TEK5040 - Mandatory Assignment 1 - Road Segmentation

1 Introduction

In this assignment we consider the task of semantic segmentation of optical images.

Semantic segmentation is an important aspect of many autonomous platforms. Road segmentation can obviously be very helpful for autonomous cars. After completing this exercise, you will get a basic overview of how to do segmentation with deep neural networks. At the same time, you will get familiar with some basic operations associated with deep learning.

Your goal is to create a deep learning model that can segment the pixels representing the road in a given image. We use the Kitti road data set consisting of 289 labeled images (with labels for each pixel). You may see the training images and labels at **data_road/training/image_2/** and **data_road/training/gt_image_2/** respectively.

2 Tasks and questions

2.1 Run the train script

train.py is a python script that has the basics for reading the image data and training a very simple neural network for the road segmentation task. How you run this will depend on your setup. If you find yourself in a terminal on a mac/liunx machine you could e.g. run the script (assuming that you are in the directory of the train script) by

```
1 python3 train.py <train_dir>
```

where `<train_dir>` is a name you choose, e.g. **out_01**. If you try to train another model, you can rerun the script with e.g. **out_02** as argument, though more descriptive names are recommended.

2.2 See results in TensorBoard

Results of training and validation can be visualized using Tensorboard which is started using the log directory as an argument. In this case the log directory is logs. Start Tensorboard with the command `tensorboard --logdir=<train_dir>`, where <train_dir> is the name of the train-directory specified above. If you have several train directories you could also specify a root directory which contains all of them, and you can then compare the different runs in TensorBoard. If the command is successfull it will print out an URL. Open this URL in a browser and examine the *Scalars* and *Images* tabs.

2.3 Epochs and train steps

How many training steps are there per epoch? How many training steps are there in total?

2.4 Metrics

Development of training and validation losses indicates how well the training progresses. However, there are other metrics such as accuracy, precision and recall which give more insight into the training process. We have added accuracy already as a metric. For certain classification tasks the accuracy may however not always be the most appropriate metric. Can you think of any cases where you can get good accuracy result without the model having learned anything clever? Can you think of any other metrics that might shed some additional light on the performance of your model? You may *optionally* (NOT REQUIRED) choose to implement and add these to your metrics and summaries in **train.py**.

2.5 Implement U-net

The model used in the activity so far is a very simple one. An architecture called U-net is known to give good results for segmentation tasks. See `https://arxiv.org/abs/1505.04597`. Implement the U-net architecture. You may make use of the provided template for this task in **segmentation_models.py**. Comment out the line using *simple_model* and *uncomment* the line uing unet in "*train.py". We are making two changes compared to the model illustrated in Figure 1 in the paper:

- We have changed the initial number of filters from 64 to 8 so that it uses less memory and faster to train. You should then use 16 instead of 128 filters in the next layer, and so on (divide all by 8).

- We use padding for our conv layers, in that way the segmentation mask will be the same size as the input image, and also no cropping is needed.

Hint: for *up_ conv* you may find the `Conv2DTranspose` layer useful.

After you have finished the implementation, please answer the following questions:

- How many trainable parameters do your model have? You could see if you are able to calculate it by hand, but you may also use `model.summary()` to check.
- Do your model have different behaviour under training and test?

Train your model. Did you notice any improvements over the orignal model?

2.6 Extra: Data augmentation

This is optional and not required to pass the assignment. Data augmentation is perhaps the most important regularizer we have when it comes to training models in deep learning. We have not done any of that so far. Implement

3 Handin

We will use devilry for this assignment. Please upload a zipped file containing:

- Your code. We should be able to run it with

```
python3 train.py <log-dir>
```

- A report width some observations on how the training went, and how well your model was able to handle the task. Include the highest accuracy you achieved on the validation set, and any other metrics you may have implemented. You should also include answers to the questions in the text above.
- Log-directory of train run with U-net implementation. Note that there is a size limit on the uploads, so don't include too much else.