**Homework\_1\_AN2DL**

**A. Casaroli – B. Gallazzi – A. Maserati**

We initially decided to split the train dataset in 80-20 using *‘flow\_from\_dataframe’* method to obtain training and validation sets, used to evaluate the model before testing it on Kaggle. Initially we considered to select an image size of 256x256 and a soft augmentation on train dataset.

We started with a basic structure (i.e. made by convolutional layer, activation layer and maxpooling), doubling the number of filters in each sub-sequence. After various trials (including depth, number of filters, searching for the best learning rate (lr), best batchsize, best classification layers and activation functions), our best result was of 0.67 accuracy on test set **(see notebook ‘homework\_1\_a.ipynb’ for details).**

Therefore we decided to move onto a more complex structure, getting inspired by consecutive convolutional layers used in VGG16 seen at lectures. In these following custom models, a static split 90-10 of train dataset is preferred (made once with ‘dataset\_preparation.ipynb’, then calling data using *‘flow\_from\_directory’* method), to be able to train on more images (obtaining about 80-10-10 split overall) and to make the experiments reproducible and comparable; further operation on data are directly done in the notebooks. A more aggressive augmentation is used. Finally, in these types of structure, a higher batch size allows to reach better performances.

In ‘**homework\_1\_b.ipynb**’ we decided to have 4 conv.-conv.-maxpool subsequences, increasing the number of filters from 16 to 32, 64 then 128; we included dropout at high rate (0.5) and batch normalization to use regularization and avoid overfitting. We decided to increase the patience of early stopping, since with a low one the train was stopped while the network was still learning; we included ‘restore\_best\_weights=True’ argument. Finally, lr=1e-3 was the best rate for this model. Since a good performance was reached (~ 0.8 accuracy on train and validation, 0.846 on test), but in a long train, we decided to try to find a better model to reach a comparable result in less epochs.

In the following, the main aspects on which we focused: number of filters, number of consecutive conv. layers, depth, classification layers,dropout rate, batch size, learning rate (‘reduce lr on plateau’ class included, but with poor performance).

We could appreciate how data augmentation is relevant for a better performance of the network, especially having a small dataset; furthermore, consecutive convolutional layers allow to reach the same receptive field with lower number of parameters to train and higher number of nonlinearities. Many other configurations were giving about the same result of 0.8 accuracy in train, validation and test (**see notebooks ‘homework\_1\_c/d/e.ipynb’ for details**).

Our best result is reached in **‘homework\_1\_f.ipynb’** where after two sequences of conv.-conv.-maxpool with respectively 32 and 64 filters, two sequences of conv.-conv.-conv.-maxpool with 128 and 256 filters are used; dropout rate=0.1, lr=1e-3 and only one dense layer with 512 neurons and relu activation function. With this configuration, we obtained 0.83 accuracy in train, 0.85 in validation and 0.855 in test.