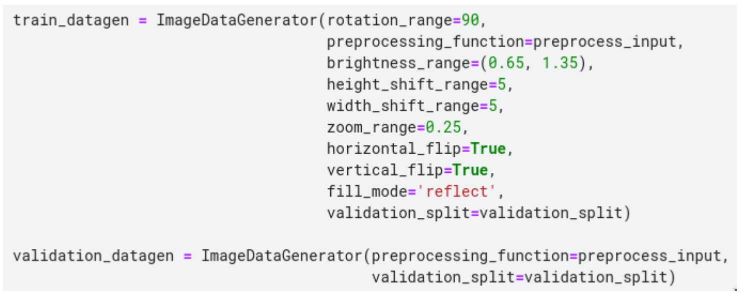
Artificial Neural Network and Deep Learning
Homework 1

Angelo Capponcelli   Pasquale Castiglione   Mehdi Samimi

Data Load and Augmentation

In order to estimate the model's capabilities and tweaking the hyperparameters consequently, we have split the data in training-set and validation-set. During the development a ratio of 80/20 turned out to be the best choice for the splitting.
Because of the relative small amount of training data, *image augmentation* was performed. In particular, given the nature of the training images, we considered worthwhile to apply **Random Flip**, **Random Rotation**, **Random Brightness Adjustment**, **Random Zoom** and **Random Shift** since, all these operations wouldn't have changed the meaning of the data regarding this classification task.
During development, we also tried, to use *Oversampling* and *Downsampling* in order to try to solve the problem of class imbalance by equaling the number of samples for each class but this led to no improvement in the result.
For the models that used Transfer Learning a **preprocess_function** was also used.



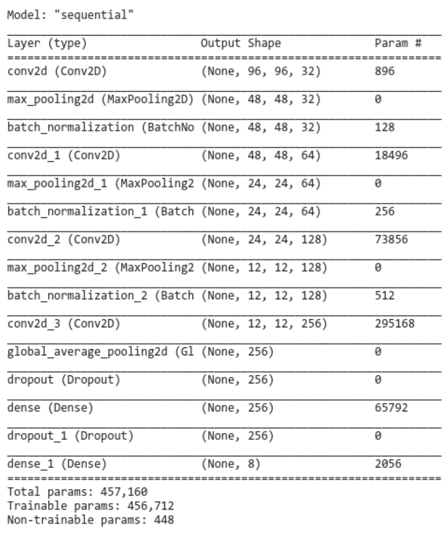
```
train_datagen = ImageDataGenerator(rotation_range=90,
                                   preprocessing_function=preprocess_input,
                                   brightness_range=(0.65, 1.35),
                                   height_shift_range=5,
                                   width_shift_range=5,
                                   zoom_range=0.25,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   fill_mode='reflect',
                                   validation_split=validation_split)

validation_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                        validation_split=validation_split)
```

Figure 1: ImageDataGenerators

First Approach

The first model developed was a simply CNN with four convolutional blocks.
Each block was composed of one convolutional layer with a 3x3 filter, a *ReLu* activation function and a *MaxPooling* layer. The number of filters of the convolutional layers went from 32 in the first block to 256 in the last, doubling it in each block. For the fully connected part we tried with both 256 and 512 neurons, experimenting with different number of hidden layers. But, at the end, just a single hidden layer with 256 neurons showed the best result.
Between the convolutional part and the fully connected part we used a *Global Average Pooling* layer instead of a *Flatten* layer in order to improve the generalization capabilities of the model.
This first approach lead us to an initial accuracy of ≈ 0.60 and, by adding *BatchNormalization* layers and tweaking the parameters, we've managed to reach firstly an accuracy of ≈ 0.70 and, in the end, an accuracy of ≈ 0.77 with both the provided and the complete dataset.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 96, 96, 32)        896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 48, 48, 32)        0         
_________________________________________________________________
batch_normalization (BatchNo (None, 48, 48, 32)        128       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 48, 48, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 64)        0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 64)        256       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 128)       73856     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 128)       0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 128)       512       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 256)       295168    
_________________________________________________________________
global_average_pooling2d (Gl (None, 256)               0         
_________________________________________________________________
dropout (Dropout)            (None, 256)               0         
_________________________________________________________________
dense (Dense)                (None, 256)               65792     
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 2056      
=================================================================
Total params: 457,160
Trainable params: 456,712
Non-trainable params: 448
```

Figure 2: One of the model of the First Approach

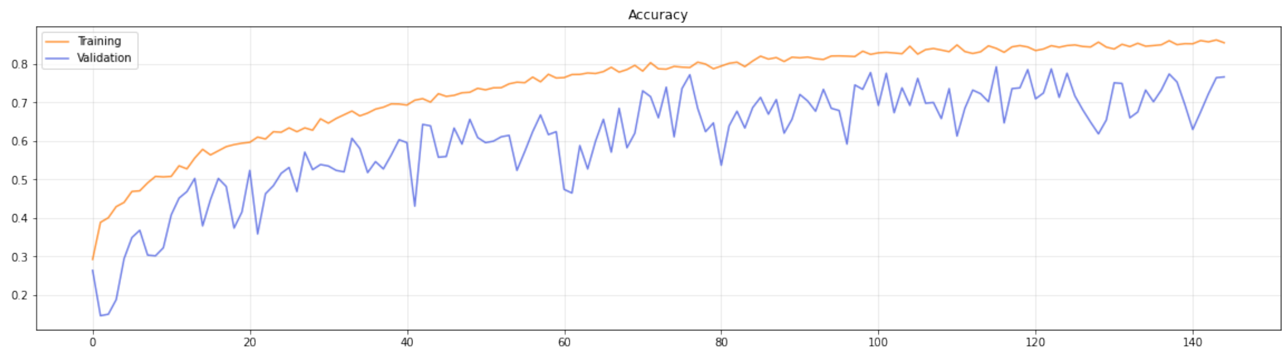


Figure 3: Accuracy in the first approach

Transfer Learning

Getting no more substantial improvements, we decided to change strategy and perform *Transfer Learning*. After experimenting with *VGG16*[1], which lead us to an accuracy score of ≈ 0.81, we went with *Inception_V3*[2].
We imported only the feature extraction layers and added a simple fully connected part, which, at the end, consisted of two hidden dense layers of 256 neurons each and a final output layer. *Dropout* layers were used in between the dense layers in order to reduce overfitting. Input shape was set to (139,139,3).[3]
The training consisted of three, consequent, phases:

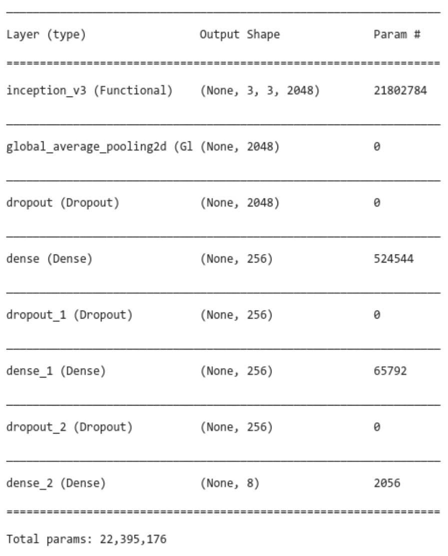
- **Transfer Learning**: In this phase we froze the first 288 layers of the model in order to have a rough initialization of the dense layers and so we trained the model with ADAM's default learning rate. In this phase accuracy reached ≈ 0.70. In the beginning we froze all the layers as seen during the labs, but allowing some of last layers to be trained gave us better results.

- **Partial Fine Tuning**: At this point we unfreeze the layers up to the 215th one, we reduced the learning rate to 8e-4 and retrained the model for less epochs. Accuracy, at this point, reached ≈ 0.75.

[1] `https://arxiv.org/abs/1409.1556`
[2] `https://arxiv.org/abs/1512.00567`
[3] During development, we wrongly read that for inceptionV3 "width and height should be no smaller than 139" but, actually, the keras documentation says "width and height should be no smaller than 75". However (139,139,3) gave us better results

- **Full Fine Tuning**: In this last part we unfreezed the entire model and reduced the optimizer's learning rate to 1e-5 in order to just adapt the pretrained features to the specific task and avoid too much overfitting. We were able to get an accuracy of ≈ 0.84

During all this three training phases *Early stopping* on validation loss was used in order to prevent overfitting and to reduce the training time.
Since the dataset was imbalanced, we also used the *class_weight* attribute to try to solve this problem but this didn't give us the desired outcome.
This model turned out to be pretty much robust during both Development and Final phase of the competition achieving the same performance.



```
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
inception_v3 (Functional)   (None, 3, 3, 2048)        21802784
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)             0
_________________________________________________________________
dropout (Dropout)           (None, 2048)              0
_________________________________________________________________
dense (Dense)               (None, 256)               524544
_________________________________________________________________
dropout_1 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_1 (Dense)             (None, 256)               65792
_________________________________________________________________
dropout_2 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_2 (Dense)             (None, 8)                 2056
=================================================================
Total params: 22,395,176
```

Figure 4: Summary of the Final Model

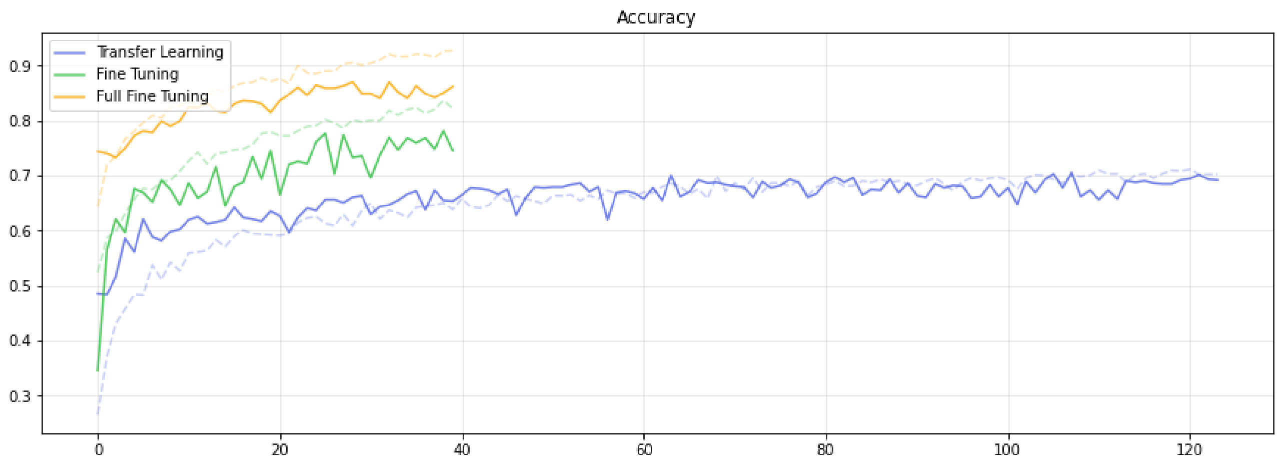


Figure 5: Accuracy of the Final Model

Conclusion

For the development we mainly use Kaggle in order to take advantage of the free GPU power and so to reduce the training time.
In order to analyze the models and take decisions accordingly, we used the plots of the accuracy and the loss over the epochs to understand better what was going on.