JOHANNES KEPLER
UNIVERSITÄT LINZ

Submitted by
Fabio Pöschko

Submitted at
Institute of Computer Graphics

Supervisor
Univ.-Prof. DI Dr. Marc Streit

Co-Supervisor
Conny Walchshofer, Andreas Hinterreiter

September 2022

Automatic Extraction of Chart Specifications from Images

Bachelor Thesis
to obtain the academic degree of
Bachelor of Science
in the Bachelor's Program
Artificial Intelligence

Abstract

Users of interactive visual analytics tools naturally come up with analysis strategies. Deriving those strategies is a challenging research topic. Current approaches primarily log interactions, such as click events with the system, to replicate the analysis workflow. However, visual analytics tools often do not make the raw data easily accessible. We propose that there is a way of using simple bitmap images or even sampled video sequences to get these click events. We first capture where text is in an image with a convolutional neural network and then use this information to compute bounding boxes for each letter, merge the boxes to words and classify the role of the boxes. We also classify the mark type of the chart, with the well known convolutional neural network designed by Alex Krizhevsky et al., named Alexnet, with all this information we can infer the specifications of each bitmap chart. We then can use this information to infer what changed from one picture to another.
We also evaluate our method of text localization and recognition with examples, discuss common pitfalls, and we check the performance of our mark classifier with our testset.
The contribution of this thesis is a reproduction of an earlier paper on Reverse-Engineering Visualizations from Chart Images (Poco and Heer,2017), with modern infrastructure and a slight differences in the approach, like applying the heatmap differently and using different pruning methods for trees in the text localization step, leaving out some features and using a random forrest classifier instead of an SVM in the text localization step, a different dataset for mark classification, and the first tries to apply this process to generate a pipeline to automatically extract click events, by only looking at different states of visualizations.

Contents

1 Introduction

Users of interactive visual analytics tools instinctively come up with strategies to analyze data with these tools. Deriving such strategies is a difficult research topic. Most of the time current approaches rely on manually logging events, such as mouse clicks and movements, to copy a humans workflow, often these workflow logs are not accessable for researchers. We propose, that there is a way of using images of these visualizations to get such click events. We begin locating text in an image with a convolutional neural network (CNN), then we compute bounding boxes for each letter and later words, with this information. In the next step we use a Random Forrest classifier to classify the role such a box plays in a visualization. Afterwards we also classify the mark type of the chart and we use all this information to create a specification for the given visualization.
The primary contribution of our work is an approach to extract information from bitmap images to help researchers in analyzing human-computer interactions, just from screenshots or videos of a person performing tasks. To get these interactions, we first have to look at the states of the visualization image and transform it into text. For this we first use a CNN to get the mark type of the image, then we use another CNN and some connected components algorithms and Trees to get bounding boxes of the words, and then we use a random forest to classify the role of the box. Then we can put this all together into a python dictionary format to show the specification of the visualization. With this specification we can then easily see what changed from one frame to another, by comparing the differences of the dictionaries.

Section 2 of this thesis covers related work. Next in the methods section, our approach and implementation details are described. Afterwards, the results of our implementation are shown and evaluated, with quantitative metrics and examples, the limitations of this thesis are described and future work is discussed. Finally in the conclusion, a summary of the obtained results is given.

2 Related Work

In this section, we discuss previous approaches to the three areas of related reverse engineering visualizations, which are chart image classification, text detection and recognition and data extraction from charts.

2.1 Chart Type Classification

Image classification of computer generated charts is a well studied problem, some papers proposed that algorithms used in classification of natural scenes can be applied [19] [17] and others proposed specialized techniques for classification [7].
Prasad et al.[12] considered five categories in their approach for classifying chart images. They classified bar-charts, curveplots, pie-charts, scatter-plots and surface-plots, by introducing two new features for representing structural information and using Histograms of Oriented Gradients (HOG) and the Scale Invariant Feature Transform (SIFT) descriptors to characterize the local shape, with this they represented the chart image as a set of feature vectors, which they then classified using a support vector machine (SVM).
Savva et al. [14] developed a classification method which first extracted low-level image features using a SVM and then they extracted text features and use OCR, with their method they achived an accuracy of over 96% on the categories Area Graphs, Bar Graphs, Curve Plots, Maps, Pareto Charts, Pie Charts, Radar Plots, Scatter Plots, Tables and Venn Diagrams.
Jung et al. [3] classified the chart type by using deep learning in their approach.
Tang et al. [18] developed DeepChart, they first used convolutional networks to extract deep hidden features of charts and then they used deep belief networks to classify the charts based on their deep hidden features. They classified the charts into 5 different categories, and their approach was evaluated against other existing methods.

2.2 Text Localization and Recognition

Chart images contain graphical elements and text, the content and the position. These textual components can be extracted from the image using text localization, detection and recognition methods.
Kataria et al. [4] showed how data and text could be automatically extracted from chart images, by extracting connected components and treating the components as candidates for text characters. Components with a too large area were removed and words were merged by considering the distance between the components.
Poco and Heer [11] created a less error-prone approach than Kataria et al. [4], by using a CNN to generate a heatmap which they used to remove non-text pixels. Poco and Heer [11] generated the heatmap using prior work from Moritz [8], he generated the heatmap using Darknet [13] and around 500000 samples of self collected training data.

2.3 Text Role Classification

The task of identifying the role a text plays in a chart, like axis label or legend label, was used mostly for creating better search results in search engines, but is also vital for reverse engineering visualizations.

In their search engine improvement Chen et al. [1] use bounding box feature vectors for classification and Choudhury et al. [2] used text bounding boxes and text content for a similar task. Poco and Heer [11] used a similar approach, but without the text content, since the OCR error would be further propagate through their method. We also followed a similar approach.

As we see in this Section, there is no prior work which has an entire workflow for extracting a chart specification, which is actually usable, there is the work from Poco and Heer [11], which works well according to their paper, but they did have some missing code files in the github repository, where they published their code, other files were only partly there, they wrote their project in a python 2 version and they used caffemodel, which is a python deep learning framework, which was last updated in 2017 and is outdated by now. With this knowledge, the first goal this of thesis was to reproduce their work, to use it for analyzing sequences.

3 Method

This section shows all steps and implementation details, we used to obtain the results of chapter 4. Firstly, the text localization method is described, then text role classification and mark classification. Things like evaluation metrics are described right before they are used.
reasons: missing files other files are only partially there old code, python 2.7 caffeemodel endless dependencys when importing some models with pickle

3.1 Text Localization

The first task we had to solve was to localize the text, because Poco and Heer [11] already created a well working way for text localization, we tried to use their code, but since some files in their github repository were missing and others were only partially there we could not use their code for our method and we decided to adapt their approach as well as we could.

3.1.1 Preprocessing

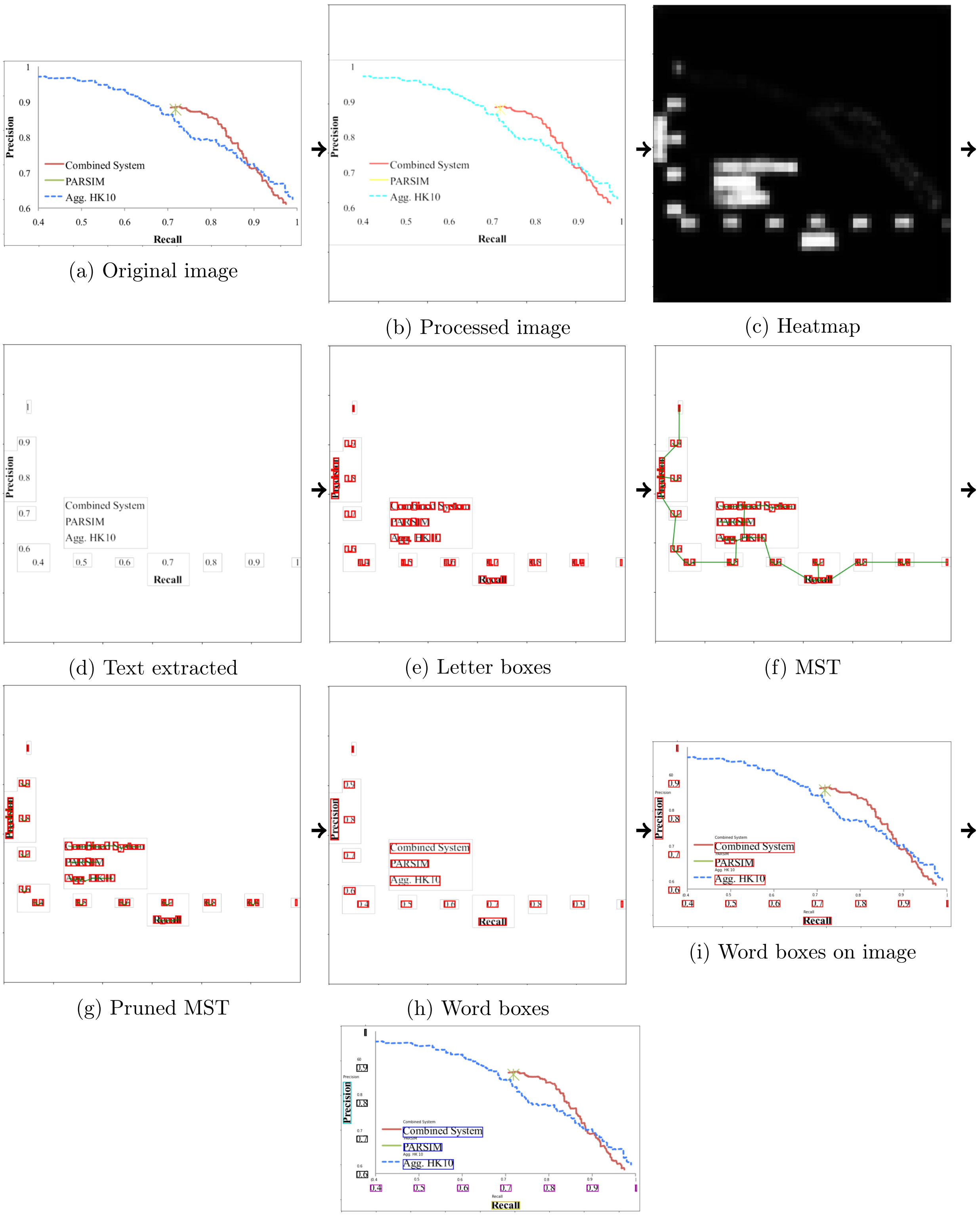
Our Method starts by padding the original bitmap visualization to a square, and then we either increase or decrease its size to fit 1200x1200 pixels. In our next step the image gets binarized using Otsu's threshold [9], this threshold assumes each pixel is either foreground or background and tries maximizing inter-class variance to get the threshold. An example of this step can be seen in figure 1b.

3.1.2 Heatmap

We generated our heatmap we need for removing non text pixels using almost the same CNN Moritz used in his project domoriz label generator [8], using his net would be impractical, since it was trained with Darknet [13]. We only left out the cropping layer, because it was not needed since our input images have the same input size and shape. Our dataset for training this heatmap was collected using the data provided by Moritz on his "Amazon Web Services S3 Bucket", and we only used the acl anthology part of the data, because the arxiv data would have been more than we could store on our computer. In Figure 1c an example heatmap is shown.
This heatmap is then used to remove non text pixels. To do this we resized the heatmap to the same size as the preprocessed image and then ran the marching squares method to find constant valued contours, from the original image we then kept everything inside the contours and everything else was deleted,the result of this can be seen in figure 1d. We decided to do this differently to the approach of Poco and Heer [11], because it was more intuitive this way, and we could later on change some contour finding values easily, to make the contours more slim or wide, which can change the outcome of the entire text localization step.

3.1.3 Connected Components

We decided to run a connected components algorithm to find characters in the image. From this algorithm we get bounding boxes for each connected component in the image. To filter out boxes which are very unlikely to be actual letters, we delete boxes where width/height is smaller than 0.1 or bigger than 10. Another filter criterion was, that the area of the box had to be between 4 and 4000, this proved to make the most sense in our experiments, and it also makes sense intuitively, because if a letter would have less than 4 pixels in a 1200x1200, it would only make up 1/360000 of the image which would be somewhat unreadable, and on the other side the letter can maximally take 1/360 of an image. An example of such cases can be seen in Figure 2, where at the top left a slightly too big "8" is displayed and on the bottom left a 4 pixel i is displayed, but it is almost invisible, because it is so small.



(a) Original image (b) Processed image (c) Heatmap

(d) Text extracted (e) Letter boxes (f) MST

(g) Pruned MST (h) Word boxes (i) Word boxes on image

(j) Classified boxes

Figure 1: Text localization and classification pipeline

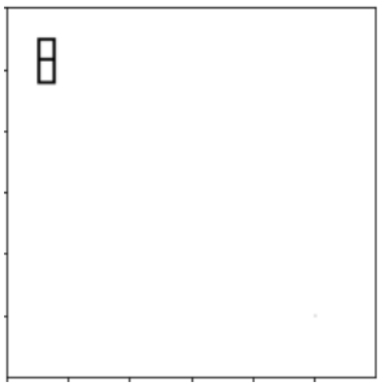


Figure 2: example invalid letters

3.1.4 Minimal Spanning Tree

To make bounding boxes for the words of the boxes we have from the letters, we created a minimal spanning tree (MST) with Kruskal's algorithm [6], to do this we used the distance between the letters as the edge weight and the letters as nodes. In Figure 1f such a minimal spanning tree is displayed.

3.1.5 Pruning

To isolate words we decided to additionally prune the MST, we defined 3 criteria for removing edges:

- edges that are too long
  - To find edges which are too long, we created a set with all candidate characters and took the most common height of a box, since normally most characters are upright in a visualization this should give a good estimate of character height, we then discard edges which are smaller than $1.5 \times l$, where l is the most common character height. In the approach by Poco and Heer [11] they made a set of all character heights and widths, but in our experiments we realized, that the most common element of this is character width. Though this would have also worked in the pruning step, we decided to go with character height.
- edges which are not aligned
  - To find edges which were not aligned for each edge we look at the bounding boxes. Because we have the general assumption, that the words are either at a 0 or 90 degree angle, if the boxes are aligned at such an angle, they either overlap horizontally or vertically. Therefore we identify boxes which overlap vertically of horizontally at least half of the height of the smaller box as aligned and we keep the edge, other edges are removed.
- edges which lead to different colors
  - To filter out some bullet points before words, we remove edges which lead to different colors. We first calculated the CIE LAB color for each pixel in a letter and then we delete the edge if the CIEDE2000 color difference [15] is bigger than 20. We decided to use the CIE LAB color scheme, because this approach worked for Poco and Heer [11] and they figured out, that a color difference of 20 would remove enough edges.

3.1.6 Merging Boxes

To create word boxes from the letter boxes we look at every chain of connections and create the word box of the most outer part of each letter box, an example of this can be seen in Figure 1g and Figure 1h.

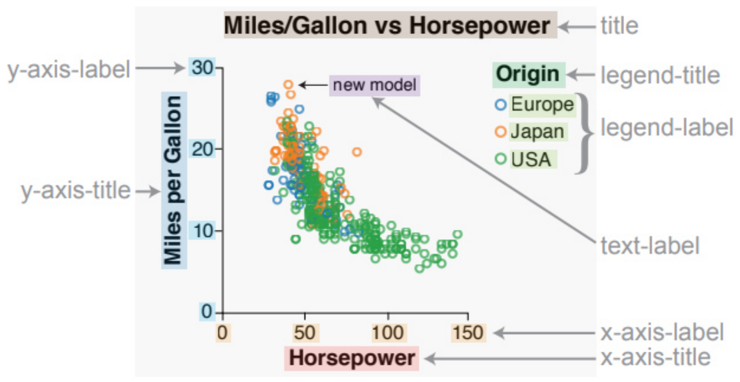


Figure 3: Example text roles in a scatterplot (reproduced from Poco and Heer [11]

3.1.7 OCR

To recognize which characters and words are in the bounding boxes, we first rescaled the word box with the factor 3 in order to increaze the resolution, which should help to make the performance of our optical character recognition (OCR) Tesseract [16], better. Because we also get a confidence score from Tesseract, we rotate the image by 0,90 and 270 degrees and take the rotation with the highest confidence for the final prediction. If the highest confidence is below 25 we discard a box, because then it is unlikely to be a word. We also noticed, that the OCR engine often confuses "0" for "o" or "O", and "1" for "|" or "I", since it is way more likely to be a number than a vowel, because often an axis starts or ends at 0 or 1, we decided to switch every single 'O' or 'o' to '0' and 'I' to '1'.

3.2 Text Role Classification

The second task we had to solve is to now classify the role of the localized text in the visualization, we decided to use the same 8 types Poco and Heer [11] used in their paper, an example can be seen in Figure 3. We calculated the role of each box by first calculating the features which we then input into a random forrest. We decided on using a random forrest classifier instead of the SVM classifier Poco and Heer [11] used. We made this decision because we were not satisfied with the results we got from the SVM and the random forrest classifier worked a little better, when applying it to actual examples, where we got the word boxes from the earlier method.

3.2.1 Feature Calculation

Because the approach with the calculated features achieved a satisfactory accuracy for Poco and Heer [11] and their data was usable, we decided to keep mostly keep it the same. We decided to leave out some features, because we thought they do not make sense intuitively and we wanted to only keep features which are easily interpretable. To make sure this does not impact performance, we compared an SVM with all features from Poco and Heer [11] and an SVM with only the features we intended to use and we only noticed a drop in F1 score from around 98% to around 97% which was still satisfying.

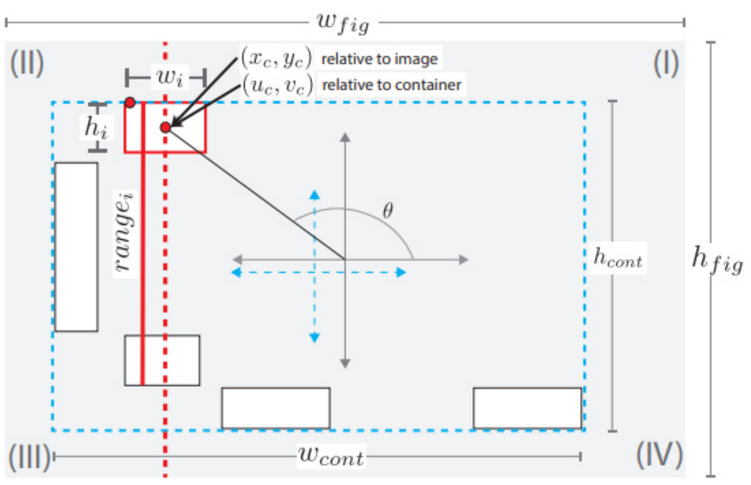


Figure 4: Bounding box features (reproduced from Poco and Heer [11]

An example of every variable we use to explain the features can be seen in Figure 4.
The features we used are:

- normed x coordinate
  - x_c/w_{fig}
    x_c is the x coordinate of the bounding box center
    w_{fig} is the figure width
- normed y coordinate
  - y_c/h_{fig}
    y_c is the y coordinate of the bounding box center
    h_{fig} is the figure height
- aspect ratio
  - w_i/h_i
    w_i is the bounding box width
    h_i is the bounding box height
- quadrant
  - Figure 4 illustrates this concept well, the roman numerals in the brackets are the quadrant numbers, and the gray origin of the coordinates, used for θ in the middle of the figure can be seen as the border from one quadrant to another.
- normed container height
  - h_{cont}/h_{fig}
    h_{cont} is the container height
    h_{fig} is the figure height
- normed container width
  - w_{cont}/w_{fig}
    w_{cont} is the container width
    w_{fig} is the figure width

- normalized center x-coordinate relative to the container
  - u_c/w_{fig}
    u_c x-coordinate relative to the container
    w_{fig} is the figure width
- normalized center y-coordinate relative to the container
  - v_c/h_{fig}
    v_c y-coordinate relative to the container
    h_{fig} is the figure height
- vertical score
  - Vertical score is defined as the number of all vertically intersecting boxes, with the current box divided by the number of all boxes we found in the picture. For example in Figure 4 the red box intersects with another box and itself vertically, therefore its verticals score is 2/5. This is conflicting to how Poco and Heer [11] described this feature in their publication, but this choice proved more suitable for reproducing their results.
- horizontal score
  - same as vertical score, but horizontally

3.3 Mark Classification

Another vital part of a chart specification is the mark type, in general there are 3 types of marks areas, lines and points. For our purposes we decided to make a classifier which classifies lines, points, bars and areas.

3.3.1 Dataset

We got our dataset from the github repository of the paper, by Poco and Heer [11], where they shared a google drive link to download some of the data they used for their model. The classes were pretty imbalanced, we had 501 bar charts, 550 line charts, 292 scatterplots, 153 area charts. Before feeding the images into the model, we also decided to preprocess the images, by padding the image to a square and then rescaling it to 1024x1024 and we split the dataset into 80% training data and 20% test data.

3.3.2 Model

As a model we used the Alexnet [5] implementation from pytorch [10], we did not freeze any layers, unlike Poco and Heer [11], because in our experiments we got way better accuracy on the testset with this approach, and the extra training time this needed was not an issue for us, because we were able to utilize powerful GPU's from Google Colab.

3.4 Specification Induction

With the classified bounding boxes, the text of these boxes and the mark type we could create a visual encoding specification for a chart image, as shown in Figure 5. To get this specification we used the results of the methods shown earlier in this section and we also had to compute things, like range or type.

```
{'width': 861,
 'height': 534,
 'chart type': 'line_chart',
 'mark type': 'line',
 'title': '',
 'encoding': {'x': {'field': 'Recall',
   'axis': {'title': 'Recall'},
   'type': 'quantitive',
   'scale': {'labels': ['(0.4', '0.5', '0.6', '0.7', '0.8', '0.9', '|'],
    'values': [0.5, 0.6, 0.7, 0.8, 0.9],
    'range': [0.5, 0.9],
    'type': 'linear'}},
  'y': {'field': 'Precision',
   'axis': {'title': 'Precision'},
   'type': 'quantitive',
   'scale': {'labels': ['|', '60', '0.8', '0.7', '0.6'],
    'values': [60.0, 0.8, 0.7, 0.6],
    'range': [0.6, 60.0],
    'type': 'undefined'}}}}
```

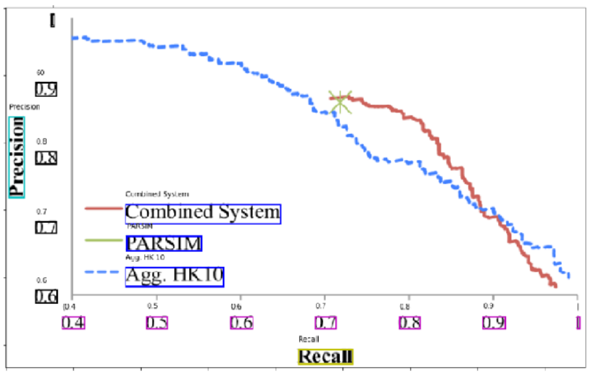


Figure 5: Example specification of our method

Next we will go through every specification and explain how we get it.

- width and height
  - For width and height we decided to just use the width and height of the input image, since the input image should display only the visualization.
- chart type and mark type
  - Here we could just use the result of feeding the picture into the method of Section 3.3. Then we get something like line chart or bar chart for the chart type, and then the mark type is line or area.
- title, field, axis title and labels
  - We got title, field, axis title and labels by using the classified boxes, which were detected by the method from Section 3.1 and then we used the procedure described in Section 3.2 to get the role of these boxes, title corresponds to the role title, field shows the title for a given axis, axis title does the same, labels show the text, which was found by the OCR procedure described in Section 3.1.7, for all boxes classified as label of a specific axis by the method in Section 3.2 as a list.
- values
  - To get the values of the x- or y-axis labels, we used the labels, explained right above. We go through every label, at first we just check, if we can convert it to a number without any problems If this is not possible, we split the string into numbers and letters, if there is no number, we move onto the next label. If there is a number we also check the letters for a common modifier, like %, which would mean that the number should be multiplied by 0.01, or M, which would mean, that the actual number should be multiplied by 1000000. If no such multiplier is found we just assume that the detected letter was just a mistake by any of the systems before. The last step is now to multiply the number by the multiplier and put the result in the values list
- type
  - We set the type to quantitative, if there were any numbers in the values, and nominal if there were no numbers found for the values list.
- domain

  - The domain is a list of the minimum and maximum value found in a label box of an axis.

- range
  - We defined the range as a list with the minimum and maximum coordinate where a box was found, for the x-axis we took the x-coordinates and for the y-axis we took the y-coordinate.

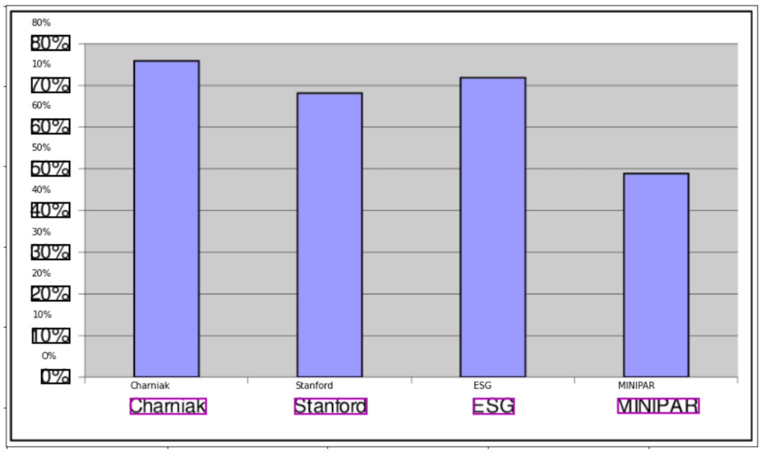
- type
  - This type is a more specific type than the one above, if the above type is nominal, than this also set to nominal, but if the above type is set to quantitive, then this type tries to specify more specifically if the axis is logarithmic or linear.

3.5 Sequential Analysis with Pictures

With the entire pipeline before we can compare specifications and then say what changed from one picture to another. To do this we did not write a specific algorithm, because this would be beyond the scope of this bachelor thesis, but we can compare the specifications by just reading the output of our method. Some examples can be seen in Figure 8.

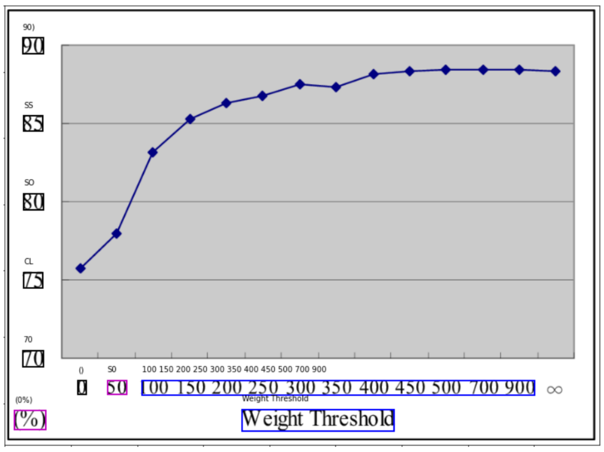
4 Results

Finally, the results of the methods described in the last Section, are presented. First we will go through every step and evaluate it.



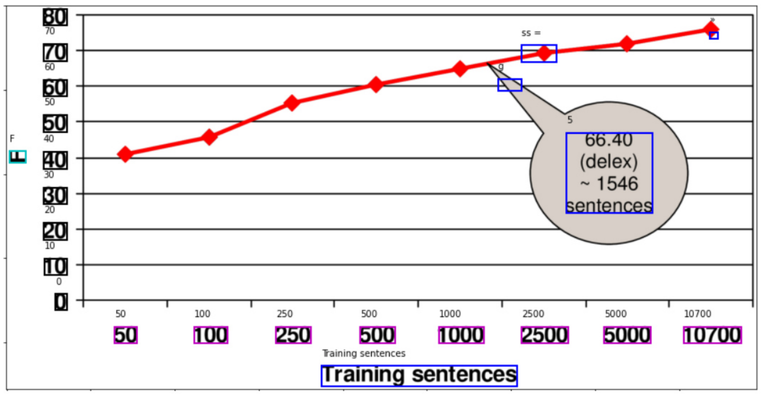
```
{'width': 936,
 'height': 549,
 'chart type': 'scatterplot',
 'mark type': 'point',
 'title': '',
 'encoding': {'x': {'field': '',
   'axis': {'title': ''},
   'type': 'nominal',
   'scale': {'labels': ['Charniak', 'Stanford', 'ESG', 'MINIPAR'],
    'values': [],
    'domain': [],
    'range': [201.63, 811.98],
    'type': 'nominal'}},
  'y': {'field': '',
   'axis': {'title': ''},
   'type': 'quantitive',
   'scale': {'labels': ['80%',
     '10%',
     '60%',
     '50%',
     '40%',
     '30%',
     '20%',
     '10%',
     '0%'],
    'values': [10.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 80.0],
    'domain': [10.0, 80.0],
    'range': [46.34999999999999, 461.7],
    'type': 'linear'}}}}
```

(a) Example 1: result of the method on a visualization, where everything worked great



```
{'width': 494,
 'height': 255,
 'chart type': 'line_chart',
 'mark type': 'line',
 'title': '',
 'encoding': {'x': {'field': '',
   'axis': {'title': ''},
   'type': 'quantitive',
   'scale': {'labels': ['0', '3', '', 'a', '|', '7', '4', 'A', 'Pa'],
    'values': [0.0, 3.0, 4.0, 7.0],
    'domain': [0.0, 7.0],
    'range': [105.18083333333334, 324.1875],
    'type': 'linear'}},
  'y': {'field': '',
   'axis': {'title': ''},
   'type': 'nominal',
   'scale': {'labels': ['Ne pee', 'Poses mascata'],
    'values': [],
    'domain': [],
    'range': [80.36416666666665, 153.0233333333333],
    'type': 'nominal'}}}}
```

(b) Example 2: result of the method on a visualization, where the text labels merged, because they were close to each other, and the box in the bottom-left corner was misclassified, because it is a box in a really uncharacteristic position



```
{'width': 1785,
 'height': 912,
 'chart type': 'scatterplot',
 'mark type': 'point',
 'title': '',
 'encoding': {'x': {'field': '',
   'axis': {'title': ''},
   'type': 'quantitive',
   'scale': {'labels': ['50',
     '100',
     '250',
     '500',
     '1000',
     '2500',
     '5000',
     '10700'],
    'values': [50.0, 100.0, 250.0, 500.0, 1000.0, 2500.0, 5000.0, 10700.0],
    'domain': [50.0, 10700.0],
    'range': [284.1125, 1679.3875],
    'type': 'undefined'}},
  'y': {'field': 'F',
   'axis': {'title': 'F'},
   'type': 'quantitive',
   'scale': {'labels': ['80', '70', '60', '50', '40', '30', '20', '10', '0'],
    'values': [0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0],
    'domain': [0.0, 80.0],
    'range': [27.599999999999987, 704.4125],
    'type': 'linear'}}}}
```

(c) Example 3: result of the method on a visualization, where some plotting symbols are wrongly classified as text

Figure 6: Example results of the method on different visualizations

4.1 Results for Text Localization

Our text localization pipeline was able to localize some text, create bounding boxes around the given text, and use an OCR program to read the text. The text localization pipeline was successful in most cases, but like in the paper by Poco and Heer [11] it still has some clear flaws. Some examples of the text localization can be seen in Figure 6, by looking at where the boxes are located. For looking at the text localization, only the location of the bounding box, and the text on top of it are relevant.

4.2 Results for Text Classification

The trained random forrest classifier reached an accuracy of around 98% on the testset on most tries. We tried the same with the SVM proposed by Poco and Heer [11], the SVM reached an accuracy of around 97% on the testset on most tries. However when looking at the classification, on the boxes we got with the text localization method (Section 3.1) on example pictures we noticed, that even from just slightly different boxes the class was then often wrong and the accuracy on our boxes was just around 60% with the SVM, even after their post processing step and with the random forrest, the accuracy on our boxes was around 85%.

4.3 Results for Mark Classification

With our mark classification net we were able to classify 95% of our testset images correctly. The only alarming thing is, that our trainset accuracy is 100%, which could indicate overfitting of our model, we tried to solve this by freezing some layers, or implementing some dropout layers, but they all led to a worse accuracy on the testset, therefore we accepted the trade-off. In Section 3.3 we mentioned, that our dataset is imbalanced, in the confusion matrix displayed as Figure 7, it can be seen, that this imbalance did not cause any problems, because there is not a class which the classifier never predicts and there is no misclassification, of for example lots of Barplots, that get classified as Lineplots.

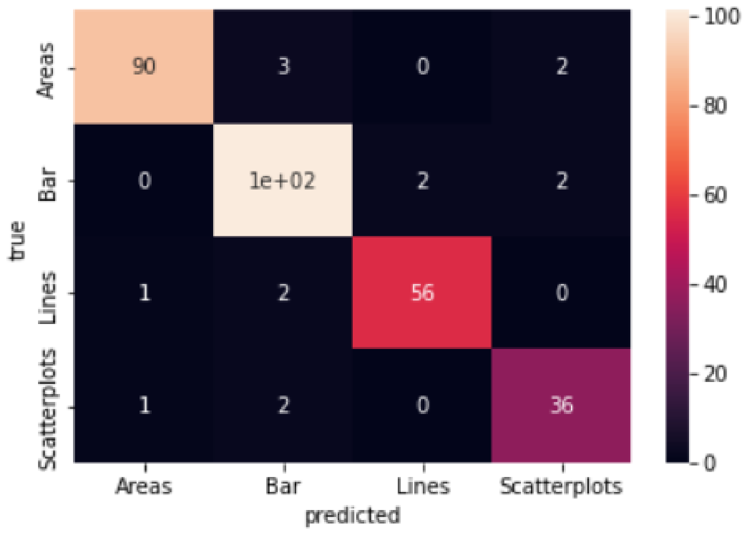


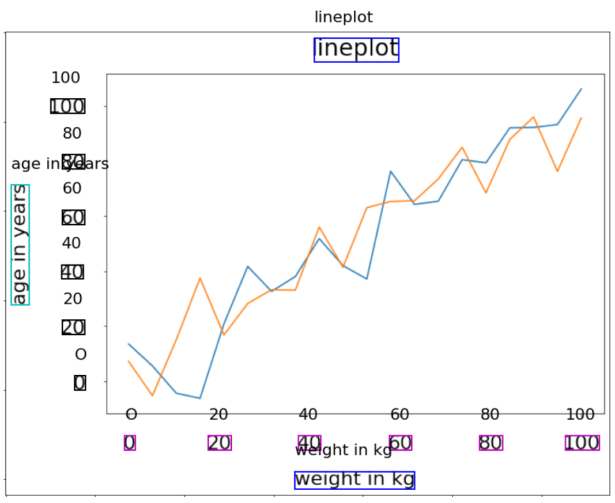
Figure 7: mark classification confusion matrix on the testset

4.4 Results for Specification Induction

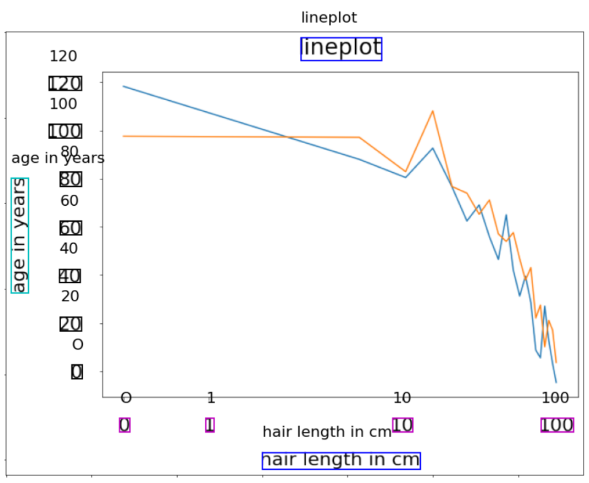
Our specification induction results were difficult to express in numbers, like accuracy, f1-score or other metrics, because we did not have a real dataset with these specifications, therefore we evaluated it by producing some example specifications and looking at the pictures. In Figure 6 such specifications can be seen. In general, there were three different kind of mistakes the pipeline made, the first mistake is merging some boxes because the labels were too close together, this can be seen in Figure 6b, where the x-axis label boxes merged together, the second common mistake is misclassifying boxes which are in uncharacteristic positions, like the % sign in Figure 6b and the last and most common mistake the method makes is to wrongly classify some plotting signs in the middle of the plot as characters, an example can be seen in Figure 6c. There were also some rare visualizations, like Figure 6a, where we managed to get an accurate specification.

4.5 Results for Sequential Analysis

When we tried our method on screenshots or a sampled down version of a video, we had the problem, that the otsu thresholding used during the text localization process displayed some odd behaviour, which can be seen in Figure 9a, where from the letters the inner colour got removed. Because of this we got a heatmap, which detected no text, which led to the heatmap removing almost every bit of text in the image and then we did not find any text. Because of time constraints, we did not examine this error any further.
When we are able to save the figure automatically from the program it was generated with, then we were able correctly generate the specification, as it can be seen in Figure 6, with this it would be possible to recreate change logs from visual analytics tools, but only if the visualizations don't fall into any of the common mistakes our method makes, which were mentioned in Section 4.4.

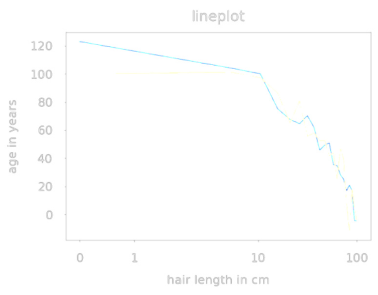


```
{'width': 677,
 'height': 518,
 'chart type': 'scatterplot',
 'mark type': 'point',
 'title': '',
 'encoding': {'x': {'field': '',
   'axis': {'title': ''},
   'type': 'quantitive',
   'scale': {'labels': ['O', '20', '40', '60', '80', '100'],
    'values': [20.0, 40.0, 60.0, 80.0, 100.0],
    'domain': [20.0, 100.0],
    'range': [139.06708333333336, 645.9708333333333],
    'type': 'linear'}},
  'y': {'field': 'age in years',
   'axis': {'title': 'age in years'},
   'type': 'quantitive',
   'scale': {'labels': ['100', '80', '60', '40', '20', 'O'],
    'values': [20.0, 40.0, 60.0, 80.0, 100.0],
    'domain': [20.0, 100.0],
    'range': [82.69791666666664, 392.1433333333333],
    'type': 'linear'}}}}
```

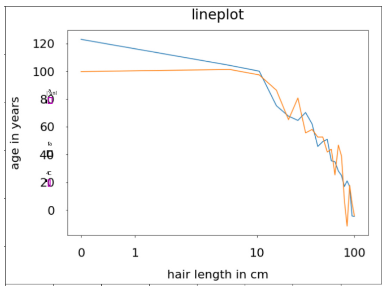


```
{'width': 677,
 'height': 518,
 'chart type': 'scatterplot',
 'mark type': 'point',
 'title': '',
 'encoding': {'x': {'field': '',
   'axis': {'title': ''},
   'type': 'quantitive',
   'scale': {'labels': ['O', '1', '10', '100'],
    'values': [1.0, 10.0, 100.0],
    'domain': [1.0, 100.0],
    'range': [139.06708333333336, 645.9708333333333],
    'type': 'logarithmic'}},
  'y': {'field': 'age in years',
   'axis': {'title': 'age in years'},
   'type': 'quantitive',
   'scale': {'labels': ['120', '100', '80', '60', '40', '20', 'O'],
    'values': [20.0, 40.0, 60.0, 80.0, 100.0, 120.0],
    'domain': [20.0, 120.0],
    'range': [59.84916666666663, 397.2208333333333],
    'type': 'linear'}}}}
```

Figure 8: Example result, where the output specification can be compared by a computer and the difference between the visualizations can be found with our method



(a) Result of Otsu thresholding on a screenshot



(b) Result of the method on a screenshot

Figure 9: Results on a screenshot

5 Limitations

One of our biggest limitation was data, it was hard to find data, which was good and looked like the data we would like to use the method for. We had to use old and outdated visualizations, like the ones in Figure 10 for training the mark classifier, because this was the best we had access to. We achieved a good accuracy on this data, but we do not know, how the classifier performs on newer visualizations with modern standards, even though we think that it should not be much worse since modern visualizations have gotten a lot more clearer and easier to understand for humans.
Another limitation of our work is, that only specific visualization types work. The challenges with making it work for more visualization types would be, that for mark classification, there is not a lot of good data for the mark classifier and text role classification could become problematic, because some features might not work anymore.
Moreover, this work was limited by the time we had to make each step work, there could be a lot more approaches which could be tested, like trying to create distinct methods for different mark types, or trying to improve each step further.

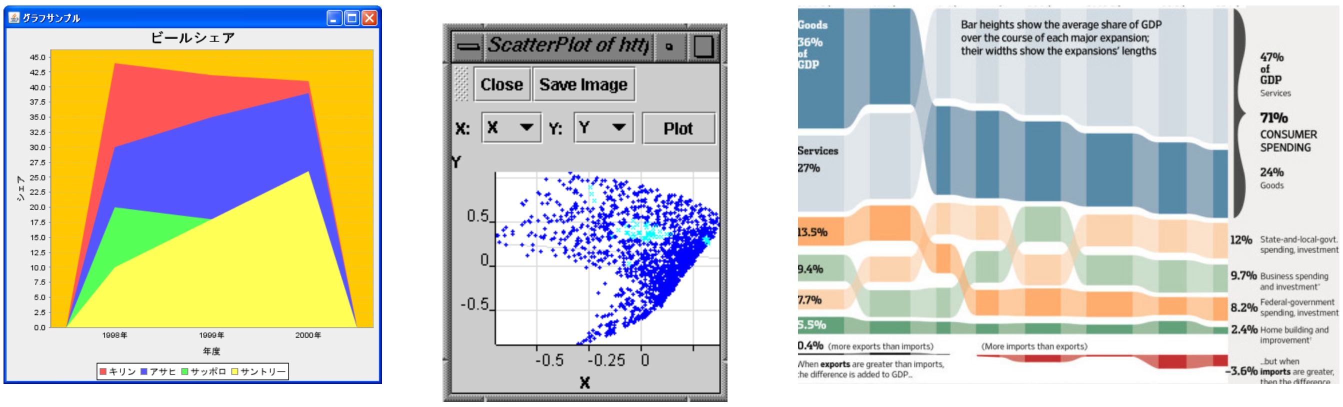


Figure 10: Examples of data used to train the mark classifier

6 Future Work

The specified limitations lead to some future work, first of all it would be interesting to see if each step could be improved further. For mark classification it would be interesting to see different net architectures, or to automatically create modern charts, with random data and train the classifier on this charts. Text localization could be improved, by making the heatmap more accurate, or by implementing a rule which sets the entire field between the axis to 0 in the heatmap, such that the plotting symbols get detected less. Text role classification could be improved, by looking at the training data and adding some new samples, which were actually detected by the text localisation method, this could maybe solve the problem, that the test data has a way higher accuracy than the real world examples. Another way to solve this problem would be to augment the existing data, by adding or subtracting small random values from the boxes.

Also there could be more metrics implemented to evaluate each step, for example Poco and Heer [11] evaluated their classifier against other popular classifiers like ReVision [14] or ChartSense [3].

7 Conclusion

In this work, an approach was introduced to reverse engineer images of visualizations, applying different neural networks and existing algorithms to the images, which could help researchers in analyzing human-computer interactions, just from screenshots or videos of a person performing tasks. Our approach consists of the following steps, first we localize the text in an image, then we classify the role of the found text, next we classify the mark type of the visualization and then we try to generate the chart specification.

With following these steps we were able to correctly generate the specifications of some visualizations, but most of the time our method made mistakes in one of the steps, which then propagates through the entire method. But even when this happens, the generated specifications still have some true parts.

In conclusion, the fact that the method works for specific visualizations and has some very predictable mistakes shows, that the there is potential for this method to be used in the future to extract chart specifications from chat images, but there is still some work to be done to make the results more accurate and reliable. We hope that our work inspires scientists to do more research in the field of reverse engineering visualizations from bitmap images.

References

[1] Zhe Chen, Michael Cafarella, and Eytan Adar. Diagramflyer: A search engine for data-driven diagrams. New York, NY, USA, 2015. Association for Computing Machinery.

[2] Sagnik Ray Choudhury, Shuting Wang, and C. Lee. Giles. Scalable algorithms for scholarly figure mining and semantics. New York, NY, USA, jun 2016. Association for Computing Machinery.

[3] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong-in Hwang, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. Chartsense: Interactive data extraction from chart images. New York, NY, USA, 2017. Association for Computing Machinery.

[4] Saurabh Kataria, William Browuer, Prasenjit Mitra, and C. Giles. Automatic extraction of data points and text blocks from 2-dimensional plots in digital documents. volume 2, pages 1169–1174, 01 2008.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[6] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, 2 1956.

[7] Ales Mishchenko and Natalia Vassilieva. Model-based chart image classification. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Song Wang, Kim Kyungnam, Bedrich Benes, Kenneth Moreland, Christoph Borst, Stephen DiVerdi, Chiang Yi-Jen, and Jiang Ming, editors, *Advances in Visual Computing*, pages 476–485, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[8] Dominik Moritz. Text detection in screen images with a convolutional neural network. *The Journal of Open Source Software*, 2(15):235, July 2017.

[9] Nobuyuki Otsu. A threshold selection method from gray-level histograms. page 5. IEEE Transactions on Systems, Man and Cybernetics, January 1979.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[11] Jorge Poco and Jeffrey Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. page 11, Washington, January 2017. EuroVis.

[12] V. Shiv Naga Prasad, Behjat Siddiquie, Jennifer Golbeck, and Larry S. Davis. Classifying computer generated charts. In *2007 International Workshop on Content-Based Multimedia Indexing*, pages 85–92, 2007.

[13] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[14] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. Revision: Automated classification, analysis and redesign of chart images. New York, NY, USA, 10 2011. Association for Computing Machinery.

[15] Gaurav Sharma, Wencheng Wu, and Edul Dalal. The ciede2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30:21 – 30, 02 2005.

[16] R. Smith. An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633, 2007.

[17] Binbin Tang, Xiao Liu, Jie Lei, Mingli Song, Dapeng Tao, Shuifa Sun, and Fangmin Dong. Deepchart: Combining deep convolutional networks and deep belief networks in chart classification. *Signal Processing*, 124:156–161, 2016. Big Data Meets Multimedia Analytics.

[18] Binbin Tang, Xiao Liu, Jie Lei, Mingli Song, Dapeng Tao, Shuifa Sun, and Fangmin Dong. Deepchart: Combining deep convolutional networks and deep belief networks in chart classification. *Signal Process.*, 124:156–161, 2016.

[19] Gongming Wang, Junfei Qiao, Xiaoli Li, Lei Wang, and Xiaolong Qian. Improved classification with semi-supervised deep belief network. 11 2017.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Bachelorarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

03.11.2022, Linz

Ort, Datum

[illegible]

Unterschrift