Image Classification using Deep Learning

Marwan Agourram

Department of Computer Science, LM Data Science, Via Celoria 18, Milan, 20133, Italy.

Contributing authors: marwan.agourram@studenti.unimi.it;

Abstract

Two different settings are explored in order to perform image classification on data from the Prado Museum of Madrid. The task is carried out by implementing three different convolutional neural networks, alternating a base model to one with data augmented inputs, and lastly by performing hyper-tuning on a predefined model structure. Finally, an alternative exercise is considered, in which the data setting are slightly modified, and in which the same models have been implemented.

Keywords: Deep learning, Convolutional Neural Networks, Hyper Parameter Tuning

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work, and including any code produced using generative AI systems. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Contents

List of Figures

1 Introduction

This report aims at implementing Convolutional Neural Networks to perform image classification on image data collected from the art gallery of the Prado Museum, Madrid.

In Section 2 the discussion is focused on understanding the issue related to class definition, using a consistent subset of authors as the main reference. While maintaining this objective, a brief analysis of the dataset and a methodology to define classes is discussed.
In Section 3 I provide the details about the implemented models, such as the structure and the parameter values.
In Section 4 the obtained results are discussed. Finally, in Section 4 a conclusion is drawn from obtained results.

In Appendix A an alternative setting is introduced, in which classes are defined by using supports instead of authors. Section A.1 described the main changes in the data setting, while A.3 describes the obtained results.

2 Dataset

The dataset called 'Prado Museum Pictures' is retrieved from Kaggle using Kaggle API and it comes in `.zip` file, which contains two elements:

- `prado.csv`: a file that contains all information related to the images. In particular, in the file are reported information about the web url in which we can display the image, information related to the author (such as name, the unique id `author_id`, or a brief biography), information related to the artwork (the type of technique used, the century in which it has been created), and other 'internal' information related to the artworks (such as the location in which it is exposed). The size of the files is $(13.487, 30)$.
- `images`: This file contains 13.472 images in `.jpg` format. The images where named after their relative `work_id`, which is the unique identifier associated with each artwork.

The most important thing about the dataset is that the suitable structure for implementing image classification techniques is not provided directly. In fact, the first task was to understand how to set up the structure in order to ensure consistency and robustness.
The second task consisted in defining an appropriate organization of classes. For this purpose, the adopted approach consisted in considering the authors name[1] as a baseline for defining classes.

Using Kaggle API, the folder was downloaded directly into Google Colaboratory and then extracted into a separate folder (`/zip_file/`). The aforementioned folder resulted to be structured as follows:

```
/content/zip_file/
├── images/
│   └── images/
│       ├── image_1.jpg
│       ├── ...
│       └── image_i.jpg
└── prado.csv
```

A `pandas.DataFrame` data structure has been assigned to the `.csv` file and then a new column `work_id` has been added to it in order to extract for each row in the dataframe the related image name; Then most of the columns have been removed from the dataframe, leaving just `author, author_id`, and `work_id`.

A brief analysis on the number of artworks related to each single artist has been deployed, which lead to noticing that the most frequent artist in the dataframe was labelled as 'Anònimo', that is, an unspecified artist. Therefore, I proceeded to remove all images associated with the 'Anònimo' id for an obvious reasons: since the approach consisted in implementing Convolutional Neural Networks on the dataset, maintaining a class of images which contains heterogeneous types arts (from paintings

[1] More precisely, I used `author_id` in order to avoid any misspellings or typographic errors. In particular, several `author_id` were associated with different `author` names.

to sculptural art) may increase the chances of obtaining bad results[2]. This leads us with an initial subset of 9.661 images. See Figure 1 for more details.

Then, I arbitrarily decided to maintain images related to authors who had at least 200 images associated with their `author_id`. This reasoning leads to the solution where the only images that will be processed are those related with the authors reported in Table 1. Therefore, the total number of images used in this report is 1.000 and the total number of classes is 5.
Since with the current approach the CNNs will rely on a selected subset of images, the remaining ones are removed in order to save hard disk space on the virtual machine.

author_id	author	total_occurrences
39568a17-81b5-4d6f-84fa-12db60780812	Goya y Lucientes, Francisco de	1097
b218fee4-053b-4656-8577-9aa001ad1989	Bayeu y Subías, Francisco	446
1e7197fc-e941-418b-a0c9-4ce34ce633e9	Haes, Carlos de	326
0b7d7dd0-cb66-4ee0-9f01-1cc526587980	Pizarro y Librado, Cecilio	290
29dffe55-c31e-4737-aca1-f538ebaea0fe	Ribera y Fieve, Carlos Luis de	222

Table 1: The 5 most recurrent authors, their unique id, and the number of artworks related to each one of them.

In order to recreate an appropriate structure that better suits the deployment of convolutional neural networks, a new directory have been created with the purpose of moving images from the directory in which I extracted the image files directly into the correct sub-folder (train, validation, test), which is named after the stage process where the files will be used. Moreover, each sub-folder will then have a specific number of sub-folders, each one named after the related author. This approach will allow us to later on implement specific methods (i.e., `tf.keras.utils.image_dataset_from_directory`) that construct dataset directly from the folder that contains images.

In order to separate the images with respect to their relative purpose, the following ratios have been selected and implemented within the `Sklearn.StratifiedShuffleSplit()` method, which is suggested when the classes of images are initially unbalanced. See Table 2 for more details.

Finally, the obtained result consist in three folders (train, test, validation), each one of them containing five sub-folders (which represents the classes of images, in this case, the considered authors), and each sub-folder contains images related to the specific author/class.

[2]The motive behind this reasoning consists in the fact that an artist is more likely to develop and maintain a certain patter in terms of the produced arts. However, multifaceted artists such as Salvador Dalì have proved the capability to master different disciplines.
From a technical point of view, considering heterogeneous images under the same class makes the process of identifying patterns more difficult.

purpose	ratio	images in folder (purpose)	images in folder (class)
train	0.6	600	120
validation	0.2	200	40
test	0.2	200	40

Table 2: Details about the train-test split implemented in the report.

3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a particular type of neural network that works best when the input data consists of images. In this report, a base model is defined from scratch and is initially implemented on input data, and then it is applied a second time on the input data after data augmentation. Lastly, a final model which hyper-parameters are search and fine-tuned using `Keras Tuner` is implemented.

Before diving deeper into the models, several operations have been performed on the raw image data in order to simplify and enrich the input data. In particular, the first operation performed on the data consisted in generating *TensorFlow* dataset via `tf.keras.utils.image_dataset_from_directory()` method. This method simply load the data (which is located in specific folders) by creating data frames. In particular, images have been resized into a (64×64) pixel size; Mitchell cubic interpolation is implemented (which is a type of cubic convolution kernel known to guarantee balance between sharpness and smoothness); and finally the images have been cropped to the ratio (64×64) to ensure that no distortion happens during the resizing of the images.

Then, data augmentation is defined. Instead of applying the augmentation to all images, the process have been implemented within a layer of the CNN. This will allow us to compare different results also by considering if any augmentation stage is performed[3]. The specific parameters of the data augmentation process are listed below:

- Random horizontal and vertical flips.
- Random zooming with parameters ranging between $(-0.05, -0.15)$ for both height and width. Any empty area generated from zooms are replaced by the nearest values.
- Random contrast, translation (both horizontally and vertically), and rotation.

In the following subsections, a brief description of the structure of each model is provided. For all of the following models, the size of the batches is fixed at 20 as well as the number of epochs. Moreover, an early stopping function is defined in order to reduce the risk of overfitting and reduce the amount of inefficient processing. The stopping function is measured with respect to the values of the accuracy on the validation data, with `patience` set to 10 epochs.

[3]When the data augmentation layer is implemented it will be specified within the description of the model structure.

3.1 Model with no Data Augmentation

The first model is a simple approach to implementing CNNs on the input data. The structure details of the model are listed below:

- Rescale layer with factor $1/255$, and the output data having the size of $(64, 64, 3)$.
- Four convolutional layers (with filter size in increasing order: $16, 32, 64, 128$) and kernel size fixed at $(3, 3)$. Rectified Linear Unit as the activation function is adopted.
- Four maxpooling layers with size $(2, 2)$, after each convolutional layer.
- One flattening layer after the last maxpooling layer.
- Two densely connected layers, the first of them having 128 units, and the last one having 5 units (same as the number of classes).

The model is compiled using Adam as the optimization function with fixed learning rate 0.001, loss function defined to be sparse categorical cross entropy with the parameter `from_logits=True` due to the absence of the softmax activation function in the last densely connected layer[4]; finally, the accuracy is the considered metrics.

3.2 Model with Data Augmentation

The second implemented CNN model has two slightly different changes compared to the model described in Section 3.1. These differences are:

- The size of the filters in the convolutional layers are (in order): $16, 32, 128, 256$
- The first densely connected layer has now 256 units. The same is followed by a dropout layer, with rate fixed at 0.5.

For what it concerns the rest of the architecture as well as for the compiling settings, the model is exactly identical to the aforementioned one.
However, it is important to notice the presence of the Data Augmentation layer at the input layer. This is ensure that the input data is augmented (see Section 3 for the details) before performing any convolutional operation on it.

3.3 Hyper Model

The last CNN model is fine tuned via the adoption of the `Keras tuner` library. By defining the architecture of the CNN and by specifying the range in which the hyper-parameters can be chosen, the library performs several trials in order to detect the best value for each hyper-parameter.
The adopted structure is described below and it returns a hyper model:

- A rescaling layer with factor $1/255$.
- Four convolutional layers with filter size value (an integer) that can range in the interval $[16, 128]$ with step size (the distance between two consecutive samples in the range) fixed at 16. The sampling argument is specified to be `linear` and therefore the sampled value follows the following equation: min_value + sampled_value $*$

[4] By using ReLU instead of SoftMax activation function, we expect the output layer to contain values that range $[-\infty, +\infty]$. If SoftMax activation function were implemented, the obtained result values will range across $[0, 1]$.

(max_value − min_value). The kernel size hyper-parameter is obtained by implementing the `choice()` method, which simply selects the hyper-parameter value across predefined set of possible values (in this case, $[3, 5]$). Finally, the adopted activation function is the ReLU.
- Four MaxPooling layers, each one after a convolutional layer, with fixed size $(2, 2)$.
- A flattening layer after the fourth MaxPooling layer.
- Two densely connected layers. The unit value for the first one is sampled across the interval $[16, 256]$, with step size fixed at 16. The sampling technique is `linear` and the activation function is the ReLU. The second layer has the same unit size as the number of classes (i.e., 5).

For the compile stage, the same specification described in Section 3.1 are adopted.
Once the architecture is specified, the parameters for the fine-tuning process are set: the objective is to obtain optimal results by considering the accuracy metrics on the validation data; this process is performed for at most 10 epochs with a reduction factor of 3.
Finally, the optimization process is combined with an Early Stopping function with patience 5 in order to avoid useless research.

4 Results

In the current section, the obtained results are briefly analysed.

4.1 Model with no Data Augmentation

Figure 2 display the performance of the model without applying any data augmentation layer. Starting with the performance on the train and validation data, on the left of Figure 2 we can see the performance of the loss function. For what it concerns the train data, there is a smooth and decreasing trend, while for the validation data there is an initial decrease (before the tenth epoch), which is then followed by an unstable progress that seems to be increasing at the last epoch. This is a clear sign of overfitting, which is also confirmed by the accuracy plot in Figure 2, since the accuracy on the validation data stops increasing after peaking at the tenth epoch.

The performance of the model on the test data is reported in Figure 3, which display a confusion matrix, as well as in Table 3. In particular, we can notice how several images were mislabelled. However, across the ten selected images from the test data, 7 of them are correctly labeled by the model.

Finally, the model evaluated on the test data returns loss of 0.8201 and accuracy of 0.8050.

class	precision	recall	f1-score	support
0	0.0000	0.0000	0.0000	1
1	0.7500	0.7500	0.7500	4
2	0.5000	1.0000	0.6667	2
3	1.0000	0.5000	0.6667	2
4	1.0000	1.0000	1.0000	1
accuracy			0.7000	10
macro avg	0.6500	0.6500	0.6167	10
weighted avg	0.7000	0.7000	0.6667	10

Table 3: Report showing the main classification metrics obtained by the model.

4.2 Model with Data Augmentation

Figure 4 displays the performance on the train and validation data obtained by the model in which I included the data augmentation layer. Starting from the loss function, there is a constant decreasing trend on the train data before the tenth epoch, which then is followed by a performance that does not improve. This pattern is also replicated on the validation data, which however is less smooth. In particular, after the eighth epoch the model seems to struggle in generalizing new data.
On the other hand, considering the accuracy metrics, we can see that the same (inverted) trend of the loss function is displayed. One should notice that for the last five epochs, the model seems to perform better on the validation data compared to the train.

Figure 5 displays the confusion matrix obtained by running the model on ten images from the test data. Once again, four out of ten images are misclassified and the fifth class lacks any prediction. For more details about the classification metrics, see Table 4.

Finally, the model is evaluated on the test data, returning loss of 0.9700 and accuracy of 0.6150.

class	precision	recall	f1-score	support
0	1.0000	1.0000	1.0000	1
1	0.5000	0.5000	0.5000	2
2	0.6667	1.0000	0.8000	2
3	0.5000	0.6667	0.5714	3
4	0.0000	0.0000	0.0000	2
accuracy			0.6000	10
macro avg	0.5333	0.6333	0.5743	10
weighted avg	0.4833	0.6000	0.5314	10

Table 4: Report showing the main classification metrics obtained by the model.

4.3 Hyper Model

Figure 6 displays the performance of the Hyper model on train and validation data. For the loss, there is a decreasing trend on the train data that is partially matched by the one one the validation data. However, after the tenth epoch, the performance of the model on the validation starts to increase. On the other hand, the accuracy on the train data is smooth and increasing throughout the 20 epochs, while the one for the validation obtains the best results around half of the training.

Figure 7 displays the confusion matrix. Compared to the previous results, this is the worst model in terms of correct assigned labels out of ten images. In fact, 5 out of ten images are wrongly labelled. However, we can see how the fourth class was the one with only correctly labelled images. For more details about the classification metrics, see Table 5.

Finally, the hyper model is evaluated on the test data, leading to a loss of 1.1668 and an accuracy of 0.6300.

class	precision	recall	f1-score	support
0	1.0000	0.3333	0.5000	3
1	0.0000	0.0000	0.0000	2
2	0.5000	1.0000	0.6667	1
3	0.5000	1.0000	0.6667	2
4	0.3333	0.5000	0.4000	2
accuracy			0.5000	10
macro avg	0.4667	0.5667	0.4467	10
weighted avg	0.5167	0.5000	0.4300	10

Table 5: Report showing the main classification metrics obtained by the model.

5 Conclusion

Three different CNN approaches were implemented in this report: starting with a "simple" model that takes input data without applying augmentation, we moved to a second model that had subtle changes in the structure compared to the initial one. Moreover, the data augmentation layer is implemented. Finally, thanks to the Tuner library in Keras, a third hyper model was also tested.

Starting from the performance of the models, the first one performed well in terms of predictions and also in terms of metrics results on the test data, striking the lowest loss and the highest accuracy across the three models. However, as we can see from Figure 3, the model failed to classify the first class, although there was only one image for support.
The model with data augmentation shows a decrease in the performance and fails to classify both the images related to the fifth class. More in details, the performance during the training stage was already signaling the obtained results, since the performance on the validation was not smooth at all, alternating various peaks to troughs.
Lastly, the hyper model, although being tweaked to reach the optimal hyperparameters, obtained the worst performance in this report. The models seemed to be learning to much details in the train data after few epochs, which may be due the structure of the CNN. However, it is also important to notice the the optimization and the fine-tuning were truncated to a small number of trials (less than 25) and also that the model has been deployed for 20 epochs during the training.

There is surely room for improving across the three applications, by modifying the structures or by implementing pre-trained models and then adapting the obtained weights, or simply by increasing the epochs during the training phase. However, the obtained results seems to be quite comfortable, since all of the CNNs were trained using 600 images, validated using 200 images, and tested using 200 images.
The task was challenging since depicting specific patterns that defines an author may require more processing of the provided data. A comparative approach is described in Appendix A, where instead of classifying images by authors, classification by support is instead considered.

A Image Classification by Support

A.1 Dataset

The main approach described in Section 2 is also adopted for this alternative solution. However, the main change consisted in the definition of classes in terms of support (which, in the dataset, is represented in the column `technical_sheet_tecnica`).
A new column `support` is obtained by extracting the main information from `technical_sheet_tecnica`) in order to have the significant types of support that are represented in the dataset (see Figure 8 for more details). From the histogram we can clearly see that several classes have more than 500 images. Therefore, classes are defined depending on the supports that have more than 500 images. In Table 6 are reported the 7 types of support (that will be used as classes) and the related number of images.

technical_sheet_tecnica	support	total_occurrences
Óleo	Óleo	4166
Acuñación	Acuñación	1120
Aguada; Albayalde; Pluma	Aguada	909
Aguada parda; Lápiz negro; Pluma	Aguada parda	681
Clarión; Lápiz negro	Clarión	663
Pluma; Tinta parda	Pluma	633
Esculpido	Esculpido	558

Table 6: The 7 most recurrent types of support and the number of artworks related.

Since the total number of images that we will use throughout the process is 3,500 (7 classes with 500 images each), the number of images per folder after the train-test split is reported in Table 7.

purpose	ratio	images in folder (purpose)	images in folder (class)
train	0.6	2100	300
validation	0.2	700	100
test	0.2	700	100

Table 7: Details about the train-test split implemented in the report.

A.2 Convolutional Neural Network Models

The implemented models are the same ones described in Section 3.

A.3 Results

Model with no Data Augmentation

Figure 9 and 10 reports the results of the CNN model described in Section 3.1 on the data. Starting with the performance on the train and validation data, measured according to the loss and accuracy, we can see that the performance on the train data is smoothly decreasing (increasing) for the loss (accuracy) metrics . This signals that the model is continuously learning on the train data. However, this performance is not replicated on the validation data, since the model fails to generalize on new data: after an initial decrease for the first 10 epochs, the loss functions starts to increase, signaling a worsening of the performance. For the accuracy, after an initial increase, the model stops predicting the correct labels on the validation data.
On Figure 10 we can see how the model performed with respect to correctly classifying 70 images extracted from the test data, while in Table 8 we can see the classification metrics related to the confusion matrix.

classes	precision	recall	f1-score	support
0	1.0000	0.9000	0.9474	10
1	0.5000	0.2857	0.3636	7
2	0.5000	0.2857	0.3636	7
3	0.7500	0.8182	0.7826	11
4	1.0000	0.9167	0.9565	12
5	0.5625	1.0000	0.7200	9
6	1.0000	1.0000	1.0000	14
accuracy			0.8000	70
macro avg	0.7589	0.7438	0.7334	70
weighted avg	0.8045	0.8000	0.7876	70

Table 8: Report showing the main classification metrics obtained by the model.

Model with Data Augmentation

Figure 11 and 12 reports the results of the CNN model described in Section 3.2 on the data. Compared to the previous results, we can see from Figure 11 that the performance on the train and validation data is improved, since both loss and accuracy follows the same trend (decreasing for the loss function, increasing for the accuracy). However, we should mention that for the train data the performance is smoother.
Figure 12 display the confusion matrix, with the details reported in Table 9. The performance increases, although the number of images correctly classified is the same as the one obtained without data augmentation. However, it should be mentioned that the model is able to depict more features across all of the classes, as it is proved by the number of images correctly labelled across classes 2, 3, and 4.

Hyper model

Figure 13 and 14 reports the results of the CNN model described in Section 3.3 on the data. Figure 13 highlights the same issues that we described in Section 4.3, since

classes	precision	recall	f1-score	support
0	1.0000	0.8750	0.9333	8
1	0.5000	0.8000	0.6154	5
2	0.7143	0.7143	0.7143	14
3	0.7000	0.7778	0.7368	9
4	0.7692	0.9091	0.8333	11
5	1.0000	0.7273	0.8421	11
6	1.0000	0.8333	0.9091	12
accuracy			0.8000	70
macro avg	0.8119	0.8053	0.7978	70
weighted avg	0.8323	0.8000	0.8073	70

Table 9: Report showing the main classification metrics obtained by the model.

the hyper model once again fails to perform well on validation data. However, we can see that the model is learning features from the training data.
Figure 14 displays the confusion matrix and, again, there are several images that were wrongly classified, although the model is able to correctly predict the same number of images across all classes. More details about the classification metrics are reported in Table 9.

classes	precision	recall	f1-score	support
0	1.0000	1.0000	1.0000	10
1	0.8000	0.5333	0.6400	15
2	0.5000	0.3333	0.4000	9
3	0.8182	0.8182	0.8182	11
4	0.8750	0.7778	0.8235	9
5	0.3750	1.0000	0.5455	6
6	0.8889	0.8000	0.8421	10
accuracy			0.7286	70
macro avg	0.7510	0.7518	0.7242	70
weighted avg	0.7788	0.7286	0.7329	70

Table 10: Report showing the main classification metrics obtained by the model.

A.4 Conclusion

The Appendix aimed at testing the ability of the provided model on the same images, although labelled in a different manner. Having more than the double of the images implemented in the initial approach helped the models to perform generally better. In particular, the data augmentation layers seemed to be improved in this setting, compared to the one described in Section 3.2.

B Figures

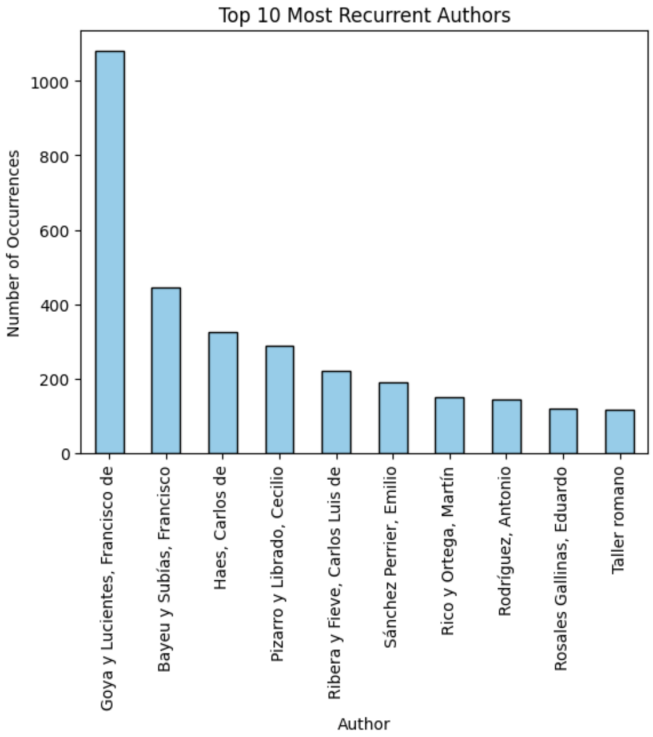

Fig. 1: On the x-axis are reported the complete names of the 10 most present authors. On the y-axis there is the total number of occurrences. The plot displays the total number of occurrences related to the 10 most recurrent authors.

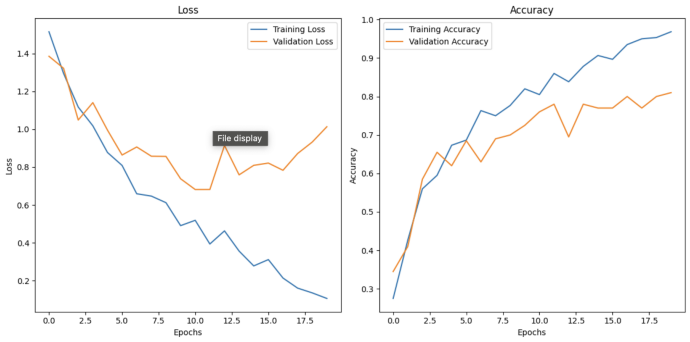

Fig. 2: First model training performance (loss and accuracy) on train and validation data.

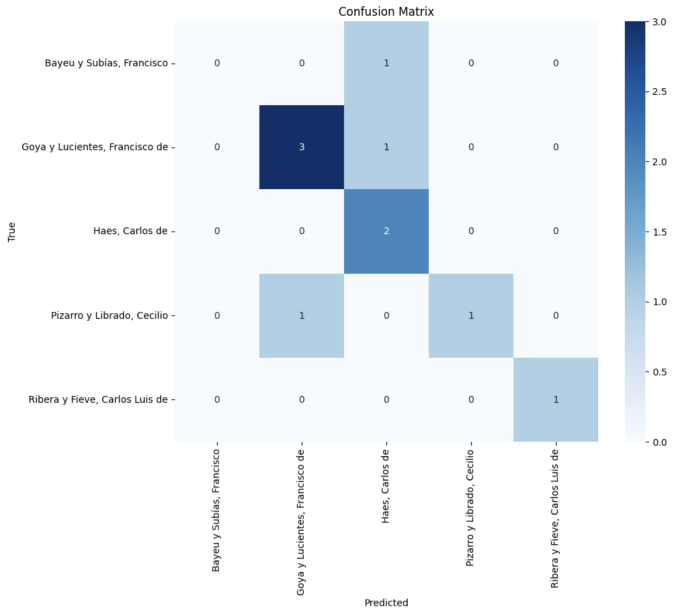

Fig. 3: Confusion matrix for the first model: on the y-axis are reported the true labels, and on the x-axis the predicted ones.

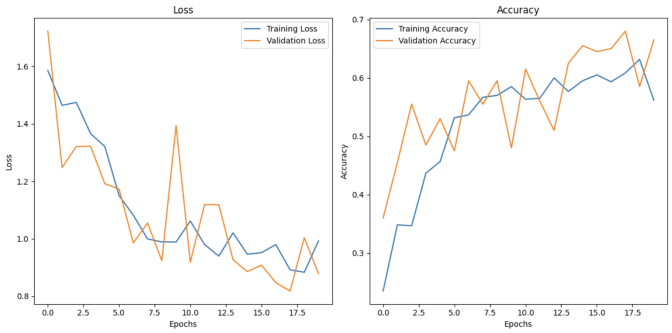

Fig. 4: First model (with data augmentation) training performance (loss and accuracy) on train and validation data.

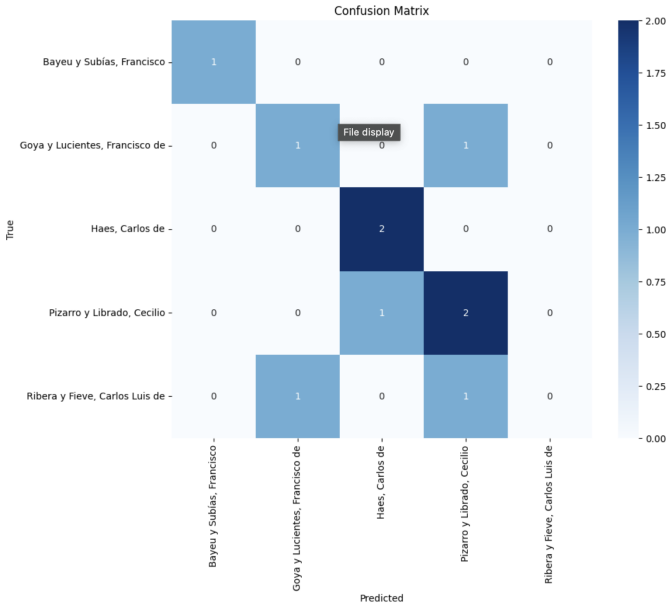

Fig. 5: Confusion matrix for the first model (with data augmentation): on the y-axis are reported the true labels, and on the x-axis the predicted ones.

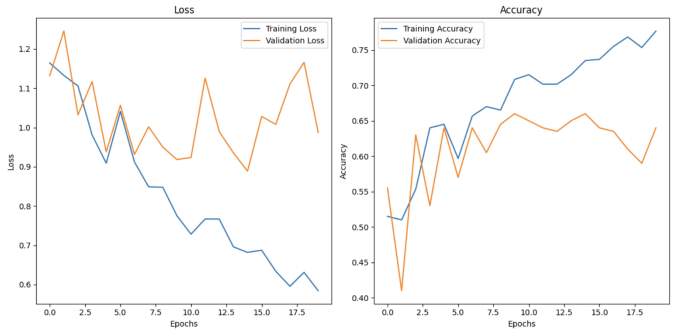

Fig. 6: Hyper-parameter model training performance (loss and accuracy) on train and validation data.

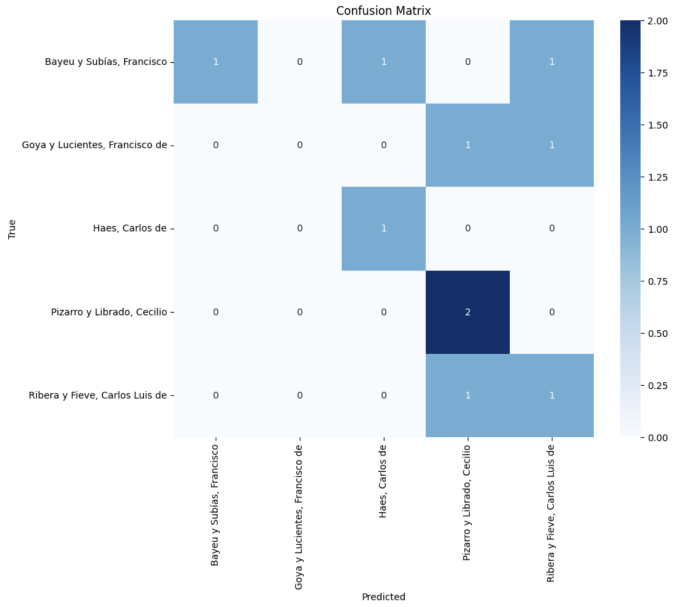

Fig. 7: Confusion matrix for the first model hyper-parameter model: on the y-axis are reported the true labels, and on the x-axis the predicted ones.

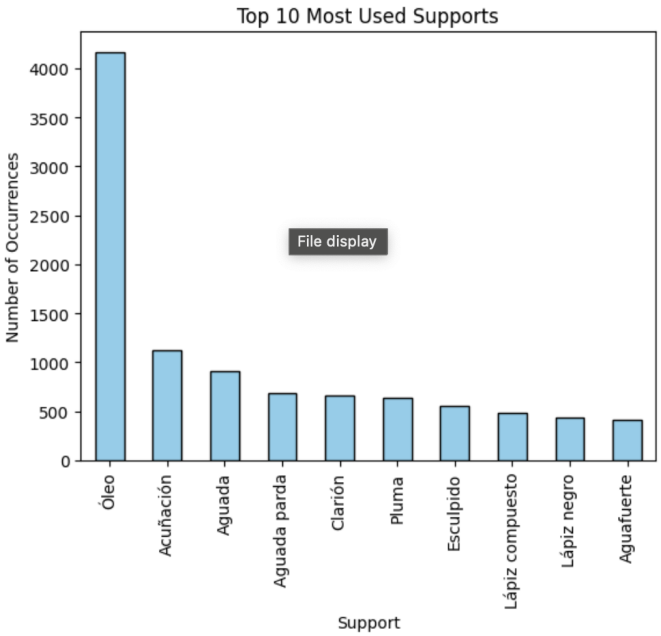

Fig. 8: On the x-axis are reported the complete names of the 10 most used supports. On the y-axis there is the total number of occurrences. The plot displays the total number of occurrences related to the 10 most used supports.

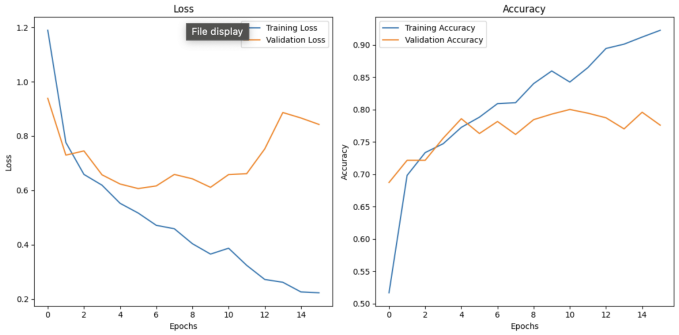

Fig. 9: First model training performance (loss and accuracy) on train and validation data.

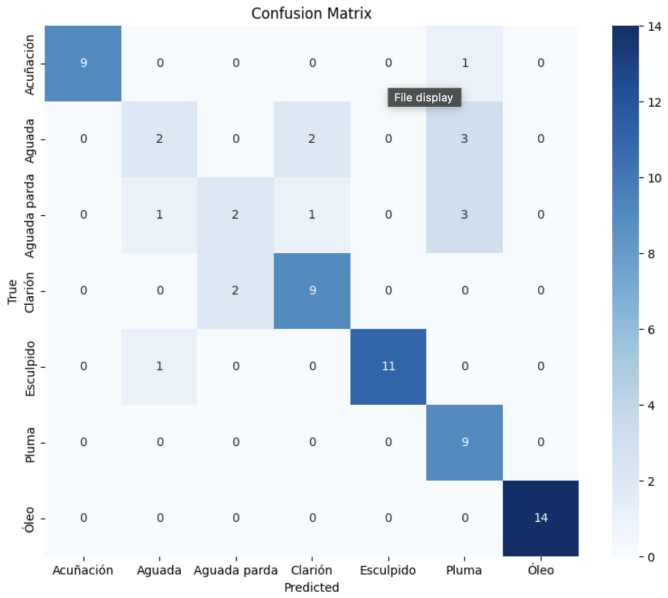

Fig. 10: Confusion matrix for the first model: on the y-axis are reported the true labels, and on the x-axis the predicted ones.

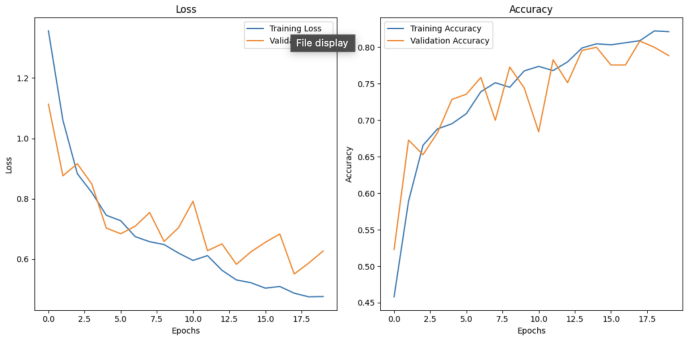

Fig. 11: First model (with data augmentation) training performance (loss and accuracy) on train and validation data.

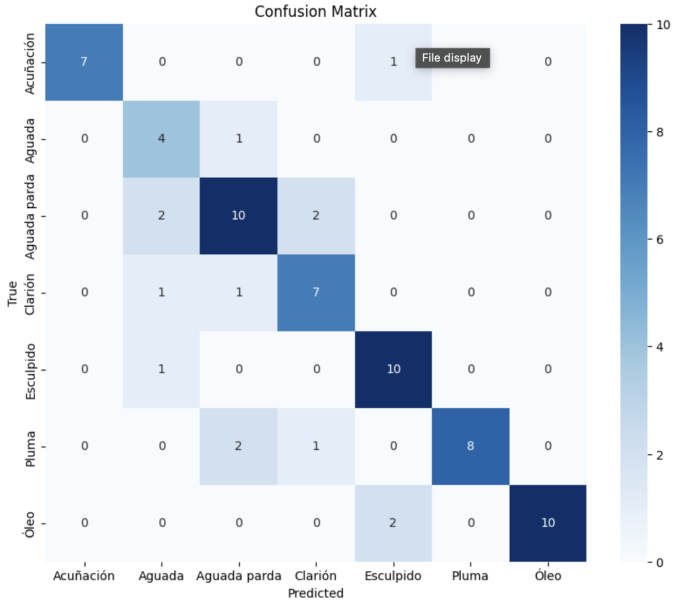

Fig. 12: Confusion matrix for the first model (with data augmentation): on the y-axis are reported the true labels, and on the x-axis the predicted ones.

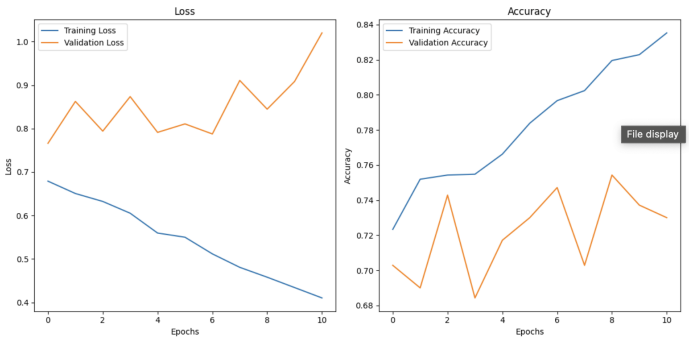

Fig. 13: Hyper-parameter model training performance (loss and accuracy) on train and validation data.

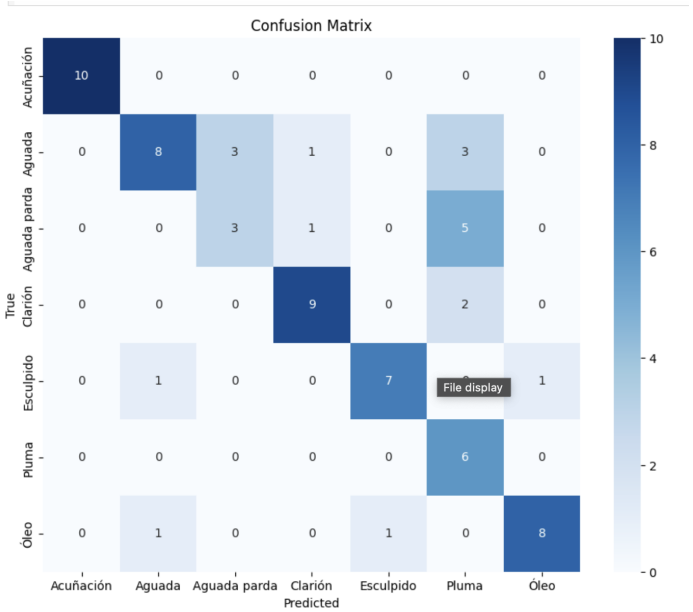

Fig. 14: Confusion matrix for the first model hyper-parameter model: on the y-axis are reported the true labels, and on the x-axis the predicted ones.