Heraldic Symbol Detection

Author: Stefan Schwanecke
Program code: [tehNewton/heraldicSymbolDetection]

Task

The task was to automatically detect heraldic symbols from images and also localize the textual annotations, belonging to these heraldic symbols and digitize them. The task was divided into two parts: An object detection task, and an optical character recognition task.

Object Detection

Object detection is a problem, that can be tackled with the help of convolutional neural networks (CNNs). For this problem, a faster RCNN network with a resnet-50 backbone was chosen as the faster inference times of single shot detection architectures are not required for this task. To exploit transfer learning, a model, which was pre-trained on the COCO dataset, was fine-tuned.

To be able to fine-tune the model, a set of 77 images was annotated with bounding boxes of heraldic symbols and their class labels comprising a total of 728 symbols and 970 text boxes, with between 1 and 16 symbol boxes and between 1 and 19 text boxes for each image. An annotated version of an image can be seen in the following figure, with purple boxes showing symbols and cyan boxes indicating text.

For the image pre processing, two transformations were made: As the images of the dataset are photographed pages from a book, a noticeable variance in brightness between different images is expected. To prevent the model from overfitting the training set too much in regards of image brightness, the training and validation images were augmented with a random brightness contrast transformation. In addition to that, another transformation was performed afterwards, with the goal of resizing the images to a uniform size, as it is required by the chosen architecture. The resolution of the resized images was selected to be 1024x1024, as it is a decent compromise between retaining the information of the original images, which were around 3000x2000 pixel in size, and reducing the required GRAM for storing them.

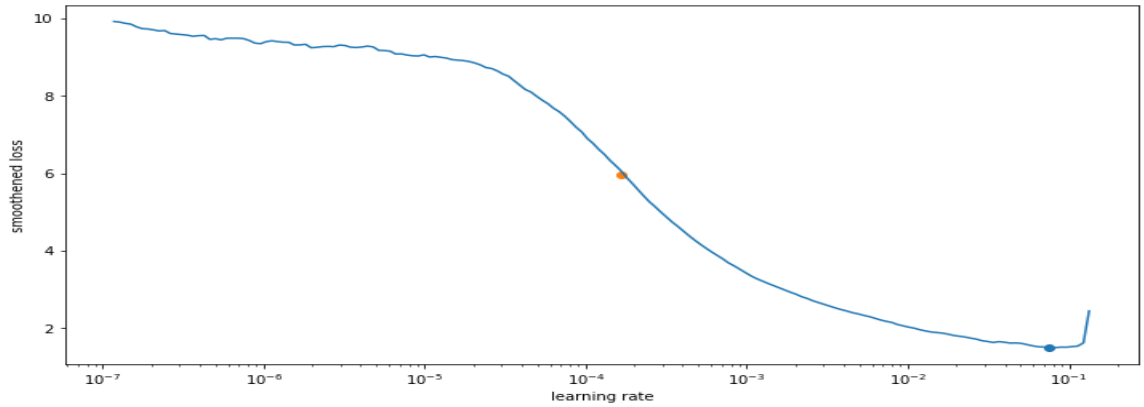
There were two different approaches tried for fine-tuning the pre-trained faster-rcnn model: The

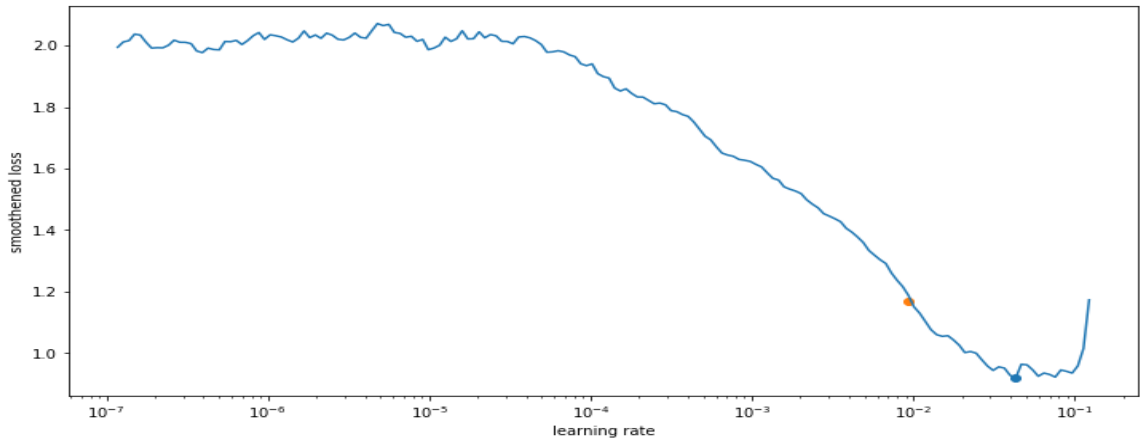
first approach was to tune the model for recognizing bounding boxes of symbols and text separately and deriving a merged bounding box afterwards. The second approach was to train the model to find the already merged bounding boxes of symbols and associated text.
As a result, two different datasets were created. The first containing annotated images with separated text and symbol boxes, the second one containing images, annotated with merged bounding boxes. For that purpose a function for merging bounding boxes was implemented.
The function iterates all annotated bounding boxes and attempts to find the closest text box within a specific distance threshold for each symbol box. If a suitable textbox is discovered, the symbol box and the text box are merged by calculating a bounding box that includes both boxes.

For both approaches, the training/validation split was done by selecting a random set of images for validation. The validation set included approximately 20 percent of the total annotated images with the rest of the images comprising the training set.

For each training method, an instance of a pre-trained faster-rcnn model was created, with a slightly adjusted head comprising two fully connected linear layers that output bounding boxes and class probabilities for three (text, symbol, background) and two (mergedBB, background) classes respectively. For the optimization of model parameters, a simple stochastic gradient descent is used as it outperformed typical optimization algorithms like adam during some test runs. For the learning rate scheduler, a cosine annealing approach with warm restarts was chosen as it can help in preventing the model from getting stuck at local minima. The initial interval was selected at an epoch length of one, with a multiplication factor of two.

To find a suitable initial learning rate, a separate training session was executed. This training session monitored the development of the loss values after each mini batch. In addition, the learning rate got multiplied by a factor between one and two after each mini batch. This lead to the following loss vs learning rate plots (first separated approach, second merged approach):





The yellow dots indicate the learning rate for which the steepest descent in loss was detected, the

blue dot shows the learning rate for which the minimum loss was detected. This lead to the selection of a learning rate of around 6*10^-4 for the first model and around 4*10^-3 for the second one.

To prevent under and overfitting, the number of epochs for training was not fixed to a set value. Instead, after every epoch, the mean average precision (MAP) for the validation set was logged. If the MAP did not improve after ten epochs or decreased by more than ten percent, compared to the best result, the training was stopped.

For separated bounding boxes, the model has achieved the best results after 48 epochs of training. For merged bounding boxes, the best result on the validation set was achieved after 45 epochs.

For the model, trained to detect merged bounding boxes, the distinct symbols from an unknown image can be recovered by scaling the image size to 1024x1024 pixels, performing inference on the model and scaling the predicted bounding boxes back to the original image size. For the model that detects text and symbol boxes separately, some additional post processing steps need to be performed after inference: Firstly, after the model prediction is recovered, candidate bounding boxes are selected based on a confidence threshold. Secondly, those candidate bounding boxes are inspected for possible duplicate boxes, that refer to the same object as another candidate box. If such a conflict is detected, the box, which the model is less confident in, gets dropped. Finally, after the elimination of duplicate boxes, symbol and text boxes are merged. This process is shown in the following image:



Evaluation

The performance of both trained models on the validation sets is shown in the following images:

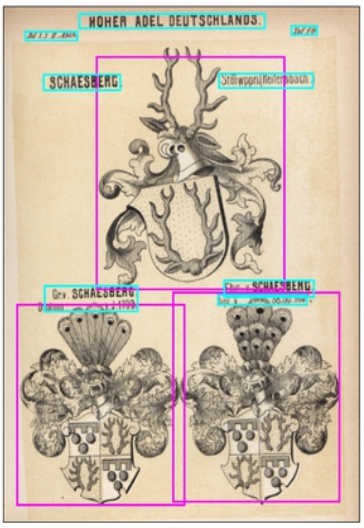
Separated:

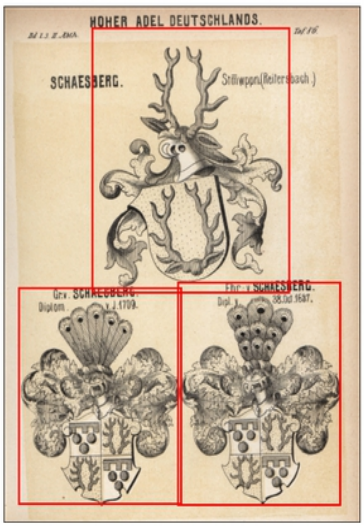
```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.751
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.964
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.846
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.614
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.614
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.685
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.090
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.650
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.805
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.644
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.705
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.738
```

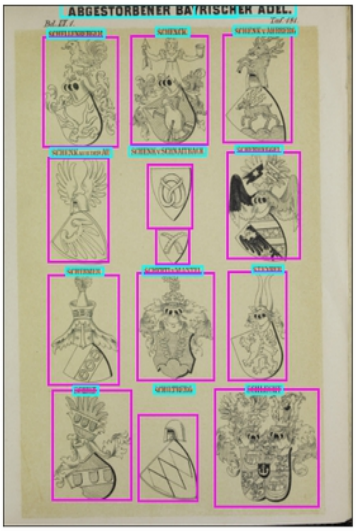
Merged:

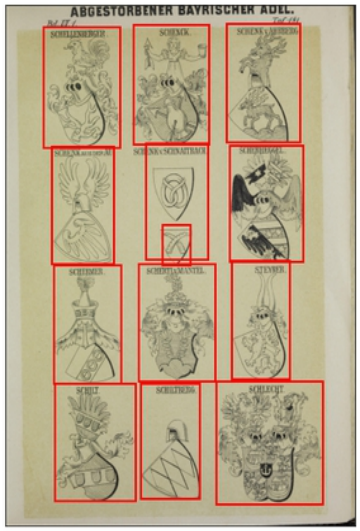
```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.890
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.959
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.890
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.091
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.793
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.915
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.915
```

The higher scores in precision and recall for the merged model are expected as the task is reduced to only regressing bounding boxes, instead of a box regression and a classification task for the separated model.
In addition to the precision and recall metrics for the validation set, the performance of the models can also be evaluated by looking at their predictions for some previously unknown test images. The following images show some good and bad performances of the trained models, with the separated models predictions depicted on the left and the merged models predictions depicted on the right side.









These test images show, that the separated model generally does a good job in recognizing symbols and text boxes, but sometimes fails to separate neighbouring text boxes or creates multiple small text boxes for each word.
Additionally, sometimes crowns get mistakenly detected as text.
For the detection of heraldic symbols, the separated model does perform better, than for the recognition of text. The only reoccurring issue in symbol recovery is the failure to fully recognize symbols that are divided into multiple smaller parts.
The merged bounding box model does find a suitable merged bounding box for most symbol text combinations. One major issue of this model is its high weight on the bounding boxes of symbols over the bounding boxes of text. This might be due to the fact that usually, the symbol makes up the larger part of the merged bounding box. As a result, only predicting the bounding box of the symbol and disregarding the text can lead to good estimations for merged bounding boxes in most cases.This can cause the model to disregard text that is horizontally larger than the symbol.

Optical Character Recognition

The second part of the task was to automatically extract text from the bounding boxes, that were detected in the previous, object detection, step. Optical character recognition (OCR) problems can be tackled with the help of long short-term memory (LSTM) networks. For this OCR task, the Tesseract OCR engine was chosen as it offers numerous pre-trained models and can easily be

integrated to a python project through the pytesseract library.

The first step in the character recovery process was image pre-processing, which was realized with the use of the opencv library. The pre-processing steps were the following: The first step was a greyscale conversion of all target images. Afterwards, all previously discovered text boxes were recovered from the greyscale image and relocated to newly created, separate images. These images were further pipelined through a binarization, resizing, a deskewing and a noise removal step.
For the binarization step, an adaptive gaussian threshold was selected as binarization with fixed thresholds achieved bad results due to the lighting variations encountered in some of the images.
The resizing step was designed to ensure a minimal height and width for each image. The minimal height was set to 300 pixel, such that characters from textboxes, that comprise multiple lines, are large enough to show characteristic structures and edges. The minimal width was set to 600 pixel to ensure that the OCR engine is able to distinguish between the individual characters in the case of short words with little space between characters. In addition to the resizing, a small, white border was added to each image to enable the OCR engine in detecting character borders that are located close to the image border.
The deskewing step was designed to rotate each image by a small angle in order to align lines of text to the x-axis. This angle was calculated by finding a rotated version of the image, such that a minimal bounding box that contains all foreground pixel claims the least space possible.
The noise removal was realized by the application of a sequence of transformations like erosion, dilation and blur.
The effects of the image pre-processing transformations can be seen on the following example image:
after greyscaling:

after all pre-processing transformations:

After the image pre-processing, a character sequence for each discovered text box can be discovered by invoking the Tesseract OCR engine. The language data, that was supplied to Tesseract was a pre-trained language model for the German language, available on the Tesseract github page. The selected model belongs to the tessdata-best set, that consists of models that prioritize accuracy over inference speed.
To prevent the incorrect recognition of exotic special characters, a character whitelist was also created. For the domain of historic documents, the character set was reduced to alphabetical letters, digits, separators, and German umlauts. This limitation eliminates the possibility of discovering exotic characters by mistake, at the cost of making it impossible to recover some non-German texts correctly.
In addition to the set of characters, Tesseract can also be configured to assume certain text layouts like uniform blocks, or lines. For the task of detecting textual annotations for heraldic symbols, four layout assumptions achieved good results on some images:
The assumptions of a single uniform block of text, a single line of text, a single word, or a raw line of text.
As the textual annotations encountered in the dataset often fit into one or more, but not all of these categories, the OCR engine was invoked multiple times for the same image with different assumptions on the text layout. Afterwards the result with the highest confidence was selected as the

final result of the OCR process.
This technique was extended to multiple invokations with differently pre-processed images for each layout assumption, as the dataset comprises vastly different qualities of texts. This resulted in improvements of OCR performances after the application of dilation transformations for some images, and improvements after the application of erosion transformations for other images. As a consequence, for each image a total of twelve invokations were performed, which improved the OCR performance at the cost of speed.

To further remove noise from the predicted texts, a final post-processing step was created.
This post-processing step removes words that only contain non-letter and non-digit characters as they are very likely noise. Additionally long sequences of dots and commas were reduced to just one instance of that character and special characters located at the start and end of words were removed.

Evaluation

The performance of the OCR pipeline was evaluated on a set of test images, previously unknown to the object detection and OCR models. Some results are depicted in the following images:

0: WAYNA, I.

1: WAYNA, I.

2: WEBER, I.

3: Taf 255.

4: WEBER,L

5: Ba. IV. 4.T.

6: NIEDEROESTERR. STAND. ADEL.

0: HERSPRUCKER.

1: RYTZENWINKLER II.

2: HILLEBRANDES

3: HINTERHOLTZER

4: HORERSTORFER.

5: RYTZENWINKLER.I.

6: OBEROESTERREIGCIHSCHER ADEL.

7: HILDPRANDT.

8: Taf 37

9: BA. MV.

10: HINGENAU.

11: HOCHENHAUSEN

12:)FMANNv. GRUNBUCHEL L. HOF MANN z GRUNBUCHEL, II. HOF

13: HOFMANN z.GRUNBUCHEL II.

These images show, that the OCR is able to extract straight line text which can clearly be separated from heraldic symbols with little to no errors. Some issues do occur at the borders of text boxes,

especially if the object detection model is not detecting them precisely. Additionally the detection of Roman numerals is error prone and scenarios in which the text boxes also contain parts of heraldic symbols do lead to bad results for the OCR.
To further evaluate the performance of the OCR pipeline, a set of 27 fresh test images was annotated with texts. For these images the character error rates (CER), and word error rates (WER) were calculated. The results are shown in the following table:

	OCR from perfect boxes	OCR from predicted boxes
CER	11.6%	19.5%
WER	64.4%	81.5%
WER (tolerance of 1 error)	31.8%	44.2%
WER (no special characters)	28.0%	40.1%

The resulting values for CER support the observations on the test images. The OCR was able to recover most of the depicted characters, when supplied with the perfect bounding boxes. Some single caracter mistakes and difficulties in recognizing Roman numerals added up to an error rate of 11.6%. This error rate almost doubled when predicted bounding boxes are supplied to the OCR, in stead of the perfect, annotated bounding boxes.
Further issues of the OCR can be obeserved for the worst performing images of the set:

The bad performances of OCR in these images can be explained by two common characteristics: The text in both images is written in an uncommen font or even handwritten in some instances. Additionally the text is tough to seperate from the heraldic symbols by a simple bounding box.

When looking at the raw WER, the OCR performance is comparatively bad, with a 64.4% WER when supplied with perfect bounding boxes and a 81.5% WER when supplied with regressed bounding boxes. These bad results can be explained by looking at the patterns in faulty recoveries

that caused these bad results. A large portion of the errors were either single character confusions such as 'E' and 'F', or mistakes revolving around special characters like dots and commas. When removing special characters from consideration, the WER gets more than halved from its original value. Similarly, when considering words as equal if they only differ in one character, the WER is also approximately halved. In summary, while the WER shows that the OCR is far from perfect, the issues are not as severe as the number suggests.

Future Work

The evaluation shows that the designed pipeline for the detection of heraldic symbols and the recovery of text from images achieves decent results for most images of heraldic symbols from the dataset. While these results are a decent starting point, there are certainly improvements, that can be made:

For the object detection task, only a few architectures and pre-trained models were considered. As many developments in the field of object detection have been made recently, there is lekely a neural network architecture, that can achieve better results than the chosen faster-rcnn architecture. In addition to that, the weaknesses of not detecting heraldic symbols that comprise multiple parts, and falsely detecting crowns as text could be addressed by training this model on more images, that contain these problematic structures.
Another approach to improve object detection performance is the consideration of image pre-processing steps before passing images to the model.
Furthermore, the application of different augmentations to the training set might improve the trained models performance on unseen, "difficult" input images.

For the OCR task, the Tesseract OCR engine was used with pre-trained German language model data. This is certainly not optimal as many textual annotations to heraldic symbols do contain structures that are not commonly encountered in modern German language. Fine-tuning this model with the help of a representative dataset would certainly improve OCR performanec in recognizing structures such as roman numerals. Fine-tuning the language model might also help in improving the OCR performance on textual annotations that are handwritten or displayed in uncommon, old fonts, as well as extending the language model to recognize heraldic symbol annotations that do not originate from regions where German is the main language.
In addition to the language model, Tesseract OCR also can also utilize a word list and a pattern list to aid in the OCR task. Supplying a word list with commonly encountered word structures like "Frhr.", "Adel", "Grf." or patterns of roman numerals can aid Tesseract in finding those structures more consistently.
During the design of the OCR pipeline, many trade-offs between OCR performance and inference time have been made in favour of OCR performance. This lead to a comparatively slow inference, that makes the OCR pipeline impractical when tasked with the recovery of text from a large set of images. Any improvements to the speed of the OCR pipeline would increase its practicality noticeably. The majority of the time requirements are caused by the twelve different invokations to the Tesseract OCR with differing combinations of pre processed images and layout assumption settings. Reducing this number by discovering a set of configurations with less combinations that does not sacrifice OCR performance too much could achieve a large inference time reduction.
In the current setup of the OCR pipeline, after the twelve predictions for each text box are gathered, the text predictions, that Tesseract is most confident in, gets selected. A more sophisticated approach than the simple selection of the highest confidence might achieve better results for OCR performance. One possible approach to that might be the consideration of similarities between the different predicted text fragments in addition to their confidence scores.