Practical Work Report: Understanding Analytical Strategies using an Image-based Approach

Fabio Pöschko
JKU
Linz, Austria
fabio.poeschko@gmail.com

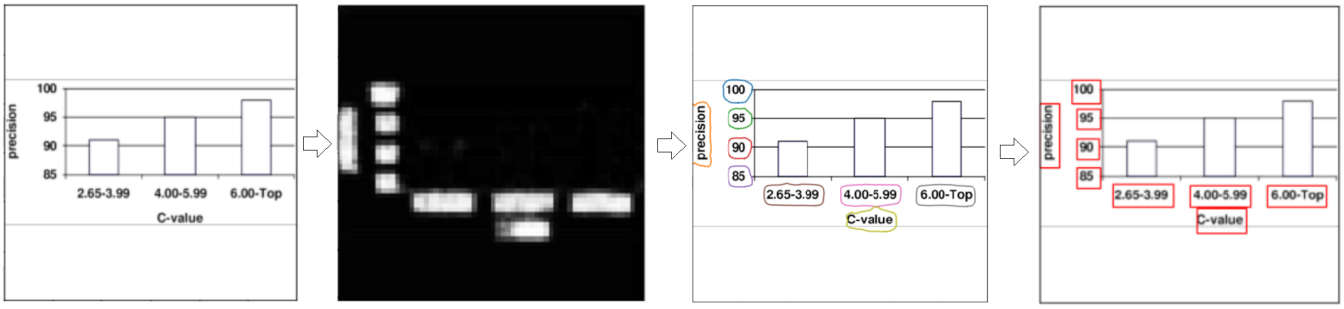

Figure 1: Visualisation of my text detection pipeline

1 GOAL

The goal of this project was to reverse engineer bitmaps of visualisations, to identify text elements such as chart title, x-axis, or y-axis, such that this can be used to derive analysis strategies of users of visual analytics tools. To come up with these analytical strategies a recurrent neural network can be trained to automatically classify the changes made between two visualisations.

2 RELATED WORK

My work builds upon prior research in the area of text localisation.

2.1 Reverse-Engineering Visualisations [3]

In the paper "Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images" the authors present a way to decode a bitmap image into its visual encoding specifications. With their method they already achive good results, reporting F1-scores of ca.88% in Text localisation, ca.99% in text role classification and ca. 94% in mark type classification, but their Github repository has a lot of missing files, which makes it impossible to use their code even though their approach works, according to their paper.

2.2 Domoritz Label Generator [1]

In this article the author explains how to classify text pixels in scientific visualisations. To do this the author uses a CNN in the Darknet [4] which takes as input the visualisation and outputs a heatmap of text pixels.

3 METHODS

I started by trying to make the code from the project Reverse-Engineering Visualisations [3] work, but when doing so i realised, that some of their code files were missing, so i moved on to recreating the project in python 3, below one can see what i have recreated.

3.1 Preprocessing

The Method starts by padding the original bitmap of the visualisation, such that it is a square, then resizing the image to 1200x1200 pixels and binarizing the images with Otsu's threshold [2], like in the original paper.

3.2 Heatmap

The heatmap is generated using a CNN which is modeled after the project domoritz label generator [1], this heatmap displays where the text is in the visualisation, as seen in the second picture of Figure 1. The original project used a CNN which was trained with the Darknet [4] which i thought was impractical to use, so i downloaded the acl anthology part of the training data from the "Amazon Web Services S3 Bucket", and trained a similar CNN with pytorch, there would be a lot more training data from arxiv, but this is so much data that the free storage of my computer would not be enough to store it.
I got the network architecture by looking at the file with the network architecture in the domoritz label generator [1] github repository, i only had to do some small adjustments, like leaving out the croping layer and implementing the ramp activation function, because pytorch did not offer it.
In Figure 3 the exact architecture I ended up using is displayed.

3.3 Contour Lines

I then used contour lines to draw circles around the pixels where text might be, in the third picture of Figure 1 some contour lines are shown.

3.4 Bounding Boxes

To get the bounding boxes i used the max distance of the circle pixels as boundaries boxes, an example can be seen in the fourth picture of Figure 1. With these on these boundary boxes i then used

pytesseract a python wraper for tesseract [5], an optical character recognition module, to give me the text in the box, because the text sometimes has a different orientation i rotate the box and then take the text in which pytesseract is most confident in.

4 RESULTS

I implemented a way to locate text in visualisations and create the text characters from a bitmap image, this of course has some flaws, which are problems with text localisation if the labels are tightly spaced and if someone uses for example "O" as a plotting symbol it is recognized as text. These flaws are also listed as flaws in the original paper, there are some examples of these mistakes in Figure 2.

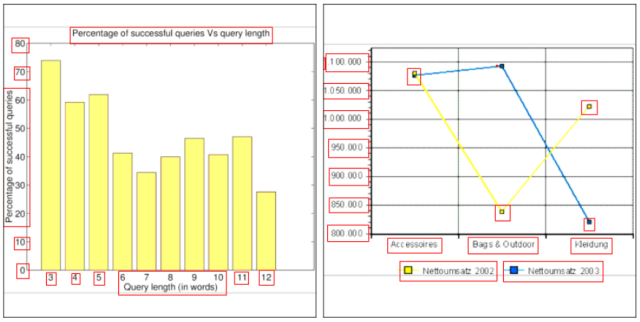

Figure 2: Common problems, on the left there are merged bounding boxes, and on the right are plotting symbols which get identified as characters

5 CONCLUSION

So the first part of the original paper was done, the part with the text localisation, in the github repository of the original paper there are models, which can theoretically be used for the later parts, but i already tried to make them work, when beginning to work on this project to see what i could use from the Reverse Engineering Visualisations Paper, but i just could not make them work.
So to finish the project, someone would need to rebuild the text role classifier and the mark classifier, with these tasks the reverse engineering part would be complete and then one could use RNN´s to identify the changes between each interaction.

6 DOCUMENTATION OF MY CODE

All the code I used for the project can be found on my github repository at https://github.com/FabioPoe/PW, generally everything should be clear, from the filenames and the comments in the files, but the files data_preprocessing.py, domoritz.ipynb and domoritz_colab.ipynb were used to train the CNN which was used to generate the heatmaps used for text detection.

6.1 domoritz_colab.ipynb

The version of the CNN training for colab was made, because i realized, that a CNN would probably be way faster on a GPU than my normal CPU, and since I don't really have a good GPU in my PC, I decided to use google colab for my training, with their GPU it trained 10 times faster. Of course in the files there are some absolute paths, which one would have to change, before recreating the project.

6.2 CNN

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # i decided that all input images will be preprocessed like in the reverse ingeneering paper
        # to 256x256 such that i dont need the croping layer
        # and i changed the kernel size of the last kernel to get the output of 64x64
        self.conv1 = nn.Conv2d(in_channels=1,out_channels=32, kernel_size=[7, 7],stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32,out_channels=32, kernel_size=[3, 3],stride=2, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32,out_channels=32, kernel_size=[3, 3],stride=1, padding=1)
        self.conv4 = nn.Conv2d(in_channels=32,out_channels=1, kernel_size=[2, 2],stride=1, padding=1)
        self.sigmoid = nn.Sigmoid()

    def ramp(self,x):
        #ramp activation function
        x[x < -1] = -1
        x[x > 1] = 1
        return x

    def forward(self, x):
        x = self.ramp(self.conv1(x))
        x = self.ramp(self.conv2(x))
        x = self.ramp(self.conv3(x))
        x = self.sigmoid(self.conv4(x))
        return x
```

Figure 3: The exact architecture of the heatmap generating CNN

This is the CNN that was used in the domoritz project, rebuilt in pytorch, only with some small changes, like no cropping layer and a smaller kernel size in the last layer to get the desired output

6.3 How to run my code

To run my code first you need to install tesseract, and then a environment which fulfills the requirements in environment.txt needs to be installed, if one has conda, one can just run the command:

$ conda env create –name <environment name> –file <filepath of environment.yaml>

after this the environment should be set up and the only thing left to change in the code would be some absolute paths, like teh path of tesseract, or where data is stored.

REFERENCES

[1] Dominik Moritz. 2017. Text detection in screen images with a Convolutional Neural Network. *The Journal of Open Source Software* 2, 15 (July 2017), 235. https://doi.org/10.21105/joss.00235

[2] NOBUYUKI OTSU. 1979. A Threshold Selection Method from Gray-level Histograms. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, 5.

[3] Jorge Poco and Jeffrey Heer. 2017. Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images. EuroVis, Washington, 11.

[4] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.

[5] R. Smith. 2007. An Overview of the Tesseract OCR Engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Vol. 2. 629–633. https://doi.org/10.1109/ICDAR.2007.4376991