AN2DL HW 1 Angboolers Team

Arnab Mondal 10714338
Massimiliano Sica 10558133
Marco Labate 10491539

22 November 2020

1 Introduction

For this assignment we were requested to design a convolutional neural network for image classification. In particular, we were provided with a dataset containing pictures of people wearing masks to be divided in 3 classes where the first class was characterized by all the people wearing a mask, the second one by someone not wearing a mask and the last one by no one wearing a mask. We solved the problem with both a custom and transfer learning approach.

2 Custom Model

The very first thing we noticed was that the images of the training set were not labeled, instead the labels were provided in a separate json file. To solve this issue, we wrote a code snippet to create a pandas dataframe with all the images saved with their names and the corresponding labels. Once the training dataframe was retrieved we shuffled it in order to deal with the fact that the images were provided in order and that would cause issues during validation. We decided to test our model with different values of the validation set (taken from the training set) ranging from 10% to 25% (10% provided the best answer) in order to deal with overfitting. Another issue to be dealt with coming from the structure of our dataset refers to the fact that images have variable size. This issue was simply solved by forcing a resizing of the images to (256,256) using the target_size parameter associated to the train_datagen.flow_from_dataframe method. We then decided to exploit data augmentation to increase the accuracy of our model, many different combinations of parameters were tried to obtain a better result. Another parameter that we tuned a lot was the batch size which ranged from a minimum value of 7 to 128, the best result was obtained with value of 10. Then we started working on the definition of our model by setting the number of convolutional, activation and maxpooling layers. We modulated the architecture seen in class by trying many different possible combinations of depth and features, we also tried to substitute the FC layers with GlobalAveragePooling layer which, however, didn't provide satisfactory results with

respect to the "classical" architecture. Furthermore, we added some dropout and batch normalization layers to deal with overfitting and thanks to these we reached to the optimal solution. We also modulated the learning rate values from 1e-3 to 1e-6 eventually concluding that the best one was 1e-4. In all of our model we exploited early stopping with weight restoration for 100 epochs. However, the best result was obtained by training only for 25 epochs (a result obtained experimentally). Once this was done, we exported everything as csv for the upload on Kaggle. Our best accuracy was 0.84222.

3 Transfer Learning Model

- VGG16: we performed fine tuning starting from the model given in class, and we added a global average pooling layer at the output of the VGG16 then a dense layer with some regularizers (L1,L2), a dropouts to deal with overfitting and a softmax as output. This model gave us a testing accuracy of 0.8666

- Xception: for this model we retrained completely the whole network from scratch without performing fine tuning and without importing the top part. Then, after flattening the output, we added four blocks each made of a dense layer, a dropout layer, a batch normalization layer and a Relu layer. At the end of the network we added a 3 node softmax layer to get the output. In addition, we also used the ReduceLROnPlateau callback to reduce the learning rate when the accuracy stops improving. This model got us a testing accuracy of 0.90666.

- DenseNet121: for this model we imported the pre-trained DenseNet121 architecture on "imagenet" weights and we changed the image target_size to (224,224). After this we added one average pooling layer, a flatten layer, a dense layer, a dropout layer and a softmax layer as output. This model gave us a testing accuracy of 0.9000