SHEL
Single-German playing card detection using YOLO

Matthias Moik
11810738

November 2, 2023

1 Introduction

Playing card detection is a specific topic in the area of computer vision and there are already working models out there that can correctly identify playing cards in real time. This can be used to track a game for spectators, like it is stated in the motivation for this paper about dataset generation for the game Duplicate bridge, to help players train card counting, or to keep track of the scores for each player. The latter application is also interesting for a game played in Vorarlberg, Austria called "*Jass*". In the end of each round, every player has to calculate their score by adding up the different values of the cards they won during the round. The automation of this will be the problem that we try to solve with this project.

1.1 Problem

In more detail, the problem we are trying to solve is keeping score for each player to avoid tedious counting by hand and eliminating errors that occur when counting by hand due to some special rules in the point distribution. By capturing live video of the game, we have images we can feed to a neural network to detect the cards played.

As we already mentioned above, models that can do this exist, but unlike the cards used in these examples, Jassa uses *Single German playing cards*, that are not as commonly used in other places. Because of this we were not able to find any dataset containing this type of cards, and no model that can identify these cards, which is the core problem we try to solve within this project.

1.2 Approach for solution

Why is deep learning a solution?
First we want to note that this is a classic problem in the area of computer vision. As input we get images and the desired output should contain the objects, in our case playing cards, that are on the image. Doing this without deep learning an another approach is fairly difficult, because it requires the ability to recognize complex patterns and features

within the images, which traditional algorithms often struggle to achieve.

What is our solution?
The main part of this project will be to generate a suitable dataset for neural network training for Single German playing cards. This dataset will then be used to fine-tune a pre-trained model, that is able to detect multiple objects in an image very fast. In the end we should be able to place multiple playing cards under a camera and let the program detect the position using bounding boxes and also the suit and value of the card. How the dataset is build and the models are trained will be explained in the next section.

To wrap up the whole project, this playing card detection model will be integrated into an application, that is able to track a whole game and calculate the scores for each player online, during the ongoing game.

2 Results

This section revolves around the implementation of the different parts of this project, as well as a short summary of the model evaluation.

2.1 Dataset generation

To generate the dataset, we took pictures of each playing card with our improvised setup, seen in Figure 1.

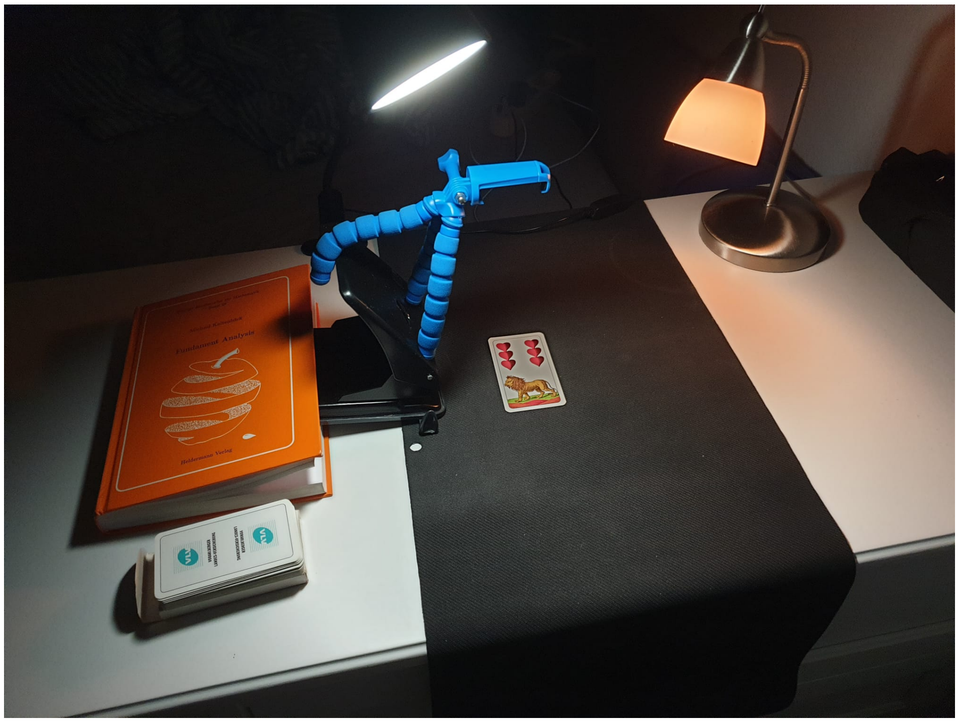

Figure 1: Our setup to take good pictures of the playing cards

These photos were then cut out automatically using functions from the package `OpenCV`. The cropped playing cards were then placed randomly on a background chosen from DTD. There are a few parameters that can be set for the dataset generation:

- number of images to generate
- maximal number of cards per image
- minimal size of the cards
- maximal size of the cards
- allow cards to overlap

Other augmentation like position, rotation and brightness of the placed cards are also chosen randomly. In figure 2 is an example image generated with our code.

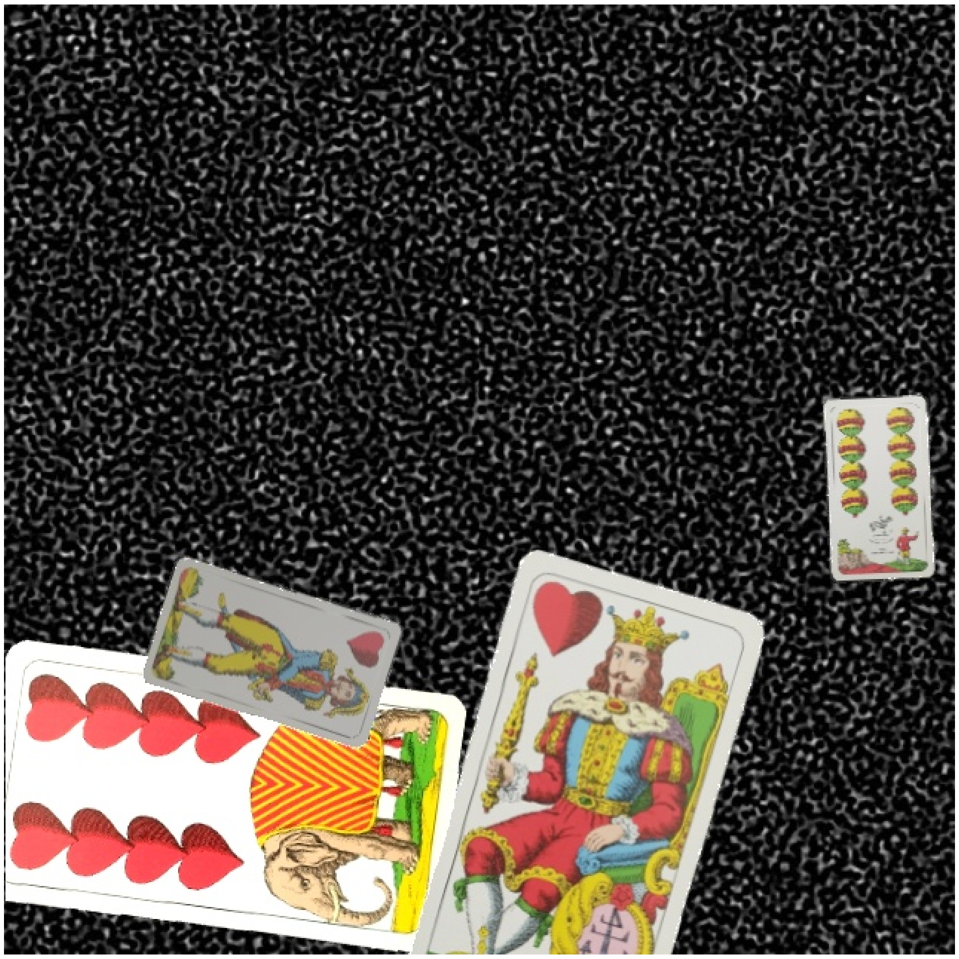

Figure 2: One example image from a generated dataset

2.2 Model evaluation

In this section the results of our trained models are discussed. We varied the dataset size, as well as the boolean variable, which decides if overlapping cards are allowed, to generate different datasets. The training of a model takes quite some time, which is why we stuck only to these two variables, and fixed the others to the following values:

- max_number_of_cards_per_image = 4
- min_size = 0.2
- max_size = 0.7
- seed = 42

We achieved the following values for our chosen error metrics:

Dataset size	Overlapping	Epochs	Precision	Recall
6,000	True	10	0.97032	0.9245
6,000	False	10	0.90016	0.89837
60,000	True	10	**0.99256**	**0.97052**
60,000	False	10	0.89694	0.89706

Table 1: Results of models predicting on the test set

We can see that, against our expectations, the dataset with overlapping cards achieves higher values for precision and recall. Training on more data lead to better results for the overlapping case, while the plateau for non-overlapping cards is already reached with the 6000 image dataset. This can also be seen in the plots in figure 3.

Additionally, we trained on a 60000 image dataset for 20 epochs, with varied card sizes (min_size = 0,1 max_size = 0,8). This lead to a precision of 0.99491 and recall of 0.9459. This is the best result regarding precision, but not for recall. We still ended up choosing this as the model for deployment, because the lower recall can be explained with the larger range of card-sizes, because they lead to more completely covered cards than with the other parameters, but leaves us with a broader spectrum of card sizes, which also has influence on live predictions.

Training time of the models was around 1 hour (2 hours for 20 epochs), using the V100 GPU, that is provided when using Google Colab Pro.

2.3 Application

The application was implemented using one of the mentioned possibilities in the assignment, Streamlit. This was pretty straight forward, although the connecting the camera was a bit tricky, which is also the reason we could not deploy the app on Streamlit itself. Instead we used Docker to make it as simple as possible to gain access to the demo application. We also implemented CI, that builds a new image on each push tho the repositiory that can then be downloaded. How to use the application is described in the `README.md` in the repository.

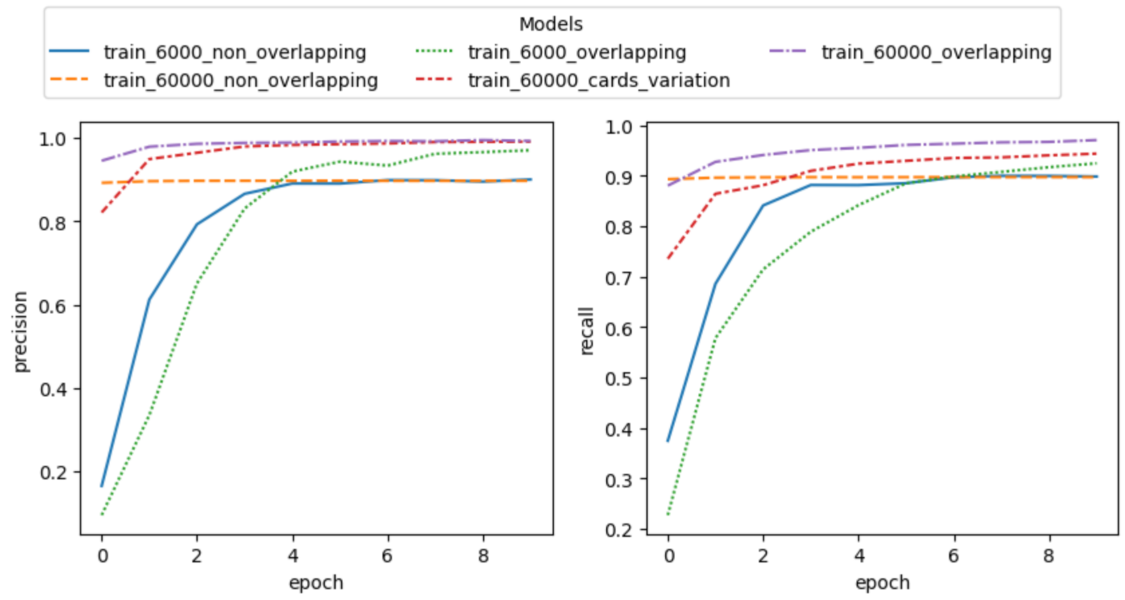

Figure 3: Comparison of precision and recall of models trained on different datasets

3 Gained insights

The main takeaways for us from this project were that creating a clean pipeline to train and compare models is a lot of work when it should be clean, especially when working with online resources.

Otherwise, using the right dataset is crucial for a good performing model. We started out with detecting overlapping cards as a bonus, and initially sticking to non-overlapping examples. But as the results show, the overlapping dataset leads to even better results. Trying different settings and parameters for a dataset is as important as the model parameters themselves.

Regarding the application, Streamlit is a great way to build a small demo application without any previous knowledge of web-applications. Using Docker to handle dependencies and run the application is a great way to make it available for anyone. The automatic build and publish of the image on each push gave us great insights in what can be done when taking time to setup a CI, which admittedly was done a bit too late in this project.

4 Time management

The time management for this project can be seen in table 2. A dash symbolizes that this part was not in the original time management plan.

As we can see, the original time management was not completely wrong, but some parts were greatly underestimated or just forgotten. We ended up spending more time on the

Task	Estimated time	Spent time
Find project idea	1	2
Research approaches	5	3
Implement dataset generation	30	30
Train and evaluate	15	35
Simple live and unittests	5	3
Build application	18	15
Release app with docker	-	5
Write report	-	3
Prepare video	-	2

Table 2: Time management of the project

project as intended, but learned some valuable things concerning docker and releasing something using Github.

The most underestimated parts was building a training pipeline. Not only the impementation, but also training the models was underestimated. Training is not really time spent, because other tasks could be completed in the meantime, but waiting for hours to get a trained model disrupted the workflow, which was not great.

The hardest thing when building the training pipeline was actually the fact that it was on Google Colab and not locally like the rest. This meant we had to come up with a somewhat clean way to provide the needed dependencies, datasets and stored models. Doing this using Google Drive is a feasible solution in our opinion, but it took a while to get it in a simple, userfriendly way.

5 Conclusion

In conclusion, this project was very educational and insightful regarding the workflow for a project in the area of deep learning. It was fun to work on this project and I would do it again. But in hindsight there are some things I would do different, like set clear goals regarding deployment and gather more information on how to get there. I was a bit stuck and needed to re-implement some parts because in the end it didn't work out in the way I wanted while building the application.

Having an own GPU would also make things easier for training. Working locally with one part of the project and on Google Colab with the other was not that handy, especially when loading datasets. But I think the solution to load these from my Google Drive is still quite clean and easy to use.

All in all, I am happy with the outcome of this project and I learned some valuable lessons on the way.