TEK5040 - Mandatory Assignment 1
Road Segmentation

By Ida Johanne Austad, Autumn 2019

2.1 Run the train script

Summary output - showing last printed step per epoch:

step 60. sec/batch: 0.267912. Train loss: 0.557062
Training epoch: 1
step 130. sec/batch: 0.263379. Train loss: 0.483202
Training epoch: 2
step 200. sec/batch: 0.263849. Train loss: 0.392161
Training epoch: 3
step 270. sec/batch: 0.267845. Train loss: 0.33704
Training epoch: 4
step 340. sec/batch: 0.251621. Train loss: 0.316385
Training epoch: 5
step 400. sec/batch: 0.300723. Train loss: 0.293361
Training epoch: 6
step 470. sec/batch: 0.254134. Train loss: 0.284794
Training epoch: 7
step 540. sec/batch: 0.353006. Train loss: 0.297597
Training epoch: 8
step 610. sec/batch: 0.247758. Train loss: 0.280232
Training epoch: 9
step 680. sec/batch: 0.281699. Train loss: 0.262724
Finished training 10 epochs in 3.5076 minutes.

2.2 See results in Tensorboard

See training and validation history in 2.5.

2.3 Epochs and train steps

How many training steps are there per epoch?
There are 68 steps per epoch.

How many training steps are there in total?
There are 680 steps in total.

2.4 Metrics

Can you think of any cases where you can get good accuracy result without the model having learned anything clever?

Accuracy is the proportion of correctly predicted labels across the total number of predictions. However, one might be very good at predicting the negatives (True Negatives), i.e. not pixels containing road, and not very good at predicting positives (True Positives). Especially, if the dataset is unbalanced (i.e. there are more true negatives than true positives in the samples for instance) then it might seem like the results are good, but in fact, the proportion of true positives correctly labeled is low. Whether this is problematic or not has to be evaluated towards what is the purpose of the model and how the predictions will be used of course.

Can you think of any other metrics that might shed some additional light on the performance of your model?

In the case of road segmentation one may argue that incorrectly labeling something as road (positive) when it is in fact not road is the most unfortunate case (i.e. high number of false positives). In general, other metrics such as precision (true positives / (true positives+false positives)), recall (true positives / (true positives+false negatives)) and their weighted value F1 are considered better for binary classification problems as they provide information especially about the number of false positives and false negatives.

The AUC-ROC score captures the ration between *recall* and false *positive rate* (false positives / (false positives + true negatives)), and might also be a good metric.

2.5 Implementation of U-net

See implementation in attached code.

Summary output - showing last printed step per epoch:

step 60. sec/batch: 1.30779. Train loss: 0.645874
Training epoch: 1
step 130. sec/batch: 1.13909. Train loss: 0.411667
Training epoch: 2
step 200. sec/batch: 1.47827. Train loss: 0.242289
Training epoch: 3

step 270. sec/batch: 1.3538. Train loss: 0.208468
Training epoch: 4
step 340. sec/batch: 1.06194. Train loss: 0.201376
Training epoch: 5
step 400. sec/batch: 1.24766. Train loss: 0.190942
Training epoch: 6
step 470. sec/batch: 1.10283. Train loss: 0.195395
Training epoch: 7
step 540. sec/batch: 1.18371. Train loss: 0.182554
Training epoch: 8
step 610. sec/batch: 1.14666. Train loss: 0.174369
Training epoch: 9
step 680. sec/batch: 1.23403. Train loss: 0.169872
Finished training 10 epochs in 14.2186 minutes.

How many trainable model parameters does your model have?
The U-net implementation has 485,817 parameters, given by model.summary().

Does your model have different behavior during training and test?
There are differences between the results for both the models in the training and test/validation. In general, the development during the training is more variable (i.e. spikes) compared to validation. The reason for the spikes are believed to be because of the updating of weights after each batch during training, while in validation the same weights are used for the entire validation dataset after each epoch. However, they do follow the same overall development. For instance, for accuracy, the development is rather flat until 300 steps and then it improves. Although the improvement comes later (after 300 steps) during training but is steeper.

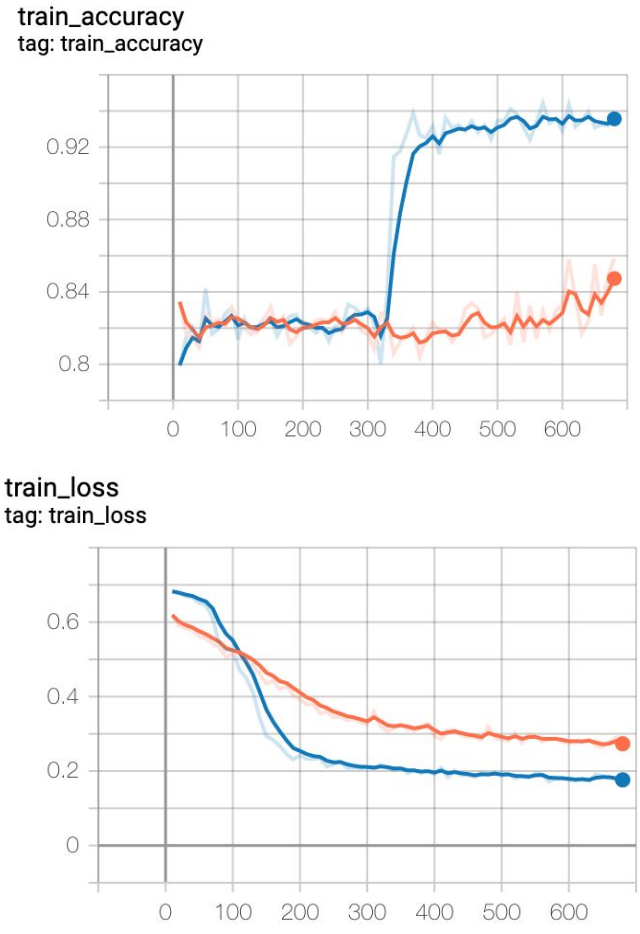
Did you notice any improvement over the original model?
From the training/test history we can see that the U-net model provides better results for both loss and accuracy after 10 epochs. This is believed to be due to the increased complexity of the U-net model compared to the simple model. The large increase in parameters is likely to be better able to capture nonlinear relationships in the data, and thus obtain better results.

By investigating example detections:

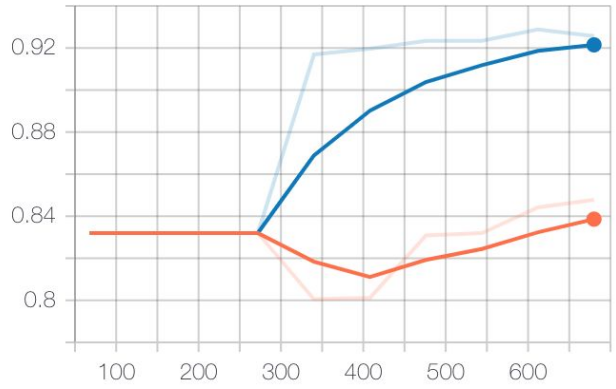
The original model detect both the sky and other objects, such as building walls, as road as well. It also fails to detect a lot of cases of road which is covered by shadow. In the U-net case, these “failures” are more seldom. One such example is given by the two last images in this report.

Additionals:
Best accuracy on validation set: 0.9257

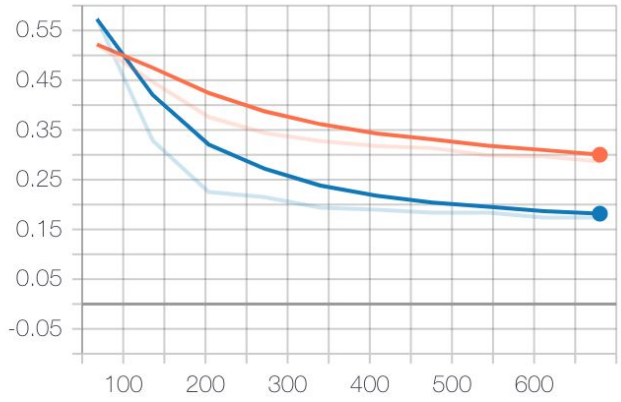
Images from Tensorboard for both the simple model (orange) and the U-net implementation (blue) are given below and support the answers below.



val_accuracy
tag: val_accuracy



val_loss
tag: val_loss



Example segmentation result from simple model, where both the sky and parts of a building wall has been detected as road.

Example segmentation result from U-net model. It does not detect sky as road, and is able to detect the road close to the camera although it is in the shadow.