Artificial Neural Networks & Deep Learning Project

Team 10520637_10527649

November 29, 2019

1 Convolutional Neural Network

We started from the model provided during the lab lectures to understand how the process worked.

1. To solve the splitting between the validation and training set we used the parameter validation_split=0.8 in the ImageDataGenerator constructor.
2. To adapt the test images to the input size we use the Image.resize() method.

`train_acc=0.8035 val_acc=0.5309`

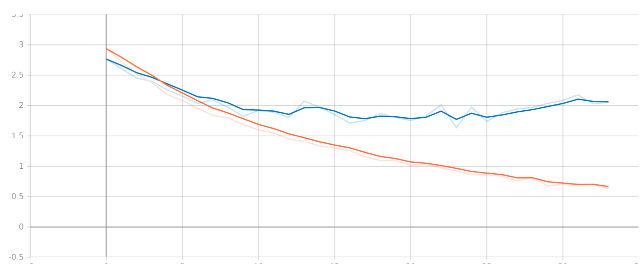

Figure 1: CNN `val_loss` plot

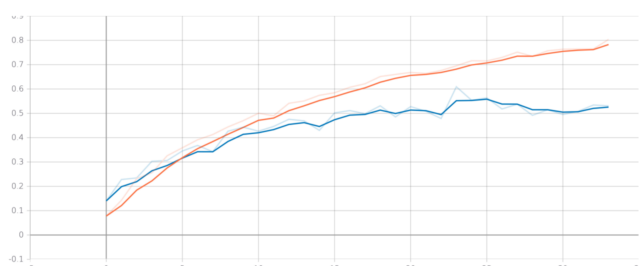

Figure 2: CNN `val_acc` plot

1.1 Dropout and Early Stopping

After our first submission we tried to change the architecture adding a dropout layer. The new model showed a slow learning rate and no particular improvements.
We modified the patience and the min_delta values of the Early Stopping callback. Increasing the patience of the callback showed us that the model stopped learning and the performance increases were very little.

1.2 Network Depth

We tried to add some convolutional layers to the network by simply changing the depth parameter, after some epochs in which the network didn't learn we understood that the receptive field was too high and the max-pool layers should be reduced.
We tried to modify the convolutional block itself by doubling the convolutional layers in each block.
We tried to completely modify the network structure by building sequentially the architecture, consisting of 2 blocks of 2 convolutional layers and 3 blocks of 3 convolutional layers.
We tried to modify the classifier by adding two layers resulting in a fully connected network with 512 → 256 → 128 neurons.
We repeated the experiment with a network with 512 → 512 → 256 neurons.
After all these tries the validation accuracy did not increase.

2 Transfer Learning: VGG16

We firstly tried with the code provided during the laboratory session, we set the freeze value to 18 thus training only the last 5 layers plus the classifier itself. This turned out in huge performance improvement.
`train_acc=0.9126 val_acc=0.6808`

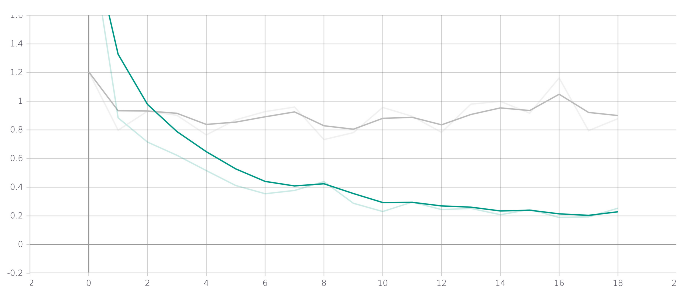

Figure 3: VGG16 `val_loss` plot

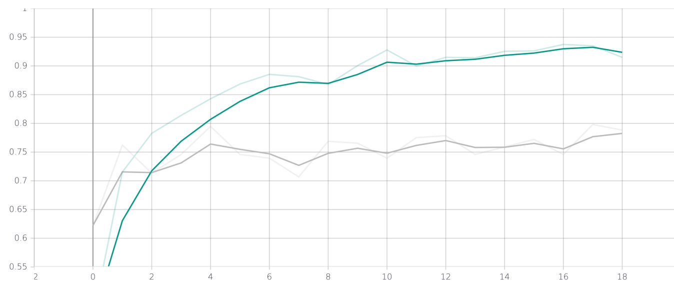

Figure 4: VGG16 `val_acc` plot

2.1 Data Augmentation Correction

Through the development of our project, we observed that applying data augmentation to the validation set could turn out in a pessimistic evaluation of the model performances. Instead of using the internal function of DataImageGenerator we implemented an algorithm to split the training set into two separate folders and then apply data augmentation only on the training dataset. This resulted in a more accurate validation score w.r.t. the test performance.

3 Transfer Learning: Xception

Considering all the available architectures saw during lectures VGG16 looked too heavy and not so performing. We applied Transfer Learning with Xception architecture without fine-tuning, we trained only the classifier. This showed immediate performance improvements.

`train_acc=0.9801` `val_acc=0.9698`

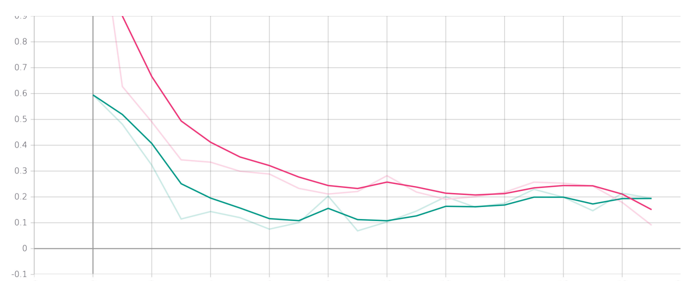

Figure 5: XCEPTION `val_loss` plot

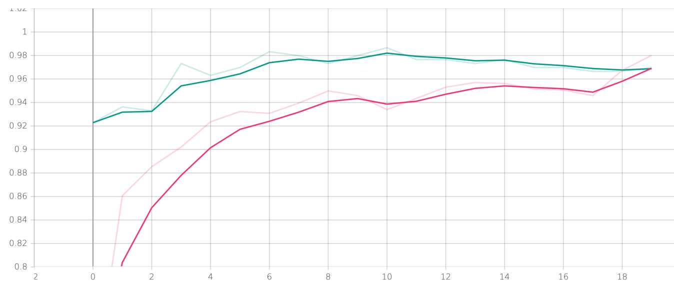

Figure 6: XCEPTION `val_acc` plot

3.1 Fine Tuning

1. We started training only the classifier with a specific dataset split.

2. We shuffled again the dataset, changing the validation split. Then we trained the network unfreezing the weights of the last 6 layers of Xception.

This showed a small improvement in validation accuracy.

3.2 Global Average Pooling & Max Pooling

We tried to add a Global Average Pooling layer at the end of the network and restart the process of fine-tuning. Then we repeated this process with Global Max Pooling. The best results were found with the second architecture.