Artificial Neural Networks & Deep Learning Project

Team 10520637_10527649

December 15, 2019

1 Lab Model

We implemented a first model based on the notebook provided during the lab session.
Setup summary:

- Validation Split inside the dataset
- Batch Size = 16
- Encoder depth=4
- Encoder block: Conv2D+ReLU-¿ Conv2D+ReLU -¿ MaxPool2D
- Last Encoder layer: Conv2D+ReLU-¿ Conv2D+ReLU -¿ MaxPool2D
- Decoder depth= 4
- Decoder block: UpSampling2D -¿ Conv2D+ReLU -¿ Conv2D+ReLU
- Loss function: SparseCategoricalCrossEntropy
- Optimizer: Adam(learning_rate = 1e-3)

This turned out in `train_loss=0.2949` , `valid_loss=0.3012` and `kaggle_score=0.21515`.

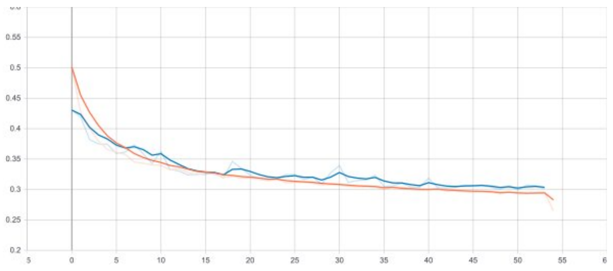

Figure 1: CNN `loss` plot

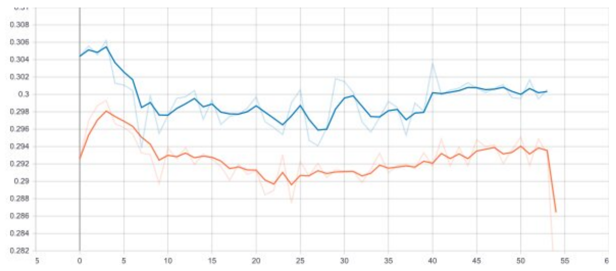
Figure 2: CNN `My_IoU` plot

The graphs shows a sudden decrease because the training was interrupted before terminating.

1.1 First Model with concatenation

We tried to add concatenation to the model provided during the lab session. The setup did not change noticeably. This turned out in `train_loss=0.250`, `valid_loss=0.300` and `kaggle_score=0.37682`.

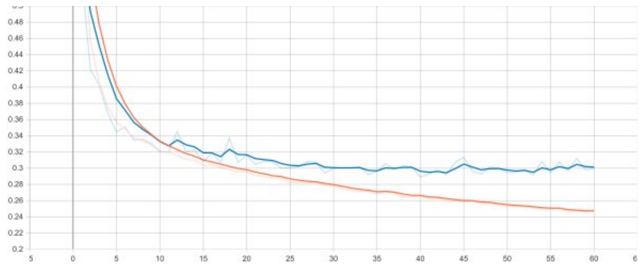
Figure 3: CNN with concatenation `loss` plot

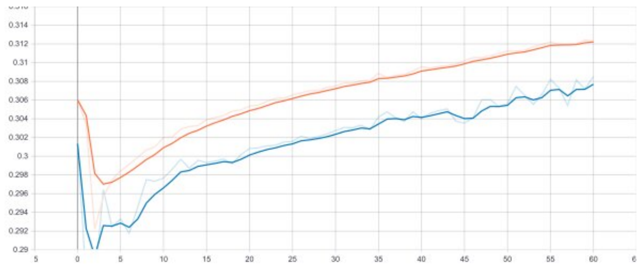
Figure 4: CNN with concatenation`My_IoU` plot

2 UNet

We realized we were not applying concatenation correctly so we reimplemented a UNet style model using Keras Functional API, explicitly declaring each layer. We used the same setup as before. Here we realized the model was not performing as good as we expected, the graphs showed that the model was not learning. This started a series of attempts to change the model hyperparameters. Our best attempt resulted in `train_loss=0.21`, `val_loss=0.31` and `kaggle_score=0.39594`

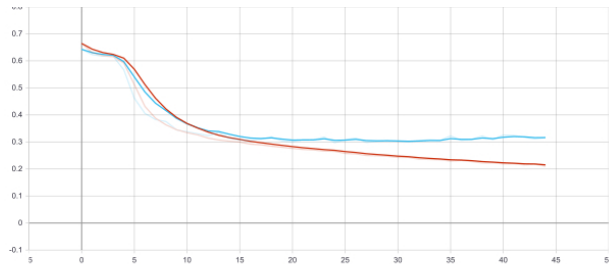

Figure 5: UNet `loss` plot

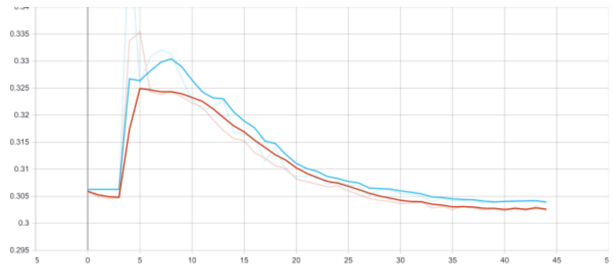

Figure 6: UNet `My_IoU` plot

3 Failing attempts

The main issue with the model was that it went straight to a local optimum with `train_loss=0` and `train_my_IoU=0`.
We noticed that the `num_classes` parameter was used to define the number of filters in the last layer, since we needed just one image as prediction output we changed to `num_classes=1`.
Adding Stochastic Gradient Descent to model optimizers slightly improved the model performances since it didn't immediately go down to `my_IoU=0`.

3.1 Loss Functions

We thought the model was going into a local optimum due to the loss function we were using. To overcome this problem we tried different loss functions.

3.1.1 Binary Cross Entropy

The Cross Entropy loss functions measures the accuracy of the network over each single pixel.

$$\mathrm{CE}\left(p, \hat{p}\right) = -\left(p \log\left(\hat{p}\right) + (1-p)\log\left(1-\hat{p}\right)\right)$$

3.1.2 Dice Loss

The Dice loss gives a feedback closely related to IoU. It measures the average prediction accuracy over the target instead of looking at the precision of single pixels.

$$\mathrm{DC} = \frac{2TP}{2TP + FP + FN} = \frac{2|X \cap Y|}{|X| + |Y|}$$

- TP = True Positives
- FP = False Positives
- FN = False Negatives
- X = Prediction
- Y = Target

3.1.3 Combined Loss

We decided to use Binary Cross Entropy in combination with Dice Loss, giving the same weight to both, in order to find a model able to identify high-level features and achieve good accuracy at a local scale.

3.2 Dataset Split

Following the advise of tutor Lattari we changed our dataset split method and rolled back to the `ImageDataGenerator` built-in `validation_split` feature. This showed immediate improvements meaning we had some errors in the code used to split the dataset.

4 Transfer Learning

All the previous attempts we tried on the last model turned out in very low Kaggle scores. This meant we had a very low performing model so we tried to implement Transfer Learning.

4.1 VGG16

We used VGG16 as Encoder with fine-tuning of last six layers and the UNet Keras-Functional-API implementation Decoder mentioned before. We used Combined Loss as loss function.
This model ended with `train_loss=0.4653`, `valid_loss=0.5639` and `kaggle_score=0.47406`.

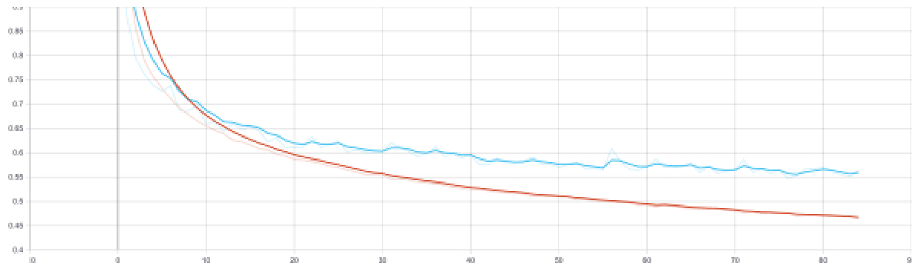

Figure 7: VGG16 `loss` plot

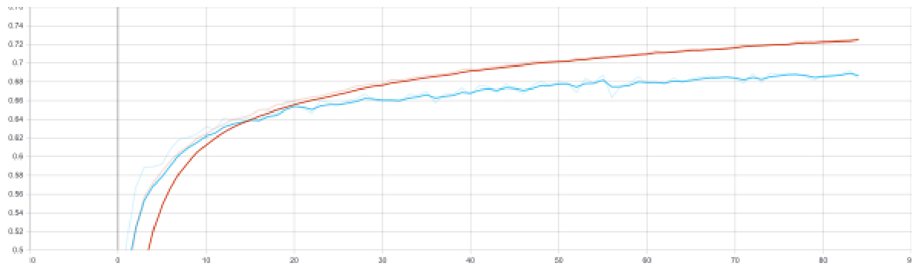

Figure 8: VGG16 `My_IoU` plot

4.2 VGG16 with concatenation

We implemented VGG16 using Keras Functional API and initiliazed the model with downloaded weights.

4.3 VGG19 with concatenation

We repeated the attempt with a more recent architecture.