task 1

- **Question:** What network structure did you choose and why?
  - **Answer:** We chose a sequential convolutional neural network with a fully connected layer because it's often used for image classification. In the lecture, such a structure was also used, which is another reason for why we chose this structure.
- **Question:** What hyperparameters and loss function did you use, and why?
  - **Answer:** We used binary cross-entropy for the loss function, as we had to deal with a binary classification problem with a balanced amount of both classes. As such, this loss function is applicable. Adam was used as an optimizer because most tutorials used it, and it was also recommended in the lecture. Additionally, it helps reduce the vanishing gradient problem and gives good results with little hyperparameter tuning according to this link. The learning rate was not explicitly specified, since the model converged in a reasonable amount of time (according to this, learning rate is 0.001 by default when using adam). Because the model had a particularly bad performance on the pipeline's `test_front_2.png`-image, the epochs were set to 8 epochs and regularization was introduced. The model performed the best on that image, when the accuracies were around 0.9 - leading us to believe the model was overfitting before.
- **Question:** Provide a (tensorflow model) summary of your network (layers, parameters, etc.)
  - **Answer:** see *"task_1_classification_model_summary.txt"*-file
- **Question:** Report your training and test accuracies
  - **Answer:**
    - training accuracy: 0.9000
    - validation accuracy: 0.8938
    - test accuracy: 0.93
- **Question:** Provided the required data, what would you have to change in your model to classify not only the presence of German ID cards, but also ID Cards of other countries?
  - **Answer:** Given a properly annotated dataset, we would need to change the loss function from binary cross-entropy to (sparse) categorical cross entropy and change the last layer's (output layer's) activation function from sigmoid to softmax. We have to make these changes, as we now don't have a binary classification problem anymore but a multi-class classification problem.

Bonus

- **Question:** try to optimize the robustness of your model with data augmentation
  - **Answer:** We did try to perform small amounts of data augmentation for improving the model's performance on the `test_front_2.png`-image, whose classification was awful - as already explained before. On that image, the color of the ID and the table underneath were quite similar. As such, We tried to randomize the brightness of the training and validation images between 1.0 and 1.5 in the hopes that this would result in some of the images becoming similar to the problematic image. Aside from that, we also changed the interpolation method to lanczos in hopes of preserving more information when downsampling the images.
  - **Question:** does it improve the classification accuracies? If yes, why and for which cases, if no, why?
    - **Answer:** Increasing the brightness did not have a significant impact on the problematic image, but it also didn't really change the classification accuracies in a meaningful way. The reason for that may be that there were no images with such brightness in the test set. Changing the interpolation method to lanczos also seemed to have no real impact on the classification accuracies. That may be due to the fact, that the classification is such a simple problem, that finder details don't really play a big role.

task 2

- **Question:** What network structure did you choose and why?
  - **Answer:** We chose a U-Net for our network architecture. It's the method proposed in the lecture, and it's applicable to segmentation problems. Aside from it being a standard method, using its encoding-decoding architecture, it is possible to store spatial information from the different layers so that the information we want to extract from the input can be properly carried through bottleneck and mapped onto the input pixel dimensions.
- **Question:** What hyperparameters and loss function did you use, and why?
  - **Answer:** We used binary focal cross-entropy for the loss function, as we had to deal with a binary segmentation problem (i.e. "foreground vs. background") with an unbalanced amount of both classes. The binary focal cross-entropy more heavily weights the class that occurs less frequently, which is in our case the ID card. Adam was used as an optimizer without the learning rate being specified, while the reasoning for that is the same as in task 1. The model was trained for 100 epochs and the weights of the model were saved according to the intersection-over-union with respect to the foreground class. With that, we wanted to ensure that the model chooses its weights such that the correct prediction for foreground-pixels in particular is increased.

- **Question:** Provide a (tensorflow model) summary of your network (layers, parameters, etc.)
  - **Answer:** see *"task_2_segmentation_model_summary.txt"*-file
- **Question:** Report your training and test accuracies/losses
  - **Answer:**
    - training accuracy: 0.9939
    - validation accuracy: 0.9933
    - test accuracy: 0.9931
    - training intersection-over-union: 0.9777
    - validation intersection-over-union: 0.9704
    - test intersection-over-union: 0.9753
- **Question:** Provided the required data, what would you have to change in your model to segment multiple ID cards per image (multiple solution strategies)
  - **Answer:** To segment multiple ID cards per image, one possibility is to provide images with corresponding ground truth masks that contain multiple ID cards to train the model with. Afterward, it may be possible to use OpenCV to find not just the one biggest contour in the image to crop it, but rather define a minimum area the contour has to cover for it to be seen as an ID card. Another method would be to use a Mask R-CNN architecture (see here), that makes it possible to classify the different IDs not just into foreground or background, but rather into $n + 1$ classes for n different types of IDs and the background.

task 3

- **Question:** How did you implement your method?
  - **Answer:** We decided to use the probabilistic hough transform. That means, we first find out the angle relative to the horizontal line and then rotate it by the smaller angle (i.e. we assume the images are "generally" oriented upright). For finding out the angle, the images are preprocessed via opening, performing erosion and then dilation by the same kernel size, closing the image's bright spaces. Then, canny edge detection is used to reduce the image to only its edges. Afterward, the probabilistic hough transform is used to find the lines in the image. Lastly, we calculate the angles between the horizontal line and all the lines we found. The dominant angle is determined by the angle that occurs most often. With the angle found, we can then rotate the image by that angle.
- **Question:** What are advantages/disadvantages you notice about your method?
  - **Answer:** Our answer relates heavily to the lecture. Hough transform doesn't require an already properly rotated image of what we want to de-skew as a template, like SIFT does. But, since SIFT uses keypoints to rotate an image to the position of the

template, it should (in theory) always rotate the image properly. Hough transform does not guarantee that. The image may be upright or turned on its head, but we can't really make sure just with the hough transform itself.

- **Question:** Are there any skew angles that are (un)favourable for your method?
  - **Answer:** In our case, there haven't seen any skew angles that were particularly (un)favourable.
- **Question:** How would you implement upside-down detection without SIFT?
  - **Answer:** There may be one possible solution we thought of, which involves increasing the kernel size used for the opening. When increasing it to 75×75, we may be able to tell apart the front and back of ID cards and their orientation. With a kernel size this big, the resulting images will have big keypoints in certain positions - the front will have the face of the ID card's owner as a big blob on the left side of the image and the back will have a long rectangle with two blobs sticking out on top. Maybe it is possible to use OpenCV to localize those as contours and thus decide how the image is oriented.
- **Question:** What are options to segment and de-skew ID cards in one step?
  - **Answer:** Possibly, you could train an end-to-end network similar to the Fast RCNN architecture with two heads, where one head predicts the segmentation and the other head predicts the skew angle. Another possibility may be to train two networks for each task and assemble them. Lastly, one possible approach could be found in the following paper https://link.springer.com/article/10.1007/s00371-020-01952-z, where cropping and de-skewing seems to be applied in one step (judging from reading only the abstract).

Bonus

- **Question:** Improve your method with image pre-processing
  - **Answer:** We used opening with a kernel size of 25 pixels to pre-process the images. Just like it was shown in the lecture, this caused the lines of the ID cards to become more distinct. Afterward, we used canny edge detection to end up with the edges of the image, which are then used to perform the hough transform.

task 4

- **Question:** What network structure did you choose and why?
  - **Answer:** We chose a U-Net for our network architecture. It's the neural method proposed in the lecture for cleaning and binarization. Since this is again a segmentation problem, the same argumentation as in task 2 holds here as well. The U-Net architecture makes it possible to store spatial information throughout the whole structure, which is important for segmentation.

- **Question:** What hyperparameters and loss function did you use, and why?
  - **Answer:** We used binary focal cross-entropy for the loss function, as we had to deal with a binary segmentation problem with an unbalanced amount of both classes, just as in task 2. In this task especially, we believed that heavier weighting of the class representing the text is even more important. Adam was used as an optimizer without the learning rate being specified with the same reasoning as in the previous tasks. The model was trained for 100 epochs and the weights of the model were saved according to the intersection-over-union with respect to the text. Again, we wanted to make sure that the weights that result in the best prediction for the text pixels are saved.
- **Question:** Provide a (tensorflow model) summary of your network (layers, parameters, etc.)
  - **Answer:** see *"task_4_binarization_model_summary.txt"*-file
- **Question:** Report your training and test accuracies
  - **Answer:**
    - training accuracy: 0.9517
    - validation accuracy: 0.9516
    - test accuracy: 0.9521
    - training intersection-over-union: 0.9668
    - validation intersection-over-union: 0.9677
    - test intersection-over-union: 0.9681
- **Question:** What is a fundamental issue with this approach that some text has to be removed, while other text is to be preserved?
  - **Answer:** The fundamental issue is that the CNN classifies each pixel, whether it's correct or not. Thus, it has to learn which text is the correct one to keep and which has to be removed without being able to understand the "meaning" for why we have to get rid or keep certain text.
    - **Question:** what could be a reason as to why the approach does work anyway?
    - **Answer:** Generally speaking, the input images that have to be cleaned and binarized don't have that much variance in terms of position of the relevant/irrelevant text. With that, the CNN works anyway because at the end of the day, there are only two different layouts (i.e. front and back) of the IDs. It may learn those complex spatial features for the two layouts and may learn different feature maps for the different layouts.
    - **Question:** what would be an approach that is more „aligned“ with how image-based CNNs work?
    - **Answer:** An approach more "aligned" with how image-based CNNs work could be to use two separate neural networks for the front and the back, coupled with a classification network that first classifies a given ID as either being the front or

the back. This way, each network is very specialized on either the front or the back and can maybe infer the correct text to be removed from the position of the text more easily.

task 5

Bonus

- **Question:** Compare the OCR output on an uncleaned ID card to that of a clean one in terms of precision and recall on correct words
  - for that, choose a handful of id cards and build your own ground truth text (cf. task 6)
- **Answer:** I used 5 images from task 3 as the base images, since I can be sure that there are no images turned on their heads here. For those 5 images, a JSON-file was created which looked exactly the same as the one for task 6. The results can be seen below. It shows that recall is actually pretty similar when comparing the uncleaned and cleaned images, but precision is very different. It turns out that in the uncleaned images, the optical character recognition finds significantly more words than in the cleaned images. This makes sense, since there is other text on the ID cards that we get rid of in the cleaning and binarization step. Furthermore, the OCR may interpret parts of the background as text as well, which could contribute to such a high number of words found in the image.
  - In conclusion, comparing recall and precision of uncleaned and cleaned images, we can say that recall is a bit worse on uncleaned images, while precision is much worse on uncleaned images.

```
--------------------
Uncleaned Images:
Total number of ground truth words: 113; Number of ground truth words found
in image: 77; Number of words found in image: 534
With that, the recall is: 0.681
With that, the precision is: 0.144
--------------------


--------------------
Cleaned Images:
Total number of ground truth words: 113; Number of ground truth words found
in image: 89; Number of words found in image: 121
With that, the recall is: 0.788
With that, the precision is: 0.736
--------------------
```

task 6

- **Question:** report your recall on correct words w.r.t. the test set
  - recall: `#words` in GT found / `#words` in GT
  - **Answer:** $\text{recall} = 54/65 = 0.831$
- **Question:** what did you have to change to plug individual components together?
  - **Answer:**
    To preserve as much information as possible, we decided to change tasks 2 and 3. In task 2, the segmentation was not applied to the resized image as before, but rather the original image. In task 3, while a gray scale image was used as input to determine the angle, the image to be rotated was changed to be the output of task 2. Task 1's model had to be retrained in order to properly classify all images of the test set.
    In terms of difficulties during the implementation, there were two things in particular we had to keep in mind, relating to the different components: (1) what shape does a component expect and (2) what pixel values does a component expect. When using any of the models (tasks 1, 2, & 4), it was important to keep to not forget the batch dimension - otherwise an error was thrown. A more tricky problem was dealing with the pixel values, because TensorFlow's models may not throw an error when the pixel values of input images aren't defined as float values between 0 and 1. Instead, the output would just be awful.
- **Question:** how does your end-to-end pipeline handle the sample ids?
  - **Answer:** When looking at the recall, it handles the sample IDs well, but not perfectly. Especially `test_front_2.png` made problems for the classification and still makes some problems for the segmentation, for instance.
- **Question:** what are problems, what worked directly, what do you notice?
  - **Answer:** One of the problems was the `test_front_2.png`-image, which the classification network very confidently classified as not being an ID. We had to apply regularization on the first task so it was classified correctly.

Bonus

- **Question:** what about precision (`#words` in GT found / `#words` found)?
  - **Answer:** $\text{precision} = 54/70 = 0.771$
  - **Question:** is it high/low? why?
    - **Answer:** Precision is lower than recall. The reason is likely that in task 4, not all text has been perfectly cleaned and thus the OCR either reads words that shouldn't be there or it cuts words that should be there into two words.
  - **Question:** would you prefer to measure recall over precision or vice versa?

- **Answer:** Recall tells us the ratio of the ground truth we found to the ground truth itself, while precision tells us the ratio of the ground truth words found to all the words we found. In that sense, recall gives us more-so an idea of how many words we correctly found, while precision gives us an idea of how many words we mistakenly found. A high recall gives a good indication whether the pipeline performs well, since it directly involves the real number of ground truth words. Precision could be misleading. As an example, if we only found 5 words and those just happened to be in the real ground truth words, we would have a precision of 1. Recall could also be misleading, if the model found a lot of words from the ground truth but also found a lot of words that weren't in the ground truth - in that case recall would be high, but precision would be low. We believe both measures in conjunction would give the best idea about the performance. If we had to choose, we would prefer recall because it involves the total number of ground truth words and may be a better measure for the success of the pipeline.