UNIVERSITÀ
DEGLI STUDI
DI MILANO

Neural Network Binary Classification

Alessandro Compagnoni

A.A. 2023-2024

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 offenses 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

1 Introduction on Binary Classification

Binary classification is used in the machine learning domain commonly. It is the simplest way to classify the input into one of two possible categories. With the help of effective use of Neural Networks (Deep Learning Models), binary classification problems can be solved to a fairly high degree.

Here we are using Convolution Neural Network (CNN). It is a class of Neural network that has proven very effective in areas of image recognition, processing, and classification. Specifically, we will utilize CNNs to classify images, that are stored as three-dimensional arrays of pixels defined by height, width, and depth. Height and width vary depending on the image size, while depth refers to whether the image is in grayscale or RGB. The pixel values, regardless of the image format, typically range from 0 to 255. However, due to the wide range of these values, to simplify the optimization processes within the CNN and handle color scale variations more efficiently, pixel values are normalized in a range between 0 and 1.

CNN models require training data for training weights and validation for checking its performance. Each input images passes through a series of layers:

- Convolution Layers
- Pooling Layers
- Dense Layers

the Convolution and Pooling layers act as feature extractors from the input image while a Dense layer acts as a classifier. For example, on receiving an image as input, the network assigns the highest probability for it and predict which class the input image belongs to.

The functioning of each of the layers is explained in detail below.

1.1 Convolution Layers

Convolution is the process of using various kinds of filters to extract features from a given image, this extraction is achieved using a filter matrix called kernel, typically with dimensions 3 × 3 while the image matrix is divided into submatrices of the same size as the kernel. For each submatrix, the dot product with the kernel is calculated and, after performing the scalar product, a bias term is added, allowing us to handle potential changes in the data. The values within the filter and the biases are initially set randomly and then optimized with backpropagation, which will be explained later.

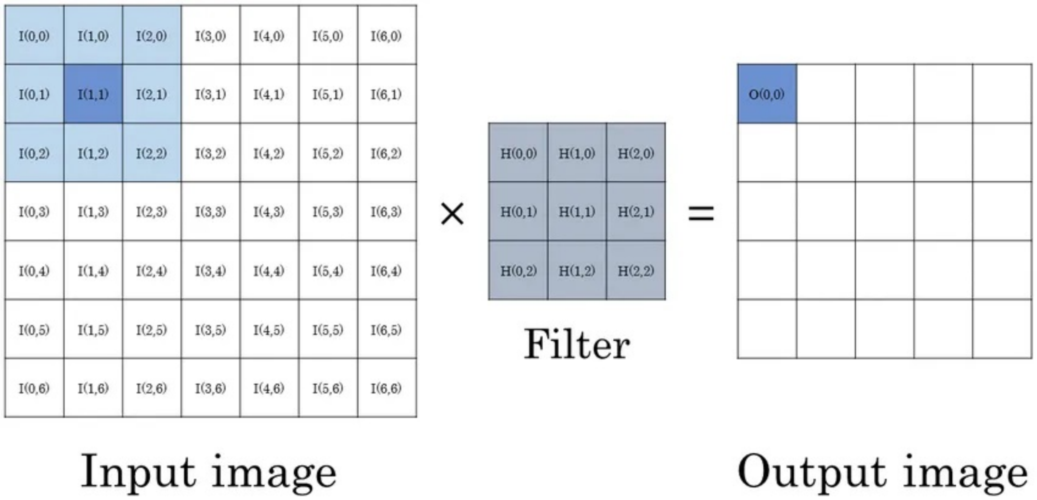

I(0,0) I(1,0) I(2,0) I(3,0) I(4,0) I(5,0) I(6,0)
I(0,1) I(1,1) I(2,1) I(3,1) I(4,1) I(5,1) I(6,1)
I(0,2) I(1,2) I(2,2) I(3,2) I(4,2) I(5,2) I(6,2)
I(0,3) I(1,3) I(2,3) I(3,3) I(4,3) I(5,3) I(6,3)
I(0,4) I(1,4) I(2,4) I(3,4) I(4,4) I(5,4) I(6,4)
I(0,5) I(1,5) I(2,5) I(3,5) I(4,5) I(5,5) I(6,5)
I(0,6) I(1,6) I(2,6) I(3,6) I(4,6) I(5,6) I(6,6)
×
H(0,0) H(1,0) H(2,0)
H(0,1) H(1,1) H(2,1)
H(0,2) H(1,2) H(2,2)
Filter
=
O(0,0)
Input image
Output image

Figure 1: Convolution on an image

The dimension of the feature map obtained is dependent on wether padding is applied or not and on stride. If there is no padding, the feature map will not be of the same size as the input while if padding is present the size will be the same as the input image. The stride indicates how many pixels the submatrices should be spaced apart.

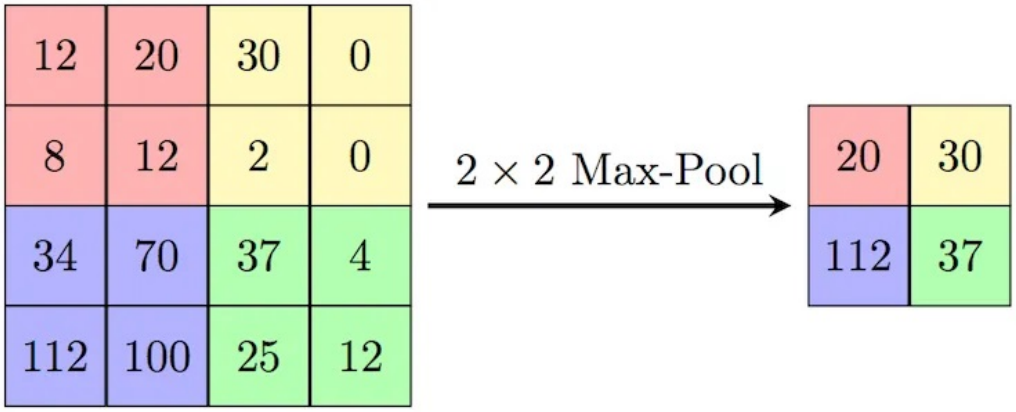
After obtaining the feature map, an activation function is applied. The activation function most commonly used in this context is the Rectified Linear Unit (ReLU), defined by f(x) = max(0, x) so that negative values are set to 0, while positive values remain unchanged.

1.2 Pooling Layers

The pooling layer does a downsampling operation along the spatial dimensions (width, height) dividing the matrix in submatrices and extracts the most important informations from each of them, creating a result matrix called pooling matrix, the extracted informations depends on the type of pooling used between the following 3:

1.2.1 Max Pooling

Max pooling extracts from each submatrix only the features which have the highest value among those contained within.



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

Figure 2: Max pooling on a feature map

1.2.2 Average Pooling

Average pooling extracts from each submatrix the average value of those contained within.

1.2.3 Sum Pooling

Sum pooling extracts from each submantrix the sum of the values contained within.

1.3 Dense Layers

The obtained pooled feature maps then undergo a process called Flattening, which reduces them to a single column, which is then used as input for the dense layers which are series of layers where each node is connected to every other node in the adjacent layers (here as well, the weights and biases are optimized through backpropagation).

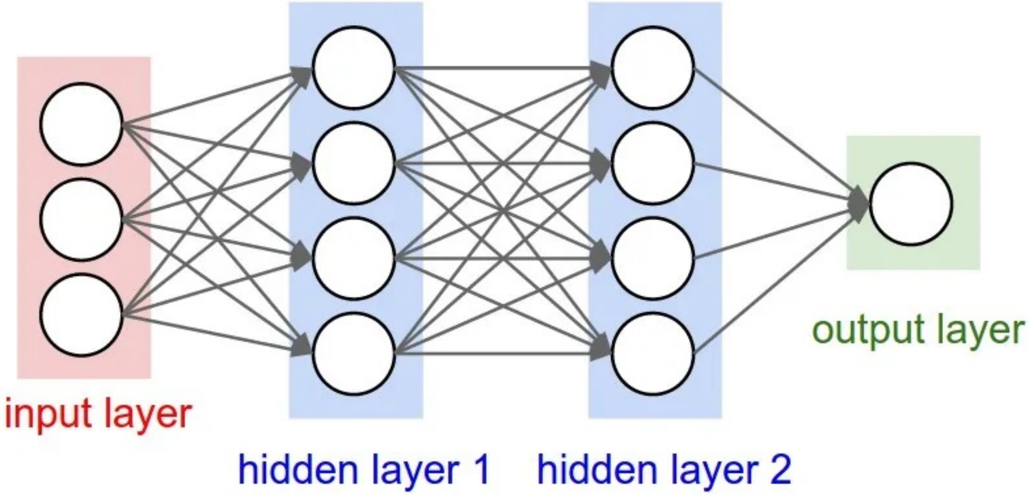


Figure 3: Example of dense layers

The final layer is the output layer which contains only one node which gives either 1 or 0 as output as it is using sigmoid as an activation function. This value determines wether the image belongs to one class rather than the other.

1.4 Backpropagation and Optimizers

In the initial phase of training, some parameters essential for the good quality of the model are randomly generated. In this initial configuration, the model performs image classification and computes the loss function, which represents how much the model's predictions deviate from reality. The objective of backpropagation is to minimize this loss function.

The backpropagation is a procedure that starts from the end of the neural network: for each parameter, the gradient descent is calculated, which is the derivative of the loss function with respect to the parameter, the new value of the parameter is then determined by the difference between the value taken in the previous step and the product of the derivative value and the learning rate. In the initial phase, the learning rate is high, but will decrease after several steps.

The optimizer employed in the project is Adam, the learning rate defined in the code will be always equal to 0.0001.

2 Project Phase 1: Settings

In this section, we will discuss the work done on the image sets to prepare them for the training on the various models realized.

2.1 Data Pre-Processing

The dataset given contained 3199 chihuahua images and 2718 muffin images, these images were then organized into a dataframe which indicated the path and label for every single image and finally separated into 3 sets:

- a test set which contained the 20% of the total images.

The remaining images where split again in 2 sets:

- a training set which contained the 85% of the images.
- a validation set which contained the remanining 15%.

From these sets we created image generators using the ImageDataGenerators class from Keras with normalized pixels. The parameters to create these image generators are given by the method flow_from_dataframe:

- dataframe: reference to the dataframe.
- x_col: the dataframe column containing the path to the image.
- y_col: the dataframe column containing the image label.
- target_size: tuple specifying the height and width of the images.

- batch_size: the number of splits performed by the image generator. This split is crucial during model training for optimizing parameters based on the loss function calculated for each batch. The process repeats until a stopping criterion is met. The split is also done for the test set for efficiency, calculating the loss function for each batch and averaging for performance evaluation.

- class_mode: the type of classes.

- color_mode: specifies wether image is RGB or grayscale.

- shuffle: shuffles images after all batches have been examined.

In the project code, the target size is set to (224, 224), the batch size is 16, class mode is set to "binary", and color mode is set to RGB for all generators. For the training and validation generators, shuffle is enabled, while for the test generator, it's disabled.

Two additional generators are created for data augmentation using additional parameters, but those will be explained in detail as part of the data augmentation model subsection in section 2

2.2 Evaluation Metrics and Stopping Criteria

The model evaluation is done by observing the learning curve plots of loss and accuracy through epochs. The loss function chosen is the binary cross-entropy that measures the deviation between predicted probability distribution and the true labels of the image set. It also emphasizes errors, especially when the class probability associated with the observation deviates greatly from the true label. High values of binary cross-entropy indicate poor model performance, while low values indicate good model adaptability.

The other parameter to observe, accuracy, is, on the other hand, simply the proportion of correctly classified observations.

By examining the plot curves we can determine 3 types of fit for a model and evaluate its performance accordingly:

- Underfitting: the training loss curve increases, while the validation loss curve is higher and does not decrease.

- Overfitting: the training loss curve is low while the validation loss is much higher.

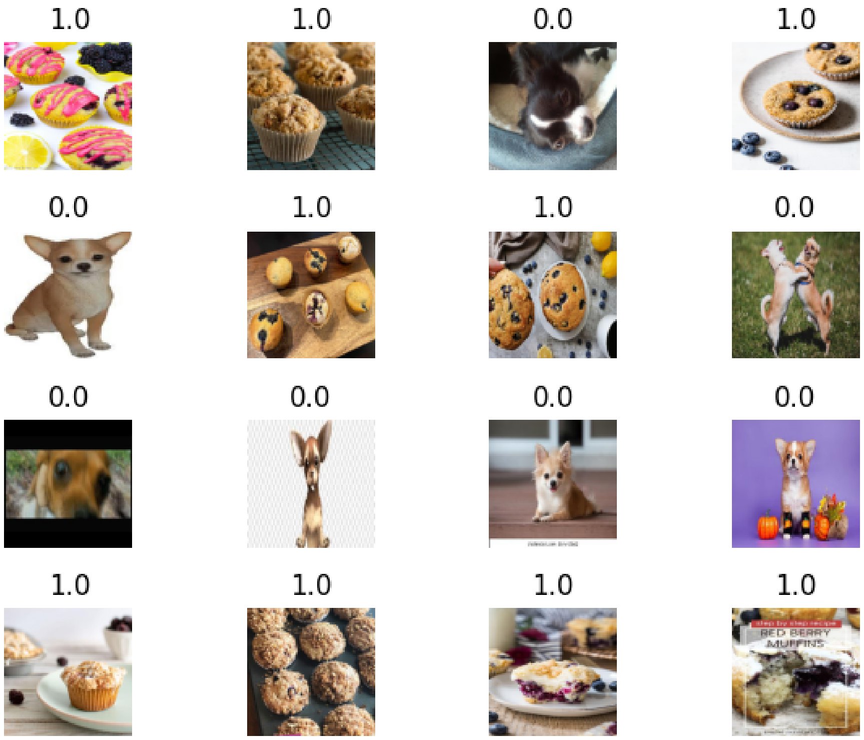


Figure 4: Example batch of normalized training images with their respective binary classification using one-hot encoding (0.0 = chihuahua; 1.0 = muffin)

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(p_i) + (1 - y_i) log(1 - p_i))$$

Figure 5: formula for binary cross-entropy loss function

- Good Fit: both curves are decreasing with a similar trend.

A final parameter to check that may lead to a bad fit for the model is the number of epochs: if there are too few the model doesn't learn enough and underfits, while with too many epochs lead the model to memorizing the training data and overfit. The stopping criteria introduced stops the training process when the validation loss stops improving after 5 epochs, in doing so avoids overfitting ensuring that the model doesn't memorize the datas too well. Considering this criteria, the max number of epochs has ben set to 50 to ensure balance between time spent and model performance.

3 Project Phase 2: the Models

In this section, we will discuss the structure of each model and the result of training on them, evaluating performances and results.

3.1 Model 1: Simple Model

3.1.1 Structure

This first model adopts a simple architecture with the following layers:

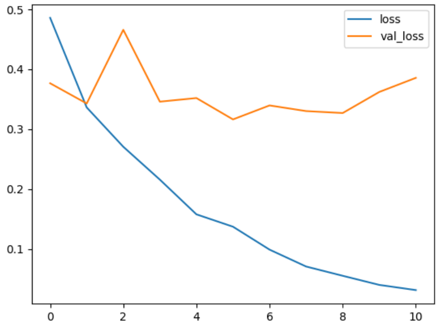
- Convolutional Layer 1:
  - 32 filters of size 3 x 3 with padding to maintain matrix size
  - ReLU activation function
- Pooling Layer 1:
  - Max Pooling
  - Pooling size 2 x 2 with stride 2
- Convolutional Layer 2:
  - 32 filters of size 3 x 3 with padding to maintain matrix size
  - ReLU activation function
- Pooling Layer 2:
  - Max Pooling
  - Pooling size 2 x 2 with stride 2
- Dense Layer
  - 64 Dense neurons
  - ReLU activation function
- Output Layer

- Sigmoid activation function

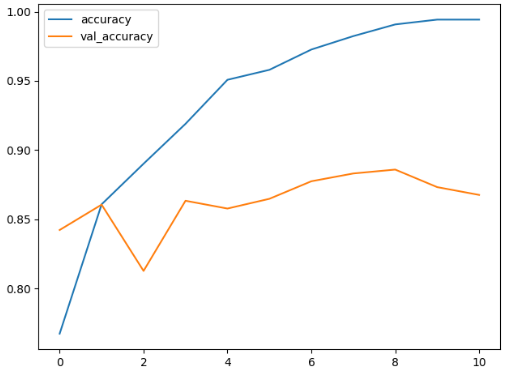
The input data utilized consisted of normalized images as seen previously.

3.1.2 Results

After model fitting, results were shown after 11 epochs:



Loss and Validation Loss curve



Accuracy and Validation Accuracy curve

The results show clearly an issue of overfitting that could be caused either by the excessive simplicity of the model architecture or by the model's excessive adaptation to the details of the training images.

3.2 Model 2: Data Augmentation

To address the overfitting problem of the first model, it was decided to employ data augmentation on the data sets to apply realistic tranformation to the images. This process would add more varied and complex data to be memorized by the model.

The transformations applied include random rotation within a range of -10 to +10 degrees, width shift of up to 30% both to the right and left, height shift of up to 30% both upwards and downwards, a stretch transformation of 15%, a zoom range of 20%, channel color shift (expressed in pixel intensity) of up to 10, and random horizontal flipping.

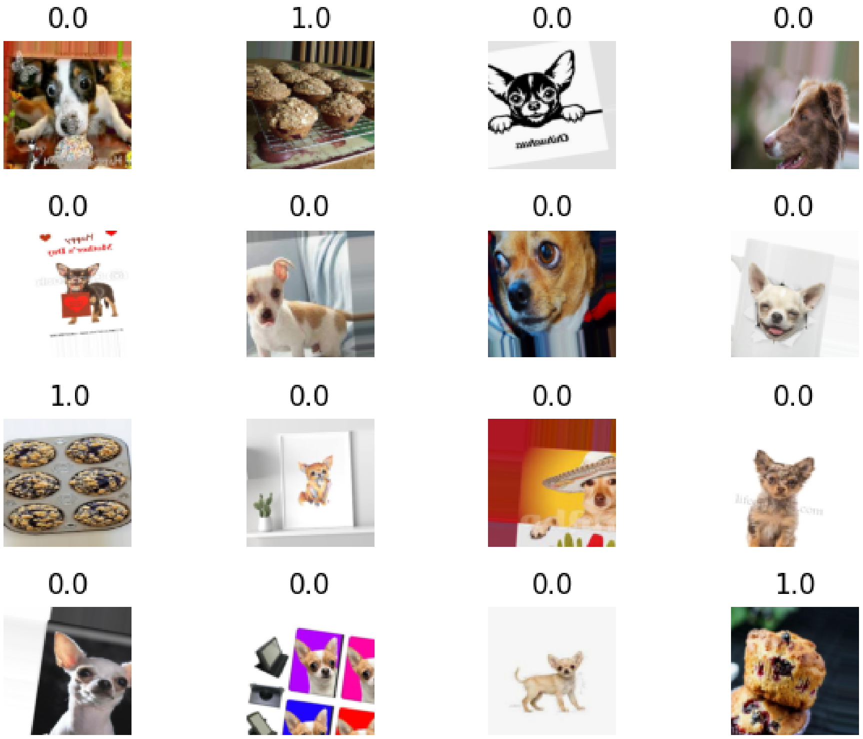


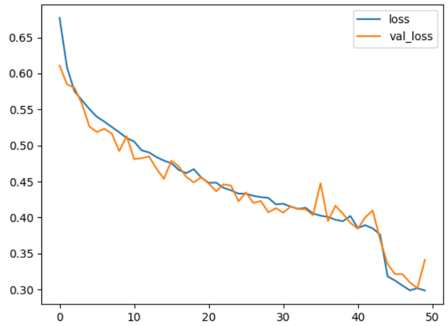
Figure 7: Example batch of augmented training images with their respective binary classification using one-hot encoding (0.0 = chihuahua; 1.0 = muffin)

3.2.1 Structure

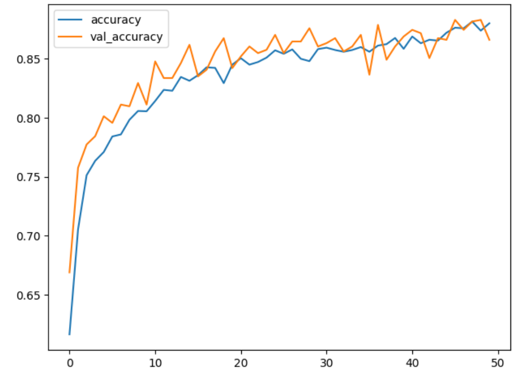
Aside from the augmented input images no modifications were made to the model structure that remains the same as the previous model.

3.2.2 Results

After model fitting, results were shown after 50 epochs:



Loss and Validation Loss curve



Accuracy and Validation Accuracy curve

The results are satisfactory and show a significant improvement.

3.3 Model 3: Dropout Layer

To reduce the model's dependence on the training data and address overfitting issues, this model introduces dropout layers. This type of layer deactivates a portion of neurons randomly during training, preventing the model from overly focusing on image details. The model also utilizes augmented images as input.

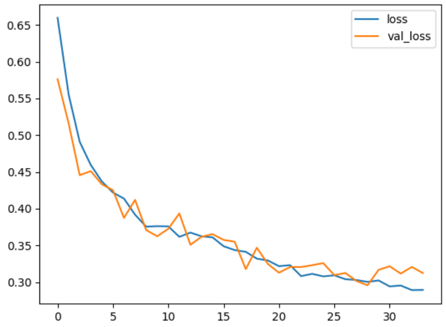
3.3.1 Structure

The architecture of this model is the same as the first and second. However, after each Pooling layer, a Dropout layer has been added with the following settings:

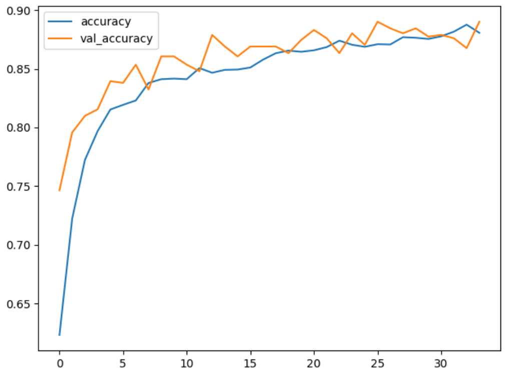
- Dropout Input Layer (located after first Pooling layer):
  - 10% of neurons deactivated
- Dropout Hidden Layer (located after second Pooling layer):
  - 30% of neurons deactivated
- Dropout Dense Layer (located after Dense layer):
  - 30% of neurons deactivated

3.3.2 Results

After model fitting, results were shown after 33 epochs:



Loss and Validation Loss curve



Accuracy and Validation Accuracy curve

The results are satisfactory and show negligible overfitting but are slightly worse than the previous model results.

4 Project Phase 3: Hyperparameter Tuning

Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. To determine the optimal hyperparameters we employ a process called hyperparameter tuning that, in this project is possible thanks to the KerasTuner library.

The first step involves defining a function for building the model. In this function, the architecture follows the structure of the third model, but the hyperparameters are not predefined. Instead, they are set to specific ranges with defined step sizes.

4.1 Model Structure

- Convolutional Layer 1:
  - 16 to 64 filters with step of 16 and size 3 x 3 with padding to maintain matrix size
  - ReLU activation function
- Pooling Layer 1:
  - Max Pooling
  - Pooling size 2 x 2
- Dropout Layer 1:
  - 5 to 20% of neurons deactivated with step of 5%
- Convolutional Layer 2:
  - 32 to 128 filters with step of 32 and size 3 x 3 with padding to maintain matrix size
  - ReLU activation function
- Pooling Layer 2:
  - Max Pooling
  - Pooling size 2 x 2

- Dropout Layer 2:
  - 20 to 50% of neurons deactivated with step of 10%
- Dense Layer
  - 64 to 256 Dense neurons with step of 64
  - ReLU activation function
- Dropout Layer 3:
  - 20 to 50% of neurons deactivated with step of 10%
- Output Layer
  - Sigmoid activation function

The tuner chosen is the Bayesian optimization with the goal of maximizing validation accuracy. This tuner operates as follows:

1. Execution of model and measurement of Validation Accuracy: Initially, the model is trained using a random set of hyperparameters, and its accuracy is evaluated on the validation set.
2. Creation of Surrogate Function: Using the results obtained from initial training steps, a surrogate function is constructed to approximate the relationship between hyperparameters and validation accuracy.
3. Selection of new Hyperparameters: New hyperparameters are chosen to explore new search regions to exploit the ones with promising objective function values.

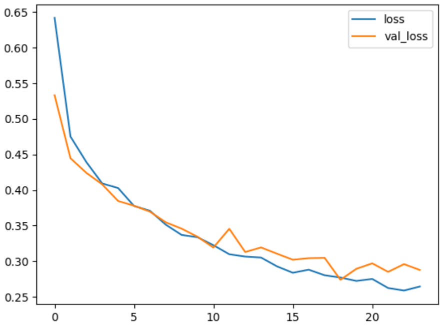
Step 3 is iterated until convergence or until the stop criteria is met. Once the process is completed every set of hyperparameters is evaluated and the one which optimizes the objective function (in our case validation accuracy) is chosen. If convergence or stop criteria are not reached, then a new surrogate function is created and step 3 starts anew. In this project the maximum number of iteration allowed is 4.

4.2 Results

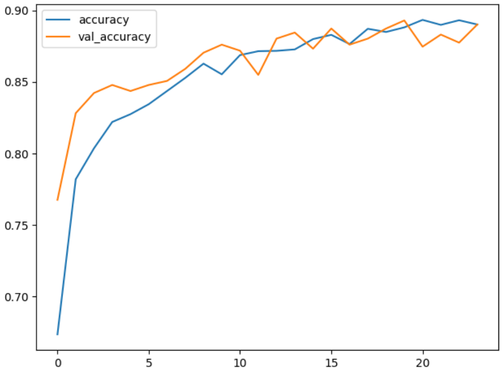
After the process the Hyperparameters values obtained were the following:

- Convolutional Layer 1:
  - 64 filters
- Dropout Layer 1:
  - 5% of neurons deactivated
- Convolutional Layer 2:
  - 128 filters
- Dropout Layer 2:
  - 20% of neurons deactivated
- Dense Layer
  - 192 neurons
- Dropout Layer 3:
  - 20% of neurons deactivated

After epochs the model evaluation curves were the following: The model exhibits a good fit and we can proceed with evaluating the results using the test set.



Loss and Validation Loss curve

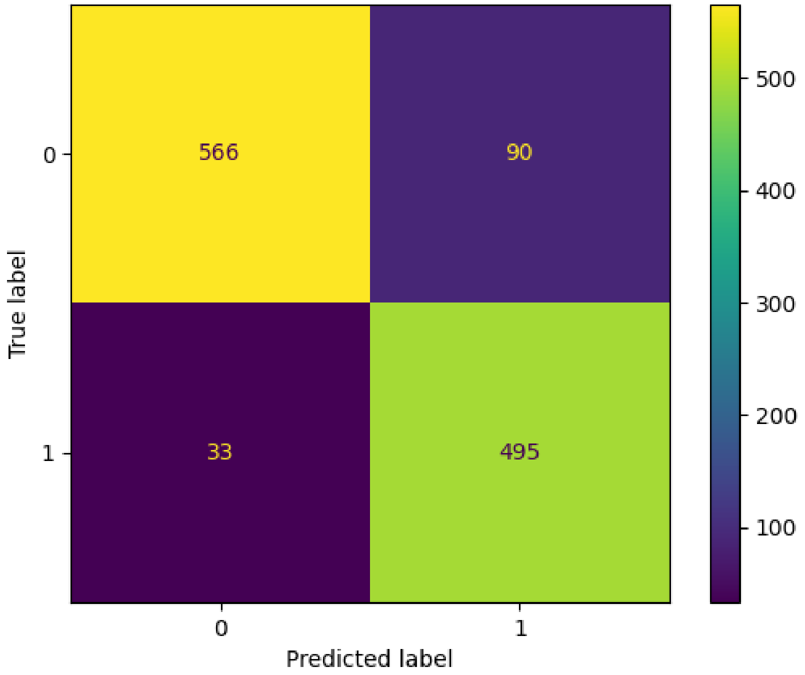


Accuracy and Validation Accuracy curve

5 Project Phase 4: Evaluations and Results

5.1 Predictions on Test Set

To visualize cleary the performances of our tuned model we utilize a tool called Confusion Matrix, which appears as following:

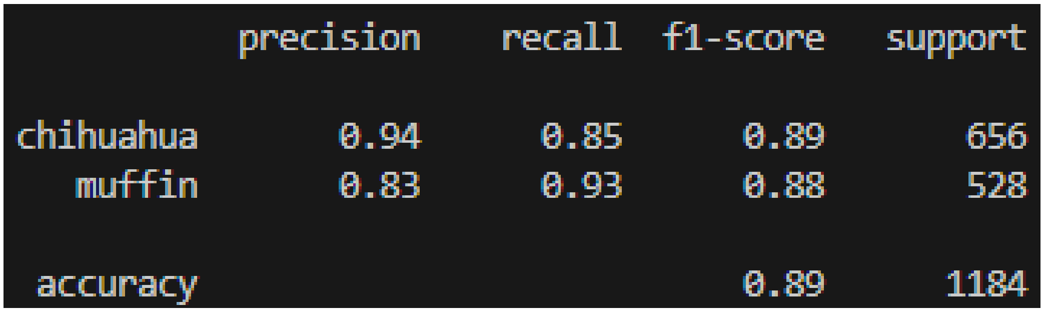


The matrix reads as follows:

- 566 images of Chihuahua were correctly classified.
- 90 images of Chihuahua were misclassified as muffins.
- 495 images of muffins were correctly classified.
- 33 images of muffins were misclassified as Chihuahua.

Furthermore, we can utilize some metrics obtained form the prediction to help us understand the quality of classification:

- Accuracy is a measure that allows us to understand how often the model predicts correctly and is represented by the proportion of correct classifications over the total predictions.
- Precision allows us to understand how often a single class is predicted correctly and is given by the fraction of the total number of observations correctly classified for a given class over the total number of observations classified as belonging to the class.
- Recall is a metric used to measure how good the model is at identifying all instances belonging to a single class and is given by the fraction of correct classifications for a single class over the total number of instances belonging to that class.
- F1-score is a combination of precision and recall and computes how many times the model predicted instances correctly across the entire dataset.

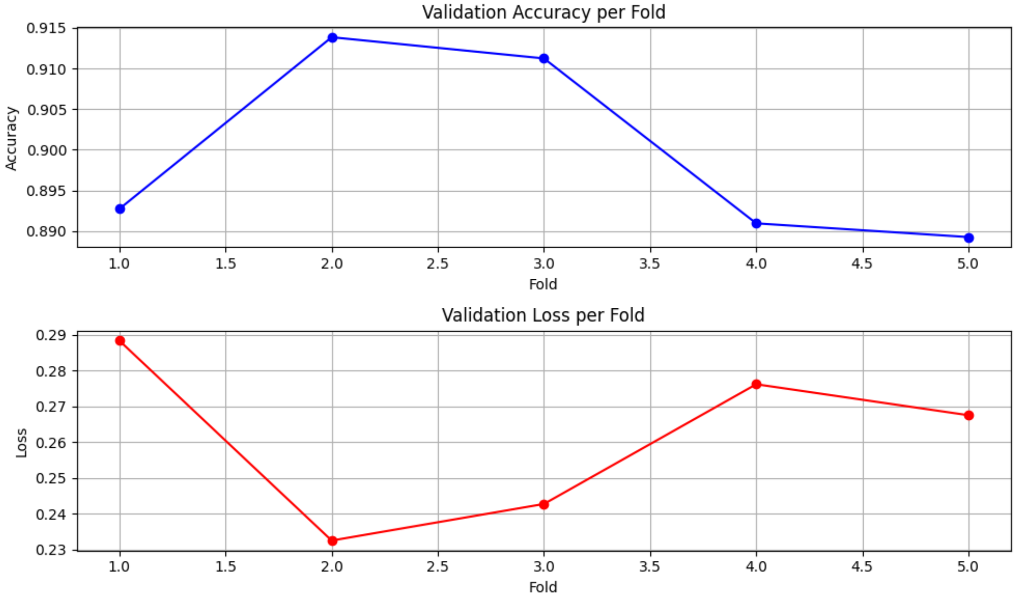


```
              precision    recall  f1-score   support

   chihuahua       0.94      0.85      0.89       656
      muffin       0.83      0.93      0.88       528

    accuracy                           0.89      1184
```

Looking at these values we can observe that the precision value is slightly higher for chihuahuas, while recall is higher for muffins: this means that the model has a small tanedency to missclassify images of chihuahuas as muffins. Aside from this minor imprecision the values are good and we can conclude that the model is surely improvable given its simplicity but nontheless sufficiently solid.

5.2 Cross Validation

K-fold cross-validation implies the division of the training set into k subsets called folds. The selected model is trained k times, each time using k-1 folds as the training set and the remaining fold (which is different every iteration) as the validation set. Through cross-validation, we ensure that the model architecture performs well when exposed to diverse input images. Below the results obtained from cross validation:



The average validation loss and validation accuracy value obtained is 0.90 and the average validation loss is 0.26. Both are very similar to the results obtained from the tuned model. Based on that we can affirm the robustness of the model.