TEK5040 | Assignment 1: Road Segmentation

markuhei - Markus S. H.

2.1 Implement train__step and train__val functions

```
def train_step(image, y):
      """Updates model parameters, as well as `train_loss` and `train_accuracy`.

      Args:
        image: float tensor of shape [batch_size, height, width, 3]
        y: ground truth float tensor of shape [batch_size, height, width, 1]

      Returns:
        float tensor of shape [batch_size, height, width, 1] with probabilties
      """

      with tf.GradientTape() as tape:
          y_pred = model(image, training=True)
          loss = loss_fn(y, y_pred)

      trainable_vars = model.trainable_variables

      gradients = tape.gradient(loss, trainable_vars)

      optimizer.apply_gradients(zip(gradients, trainable_vars))

      train_loss.update_state(loss)

      train_accuracy.update_state(metric_fn(y, y_pred))

      return y_pred
def val_step(image, y):
      """Update `val_loss` and `val_accuracy`.

      Args:
        image: float tensor of shape [batch_size, height, width, 3]
        y: ground truth float tensor of shape [batch_size, height, width, 1]

      Returns:
        float tensor of shape [batch_size, height, width, 1] with probabilties
      """

      y_pred = model(image)
```

```
    loss = loss_fn(y, y_pred)

    val_loss.update_state(loss)

    val_accuracy.update_state(metric_fn(y, y_pred))

    return y_pred
```

By looking at different training examples for other models, I was able to implement my first training and validation steps. TF uses automatic differentiation, which means that we can use the "gradient tape" to find the gradients of the model parameters

2.2 Run the train script

By observing declining loss over the training, I could verify that my implementations worked.

2.3 Observe results in tensorboard

- You may observe that the accuracy is quite high already after one epoch, what is the main reason for this?

The reason why the models get so high accuracy is because the majority class (i.e. non-road) is huge compared to its complement. This means that if the model says for each image that there is no road, it will get a high accuracy as we measure for each pixel of each image.

- 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?

The loss function evaluates the error of the probabilities, but when evaluating accuracy, we only look at whether the probabilities are greater than 0.5. This entails that in order to classify positives, the model has to be confident enough, which might take some time during training.

2.4 Epochs and train steps

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

There are 68 steps per epoch

There are 12 epochs In total there are 816 steps in total

2.5 Metrics

If the dataset we train and validate on has a vast majority class, a classifier can predict the majority class for every datapoint and still get a high accuracy.

F1-score can be used to balance out the results, as it both utilizes precision and recall and evaluates with respect to both the metrics

2.6 Implement U-net

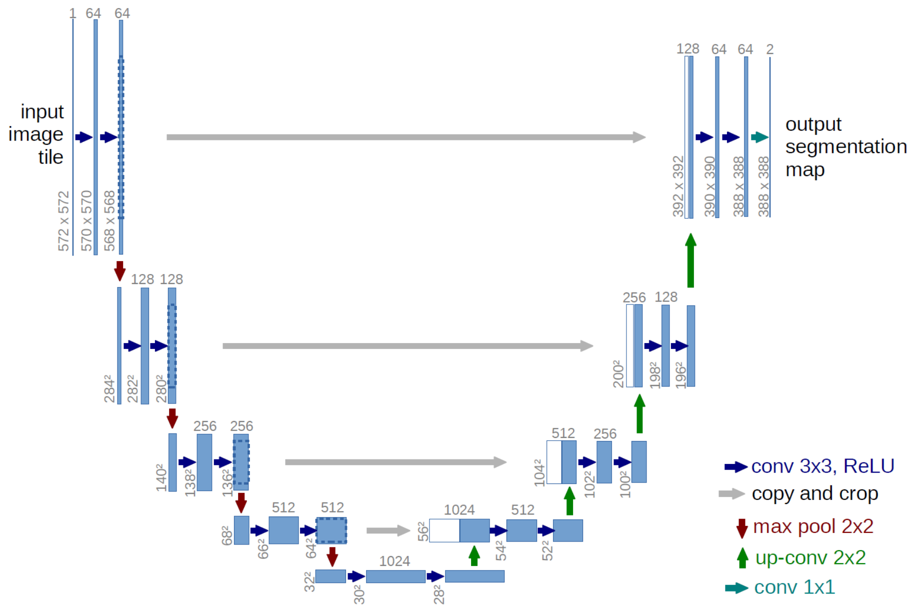

Figure 1: u-net structure

When calling model.summary() i found that the u-net has 8889 parameters in total:

```
Total params: 8,889
Trainable params: 8,889
Non-trainable params: 0
```

It takes way longer to train the U-net than the simple model. This is trivially because the u-net has more (and more complex) layers. Training the U-net for 12 epochs took 9.4 minutes while training the simple cnn on the same data took 2.4 minutes. By the end of the training, the u-net had a loss of 0.18, much lower than the simple cnn that got stuck at 0.27 the last 3 epochs. The u-net got better results on both the sets with accuracies of 92% on the train set and 91% on the dev set whereas the simple model got 86% on train and 85% on dev.