UNIVERSITÀ
DEGLI STUDI
DI MILANO

MUFFINS VS CHIHUAHUAS IMAGE CLASSIFIER

Professor: Nicolò Cesa-Bianchi

Student: Matteo Matone, id: 12628A

email: matteo.matone@studenti.unimi.it

May 2024

Abstract

This project explores the application of neural networks for the binary classification task of distinguishing between images of muffins and Chihuahuas using the Keras framework. Images from the provided dataset were preprocessed by converting them to grayscale and scaling them down for efficient processing. Three principal base-line neural network architectures were designed and evaluated, with hyperparameter configuration tested to identify the most effective model. 5-fold cross-validation approach was employed to estimate the risk, with performance measured using the zero-one loss criterion. The findings emphasize the importance of architectural choices and parameter adjustments in achieving optimal classification accuracy, providing a detailed analysis of the relationship between these factors and the cross-validated risk estimates.

Contents

Chapter 1

Introduction

In this project, we explore the capabilities of Convolutional Neural Networks (CNNs) in performing binary classification of images, specifically distinguishing between chihuahuas and muffins. The classification task presents an interesting challenge due to the visual similarities between these two categories in certain contexts, making it an excellent test case for the efficacy of CNNs in image recognition tasks. To implement this image classification task, we utilize **Keras**, a high-level neural networks API, which facilitates building and training deep learning models. By leveraging Keras, we aim to design, implement, and optimize a CNN model that can classify images as either a chihuahua or a muffin. Through experimentation with different *network architectures* and *hyperparameters*, and utilizing *k-fold cross-validation*, we strive to identify the most effective model configuration. This project not only demonstrates the practical application of CNNs in a real-world scenario but also provides insights into the importance of data preprocessing, model selection, and hyperparameter tuning in achieving high classification accuracy.

1.1 Neural Networks

Neural Networks (NNs) are a class of machine learning algorithms inspired by the structure and function of the human brain. They consist of interconnected layers of nodes (neurons), where each connection has an associated weight. NNs are capable of learning complex patterns and representations from data through a training process, which involves adjusting these weights based on the input data and the corresponding output labels.

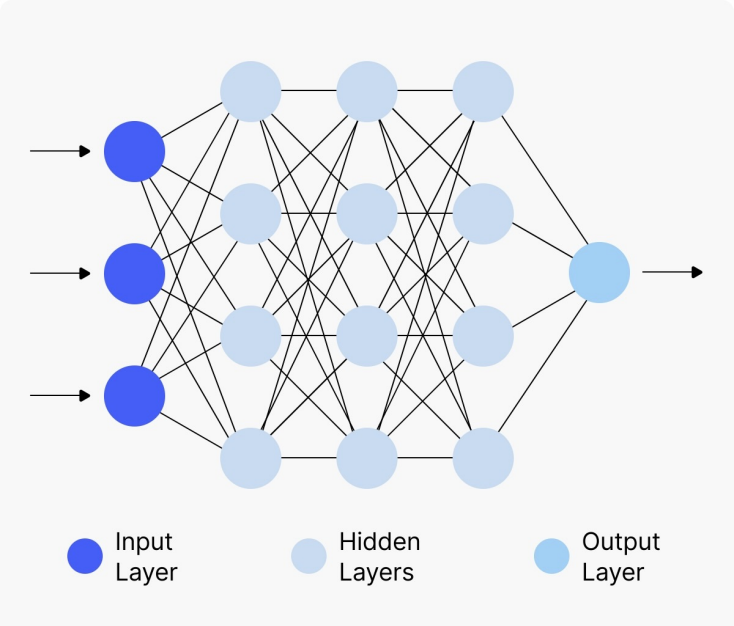

Figure 1.1: Basic Neural Network

A basic neural network (Figure 1.1) is composed of some *input layers*, *hidden layers*, and an *output layer*. Each **neuron** in a layer receives input from the previous layer, applies a linear transformation followed by a non-linear activation function, and passes the result to the next layer. This structure enables neural networks to model intricate relationships in the data, making them suitable for a wide range of tasks, including classification.

1.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a specialized type of neural network designed for processing structured **grid data**, such as images. Unlike traditional neural networks, CNNs are particularly effective in capturing spatial hierarchies in the data through the use of convolutional layers. A typical CNN architecture consists of several key components:

- **Convolutional Layers**: These layers apply convolutional filters to the input image, sliding them across the image to detect local patterns such as edges, textures, and shapes. Each filter produces a feature map that highlights specific aspects of the input.
- **Pooling Layers**: Pooling layers reduce the dimensionality of the feature maps while retaining the most important information. This is typically done using max pooling, which selects the maximum value from a patch of the feature map, thereby reducing its size and computational complexity.
- **Fully Connected Layers**: After several convolutional and pooling layers, the feature maps are flattened and passed through fully connected layers. These layers perform high-level reasoning and classification based on the features extracted by the convolutional layers.
- **Activation Functions**: Non-linear activation functions, such as **ReLU (Rectified Linear Unit)**, are applied after each convolutional and fully connected layer to introduce non-linearity into the model, enabling it to learn complex patterns.
- **Output Layer**: The final layer of a CNN is a sigmoid layer in the case of classification tasks, which outputs a probability distribution over the possible classes.

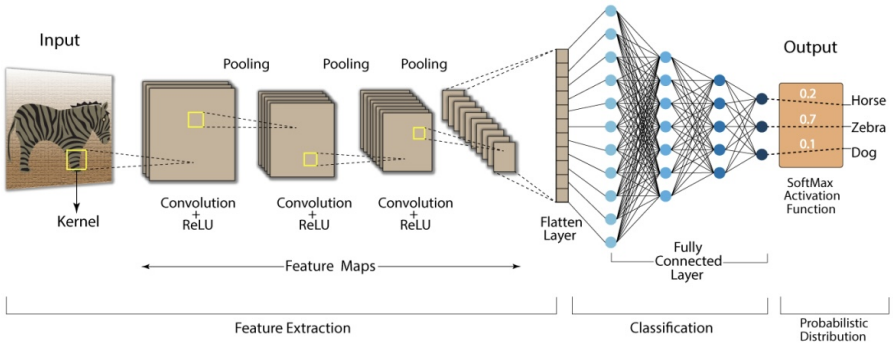

Figure 1.2: Convolutional Neural Network architecture (example)

CNNs have revolutionized the field of computer vision by achieving state-of-the-art performance on a variety of image recognition tasks. Their ability to automatically learn and extract relevant features from raw image data makes them highly effective for tasks such as object detection, face recognition, and image In this project, we leverage the power of CNNs to tackle the binary classification challenge of distinguishing between chihuahuas and muffins. By experimenting with different CNN architectures and optimizing hyperparameters, we aim to develop a robust model capable of achieving high classification accuracy using Keras.

Chapter 2

Dataset and Data Preprocessing

The dataset, consisting of 5,917 images, was divided into training (4733 images), validation (591 images), and test (593 images) sets in a 80:10:10 ratio. This split ensured a robust evaluation of the model's performance.
The dataset was preprocessed through several key steps to ensure consistency and readiness for training the neural network.
All images were checked for corruption, ensuring that the entire dataset was usable without any missing or damaged files and fortunately none of those have been found, meaning that were possible to work with the entire dataset.

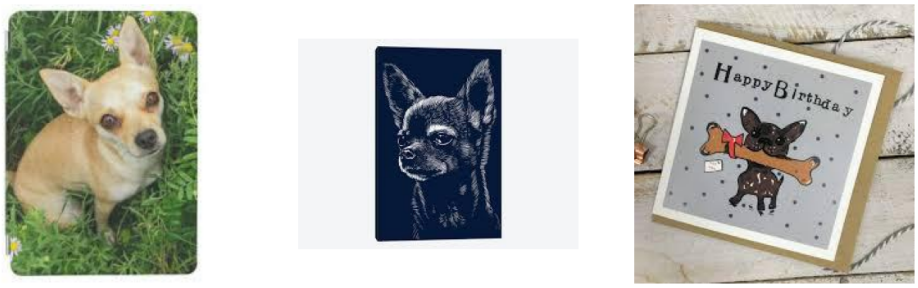

Figure 2.1: Sample of Chihuahua Images

Figure 2.2: Sample of Muffin images

As we can see in Figure 2.1 and Figure 2.2, given that the images of the dataset have different sizes, all of them were resized to a fixed dimension of **150x150 pixels**. This standardization is crucial for feeding the images into the convolutional neural network (CNN), plus the images were converted to **grayscale** to simplify the model and reduce computational load, while retaining essential features for classification. This is a very important process given the significant variability in both chihuahua (different poses, colors, background and sometimes also different dog race) and muffin (different texture, toppings and lighting conditions) images. Finally the images were stored in **NumPy arrays** and pixel values were normalized to the range **[0, 1]** by dividing by 255. Those last two steps make more efficient the access and manipulation and help in stabilizing and speeding up the training process.

Figure 2.3: Overall caption for the three figures

The images in Figure 2.3 are samples of the new images with the corresponding labels, with fixed size and in grayscale.

Chapter 3

Convolutional Naural Network Implementation

3.1 Introduction

In this chapter, we detail the implementation of Convolutional Neural Networks (CNNs) for the binary classification task. Leveraging the Keras framework, we designed and experimented with different models, each based on one base-line architecture. The models were modified and experimented with to explore various architectural configurations and hyperparameters, aiming to achieve optimal performance in classifying the images. The **Sequential()** model has been employed to create a linear stack of layers.

3.2 Model 1

3.2.1 Architecture 1

The first base-line architecture has a simple structure to provide an initial understanding of its behavior.
The layers implemented are:

1. **Input** layer with shape (150, 150, 1)
2. **AveragePooling2D** layer with pooling size 2x2
3. **Flatten** layer
4. **Dense** layer with 64 neurons and "ReLU" activation function
5. **Dense** layer with 1 neuron and "Sigmoid" activation function

The architecture begins with an input layer of shape (150, 150, 1), indicating grayscale images of size 150x150 pixels. Following the input layer, an

AveragePooling2D layer is utilized for downsampling, aiming to reduce the spatial dimensions of the input. The subsequent **Flatten** layer converts the two-dimensional feature maps into a one-dimensional array, preparing them for processing by the dense layers. Two dense layers are then added: the first with *64* neurons and a **ReLU activation function**, and the second with a single neuron and a **sigmoid activation function**, serving as the output layer for binary classification. The model is compiled with binary cross-entropy loss and the Adam optimizer.
The model is trained for *20* **epochs**, indicating the number of times the entire training dataset is passed forward and backward through the neural network. A **batch size** of *32* is used, meaning that the model is updated after processing 32 samples. Then validation data (X_{val} and y_{val}) is used to evaluate the model's performance on unseen data during training.

Results

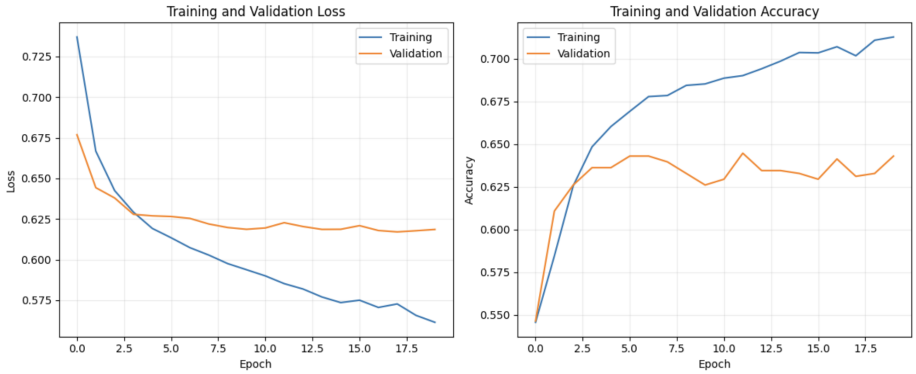

Figure 3.1: Base Architecture - Loss and Accuracy

Table 3.1: Base Architecture results

	Accuracy	**Loss**
Training	0.71	0.55
Validation	0.64	0.61
Test	0.65	0.60

Training accuracy steadily increased over epochs, reaching approximately *71.88%*, while training loss steadily decreased. This suggests the model effectively learned to minimize errors between predicted and actual labels during training.
In terms of generalization ability, validation accuracy stabilized around *64.30%*, indicating consistent performance on unseen data. Validation loss also stabilized, suggesting reliable predictions on the validation set.

Evaluation on the test set yielded an accuracy of approximately *65.09%* and a test loss of approximately *60.28%*. These results indicate that, even though the loss values are still high, the structure of this architecture is promising, there is room for improvement to achieve higher accuracy. Subsequent modifications and experimentation with different architectural configurations aim to enhance the model's performance further.

3.2.2 Architecture 2

The second architecture introduces several modification to the previous one. The key changes include the addition of a convolutional layer, a max-pooling layer, and a dropout layer.
The layers implemented are:

1. **Input** layer with shape (150, 150, 1)
2. **Conv2D** layer with 64 filters and 3x3 kernel and "ReLU" activation function
3. **MaxPooling2D** layer with pooling size 3x3
4. **Flatten** layer
5. **Dense** layer with 64 neurons and "ReLU" activation function
6. **Dropout** layer with dropout rate at 50%
7. **Dense** layer with 1 neuron and "Sigmoid" activation function

The architecture adds to the base one a **Conv2D** layer with *64* filters and a *3x3 kernel*, followed by a ReLU activation function. This convolutional layer helps the model capture local patterns in the images, which is crucial for image classification tasks; a **MaxPooling2D** layer with a *3x3 pool size* is added to downsample the feature maps, reducing their spatial dimensions and computational complexity.
The architecture then before the final output layer, a **Dropout** layer is added with a *0.5* dropout rate is introduced to reduce overfitting by randomly setting *50%* of the input units to zero during each training update. Finally, the Dense layer with *64* neurons and a ReLU activation function, and the output **Dense** layer with a sigmoid activation function for binary classification, remain the same as in the base model.

Results

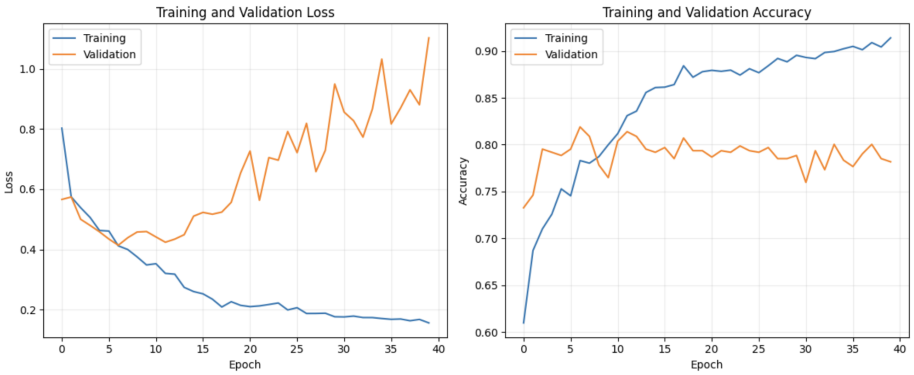

Figure 3.2: Architecture 1.2 - Loss and Accuracy

Table 3.2: Architecture 1.2 results

	Accuracy	**Loss**
Training	0.91	0.15
Validation	0.78	1.10
Test	0.81	0.92

This second architecture demonstrates significant improvements in terms of training and validation performance. During training the model reaches an accuracy of approximately *91.51%* and the loss decreases consistently down to *15.33%*, reflecting the model's growing proficiency in minimizing errors between the predicted and actual labels.
In terms of validation performance this model consistently outperformed the base one. The validation accuracy peaked at around *81.90%*, significantly higher than the base model's *64.30%*. This suggests that the additional layers and dropout mechanism helped the model generalize better to unseen data. However, the validation loss showed some fluctuation, with the final epoch reaching a loss of *1.10* meaning that strong overfitting occurs.
Finally, the test accuracy increases up to *81.27%* (against the *65.09%* of the previous architecture) but also the loss increases up to *92.63%*: this higher loss, in conjunction with higher accuracy, suggests that the model is learning more complex patterns and features from the data, which may lead to occasional misclassifications but overall better generalization.

3.3 Model 2

3.3.1 Architecture 1

In the second model, the architecture introduces additional convolutional and pooling layers compared to the previous models. This increased complexity aims to improve feature extraction and classification performance.
The layers implemented are:

1. **Input** layer with shape (150, 150, 1)
2. **Conv2D** layer with 64 filters and 3x3 kernel and "ReLU" activation function
3. **MaxPooling2D** layer with pooling size 3x3
4. **Conv2D** layer with 32 filters and 3x3 kernel and "ReLU" activation function
5. **MaxPooling2D** layer with pooling size 3x3
6. **Dropout** layer with dropout rate at 50%
7. **Flatten** layer
8. **Dense** layer with 64 neurons and "ReLU" activation function
9. **Dense** layer with 1 neuron and "Sigmoid" activation function

This architecture incorporates additional convolutional and pooling layers, effectively increasing the network's depth and capacity for feature extraction. This **Dropout** layer has been relocated from the fully connected layers to after the second MaxPooling layer with the same dropout rate. This adjustment aim to introduce regularization earlier in the network, potentially enhancing its ability to prevent overfitting.

Results

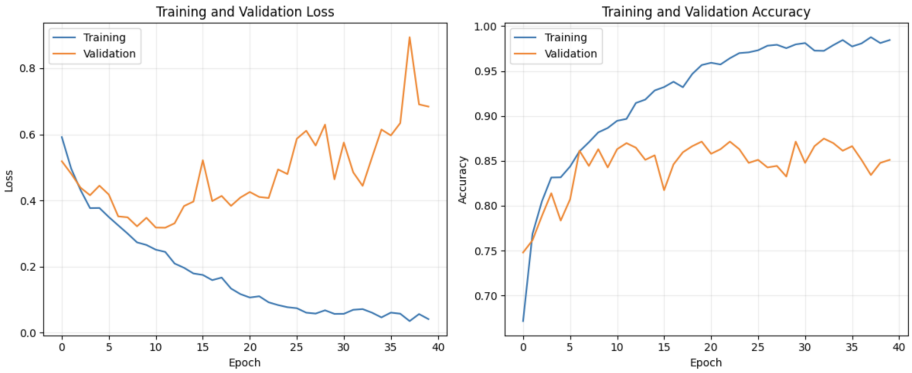

Figure 3.3: Architecture 2_1 - Loss and Accuracy

Table 3.3: Architecture 2_1 results

	Accuracy	**Loss**
Training	0.98	0.045
Validation	0.85	0.68
Test	0.82	0.71

This architecture showed improvements in accuracy on the validation set compared to the first architecture. The accuracy increased steadily over the epochs, reaching a peak of around *98.76%*. However, the loss on the validation set started to increase slightly after around 25 epochs so, despite the fact that the training loss reduces significantly to *4.59%* by the *40th epoch*, the validation loss starts increasing after a certain point, indicating overfitting; the loss on test set is lower here (*71.62%*) compared to the previous architecture (92.63%).

3.3.2 Architecture 2

The second architecture expands upon the previous model by introducing an additional convolutional and max-pooling layer. This architecture aims to improve feature extraction capabilities and, potentially, overall model performance by incorporating more layers to capture intricate patterns in the data.
The layers implemented are:

1. **Input** layer with shape (150, 150, 1)
2. **Conv2D** layer with 64 filters and 3x3 kernel and "ReLU" activation function
3. **MaxPooling2D** layer with pooling size 3x3

4. **Conv2D** layer with 32 filters and 3x3 kernel and "ReLU" activation function
5. **MaxPooling2D** layer with pooling size 3x3
6. **Conv2D** layer with 32 filters and 3x3 kernel and "ReLU" activation function
7. **MaxPooling2D** layer with pooling size 3x3
8. **Dropout** layer with dropout rate at 50%
9. **Flatten** layer
10. **Dense** layer with 64 neurons and "ReLU" activation function
11. **Dense** layer with 1 neuron and "Sigmoid" activation function

Results

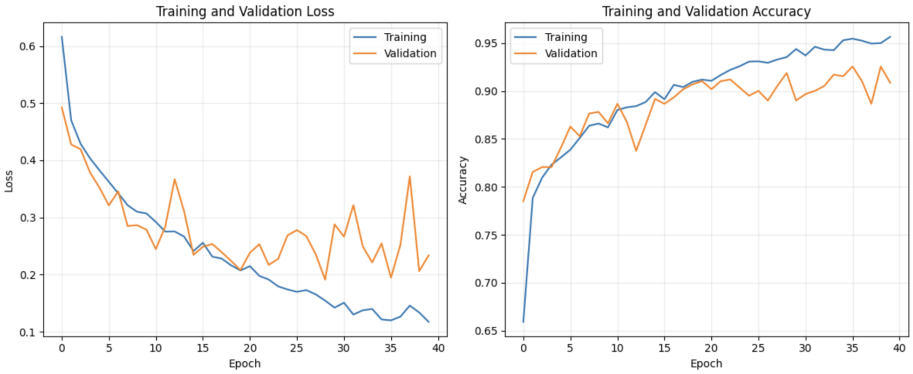

Figure 3.4: Architecture 2_2 - Loss and Accuracy

Table 3.4: Architecture 2_2 results

	Accuracy	**Loss**
Training	0.95	0.12
Validation	0.90	0.23
Test	0.92	0.21

The model shows consistent improvement in both training and validation accuracy across epochs, suggesting effective learning. Early epochs witness significant gains in accuracy, which then plateau, indicating the model is nearing its optimal performance.
The validation accuracy peaks around *92.55%* but slightly fluctuates towards

the end, indicating some degree of overfitting. However, the drop is not substantial, suggesting the model generalizes relatively well. The test accuracy of *92.92%* is in close alignment with the validation accuracy, reinforcing the model's capability to generalize well on unseen data.
Wee can see that adding an extra convolutional layer allows the model to learn more complex patterns. This results in improved accuracy, reduced loss and reduced overfitting, even though the validation loss and accuracy are a bit volatile. Despite the slight indication of overfitting, the model's performance on the test set confirms its robustness and applicability for the task at hand. Further fine-tuning, such as adjusting dropout rates or introducing learning rate schedules, could potentially enhance performance even further.

3.3.3 Architecture 3

In this new architecture there are some notable changes compared to the previous one.
The layers implemented are:

1. **Input** layer with shape (150, 150, 1)
2. **Conv2D** layer with 64 filters and 2x2 kernel and "ReLU" activation function
3. **AveragePooling2D** layer with pooling size 2x2
4. **Dropout** layer with dropout rate at 30%
5. **Conv2D** layer with 64 filters and 2x2 kernel and "ReLU" activation function
6. **MaxPooling2D** layer with pooling size 2x2
7. **Dropout** layer with dropout rate at 30%
8. **Conv2D** layer with 256 filters and 3x3 kernel and "ReLU" activation function
9. **AveragePooling2D** layer with pooling size 3x3
10. **Dropout** layer with dropout rate at 50%
11. **Flatten** layer
12. **Dense** layer with 64 neurons and "ReLU" activation function
13. **Dropout** layer with dropout rate at 30%
14. **Dense** layer with 1 neuron and "Sigmoid" activation function

The utilization of **AveragePooling** layers instead of MaxPooling might capture a broader sense of spatial information across the feature maps, potentially leading to a more generalized representation of the data. The dropout rates are tailored differently in this architecture: after the first AveragePooling layer with a dropout rate of *30%*, after the MaxPooling layer with a dropout rate of *30%* again and after the second AveragePooling layer with a dropout rate of *50%*. This structure possibly reflect a different approach to regularization.
By increasing the number of filters in the third convolutional layer (from 32 to *256*), also modifying the filters size of *2x2* and *3x3* and adjusting dropout rates, this architecture introduces increased model complexity, potentially allowing it to learn more intricate patterns.
The number of epochs is increased up to *60*, but **EarlyStopping** with **patient** of *8* is introduce allowing the training process to halt when there is no improvement in the **val_loss metric** for 8 consecutive epochs. This prevents overfitting by stopping the training process before the model starts to memorize the training data too closely.

Results

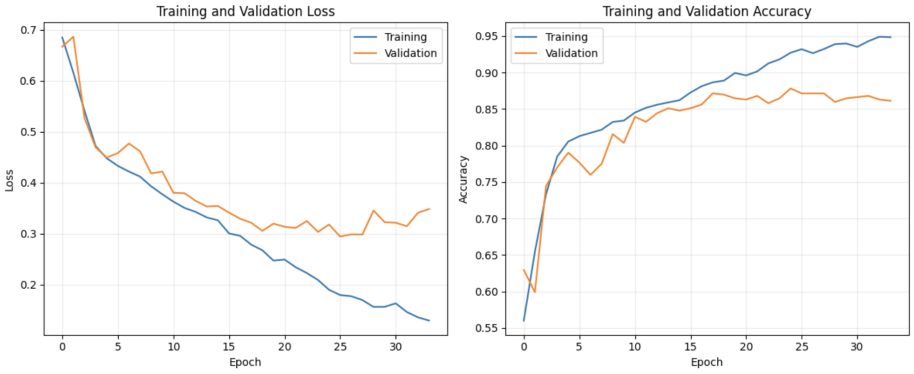

Figure 3.5: Architecture 2_3 - Loss and Accuracy

Table 3.5: Architecture 2_3 results

	Accuracy	**Loss**
Training	0.94	0.14
Validation	0.86	0.34
Test	0.91	0.25

During training, the model's accuracy increased steadily, reaching approximately *94.4%* on the training set. The validation accuracy fluctuated but generally increased, reaching around *86.1%*. The training loss decreased steadily,

indicating effective learning. However, the validation loss fluctuated, ultimately settling around *34.8%*. Early stopping was triggered after *34* epochs due to no improvement in validation loss for 8 consecutive epochs. When evaluated on the test set, the model achieved an accuracy of approximately *91.2%* and a loss of around *25.30%*.
Overall, the model performed reasonably well, demonstrating good accuracy on both training and validation sets, as well as generalization to unseen data, as evidenced by its performance on the test set. The validation accuracy and loss are somewhat volatile but not excessively so. The fluctuations observed during training are relatively normal and within an acceptable range, indicating that the model is learning effectively and is not overfitting or underfitting significantly. Despite the variations, the overall trend shows improvement, and the final validation accuracy and loss suggest that the model's performance on unseen data is reliable. However, there may still be room for further optimization.

3.4 Model 3

3.4.1 Architecture 1

The model introduces a more complex architecture compared to the previous architecture. The primary differences lie in the presence of one more **convolutional layer**, **increased number of filters**, **deeper layers**, and **higher dropout rates**. These changes aim to capture more intricate features and potentially improve the model's robustness against overfitting.
The layers implemented are:

1. **Input** layer with shape (150, 150, 1)
2. **Conv2D** layer with 128 filters and 2x2 kernel and "ReLU" activation function
3. **MaxPooling2D** layer with pooling size 2x2
4. **Dropout** layer with dropout rate at 50%
5. **Conv2D** layer with 128 filters and 2x2 kernel and "ReLU" activation function
6. **MaxPooling2D** layer with pooling size 2x2
7. **Dropout** layer with dropout rate at 50%
8. **Conv2D** layer with 256 filters and 2x2 kernel and "ReLU" activation function
9. **MaxPooling2D** layer with pooling size 2x2
10. **Dropout** layer with dropout rate at 50%

11. **Conv2D** layer with 512 filters and 2x2 kernel and "ReLU" activation function
12. **MaxPooling2D** layer with pooling size 2x2
13. **Dropout** layer with dropout rate at 70%
14. **Flatten** layer
15. **Dense** layer with 256 neurons and "ReLU" activation function
16. **Dropout** layer with dropout rate at 60%
17. **Dense** layer with 1 neuron and "Sigmoid" activation function

Starting with *128* filters in the first two convolutional layers and escalating to *256* and *512* filters in the subsequent layers, this architecture can capture more detailed and complex features from the input images.
The dropout rates are progressively increased from 0.5 to *0.7* across the layers. The fully connected dense layer now has *256* units, providing a rich feature representation before the final classification.

Results

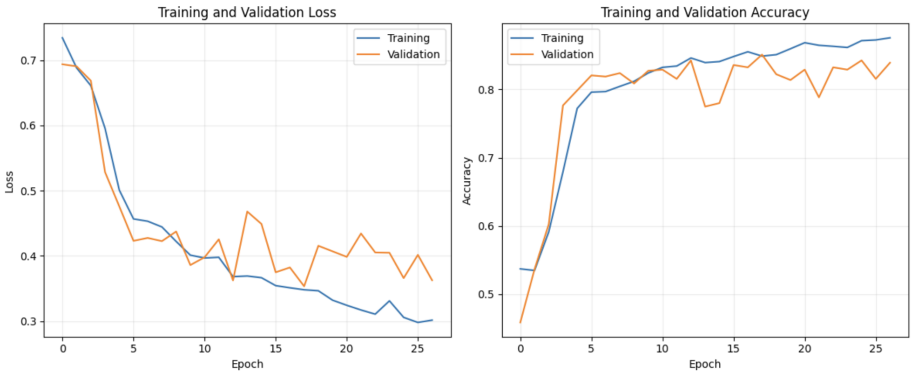

Figure 3.6: Architecture 3_1 - Loss and Accuracy

Table 3.6: Architecture 3_1 results

	Accuracy	**Loss**
Training	0.86	0.31
Validation	0.83	0.36
Test	0.86	0.31

The training of model was carried out over *60* epochs.
At the beginning of the training, the model exhibited relatively poor performance, with an accuracy of around *54%* and a high loss of *88%* on the training set. Validation accuracy was even lower, around *46%*, with a validation loss of *69%*. Over subsequent epochs, there was a notable improvement in both training and validation metrics. The accuracy steadily increased, reaching around *86%*, while the loss decreased to approximately *32%*. Similarly, validation accuracy improved to approximately *85%*, with validation loss dropping to about *35%*.
The training was **halted at epoch 27** due to no significant improvement in validation loss for the past 9 epochs. The model's performance seemed to plateau, indicating that further training might not yield substantial benefits.
Finally, when evaluated on the unseen test set, the model achieved an accuracy of approximately *86%* and a loss of *32%*. This suggests that the model generalized reasonably well to unseen data, considering its performance on the test set was consistent with the validation set.

3.4.2 Architecture 2 - Data Augmentation

This architecture incorporates data augmentation techniques using TensorFlow's **ImageDataGenerator** along with a custom-defined data augmentation model (**data_augmentation3_2**). The data augmentation techniques applied include random horizontal flipping, random rotation, and random zooming. Data augmentation is a common technique used to artificially expand the size of the training dataset, which helps improve the model's generalization and robustness.
The layers implemented are:

1. **Input** layer with shape (150, 150, 1)
2. **data_augmentation3_2**
3. **Conv2D** layer with 32 filters and 3x3 kernel and "ReLU" activation function
4. **MaxPooling2D** layer with pooling size 2x2
5. **Dropout** layer with dropout rate at 20%
6. **Conv2D** layer with 64 filters and 3x3 kernel and "ReLU" activation function
7. **MaxPooling2D** layer with pooling size 2x2
8. **Dropout** layer with dropout rate at 30%
9. **Conv2D** layer with 128 filters and 3x3 kernel and "ReLU" activation function
10. **MaxPooling2D** layer with pooling size 2x2

11. **Dropout** layer with dropout rate at 40%
12. **Conv2D** layer with 128 filters and 3x3 kernel and "ReLU" activation function
13. **MaxPooling2D** layer with pooling size 2x2
14. **Dropout** layer with dropout rate at 40%
15. **Flatten** layer
16. **Dense** layer with 256 neurons, "ReLU" activation function and L2 kernel regularization (0.01)
17. **Dropout** layer with dropout rate at 50%
18. **Dense** layer with 1 neuron and "Sigmoid" activation function

The inclusion of data augmentation techniques enhances the model's ability to learn diverse patterns from the training dataset. Random transformations like flipping, rotation, and zooming help the model generalize better to unseen data by exposing it to variations of the original images.
L2 regularization is applied to the first dense layer to further control overfitting. An exponential decay learning rate schedule is implemented using **ExponentialDecay**. It gradually reduces the learning rate during training, which can help the model converge faster and potentially find a better minimum. **EarlyStopping** is used to monitor the validation loss and stop training when it stops decreasing, thus preventing overfitting. **ReduceLROnPlateau** reduces the learning rate when the validation loss plateaus, and **ModelCheckpoint** saves the best model during training.

Results

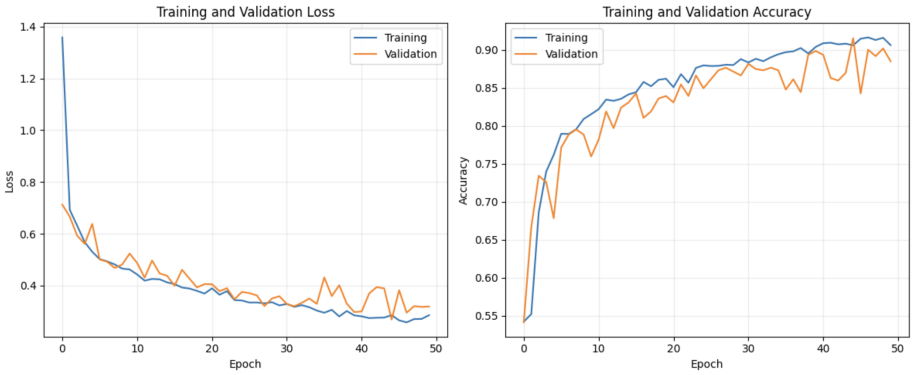

Figure 3.7: Architecture 3_2 - Loss and Accuracy

Table 3.7: Architecture 3_2 results

	Accuracy	**Loss**
Training	0.90	0.29
Validation	0.88	0.31
Test	0.89	0.29

This architecture shows promising outcomes:
The training accuracy and loss steadily improve over epochs, indicating effective learning. Validation metrics closely track training, suggesting good generalization. Validation accuracy peaks at approximately *88.49%*, with a loss stabilizing around *31%*. Test performance is robust, with an accuracy of about *89.21%* and loss of *29%*, confirming generalization capability. Compared to previous architectures, this one exhibits notable enhancements in validation and test accuracies, signifying the efficacy of augmentation and regularization techniques. Data augmentation diversifies training examples, while regularization prevents overfitting. Overall, this model represents a significant advancement.

Chapter 4

Hyperparameter Tuning

Hyperparameter tuning is a pivotal aspect of machine learning model development, wherein the optimal configuration of parameters that significantly influence a model's performance is determined. Effectively tuning hyperparameters is crucial for enhancing a model's predictive capability, improving generalization to unseen data, and ultimately maximizing performance.

Keras Tuner is a library that streamlines the hyperparameter tuning process by automating the search for the optimal configuration. Keras Tuner provides several algorithms for hyperparameter optimization, including Random Search, Bayesian Optimization, and Hyperband.

Among these algorithms, ***Hyperband*** stands out for its efficiency in exploring the hyperparameter space while maximizing computational resources. It is an adaptive resource allocation algorithm that balances the exploration of different hyperparameter configurations with the depth of their evaluation. Hyperband works by iteratively running multiple configurations in parallel, gradually allocating more resources to promising configurations while pruning less promising ones.

Results

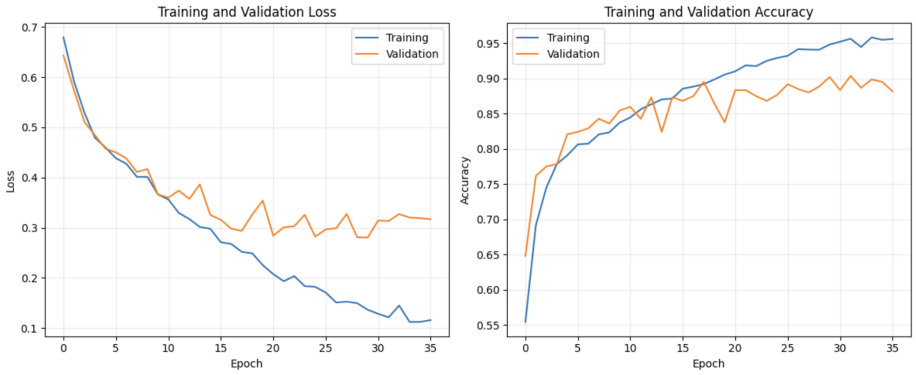

Figure 4.1: Hypermodel - Loss and Accuracy

Table 4.1: Hypermodel results

	Accuracy	**Loss**
Training	0.95	0.10
Validation	0.88	0.31
Test	0.83	0.40

The results show a bit presence of **overfitting**. The Hyperparamenter Tuning doesn't represent the best choice for this kind of task. Despite the good results reached during training, accuracy around *95%* and loss around *10%*, the results on the test set, where the model reached an accuracy of around *83%* and a loss of around *40%*, are not promising or, at least, not as much as the last model.
The last model, implemented with Data Augmentation, reached an accuracy of *89%* and a loss of *29%*, plus there is almost no presence of overfitting and the model evaluation results are more stable and less volatile.
So, the **final** model is the 3_4 one.

Chapter 5

5-fold Cross-Validation with zero-one loss

K-fold cross validation is a robust technique for evaluating the performance of machine learning models. It works by splitting the dataset into k equal folds. For each fold, the model is trained on the remaining k-1 folds and evaluated on the held-out fold. This process is repeated k times, ensuring that each data point is used for validation exactly once. In this project, we utilized **5-fold cross validation (k=5)** to obtain a more comprehensive picture of the model's generalization ability.
For evaluating classification performance, we employed the zero-one loss. This loss function assigns a value of 1 to any incorrect classification (either Muffin classified as Chihuahua or vice versa) and 0 to correct classifications. By averaging the zero-one loss across all folds in the k-fold process, we obtain a reliable estimate of the model's true classification error on unseen data.
Using 5-fold cross validation with zero-one loss allows us to assess the model's performance beyond the training data and provides a more robust measure of its effectiveness in classifying new muffin and Chihuahua images.

Results

Table 5.1: 5-fold Cross Validation results

K-fold	**0-1 Loss**
Fold 1	0.11
Fold 2	0.07
Fold 3	0.069
Fold 4	0.047
Fold 5	0.06
Average	0.07

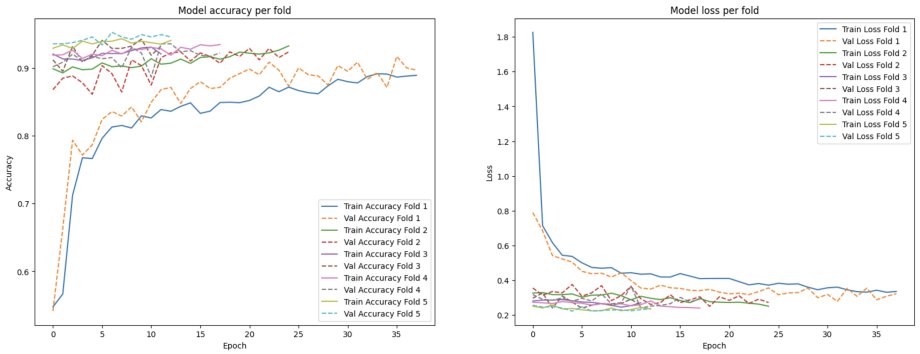

Figure 5.1: 5-fold - Model 3_2

Across all five folds, the model showed consistent improvement in accuracy and reduction in loss over successive epochs. This indicates that the training process effectively enhances the model's performance. The zero-one loss values were relatively low in all folds, with an average around 7%. This low error rate highlights the model's precision in making accurate predictions.
The consistent performance across different folds suggests that the final model (model3_2) is robust and reliable. The model performs well regardless of the specific partition of the data, indicating stability in its learning process and outcome.

Chapter 6

Conclusion

Throughout this project, various CNN models were implemented with the goal of creating a classifier capable of distinguishing between images of chihuahuas and muffins. Following an overview of the dataset and preprocessing steps, three models were constructed based on different baseline architectures. Despite initial attempts at hyperparameter tuning, which failed to meet expectations, it became evident that automated method alone would not suffice. Ultimately went through the development of a final model that demonstrated superior performance, achieving an average 0-1 loss of 0.07 through a rigorous 5-fold cross-validation process.

Bibliography

[1] Solving real world data science problems with Python! (computer vision edition), https://www.youtube.com/watch?v=l-NAT4H4384&t=2488s

[2] Keras with TensorFlow Course, https://www.youtube.com/watch?v=qFJeN9V1ZsI

[3] TensorFlow Documentation, https://www.tensorflow.org/decision_forests/tutorials/automatic_tuning_colab#:~:text=Decision%20Forests%20library.-,Hyper%2Dparameter%20tuning%20algorithms,a%20new%20model%20each%20time.

[4] Keras Documentation, https://keras.io/api/layers/preprocessing_layers/image_augmentation/