NAME\_1: Marco

LAST-NAME\_1: Introvigne

STUDENT-ID\_1: 10750466

NAME\_2: Antonio

LAST-NAME\_2: Urbano

STUDENT-ID\_2: 10527285

NAME\_3: Enrico

LAST-NAME\_3: Voltan

STUDENT-ID\_3: 10525467

LEADERBOARD NICKNAME: TLN (Three Little Neurons)

First of all we analysed the dataset and we noticed that the dataset is slightly unbalanced, so we have created two functions to equalize it.

1.Up-sampling: we have randomly duplicated some samples from the minority classes, setting the number of samples equal to the one of the majority class

2.Down-sampling:we have randomly removed some samples from the classes having more samples and set their size equal to the size of the minority class.

Before building our model architecture we split our training dataset into training and validation sets, in order to better understand the performances helping us with some plots.

For the training set we used the Data Augmentation preprocessing technique by applying some transformations on the images (like rotations, shear, zoom, brightness,...). For this purpose we started with no Data augmentation and then we gradually increased the type and the value of the parameters in the ImageDataGenerator constructor.

Then we started working in parallel with two approaches:

1. Build our own network, by learning it from scratch through trial and error

For this approach we implemented a very simple network similar to the one seen during lectures and starting from that we follow a greedy approach, gradually improving our results.

At each execution we analysed the obtained results, in order to understand if our network was underfitting or overfitting and we acted consequently by respectively increasing or reducing the network complexity.

The best result we obtained was with a network structured as follows:

two main blocks, the first consists of two convolution layers followed by maxpooling (repeated twice) and the second with only one convolution layer followed by maxpooling always repeated twice. All convolution layers have a relu activation function, 128 filters of size 3x3 and padding equal to 1. The top layer is composed of a flatten layer, a dense layer with 512 neurons and the output layer.

With this network we reach an accuracy of: [0.7644%]

2. Using transfer learning

We tested different solutions. We tried with several pre-trained networks, in particular VGG16, Resnet152V2, NasnetLarge, InceptionV3, InceptionV2+Resnet, MobileNetV2.

The “InceptionV2+Resnet” gives us the best result.

So we started from that pre-trained model and we ran cross validation on this network to increase the accuracy. We also tried it with and without fine tuning and changing several optimizers in order to reach better results.

We have tried both the Sequential and the Functional approaches.

Also in this scenario, by analyzing the validation loss, we tried to understand the behaviour of the implemented models by adding some intermediate layers. More precisely we tuned the number of layers to be trainable when the fine tuning was active, we added the GlobalAveragePooling in order to minimize overfitting by reducing the total number of parameters in the model.

Finally we test both the performance with and without the BatchNormalization layer.

As final result we implement the following network, which give as a final accuracy of [0.9488%]

For both the approaches, we also tried several configurations of the image\_size and the batch size in order to improve the performances. The best performance we obtained is with

* img\_size = [204, 204, 3]
* batch\_size = [16]

for the network without transfer learning, and with

* img\_size = [400, 400, 3]
* batch\_size = [16]

for the network with transfer learning.