**Group Tis**

**Manuel Bressan, Federica Maddaloni**

First of all, once we have downloaded the data, we noticed that the validation set was not given. So we developed an easy script to randomly split the training data into a training set (84%) and a validation set (16%). (‘*Database\_Preparation\_Snippets.ipynb’)*

We have started by using the notebook file created during the first lab session as a framework, changing the hyperparameters and the number and type of layers.

We then tried training lots of different models from scratch, but without reaching any better result. This involved changing the *ImageDataAugmentation* parameters, the batch size, the size of the input image, the learning rate (with the introduction of the *ReduceLROnPlateau* callback), the number of filters, their size and other hyperparameters. We obtainined a val\_accuracy of 0.7168. (In *‘Model\_From\_Scratch.ipynb’* there is the implementation of the best model reached from scratch)

Eventually, to improve our score we started using Transfer Learning, but we realized that, since *ImageDataAugmentation* augments images in real-time, the training was too slow and it did not allow us to perform specific preprocessing procedures. So, we decided to change the data pipeline to fully exploit the computational power of the GPU. In order to do that we had to load all the images into a nparray and we saved it on the drive.

Then, following many guides online, we understood that, to create an efficient pipeline, we had to implement a custom image preprocessor and generator.

This new framework also enlarged our possibilities for data augmentation and normalization. Regarding the first one we tried to crop, rotate, apply Gaussian blur and flip (only left-right) the images. For the latter we tried three different approaches: the usual rescaling in range [0,1], the rescaling in range [-1,1] and the standardization aiming to a mean equal to 0. We found out that the best result was reached with the first one.

Later on, we finally employed transfer learning, starting with several famous architectures (*VGG16*, *VGGFace*, *ResNet50*, *MobileNet* and *EfficientNet*). Firstly, we trained the models with our classes, but freezing all the layers of the employed architecture. That brought to an improvement and we were able to reach around 0.9 of val\_accuracy. Then we tried to gradually unfreeze all the layers, but in many cases the results were not as good as expected. Eventually, we even tried to train the whole architecture.

Among all the architectures, the one which reached the highest value was *EfficientNetB5*: this model was introduced last year in 7 versions ranging from B0 to B7, claiming to reach the state-of-the-art performance on the ImageNet database. Moreover, this year was released a new set of weights called ‘*noisy-student’* that guarantee an even greater accuracy. For this reason, we downloaded and exploited them in the model and we actually got a val\_accuracy of 0.9504.

The main problem at this point was that by cropping the images we were losing a lot of information. So we needed to come up with an idea to overcome this issue: we tried with a sliding window creating 2*n*+1 different test predictions, *n* for each side and a central one, each with its own weight. Eventually we got an improvement with *n*=5 and all equal weights (test accuracy of 0.9555). *(‘EffNetB5\_SlidingWindow.ipynb’)*

This method, though, led us to a dead end because we had no more room for improvement. For this reason, we decided to start again from the images and we also added a FC layer. We thought that the central focus of all the images were the faces, so we employed *DSFD (Dual-Shot Face Detector)* to recognize them. With those faces we created a new dataset where every image is transformed into a 3x3 grid, in each of whose squares we put a face (test accuracy of 0.9644). (‘*Database\_Preparation\_Snippets.ipynb’, last part)*

In order to minimize the detection error, we firstly removed all the “faces” smaller than a certain threshold (found by trial and error) and then increased the size of the bounding boxes because it often left masks out of them. This led us to a final 0.9755 test accuracy. *(‘EffNetB4\_Grid.ipynb’)*