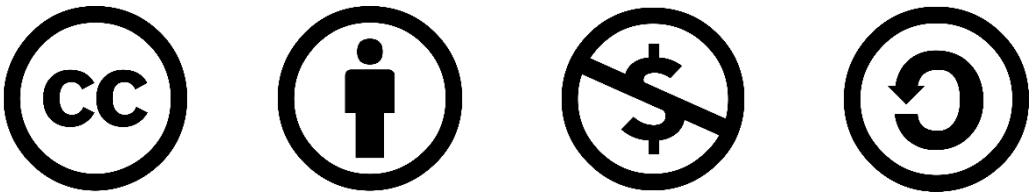


Report Machine Learning
Painter Style Recognition
A.A. 2021/2022

Lecturer:
Pasquale Foggia - pfoggia@unisa.it
Diego Gragnaniello - dgragnaniello@unisa.it
Mario Vento - mvento@unisa.it

Students:
Avitabile Margherita – 0622701825 – m.avitabile6@studenti.unisa.it
Battipaglia Lucia - 0622701758 - l.battipaglia6@studenti.unisa.it
Canzolino Gianluca - 0622701806 - g.canzolino3@studenti.unisa.it
Farina Luigi - 0622701754 – l.farina16@studenti.unisa.it

Summary

1 Introduction

1.1 Overview

Painter style recognition project aims to recognize photos of paints of three different painters: Caravaggio, Manet and Van Gogh, using Convolutional Neural Networks and Fine Tuning to train a model to be able to classify the author of the painting. To do this, it was necessary to build a dataset that was as complete, general and robust as possible. The task to address involved the analysis of scenes representing different genres (religious subjects, still life, landscapes, etc…) and pictures included the frame, the background, and the acquisition by different angles. Finally, it was added images with some over imposed occlusions (e.g., text). Below, we report the analysis of the dataset and its division, the architecture chosen, and the results obtained. Also, we study the main techniques used for the classification of images and the countermeasures used to improve performance.

1.2 Hardware and software environment

This chapter contains a detailed description of all hardware and software we used for the development of the project. In particular, the hardware components offered by Colab and the framework will be described.

1.2.1 The machine

All our work has been done on a machine offered by Google Colaboratory. Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser. The machine is composed of:

1. Storage: 166 GB

2. RAM: 13 GB

3. CPU: Intel® Xeon® 2.20 GHz

4. GPU: NVIDIA Tesla P100: 9.52 TFLOPs, 16GB VRAM

1.2.2 Framework

The major components of the project are Python and Colab. Python is a general-purpose programming language, while Colab is a client-server application that allows editing and running notebook documents via a web browser. Colab Notebooks are documents that contain both code (e.g., python) and rich text elements (paragraph, equations, figures, links). Notebook documents are powerful because they are human readable documents that contain descriptions and results, as well as executable documents which can be run to perform computations. All the work is based on Python, which has been chosen because it is the language of choice of the deep learning community and because it has solid scientific libraries. This allows us to experiment solutions and iterate extremely fast.

The main Python libraries used in this work are:

- ***NumPy***: The fundamental computing library of Python. It adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical routines.
- ***Matplotlib***: A library to plot 2D graphics.
- ***SKLearn***: A library used to evaluate the performance of the Neural Network.
- ***OS***: A library used to perform operative system operations.

Deep learning libraries used are Keras and Tensorflow.

TensorFlow is an open-source software library for numerical computation that uses data flow graphs. Tensorflow allows deploying computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

Keras is a high-level neural networks interface compatible with the TensorFlow and CNTK backends. It was developed with a focus on facilitating fast experimentation. Keras was chosen because it is the "lingua franca" of deep learning, meaning that we can define a model with keras and then convert it to different frameworks that may work better in production.

2 Data collection

2.1 Type of data

The first step was to prepare the dataset, which is the source of information for the classification problem. For that, we searched and downloaded the images representing the paintings of the requested artists and divided them into their respective folders trying to have a number of images per author as fair as possible. They all have different dimensions from each other, subsequently we took care of the resizing to standardize them to 224x224 pixels (which will be performed at runtime).

We tried to get a dataset that covered the painter production as much as possible, that it was generic, so as to prepare the model to classify images of paints not included in our training set, and that it was robust, i.e. not affected by frames, backgrounds, occlusions, etc ...

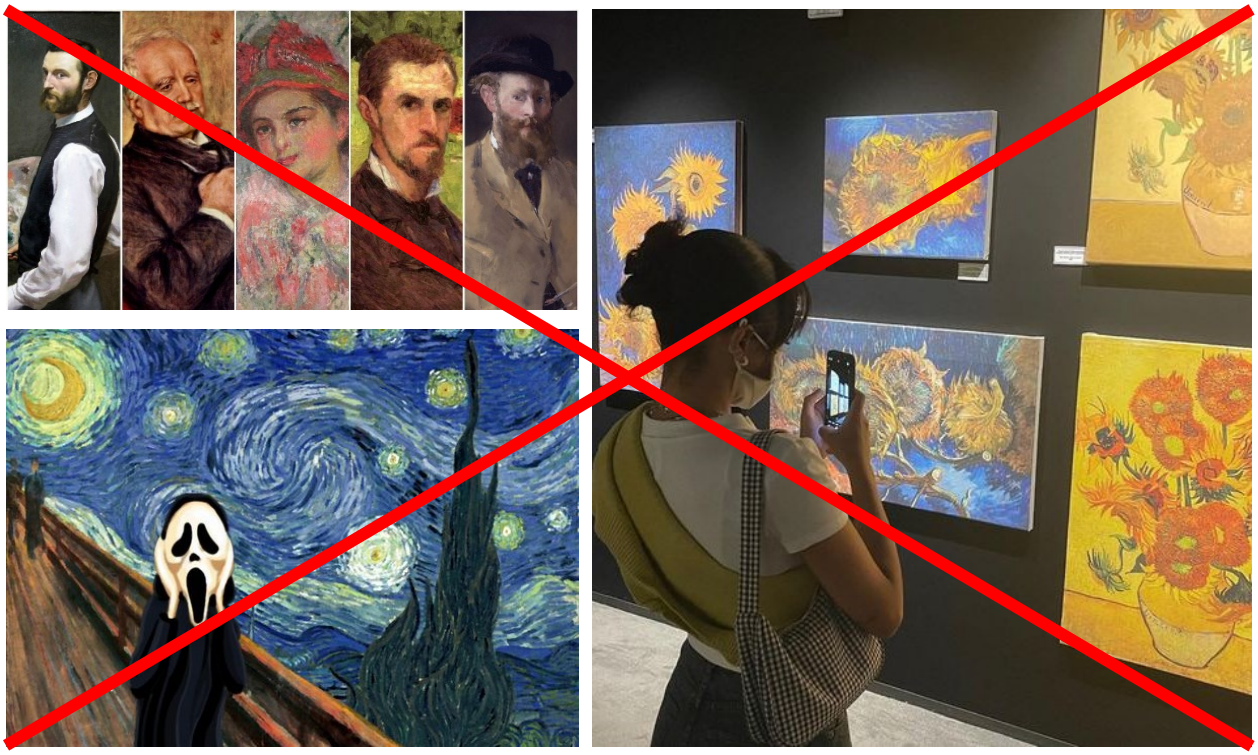
Furthermore, an attempt was made to obtain as many images as possible and depict the different painted subjects, for example religious subjects, still life, landscapes, people in different poses, etc. In addition, we have also tried to obtain different perspectives of the same image, for example by including the frame or not, seeing it from different angles, including some over imposed occlusions (e.g., text). We also considered the case in which not all the painting was visible due to the presence of people.

2.2 Source of data

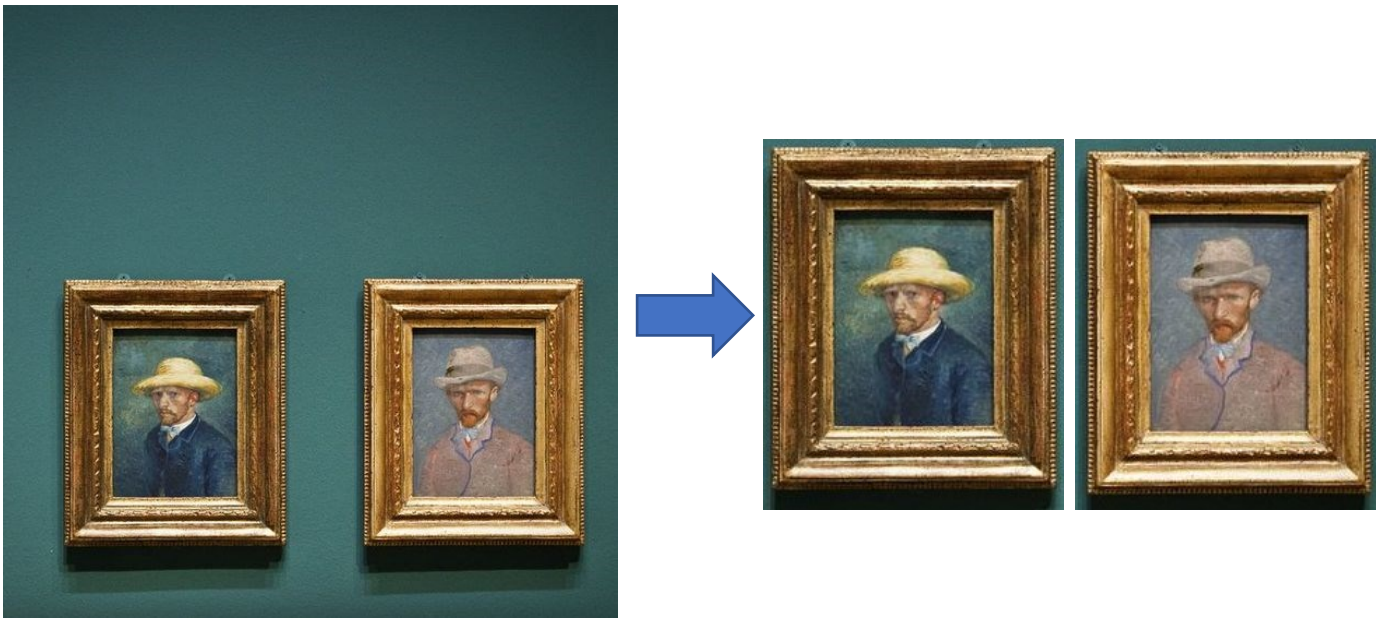
The main data sources for this project are: Kaggle, WikiArt, Getty Images, Pinterest, Dreamstime and Google Images. A tool that was useful for getting images quickly was the google extension “Image Downloader”.

2.3 Data exploration and data cleaning

At the end of the data collection, a phase of exploration of all the images was carried out to evaluate their quality. In fact, all unsuitable images have been discarded such as duplicate images, images with multiple paintings or with subjects covering most of the painting, modified imitations and objects that do not represent a full painting.

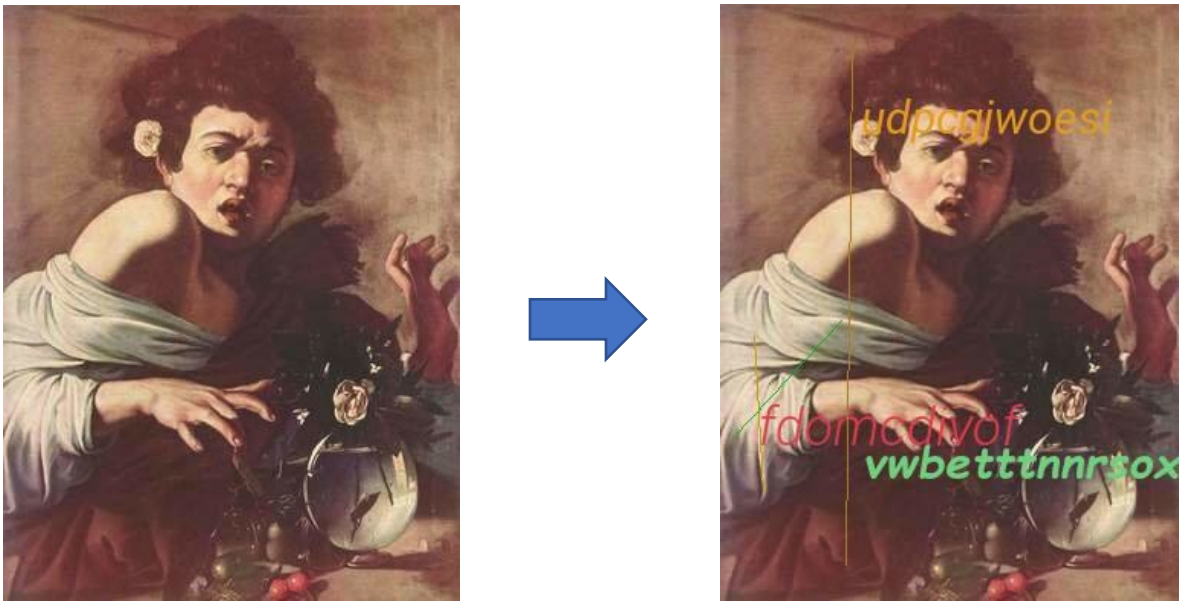


Other images, however, have been adapted. For example, as follow, some images in which the painting represented only a small part of the whole image were cropped in such a way as to obtain a more representative image of the painting.



2.4 Augmented data set

A python code has been implemented to add lines and writings of random sizes and colours to the images obtained. In this way we increased both the size of the data set by about 50% and the images with occlusions (since we had found very few samples of this type), reaching a total of 6517 samples. This was done to increase the robustness of the network, so that if the images that are provided are disturbed by the presence of lines, etc., it should still be able to identify them.



2.5 Dataset Splitting: Training and Validation Set

On the cleaned and complete dataset, for each author, it is possible to divide the images into the following macro-categories:

- Front paintings;
- Paintings acquired from different angles;
- Framed paintings;
- Paintings with subjects that partially occupy the scene;
- Paintings with text and artifacts.

An accurate analysis made it possible to assign each image of the dataset to the respective category.

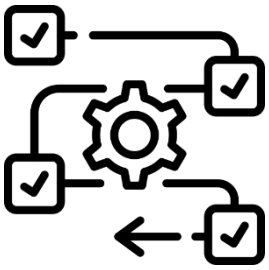
After this classification, the images of each category were divided into 80% for the Training Set and into 20% for the Validation Set. The subdivision was made to have complete independence between the Training Set and Validation Set images. Having dependent images between the two sets would correspond to have unreliable results because the model has already been able to process that specific image, learning its specific characteristics.

The next step was to combine the Training Set and the Validation Set for each author in another Training Set containing the 80% of Van Gogh, Manet and Caravaggio's paintings with those categories, and in another Validation Set containing the 20% of Van Gogh, Manet and Caravaggio's paintings with those categories. This division with high degree of accuracy allowed to obtain a Validation Set as robust as possible and to improve the performance consequently.

3 Image pre-processing and data augmentation

3.1 Pre-processing

Every image can be represented as a set of matrices (tensor) of pixel values. Each matrix corresponds to a channel, the conventional term used to refer to a colour component of an image. An image from a standard digital camera has three channels: red, green, and blue. Each channel is a 2d matrix having pixel values in the range [0,255]. To feed the image to the classifier it is necessary to perform different operations that can include the grayscale conversion, the resizing, and the value scaling. In particular, a pre-processing pipeline was adopted, resizing our dataset to 224x224 pixels, and using a normalization level that normalize our dataset between -1 and 1.



3.2 Data Augmentation

Deep neural networks need millions of images to be trained. Unfortunately, this is rarely the case because collecting and annotating a large number of images is expensive. This problem is usually solved by generating new images through simple transformations that do not change the semantic class of the image. This approach is called data augmentation.

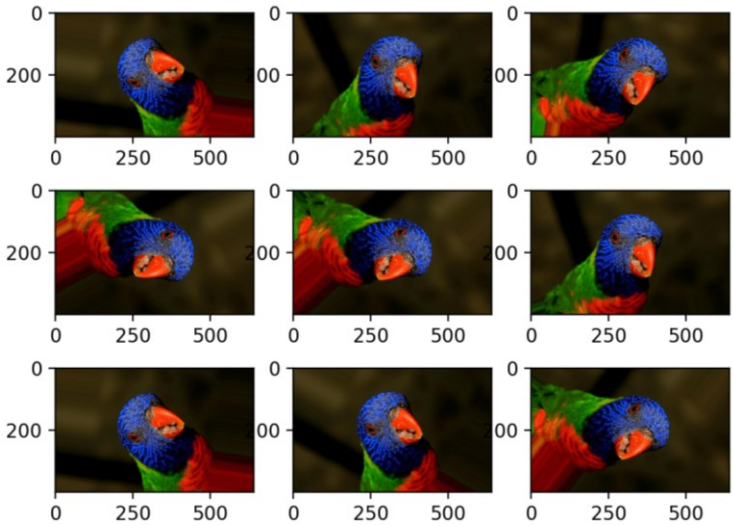
Our solution is to do the data augmentation online during the training process. When the training starts, at each iteration the dataset is shuffled and the batches that will be used for the training are generated. Each batch is processed sequentially: the process reads the images and then apply the augmentation pipeline. When the list of batches of a process ends, the process restarts, shuffling the dataset and generating the new batches. This solution is at the same time simple and efficient and allows us maximum flexibility.

3.3 Image transformations

For what concerns the data augmentation process, it is a rather old technique that tries to make a model more robust by generating new samples. Samples are generated from existing ones with procedures that are domain dependent (image, text, speech), but in general, the augmented data needs to be part of the distribution we want. From a probabilistic point of view, the model is trying to approximate a distribution, and the lack of samples limits us from achieving this. By augmenting the data, we can extract new samples from the distribution, with the hope that they can ease our work. The most important thing that should always be considered is that the generated samples should be part of the distribution that we want to model. If we generate data different from the real distribution, we end up approximating a different distribution, different from our target and the model will not work in deployment. The transformation that was performed for our problem are reported below.

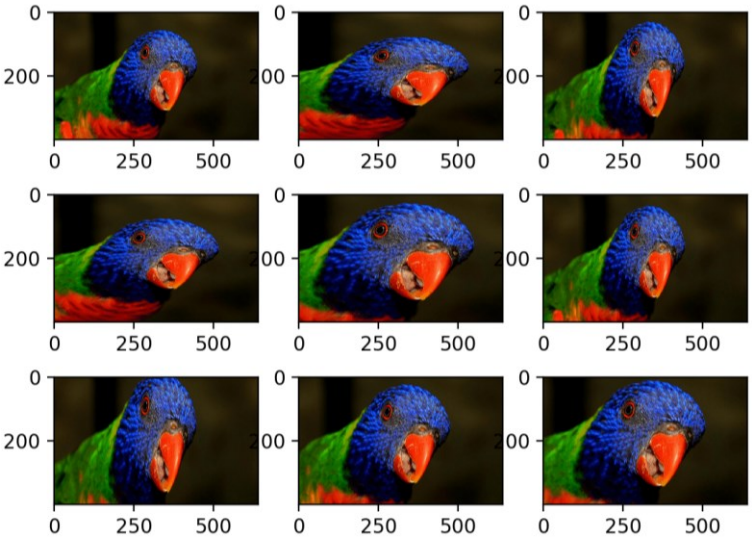
3.3.1 Rotation

Regarding the implementation of the rotation augmentation, we do not perform the augmentation over 360 degrees. Instead, we rotate it by 90 degrees at maximum.



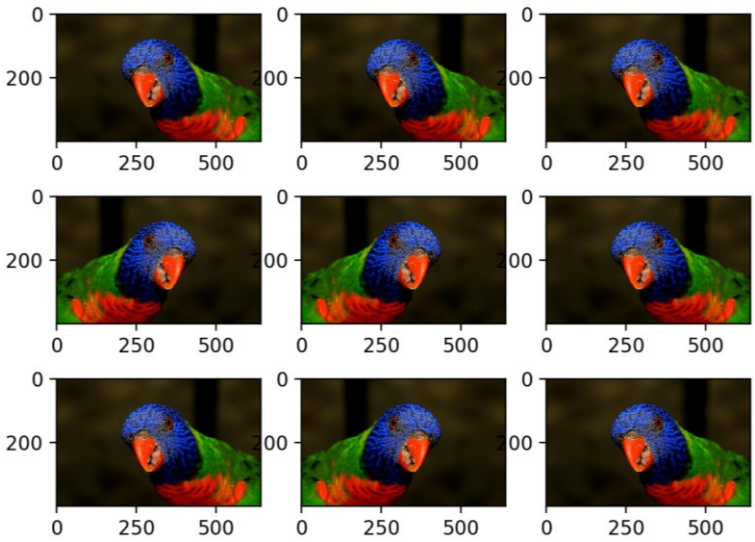
3.3.2 Zoom

Regarding the implementation of the scaling, the scaled image will have a scale between 80% and 120% of the original image. Whenever the scale is less than 100% we have to add a padding to the image.



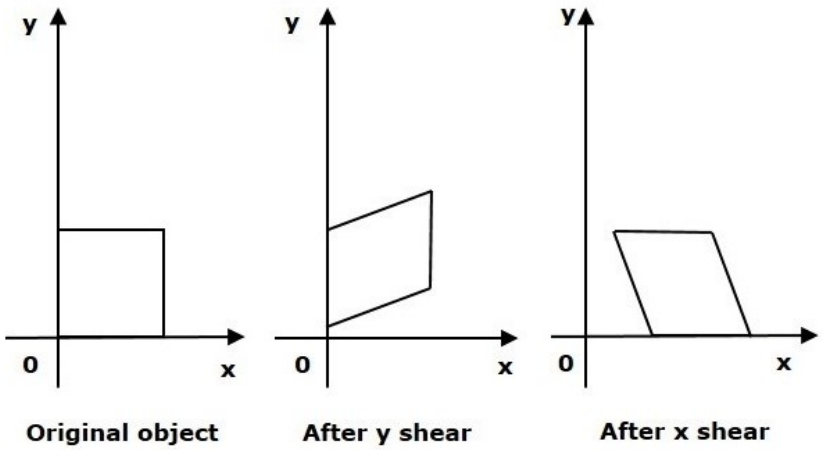
3.3.3 Flip

Regarding the implementation of the flip, the image can only be flipped horizontally.



3.3.4 Shear

Regarding the implementation of the shear, it means that the image will be distorted along an axis, mostly to create or rectify the perception angles. It's usually used to augment images so that computers can see how humans see things from different angles. The modified image will have a shear with maximum range equal at 0.2.



4 Neural Network Architecture

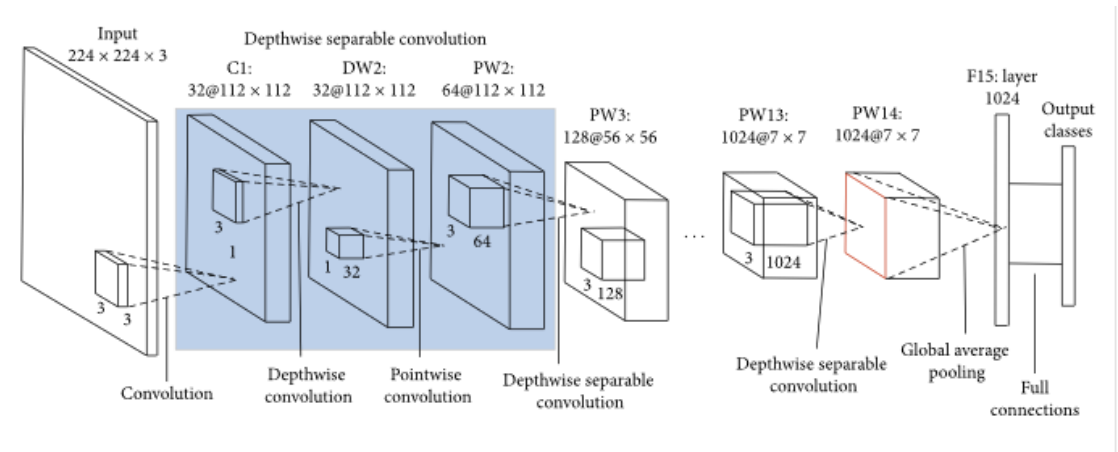
Regarding the neural network architecture, it was decided to use an approach based on fine-tuning, as the data that were collected were not enough. In particular, the fine-tuning approach, unlike the transfer learning technique, consists in using only a part of the weights of the pre-trained neural network, this allowed to reuse a good part of the layers, that detect more basic features, who have been trained on the imagenet dataset. The final part of the deep layers has been retrained, to allow the network to identify features that were more suitable for solving our problem.

4.1 MobileNet

Initially, we started with an architecture based on ResNet which, being a very deep neural network, takes advantages of one of the advantages of deep learning, that is, having more deep layers increase the performance. A second advantage it offers is to implement skip connections which allow to reduce the degradation problem. However, it presented problems of overfitting and two approaches were tried to reduce it, such as the introduction of Dropout layers and data increasing. The performance has increased reaching about 82 % on the validation set but, despite this, the network still presented some overfitting problems. Other architectures such as Inception and Xception have also been tried but performance has not improved. It was therefore decided to change the basic architecture and therefore the pre-trained neural network, trying to adopt a solution that was simpler as the previous architectures were very complex. So, it was decided to use a simpler pre-trained neural network that was able to work well with the collected dataset. The final architecture that was chosen is based on MobileNet.

4.1.1 Architecture

MobileNet is an efficient architecture, that is widely used for artificial vision applications on mobile devices such as smartphones and tablets. MobileNet is based on the concept of depthwise separable convolution that allows the creation of smaller and faster deep neural networks without sacrificing performance. The depthwise separable convolution divides the convolution into two parts: the first one named depthwise convolution, which applies a single filter to each channel thus obtaining the intermediate features, the second one named pointwise convolution, which performs a linear combination of the intermediate features. The depthwise separable convolution allows to greatly reduce the number of weights of the network allowing to have a simpler model, which therefore manages to work better even with few data. Furthermore, it also allows for faster computation. MobileNet also reduces the problem of degradation as there are BatchNormalization layers inside it.

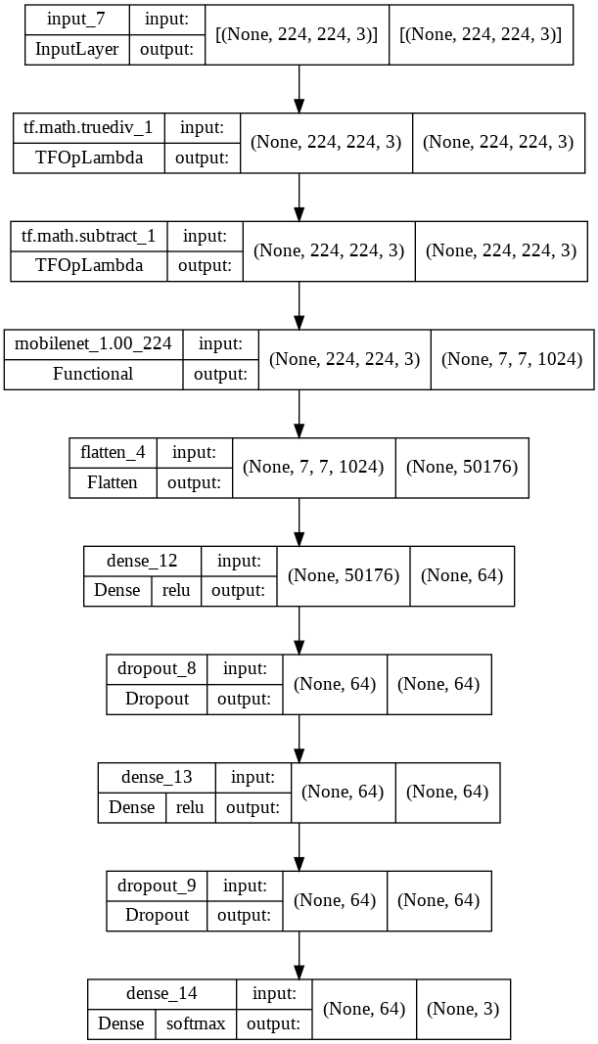


4.2 Final model

Regarding the final model, the fine-tuning method was used. It was used the MobileNet neural network to compute the feature vector for our problem and finally we added a set of Dense layers and Dropout layers to perform the classification operation. The remaining deep layers of the MobileNet neural network and the top layers were trained.

4.2.1 Architecture

The final model is composed by four levels: the first level, Input, is the input layer that contains the batch of data we want to pass to our neural network, the second level named pre-processing, representing by the second and third layers of the neural network performs a normalization operation, given a batch of data in input, this layers normalize the batch of data between -1 and 1. This level is important because the MobileNet was trained with images normalized between -1 and 1, so these layers allow to use better our pre-trained neural network. The third level is the MobileNet that, given a batch of normalized images, computes the feature vector. Finally, the fourth level, representing by a set of Dense and Dropout layers, performs the classification operation. The feature vector was vectorized by using a flatten layer, allowing to adapt our feature to the Dense layer. It was chosen an input dimension of 224x224 to use better our pre-trained neural network because the MobileNet was trained with images 224x224.



4.2.1.1 Model summary

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_7 (InputLayer)         [(None, 224, 224, 3)]     0

 tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)       0
 da)

 tf.math.subtract_1 (TFOpLam  (None, 224, 224, 3)       0
 bda)

 mobilenet_1.00_224 (Functio  (None, 7, 7, 1024)        3228864
 nal)

 flatten_4 (Flatten)          (None, 50176)             0

 dense_12 (Dense)             (None, 64)                3211328

 dropout_8 (Dropout)          (None, 64)                0

 dense_13 (Dense)             (None, 64)                4160

 dropout_9 (Dropout)          (None, 64)                0

 dense_14 (Dense)             (None, 3)                 195

=================================================================
Total params: 6,444,547
Trainable params: 5,347,139
Non-trainable params: 1,097,408
```

4.2.2 Training

Regarding the training of the neural network, two generators were used, one for the training set and one for the validation set. This approach has the advantage of not being limited to the amount of memory we have on our machine because data are taken directly from the disk. Another advantage they offer, is that of being able to directly resize data to adapt them to our model. Within the training set generator, transformations to be carried out on the data have been specified in order to perform data augmentation during training. Callbacks were also used, that allow to save the model with the best accuracy on the validation set and to save all training information in order to generate a final graph with Tensorboard.

4.3 Hyperparameters

4.3.1 Activation functions

The activation functions that have been used are mainly two: the ReLu activation function, used for hidden layers which allows to reduce the vanishing gradient problem allowing to have a faster training for our model and the Softmax activation function, used for the output layer, which makes the sum of the probabilities provided by output neurons equal to 1.

4.3.2 Loss function

Since the problem addressed is a multi-class classification problem, Categorical cross-entropy was used as Loss function.

4.3.3 Other Hyperparameters

The other types of hyper-parameters that have been chosen are the batch size equal to 64, the number of untrained layers equal to 60, this allowed a performance increasing, the number of neurons of the Dense layers equal to 64, the probability of the Dropout layers equal to 0.4 and the stochastic gradient descent algorithm chosen is rmsprop.

5 Results

The results obtained are shown below, they include accuracy and error on the Training Set and on the Validation Set.

5.1 Graphics Result

The graphs show the training results for our neural network. They are divided into two parts because two training phases were carried out, the first with 90 epochs, while the second with a further 30.

The graphs show the accuracy on the training set which is 99.11%.

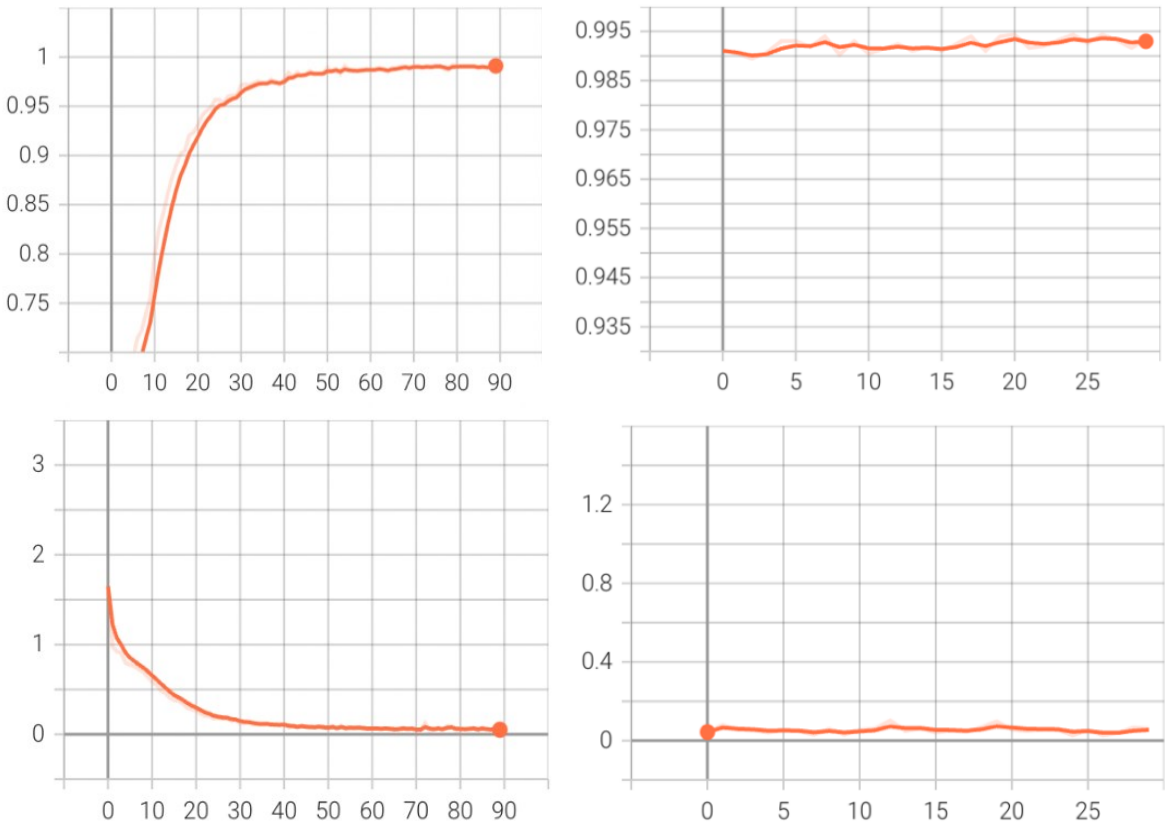


Figure 1 Accuracy and Loss on Training Set

Instead, an accuracy of 96.72% was obtained on the Validation Set (epoch 107).

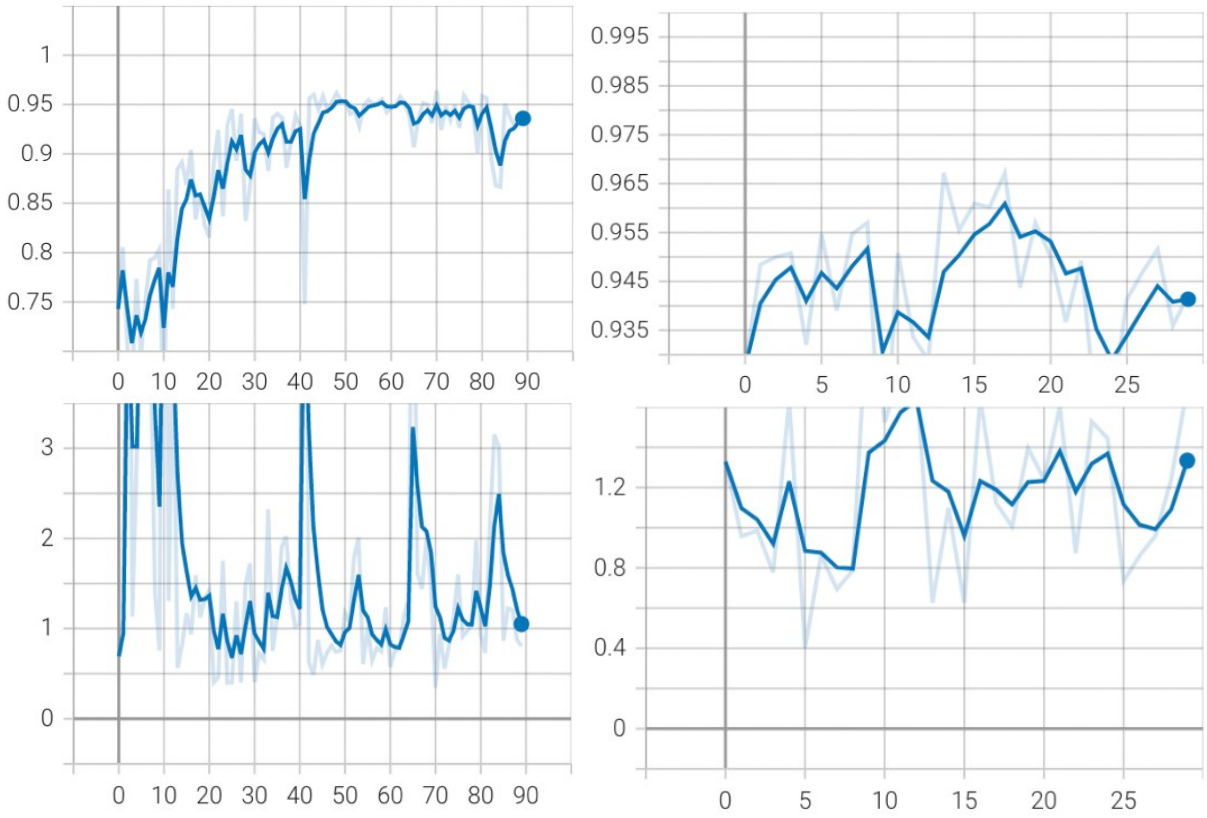
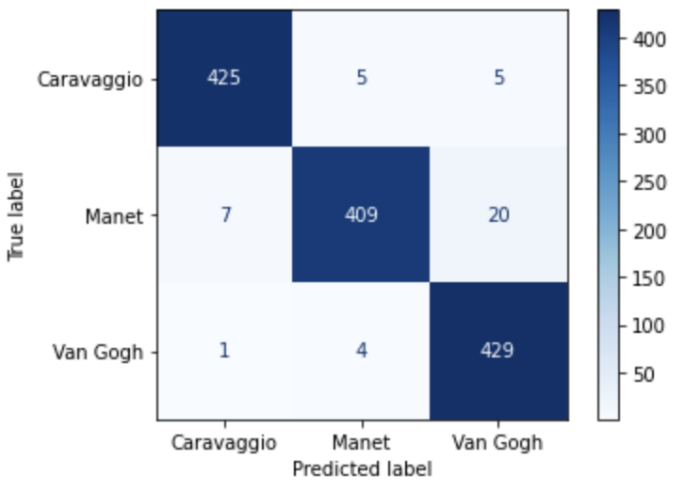


Figure 2 Accuracy and Loss on Validation Set

The results obtained show that there is a slight overfitting of about 2%.

Below is the classification matrix of the neural network on the validation set. From the classification matrix we can easily verify that the performance on the validation set is equal to 96.72%. As we can see, the neural network has a slightly greater error for the author Manet. This result could be due to the fact that in the training set for the author Manet, there are no images that contain the information necessary to correctly classify even the images on which the network fails. This problem can be solved by carrying out an error analysis and trying to increase the data along the error direction.



6 Conclusions and future work

6.1 Objectives and findings

In this project we applied a deep learning-based approach to solve the painter style recognition problem and obtain an accuracy on the validation set equal to 96.72%. This result was achieved thanks to a careful selection of the images to be included in our dataset and to a targeted adjustment and cleaning work. Furthermore, MobileNet, thanks to a considerable reduction of the parameters, has allowed a better work with the collected dataset. However, to arrive at this result several pre-trained neural network architectures have been tested such as ResNet, Inception and Xception whose performance were not satisfactory for the purposes of our problem as mentioned above. Furthermore, in order to use better our architecture, the size of the input images has been adapted to 224x224 and they have been normalized between -1 and 1, this because the MobileNet has been trained with these hyper-parameters.

6.2 Future improvements

As suggested by our results, our network fails more frequently than other artists to classify Manet's paintings. As suggested above, this problem could be solved by increasing the data along the direction's error. The use of bigger dataset should enable to improve our network and should improve the accuracy of our model.