Resnet-101 Semantic Segmentation Convolutional Neural Network for Gate Detection in Drone Racing

L.J.A. Kloosterman, 4697006, Technical University of Delft
https://github.com/ljakloosterman/AFMAV_individual

Introduction

Artificial intelligence has experienced an accelerated development in the last years and is used in more and more fields. Also with drone racing, AI algorithms have shown to perform excellent and are becoming more popular. In this paper, the implementation of the Resnet-101 model as a semantic segmentation convolutional neural network is described. The first section provides a literature study which compares multiple convolutional neural networks. After this, the implementation of the algorithm is described and an explanation is given on the different Python files. In the third section, a description is given on the training of the model and some parameter tuning will be executed. Next, the computational effort is discussed by elaborating on the computation time of image processing. Finally, the visual results and the ROC curve to a test data set will be shown and a conclusion on the performance of the model is drawn.

1 Literature study

In this short literature study an overview will be given of some convolutional neural networks which are used for object detection. Unfortunately, in the field of autonomous micro air vehicles convolutional neural networks are not yet widely used because they are often computationally too heavy reducing flight time. On the other hand, computational power is getting is becoming cheaper and new convolutional neural network models are designed every day. In this section, three different convolutional neural network models for object detection are considered.
The first model is the *'Fast R-CNN'* [1] algorithm which feeds the input image to a convolutional neural network to generate a convolutional feature map, identifies the region of proposals and converts them into fixed size squares which are categorised. The Fast R-CNN algorithm is accurate in object detection but due to the region proposals, it processes an image in 2.3 seconds which is too slow for real-time gate detection.
Another model to consider is the *'You Only Look Once'* (YOLO) [2] algorithm. It is capable of processing images real-time at 45 images per second. It divides an image in a grid of small boxes and predicts a bounding boxes, confidence for those boxes, and class probabilities for each grid cell. As an end result it draws a square box around each object containing one or more grid cells. Despite the excellent speed of the algorithm it is not useful for gate detection due to the square size of the bounding boxes. The box can not accurately line up with the gate edges as they can appear as a concave polygon. Furthermore, the roll angle of the drone will influence the vision making level square boxes useless.
The final model considered is the *'Resnet-101'* [3] algorithm. It is different from Fast R-CNN and YOLO due to the fact that it uses semantic segmentation which classifies each pixel of an image into a class. It uses five fully convolutional layers to construct a matrix where each value is an integer which corresponds to the class of the pixel. As the algorithm assigns every pixel to a class it is capable of lining up with the edges of a gate and can handle concave polygons. Additionally, processing a single image takes around 0.02 seconds making it slightly faster than the YOLO model. Because the Resnet-101 model is fast enough for real-time image processing and classifies each pixel separately it will be used for this application.

2 Implementation of the algorithm

The Resnet-101 model is implemented using three different Python files with an additional file containing some image transformation functions. The data set provided has to be used for training as well as testing. Therefore, the first file sub-samples it into two different data sets. Subsequently, training and testing is done on the data sets in separate files.

2.1 *Sampling*

The data set provided contains 308 images of drone racing gates with corresponding masks all in one folder. In order to make training and testing of the algorithm possible the images and masks are divided into four sub-folders. The algorithm first locates all 308 images inside the folder and randomly samples 10% of them to use for testing. The 30 sampled images are stored in a different folder separately from the 278 training images. For all images the corresponding mask is located and stored in different folders for training and testing.

2.2 *Training*

First the algorithm prepares itself for training by locating all training images and masks, selecting the device to train on, loading the untrained model into the device, loading all images into the data generator and setting the training parameters. In the main loop of the algorithm all training images are fed into the model every epoch and the loss between the image and the mask is calculated. After this, the back propagation fine-tunes the weights based on the loss. The mean loss of all images in an epoch is stored in a list and plotted after training to obtain the training results. Every 10 epochs the weights of the model are saved.

2.3 *Testing*

Before any testing can be conducted, all 30 test images are located, the device to test on is selected and the untrained model and the weights are loaded and send to the device. After this, the algorithm will enter the main loop where a test image is fed into the model every iteration. In one iteration it appends the original image, the output, the test mask, the false positive ratio, the true positive ratio, the area under the curve and the image processing time to a list. Furthermore, it prints the processing time and individual area under the curve for each test image. After testing, the area under the curve for all test images is computed and the ROC curve is plotted. Finally, a few lines of code are added to visualise the result by showing the original image, the output, the mask and the individual ROC curve of each image.

3 Training the model

The model is trained on 278 training images from the data set for 50 epochs on the NVIDIA Tesla P100 GPU available through Google Colab Pro. The total training time of the model is approximately two hours and the performance can be seen in Appendix A. The loss between the output of the model and mask is calculated using the mean squared error and the weights are updated using the Adadelta optimiser [4]. This optimiser is chosen because it dynamically reduces the learning rate during training, eliminating a parameter to be tuned. In most deep learning projects the test and training images originate from different data sets and a part of the training set is sub-sampled for validation during training. However, for this problem all images originate from one data set making sub-sampling for validation unnecessary.
The only parameter left to be tuned is the batch size. A larger batch size will generally speed up the training process but decreases the accuracy and increases the memory needed on the GPU. As training is done on complete images, which require relatively high memory, small batch sizes of 1, 2, 4 and 8 are tested with 20 epochs. In figure 1 it can be seen that a batch size of 1 has the fastest convergence and has a slightly lower loss compared to the others. Regarding the time per epoch, batch sizes 1 and 8 take 65.9 and 55.1 seconds respectively to complete one epoch. From this it can be concluded that it is most optimal to use a batch size of 1 because less epochs are required and the time gained per epoch for different batch sizes is not significant.

4 Computational effort

Testing of the images is also done using the NVIDIA Tesla P100 GPU which is not likely to be found on a drone. On this high-end state of the art GPU an average image processing time of approximately 0.021 seconds is observed. However, when testing the images using the NVIDIA Tesla K80 GPU, which is a more low-end model, image processing takes roughly the same time. The image processing time is fast enough already for real-time gate detection but can be reduced more by reducing the size of the test images. Reducing the height and width 2 or 4 times results for both cases in a processing time reduction of 0.001 seconds while the accuracy is decreasing rapidly. Hence, the model will perform optimally using the full resolution of the test images.

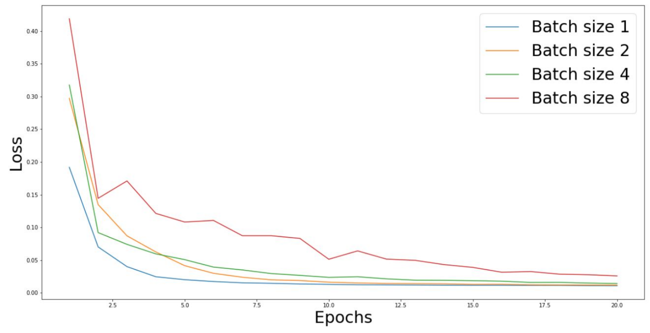


Figure 1: Epoch loss for different batch sizes

5 Results

In Appendix B the qualitative results of the algorithm are shown for the test data set images trained on 50 epochs. From left to right it shows the original image, the output of the model, the original mask and the individual ROC curve. As shown, the algorithm is accurate in detecting the gates regardless of the distance, the number of gates, the place a gate appears in the image and the drone's attitude with respect to the gate. Nevertheless, it is less accurate if a gate is placed far away from the drone behind another gate. However, it should accurately find the gate when it passes the first gate.
Figure 2 shows the ROC curves for all test images with the model trained for 1, 5, 10 and 50 epochs. Surprisingly, the algorithm already is significantly accurate in detecting the gates after just one epoch. Further increasing the number of epochs to 5 already results in an almost perfect gate detection and increasing it further to 10 or 50 epochs results in similar performance. Therefore, it can be concluded that the Resnet-101 algorithm is useful for gate detection in drone racing even without extensive training.

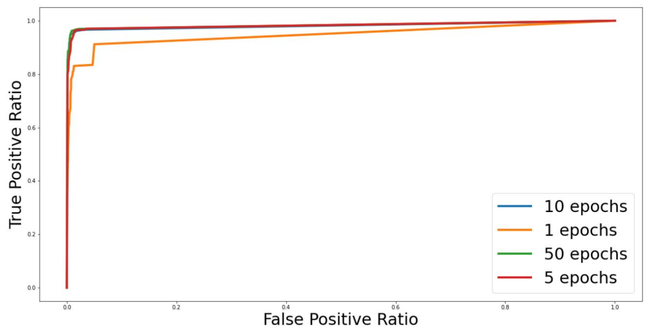


Figure 2: ROC curves generated using the complete test data set trained for different number of epochs

References

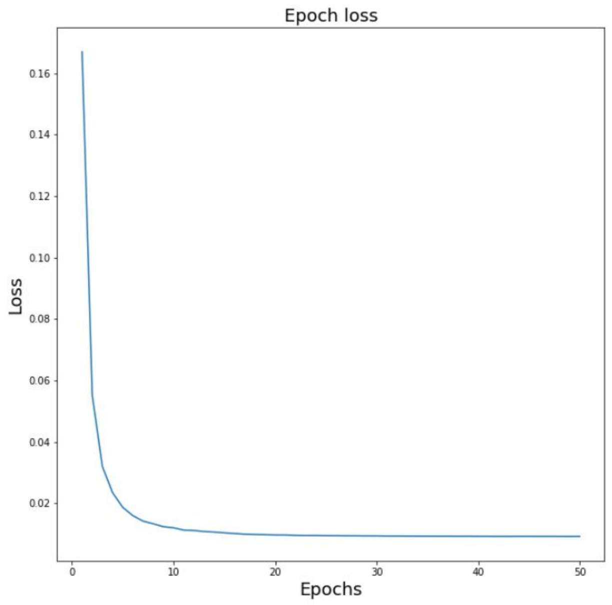
[1] Ross Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015.

[2] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

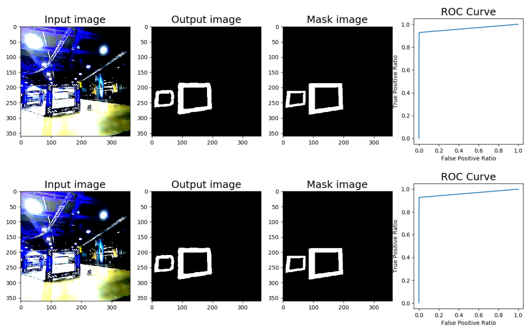
[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

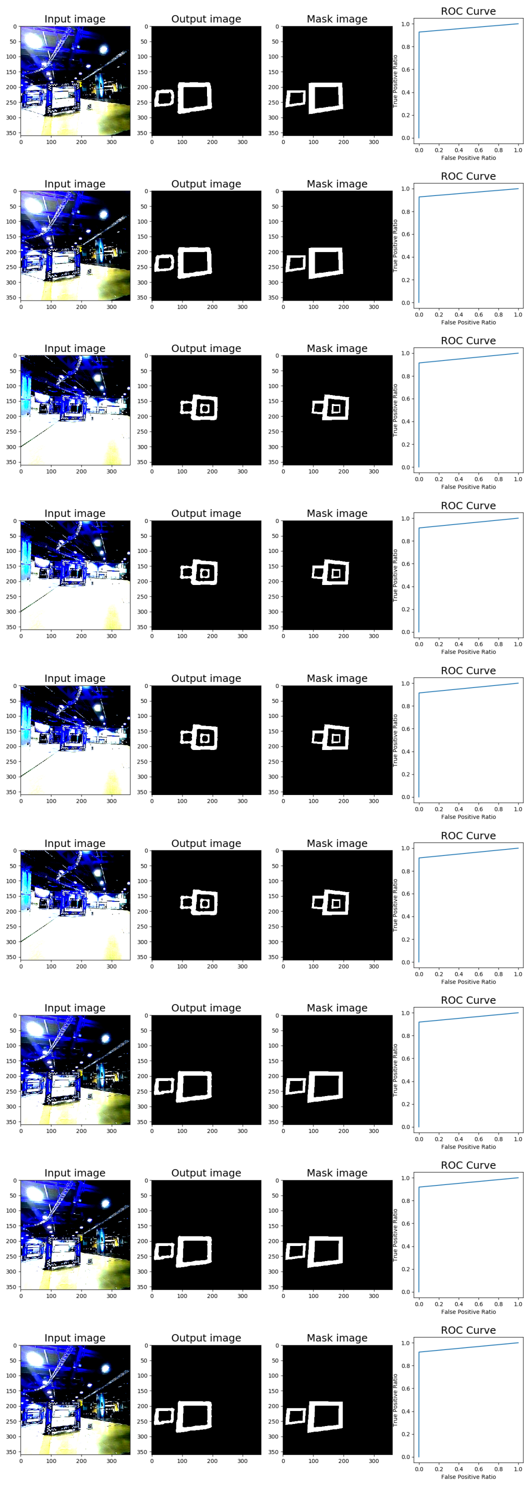
[4] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *ArXiv*, abs/1212.5701, 2012.

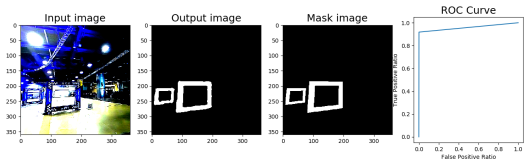
Appendix A Training performance

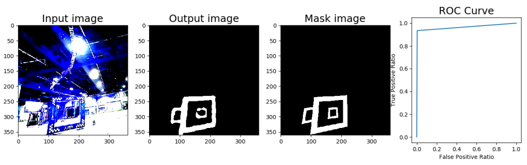


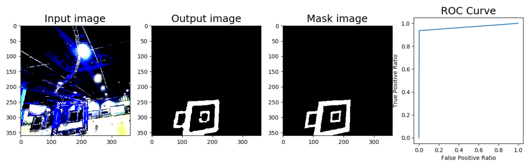
Appendix B Visual results

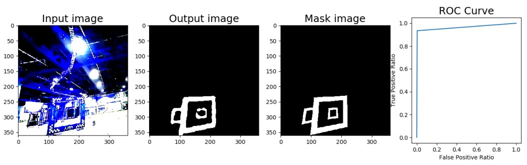


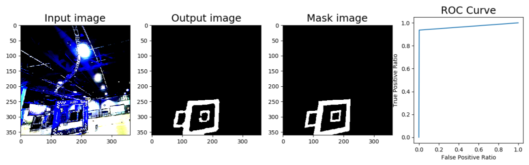


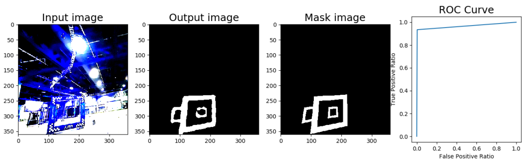

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

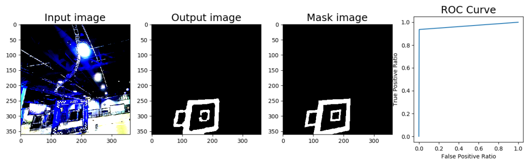

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

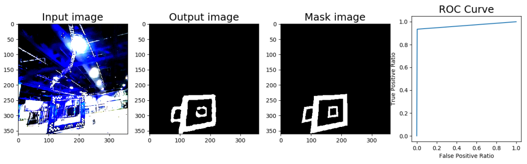

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

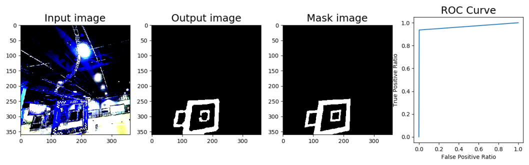

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

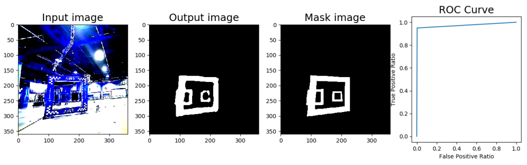

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

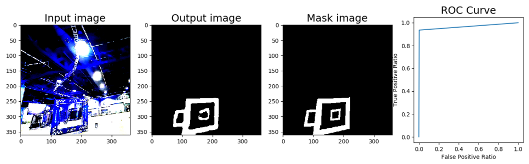

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

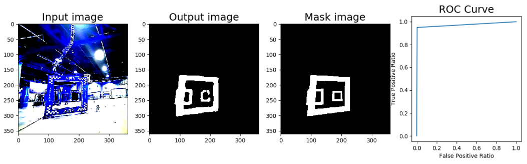

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

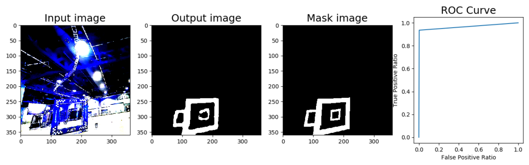

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

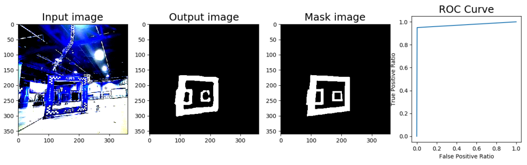

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

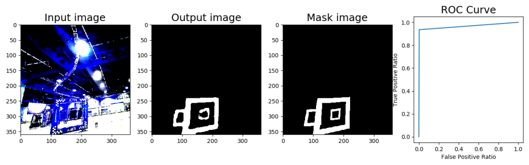

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

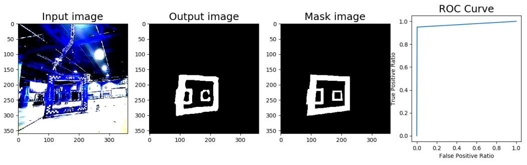

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

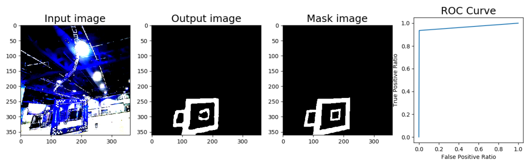

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio

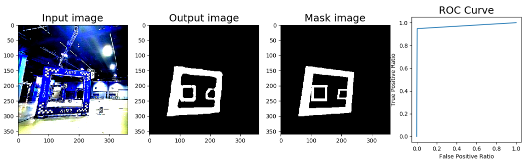

Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio


Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio


Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio


Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio


Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio


Input image
Output image
Mask image
ROC Curve
True Positive Ratio
False Positive Ratio