Lakasz Cristina: cristila
September 24, 2023

TEK5040 Assignment 1

Answers to the questions

1. The accuracy is quite high already after one epoch, what is the main reason for this?
Since the accuracy is high and around the same both for the training set and for the validation set, I exclude the possibility of overfitting right after the first epoch. Since the batch size is small, and the data is quite redundant, I believe that the model is learning the most important features already in the first epoch. As I can see from the Tensorboard images, the road is segmented quite well both in training and validation steps, with some exceptions as part of the sky is also considered as road sometimes.

2. 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?
In my case, as the graph show, the loss is decreasing from the beginning and the accuracy is increasing from the beginning. In general, in models where, for example, the initial parameters are far from correct, the loss would be decreasing from the beginning (as it is a matter of loss minimization) while the accuracy will start increasing after a while, as the classification of the data is defined by a threshold and the sigmoid function.

3. How many training steps are there per epoch?
One epoch is a complete iteration through all training data. For each training step one batch of images is processed. Since the batch size is 4 and the total number of training images is 273, as in train.py: train_batch_size = 4 and train_data = get_train_data(indices[:272], train_batch_size).
Therefore the number of training steps per epoch is:
training steps per epoch $= \frac{\text{total number of training images}}{\text{batch size}} = \frac{273}{4} = 68.25$

4. How many training steps are there in total?
The total number of training steps is the number of steps per epoch multiplied by the number of epochs. The total number of epochs is 12, as in train.py: train_epochs = 12
total training steps = training steps per epoch $\times$ number of epochs $= 68.25 \times 12 = 819$

5. Can you think of any cases where you can get good accuracy result without the model having learned anything clever?
A model achieves very good accuracy without having learned anything when it is overfitting the data. The model in this case merely memorizes the training data, leading to poor performance on the unseen testing data. Another case could be an imbalanced dataset, where data is divided in classes and there is a dominance of one class in the data, leading the model to always predict that class, but without having learned any underlying information about the data.

6. Can you think of any other metrics that might shed some additional light on the performance

of your model?
Besides accuracy, precision and recall, other metrics that can give us some information about the performance of the model, which in our case follows logistic regression, are: Area Under the ROC Curve (AUC-ROC) and F1-score. In linear regressions other metrics could be: Mean Absolute Error (MAE), Mean Squared Error (MSE) and R^2 (R-Squared), but these are not well suited in our case as we are dealing with logistic regression.

7. Briefly describe transposed convolution also known as deconvolution.
A transposed convolution reverses the spatial transformation of a regular convolution. In a convolution the kernel is slided over the input image while performing element-wise multiplication and summation. On the other hand, in a transposed convolutional layer the operations are computed in the opposite order, so that if the convolution reduces the image size, a transposed convolution upsamples it to a grater size. This is done by multiplying each value of the input layer with the kernel and then combining the results according to the initial positions of the input layer and summing the overlapping values. The size of the output can be controlled by the stride and padding parameters of the layer.

8. How many trainable parameters do your model have?
Using model.summary() I could find that the trainable parameters the model has are: 485817

9. 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.
It could behave differently when it is not learning from the data but just memorizing it, so during training the model will perform well, but during testing it will perform poorly. I this case, tho, the input is still different, as it should never be the same: in training mode the model will use the training part of the dataset, and in testing mode the model will use the testing part of the dataset. When Dropout or Batch normalization are used, according to the mode they are set to (training or testing), the model will behave differently under the same input. However, these techniques are not used here, so the model would behave the same way.

10. If your task was to perform segmentation with more than two classes, how would you change the activation function and number of output channels?
The model uses the sigmoid activation function, as it performs binary classification. In a multiclass classification to perform segmentation, using a softmax activation function would be more appropriate, as it produces a probability distribution over the classes. In addition, for each class, there should be an output channel, therefore number of classes = number of output channels. In fact, each output channel will then be a mask of one class.

11. Briefly explain why skip connections, shown in gray arrows in Figure 1, are useful in segmentation.
Skip connections in U-Net concatenate the features from the encoder stage to the symmetric counterpart in the decoder. Therefore, subsequent convolutional layers can perform over both encoder and decoder features. The decoder ones in fact, offer semantic information (general area where the road is) whereas the encoder ones offer more spatial information (the exact pixels of the road). Combining these features using skip connections, is useful in segmentation as it helps provide a

more pixel precise segmentation, enhancing the model ability to capture fine details.

12. If your task is classification of a given image, how do you modify the model architecture and the loss function?
U-Net is used for image segmentation, where the objective is to classify each pixel of the image. Therefore, to perform image classification, the decoder part of the network can be removed, as there is no need to upsample the feature map to the original image size. After the last layer of the ecoder, a global max/average pooling layer and then a fully connected layer can be added, to output the probabilities of each class. Binary cross-entropy loss function can be used for binary classification, and categorical cross-entropy loss for multiclass classification.

13. If your task is to transform each input image to an image similar to another given image, how do you modify the loss function?
A loss function that could be suited for this problem is contextual loss, which compares regions with similar semantic meaning. It is indicated when working with image transformation with non-aligned data.

14. Did you notice any improvements over the original model?
The U-Net model performs better than the simple model. The segmentation is performed more pixel-wise precise and there is much less occurrence of inaccurate segmentation (e.g. segmenting part of buildings as road). Following is a considerable example of performance improvement.

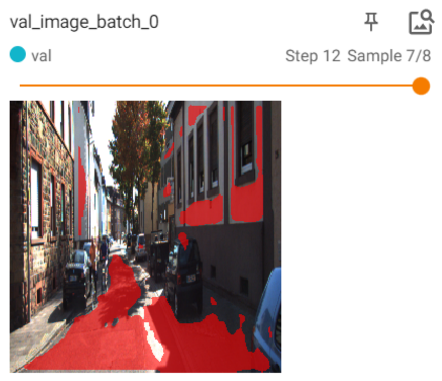

Figure 1: Simple model

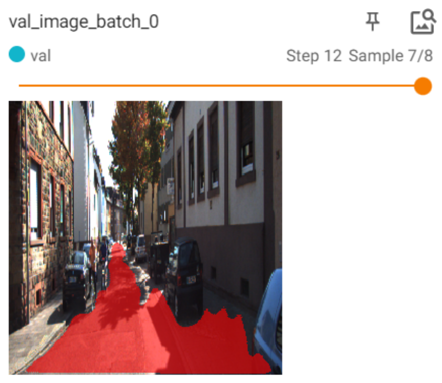

Figure 2: U-Net

Ulterior proof of better performance can be seen comparing the accuracy and loss graphs. U-Net achieves higher accuracy and smaller loss. Following are the screenshots of accuracy and loss development curves for the two models.

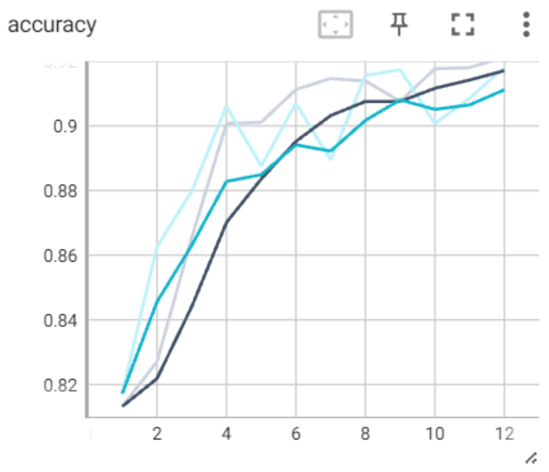

Figure 3: Simple Model Accuracy

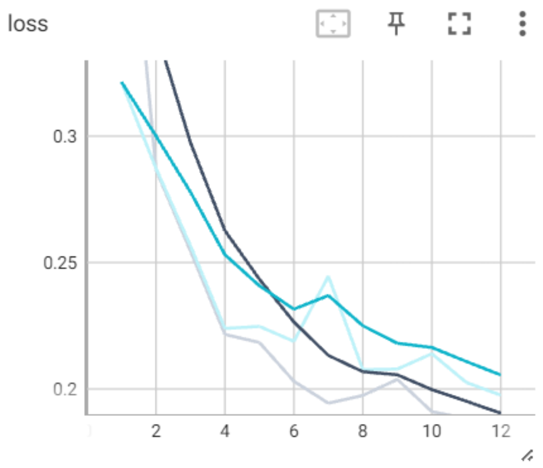

Figure 4: Simple Model Loss

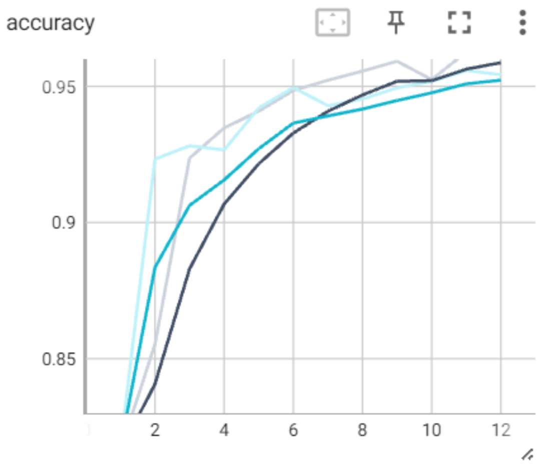

Figure 5: U-net Accuracy

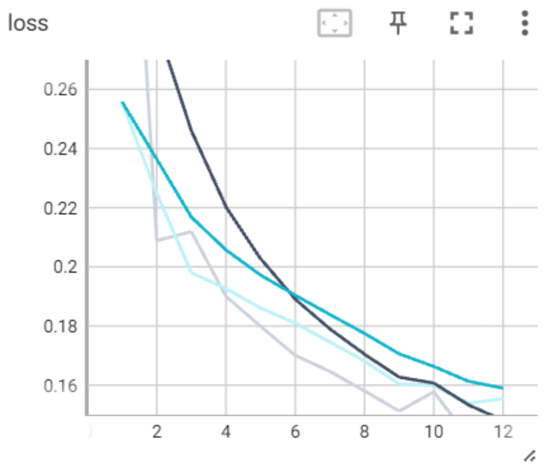

Figure 6: U-Net Loss