – networks trained on a dataset that is similar to the test dataset will perform better than dissimilar training/test sets. Additionally, this gives an indication on the effect of adding additional datasets to our training set. We trained the network on all five individual datasets – i. e. CHIME-R, CHIME-S, DeGruyter, EconBiz and DeTEXT – and on all five datasets combined. Afterwards, we compare the test results on all individual and combined datasets to determine the effect of the different datasets on our training process.

Trained on	Results AP75					
	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT	Combined
CHIME-R	92.77%	61.19%	12.20%	2.62%	33.83%	40.52%
CHIME-S	81.23%	**88.72%**	7.27%	1.56%	32.20%	42.20%
DeGruyter	16.68%	1.05%	66.78%	56.31%	11.32%	30.43%
EconBiz	1.11%	0.96%	57.94%	**98.89%**	8.77%	33.53%
DeTEXT	40.10%	33.56%	18.11%	5.00%	**67.15%**	32.78%
All Combined	**96.48%**	64.63%	**70.17%**	94.77%	55.09%	**80.02%**

Table 4: Comparison: Results for training on single datasets for $100,000$ iterations on ResNet101. The experiments show similarities between the different datasets. We used the AP75 measure to compare the results.

Pretraining:	none	on TTD	on COCO-Text
AP50	91.35%	94.49%	**95.21%**
AP75	63.49%	75.51%	**76.33%**
AP	58.37%	65.73%	**65.98%**

Table 5: Comparison of the AP, AP50, AP75 measures for ResNet101 with and without pre-training on COCO-Text or TTD.

	AP	AP75	AP 50
no augmentation	52.90%	53.02%	90.34%
augmentation	**60.81%**	**67.57%**	**92.88%**

Table 6: Experiment that shows the effect of our data augmentation methods on the AP, AP50 and AP75 measures when training on ResNet101.

The results can be seen in Table 4. The first observation is that there seems to be a big difference between the CHIME datasets and both the DeGruyter and EconBiz dataset. While training on one CHIME set yields acceptable detections on the other CHIME set, the results for EconBiz and DeGruyter are very poor. This is expected, because CHIME-S consists of synthetic data which is generated to be similar to CHIME-R. Interestingly, a similar relation holds true for EconBiz and DeGruyter. Training on one of these datasets gives acceptable detection results for the other, but very poor results for both CHIME datasets. DeTEXT does not show these big discrepancies and the test results are similar to each other.

The combination off all datasets improves the results on CHIME-R, CHIME-S and DeTEXT, and only performs slightly worse on DeGruyter and EconBiz. As expected, the results, when testing on all five datasets, improve by a factor of 2, when when the network was also trained on all datasets.

It is promising that our network architecture is quite adaptive – only loosing $1\% - 2\%$ accuracy on DeGruyter and EconBiz, but gaining up to 10% precision on the other datasets.

5.3 Generalization

We also tested the generalization capabilities of our pipeline. Because of the lack of additional annotated data, we decided to perform five generalization tests. For each test, we choose one of our datasets for testing and train our pipeline on the remaining four.

Tested on	AP50	AP75	AP
CHIME-R	80.45%	45.19%	45.82%
CHIME-S	87.73%	30.59%	41.12%
DeGruyter	86.63%	35.88%	43.06%
EconBiz	84.61%	15.88%	34.03%
DeTEXT	70.32%	23.41%	33.21%

Table 7: Testing the generalization capabilities of our localization network. We used pre-training on COCO Text and ResNet101 as backbone. The network is trained on four of the datasets and tested on the fifth. We compare the results using AP50, AP75 and AP measures.

The results are shown in Table 7. The generalization results on CHIME-R, CHIME-S, DeGruyter and EconBiz look promising. The AP50 measure of $\geq 80\%$ is enough in most cases to extract the text even though the bounding box precision is much worse compared to our previous tests. The results of DeTEXT are different and only reach an AP50 value of 68.67%. The difference between DeTEXT and the other datasets is, that the former's infographics contain many complex graphical elements and real life medical images, which are very rare in the other datasets.

5.4 Final Results

For our final results, we conduct a 5-fold cross validation. The average of these results can be seen in Table 8, which is our final result and point of comparison. We compare our text extraction results to the BS-4OS-O pipeline [11], because it surpassed the other text extraction approaches in Böschen et al.'s survey [11]. The results are shown in Table 9.[3] Our pipeline improves the results in all measured categories. Precision and recall increase by 19% and 20% respectively, the F1 measure improves accordingly. The average local Levenshtein Distance is reduced by 1.26 and the global LSD is halved when using our pipeline compared to the BS-4OS-O pipeline.

Additionally, we use Gestalt Pattern Matching to get additional insight into our text extraction results. Our pipeline achieves an average of 0.84 which is far above a good result of 0.60. This result

[3]During the last tests of our network, we discovered that one of the training and test set images of DeGruyter were corrupted during their initial transformation from transparent png to jpg. These images were corrupted for all previous tests and therefore did not change relative results. For our final Tests we chose to repair the corrupted image, which resulted in better results (previously there were 0 correct detections in the corrupted image, after repairing all text was located correctly).

	Average (SD)
AP50	94.71% (0.89)
AP75	78.48% (2.36)
AP	67.16% (0.76)
Precision	0.86 (0.032)
Recall	0.83 (0.034)
F1 (SD)	0.87 (0.0279)
LSD_avg (SD)	3.44 (0.47)
LSG (SD)	39.11 (5.92)
GPM_avg	0.8454 (0.0287)

Table 8: Average results of our final 5-fold cross validation experiments. We pre-trained our ResNet101 based network on COCO Text, used our data augmentation methods, preprocesing for Tesseract 4.0 and the previously determined optimal hyperparameters. We use all measures to evaluate these experiments.

means that we do not only have short minimum edit sequences, but the predicted words are very similar to the gold standard.

6 DISCUSSION

Comparing the different configurations of our localization network leads to the conclusion, that the added layers in ResNet-101 improve the results compared to ResNet-50. Interestingly, the added complexion of ResNeXt in all tested configurations did not result in the same positive effect. Both observations are true when using only the basic datasets and when increasing the training set size by data augmentation techniques.

The implemented data augmentation methods emerge as being very effective at increasing our test precision and quality. Despite using – comparatively – simple methods like rotations, adding noise, resizing or flipping the image, the effect of these methods was very convincing. It would be interesting to see the effect of more complex augmentation methods, like creating additional training data with GANs [36] or straightforward synthetic data generation methods [38, 46, 49] and see if this improves the results and generalization capabilities of our pipeline even further.

Comparing our results to the survey paper by Böschen et al. [11] shows that our results are very competitive. It is important to notice that there are fundamental differences between the approaches presented in the survey and our pipeline. Namely, we use a neural network that needs training data and limits our ability to test our methods on all the data contained in the datasets. But at the same time, this difference could also be seen as an advantage, because the effectiveness and generalizability of our networks mostly depends on the available training data. As a result, the advantage of our pipeline compared to the TX pipeline could be improved further by adding additional datasets to our training data.

While the generalization capabilities of our pipeline are good, there is – as may have been expected – a problem with generalizing to new types of content. Using our trained pipeline on infographics that contain charts or other graphical elements that were not present in our training set might be problematic. It is easy to see these problem in Table 4 where training on CHIME-R or CHIME-S – both of which only contain bar, pie and line charts – produces very poor results when tested on the other, more diverse datasets.

It is also apparent, that the pipeline performs better on infographics that only contain (simple,) graph-like structures. The results on CHIME-R, CHIME-S and DeGruyter are much better than the results on EconBiz or DeTEXT. The former contain mostly graphs, while EconBiz has many more elaborate graphical elements (like maps), and DeTEXT's infographics include many medical images. The quantity (AP50) of localized text when comparing these two subsets is only different by $\sim 5\%$, but the difference in quality of the detections (AP75) is as high as $\sim 40\%$. While the results are precise enough to extract the correct text in most cases, this implicates a problem with complex structures, which might be solved by adding more complex training data. The generalization test shows that our pipeline performs $10 - 19\%$ worse when testing on DeTEXT compared to the other four datasets. Interestingly, the pipeline performs as well on EconBiz as on CHIME-R, CHIME-S and DeGruyter. The biggest problem for the generalization of our pipeline seem to be the complex medical images contained in DeTEXT, which might indicate a more general problem with real life images.

7 CONCLUSION

Our experiments show, that it is very effective to use neural networks for text extraction from infographics. Comparisons with Böschen et al.'s survey [11] indicate, that our pipeline has a big advantage over the presented CCL based methods. At the same time, they underline one of the main problem of neural networks: a lack of suitable training data. Our non-augmented tests already had convincing AP50 results, but they generally lacked in prediction precision, which is indicated by AP75. Our added data augmentation methods show, that our pipeline scales well with added training data and indicate that the results could improve again by adding more data or more complex data augmentation methods.

While the datasets were limited, our pipeline performed well on generalization tasks. Compared to our previous tests, the generalization results lacked in the AP75 measure, but retained (mostly) convincing results when using AP50. The only problem we encountered with the pipeline is generalization to data which contains a lot of complex real life/medical images, but we are positive, that adding more images of that type to the training set could improve the results in that regard.

	Precision	Recall	F1 (SD)	LSD_avg (SD)	LSG (SD)
BS-4OS-O pipeline [11]	0.67	0.63	0.67 (0.22)	4.71 (4.66)	95.49 (94.80)
Our pipline (average)	**0.86**	**0.83**	**0.87 (0.11)**	**3.44** (5.15)	**39.11 (40.55)**
Changes	**+0.19**	**+0.20**	**+0.20 (-0.11)**	**-1.26** (+0.49)	**-56.38 (-52.27)**

Table 9: Comparison of the best pipeline from Böschen et al.'s survey [11] – BS-4OS-O – and our pipeline. We use all measures that both papers have in common to compare the experiments. Our results are the average of our final 5-fold cross validation.

A EXTENDED RELATED WORK

One topic that has not been mentioned before is the usage of the term *infographic*. We use this term based on the usage by Böschen et al. [11]. There are several different terms that are used in other scientific publications, which describe the same type of image we call infrographic. Carberry et al. [16] call them *information graphics*, Chen et al. [21] *data-driven diagrams*, Ray et al. [63] *figures* and Huang et al. [45] simply *charts*.

Böschen et al. published multiple papers [11–14] on the topic of extracting text from infographics. We compare our results to the newest and most extensive article [11], which is in itself a comparison of 32 different approaches for text extraction from infographics. They implemented an adaptive pipeline, that allows to freely combine different modular parts with each other. They evaluated 44 different configurations of the pipeline and we compare the results of our own pipeline to their best configuration. Four of the five datasets we used, have been used by Böschen et al. as well. Additionally, we adapted F-measure, precision, recall and Levenshtein distance from Böschen et al.'s paper to compare the results to each other.

There are several different approaches to text extraction, most of which are already part of Böschen et al.'s survey [11]. It is still interesting to take a cursory look at them. In 1995 Deseillingy et al. [28] describe a rotation-invariant connected components analysis module, that is able to automatically read character strings from scanned topographic maps. Chester et al. [22] designed a system in 2005 to allow sight-impaired users to access the textual information contained in infographics. Their goal was an interactive system that allows sight-impaired users to selectively access the information. Additionally, they constructed an *evidential reasoning component* which tries to hypothesize the text message that is contained in the infographic. In the same year Gllavata et al. [33] implemented a text extraction system to help create automatic content based annotations and allow retrieval of images. They used clustering to segment the text elements and a rating scheme to classify the individual clusters. The test set of this paper contained only 2, 684 characters. In 2007 Jayant et al. [48] wanted to translate scientific infographics into a tactile form suitable for blind students, i. e. into Braille. Their Tactile Graphics Assistant uses machine learning, computational geometry and optimization algorithms to transform text into Braille. They translated more than 2, 300 infographics from four different textbooks and need ~ 10 minutes per image. Lu et al. [58] aims to convert text in infographics into a machine-processible form, to allow automatic data analysis and comparison with other data. They describe a system that extracts text automatically by using a supervised learing-based algorithm – i. e. machine learning – to classify figures into five different classes: photographs, 2D plots, 3D plot, diagrams and others. After the classification they extract numerical data from data points, lines and their labels. Xu et al. [82] analyze biomedical images, because they often contain the key findings of a scientific paper. They introduce a new text detection algorithm for text extraction based on iterative projection histograms and achieve an F-measure of 0.60 for biomedical infographics on their dataset. In 2012 Carberry et al. [17] published a paper with the focus on helping individuals with disabilities. They present an intelligent interactive system, that helps visually-impaired individuals to access the content of infographics. One year later, Sas et al. [71] present an approach to localize text elements in infographics. They propose a three-stage method using OCR engines to detect text regions. In 2015, Chiang et al. [23] try to use small amounts of user effort to semi-automatically recognize text labels in maps. They use cartographic labeling principles to locate text, rotate the text and then use an OCR engine to extract the text. In the same year, Lu et al. [57] first create candidates for text boundaries using image edges and then extract the text from these candidates using a support vector regression model. They achieve an F-measure of 78.19% for the scene text detection and segmentation masks. Finally, Olszewska [60] present a OCR engine based approach in 2015, that automatically extracts digits from images and videos. They use active contours for detection and recognize the characters with template matching. They tested their approach on sport team players' numbers. In 2016 Gomez et al. [34] used agglomerative similarity clustering on regions to identify region groupings that correspond to text. They identified differently oriented text occurrences by converting the original image into specifically designed feature maps. The authors use these feature maps to create similarity hierarchies, which are then used to produce text group hypotheses. This approach is combined with a stopping rule, which combines a discriminative classifier and a probabilistic measure of group meaningfulness. In the same year, Zhang et al. [88] extract text from natural images by combining a fully convolutional neural networks with text line hypotheses. First, they use a FCN to predict map regions in the picture. These regions are then used to create text line hypothesis. Lastly, they use an additional FCN classifier to predict the centre of the individual characters in order to remove the false hypotheses. The approach produced state-of-the-art performances on several text detection benchmarks, including ICDAR2015 and ICDAR2013. In 2018 Yan et al. [83] presented a text extraction approach specifically designed for minority languages. They used the Uyghur language that is spoken in the Xinjiang Uyghur Autonomous Region of Western China by 10 – 25 million people. They propose an effective Uyghur language text detection system for images with complex backgrounds. The approach is designed with intelligent vehicles in mind, which should be able to detect (and extract) text in the everyday world. They use a new channel-enhanced maximally stable extremal regions (MSERs) algorithm to detect text candidates. After the MSERs, they implemented a two-layer filtering mechanism, which removes non-character regions. Then they connect the predicted components into chains. Lastly, another two-layer filter removes non-text chains. Their system reaches an F-measure of 85% on their own dataset.

We chose to use Faster R-CNN [67] and ResNet [39] in our pipeline. But there are several alternative architectures and neural network models for object detection. Lin et al. [54] developed RetinaNet. In contrast to R-CNN RetinaNet is a one-stage detector. RetinaNet uses a new Focal Loss, that focuses training on a small set of hard input images and prevents the easy negatives from overwhelming the detector. This is done to avoid a foreground-background class imbalance during training, that usually makes one-stage solutions inferior to two-stage solutions. This allows RetinaNet to perform like a two-stage network. The most interesting

development from this paper is surely the Focal Loss, which might be applied to other approaches as well.

Dai et al. [26] presented the so called R-FCN, which is a region based fully convolutional network. In contrast to Fast R-CNN and Faster R-CNN, this network shares almost all computation on the entire image – instead of repeating calculations in per-region subnetworks. The authors introduce position-sensitive score maps to avoid a conflict between translation-invariance in image classification and translation variance in object detection.

Another interesting development are the YOLO Architectures [64, 65] by Redmon et al.. The initial You Only Look Once (YOLO) [64] approach repurposes classifiers to perform detection instead. Object detection is reformulated into a single regression problem that uses the input pixels to determine the bounding box's coordinates and the classification. YOLO is – like R-FCN – a one-step approach that is optimized for speed. The network can process images in real-time at 45 frames per second, which can be increased to 155 frames by using a variant called Fast YOLO. Because of this speed optimization, YOLO is prone to making localization errors, but, according to the authors, avoids false positives and outperforms methods like R-CNN in natural images and artworks. The second version YOLOv2 and YOLO9000 [65] improved these concepts further. YOLO9000 can detect over 9000 object classes and YOLOv2 offers easy trade-offs concerning speed and performance, both of which it can be fine-tuned for. Additionally, the authors propose a method to train the network for object detection and classification at the same time, which allows YOLO9000 to predict detections for object classes that do not have labeled detection data. The main focus of all YOLO architectures is the real-time application.

Similarly, the Single Shot MultiBox Detector (SSD) [55] proposed by Liu et al. is a real-time optimized, one-stage object detector as well. The network separates the output space into multiple default bounding boxes. During prediction time, SSD generates scores for each default bounding box and adjusts the bounding box to better match the detected object shape.

A very interesting appraoch is the Neural Architecture Search Net [90] by Zoph et al.. It is a neural network, that is designed to create problem specific neural networks to solve a given task. They use a Recurring Neural Network (RNN) with reinforcement learning that uses Neural architecture search (NAS) to automate the design of problem specific neural networks.

Regarding network backbones, ResNet and its variants are currently the most effective and popular architectures. We explicitly tested ResNet and ResNeXt, but there are other variants and modifications that could bring their own advantages.

Huang et al. [43] designed a network to tackle the problem that, because of its depth, ResNet needs a long time to train. To this end, they created Stochastic Depth, a method that randomly drops layers during training. This method is very similar to Dropout [75] which is a method used to avoid overfitting, but instead of dropping parts of a layer, Stochastic Depth drops the whole layer. The authors show in their experiments, that combining ResNet with Stochastic Depth improves the results compared to the classic ResNet approach and decreases training time.

Another architecture by Huang et al. [42] is their Densely Connected Convolutional Neural Network (DenseNet). This network further explores the concept of interconnecting layers and connects all layers directly with each other. The input of a layer consists of the output of *all* previous layers of the network. According to the authors, this new design tackles the vanishing gradient problem and increases the chance of feature reuse, which makes the architecture more efficient.

Lastly, Veit et al. [78] propose a similar idea to Stochastic Depth, but instead of dropping layers during training, they dropped layers of the trained network. Interestingly, the experiments show that dropping layers of a trained ResNet keeps the performance of the whole network similar, but dropping layers of a trained VGG neural network results in a dramatic loss of performance. At the very least, this shows one of the interesting properties of ResNet, namely, that is has multiple independent paths. The identity paths create multiple paths through the network and the majority of these paths remains intact, when removing single layers.

B METHODOLOGICAL DETAILS

We want to give a few more insights into the methodological details of this thesis. In this chapter, we will give additional information about the fundamentals of our pipeline and how the text extraction process works.

B.1 Neural Network Architectures

We previously gave short introductions to the neural network architectures we used in our research. In this chapter, we present some of the concepts we used and considered for our pipeline in more depth. We focus on giving insights into the chosen architecture for our localization network which is a type of R-CNN.

B.1.1 R-CNNs. Girshick et al.'s original R-CNN [31] was quite slow, because it needed one forward pass through the Convolutional Neural network for each region proposal. If we use the numbers we used in this thesis, this means that we would need 512 forward passes for each image. Additionally, it consisted of three different models that each had to be trained independently. One model to generate the image features, one to predict the class and one for the regression to fit the predicted bounding box to the text object.

Because of these disadvantages Girshick et al. developed the idea further and created Fast R-CNN [30] to solve both mentioned problems. The first problem – needing multiple forward passes for a single image – was solved by introducing Region of Interest Pooling. RoI Pooling reduced the number of passes needed to generate the wanted region proposals to only one. To achieve this, the network performs one forward pass through the Convolutional Neural Network to create a feature map. This feature map is then used for all regions proposals, by selecting the feature map's output for the respective region and max-pooling those features. The second change was to combine all three models into a single neural network. To achieve this, they added two parallel paths after the Convolutional Neural Network. The first path uses a softmax layer on top of the CNN's feature map to classify the bounding boxes. The second path adds a linear regression layer to create more precise bounding box coordinates. Fast R-CNN solved the two main problem of R-CNN, but one further problem remained: Fast R-CNN still used Selective Search [76] to create the potential bounding boxes, which turned out to be the bottleneck of the network.

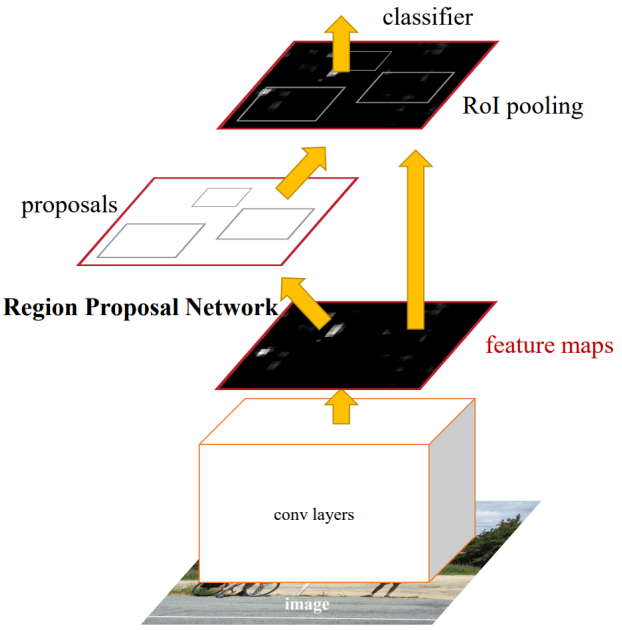


Figure 10: The Faster R-CNN Object Detection Network [67]. The Region Proposal Network is integrated between the Convolutional Layers and the Classifier. Both, classifier and RPN, use the CNN's output feature map. The original FR-CNN uses RoIPool to prepare the output of the RPN for the classifier.

Ren et al. [66] built upon this problem and created Faster R-CNN to remove this bottleneck. The authors discovered, that the region proposals mostly depended on features of the image that could already be found in the CNN's feature map. Thus, they designed the architecture for Faster R-CNN which can be seen in Figure 10. The image shows, that the authors re-use the CNN to generate the region proposals, which are then pooled using RoIPool and are the foundation for the following classifier and regression layer to improve the bounding box predictions. To create these regions of interest, Ren et al. added a Fully Convolutional Network (FCN) – called Region Proposal Network (RPN) – on top of the CNN. The RPN uses a sliding window approach that creates and rates multiple possible bounding boxes per window. These regions are then used as the input for the rest of the former Fast R-CNN.

He et al. [40] propose Mask R-CNN, which is designed for semantic segmentation. The network is based on Faster R-CNN, but adds features, that allow the network to create pixel based classifications. While these changes are not relevant to our own research, the authors added two features that we used in our own network. They replaced the original CNN with a Residual Net [39] and RoIPool with RoIAlign.

B.1.2 ResNet. Deep Residual Networks (ResNets) [39] have been developed by He et al.. They are based on solving the degradation problem. To illustrate this problem: Compare two neural networks solving the same problem. The first network is a shallow

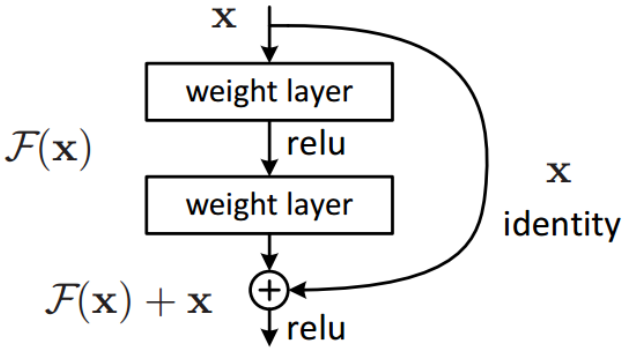


Figure 11: The main building block for ResNet. The most important change is the identity function shortcut. [39].

network, the second is its deeper counterpart created by simply adding additional layers to it. In the worst case scenario, the layers that both networks have in common should be identical, and the additional layers can simply act as the identity function and pass the input right through to the output without changing it. In theory, the added layers can only add to the approximation of the problem's solution, but in reality, this is where the degradation problem occurs. The results of the deeper network do not surpass the shallow network's approximation, but instead plateau at some point and then start to degrade and produce worse results than their shallower counterpart. The assumption of the authors was, that this problem is caused by layers that can not easily approximate the identity function if necessary. To solve this, they essentially changed the function, which the network tries to approximate.

Instead of trying to approximate a function $H(x)$ – where x is the input – ResNet approximates the function $F(x) = H(x) - x$, which can be transformed into $H(x) = F(x) + x$, where $F(x)$ represents the approximated function by the network's layers and x the identity function. This idea is implemented by the Residual Block which can be seen in Figure 11. The suggested solution is to implement identity paths around the stacked layers, that make it easy to approximate the identity function. The general structure of the ResNet can be seen in Figure 12. The blue arrows on top of the network represent the identity paths. Each ResNet block contains three layers and a identity path that goes around the block. The structure is built upon the idea of building blocks, which make it very easy to scale the network, as can be seen in the Figure. The original paper [39] contained tests with networks as deep as 1000 layers.

B.1.3 ResNeXt. Xie et al. [81] proposed a variant of ResNet called ResNeXt. The building block of ResNeXt can be seen in Figure 13. It picks up on the idea that was part of the ResNet design, which was to construct simple building blocks to allow easy scaling of the network. The authors introduce a new parameter called *cardinality*, which is the number of parallel paths inside each building block. Each path is structured in the same way and the general idea is called *split-transform-merge*. The input into the block is first *split* into the different paths, each path's first layer, beeing the same size,

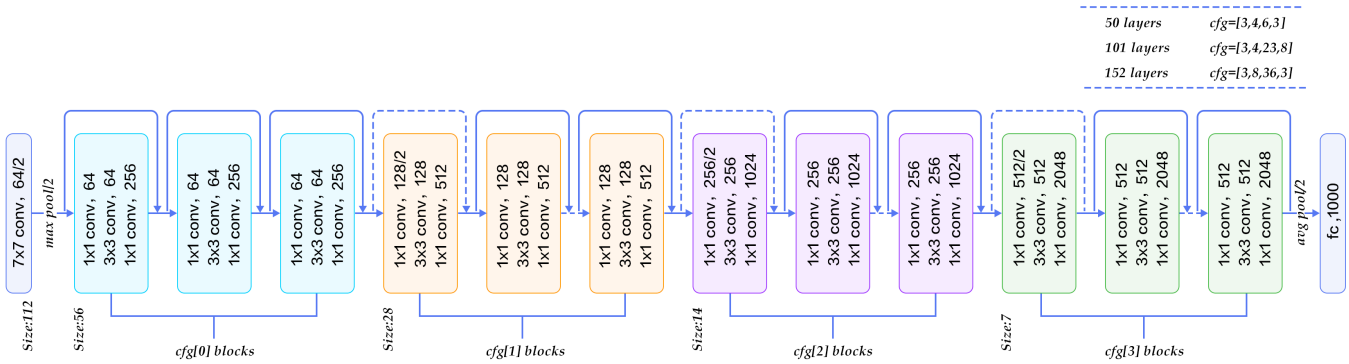


Figure 12: The basic structure of ResNet [39]. The order of the differently colored block is fixed, but the repetitions of each block are variable. There are three common configurations displayed in the top right. The numbers connote the repetitions of the respective building blocks. For example our 101 layer ResNet consists of 3 cfg[0] blocks, 4 cfg[1] blocks, 23 cfg[2] blocks and 8 cfg[3] blocks.

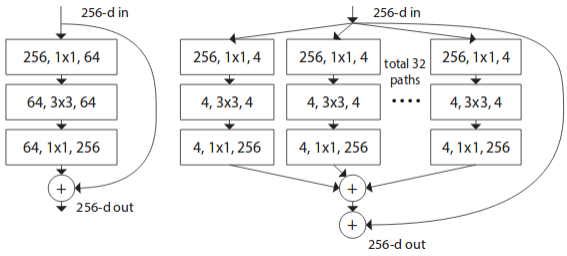


Figure 13: Comparison of the original ResNet building block (left) to the ResNeXt [81] building block (right). A layer is shown as (channels, filter size, out channels). The image shows, that the out channels – the feature maps – of ResNet are fanned out into multiple parallel pathways in ResNeXt.

but with less kernels than the corresponding ResNet path. But the total number of kernels stays the same when comparing ResNet to ResNeXt layers, only that they are divided into multiple paths in the latter case. In each path the information is then *transformed* in the same way it would be in any other neural network and after three layers all paths are *merged* into output feature maps. The original experiments show, that increasing the cardinality – which means widening the network – can improve the network's accuracy more, than increasing the network's depth.

B.1.4 RoIAlign. In the Faster R-CNN paper [66], the RPN uses RoIPool. Each Region of Interest has a different size, but the following layers need a fixed size input – whose dimensions we will call "Height x Width". Therefore all images are rescaled using RoIPool. RoIPool does this, by dividing the RoI into Height x Width sub-windows. Then applies max-pooling on each of the sub-windows to get a feature map of the desired dimensions. The problem with this process is, that dividing the large RoI feature map into smaller feature maps will cause misaligned locations, because the values have to be rounded. This alignment problem can be seen in Figure 14.

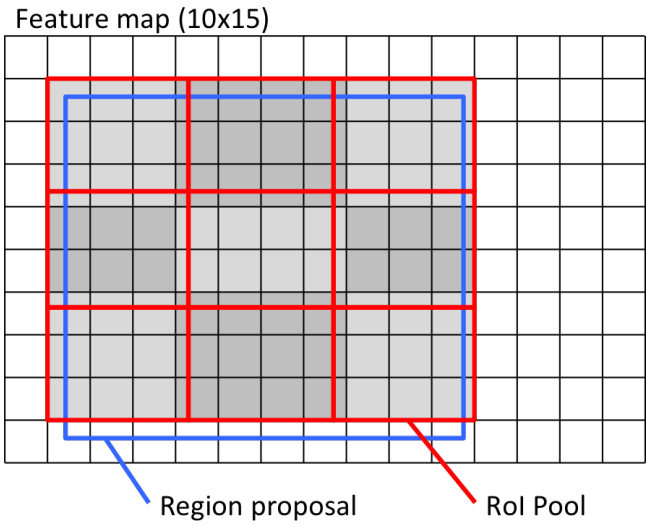


Figure 14: Visualization of RoIPool showing the misalignment problems caused by the rounding operations. [5].

RoIAlign does avoid these rounding errors by using the actual (not rounded) values to calculate the four regularly sampled location's values, which are then used to calculate the actual values of the output feature map. This can be seen in Figure 15.

B.1.5 Feature Pyramid Network (FPN). We use our RPN combined with a Feature Pyramid Network (FPN) [53]. FPN is a feature detector that works in combination with objects detectors. FPN extracts multiple feature map layers which are then used as an input for the RPN. Originally, Faster R-CNN used one feature map to create RoIs which are then – combined with the original feature map – used as input to RoIPool (or RoIAlign in our case). When using FPN, we create a feature pyramid. Each feature pyramid layer has the same content at a different scale. Depending on the size of the proposed RoI, we select the feature map layer that has the best

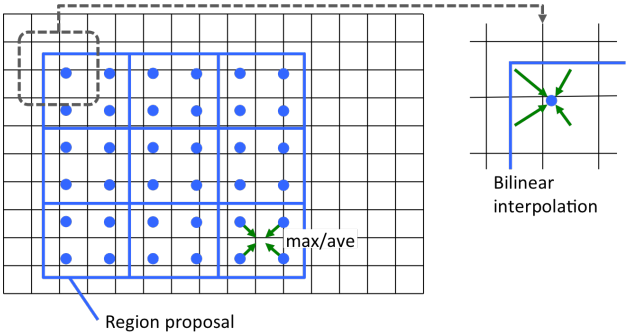


Figure 15: Visualization of RoIAlign's use of bilinear interpolation to solve the misalignment problem. [5].

scale to extract the region's features and fit it into the size-restricted input of RoIAlign. This allows to resize the content of the RoI in a way that works best for the following RoIAlign, Classifier and bounding box regression.

B.2 Relevant Datasets

During this master thesis, we considered multiple approaches and datasets for our pipeline solution. In this section, we present all relevant datasets for the different approaches we contemplated. First, we give a more in-depth introduction to the datasets we use in our final pipeline. We use COCO Text for pre-training and CHIME-R, CHIME-S, DeGruyter, EconBiz and DeTEXT for training and testing. Additionally we also experimented with pre-training our localization network on Total Text Dataset. Second, we give short introductions to the datasets we did not use in our final pipeline, but we consider relevant to different approaches for text extraction from infographics. One of the more complex solutions to the lack of training data is the generation of synthetic data. We considered two approaches to synthetic training data. The first is to superimpose text elements on empty or randomly generated graphs. This approach could be used to train the localization network on a symbol-basis – i. e. train the network to detect individual symbolds instead of general text – and may allow to create a one-step pipeline that only uses one neural network. One dataset that consists of syntheticly generated words is the Synthetic Word Dataset [6]. The most promising – but also most difficult – approach is to generate individual training data using a Generative Adversarial Network (GAN) [36]. There are several recently published papers [9, 79] using GANs to augment datasets including natural image datasets. GANs consist of two neural networks that contest each other in a zero-sum game. While one network tries to generate images that are indistinguishable from real data the second network distinguishes fake from real images. While this approach might work very well for this problem, because the training data has a relatively simple structure, it may still be difficult to implement and goes – similar to the creation of additional synthetic data – beyond the scope of this master thesis.

B.2.1 CHIME. The CHIME datasets were created by the Center for Information Mining and Extraction (CHIME), School of Computing and National University of Singapore. CHIME-R is a set

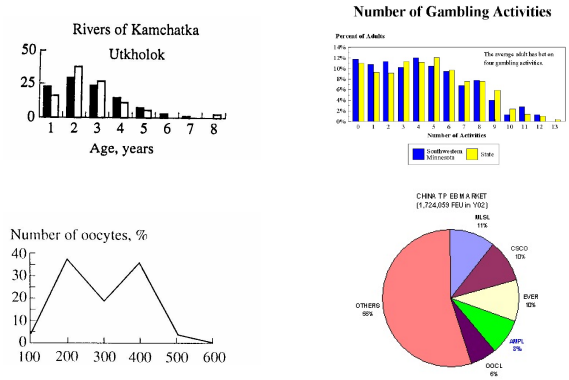


Figure 16: Four example images for CHIME-R. Most of the images in this dataset are similar to the one in the top left.

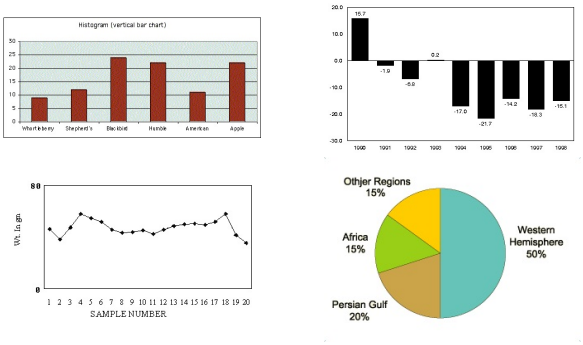


Figure 17: Four example images for CHIME-S.

of 115 real images from the internet or scanned from paper. The inforgraphics contain bar-, pie- and line-charts, as can be seen in Figure 16. The gold standard for this dataset was created by Yang Li [84]. CHIME-S is a similar dataset, containing 85 synthetically generated images. The dataset is based on CHIME-R and contains very similar infographics, which consist of bar-, pie-, and line-charts. The gold standard was created by Zhao Jiuzhou [49]. Examples for this dataset can be seen in Figure 17.

B.2.2 DeGruyter. The DeGruyter dataset is based on scientific publications by the DeGruyter publishing company. It consists of 120 infographics scanned from academic books provided by DeGruyter. Most of the images are from the chemistry domain. Figure 18 contains example images from the dataset. It is easy to see, that the images are more complex than the images contained in the CHIME datasets. They contain more text elements, longer text elements and longer words in those text elements. At the same time they incorporate real life images or other graphical elements into their infographics, which may be a combination of multiple smaller elements. For example the infographic in the top right of our examples contains a real life image of the depicted experiment and at the same time consists of four distinct infographic-elements that show different information regarding this experiment. Similarly, the

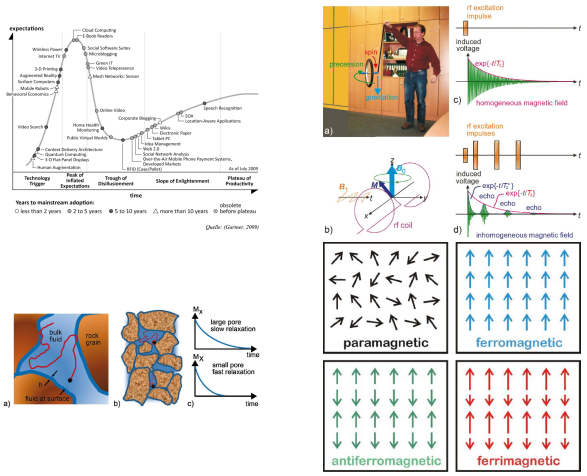


Figure 18: Four example images for DeGruyter. The dataset is very versatile, some images contain many text elements, others many graphical elements and others are combinations of different kinds of infographics.

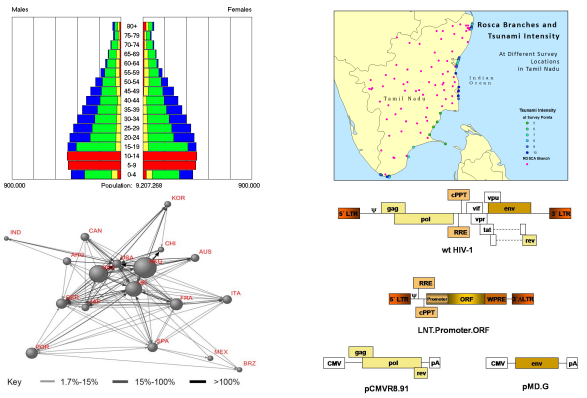


Figure 19: Four example images for EconBiz. The dataset contains complex infographics. One class of infographic, that is not in any of the other datasets, are maps, like the one on the top right.

infographic on the bottom left consists of two graphical elements and and two line-graphs.

B.2.3 EconBiz. The EconBiz dataset originated from a collection of 288, 000 scientific open access publications, which were collected from the EconBiz portal. The EconBiz dataset contains 121 infographics that were randomly selected from millions of images that were extracted from the publications. Examples for the dataset can be seen in Figure 19. The dataset contains scientific infographics containing flow-, pie-, bar-, and line-charts, histograms and maps. The images contain graphical elements which are often placed on differently colored backgrounds as can be seen on the bottom right of our examples. The gold standard was created by Böschen et al. [11].

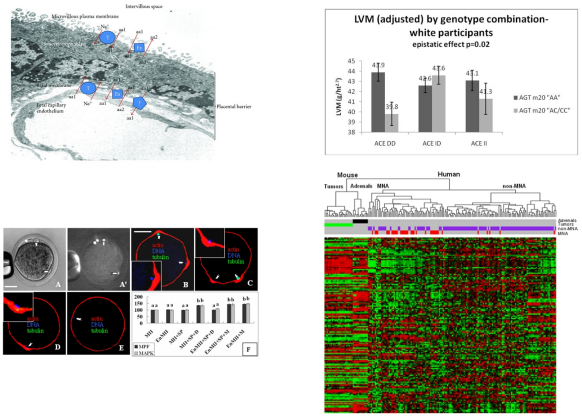


Figure 20: Four example images for DeTEXT. The dataset consists of biomedical images that contain X-ray and MRI photos and complex structures like the one on the bottom right.

B.2.4 DeTEXT. DeTEXT is a dataset that originally contained human-annotated, high quality, large-scale figures and text. It contains 288 full-text articles, 500 biomedical figures, and 9308 text regions in those figures. We were unable to locate the original dataset online and used a smaller variant that is part of the ICDAR robust reading challenge [3], which only contained 192 biomedical figures. The dataset was designed with text extraction tasks in mind. Since text is a rich source of information in figures, extracting that text could be very helpful in mining information from scholarly articles. DeTEXT represents important biomedical experimental evidence and was created to facilitate the development of automated text extraction systems. The original article [85] describes in detail how the infographics were selected and the statistics of those figures. Examples for DeTEXT can be seen in Figure 20. Some of the images contain similar infographic structures like the previous datasets. The infographic at the top right consists of simple bar-charts. But most of the images contain actual medical images – like X-Ray or MRI images – or graphical elements depicting biomedical objects, both of which can be seen in the left examples. Additionally, some of the images contain very – seemingly – noisy elements like the figure on the bottom right. These noisy elements – DNA-Sequencing visualizations in this example – might bring unique difficulties to the text localization and extraction.

B.2.5 COCO Text 1.3. COCO Text [77] is a large scale dataset for text localization and extraction in natural images created and published by Cornell University. COCO is an abbreviation for *Common Objects in Context*. It is based on the original MS-COCO [52] which contains annotations for common object classes – like human, car, bus, dog, cat, bicycle, etc. – in natural images. COCO Text uses the same images, but adds text annotations to those images. We use COCO Text 1.3 that adds 145, 859 text instance annotations to the 63, 686 images from MS-COCO. Apart from the bounding-box information the annotations contain three text attributes: machine or hand written, legible or illegible, and english or non non-english script. Because our infographics contain only

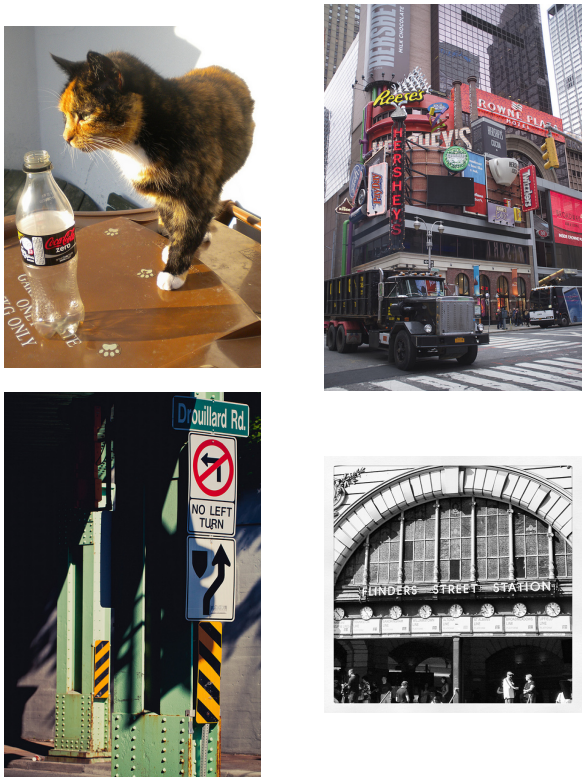


Figure 21: Four example images for COCO Text. COCO Text contains natural images with very few text elements.

Figure 22: Four Example images for Total Text Dataset. TTD is a dataset for text elements in natural images.

Figure 23: Four example images from the Synthetic Word Dataset. The dataset consists of text with different fonts, orientations and shades.

legible, english and machine written text elements, we decided to restrict our pre-training process to those text occurrences. This reduces the dataset to 13, 966 individual images and 63, 545 text annotation in those images, which results in and average of 4.55 text annotations per image. As can be seen in the example images in Figure 21, the images are very differet to our infographics. They are natural images with – relatively – few text occurrences per image. We use the dataset for our pre-training process, because there are no big datasets containing annotated infographics. The idea behind this pre-training process is train the neural network in a way that allows it to recognize certain image feature regions, like contrast changes, vertical or horizontal edges, background and foreground, and general ideas about text elements. On top of those features we can then train the network on the infographic datasets, which improves the neural network training both in speed and precision.

Recently, the new version COCO Text V2 has been released. This version contains 239, 506 annotation, but was released to late to be used in this thesis.

B.2.6 Total Text Dataset. Total Text Dataset (TTD) [25] is a dataset explicitly designed for text localization and extraction. In contrast to COCO Text the dataset is not added on to an already existing big dataset, but was built upon the idea of text in natural images. As a result, TTD contains 7.37 text elements per image and also contains higher quality annotations, i. e. there are curved annotations for curved text occurences. Apart from these features the dataset is very similar to COCO Text, as it contains similar natural images, as can be seen in Figure 22. We experimented with using TTD as a pre-training dataset for our localization network.

B.2.7 Synthetic Word Dataset. The Synthetic Word Dataset [6] contains 9 Million images covering 90, 000 English words. The words are created randomly with different fonts, slight orientation variation, bends and different back- and foreground colors. We did not use this dataset, because it lacks the infographic context, but it could be useful to teach a network to detect text elements or maybe pre-train a neural network on these images. Additionaly, the same method could be used to generate synthetic symbol-specific annotations for English words. Examples for this dataset can be seen in Figure 23.

B.2.8 Chars74k. Chars74k [27] is a dataset designed for character recognition in natural images. It contains symbols from the

Figure 24: Four example images for Chars74k. The dataset contains synthetic and real characters of different shapes, fonts, resolutions and colors.

English and Kannada[4] languages. The English subset contains 64 classes – 0-9, A-Z, a-z – in English, Latin script – excluding accents – and Hindu-Arabic numerals. It contains 7,705 characters from natural images, 3,410 hand drawn characters and 62,992 synthesized characters from different computer fonts, for a total of 72,000 character-images.

We thought about using this dataset to train a localization network, that is able to detect all 64 classes individually. This would allow us to create a single step pipeline, that is able to extract the text from infographics. One of the problems that prevented us from implementing this solution was, that we do not have a character-specific gold standard for our infographics. Because of this, we would not be able to evaluate our localization results, which would make it very difficult to fine tune the network to a state in which we can compare the extracted text to the appropriate gold standard. Examples for the images contained in the English subset can be seen in Figure 24.

[4]The script is used mostly in southern India to write Sanskrit in the state of Karnataka.

B.3 Data Augmentation

We used several relatively simple data augmentation techniques. Most of the techniques have been describes previously, but we want to go into detail for two of our methods and describe how they were implemented.

The first method we want to elaborate on is the addition of noise. The idea behind adding noise to the image is to make the neural network more resistant to changes in the image. Artifacts in images are sometimes very problematic for neural networks and very prevalent in digital images. To alleviate this problem, noise is added to the input images, which should allow the neural network to detect text elements despite artifacts in the image. We used two methods to add noise to the images: gaussian noise and salt and pepper noise.

Both types of noise are usually observed in digital images, that are not taken in optimal conditions. Gaussian noise is a type of statistical noise with a probability function of a normal distribution – i. e. gaussian distribution – which is often caused by poor illumination. Salt and pepper noise is caused by sharp and sudden signal spikes, which result in sparsely occurring black or white pixels in the picture.

The second method we want to elaborate on is the translation of the input images. The effect of this translation is, that the text elements are shifted to the bottom right. This creates images, whose text elements are in different positions and allow the neural network to learn to detect text in multiple locations, which is especially helpful when most of our text is near the edges. We used a translation matrix to translate the image. We shift the content of the image to the bottom right – 100 or 50 pixel to the right and 200 or 100 pixel to the bottom – in all cases to get additional input images with more centralized text. Because these shifts already produce images with centralized text, we decided against shifting in additional directions and using other values.

B.4 Pre-Processing

Because our initial localization and extraction network tests showed that a lot of the unreadable text was text on colored backgrounds, we decided to convert the RGB images to grayscale images. We used the findings of Kanan et al. [50], who tested the effect of using different methods to convert color-images to a grayscale color format. The result is, that the methods called *Gleam* and *Intensity* are the only methods effective in all conducted experiments. Gleam is simply the gamma corrected variant of Intensity. We decided to use Intensity, which used the very simple formula:

$$G_{Intensity} = \frac{1}{3}(R + G + B)$$

For our extraction, we used several types of pre-processing to create multiple input images from one text candidate. We added a white border of 25px, because Tesseract often failed to detect text on the border of the original image. After adding the white border, the text to extract was more centered in the image. For similar reasons we increased the shortest side of the image to 100px or 200px. The reason being, that Tesseract often failed to recognize very small text elements. Simply increasing the size – while using bi-linear interpolation improved the extraction results in some cases.

We also added several images, that we converted into binary images by using different thresholding methods to clear up the images. We used an adaptive threshold [15] that takes the lighting and spatial variations in the image. Adaptive thresholding does not use a global value to threshold the whole image. For images with very different lighting in different areas, this means that some of these areas might be totally black and others totally white, which means that information may be lost. Adaptive threshold repeatedly calculates the threshold for smaller regions of the image, which results in different thresholds for different regions in the image.

We used Otsu's Threshold [61] as our second method to calculate thresholds. The algorithm assumes, that the image contains two classes of pixels using a bi-modal histogram. Based on this assumption, Otsu's method calculates the threshold, that minimizes the intra-class variance – e.g. maximizes the inter-class variance.

For our third thresholded image, we combined Otsu's threshold with gaussian blurring. Gaussian blur is mathematically the convolution of the image using a gaussian function. In two dimensions, the gaussian formula looks like this:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The calculated values are combined into a convolution matrix, which is then used to calculate a weighted average as the new value for the central pixel. The center pixel's weight is the largest, with the other weights decreasing with distance from this center.

The results of these thresholding processes can be seen in Figure 25. Additionally, we use the inverse of each of our thresholded images, because in our experience it is often easier to detect dark text on a brighter background than the other way around.

B.5 Pipeline Procedure

Because our pipeline consists of two major parts and we pre-process the bounding boxes before extracting the text with Tesseract, it is important to give a detailed introduction about how our pipeline works, from input image to extracted text elements. Most of the work load happens during the training of our localization network and does not add processing time to the actual pipeline.

To train our localization network, we use our augmented dataset. This dataset uses 80% – ∼ 480 images – of CHIME-R, CHIME-S, DeGruyter, EconBiz and DeTEXT as training set. The other 20% – ∼ 120 images – are used for testing purposes. The data contained in the test set is converted to Grayscale images, which are then augmented with the previously mentioned methods. This results in a total of ∼ 5,760 input images used for training. We train the neural network for 250,000 iterations, which takes ∼ 44 hours.

At this point the *real* – how it would be used in a real-time application – pipeline starts. The trained neural network is used to localize text elements in new images – in our experiments the test data. Our localization network outputs a list of bounding boxes for found text elements associated with the respective images. We use these bounding boxes to create the input images for Tesseract. Using the previously explained methods we create 189 – nine rotations, three sizes, seven different thresholds – input images for Tesseract. We test one image at a time until we either reach a confidence of $> 94\%$ or all images have been processed and we use the best result. The confidence score is a feature of Tesseract and is an estimate of how likely the predicted text is actually the text contained in the input image. If our chosen confidence score has not been met, after all images have been processed, we repeat the whole process with a special "one character recognition" mode to detect single characters in our bounding boxes. We then add the best result – with the rotation of the best input – to the image-specific list for extracted text elements. This process is repeated for all found bounding boxes and at the end all results are added to an additional unified list for all tested infographics.

B.6 Gestalt Pattern Matching

Gestalt Pattern Matching (GPM) is a rarely used measure in modern publications, but is of great value in assessing the similarity of the predicted text and the gold standard instead of just calculating the minimal edit sequence. It was published in an article by Ratcliff and Metzener in 1988 [62]. One of the big advantages of GPM is that it takes the length of the words into account, while the Levensthein Distance is invariant to word length[5].

The easiest way to show how GPM works is by providing an example. Consider the gold standard of "apple" and assume our text extraction process predicted the word "a5piplre". GPM creates a list of identical substrings in both words, starting with the longest and repeating the process recursively on both sides of the found substring. The resulting list of identical substrings looks like this: $M = \{"pl", "p", "e", "a"\}$. The GPM value for this match is calculated by this formula:

$$GPM = \frac{2 \cdot \sum_{w \in M} |w|}{|W_1| + |W_2|}$$

Where W_1 and W_2 are the lengths of the gold standard and the predicted word. In our example this would result in

$$GPM(apple, a5piplre) = \frac{2 \cdot (2 + 1 + 1 + 1)}{5 + 8} = 0.76$$

, which shows that the words are similar.

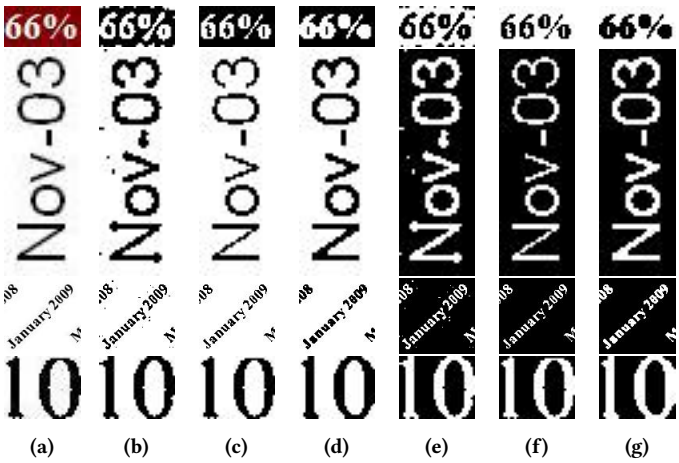
C EXTENDED EXPERIMENTS

In addition to the previously mentioned experiments, we conducted multiple experiments to optimize the configuration of our neural network, test the effect of our parameters and pre-processing steps, verify the used methods and quantify the extend of their improvements. In this chapter, we will show the results of these additional experiments and present a more in-depth look at some of the previously shown results.

C.1 Network Architecture

Our first optimization step was to determine the optimal ResNet configuration for our localization network. We experimented with ResNet [39] and the newer – supposedly improved – ResNeXt [81] with cardinality 32 and 64 and bottleneck width of 8d and 4d, respectively. While not all problems profit from deeper and more complex neural network architectures, we found, that the results of our localization task improve when using a deeper network, but not when using the more complex ResNeXt. The results can be seen in Table 10. Increasing the number of layers from 50 to 101 improved the results, especially when looking at the AP75 measure, where

[5] A LSD of 3 is a good result for a long word, but obviously bad for very short words.



(a) (b) (c) (d) (e) (f) (g)

Figure 25: Four example bounding boxes detected by our localization network with our thresholding methods for Tesseract. (a) Original bounding boxes, (b) Adaptive Threshold, (c) Otsu's Threshold, (d) Otsu's Threshold after Gaussian blurring/filtering, (e) Inverted image b, (f) Inverted image c, (g) Inverted image d

	AP	AP50	AP75
ResNet50 (800px)	53.15%	**91.79%**	54.65%
ResNet101 (700px)	**53.94%**	91.58%	**58.51%**
ResNeXt50 32x8d (750px)	51.26%	89.44%	53.14%
ResNeXt50 64x4d (750px)	51.50%	89.38%	53.66%
ResNeXt101 32x8d (600px)	49.70%	88.60%	50.41%
ResNeXt101 64x4d (600px)	50.73%	88.84%	51.05%

Table 10: Testing different neural network architectures for our pipeline. All networks used their respective maximum input image resolution on our hardware. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz. We used AP, AP50 and AP75 measures to compare the results.

	AP	AP50	AP75
ResNet50	**52.49%**	89.81%	**54.11%**
ResNet101	51.46%	**90.19%**	52.58%
ResNeXt50 32x8d	49.21%	88.11%	48.02%
ResNeXt50 64x4d	49.42%	87.86%	49.16%
ResNeXt101 32x8d	49.70%	88.60%	50.41%
ResNeXt101 64x4d	50.73%	88.84%	51.05%

Table 11: Testing the effectiveness of different neural network architectures for text detection. Input images have been fixed to be 600px wide. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz. We use AP, AP50 and AP75 measures to compare the results.

we observed an increase of ~ 4%. The values for AP50 decreased slightly – ~ 0.2% – but we rate the more difficult AP and AP75 measures higher than AP50. While the addition of layers improves the results, changing to ResNeXt did not produce the same result, even though the ResNeXt architecture is the more advanced variant. The results for ResNeXt50 and ResNeXt101 were noticeably worse than ResNet50 or ResNet101 for all measures. These experiments convinced us to use ResNet101 in our final pipeline.

Additionally, it is important to note, that we had to adjust the input image size to fit into GPU memory when using deeper networks. Due to this restriction, we had to use 800px for ResNet50, 700px for ResNet101, 750px for ResNeXt50 and 600px for ResNeXt101. Because of this limitation, we conducted a second comparison with input image size fixed to 600px. This should allow us to give a glimpse into the optimal solution without GPU memory restrictions. The results can be seen in Table 11. Interestingly, the results do not change much. ResNeXt is still producing worse results in all three measures when comparing it to ResNet, but the results of ResNet50 and ResNet101 have switches places, i. e. ResNet50 produces superior results when using AP and AP75, and using ResNet101 results in better values for AP50. The implication is, that ResNet101 profits more from increasing the resolution by 100px than ResNet50 from increasing it by 200px. This confirms our choice of ResNet101, because it does not only produce the best results with our hardware setup, but will – most likely – be superior when increasing GPU memory and allow higher resolutions.

C.2 Learning Rate

The most important hyperparameter for neural networks is still the learning rate. We use the learning rate warm up [37] used in Mask R-CNN [40]. The learning rate warm up gradually increases the learning rate to the selected value during the first 500 iterations, starting with a small learning rate and finishing with the desired learning rate. Learning rate warm up is used to avoid spikes during the initial training process, caused by difficult training images.

We contemplated using Random Search [10] to determine our optimal learning rate, but this approach does not increase results compared to testing different values. We tried different values for the learning rate and started with a common value of 0.0025. We increased the value gradually until the training process got unstable or the results stopped improving. While our initial learning rate

Learning Rate	AP	AP50	AP75
0.0025	66.66%	93.50%	59.96%
0.005	70.59%	**94.07%**	61.54%
0.0075	**74.30%**	93.83%	**62.70%**
0.01	73.60%	93.48%	62.03%

Table 12: Learning rate experiments with our augmented datasets. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz using ResNet50. We used AP, AP50 and AP75 measures to compare the results.

tests – without data augmentation – implied that a higher learning rate results in higher test precision, our final learning rate tests – with augmentation – show, that the positive effect of increasing the learning rate has its upper limit at 0.0075. This result can be seen in Figure 12: A higher learning rate increases the prediction quality up to 0.0075, when using a learning rate of 0.01 the result quality decreases. Additionally, the training process tended to fail – i. e. diverging weights or negative area predictions for the bounding boxes – with a higher learning rate. Finally, we decided to use a learning rate of 0.0075, even though some of our training attempts still failed. The improved results are worth a slight chance for instability during training. For our constant learning rate tests, we reduced the learning rate by a factor of 10 at 225, 000 and 245, 000 iterations to decrease the volatility of our results and avoid worse results caused by bad timing.

Additionally, we tried to determine the optimal learning rate using the method described in [74]. The idea is to linearly increase the learning rate during a short training period and use the maximum learning rate that increases the results before the precision starts to stagger or decrease. The minimum learning rate is the one, where the network starts to learn. We chose 0.00025 as the lower bound and – because of our previous experiments – 0.0075 as the upper bound. Our own test – see Figure 26 – do not show clear boundaries where the learning process breaks down, since our precision rises during the whole process. Additionally, we implemented Cyclic Learning Rates [74] as an alternative to the previously presented models. The original paper [74] shows, that a cyclic learning rate – in- and decreasing the learning rate continuously between a minimum and maximum value during the training process – decreases the training time. The results of these experiments can be seen in Figure 27. We were not able to reproduce the improved training results for Cyclic Learning Rates and therefore decided against using this method in our following experiments. Finally, we tested the effect of a Learning Rate Schedule which is based on the findings in [68]. The general idea is to reduce the Learning Rate at certain points during training, to fine-tune the model in later stages and disallow huge jumps in test precision. We set the learning rates at these iterations [0, 10000, 50000, 100000, 150000, 200000] to these values [0.01, 0.0075, 0.005, 0.0025, 0.001, 0.00075]. As can be seen in Figure 28 the LRS did not improve the training process or the final results.

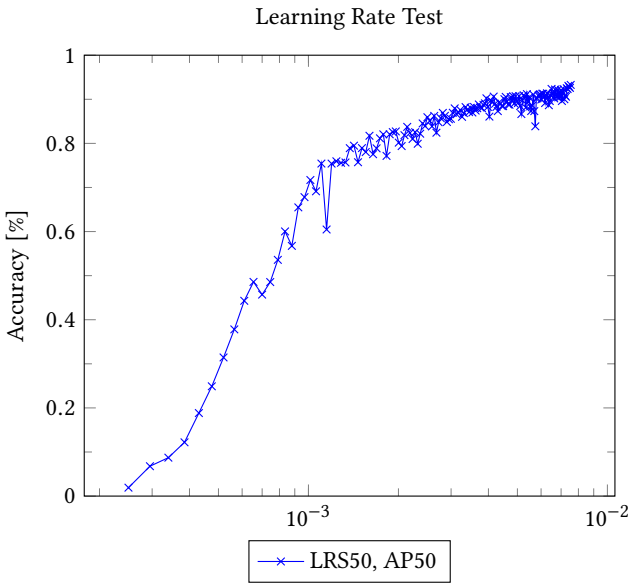


Figure 26: Learning Rate Test[74] to find the optimal learning rate for our localization network. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz using ResNet101. Accuracy for the AP50 measure.

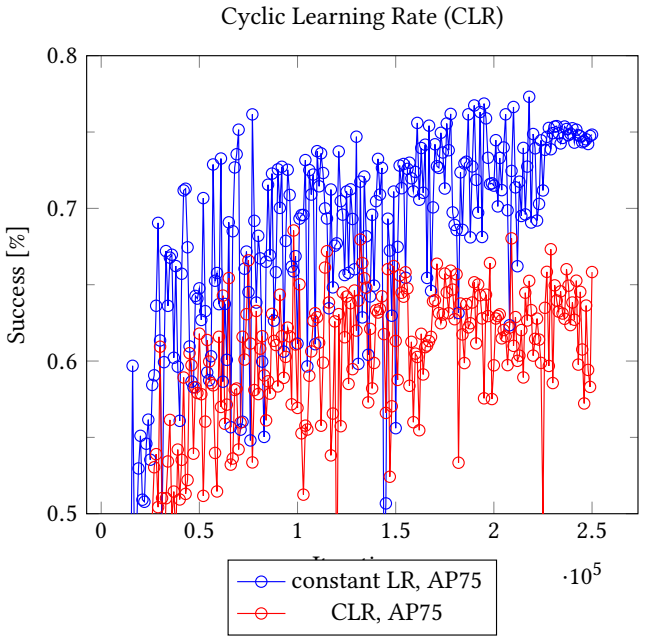


Figure 27: Comparison: Cyclic Learning Rate and constant Learning Rate. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz using ResNet101. We used the AP75 measure to compare the results.

C.3 RoIAlign

We tested the effect of RoIAlign by comparing it to its predecessor RoIPool. Because RoIAlign was originally designed to increase the precision of semantic segmentation results, it was surprising to see the improved results when using it in our experiments. The

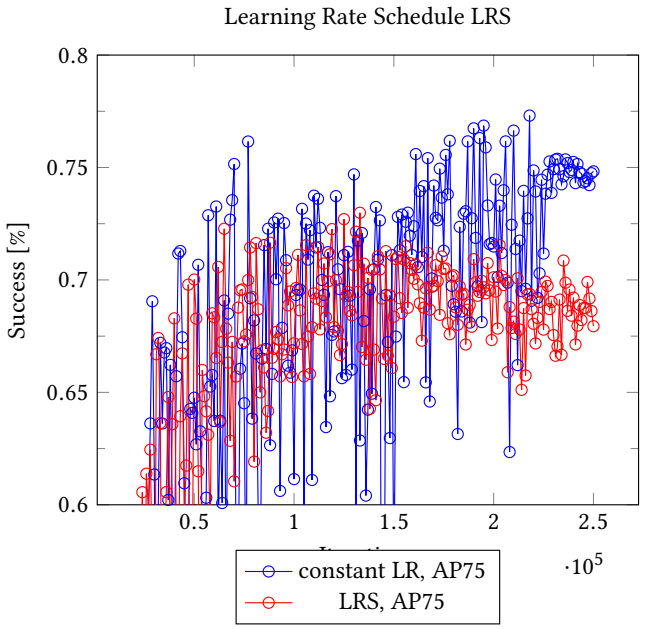


Figure 28: Comparison: Learning Rate Schedule and constant Learning Rate. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz using ResNet101. We used the AP75 measure to compare the results.

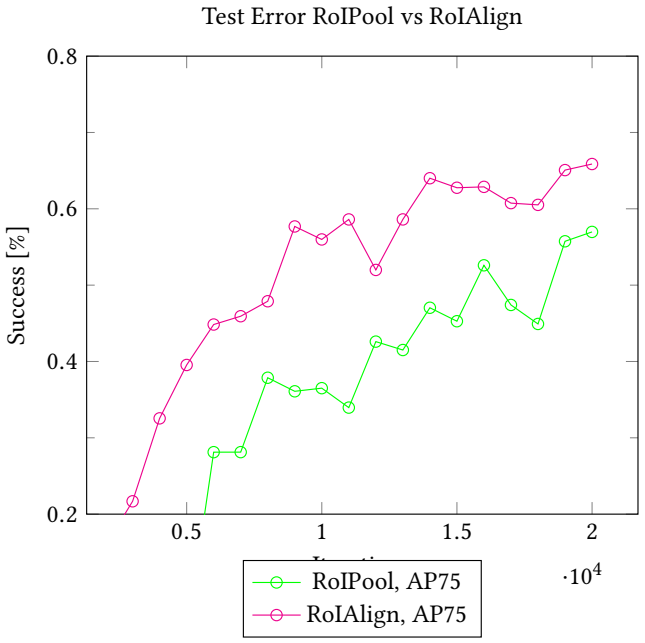


Figure 29: AP75 test error average when using RoIPool or RoIAlign. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz using ResNet101.

improvements can be seen in Figure 29. The neural network's precision increases faster when using RoIAlign compared to using RoIPool. Similarly, the final prediction precision surpasses the results when using RoIPool as well. It might seem surprising at first that this technique, developed to lessen a negative effect in semantic segmentation tasks, has a positive effect on object detection precision as well. But RoIPool generally lessens the information loss from re-scaling the feature map after the Region Proposal Network. This effect is not limited to precise semantic segmentation masks, but also applies to bounding boxes, which can be fitted more precisely when retaining the additional information from the RPN. The predicted bounding boxes can be more precise and the training process can be more flexible and incorporate even small changes into our prediction model.

C.4 RPN Proposals

One of the hyperparameters of our localization network is the number of RPN Proposals. The Region Proposal Network in our pipeline can predict different amounts of Regions of Interest, which has an obvious effect on the pipeline's results and performance. It is a classic problem: We want to generate enough RoIs to make sure, that the correct – or approximately correct – bounding boxes are part of the predictions. At the same time, every additional RoI has to be tested and verified by the classifier which takes additional time during training and testing and it adds worse RoIs to the pool of possible predictions. The values could be adjusted individually for training and testing, but we decided to keep them at the same level, because the effect on the training time was minimal. The original value for Mask R-CNN [40] was 512 RPN proposals. To make sure, that this is the correct value for our own problem, we tried to double and halve this value to 1024 and 256 RPN proposals, respectively. The results of this text can be seen in Figure 30. The experiments show, that the original value of 512 RPN Proposals is – at least – close to the optimal value. Decreasing and Increasing the amount of proposals worsens the results of our localization network. These results convinced us to keep the value of 512 RPN proposals for training and testing purposes.

C.5 Dataset Similarity

We evaluated the similarities between the five main datasets and the effect these similarities have on the experimental results. Our initial theory was, that test results mainly depend on the complexity of the chosen infographics. A complex infographic contains multiple different chart elements, mixed with real live photographic images, text of different sizes and graphical visualizations. Less complex infographics are simple black and white graphs with only one type of text and no additional graphic elements. CHIME-R and CHIME-S consist of less complex infographics, EconBiz, DeGruyter and DeTEXT of more complex infographics.

But our experiments show us different decisive similarities between the datasets. We trained our localization network on a single dataset (CHIME-R, CHIME-S, DeGruyter, EconBiz or DeTEXT) and tested the trained network on all datasets. The assumption is, that testing on a dataset will result in good results if the test-dataset is similar to the dataset we trained the network on. Initially, we did not use the pre-training on COCO-Text, but added pre-training later on. The results without pre-training can be seen in Table 13,

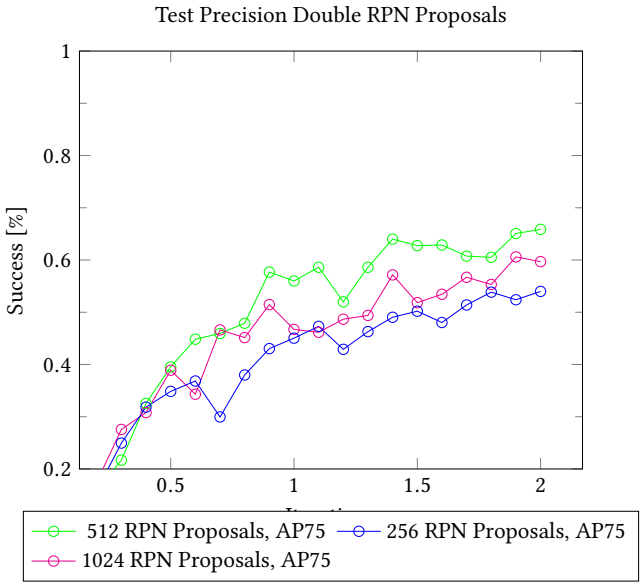


Figure 30: AP75 Test Precision average for the original values for region proposals and a version with doubled or halved RPN proposals during training. Tested on CHIME-R, CHIME-S, DeGruyter and EconBiz using ResNet101. We used the AP75 measure to compare the results.

the results with pre-training in Table 14. Pre-training the network improved most of the test results, except the very low results which dit not change much.

Concerning the similarities of the datasets, the experiments show, that CHIME-R and CHIME-S are similar to each other, but very different to EconBiz and DeGruyter. Likewise, EconBiz and DeGruyter are similar to each other and very different to CHIME-R and CHIME-S. Interestingly, DeTEXT is more similar to CHIME-R and CHIME-S than EconBiz and DeGruyter. This implicates, that the similarity between datasets can not be defined by complexity alone. To get more information, we evaluated the amount of text and the length of said text in all infographics. The results can be seen in Table 15. CHIME-R, CHIME-S and DeTEXT have a similar number of text elements per image, while the DeGruyter and EconBiz infographics contain twice the number of text elements. The relation is very similar for the number of words per infographic. Combined with the previously presented similarity-experiments, this might implicate, that one of the more important property for dataset-similarity is the amount of text elements and words per infographic. At the same time the number of characters per image, which is a lot higher in DeTEXT than CHIME-R and CHIME-S can – combined with the increased image complexity – explain, the different results for training and testing on DeTEXT.

C.6 Data Augmentation

One of the biggest problems of Deep Neural Networks is a possible lack of training data. Our datasets combined contain ~ 600 images of which we used ~ 120 – i. e. one fifth – for testing which leaves only ~ 480 images to train the network. To improve our training results, we tried to apply different kinds of data augmentation. First, we use different datasets to pre-train our network. We experimented with using the previously described modified COCO Text [77] and Total Text Dataset [25] for our pre-training procedure. We trained the localization network for 250, 000 iterations on the pre-training datasets and followed up with training on the infographic datasets for another 250, 000 iterations. The extended results for all datasets in our experiments can be seen in Table 16. There are only two cases where the variant without pre-training is superior to the pre-trained version. Both results are on CHIME-R, where the pre-training has a negative – AP75 and AP – or nearly no – AP50 – effect. In all other experiments, the effect of pre-training is positive and quite noticeable in some cases. While the effect for CHIME-S is – comparatively – minor, the effect is much more noticeable, when looking at EconBiz, DeGruyter and DeTEXT. On these three datasets, our localization network without pre-training achieved results below 60%, and even ~ 30% for EconBiz, when looking at the more difficult AP75 and AP measures. Using pre-training increases these results by ~ 10 – 36%.

When comparing pre-training on Total Text Dataset and COCO Text, both datasets have advantages on different datasets. To determine which pre-training to use in our following experiments, we decided to consider the averaged results over all test datasets, which can be seen in the "Combined" row of Table 16. In all three measures, pre-training on COCO Text provided slightly better results for our training and testing environment, which is why we chose to pre-train the our localization network on COCO Text. Next, we tested multiple methods of data augmentation. We tried three different rotations – 90°, 180°and 270°. All rotations increased the AP50 measure for our detection results. While rotating the image by 90 and 270 degrees results in images that partly contain text with our original alignment the rotation by 180 degrees produces images that contain text that is upside down and/or vertically mirrored. Interestingly the improvements for our AP50 measure introduced by the 180 degree rotations are greater than the others. All three variants had no measurable effect on the AP75 measure. This implies, that using rotated images does increase the detection quantity of text instances, but does not increase the detection quality.

We also tried to use differently scaled input-images. As per our GPU memory restrictions we could not upscale the images.

Our first test uses the original image size and the halved image sizes – i. e. 800px and 400px maximum side-length. In each training step, the input scale is chosen at random. This approach increases both detection quantity and quality greatly. Adding another size – 200px – to this process does not improve the results and seems to have a negative effect on the detection quality.

A very common method of data augmentation for object detection tasks is flipping the picture horizontally or vertically. Initially, it seemed to be unfitting for text localization, because the resulting images contains mostly text in unnatural alignment – for example upside down – and may therefore create worse results. Interestingly, this is not the case and while the horizontal flip increased both AP50 and AP75, the vertical flip still increased at least the AP50 results. This – combined with the previous results, that rotating the image by 180 degrees increases the results as well – implies that a readable alignment does not have a big impact on detection results.

Trained on	Results AP75					
	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT	Combined
CHIME-R	79.12%	40.29%	12.10%	2.54%	22.94%	31.40%
CHIME-S	64.60%	81.21%	3.44%	0.63%	19.70%	33.92%
DeGruyter	10.67%	0.42%	70.90%	48.73%	17.04%	29.55%
EconBiz	1.61%	00.31%	51.84%	49.82%	14.38%	23.59%
DeTEXT	37.42%	30.86%	12.90%	3.45%	40.71%	25.07%

Table 13: Extended results of training on a single dataset for $100,000$ iterations on ResNet101. This test shows similarities between the different datasets. The experiments were conducted without pre-training. We used the AP75 measure for these experiments.

Trained on	Results AP75					
	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT	Combined
CHIME-R	92.77%	61.19%	12.20%	2.62%	33.83%	40.52%
CHIME-S	81.23%	88.72%	7.27%	1.56%	32.20%	42.20%
DeGruyter	16.68%	1.05%	66.78%	56.31%	11.32%	30.43%
EconBiz	1.11%	0.96%	57.94%	98.89%	8.77%	33.53%
DeTEXT	40.10%	33.56%	18.11%	5.00%	67.15%	32.78%
	Results AP50					
	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT	Combined
CHIME-R	95.02%	94.99%	62.53%	46.29%	51.51%	70.07%
CHIME-S	91.17%	99.00%	66.09%	54.77%	59.33%	74.07%
DeGruyter	70.15%	59.20%	96.63%	92.70%	66.77%	77.09%
EconBiz	59.70%	44.32%	87.46%	98.89%	66.00%	71.28%
DeTEXT	47.60%	53.54%	6741%	72.04%	85.71%	65.26%
	Results AP					
	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT	Combined
CHIME-R	83.88%	58.20%	25.44%	12.46%	31.37%	42.27%
CHIME-S	66.36%	73.65%	23.24%	14.21%	33.71%	42.23%
DeGruyter	29.10%	14.98%	59.18%	52.96%	26.62%	36.57%
EconBiz	17.14%	10.64%	52.04%	89.14%	24.21%	38.64%
DeTEXT	34.55%	32.36%	30.66%	23.41%	57.33%	35.66%

Table 14: Extended results of training on a single dataset for $100,000$ iterations – $250,000$ for "All Combined" – on ResNet101. This test shows similarities between the different datasets. The experiments were conducted with pre-training on COCO Text. We used the AP, AP50 and AP75 measures for these experiments.

	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT
Number of Images	115	85	120	121	192
Text elements	14	12	24	25	14
Words	18	18	34	35	20
Characters	76	69	149	151	120

Table 15: Comparison of total number of images in the datasets and text elements, words and characters per image in each dataset. The numbers show similarities and differences between the datasets.

A method to increase robustness in object detection is to add noise to the input-images. We tried two variants which added different noise effects to the images. First we added Gaussian noise as we explained in Section B.4. We used a variance of 0.1 and a mean of 0.

For salt and pepper noise we added the same amounts of salt – white pixels – and pepper – black pixels – noise. We changed $\sim 0.4\%$ of the pixels to white noise, and the same amount to black noise.

Finally we tried translating the images. We used the translation matrix [[1,0,x],[0,1,y]] with $x_1 = 100, y_1 = 50, x_2 = 200, y_2 = 100$. Using a translation matrix works by multiplying the original image with the matrix and can be interpreted as adding a constant vector to every point of the image – or as shifting the image in the respective direction. Our translation matrix shifts the image to the right by 100 or 200 pixels and to the bottom by 50 or 100 pixels.

All results – trained on ResNet50 to avoid long training times – can be seen in Table 17. Interestingly, our data augmentation experiments show, that the added infographics do not need to contain only *real* text elements. When rotating the images by 180°, the originally horizontal text will be up-side down and the vertical

	no pre-training	pre-trained on TTD	pre-trained on COCO Text
AP50			
CHIME-R	96.51%	**96.73%**	96.61%
CHIME-S	94.41%	97.56%	**97.62%**
DeGruyter	90.14%	95.40%	**96.77%**
EconBiz	83.40%	88.67%	**91.55%**
DeTEXT	92.27%	**94.10%**	93.49%
Combined	91.35%	94.49%	**95.21%**
AP75			
CHIME-R	**92.17%**	86.75%	88.41%
CHIME-S	86.37%	86.90%	**89.76%**
DeGruyter	57.68%	**67.51%**	66.60%
EconBiz	30.50%	**49.82%**	48.71%
DeTEXT	50.74%	86.56%	**88.16%**
Combined	63.49%	75.51%	**76.33%**
AP			
CHIME-R	**79.92%**	77.28%	77.25%
CHIME-S	67.38%	67.91%	**70.12%**
DeGruyter	53.67%	**58.46%**	58.33%
EconBiz	39.48%	**49.51%**	49.26%
DeTEXT	51.43%	**75.51%**	74.95%
Combined	58.37%	65.73%	**65.98%**

Table 16: Extended comparison of our pre-training experiments. We used ResNet101 and our augmented datasets. We pre-trained on COCO Text or TTD for $250,000$ iterations and trained on the infographic datasets for another $250,000$ iterations afterwards. We used AP, AP50 and AP75 measures to compare the results.

	AP	AP75	AP50
no augmentation	52.90%	53.02%	90.34%
Rotation 90°	-0.42	+0.04	+1.45
Rotation 180°	+0.03	+0.8	+2.21
Rotation 270°	-0.03	+0.24	+1.66
800/400 scaling	+2.88	+8.65	+1.98
800/600/400 scaling	+5.29	+9.85	+2.77
Vertical Flip	-0.21	-0.44	+1.1
Horizontal Flip	+0.9	+2.77	+1.39
Translation	+1.96	+4.54	+1.2
Gaussian Noise	+0.84	+3.87	+0.05
Salt and Pepper Noise	+0.35	+3.23	+0.07
Combined	+7.91	+14.55	+1.54

Table 17: Tests to determine the effects of different augmentation methods. The experiments were conducted on ResNet50. All networks have been trained for $100,000$ iterations and only one augmentation has been added per experiment. We used the AP, AP50 and AP75 measure to compare our experiments. We signify the change in percentage points for each augmentation.

text will be oriented in an *unnatural* way[6]. Similarly, when flipping the image horizontal text will be upside-down (when flipping horizontally) or mirrored (when flipping vertically). Vertical text will – again – be written from top to bottom (when flipping vertically) or be mirrored (when flipping horizontally). Even though the text in the created images is not real text, these methods are just as effective as other approaches, that create images that contain mostly correct text. This leaves us with the question, why these methods still have a positive effect on our text localization task. We did not conduct additional experiments to answer these questions, but it is likely, that in reality the neural network does not learn to recognize *text* – i. e. letters, words, numbers, etc. – but learns to identify ẗext-likeg̈raphical elements in the infographics. And while the incorrectly aligned text is not natural text, it still has most of the graphical features – i. e. it looks like text – of natural text. This is the most likely reason why these augmentation methods work well with our problem.

[6]English *natural* text is written from left to right, when it is horizontal and bottom to top, when it is vertical. While it is not impossible to find text that is written from top to bottom, it is much rarer and feels wrong in most cases.

C.7 Generalization

We used five datasets for training and testing purposes in our experiments: CHIME-R, CHIME-S, DeGruyter, EconBiz and DeTEXT. To evaluate the generalization capabilities of our localization network, we conducted five experiments. In each experiment, we chose four datasets to train our localization network on and used the remaining dataset – which was not used for training – for testing purposes. These experiments show how well our localization network works with new data – i. e. data from datasets or domains that have not been explicitly used to train the network. Additionally, we repeated the experiments with and without pre-training the neural network on COCO Text. This allows us to evaluate the effect

Not trained on	AP50	AP75	AP
CHIME-R	80.45%	45.19%	45.82%
CHIME-S	87.73%	30.59%	41.12%
DeGruyter	86.63%	35.88%	43.06%
EconBiz	84.61%	15.88%	34.03%
DeTEXT	70.32%	23.41%	33.21%

Table 18: Extended Generalization Experiments: Training on four of the datasets using ResNet101 and augmented datasets for $200,000$ iterations without pre-training on COCO-Text. We show the test results on the dataset which was not used to train the network. We used AP, AP50 and AP75 measures to compare the results and show the results of testing on the fifth dataset.

of our pre-training on the generalization capabilities of the neural network.

The results without pre-training can be seen in Table 18. The results show that – when using AP50 as a measure for generalizability – our localization network still detects the majority of text elements in the infographics. While the detection rate has declined by $\sim 10\%$, the results show, that the network is still effective at detecting text. The results for the experiments where we pre-trained our network on COCO-Text can be seen in Table 19. To improve the visualization of the changes caused by pre-training, we combined the relevant results in Table 20.

Interestingly, pre-training only improves the results of some measures and only when testing on some datasets. CHIME-R improves in all three measures, while testing on DeTEXT produces worse results in all measures, when using pre-training. The other datasets are mixed, but the increases outweigh the smaller negative effects. This leaves DeTEXT as the most interesting case. It is difficult to ascertain the reasons why tests on DeTEXT produce worse results when using pre-training on COCO Text. One of the main differences between DeTEXT and the other four datasets is, that DeTEXT contains many natural images. There are some photographs in DeGruyter, but DeTEXT has a high rate of infographics that contain x-ray, MRI or similar images. The pre-training on the natural images of COCO Text might interfere with the capabilities of our localization to detect text elements in DeTEXT.

C.8 Tesseract

While testing our pipeline, we discovered, that most of the text elements that could not be extracted correctly or at all, were bounding boxes that contained single characters or signs. Tesseract was originally designed to convert written text-pages into digital documents in its entirety. As a result, it is not designed to recognize single characters. But the developers added additional modes called PSM (Pagesegmode) to extract text from images where some information – like text type – is already known or where additional information – like orientation – should be extracted as well.[7] When the tests of our created input images, do not result in a confidence $> 96\%$, we repeat the experiments with Pagesegmode 10, which treats the image as being the image of a single character. A comparison can be seen in Table 21. Adding the second processing run that uses PSM 10, improves our prediction results.

[7]We were not able to use the text orientation detection mode in our experiments, because it is limited to text elements, that contain more than 50 characters.

C.9 5-fold Cross Validation

To ensure that our experimental results are not solely dependent on the specific test data that we used – i. e. the pipeline works well on the chosen test set, but not on different data – we performed cross validation over five folds. As explained in Section 4.4, we split the datasets into five evenly sized parts and used one part for testing and four parts to train our localization network. We repeat this by using each fold for testing once and average our results over the five runs. The results of our 5-fold cross validation experiments can be seen in Table 22. The most interesting part of this table is the standard deviation in the "Average (SD)" column. All standard deviation (SD) values are very small, the biggest being for the AP75 measure, where the standard deviation is only 3% or 2.36 percentage points and the LSG measure, which is expected to be volatile. All other SD values are much smaller and show, that our pipeline works well independently of the chosen test dataset.

D FUTURE WORK

The positive result of our research leads to many possibilities of improving the results even further and questions that may need (extended) answers.

The biggest time consumer of our pipeline is, that we create several input images for Tesseract which have to be processed individually. We create 189 images which we use for extraction. We confirmed that all modifications may produce better results, but did no quantitative experiments to determine the relative effect of each method. Removing methods that only improve the results very slightly might be a great option to decrease processing time. We did not perform these experiments, because they are very time intensive. It is not only necessary to determine which method found the best result, but also make sure, that the same result would not be achieved by any other method in the process. At the same time, we use a fixed order when using the input images, which is obviously not optimal. It would be much better to use an adaptive system, that reacts depending on the results of the previously tested images. If a previous layer with a certain rotation already found a good – but not good enough – result, it is practical to test other images with the same, or a similar rotation. If a certain rotation resulted in very bad results, it may be helpful to change the rotation before changing thresholding or resizing methods. There are many more ways – like assuming, that images that are wider than high are oriented horizontal and images that are higher than wide are vertical or applying PSM 10 at earlier stages – that *might* improve the training time significantly. Extensive experiments would need to be conducted to determine the effectiveness of each of these possible steps. Similarly, there might be many more modification methods – i. e. different thresholds, other ways of pre-processing – that help extracting the text correctly.

We tested the generalization capabilities of our pipeline by using one of our datasets as new data – i. e. unknown to the neural network – for our experiments. The results were positive, but not overwhelmingly. While these experiments are fair, because there is

Not trained on	Results AP75					
	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT	Combined
CHIME-R	45.94%	68.76%	70.49%	97.60%	54.42%	67.44%
CHIME-S	95.09%	38.14%	58.76%	92.96%	56.72%	69.61%
DeGruyter	96.51%	78.34%	46.31%	94.28%	66.42%	76.37%
EconBiz	94.54%	78.82%	69.57%	22.10%	71.73%	67.35%
DeTEXT	96.42%	70.88%	69.87%	97.29%	20.45%	70.98%
	Results AP50					
	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT	Combined
CHIME-R	86.98%	89.40%	94.92%	98.87%	88.96%	91.83%
CHIME-S	96.31%	82.09%	94.64%	98.89%	87.74%	91.93%
DeGruyter	96.51%	96.79%	84.76%	98.85%	83.92%	92.16%
EconBiz	95.56%	96.71%	91.59%	82.34%	88.75%	90.99%
DeTEXT	98.92%	91.34%	93.66%	98.91%	61.45%	88.86%
	Results AP					
	CHIME-R	CHIME-S	DeGruyter	EconBiz	DeTEXT	Combined
CHIME-R	47.55%	59.08%	59.47%	80.41%	52.44%	59.79%
CHIME-S	81.12%	38.14%	58.76%	73.44%	56.72%	61.64%
DeGruyter	83.56%	65.92%	46.71%	73.36%	56.13%	65.13%
EconBiz	84.28%	68.88%	58.14%	34.59%	58.68%	60.91%
DeTEXT	83.81%	59.89%	58.75%	78.29%	29.31%	62.01%

Table 19: Extended Generalization Experiments: Training on four of the datasets using ResNet101 and augmented datasets for $200,000$ iterations with pre-training on COCO-Text. We show the test results on the dataset which was not used to train the network. We used AP, AP50 and AP75 measures to compare the results.

Pre-Training changes	AP50	AP75	AP
CHIME-R	+6.53	+0.75	+1.73
CHIME-S	-5.64	+7.55	-2.98
DeGruyter	-1.87	+10.43	+3.65
EconBiz	-2.27	+6.22	+0.56
DeTEXT	-8.87	-2.96	-3.90

Table 20: Comparison of the generalization experiments with and without pre-training on COCO text. We used AP, AP50 and AP75 measures to compare the results. The changes compare the pre-trained version to the variant without pre-training.

no difference between this method and using an new dataset, we think that there are many more possible types of infographics that might need to be tested. There are a lot of different scientific areas, all of which might use many different kinds of infographics. At the same time, most of our images – with a few exceptions in CHIME-R and CHIME-S – were high resolution images with limited amounts of artifacting. It would be interesting to test our network on different kinds and different image quality infographics to determine the real application possibilities of our pipeline. Another approach for getting further insights into the generalization question is to take another look at DeTEXT. The generalization capabilities of our localization network are inferior on DeTEXT when using pre-training on COCO Text compared to no pre-training. It would be very interesting to try to find reasons for this decline in prediction precision and to solve these problem or use this example to explore the more fundamental problems of pre-training or generalization.

It would be very interesting to compare our chosen neural network to other object detection networks like the YOLO variants [64, 65], SSD [55] or R-FCN [26]. Especially the one-shot based network YOLO and SSD are very promising for real-time applications. If the results are comparable to our results, this would be a big step in the direction of being able to use the pipeline in an interactive system.

At the earlier planning stages of the thesis, we thought about using synthetic data creation methods to generate additional training data. Because of time restraints we decided against implementing and testing those methods. But the convincing results of the augmentation methods that we used may translate well to more complex techniques. We looked into creating very simple synthetic datasets by simply adding random text – from some kind of English scientific library –, in the most common fonts at random locations in images containing empty graphs. This is probably the easiest way to generate additional training data. Similarly, it is probably not much more difficult to create the underlying graphs randomly as well. But these methods are restricted and they will only generate graph-based infographics, creating different kinds of infographics is much more complicated. A very interesting possible solution to this problem is, to use Generative Adversarial Networks (GANs) [36] which are able to synthetically create complex training images. Implementing and training GANs might be quite complex, but the recent results [79] using this technology are very promising.

We previously mentioned the approach to detect single symbols instead of text in the infographics. This approach could simplify

	Precision	Recall	F1 (SD)	LSD_avg (SD)	LSG (SD)
BS-4OS-O pipeline [11]	0.67	0.63	0.67 (0.22)	4.71 (**4.66**)	95.49 (94.80)
Our pipline without PSM	0.83	0.83	0.86 (0.16)	3.73 (5.04)	45.79 (42.54)
Changes	+0.16	+0.20	+0.19 (-0.06)	-0.98 (+0.38)	-49.70 (-52.26)
Our pipline with PSM 10	**0.86**	**0.83**	**0.87 (0.11)**	**3.44** (5.15)	**39.11 (40.55)**
Changes	**+0.19**	**+0.20**	**+0.20 (-0.11)**	**-1.26** (+0.49)	**-56.38 (-52.27)**

Table 21: Comparison of the best pipeline from Böschen et al.'s survey [11] – BS-4OS-O – and our pipeline. We use all measures that both papers have in common to compare the experiments. Our results are the average of our final 5-fold cross validation. We show our results with added one-character-recognition mode – PSM 10 – in Tesseract and without adding the mode. The changes show the difference difference between our pipelines and the BS-40S-O pipeline.

Fold	0	1	2	3	4	Average (SD)
AP50	94.43%	95.44%	94.01%	93.84%	95.87%	94.71% (0.89)
AP75	81.05%	76.93%	78.77%	75.33%	80.33%	78.48% (2.36)
AP	67.92%	66.06%	67.41%	66.72%	67.73%	67.16% (0.76)
Precision	0.88	0.86	0.87	0.82	0.91	0.86 (0.032)
Recall	0.87	0.85	0.83	0.78	0.85	0.83 (0.034)
F1 (SD)	0.90 (0.08)	0.89 (0.11)	0.87 (0.13)	0.83 (0.18)	0.89 (0.13)	0.87 (0.0279)
LSD_avg (SD)	3.55 (4.32)	3.15 (3.27)	4.23 (6.59)	3.24 (3.77)	3.04 (4.17)	3.44 (0.47)
LSG (SD)	37.38 (39.71)	46.38 (56.98)	32.0 (29.95)	43.94 (48.26)	35.85 (33.86)	39.11 (5.92)
GPM_avg	0.8359	0.8123	0.8811	0.8287	0.8690	0.8454 (0.0287)

Table 22: Extended 5-Fold cross validation results. Trained on a ResNet101 with pre-trained on COCO Text, data augmenation and optimal hyperparameters. Trained and tested on CHIME-R, CHIME-S, DeGruyter, EconBiz and DeTEXT. We used all chosen measures to compare the results.

the pipeline to contain only a single neural network that detects symbols, which are then clustered into text elements in a post-processing step. For this approach to work it is necessary to create a symbol based gold standard, which is – as can be expected – very work intensive. One possibility to create such a gold standard would be to use crowd-sourced annotations. There are several payed services – like Amazon's Mechanical Turk[8] or Clickworker[9] –, which allow to subcontract small tasks to a crowd of people. These services would allow to create the needed gold standard and train the network on a symbol basis.

We used COCO Text 1.3 to pre-train our localization network. Recently an updated version – COCO Text 2.0 – has been released, which contains 239, 506 text annotations compared to 145, 859 in 1.3. Additionally, this new version contains mask annotations for each word. This increase in annotated text, combined with the improved mask annotations might improve the results even further.

Lastly, it might be helpful to add pre- and post-processing steps to our pipeline. We used some pre-processing, but there are many more options to pre-process the data in front of the localization and/or the extraction pipeline which we did not deem necessary. At the same time, we did not use any kind of post-processing, because it only had very little effect in the paper of Böschen et al. [11] to which we wanted to compare our results – and we expected it would be the same for our pipeline.

[8]https://www.mturk.com/
[9]https://www.clickworker.com/

E IMPLEMENTATION

In this section, we introduce the main software packages, that we use for our pipeline. We use Facebook AI Research's Detectron for our localization network and Google's Tesseract 4.0 to extract text from the found bounding boxes.

E.1 Detectron

Detectron [32] is a software system created by Facebook's AI Research team. The system is used for object detection tasks and implements different machine learning systems. The system includes multiple object detection algorithms – Mask R-CNN [40], RetinaNet [54], Faster R-CNN and RPN [66], Fast R-CNN [30], R-FCN [26] – with different backbone network architectures – ResNet [39], ResNeXt [81], Feature Pyramid Networks [53], VGG16 [73].

We did not need to change many things to work with Detectron. We mainly added methods to make stringing together multiple training processes easier, changed processes to extract results easier and added Cyclic Learning Rate training to the configuration process of Detectron. We used the linear scaling rule [37] to adapt the parameters to our hardware – most of the parameters were designed to work on multiple GPUs, but we only used one. Detectron uses stochastic gradient decent for optimization purposes during training.

Because there are no official versions for Detectron and the system is in constant development, we can only state that we used the version that we downloaded in April 2018.

E.2 COCO Annotation Format

We decided to make the obvious decision to use a uniform standard for annotation formatting, instead of allowing multiple formats to be read by our pipeline. This allows us to use an easy and publicly available standard which has to be met for the data to be usable. We chose to use the COCO Annotation/Data Format [1], which is flexible enough to allow the network to be adapted further. Because we chose this standard, we had to convert all our datasets – interestingly, COCO text did not follow the standard – to the COCO Annotation Format. The COCO Annotation Format saves all image and annotation data inside a single file and uses the JSON format.

The COCO Data Format saves the images and annotations in a single file. It allows to add information like licences to the annotation file. The basic structure of the file looks like this:

```
{"info": [{"name": str, "licence": str,
"date_created": date}, "version": float,
"description": str] ,"images": [
{"id": int, "width": int, "height": int,
"file_name": str},...],"annotations": [
{"id": int, "image_id": int, category_id": int,
"segmentation": [polygon], "area": float,
"bbox": [x,y,width, height], "utf8_string": str},... ]
,"categories": [
{"id": int, "name": str, "supercategory": str}
,...]}
```

CHIME-R, CHIME-S, DeGruyter and DeTEXT's annotations used the same formatting. The annotations were saved on a "per image" basis inside a TSV (tab-separated values) file, where each annotation corresponds to a text line which contains the following information:

- X-coordinate of the center of the bounding box in pixel
- Y-coordinate of the center of the bounding box in pixel
- Width of the bounding box in pixel
- Height of the bounding box in pixel
- Rotation angle around its center in degree
- Text inside the bounding box

We used this data and converted it into COCO Annotation Format, but had to make one change, because COCO Annotation Format does not include the rotation of the bounding box. We increased the bounding box size and corrected the coordinates, to enclose the text in a rectangular box without the rotation. This is the bounding box we used for our training, testing and extraction purposes. For our evaluation process, we used both bounding box sets. For our AP measures, we compared the predicted bounding boxes to the bounding boxes without the rotation attribute – i. e. the bounding boxes with increased size. To allow a fair comparison to Böschen et al.'s survey [11], we used the original bounding boxes and their rotations to evaluate our results with the precision, recall, and F1 measures.

We repeated a very similar process for DeTEXT which has one annotation file per image, which simply contains one line per annotation, containing comma separated values of the four coordinates of the bounding box's corners and the string.

COCO Text had a slightly different format. All annotations were in one file, looking similar to the COCO Data Format, but there were several differences. For example the image format looked like this:

```
{"imgs": {"img_id": {"width": int, "height": int,
"file_name": str, "set": train/test, "id": int}, ...}
```

The brackets are different and the id of the image is separate from all other information. Total Text Dataset saves the annotations in one file per image and the files contain lines for each annotation. Where one line has the following structure:

```
x: [[int int int int int int]],
y: [[ int  int int int  int  int]],
ornt: [string], transcriptions: [str]
```

"ornt" is the abbreviation for orientation with these options: c=curve; h=horizontal; m=multi-oriented; #=dont care.

E.3 Tesseract 4.0

Tesseract [7] is a Optical Character Recognition (OCR) software. Since 2006 it has been developed by Google. We used the newest version which is Tesseract 4.0. The earlier versions used character patterns to extract text from images, version 4.0 added a Long short-term memory (LSTM) based approach to the program. LSTM neural networks contain LSTM memory units which are able to *remember* values over an arbitrary time interval. This allows the LSTM to process data sequentially and keep hidden states through time, which is not possible for other neural networks. This makes LSTM especially important for all problems that contain a time or sequence dependency. This is obviously the case for text elements, as the order of text elements is important.

Apart from the improvements achieved by Tesseract 4.0, we also decided to use this version – and not other available OCR engines –, because this neural network approach fits into our neural network pipeline design.

REFERENCES

[1] [n. d.]. COCO Data Format. http://cocodataset.org. Visited on 2018-10-18.

[2] [n. d.]. CS231n Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/convolutional-networks/. Visited on 2018-04-27.

[3] [n. d.]. ICDAR 2017 Robust Reading Competition. http://rrc.cvc.uab.es/. Visited on 2018-07-12.

[4] [n. d.]. Python 3.4.9 documentation. https://docs.python.org/3.4/library/difflib.html. Visited on 2018-10-05.

[5] [n. d.]. RoIAlign visualization on Qiita.com. https://qiita.com/yu4u/items/5cbe9db166a5d72f9eb8. Visited on 2018-06-26.

[6] [n. d.]. Synthetic Word Dataset. http://www.robots.ox.ac.uk/~vgg/data/text/. Visited on 2018-03-29.

[7] [n. d.]. Tesseract 4.0 with LSTM. https://github.com/tesseract-ocr/tesseract. Visited on 2018-07-16.

[8] Karim Abouelmehdi, Abderrahim Beni-Hessane, and Hayat Khaloufi. 2018. Big healthcare data: preserving security and privacy. *Journal of Big Data* 5, 1 (2018), 1.

[9] Antreas Antoniou, Amos Storkey, and Harrison Edwards. 2017. Data Augmentation Generative Adversarial Networks. *arXiv preprint arXiv:1711.04340* (2017).

[10] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[11] Falk Böschen, Tilman Beck, and Ansgar Scherp. 2018. Survey and empirical comparison of different approaches for text extraction from scholarly figures. *Multimedia Tools and Applications* (2018), 1–31.

[12] Falk Böschen and Ansgar Scherp. 2015. Formalization and Preliminary Evaluation of a Pipeline for Text Extraction From Infographics.. In *LWA*. 20–31.

[13] Falk Böschen and Ansgar Scherp. 2015. Multi-oriented text extraction from information graphics. In *Proceedings of the 2015 ACM Symposium on Document Engineering*. ACM, 35–38.

[14] Falk Böschen and Ansgar Scherp. 2017. A comparison of approaches for automated text extraction from scholarly figures. In *International Conference on Multimedia Modeling*. Springer, 15–27.

[15] Derek Bradley and Gerhard Roth. 2007. Adaptive thresholding using the integral image. *Journal of graphics tools* 12, 2 (2007), 13–21.

[16] Sandra Carberry, Stephanie Elzer, and Seniz Demir. 2006. Information graphics: an untapped resource for digital libraries. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 581–588.

[17] Sandra Carberry, Stephanie Elzer Schwartz, Kathleen Mccoy, Seniz Demir, Peng Wu, Charles Greenbacker, Daniel Chester, Edward Schwartz, David Oliver, and Priscilla Moraes. 2012. Access to multimodal articles for individuals with sight impairments. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 2, 4 (2012), 21.

[18] Liang-Chieh Chen et al. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv:1606.00915v2* (2017).

[19] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2018. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2018), 834–848.

[20] Xiangrong Chen and Alan L Yuille. 2004. Detecting and reading text in natural scenes. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, Vol. 2. IEEE, II–II.

[21] Zhe Chen, Michael Cafarella, and Eytan Adar. 2015. Diagramflyer: A search engine for data-driven diagrams. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 183–186.

[22] Daniel Chester and Stephanie Elzer. 2005. Getting computers to see information graphics so users do not have to. In *International Symposium on Methodologies for Intelligent Systems*. Springer, 660–668.

[23] Yao-Yi Chiang and Craig A Knoblock. 2015. Recognizing text in raster maps. *Geoinformatica* 19, 1 (2015), 1–27.

[24] Chee Kheng Ch'ng and Chee Seng Chan. 2017. Total-Text: A Comprehensive Dataset for Scene Text Detection and Recognition. In *14th IAPR International Conference on Document Analysis and Recognition ICDAR*. 935–942. https://doi.org/10.1109/ICDAR.2017.157

[25] Chee Kheng Ch'ng and Chee Seng Chan. 2017. Total-Text: A Comprehensive Dataset for Scene Text Detection and Recognition. In *14th IAPR International Conference on Document Analysis and Recognition ICDAR*. 935–942. https://doi.org/10.1109/ICDAR.2017.157

[26] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. 2016. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*. 379–387.

[27] Teófilo Emídio De Campos, Bodla Rakesh Babu, Manik Varma, et al. 2009. Character recognition in natural images. *VISAPP (2)* 7 (2009).

[28] Marc Pierrot Deseilligny, Hervé Le Men, and Georges Stamon. 1995. Character string recognition on maps, a rotation-invariant recognition method. *Pattern Recognition Letters* 16, 12 (1995), 1297–1310.

[29] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88, 2 (2010), 303–338.

[30] Ross Girshick. 2015. Fast r-cnn. *arXiv preprint arXiv:1504.08083* (2015).

[31] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[32] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. 2018. Detectron. https://github.com/facebookresearch/detectron.

[33] Julinda Gllavata and Bernd Freisleben. 2005. Adaptive fuzzy text segmentation in images with complex backgrounds using color and texture. In *International Conference on Computer Analysis of Images and Patterns*. Springer, 756–765.

[34] Lluis Gomez and Dimosthenis Karatzas. 2016. A fast hierarchical method for multi-script and arbitrary oriented scene text extraction. *International Journal on Document Analysis and Recognition (IJDAR)* 19, 4 (2016), 335–349.

[35] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. MIT press Cambridge.

[36] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[37] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[38] A. Gupta, A. Vedaldi, and A. Zisserman. 2016. Synthetic Data for Text Localisation in Natural Images. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[39] Kaiming He et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[40] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2980–2988.

[41] Fu Jie Huang and Yann LeCun. 2006. Large-scale learning with svm and convolutional for generic object categorization. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, Vol. 1. IEEE, 284–291.

[42] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely Connected Convolutional Networks.. In *CVPR*, Vol. 1. 3.

[43] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. 2016. Deep networks with stochastic depth. In *European Conference on Computer Vision*. Springer, 646–661.

[44] Weihua Huang and Chew Lim Tan. 2007. A system for understanding imaged infographics and its applications. In *Proceedings of the 2007 ACM symposium on Document engineering*. ACM, 9–18.

[45] Weihua Huang, Chew Lim Tan, and Wee Kheng Leow. 2005. Associating text and graphics for scientific chart understanding. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*. IEEE, 580–584.

[46] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. 2014. Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition. *arXiv preprint arXiv:1406.2227* (2014).

[47] M. Jaderberg, A. Vedaldi, and A. Zisserman. 2014. Deep Features for Text Spotting. In *European Conference on Computer Vision*.

[48] Chandrika Jayant, Matt Renzelmann, Dana Wen, Satria Krisnandi, Richard Ladner, and Dan Comden. 2007. Automated tactile graphics translation: in the field. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 75–82.

[49] Zhao Jiuzhou. 2005. Creation of Synthetic Chart Image Database with Ground Truth. *National University of Singapore, Tech. Rep., May 2005* (2005).

[50] Christopher Kanan and Garrison W Cottrell. 2012. Color-to-grayscale: does the method matter in image recognition? *PloS one* 7, 1 (2012), e29740.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[52] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312 (2014). arXiv:1405.0312 http://arxiv.org/abs/1405.0312

[53] Tsung-Yi Lin, Piotr Dollár, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie. 2017. Feature Pyramid Networks for Object Detection.. In *CVPR*, Vol. 1. 4.

[54] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002* (2017).

[55] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.

[56] Jonathan Long et al. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3431–3440.

[57] Shijian Lu, Tao Chen, Shangxuan Tian, Joo-Hwee Lim, and Chew-Lim Tan. 2015. Scene text extraction based on edges and support vector regression. *International Journal on Document Analysis and Recognition (IJDAR)* 18, 2 (2015), 125–135.

[58] Xiaonan Lu, Saurabh Kataria, William J Brouwer, James Z Wang, Prasenjit Mitra, and C Lee Giles. 2009. Automated analysis of images in documents for intelligent document search. *International Journal on Document Analysis and Recognition (IJDAR)* 12, 2 (2009), 65–81.

[59] Hyeonwoo Noh et al. 2015. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*. 1520–1528.

[60] Joanna Isabelle Olszewska. 2015. Active contour based optical character recognition for automated scene understanding. *Neurocomputing* 161 (2015), 65–71.

[61] Nobuyuki Otsu. 1979. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics* 9, 1 (1979), 62–66.

[62] John W Ratcliff and David E Metzener. 1988. Pattern-matching-the gestalt approach. *Dr Dobbs Journal* 13, 7 (1988), 46.

[63] Sagnik Ray Choudhury and Clyde Lee Giles. 2015. An architecture for information extraction from figures in digital libraries. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 667–672.

[64] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[65] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. *arXiv preprint* (2017).

[66] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[67] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR* abs/1506.01497 (2015). arXiv:1506.01497 http://arxiv.org/abs/1506.01497

[68] Herbert Robbins and Sutton Monro. 1985. A stochastic approximation method. In *Herbert Robbins Selected Papers*. Springer, 102–109.

[69] Olaf Ronneberger et al. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. 234–241.

[70] Hanan Samet and Markku Tamminen. 1988. Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 4 (1988), 579–586.

[71] Jerzy Sas and Andrzej Zolnierek. 2013. Three-stage method of text region extraction from diagram raster images. In *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013*. Springer, 527–538.

[72] Torgyn Shaikhina and Natalia A Khovanova. 2017. Handling limited datasets with neural networks in medical applications: A small-data approach. *Artificial intelligence in medicine* 75 (2017), 51–63.

[73] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[74] Leslie N. Smith. 2015. No More Pesky Learning Rate Guessing Games. *CoRR* abs/1506.01186 (2015). arXiv:1506.01186 http://arxiv.org/abs/1506.01186

[75] Nitish Srivastava et al. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR* 15, 1 (2014), 1929–1958.

[76] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. 2013. Selective search for object recognition. *International journal of computer vision* 104, 2 (2013), 154–171.

[77] Andreas Veit, Tomas Matera, Lukas Neumann, Jiri Matas, and Serge J. Belongie. 2016. COCO-Text: Dataset and Benchmark for Text Detection and Recognition in Natural Images. *CoRR* abs/1601.07140 (2016). arXiv:1601.07140 http://arxiv.org/abs/1601.07140

[78] Andreas Veit, Michael J Wilber, and Serge Belongie. 2016. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*. 550–558.

[79] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. 2017. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. *arXiv preprint arXiv:1711.11585* (2017).

[80] Zbigniew Wojna, Alexander N. Gorban, Dar-Shyang Lee, Kevin Murphy, Qian Yu, Yeqing Li, and Julian Ibarz. 2017. Attention-based Extraction of Structured Information from Street View Imagery. *CoRR* abs/1704.03549 (2017). arXiv:1704.03549 http://arxiv.org/abs/1704.03549

[81] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, 5987–5995.

[82] Songhua Xu and Michael Krauthammer. 2010. A new pivoting and iterative text detection algorithm for biomedical images. *Journal of Biomedical Informatics* 43, 6 (2010), 924–931.

[83] Chenggang Yan, Hongtao Xie, Shun Liu, Jian Yin, Yongdong Zhang, and Qionghai Dai. 2018. Effective Uyghur language text detection in complex background images for traffic prompt identification. *IEEE transactions on intelligent transportation systems* 19, 1 (2018), 220–229.

[84] Li Yang, Weihua Huang, and Chew Lim Tan. 2006. Semi-automatic ground truth generation for chart image recognition. In *International Workshop on Document Analysis Systems*. Springer, 324–335.

[85] Xu-Cheng Yin, Chun Yang, Wei-Yi Pei, Haixia Man, Jun Zhang, Erik Learned-Miller, and Hong Yu. 2015. DeTEXT: A database for evaluating text extraction from biomedical literature figures. *PLoS One* 10, 5 (2015), e0126200.

[86] Fisher Yu and Vladlen Koltun. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

[87] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.

[88] Zheng Zhang, Chengquan Zhang, Wei Shen, Cong Yao, Wenyu Liu, and Xiang Bai. 2016. Multi-oriented text detection with fully convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4159–4167.

[89] C Lawrence Zitnick and Piotr Dollár. 2014. Edge boxes: Locating object proposals from edges. In *European conference on computer vision*. Springer, 391–405.

[90] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).