TEK5040/9040 Autumn-2024
Compulsory Assignment No 1
Semantic Segmentation

September 1, 2024

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.

In semantic segmentation of optical images, we try to classify each pixel of a given input image into a set of interesting classes. In road segmentation for example, we classify each pixel into two classes: ROAD and NO-ROAD. In our course we do not have a lecture which covers semantic segmentation, so if you are not familiar with the concept, please take a look at resources on the web, for example `https://www.jeremyjordan.me/semantic-segmentation/`.

After completing this exercise, you will get a basic overview of how to do segmentation with deep neural networks. At the same time, and more importantly, you will get familiar with some basic operations (and their implementation with Tensorflow) associated with deep learning.

Your main 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 Preparation

Since you are reading this file, which is a component of the assignment package `tek5040_mandatory_assignment_1.zip`, you must have extracted the package into a folder of your choice. The contents of the package are:

- `problem_description.pdf`: This file

- `train.py`: Training script written in Tensorflow
- `segmentation_models.py`: Tensorflow script defining a simple architecture and a more advanced architecture which is incomplete and completion of that is a part of the exercise.
- `data_road`: Folder containing training and test data

You must also have Tensorflow 2.x installed on your computer. Please follow instructions at `https://www.tensorflow.org/install`. //

3 Task and questions

In this section you will find tasks (what you should do) and questions (what you should answer). The questions are highlighted with color yellow. The submission should contain the answers to the questions as well as results of the tasks. Please refer to the last section on submission for more details.

3.1 Implement train_step and val_step functions

Implement `train_step` and `val_step` in `train.py`. The function `train_step` should

- Find the predictions of the model for the batch of input images
- Calculate the loss of the predictions with respect to the ground truth.
- Find the gradients and update the model parameters using the optimizer.
- Update `train_loss` variable with the loss (this may be done with the `update_state` method of `train_loss`).
- Evaluate the predictions against the ground truth with the `metric_fn` and update `train_accuracy`.

The function `val_step` should:

- Find the predictions of the model for the batch of input images
- Calculate the loss of the predictions with respect to the ground truth.
- Update `val_loss` variable with the calculated loss value.
- Evaluate the predictions against the ground truth with the `metric_fn` and update `val_accuracy`.

3.2 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 should run the script from the directory of train.py, i.e.

```
python3 train.py ⟨train_dir⟩
```

where `⟨train_dir⟩` is a directory name you choose, e.g. `out_01` in which the results are stored. 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. If you run the script from a difference directory you need to change the path to the `data_road` directory.

3.3 Observe the results in Tensorboard

Results of training and validation can be visualized using Tensorboard which is started using the log directory as an argument, i.e.

```
tensorboard --logdir=⟨train_dir⟩
```

In this case the log directory is `⟨train_dir⟩`, that is the name of the train-directory specified earlier in training.

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 successful it will print out an URL. Open this URL in a browser and examine the Scalars and Images tabs.

You may observe that the accuracy is quite high already after one epoch, what is the main reason for this? Hint: Look at the images in TensorBoard, where the pixels in the image that are predicted as road (more than 0.5 score from the network) are colored in red.

The training loss should be decreasing from the start, but it might take some time before the accuracy increases after the first epoch, why do you think this is?

3.4 Epochs and train steps

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

3.5 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`.

3.6 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` for more details. In Fig 1, you can see the architecture of the U-net.

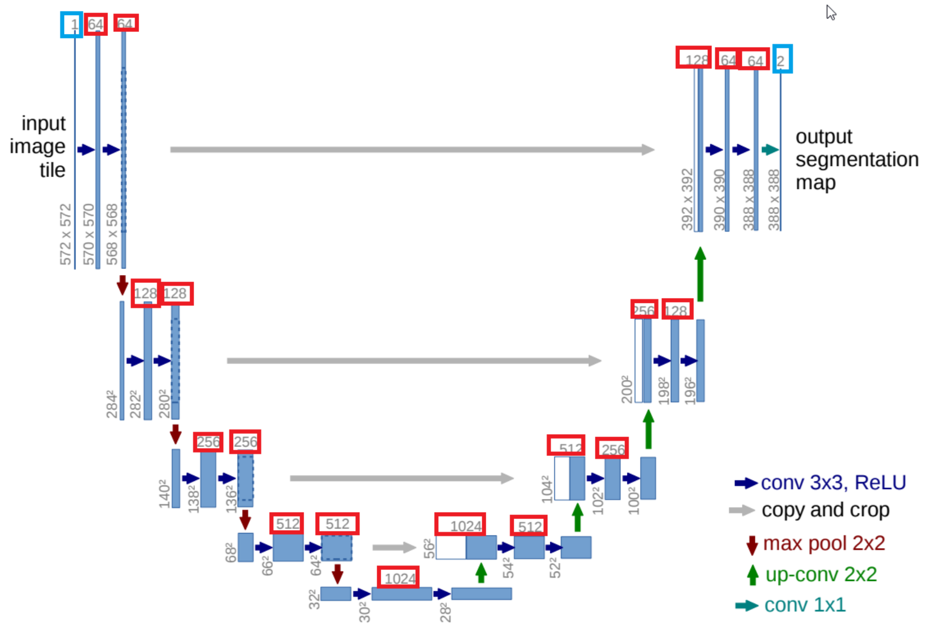

Figure 1: Unet architecture: *Note that in your implementation, the number of filters at each conv layer (shown in red boxes) should be reduced by a factor of 8. Note also that number of input and output channels (shown in blue boxes) should change to 3 and 1 respectively. Obviously all other dimensions shown in the figure will change according to the input size.*

Implement the U-net architecture. You may make use of the provided template for this task in `segmentation_models.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 `tf.keras.layers.Conv2DTranspose` layer useful. After you have finished the implementation, please answer the following questions:

- Briefly describe `transposed convolution` also known as `deconvolution`. (Hint: It may be easier to consider 1D-case.)

- 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 you expect your model to behave differently under training and test modes (i.e. for a given input do you get the same output from the model in train and test modes)? State the reason for your answer.

- In templates provided in `segmentation_models.py`, final layer activation is a `sigmoid` function and the number of output channels is 1. If your task was to perform segmentation with more than two classes (eg: four classes {ROAD, BUILDINGS, CARS, OTHER}), how would you change the activation function and number of output channels?

- Briefly explain why skip connections, shown in gray arrows in Figure 1, are useful in segmentation.

- If your task is classification of a given image, how do you modify the model architecture and the loss function?

- If your task is to transform each input image to an image similar to another given image, how do you modify the loss function?

- Minimizing common loss functions can be interpreted as maximizing the log likelihood. Give arguments for or against such an interpretation for the loss functions in the previous two points (i.e. loss functions in classification and image-to-image transform).

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

4 Submission

Please upload a zipped file containing:

- The completed code files `train.py` and `segmentation_models.py`.

- Screen-shots of accuracy and loss development curves generated by Tensorboard for the two models; simple model and the U-net model.

- Answers to the questions (altogether there are 15 questions and try to write short, concise answers).

Please do NOT upload other items which not mentioned above.