# Facial Recognition Project
# Machine Learning Course - DTM

Marco Furno
*Digital Transformation Management Second Cycle*
*Alma Mater Studiorum Università di Bologna Campus di Cesena*
*Email: marco.furno@studio.unibo.it - Telephone: 339 3610870*

***Abstract*—The project involves the analysis of faces and their attribution to two distinct classes identified as ”man” and ”woman”.**

## 1. Introduction

During the Machine Learning course we saw how it is possible to apply some techniques to classify images. We have used techniques that could reflect the properties of images, such as the recognition of some geometric figures starting from the knowledge of Cartesian coordinates, up to the interpretation of an image through sophisticated machine learning techniques, using the convolutional neural network. In this document I have decided to use three of the techniques we have studied and apply them to facial recognition, attributing the images to two different classes ”man” and ”woman”, this because we will use a supervised image dataset whose label distinguishes these two possible classes. I applied these three different techniques:

1) Machine Learning with application of features, here I chose ”HOG”;
2) Deep Learning with Neural Network training;
3) Deep Learning, with the application of Convolutional Neural Networks.

I tried to apply these three techniques on a total image database of 18,978 images of which 9,489 of ”man” and 9,489 of ”woman”.
All the images used were in JPG format, and before their use they were all transformed into a size of 244 x 224 pixels and the RGB color format was used, except in the first case, where we used grey-scale images.

## 2. Machine Learning with feature ”HOG”

For the analysis using Machine Learning techniques combined with the extraction of features from the images, I used a Python program capable of reading all the images, these were divided into three categories to which a different percentage of images was assigned, previously shuffled:

1) training 70% corresponding to 13,284 values
2) validation 20% corresponding to 3,796 values
3) testing 10% corresponding to 1,898 values

In this analysis, Support Vector Machines (SVM) is used to train a model to classify whether from a facial image it is possible to distinguish elements that belong to the class ”man” or to the class ”woman”. The Histogram of Oriented Gradients (HOG) is used as the feature method representation. I am aware that this type of approach should be used with a volume of data in the order of a few hundred, but I am interested not only in the accuracy of the result, but also in the calculation times of the algorithm.
Applying the HOG function to the entire dataset it took 31 minutes, returning its transformation value of the analyzed images, for each of the training, validation and testing splits. Subsequently, through the **.fit()** method we asked the classifier to learn from the images belonging to the training set. This took 120 minutes to process.
We then launched the **.score()** method on the validation data, to verify the goodness of the model. It took 31 minutes to process with a validation score of 0.84.
Finally with the **.predict()** method we tested the goodness of the model in recognizing the two classes, within the images selected in the test phase. It took 15 minutes to process.
The final check in the accuracy of the result, returns 0.86, with the following confusion matrix with absolute value:

$$ConfusionMatrix = \begin{pmatrix} 807 & 117 \\ 155 & 819 \end{pmatrix} \tag{1}$$

which in relatives values can be expressed as:

$$ConfusionMatrix\% = \begin{pmatrix} 0.87 & 0.13 \\ 0.16 & 0.84 \end{pmatrix} \tag{2}$$

the total time taken to obtain these results was approximately 4 hours

## 3. Deep Learning with Neural Network training

For the analysis using Deep Learning techniques, I divided the images into two folders:

1) the ”faces” folder with 17,080 images, where the tf.keras.preprocessing.image dataset from directory function will take the images to be used as training
2) the ”faces test” folder which contains the images that I used in the evaluation phase of the model

In the TensorFlow package, with the function tf.keras.preprocessing.image dataset from directory it is possible to insert the value of the parameters useful for the treatment of the images, in particular:

- directory=’faces’, the folder with the images,
- labels=’inferred’, labels are generated from the directory structure,
- label mode=’categorical’, labels are encoded as a categorical vector,
- class names=[’man’, ’woman’], to explicit the list of class names,
- color mode=’rgb’,
- batch size=32,
- image size=(224, 224), Since the pipeline processes batches of images that must all have the same size, this must be provided,
- shuffle=True, to mix all the images,
- seed=1821,
- validation split=0.2, fraction of data to reserve for validation, the other 0.8 is used as training set,
- subset=’validation’, it could be training or validation, it depends on the use of the function,
- interpolation=’bilinear’, used when resizing images.

Since the total data for training and validation were n. 17,080 and after shuffled, they were divided in this way:

1) training 80% corresponding to 13,664 values
2) validation 20% corresponding to 3,416 values

while the values for the test are contained in the ”faces test” folder and are n. 1,898 of which n. 949 for the class ”man” and n. 949 for the class ”woman”.

Now it is possible define the architecture model of our Multi Layer Perceptron (MLP). We create:

1) 1 input layer with 128 neurons
2) 1 output layer with 2 neurons (our classes)
3) 2 hidden layers with 64 ans 32 neurons, respectively

It is possible to create it, using some defined function of TensorFlow:

1) Normalization(): normalize input data
2) Flatten(): flattens (2D $\rightarrow$ 1D) the input
3) Dense(): just your regular densely-connected NN layer

Through the **.fit()** method the stratified accuracy values for each epoch were calculated. I chose a number of epochs of 10 obtaining a model accuracy equal to 0.5.
I used the model defined in epoch 10 as a template for the test images.
The time required to carry out all the steps was approximately 25 minutes. The longest time was used by the **.fit()** function for calculating epochs and accuracy.

The result is not satisfactory, as I obtain a trend of the accuracy curve which after the $5^{th}$ epoch, which presents an accuracy of 0.78, drops down to an accuracy of 0.5 at the $10^{th}$ epoch. I tried to use the results of the model calculated at epoch $5^{th}$, obtaining the following results:

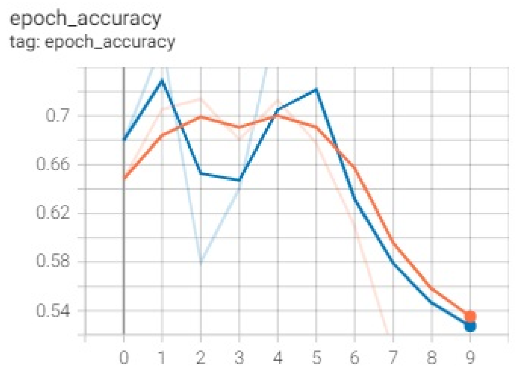

Figure 1. MLP accuracy from epochs

I have calculated the confusion matrix and report its absolute and relative values below.

$$ConfusionMatrix = \begin{pmatrix} 949 & 0 \\ 949 & 0 \end{pmatrix} \tag{3}$$

which in percentage terms can be expressed as:

$$ConfusionMatrix\% = \begin{pmatrix} 1.0 & 0 \\ 1.0 & 0 \end{pmatrix} \tag{4}$$

I was unable to identify the reason for this anomalous behavior, I tried to modify the ”man” class by putting the ”uomo” class and changing the name of all the files of the class, to verify that there was not a ”bag” related to the same ”man” suffix as the two classes, ”man” and ”woman”, but the result hasn’t changed.

## 4. Convolutional Neural Network - CNNs

For the analysis using Deep Learning techniques combined with the Convolutional Neural Networks, I divided the images into three folders:

1) training 70% corresponding to 13,284 values
2) validation 20% corresponding to 3,796 values
3) testing 10% corresponding to 1,898 values

The heart of the algorithm used for the CNNs is the use of the CustomDataGen, Python class constructor, which has as argument the function: tf.keras.utils.Sequence. As reported in the instructions of the web page dedicated to

TensorFlow, this function contains sequence of statements, *"every Sequence must implement the* getitem *and the* len *methods. If you want to modify your dataset between epochs you may implement* on epoch end*. The method* getitem *should return a complete batch. Notes: Sequence are a safer way to do multiprocessing. This structure guarantees that the network will only train once on each sample per epoch which is not the case with generators."*

Another important element used in the algorithm is the function: tf.keras.applications.MobileNet in this function it is possible to insert some parameters useful for the analysis of the dataset, but, specifying the **weight**, parameter with **'imagenet'**, it allows to use a pre-training on ImageNet, useful for improving the quality of the final result.

I chose a number of 10 epochs obtaining a model accuracy equal to 0.93, I used the model defined in the $10^{th}$ epoch as a template for the test images. The time required to carry out all the steps was approximately 20 minutes. The longest time was used by the **.fit()** function for calculating epochs and accuracy.

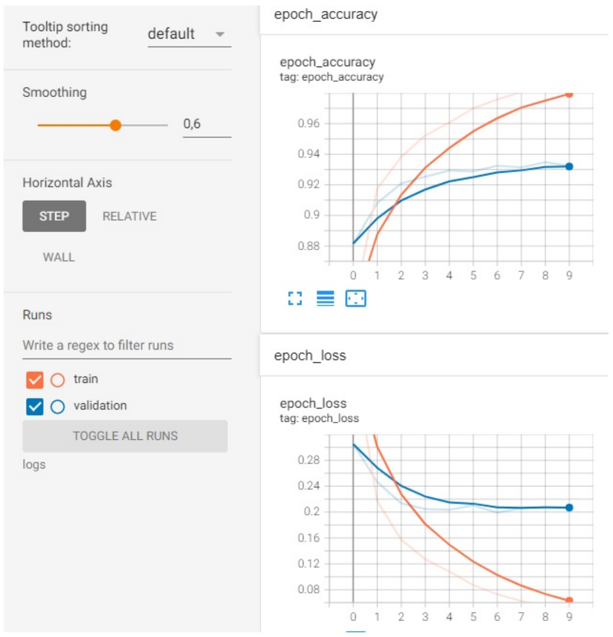

Figure 2. MLP CNNs accuracy and loss from epochs

I have calculated the confusion matrix and report its absolute and relative values below.

$$ConfusionMatrix = \begin{pmatrix} 842 & 72 \\ 76 & 908 \end{pmatrix} \quad (5)$$

which in percentage terms can be expressed as:

$$ConfusionMatrix\% = \begin{pmatrix} 0.92 & 0.08 \\ 0.08 & 0.92 \end{pmatrix} \quad (6)$$

## 5. Conclusion

The three different classification techniques used to create a facial recognition model and test its goodness, have led to the following results in terms of time taken and accuracy achieved.

| Summary of values | | |
|---|---|---|
| Description | Accuracy | Processing time (min.) |
| Machine Learning with feature "HOG" | 0.86 | 240 |
| Deep Learning with Neural Network training | 0.50 | 25 |
| Convolution Neural Network - CCNs | 0.92 | 20 |

From this result we can see how the use of CNNs has been for this dataset, represented by supervised images, an effective tool both for the definition of the accuracy and for the time taken to build the model. I still have the doubt of understanding what went wrong in the use of Neural Network training, as the accuracy achieved is equivalent to that of a coin toss.

## References

[1] https://scikit-learn.org/stable/

[2] https://www.tensorflow.org/?hl=it

[3] https://it.overleaf.com/