Binary classification of muffin and chihuahua images

Statistical methods for machine learning project 2022-2023

Arianna Converti, 991787
GitHub

May 23, 2024

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. 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.

1 Introduction

Karen Zack, a young American girl, shared collages on the social network where she compared food or objects with animals. The most popular comparison is between muffin and Chihuahua. [1]

Figure 1: image collage by Karen Zack

Confusing a muffin with a Chihuahua can be fun, but, in the image recognition, distinguishing between the two is a challenging problem. The main problem is that the images have a large number of visual similarities, making it difficult for the neural network to differentiate.
The aim of this project is to discuss the result obtaining by applying different neural network models to solve this problem.

2 Models

In this project, the purpose is to apply some Neural Networks for the binary classification of muffins and Chihuahuas. Mainly two different architectures are used:

- Multi layer perceptron
- Convolutional neural network
  - LeNet5
  - AlexNet
  - VGG16
  - ResNet50

2.1 Multi Layer Perceptron

A Multi Layer Perceptron (MLP) [2] is a feed forward Neural Networks, i.e, an acyclic graph when the information travels in one direction . It is used for predicting data that are not linearly separable.

This NN is composed of several layers linked together: the first one is called input layers, the last one is called output layers and in between, there are one or several hidden layers. Each node passes its value to the subsequent node only in the forward direction. The MLP neural network uses a Backpropagation algorithm to increase the accuracy of the training model.

The architecture of the MPL used in this work is composed of the following layers:

- Input layers is a vector x_i that represent an image of size 128x128x3 pixels. This neuron sends the signal to a Flatten layer that converts 2D input (an image) to a 1D vector;
- Two hidden Layers with p and q neurons each. Initially, p and q are set to 128 and 256, respectively. To prevent the overfitting also Dropout neurons are added, which randomly sets the input units to 0.
- Output Layer, since the binary classification is done, is formed by one neuron with a sigmoid activator that produces an output $y \in [0, 1]$.

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 49152)	0
dense_6 (Dense)	(None, 128)	6291584
dropout_4 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 256)	33024
dropout_5 (Dropout)	(None, 256)	0
dense_8 (Dense)	(None, 1)	257

Total params: 6,324,865
Trainable params: 6,324,865
Non-trainable params: 0

Figure 2: Multi layers perceptron structure

2.2 Convolutional Neural Networks

A convolutional Neural Network (CNN) is a family of deep learning algorithms specified for tasks that need object recognition, including image classification [2].

In general, the architecture of CNN is composed of the following layers:

- Convolution layer: It applies a several sliding window functions called filters to a matrix of pixels representing an image. Each of this filters are used to recognize a specific pattern from the image. A ReLU activation function is applied after each convolution operation. This function helps the network learn non-linear relationships between the features in the image [3].
- Pooling layer: It applies an aggregation operations, which reduce the dimension of the feature map to reduce the the memory during the training [3].

  The most common aggregation functions that can be applied are:

  - Max pooling, which is the maximum value of the feature map
  - Average pooling is the average of all the values.
- Fully-connected layer: It is a sigmoid prediction layer used to generate probability values for each of the two possible output labels, and the final label predicted is the one with the highest probability score [3].

To avoid overfitting, it is often useful to use a dropout layer, which consists of randomly dropping some neurons during the training process, which forces the remaining neurons to learn new features from the input data [3].

In this project different architectures are studied.

2.2.1 LeNet5

LeNet is a convolutional neural network structure proposed by LeCun in 1998, due to its architecture it is considered the simplest CNN.

It takes as input grayscale images (32x32 pixels) and produces as outputs a probability that an image belongs to a specific class. The LeNet-5 CNN architecture has seven layers: three convolutional layers, two sub sampling layers, and two fully linked layers [4].

```
Model: "leNet5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 28, 28, 6)         156

 average_pooling2d_4 (Averag  (None, 14, 14, 6)        0
 ePooling2D)

 conv2d_5 (Conv2D)           (None, 10, 10, 16)        2416

 average_pooling2d_5 (Averag  (None, 5, 5, 16)         0
 ePooling2D)

 dense_6 (Dense)             (None, 5, 5, 120)         2040

 flatten_2 (Flatten)         (None, 3000)              0

 dense_7 (Dense)             (None, 84)                252084

 dense_8 (Dense)             (None, 1)                 85

=================================================================
Total params: 256,781
Trainable params: 256,781
Non-trainable params: 0
```

Figure 3: LeNet5 structure used in the project

2.2.2 AlexNet

AlexNet was the first architecture that used GPU to boost the training performance.
AlexNet consists of 5 convolution layers, 3 max-pooling layers, 2 Normalized layers, 2 fully connected layers and 1 Soft Max layer. Each convolution layer consists of a convolution filter and a non-linear activation function called "ReLU". The pooling layers are used to perform the max-pooling function and the input size is fixed due to the presence of fully connected layers [5].

As stated in the paper, it us decided to use RGB images of size 224x224 pixels.
Figure 4 shows the architecture used in this project.

```
Model: "alexNet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 54, 54, 96)        34944

 max_pooling2d_6 (MaxPooling  (None, 26, 26, 96)       0
 2D)

 conv2d_11 (Conv2D)          (None, 26, 26, 256)       614656

 max_pooling2d_7 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 conv2d_12 (Conv2D)          (None, 12, 12, 384)       885120

 conv2d_13 (Conv2D)          (None, 12, 12, 384)       1327488

 conv2d_14 (Conv2D)          (None, 12, 12, 256)       884992

 max_pooling2d_8 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 flatten_2 (Flatten)         (None, 6400)              0

 dense_6 (Dense)             (None, 4096)              26218496

 dropout_4 (Dropout)         (None, 4096)              0

 dense_7 (Dense)             (None, 4096)              16781312

 dropout_5 (Dropout)         (None, 4096)              0

 dense_8 (Dense)             (None, 1)                 4097

=================================================================
Total params: 46,751,105
Trainable params: 46,751,105
Non-trainable params: 0
```

Figure 4: AlexNet structure used in the project

2.2.3 VGG16

VGG16 is a convolutional neural network proposed by K. Simonyan and A. Zisserman. It is able to classify 1000 images of different category with 92.7% accuracy. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks taking advantages of transfer learning[1] [6].

The network is composed of 16 layers: thirteen convolutional layers and three Dense (or Fully connected) layers. These layers are organized into blocks, with each block containing multiple convolutional layers followed by a max-pooling layer for down sampling . The VGG16 takes as input a RGB images of size 128 x 128 pixels [6].
The figure 26 shows the architecture used in the project.

2.2.4 ResNet50

ResNet-50 is a convolutional neural network that is 50 layers deep (48 Convolution layers with 1 Max Pooling and 1 Average Pooling layer). It is possible to load a pretrained network trained on more than a million images from the ImageNet database [7].

[1] a technique in which knowledge learned from a task is re-used in order to boost performance on a related task

The ResNet architecture follows two basic design rules. First, the number of filters in each layer is the same depending on the size of the output feature map. Second, when the size of the feature map is reduced by half, the number of filters is doubled to keep the time complexity of each layer constant.

The main innovation of ResNet50 is the introduction of residual block, in which the input is added to the output. This allows the model to learn difference between the input and the output, rather than trying to learn the entire mapping.

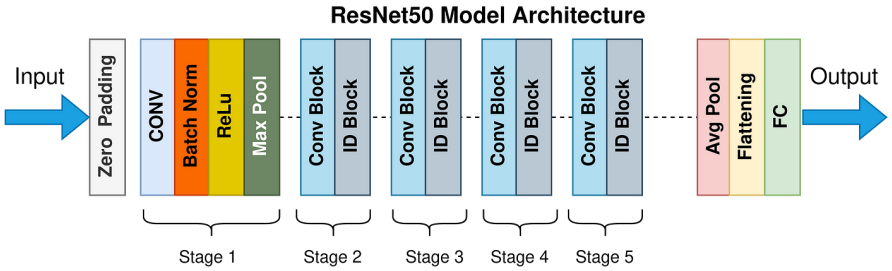

Figure 5: ResNet50 classical structure

3 Experimental setup

3.1 The considered task

The experiment consists of using Keras [8] to train a neural network for the binary classification of muffins and Chihuahuas images. This must be transformed from JPG to RGB (or grayscale) pixel values and scaled down.

Also it's asked to:

- experiment with different network architectures (at least 3) and training hyperparameters,
- use 5-fold cross validation to compute your risk estimates,
- thoroughly discuss the obtained results, documenting the influence of the choice of the network architecture and the tuning of the hyperparameters on the final cross-validated risk estimate.

While the training loss can be chosen freely, the reported cross-validated estimates must be computed according to the zero-one loss.

The experiment is performed on *Google Colab*, a platform that allows to write and execute Python in browser [9]. Due to Colab policy, a "MSI GTX 1660 super" GPU is also used.

3.2 Dataset

The dataset used is composed of approximately 6,000 RGB images scraped from *Google Images*. It is downloaded from this link, a repository on *Kaggle*.

The retrieved dataset was preprocessed by the author, who removed duplicate images and divided them into two directories: one for training the networks and the other for testing it. Each of directories has two sub-directories containing the images organized according to their class membership, which can be Chihuahua or muffin.

The retrieved data is described in the following chart 6 and table 1.

	Chihuahua	Muffin
Number of train samples	2559	2174
Number of test samples	640	544

Table 1: Information about the datasets.

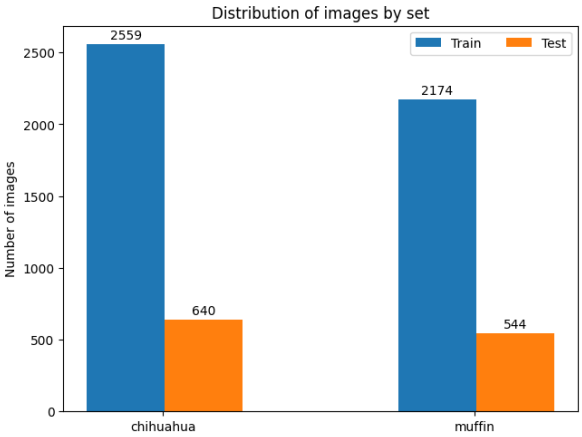

Figure 6: Muffin and chihuahua dataset

3.3 Data pre-processing

The data pre-processing is composed of three tasks:

- Data exploration and label binarization
- Class balance
- Data Augmentation and Normalization

3.3.1 Data exploration and label binarization

Analyzing the dataset, it is noticed that the images have all the same dimensions of 256x256 pixels. Therefore, an attempt will be made to find an optimal image size, considering that smaller inputs mean a model that is faster to train, and, typically, this concern dominates in the choice of image size.

For computational reason, the image size is fixed to 128x128 pixels.

Figure 7: Some images

Since this is a binary classification problem, the technique of integer encoding will be used, in which label 0 represents the class of Chihuahua and 1 represents that of muffin (as shown on image 7).

3.3.2 Class balance

Strong imbalance in datasets can provoke a problem in the learning phase [10]. If the ratio between the chihuahua and the muffin examples is too low (or high), the classifier will predict the most common class.

According to the chart 8, the classes are approximately balanced in both sets.

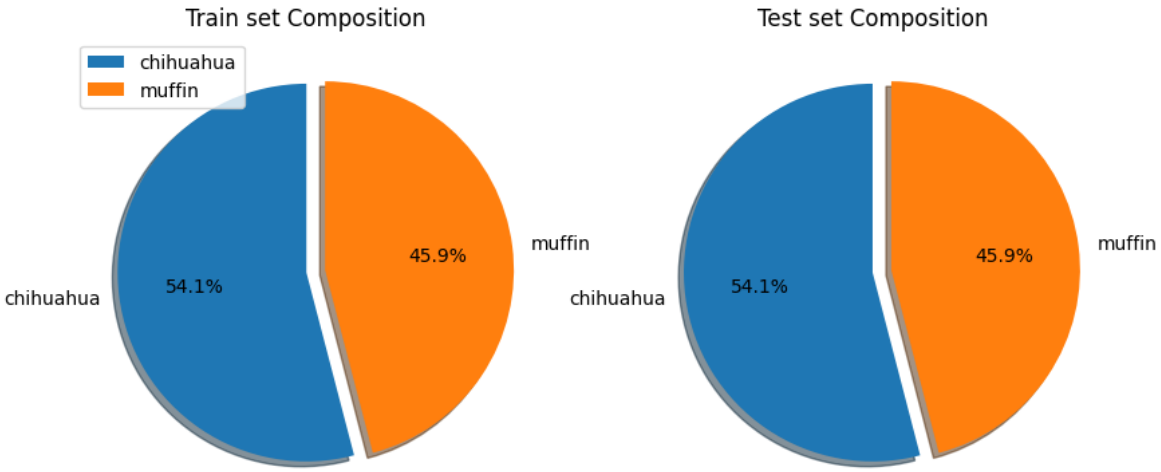

Figure 8: Dataset composition

3.3.3 Data Augmentation and image Normalization

The data augmentation is a practice used by Machine Learning to reduce overfitting [2] and improve model generalization and its performance [11]. It introduces several techniques, including geometric transformation, color space adjustments and noise injection [11]. Some of these techniques are used in this project:

- Rotation: it performs a rotation of the images by a random degree between 0 and 360. When the image is rotated, it is possible to have an empty area that need to be filled in. To do so the nearest way is used, which replaces the empty area with the nearest pixel values [12];
- Zoom: it randomly zooms in on the image or zooms out of the image [12] ;
- Width/height shift: it shifts the pixels of the image either horizontally or vertically by adding a certain constant value to all the pixels [12];
- Brightness: it randomly changes the brightness of the image. It is also a very useful augmentation technique because most of the time our object will not be under perfect lighting condition [12];
- Vertical flip: it performs a flipping of the images along the vertical axis [12] ;
- Rescale: it performs normalization of all the pixels into the range [0,1] [12].

These techniques are all included into the function *ImageDataGeneration* [12] into Keras pre-processing library. The following images 9 and 10 represent the result of the transformations using in the project.

[2] Overfitting happens when a model is too closely fitted to the training data and performs poorly on new data.

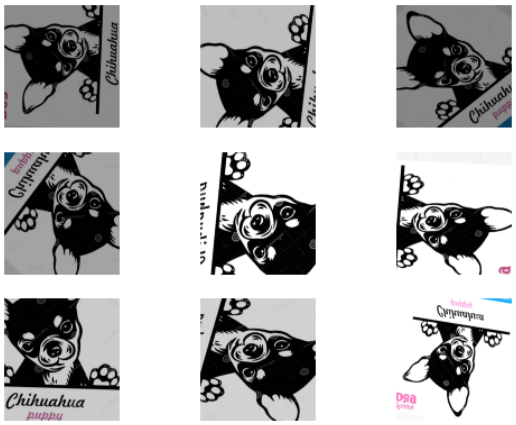

Figure 9: Data augmentation applied to a specific image

Figure 10: Data augmentation applied to a batch of the training set

3.4 5-Fold Cross Validation

All the models use the 5-Fold Cross Validation to avoid the overfitting and to estimate their performance. The procedure or in general k-fold cross validation is used to evaluate the model design, re-training it with different set. K is an integer parameter (usually small) in which the dataset will be divided. The algorithm [13] works as follow: let S the entire dataset,

1. Partition S in $K(K = 5)$ fold $S_1, \ldots, S_K$ of size m/K each.
2. For each group:
   (a) Take the group S_i as a test data set
   (b) Take the remaining groups as a training data set $S_{-i} \equiv S/S_i$
   (c) Fit and run a model A on the training set, obtaining a predictor $h_1 = A(S_{-i}), ..., h_K = A(S_{-K})$
   (d) Compute the average error on the testing part of each fold

$$l_{S_i}(h_i) = \frac{K}{m} \sum_{(\mathbf{x},y)\in S_i} l(y, h_i(\mathbf{x}))$$

3. Compute the CV estimate by averaging these errors

$$l_S^{CV}(A) = \frac{1}{K} \sum_{i=1}^{K} l_{s_i}(h_i)$$

The figure 11 provides a better explanation of how the folds are utilized.

3.5 Zero One Loss

To evaluate the models, the zero one loss function are used. This function is defined as:

$$\text{Zero-One Loss} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\hat{y}_i \neq y_i)$$

where:

- N is the total of samples
- $\hat{y}_i$ is the model's prediction of i-th sample

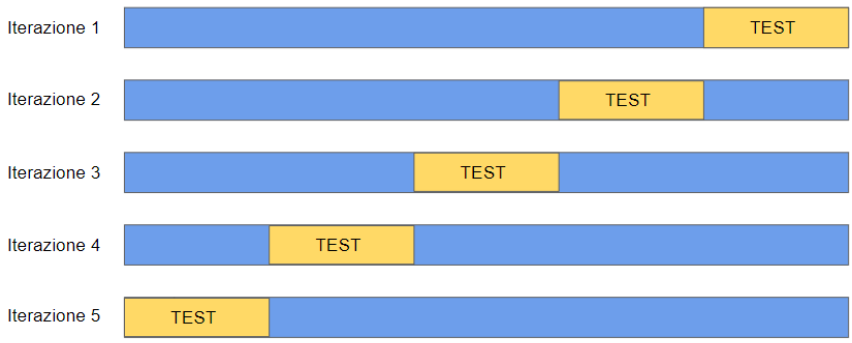

Figure 11: 5-fold cross validation: the blue folds represent the training set, while the other represents the test sets.

- y_i is the true label of *i-th* sample
- $\mathbf{1}(\cdot)$ is a Indicator function that returns 1 if the condition is true and 0 otherwise.

In a binary classification problem, zero-one loss is complementary to Accuracy [3.6] and be calculate as:

$$\text{Zero-One Loss} = 1 - \text{Accuracy}.$$

This relationship exists because each correctly classified example does not contribute to the zero-one loss, while each incorrectly classified example adds $1/N$ to the loss. Conversely, each correctly classified example contributes positively to the accuracy.

In the context of zero-one loss, lower values indicate better performance.

3.6 Result analysis

The results of the prediction of test images are analyzed using :

- Accuracy, that calculates the ratio of correct predictions out of the total number of examples [16]. The best value is 1 and the worst value is 0 . The formula for accuracy is:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

  where TP is the number of true positives , TN is the number of true negatives, FP is the number of false positives and FN is the number of false negatives. Otherwise, it can also be defined as:

$$\text{Zero-One Loss} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\hat{y}_i = y_i)$$

- Precision, that is the ratio of relevant instances among the retrieved instances [17]. The best value is 1 and the worst value is 0.

$$\text{Precision} = \frac{TP}{TP + FP}$$

  where the TP is the number of true positives and FP the number of false positives.

- Recall, that calculates how many of the Actual Positives our model capture through labeling it as Positive [17]. The best value is 1 and the worst value is 0.

$$\text{Recall} = \frac{TP}{TP + FN}$$

  where TP is the number of true positives and FN the number of false negatives.

- F1-score, that can be interpreted as an harmonic mean of the precision and recall metrics, where a F1 score reaches its best value at 1 and worst score at 0 [19]. The formula for the F1 score is:

$$F1 = \frac{2 * TP}{2 * TP + FN + FP}$$

- ROC curve, that feature true positive rate (TPR) on the Y axis, and false positive rate (FPR) on the X axis. This means that the top left corner of the plot is the "ideal" point, where both the FPR and TPR are zero. This is not very realistic, but it does mean that a larger area under the curve is usually better [20].

4 Result

Different experiments are conducted, and different results are obtained. In all the training session, *ModelCheckpoint* and *EarlyStopping* are used. The first is used to save a model or weights at certain intervals, allowing the model or weights to be loaded later to continue the training from the saved state. The second is used to prevent overfitting and it can treat the number of training epochs as a hyperparameter.
The Random Search method of Keras Tuner library is used to tune the hyperparameters of models, such as learning rate of the optimizer , the number of units in Dense layer, or the Dropout rate.

Firstly, the MLP network is trained with RGB images of size 128x128 pixel. Two training session are conducted: one with standard hyperparameters (first dropout layer equals 0.05, second layer equals to 0.05; units of Dense layer equals 256 and the learning rate equals 0.01) and the second with the tuned hyperparameters. The result of the tuning is: dropout rate 1 equals 0.2, dropout rate 2 equals 0, the units of Dense layer equals to 32 and learning rate equals to 0.0003.
During the training, a question arises: what if data augmentation is not applied? An experiment is conducted, and the answer is: *overfitting*. In other words, during the training, the validation loss is greater than the training loss. To avoid overfitting, the data augmentation is applied.

Secondly, the LeNet5 network is trained with grayscale images of size 32x32 pixel. Initially, standard hyperparameters are applied: the units in the dense layer equal 84 and the learning rate of Adam optimizer is 0.001. During training, overfitting is observed.
Therefore, hyperparameters are tuned in this way: units in the dense layer equals 64 and learning rate of 0.0001, but the result remained the same.

Next, a custom CNN is trained with RGB images of size 128x128 pixel. The custom CNN is composed as shown in figure 12, with initial hyperparameter: units in the dense layer equals to 64, dropout rate of 0.5 and learning rate of Adam optimizer of 0.001 . After tuning this parameter change to 128, 0.1 and 0.003, respectively. The result of the training is shown in chapter 6.3.

```
Model: "cnn_custom"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 126, 126, 32)      896

 max_pooling2d_4 (MaxPooling  (None, 63, 63, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 61, 61, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 flatten_2 (Flatten)         (None, 57600)             0

 dense_4 (Dense)             (None, 64)                3686464

 dropout_2 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 3,705,921
Trainable params: 3,705,921
Non-trainable params: 0
```

Figure 12: Custom CNN architecture

Following that, the AlexNet network is trained with RGB images of size 224x224 pixel. Similarly, two training session are conducted. The hyperparameters tuned are limited to three: the dropout rate for both layers and the learning rate of the Adam optimizer. Initially, they are set on 0.5, 0.5 and 0.001. After tuning, the results are 0.3, 0.3 and 0.001, respectively.

Given the result of the first networks (chapter 6.1, 6.2, 6.3, 6.4), all the previous networks overfit. It is assumed that the networks are too simple to solve the problem. It was decided to increase the complexity of the networks.

The last two training were conducted with the VGG16 and ResNet50 trained with RGB images of size 128x128 pixel. Due to the limited computational power, the hyperparameter tuning was not performed; instead a classical value of 0.001 was used to the learning rate hyperparameter of the Adam optimizer.

These networks didn't overfitting during training (chapter 6.5, 6.6), so the evaluated model can be considered valid. The results are reported in table 2.

Model	**Avg_Val_loss**	**Avg_Val_accuracy**	**Avg_Val_zero_one_loss**
VGG16	0.160	0.938	0.062
ResNet50	0.537	0.728	0.272

Table 2: Final results of training

In conclusion, it is possible to obtain good results (besides the overfitting) in the binary classification task with the convolutional neural network, while the multi-layered network showed poor results. According to the zero-one-loss metric shown in table 2, the best model is VGG16, which was loaded and tested on the images from the test set. The network achieves an accuracy of 93% and the figure 14, the ROC curve 15 and the table 13 shown the result of this prediction.

	precision	recall	f1-score	support
chihuahua	0.98	0.90	0.94	640
muffin	0.89	0.98	0.93	544
accuracy			0.93	1184
macro avg	0.93	0.94	0.93	1184
weighted avg	0.94	0.93	0.93	1184

Figure 13: Classification report of prediction

5 Future developments

The results obtained can be considered as preliminary because it is necessary to execute the models with more computational power, which would allow to have more experiments and a better tuning approach, increasing the number of epochs and the number/size of the layers.

The image segmentation and the feature selection technique that identify relevant features could be introduced. This technique is useful as it eliminates redundant or irrelevant features without removing important information. The top feature selection algorithms are *Random Forest* and *Boruta*. Implementing this approach can significantly enhance the model's discriminatory capabilities and better results could be achieved.

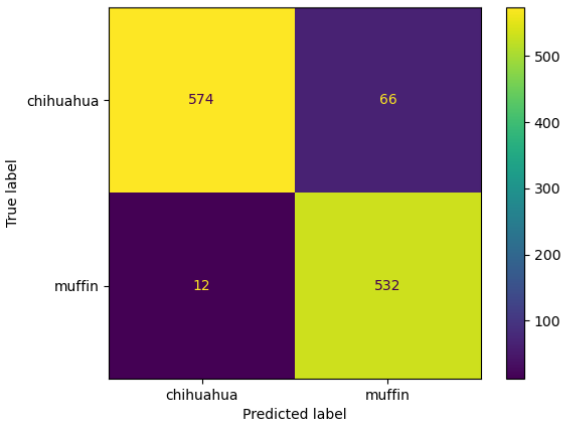

Figure 14: Confusion matrix of prediction

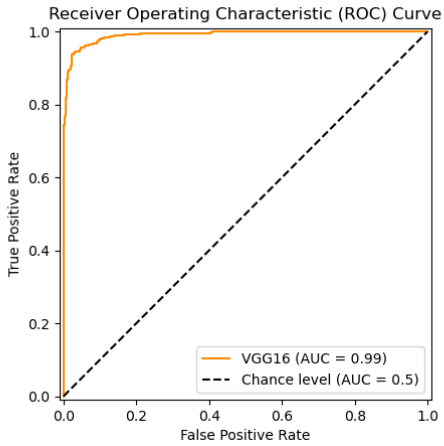

Figure 15: ROC curve of prediction

6 Graphs

6.1 Graphs of MLP

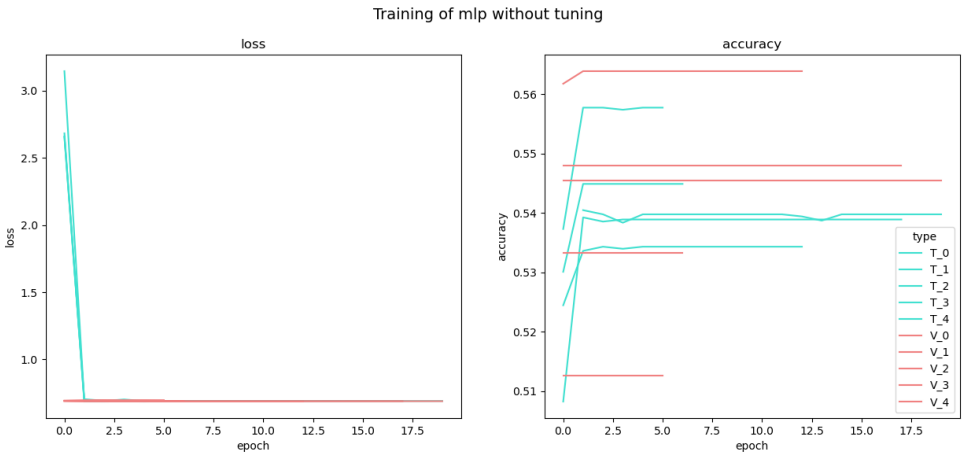

Figure 16: Final graph with standard hyperparameters

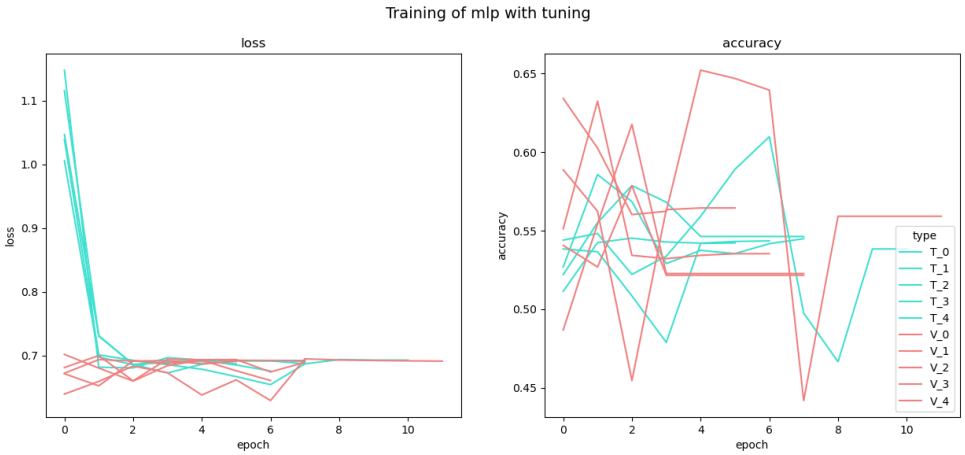

Figure 17: Final graph with tuned hyperparameters

6.2 Graphs of LeNet5

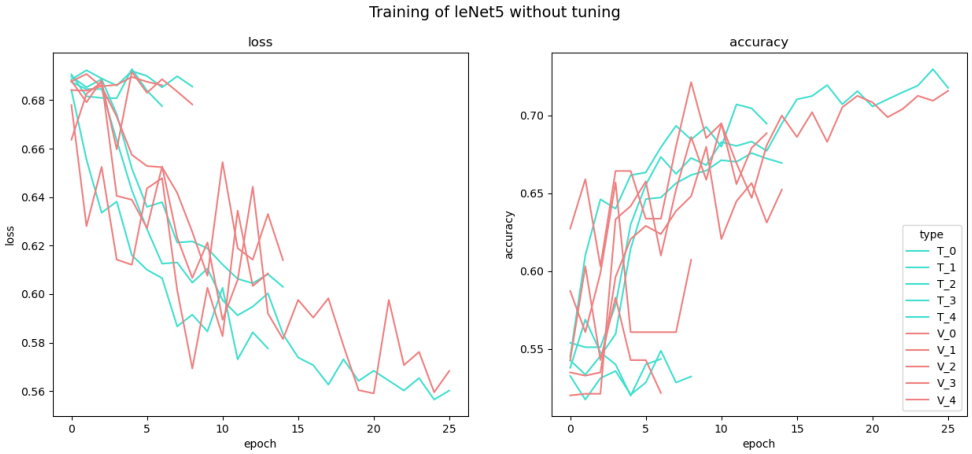

Figure 18: Final graph with standard hyperparameters

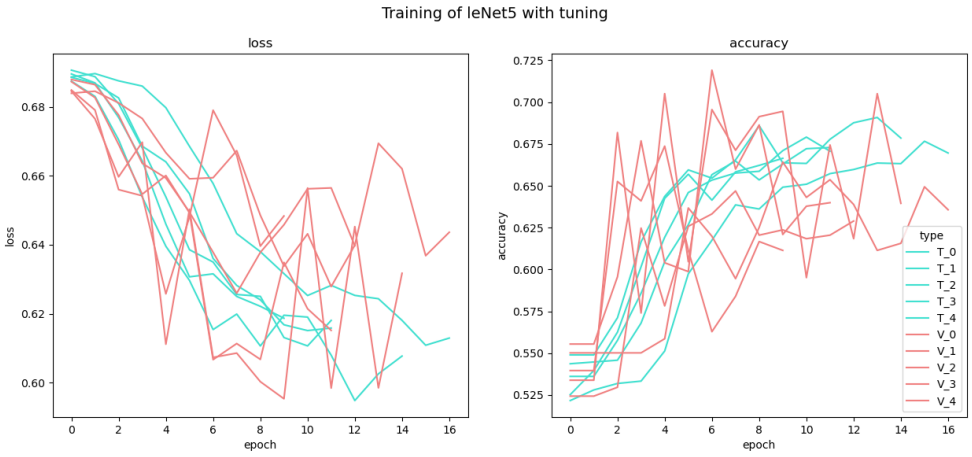

Figure 19: Final graph with tuned hyperparameters

6.3 Graphs of custom CNN

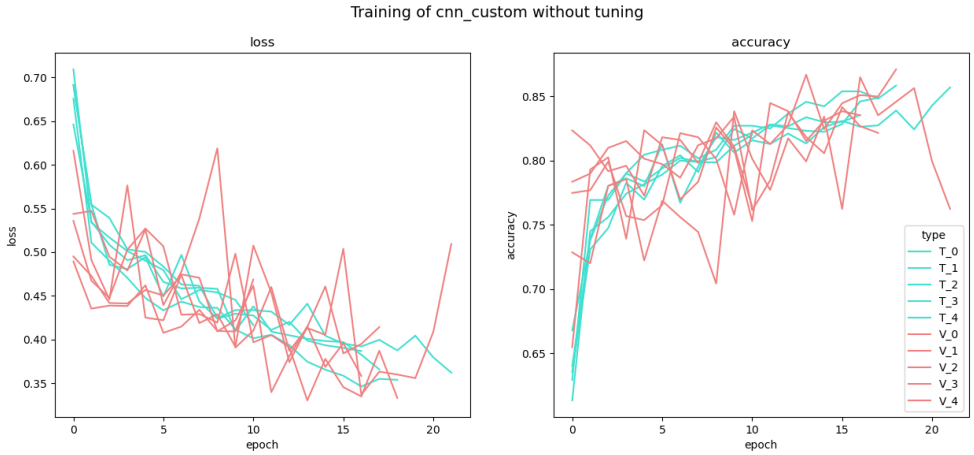

Figure 20: Final graph with standard hyperparameters

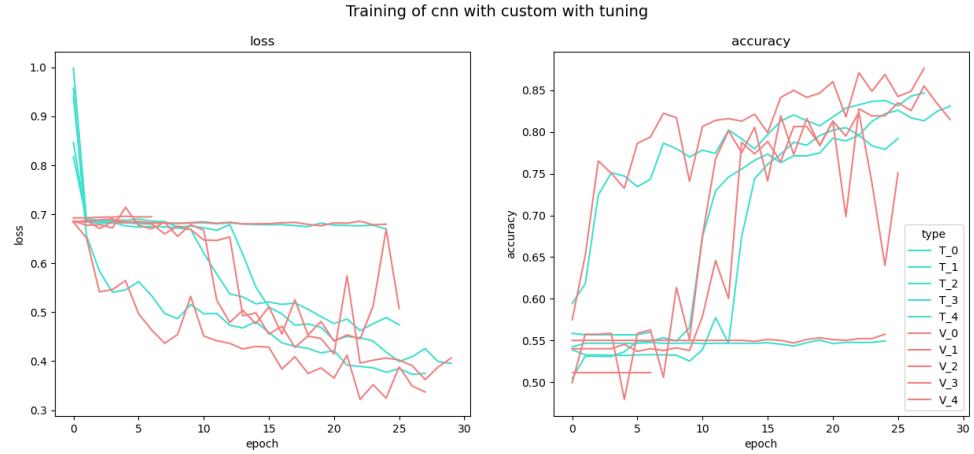

Figure 21: Final graph with tuned hyperparameters

6.4 Graphs of AlexNet

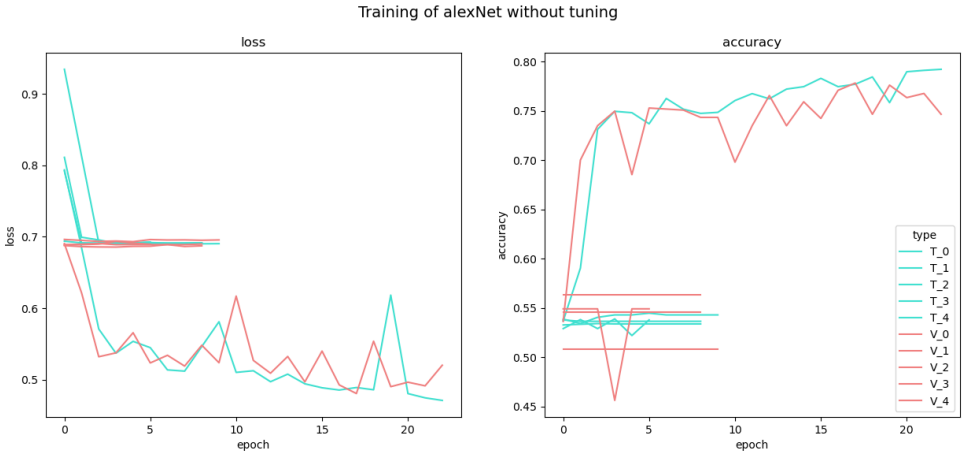

Figure 22: Final graph with standard hyperparameters

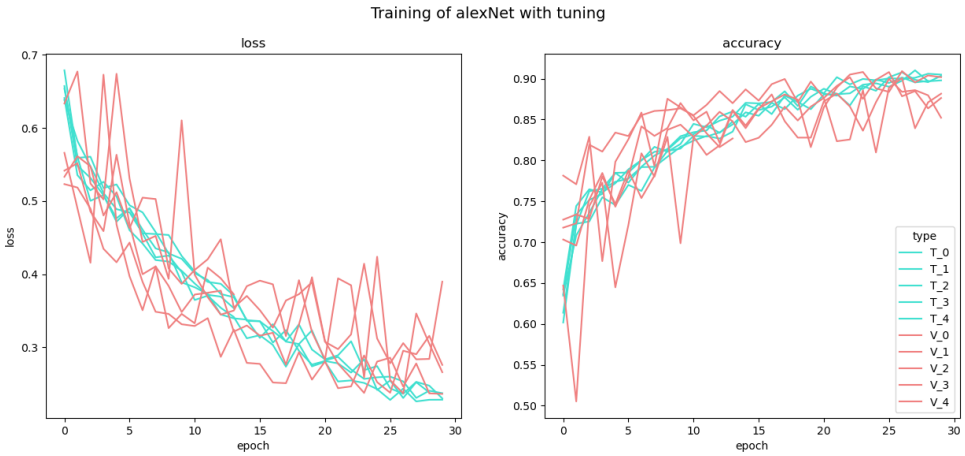

Figure 23: Final graph with tuned hyperparameters

6.5 Graphs of VGG16

Figure 24: Final graph with standard hyperparameters

6.6 Graphs of ResNet50

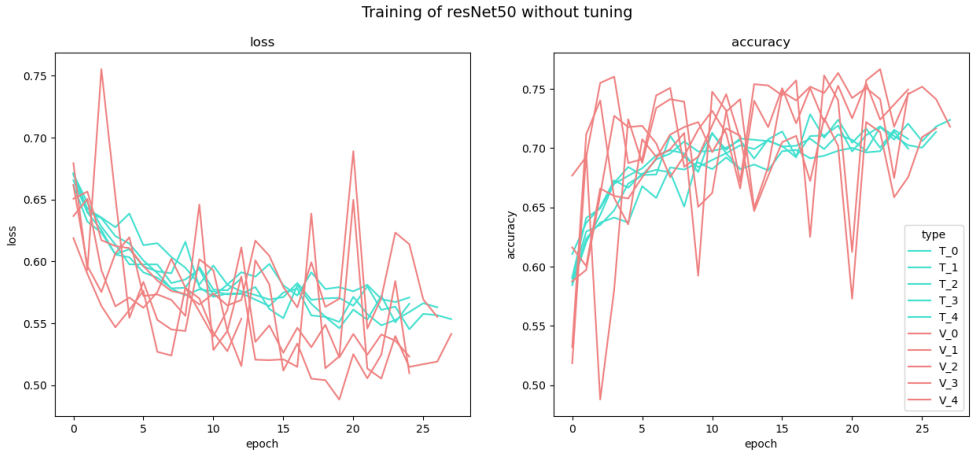

Figure 25: Final graph with standard hyperparameters

7 Appendix

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 global_average_pooling2d_2   (None, 512)              0
 (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 256)               131328

 dense_5 (Dense)             (None, 1)                 257

=================================================================
Total params: 14,846,273
Trainable params: 131,585
Non-trainable params: 14,714,688
```

Figure 26: VGG16 structure used in the project

References

[1] è un muffin o un chihuahua? Le incredibili somiglianze tra animali e cibo.

[2] Rudolf Kruse , Christian Borgelt , Frank Klawonn , Christian Moewes , Matthias Steinbrecher , Pascal Held, Computational Intelligence A Methodological Introduction. 2013.

[3] Introduction to convolutional neural networks cnn.

[4] LeNet5 Architecture Explained.

[5] AlexNet architecture explained.

[6] Everything you need to know about VGG16.

[7] ResNet-50: The basics and a Quick tutorial.

[8] Keras.io.

[9] Google Colaboratory.

[10] Class imbalance.

[11] Wikipedia, Data Augmentation.

[12] Image augmentation.

[13] K-Fold Cross Validation.

[14] Yuli Liu, Python Machine Learning by example. Second Edition, 2019.

[15] Dr.Randal S.Olson, Python Machine Learning, 2015.

[16] Accuracy and precision.

[17] Accuracy, Precision, Recall or f1? .

[18] Sensitivity and specificity.

[19] f1 score.

[20] Receiver operating characteristic.

[21]