Università degli Studi di Milano

Data Science for Economics

Chihuahuas vs Muffins binary classification

Student:
Jing Wang

ACADEMIC YEAR 2023-2024

Contents

1 Introduction **3**

2 Data preparation **4**

3 Convolutional Neural Networks **8**
3.1 A simple model 8
3.2 Another architecture 9
3.3 1° hyperparameter tuning 11
3.4 A deeper architecture 12
3.5 2° hyperparameter tuning 14

4 Final model **15**
4.1 5-fold Cross Validation 15
4.2 Confusion matrix 15

5 Conclusions **17**

6 References **18**

1 Introduction

In this experimental project we are asked to use library Keras to train a neural network for the binary image_based classification of Chihuahuas and muffins from a dataset downloadable from the following link: Kaggle link
Despite its playful context, this task illustrates the complexities of image classification and the potential for error when objects share similar visual features. Chihuahuas and muffins can have striking visual similarities, with color, texture, and even shape sometimes overlapping, leading to confusion.
The primary objective of this project is to design, train, and evaluate a Convolutional Neural Network (CNN) for binary classification to determine whether a given image is a chihuahua or a muffin. The specific goals are to:

- Develop a simple but effective CNN architecture.
- Train the model on a dataset of images representing chihuahuas and muffins.
- Evaluate the model's performance on the separate test set.
- Explore methods to improve the model's accuracy, including regularization, and hyperparameter tuning, 5_fold cross validation.

This project serves as an introductory experiment to demonstrate the capabilities of CNNs in image classification while highlighting some common challenges, such as overfitting and the need for careful model tuning. The results and insights gained from this experiment can contribute to a broader understanding of machine learning applications and inspire further exploration into more complex classification problems.
The codes are available in the following page of GitHub: Github link

2 Data preparation

Upon downloading the dataset from Kaggle website, we found that the data is pre-divided into two separate parts: a training set and a test set. This division is crucial for machine learning projects as it allows us to train our models on one set of data and then evaluate their performance on another, unseen set.

Let's start with the training set, which is used to train our model. It contains a total of 4,733 images, with 2,559 images of chihuahuas and 2,174 images of muffins. This distribution indicates that the training set has a slightly higher number of chihuahuas compared to muffins, but overall the classes are relatively balanced.

Next, we have the test set, which is used to validate the model's performance and assess how well it generalizes to new data. The test set has a total of 1,184 images, comprising 640 images of chihuahuas and 544 images of muffins. Again, the distribution shows a slightly higher count of chihuahuas but remains reasonably balanced.

This class distribution in both the training and test sets is visualized in the following plot. It's important to maintain a balanced dataset to avoid bias in the model's learning process and ensure accurate evaluation during testing. By having distinct training and test sets, we can train the model on one dataset and test its accuracy and robustness on the other, which helps to prevent overfitting and evaluate its performance more effectively.

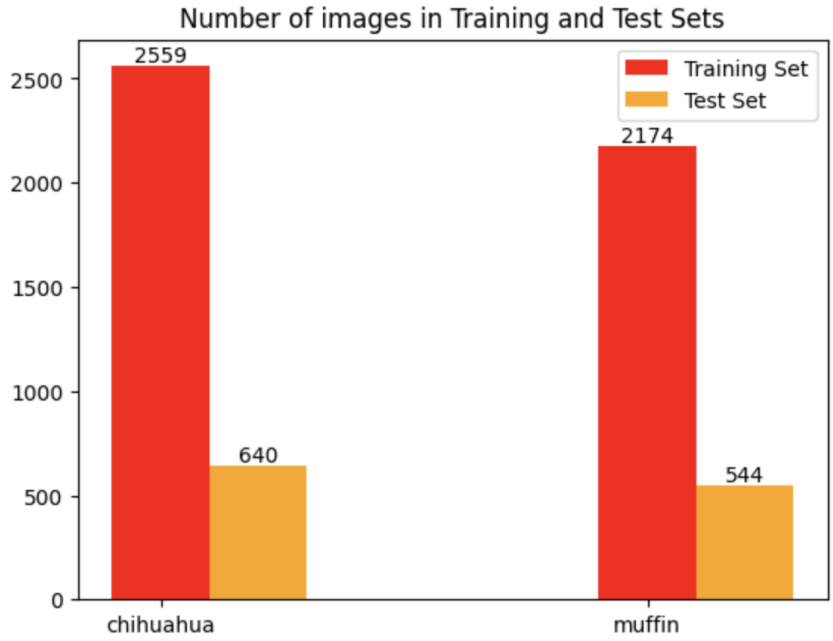

Figure 1: Class distribution

To get a general sense of the characteristics and visual features of the images in our dataset, some examples are provided from each of the two categories: chihuahuas and muffins. These sample images offer a quick visual overview, helping us understand the kinds of details the model will need to identify and differentiate.
Displaying examples from each category is an useful step, as it allows us to visualize the data and recognize any potential similarities or differences between the categories.

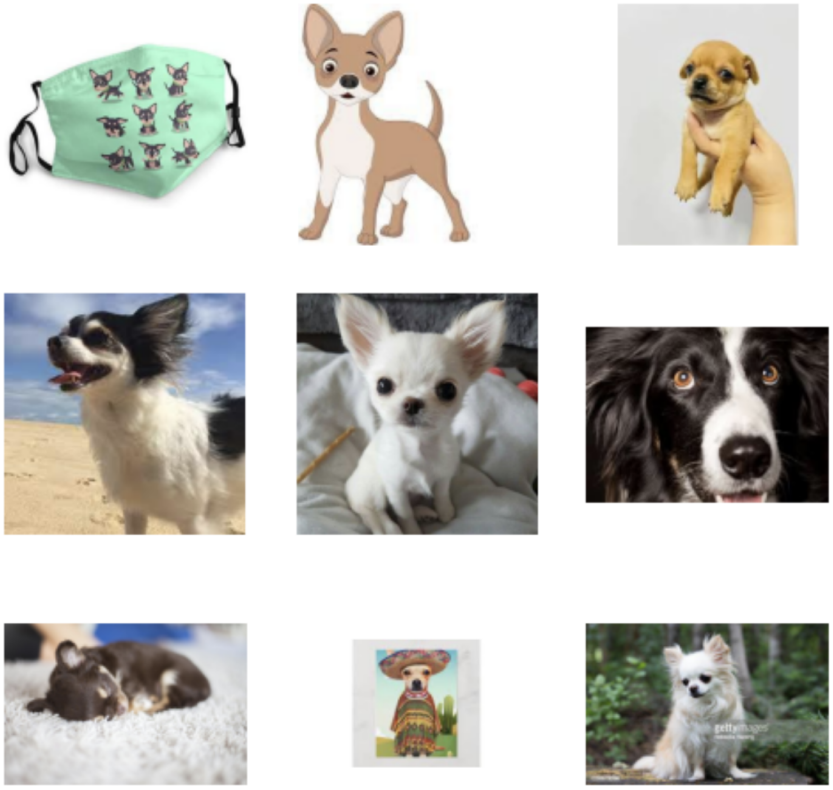

Figure 2: Chihuahuas

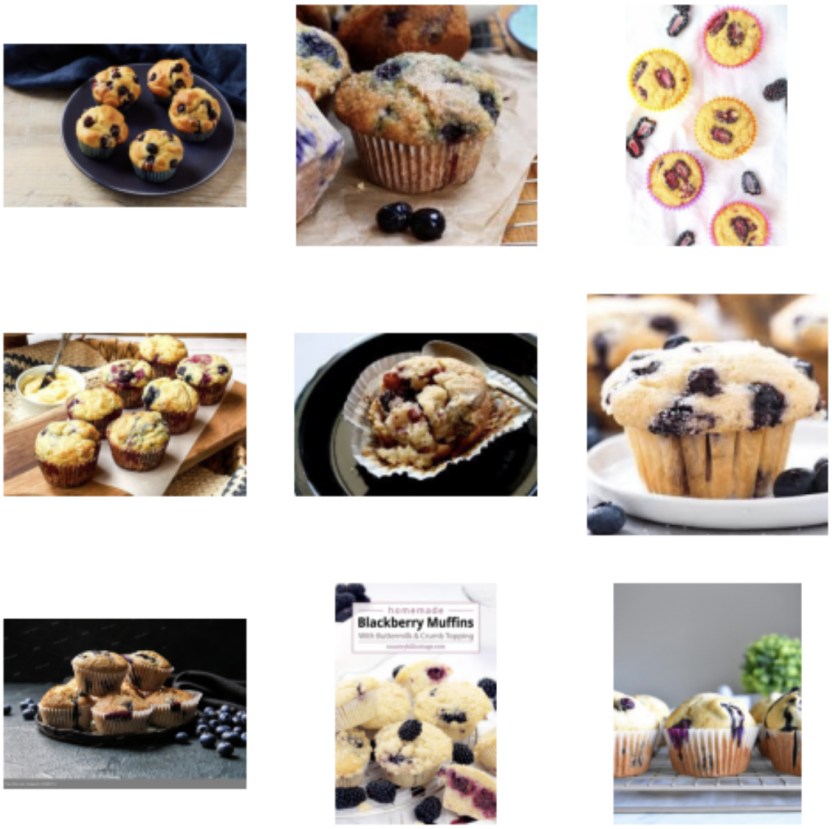

Figure 3: Muffins

Images are represented as three-dimensional arrays of pixels, where the three dimensions correspond to height (h), width (w), and depth (d). The height and width vary depending on the size of the image, while the depth represents the color information. For RGB color images, depth is typically 3, indicating the presence of Red, Green, and Blue color channels. In grayscale images, the depth is 1, indicating only a single intensity channel.
Pixel values in these images generally range from 0 to 255, where 0 represents black, 255 represents white, and values in between represent varying levels of intensity or color. This range applies to both RGB and grayscale formats.
In this project, the images were initially in JPG format and contained color information. However, they were transformed into grayscale, reducing the depth from 3 (RGB) to 1 (grayscale). This conversion simplifies the data and reduces computational complexity, as the model only needs to process one intensity channel rather than three color channels.
Additionally, the images were scaled down to smaller dimensions (64 x 64), which not only saves computational resources but also helps prevent overfitting by focusing on essential features rather than high-resolution details. By transforming and scaling the images, we aim to create a more manageable dataset that retains the critical information needed for accurate classification.
Then, each of them is represented as a 2D array of pixel values where each value corresponds to the intensity of a pixel.
Before analyzing the data with our model, all the data is shuffled. Shuffling is a critical

preprocessing step that randomizes the order of the dataset. This process serves several purposes:

- Reducing Bias: Shuffling helps prevent any inherent bias from affecting the model's learning process.
- Avoiding correlations: By shuffling the data, we minimize the risk of creating artificial correlations between consecutive samples. This is important because many machine learning algorithms are sensitive to the order of the data.
- Ensuring Better Generalization: shuffling ensures that the training set contains a diverse mix of examples, which can help the model generalize better to unseen data. It reduces the likelihood of overfitting to specific patterns in the dataset.

Thus, by shuffling the data, we aim to create a more robust and unbiased dataset that allows the model to learn effectively and generalize well, leading to better overall performance in this task.
All the data is normalized by dividing the pixel values by 255. This common practice in image processing scales the pixel intensity values from a range of 0-255 to a new range of 0-1.
Normalization is a key step in preparing image data for machine learning algorithms, particularly for Convolutional Neural Networks (CNNs). By normalizing the pixel values, we achieve several benefits:

- Consistent scale
- Improved convergence
- Reduced computational complexity

3 Convolutional Neural Networks

After completing the data preprocessing step, it is possible to begin our model analysis. For this image_based binary classification task, the most suitable algorithm is the Convolutional Neural Network (CNN), which is particularly effective for classifying visual data. The models can extract relevant features from the images and classify them accurately by selecting the appropriate architecture and hyperparameters.
CNN consists of a large number of hidden layers, in which each layer is containing a large number of neurons. Each neuron of a given layer connects with all the neurons in the previous layer but works individually. Each convolutional neural network consists of three components:

1. Convolutional layer (Conv), as the first layer is used to scan incoming inputs and distinguish the weight, height, and depth of each image. In other words, this layer is the core of the neural networks, which has hyperparameters that include the main processing, number of filters, the size of each filter, and a task to activate the ReLU layer.

2. Pooling layer, which is the layer that is located between the Conv layers and is in charge of reducing the numbers of network parameters while retaining the maximum values of input.

3. Fully Connected layer, which is the third component, and this layer is the last layer of CNN architecture to specify the class probability, and as the layer name suggests, each neuron in this layer is associated with all the neurons in the previous layer.

In the model selection, different architectures and hyperparameters were experimented with, and each adjustment was based on the previous model's performance.

3.1 A simple model

As a first try, a simple CNN model was performed. Its architecture was structured as follows:

- Conv2D layers: these are the convolutional layers that learn to extract features from the input images. The first layer has 16 filters, then the following has 32 filters. The filter size is (3,3) for all the layers. Here the activation function used is ReLu, which helps the network to learn more complex patterns.

- MaxPooling2D layers: they are used to perform downsampling, reducing the spatial dimensions of the input data and helps in decreasing the number of parameters and computations in the network. It can lead to faster training time.

- flatten layer: it is used to transform the multi-dimensional output of convolutional and pooling layers into a one-dimensional vector. This layer is crucial, because fully connected(dense) layers require one- dimensional input.

- Dense Layers: These are fully connected layers responsible for performing the classification task based on the features extracted by the preceding layers. The first dense layer consists of 32 units and uses the ReLU activation function. Following this, there is a dense layer with a single unit utilizing the Sigmoid activation function, which outputs the probability that the input image is a muffin (muffin = 1).

All the models were trained using two key metrics: binary cross-entropy as the loss function and accuracy as the performance measure. Binary cross-entropy quantifies the difference between the predicted labels and the actual class labels, guiding the model's optimization process. Accuracy reflects the percentage of correctly classified images, providing a direct measure of the model's classification performance. By monitoring both metrics, we ensure that the model is effectively minimizing the loss while also achieving high accuracy in classifying input images.
The model was then compiled using Adam optimizer, which is an adaptive learning rate optimization algorithm. Its default value is equal to 0.001.
The following two plots evaluate the model's performance, as discussed in the previous paragraph. As shown, this basic model does not perform well, exhibiting significant overfitting. While it achieves a high training accuracy of 97.67%, the validation accuracy is notably lower at 82.28%. Additionally, the training loss continues to decrease, but the validation loss starts increasing after the 6th epoch. This pattern indicates that the model is fitting the training data too closely and struggling to generalize to new, unseen data.

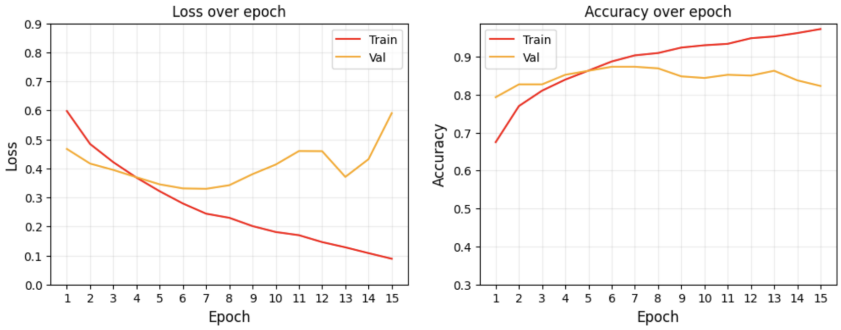

Figure 4: Loss and Accuracy plots

3.2 Another architecture

I decided to modify the CNN architecture in the second model, because Conv2D layers with 16 filters are not commonly used, since it was too simple to capture the data's underlying patterns effectively. I replaced the first Conv2D layer with a new Conv2D layer containing 32 filters and added an additional Conv2D layer with 64 filters. These changes were made to observe their impact on the model's performance. The results obtained are as follows:

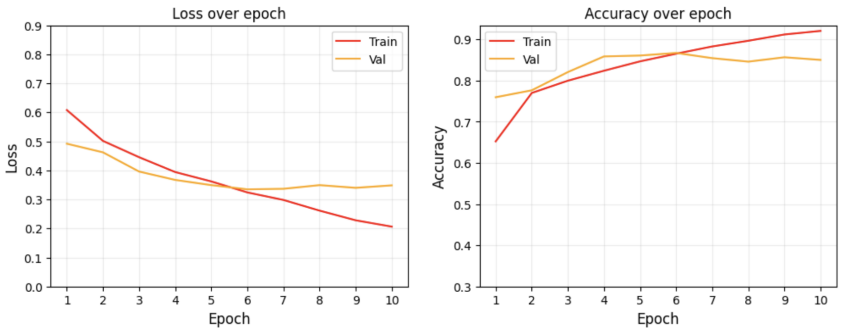

Figure 5: Loss and Accuracy plots

Comparing this model to the previous one, we can see an improvement in performance. However, there is still evidence of overfitting.

In the third model, I added a Dropout layer with dropout rate equal to 0,5, which is a regularization technique that is easy to implement and to significantly reduce overfitting in neural networks. As shown in the following plots, this modification helps the model learn more generalizable features.

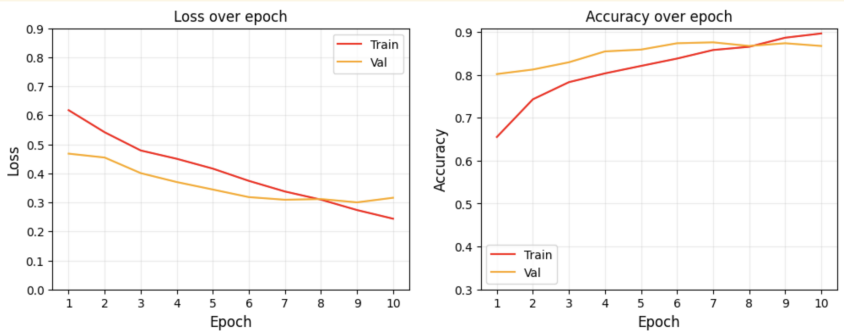

Figure 6: Loss and Accuracy plots

As we can see from the second image of the third model, the training accuracy is 89.20% and the validation accuracy is 86.71%. The smaller gap between these accuracies indicates a reduced overfitting issue.
However, despite this model's improvements compared to the previous two models, the validation loss begins to increase after the 9th epoch. This suggests that there is still room for improvement in addressing overfitting.

The model 4 introduces two elements compared to the previous model:

- Learning rate: the learning rate for the Adam optimizer has been set to 0.0001, which is lower than the default value. This smaller learning rate can help the model converge more steadily, potentially leading to better performance.
- Early Stopping: an Early Stopping callback is used during training to monitor the validation loss. With a patience of 15 epochs, this means that training will stop if there is no improvement in the validation loss for 15 consecutive epochs.

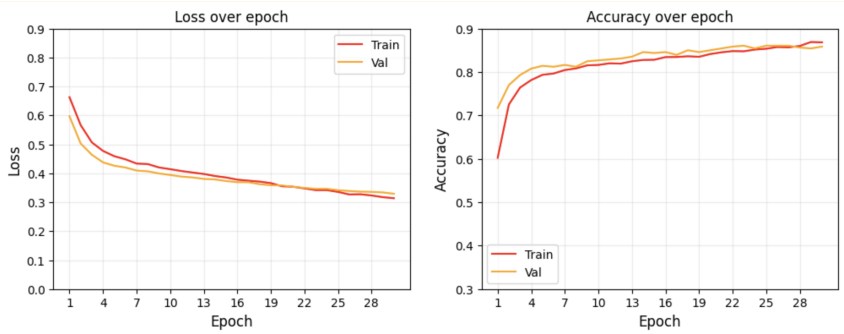

Figure 7: Loss and Accuracy plots

I also decided to reduce the dropout rate to 0.3, because a lower dropout rate can help the model retain more information during training, potentially improving its performance while still mitigating overfitting.
The plots show us great improvement in both training and validation accuracy. Any overfitting is minimal.

In Model 5, a Batch Normalization layer and L2 regularization are added compared to Model 4. The Batch Normalization layer helps normalize the inputs to the final layer, which can improve generalization. L2 regularization, with a lambda of 0.001, is added to the Dense layer to help prevent overfitting.

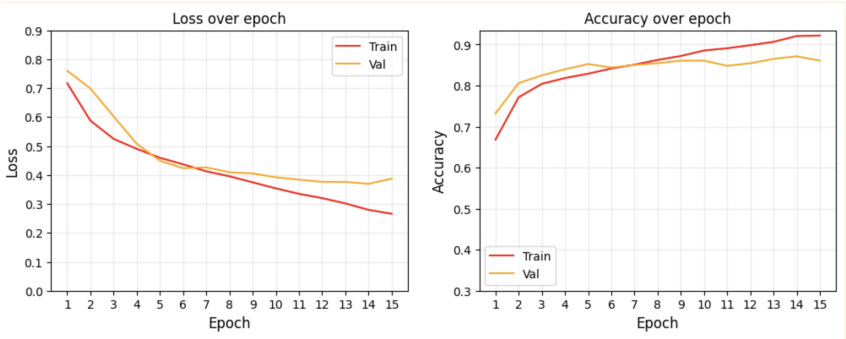

Figure 8: Loss and Accuracy plots

In this case, the training accuracy achieves 90,72% and validation accuracy 86,08% , both of which are higher than in model 4. However, the overfitting issue persists.

In Model 6, I introduced an additional Dense layer with 64 units, which increases the number of parameters in the model, enhancing its ability to learn and represent more complex patterns and relationships in the data.

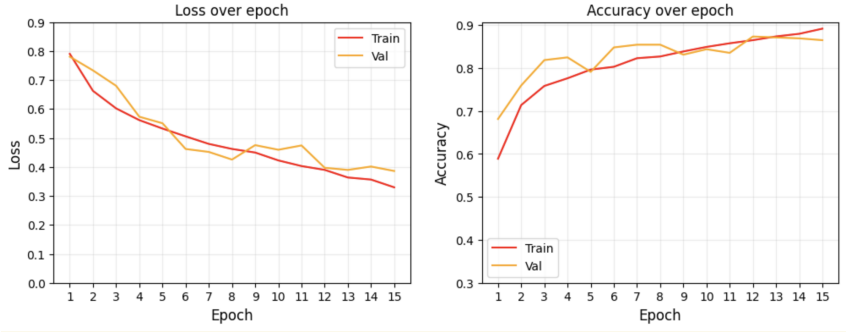

Figure 9: Loss and Accuracy plots

The plots show a good performance of the model with the training accuracy of 88.54% and validation accuracy of 86.50%.

3.3 1° hyperparameter tuning

In machine learning, as we know, hyperparameters are crucial to determine the algorithm's performance, because they influence directly the learning process.

In my project, I first conducted hyperparameter tuning through numerous manual attempts. Subsequently, I utilized the KerasTuner framework, using hyperparameter values that demonstrated better performance in previous experiments as possible choices for the tuner. KerasTuner employs a search algorithm to systematically test and evaluate different hyperparameters, selecting the best ones based on the highest validation accuracy.
Instead of using random search, which is the commonly used technique in the Frequentist approach to select the best hyperparameter values, I decided to use Bayesian optimization, because it takes into consideration also past evaluation results in the search for optimal hyperparameters. I tried to tune the model 6 in order to see if it is possible to reach higher stability and training and validation accuracies.

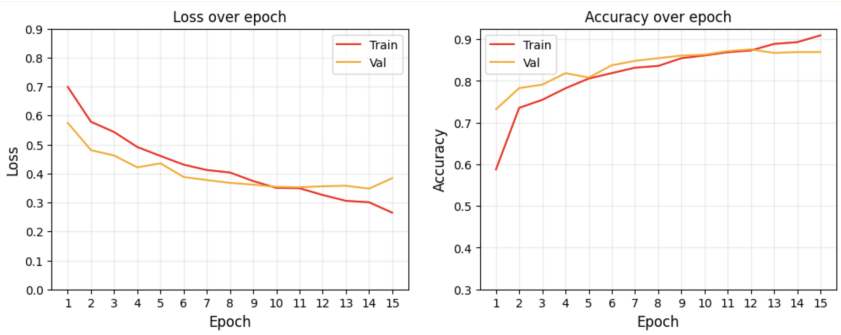

Figure 10: Loss and Accuracy plots

The result that I obtained after hyperparameter tuning is quite good. The model seems stable with training accuracy at 91.07% and validation accuracy at 87.55%.

3.4 A deeper architecture

I also experimented with deeper models. In model a we have the following architecture:

- 6 Conv2D layers with 32, 32, 64, 64, 128 and 128, respectively, each with a kernel filters of size (3 x 3). Each layer use the ReLu activation function.
- 3 MaxPooling2D layers, one after every pair of Conv2D layers.
- 3 Droupout layers with a rate equal to 0.4, one after each MaxPooling2D layer.
- A flatten layer
- a Dense layer with 64 units and ReLu activation, followed by a Dropout layer with a rate of 0.6, a Batch Normalization layer, and a final dense layer with 1 unit and Sigmoid activation function.

The model also employs L2 regularization on the first Dense layer with lambda equal to 0.001 and a learning rate equal to 0.0005.

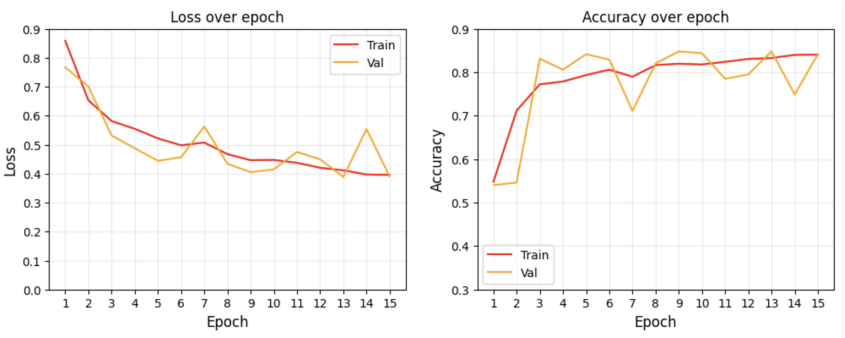

Figure 11: Loss and Accuracy plots

Clearly, the model is more complex than all previous models, but the model presents some overfitting problems.

The same model is then evaluated with a increased batch size, equal to 64. In the all models, I always set this size equal to 32. This increase could provide a better estimate of the gradient and potentially accelerate the training process.
As we can see from the plots, training efficiency, stability, and generalization were not improved significantly as the batch size changed, so I decided to keep it equal to 32 for all the following models.

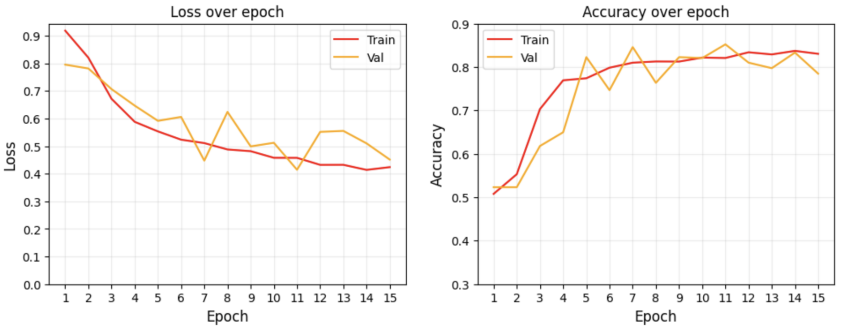

Figure 12: Loss and Accuracy plots

Then, I introduced some adjustments to this model by setting the L2 regularization lambda and learning rate both to 0.0001. Additionally, I added padding="same" to all Conv2D layers to preserve the spatial size of the input through all the convolutional layers and prevent excessive dimensionality reduction. Furthermore, I increased the units in the Dense layers to 128, 64, and 32, respectively.

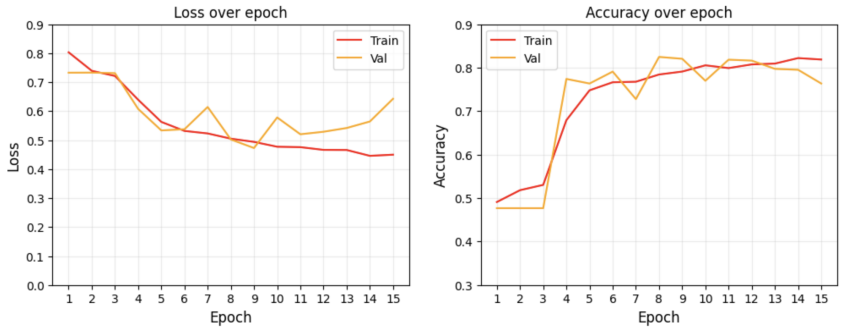

Figure 13: Loss and Accuracy plots

There is still a slight overfitting in the last epochs. Compared to the simpler models, both the training and validation accuracies are quite low. Therefore, I decided to proceed with hyperparameter tuning to select the optimal hyperparameter values and achieve higher validation accuracy as the evaluation metric for the model.

3.5 2° hyperparameter tuning

For the deeper architecture, I attempted to tune the model shown in Figure 12, hoping to achieve improvements in stability and reduce overfitting. Unfortunately, the hyperparameter values obtained by the tuner did not lead to better performance, as we can see from the following plots.

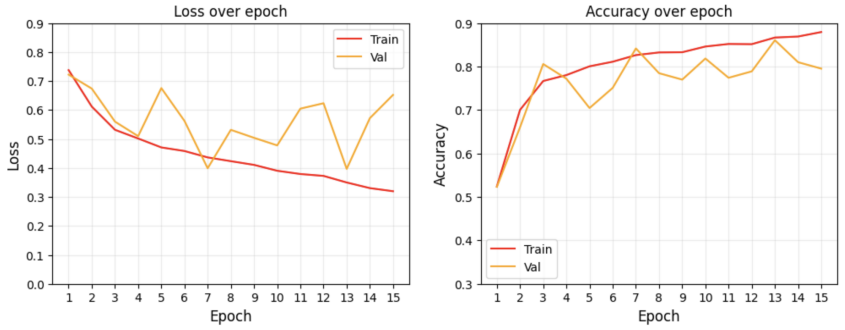

Figure 14: Loss and Accuracy plots

4 Final model

After evaluating the performance of all the models, I decided to select and evaluate Model 6 on the test set. This model obtained after the hyperparameter tuning process and presented in Figure 9, is made up of:

- 3 Conv2D layers with 32, 32, 64 kernel filters of size 3x3. Each layer uses ReLu activation and Padding is used.
- 3 MaxPooling2D 2x2 layers, one after each Conv2D layer.
- A flatten layer.
- A Dense layer with 64 units and ReLu activation function followed by a Dropout layer with dropout rate equal to 0.5. A final Dense layer with 1 unit and Sigmoid activation function.
- L2 regularization with lambda equal to 0.001.
- Adam optimizer as the default value equal to 0.001.
- Batch size of 32 and 60 epochs of training.

4.1 5-fold Cross Validation

K-fold cross validation involves randomly dividing the training set into k groups, or folds, of approximately equal size. It provides a more reliable estimate of model performance compared to a single train-test split. It uses multiple train-test splits of the data to obtain multiple estimates of model performance, which can be averaged to give a more accurate assessment.
For the project, I used a 5-fold cross validation and evaluated the risk estimates using the zero-one Loss. The results are reported as follows:

Table 1: 5-Fold Cross Validation results

K-Fold	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	AVERAGE
Zero-One Loss	0.152	0.140	0.149	0.133	0.150	0.145

The average value for the zero-one loss is 0.145.

4.2 Confusion matrix

I used the confusion matrix to show the number of correctly classified images and also those wrongly classified.

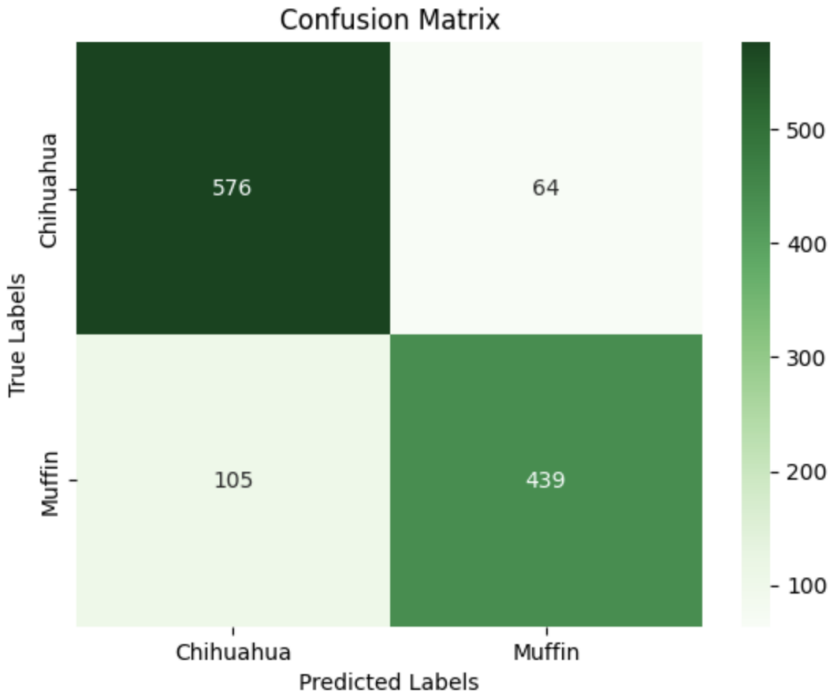

Figure 15: Confusion matrix of the final model

The model shows a higher misclassification of the images of muffins. 11.11% of chihuahuas and 23.91% of the muffins are misclassified.

Below are some examples of misclassified images. As we can see, some mistakes may be caused by poor image quality or unusual positioning. Additionally, it is apparent that there are some relatively straightforward images that were also misclassified.

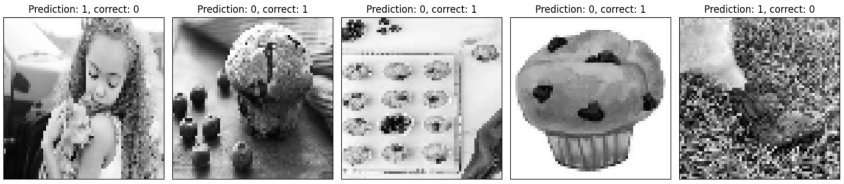

Figure 16: Misclassified images examples

5 Conclusions

In this experimental project, different CNN models were developed for this binary classification task. Initially, we started by training a simple model and evaluated its performance. Based on the evaluation results, we progressively introduced various adjustments, including regularization techniques, obtaining different CNN architectures. Finally, a best-performing model was selected and evaluated, which achieved an accuracy of 85.51% on the test set and had a cross-validation risk estimate of 0.18.

By analyzing the misclassified images, it became evident that with additional time and computational resources, it might be possible to train even better models that perform more consistently across a wider range of images.

6 References

Arman Sarraf, "Binary Image Classification Through an Optimal Topology for Convolutional Neural Networks"

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/us or any other person for assessment on this or any other course of study.