Università degli studi di Milano

Data Science for Economics

MACHINE LEARNING PROJECT

Convolutional Neural Networks: "Chihuahua vs Muffin"

Student: Ketrin Hristova

Registration number: 13209A

ketrin.hristova@studenti.unimi.it

Contents

1. **Abstract**.....3

2. **The dataset and data pre-processing**.....4

3. **Convolutional Neural Networks**.....5

   3.1 *Introduction*.....5

   3.2 *1st model: Base Model*……..…….....7

   3.3 *1st Model with Dropout*.....8

   3.4 *2nd Model: 3 Convolutional layers, 2 Dense layers*...……...9

   3.5 *3th Model: 3 Convolutional layers, 1 Dense layer*……...11

   3.6 *4th Model: 4 Convolutional layers, 1 Dense layer*……...13

   3.7 *5th Model: 4 Convolutional layers, 2 Dense layers*……...15

4. **Hyperparameter tuning**..... 16

5. **Final Model**…….....17

   5.1 *Confusion Matrix*…….....19

   5.2 *5-fold Cross-Validation*…….....20

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.

1. Abstract

This study focuses on developing an effective machine-learning model for the binary classification task involving images of Chihuahuas and Muffins. The approach centers on implementing Convolutional Neural Networks using Keras API within Tensorflow.

Following data preprocessing, which involves converting images to grayscale pixel values and scaling them down, I address the following tasks:

- Experiment with different network architectures and training hyperparameters
- Use 5-fold cross-validation to compute risk estimates
- Discuss the obtained results, with a specific focus on the influence of the choice of the network architecture and the tuning of the hyperparameters on the final cross-validated risk estimates; employing the zero-one loss metric to calculate the cross-validated estimates.

2. The Dataset and data pre-processing

The dataset consists of images featuring Chihuahuas and Muffins, varying in size, and segmented into two distinct sets: training and test.
There are a total of 5,917 images: the training set comprises 4,733 images, while the test set contains 1,184 images.

Data preprocessing is a crucial phase in enhancing neural network performance by accentuating essential features and introducing diversity to the training data, thereby aiding the model's generalization and reducing overfitting.
For neural networks to process images, they must be converted from JPEG format into a numeric representation compatible with the TensorFlow framework, typically in the form of tensors. To handle the dataset, we utilize two custom functions enabling the separate loading of the training and test set.
Given our dataset's varying image dimensions, we employ downsizing to ensure an input shape of (150, 150, 1), maintaining labels and converting images to grayscale.
Following this, we shuffle the training set to enhance the model's generalization capability.
We then generate the feature matrix, which is subsequently converted into NumPy array and scaled by dividing each pixel value by 255, to fit the range of values between 0 and 1.

3. Convolutional Neural Networks

3.1 Introduction

A neural network is a type of machine learning model inspired by the structure and function of a human brain. It comprises interconnected nodes, or neurons, arranged in intricate networks to solve complex problems. They find applications in various domains such as image recognition, predictive modeling, and natural language processing due to their ability to recognize patterns. Neural networks organize processing units into layers, commonly structured into three parts: an input layer representing input fields, one or more hidden layers, and an output layer representing target fields.

Convolutional Neural Networks are specialized neural networks renowned for their ability to classify spatial data such as images or videos. Additionally, CNNs can be adjusted to reduce overfitting and higher accuracy or minimize losses.

A complete Convolutional Neural Network architecture, also known as covnets, comprises a sequence of layers, each transforming one volume into another through a differentiable function. Here are the types of layers commonly found in covnets:

- Input Layers: These are where feed input data to the model, typically images or sequences of images in CNNs.
- Convolutional Layers: These layers extract features from the input dataset by applying learnable filters, known as kernel, to the input images.
- Activation Layers: Activation layers introduce nonlinearity to the network by applying an element-wise activation function to the output of the convolutional layer.

ReLu (Rectified Linear Unit) is an activation function that sets negative values to zero, which helps accelerate training convergence.

$$f(x) = max(0, x)$$

Sigmoid (Output) is used for binary classification tasks. It scales the output to a range between 0 and 1, making it ideal as an activation function for the output layer in our model when we need the output to represent probabilities.

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Pooling Layers: These layers are intermittently inserted in covnets to reduce the size of the volume, speeding up computation saving memory, and preventing overfitting. Common types include max pooling and average pooling.
- Flattening: Feature maps resulting from convolution and pooling layers are flattened into one-dimensional vectors to be passed into fully connected layers for classification or regression.
- Fully Connected Layers: These layers input from the preceding layer and perform the final classification or regression task.
- Output Layer: The output from fully connected layers is fed into a logistic function, such as sigmoid or softmax, for classification tasks. This converts the output of each class into a probability score.

3.2 1st model: Base Model

In this subsection, the analysis focuses on the first model implemented. The Convolutional Neural Network is structured as follows:

- *1st Convolutional Layer*:
    - **Conv2D** layer with 32 filters and a kernel_size of 3 x 3,
- *1st Pooling Layer***:**
    - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**
- *2nd Convolutional Layer*:
    - **Conv2D** layer with 64 filters and a kernel_size of 3 x 3,
- *2nd Pooling Layer***:**
    - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**
- *Flatten Layer*
- *1st Dense layer:*
    - **Dense** layer with 64 nodes,
- *Final Dense Layer with 1 node*

Each layer utilizes the ReLu activation function except for the output layer, which employs a sigmoid activation function. The model was compiled using the Adam optimizer [1]and binary cross-entropy loss[2]. It underwent training for 20 epochs with a batch size of 64.

[1] The Adam optimizer is an optimization algorithm used in CNNs that adjusts learning rates for each parameter and performs bias correction, leading to faster convergence and better performance.

[2] Binary cross-entropy loss is a commonly used loss function in CNNs for binary classification tasks. It measures the difference between predicted probabilities and actual biases, penalizing incorrect predictions.

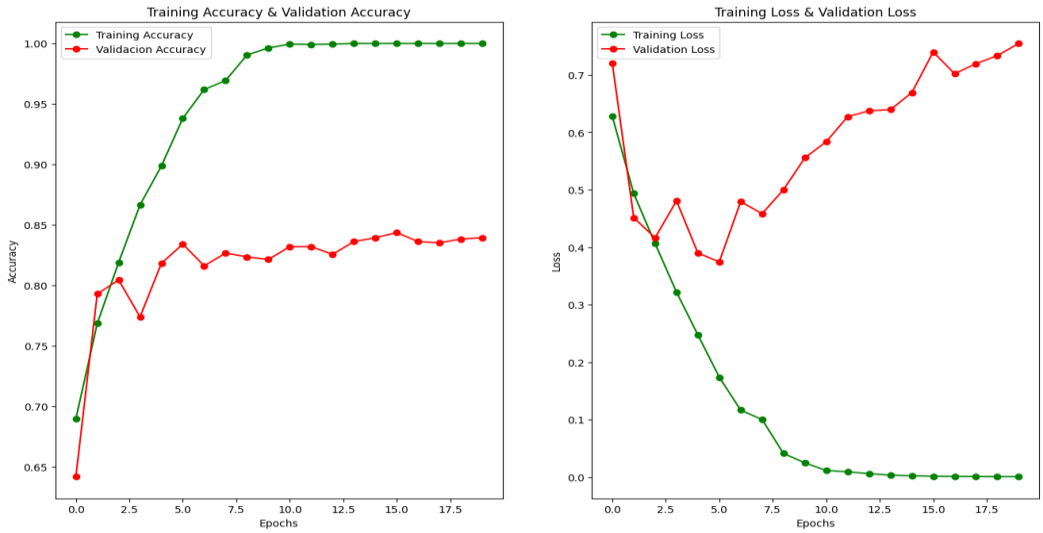

Figure 1

We can notice that the base convolutional model exhibits significant overfitting, with a validation accuracy of almost 84%. While the training loss decreases close to zero, the validation loss consistently rises, likewise, the training accuracy approaches 100% while the validation accuracy ranges around 0.8.

Considering these outcomes, it's imperative to employ techniques such as dropout to mitigate overfitting.

3.3 1st model with Dropout

To reduce potential overfitting, dropout layers can be incorporated into the base convolutional model. These layers randomly deactivate a fraction of neurons during training, thereby discouraging the model from fixating excessively on specific image features. The percentage of neurons to be deactivated is determined during the architecture design phase.

A Dropout layer is added after the Dense layer with setting $p=0.5$.

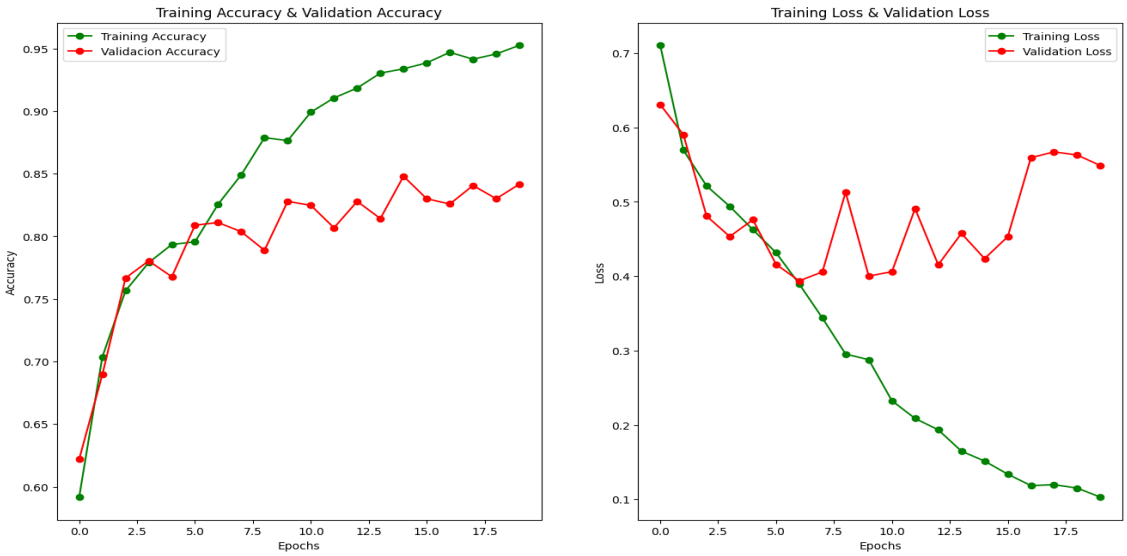

Figure 2

There's a noticeable improvement compared to the previous model in mitigating overfitting, as evidenced by the reduction in validation loss and a more stable validation accuracy. However, the validation accuracy still hovers around 0.84, indicating that further optimization may be needed to achieve higher accuracy.

3.4 2nd model: 3 Convolutional layers, 2 Dense layers

In the upcoming model, weìve introduced additional convolutional and dense layers to potentially enhance the CNN's performance by capturing more intricate features.

Here's the structure of this model:

- *1st Convolutional Layer*:
    - **Conv2D** layer with 32 filters and a kernel_size of 3 x 3,
- *1st Pooling Layer***:**
    - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**
- *2nd Convolutional Layer*:
    - **Conv2D** layer with 64 filters and a kernel_size of 3 x 3,
- *2nd Pooling Layer***:**
    - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**
- *3rd Convolutional Layer*:
    - **Conv2D** layer with 128 filters and a kernel_size of 3 x 3,
- *3rd Pooling Layer***:**
    - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**
- *Flatten Layer*
- *1st Dense layer:*
    - **Dense** layer with 128 nodes,
    - **Dropout** layer with $p = 0.5$,
- *2nd Dense layer:*
    - **Dense** layer with 64 nodes,
    - **Dropout** layer with $p = 0.5$,
- *Final Dense Layer with 1 node*

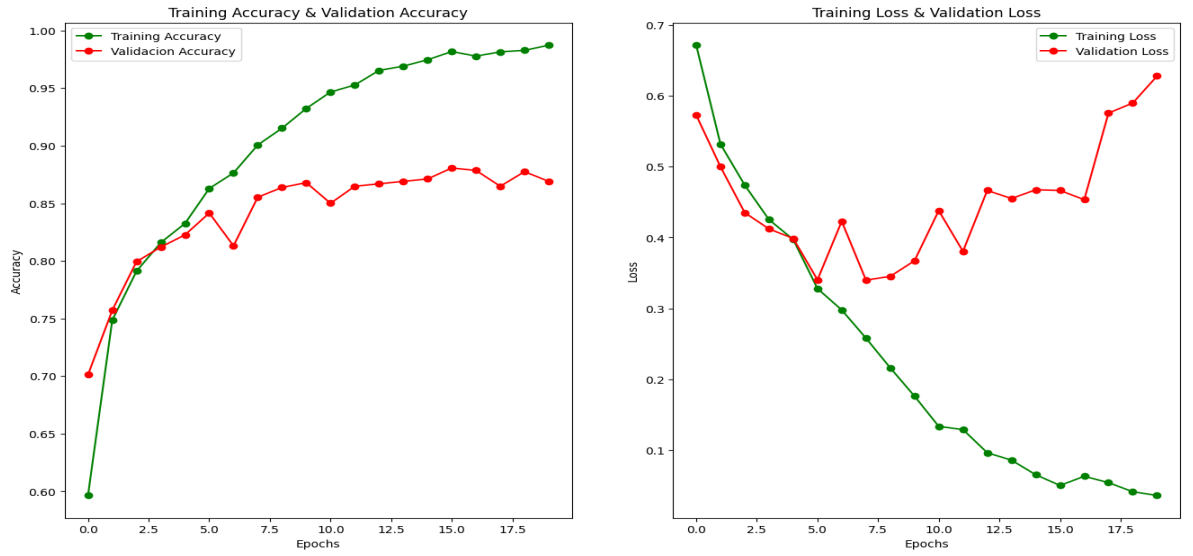

Figure 3

Increasing the number of convolutional and dense layers led to a notable improvement in validation accuracy, rising to approximately 0.87. However, the model exhibited poor performance concerning overfitting and validation loss.

3.5 3rd model: 3 Convolutional layers, 1 Dense layers

Reducing the number of dense layers can improve the model's performance by reducing and enhancing generalization.

The next model has the following structure:

- *1st Convolutional Layer*:
    - **Conv2D** layer with 32 filters and a kernel_size of 3 x 3,
- *1st Pooling Layer*:
    - **MaxPooling2D** layer with pool_size equal to 2 x 2,

- *2nd Convolutional Layer*:
    - **Conv2D** layer with 64 filters and a kernel_size of 3 x 3,
- *2nd Pooling Layer*:
    - **MaxPooling2D** layer with pool_size equal to 2 x 2,
- *3rd Convolutional Layer*:
    - **Conv2D** layer with 128 filters and a kernel_size of 3 x 3,
- *3rd Pooling Layer*:
    - **MaxPooling2D** layer with pool_size equal to 2 x 2,
- *Flatten Layer*
- *Dropout* layer with $p = 0.5$,
- *1st Dense layer:*
    - **Dense** layer with 128 nodes,
- *Final Dense Layer with 1 node*

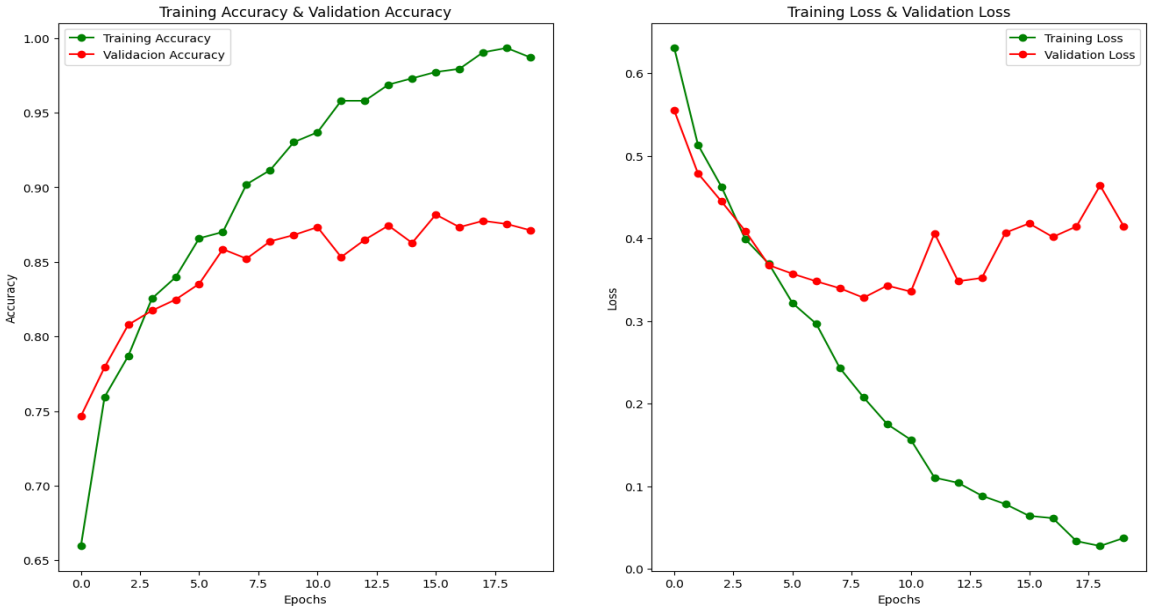

Figure 4

This model performed better since it achieved a higher validation accuracy (0.87), and a lower validation loss (0.4).

The reason for the improvement in the performance of this model lies in its simpler architecture, enhanced feature extraction capabilities, avoidance of redundancy, and improved training efficiency compared to the model with two dense layers.

3.6 4th model: 4 Convolutional layers, 1 Dense layer

Increasing the number of convolutional layers may improve performance because it allows the network to learn hierarchical features, capture spatial context, and act as a regularization exploiting parameter sharing.

The next model is structured as follows:

- *1st Convolutional Layer*:
    - **Conv2D** layer with 32 filters and a kernel_size of 3 x 3,
- *1st Pooling Layer***:**
    - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**
- *2nd Convolutional Layer*:
    - **Conv2D** layer with 32 filters and a kernel_size of 3 x 3,
- *2nd Pooling Layer***:**
    - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**
- *3rd Convolutional Layer*:

- **Conv2D** layer with 64 filters and a kernel_size of 3 x 3,

- *3rd Pooling Layer*:
  - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**

- *4th Convolutional Layer*:
  - **Conv2D** layer with 128 filters and a kernel_size of 3 x 3,

- *4th Pooling Layer*:
  - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**

- *Flatten Layer*

- *Dropout* layer with $p = 0.5$,

- *1st Dense layer:*
  - **Dense** layer with 128 nodes,

- *Final Dense Layer with 1 node*

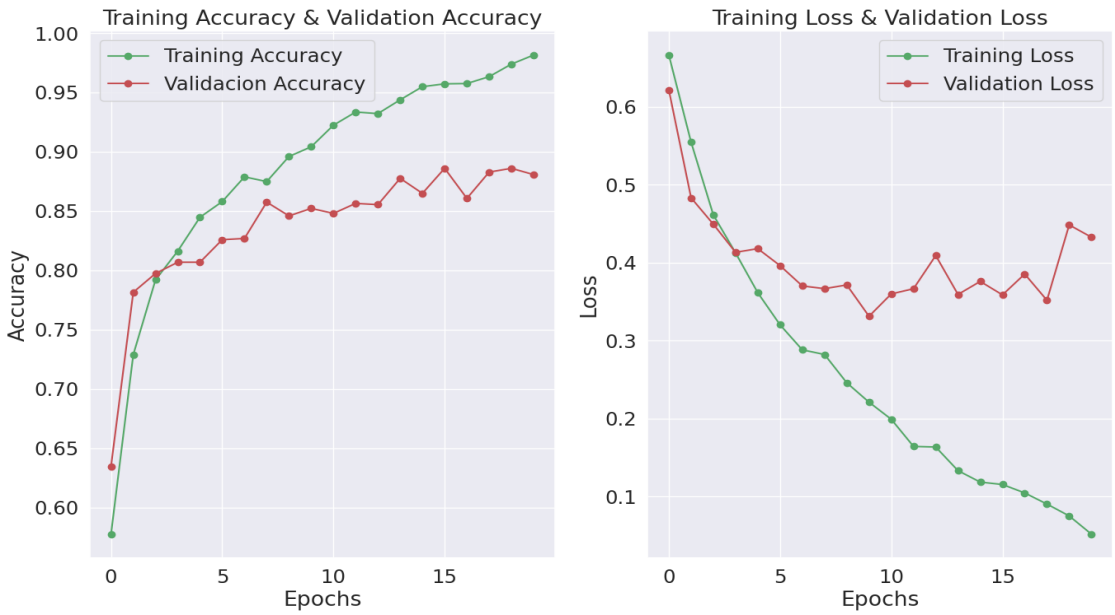

Figure 5

The model has a validation loss of 0.4 and a validation accuracy of 0.88, which appears to be a better-performing model than the previous one.

3.7 5th model: 4 Convolutional layers, 2 Dense layer

In the next step, a modification is made to the model's structure by adding a dense layer, aiming to evaluate if this adjustment enhances the model's performance.

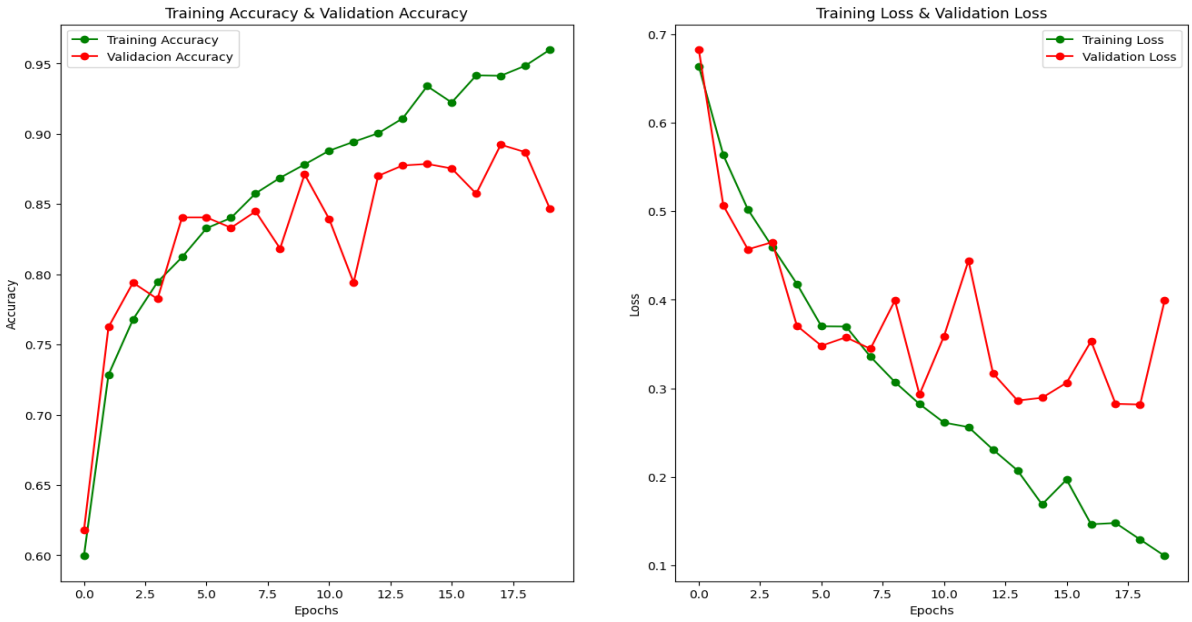

Figure 6

The results suggest that this model performs worse than the previous one. Despite a slight improvement in mitigating overfitting, the validation accuracy is significantly lower, and the validation loss remains relatively unchanged.

4. Hyperparameter Tuning

To conduct the optimization process, the fourth model is chosen as the final architecture due to its superior performance. Each parameter will be adjusted one at a time while keeping all other settings unchanged.

The first parameters to be optimized are the *learning rate* of the Adam optimizer and *the batch size*. Learning rates of 10^{-3}, 10^{-4}, 10^{-5} will be tested, along with batch sizes of 32 and 64.

After experimentation, the model with a batch size of 32 and a learning rate of 0.001 is selected. This model achieved a **validation accuracy of 0.89** and a **validation loss of 0.31**.

The next goal is to optimize the *dropout rate*, considering values ranging from 0.6 to 0.9.

After experimentation, the model with a dropout rate of 0.85 emerged as the optimal choice.

This adjustment effectively reduced the **validation loss to 0.24** while maintaining a **validation accuracy of 0.89**, effectively addressing the issue of overfitting.

The next step is to optimize the *number of convolution filters*. We consider filter numbers from the set {32, 64, 128}. A for-loop is created to train a model for each combination of these three numbers. To expedite the process due to the relatively high number of models, the patience parameter of the early_stopping callback is set to 10. After experimentation, the optimal numbers of convolutional filters are determined to be 32, 64, 128, and 128, respectively. This model achieved a **validation accuracy of 0.9**.

The subsequent parameter to fine-tune is the *number of nodes* in the dense layer. Values are tested from the set {32, 64, 128, 256, 512}. After analysis, it is found that the highest validation accuracy and lowest validation loss are attained with 128 nodes in the dense layer.

The final hyperparameters to tune are the *kernel_size and the pool_size*, both set in the range from 2 to 5. After experimentation, the combination of a pool_size of 2 and a kernel_size of 4 was chosen for the final model.

5. Final Model

After conducting hyperparameter tuning, the best-performing model has the following architecture:

- *1st Convolutional Layer*:
    - **Conv2D** layer with 32 filters and a kernel_size of 4 x 4,
- *1st Pooling Layer*:
    - **MaxPooling2D** layer with pool_size equal to 2 x 2,
- *2nd Convolutional Layer*:
    - **Conv2D** layer with 64 filters and a kernel_size of 4 x 4,
- *2nd Pooling Layer*:
    - **MaxPooling2D** layer with pool_size equal to 2 x 2,
- *3rd Convolutional Layer*:
    - **Conv2D** layer with 128 filters and a kernel_size of 4 x 4,
- *3rd Pooling Layer*:
    - **MaxPooling2D** layer with pool_size equal to 2 x 2,
- *4th Convolutional Layer*:
    - **Conv2D** layer with 128 filters and a kernel_size of 4 x 4,

- *4th Pooling Layer*:
    - **MaxPooling2D** layer with pool_size equal to 2 x 2**,**
- *Flatten Layer*
- *Dropout* layer with *p= 0.85*,
- *1st Dense layer:*
    - **Dense** layer with 128 nodes,

Final Dense Layer with 1 node

Each layer in the model is equipped with a ReLu activation function, except for the output layer, which utilizes a sigmoid activation function. The learning rate is set to 0.001, the batch size is 32, and the loss function utilized is binary cross-entropy.

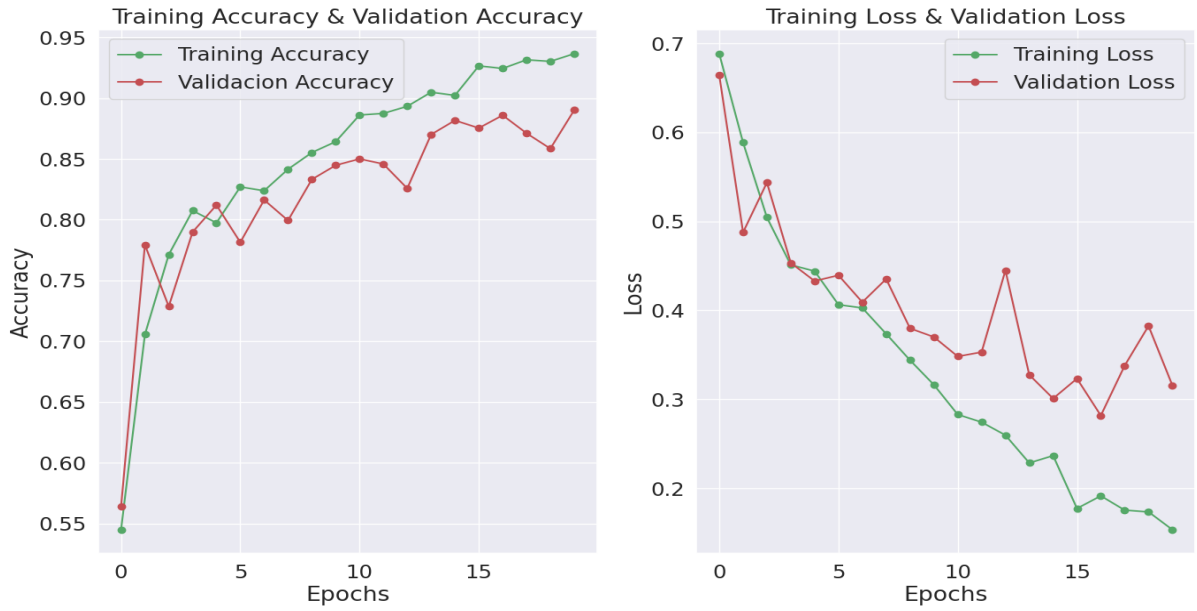

Figure 7

The results indicate that the model achieved a training loss of 0.15 and a training accuracy of 0.93, along with a **validation loss of 0.24** and a **validation accuracy of 0.9105**. This represents the best-performing model obtained thus far, with minimal overfitting and stable performance.

Additionally, the model demonstrates good generalization to unseen data, as evidenced by the test loss and accuracy, which align closely with the validation set metrics. Specifically, the **test loss is 0.24**, and the **test accuracy is 0.9105**.

5.1 Confusion Matrix

Looking at the confusion matrix, it's apparent that the model tends to predict Chihuahuas more frequently than muffins. This imbalance suggests that the network more commonly predicts Chihuahuas even when presented with images of muffins. This bias could stem from the fact that there are more images of Chihuahuas in the dataset for the model to learn from, leading it to prioritize patterns associated with Chihuahuas over those of muffins.

The misclassification rate for Chihuahua images stands at 5.625%, whereas for muffins, it is notably higher at 12.868%.

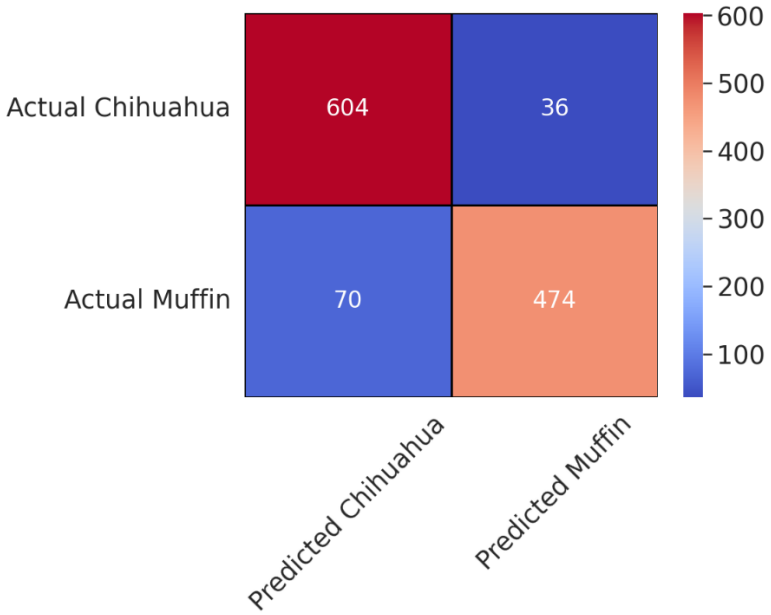

Figure 8

5.2 5-Fold Cross-Validation

In the concluding phase, cross-validation with 5 folds is executed to calculate the risk estimates of the final model. The results of the cross-validation utilize the zero-one loss:

K-Fold	*0-1 Loss*	*Accuracy*
Fold 1	0.0939	0.9060
Fold 2	0.0802	0.9197
Fold 3	0.0781	0.9218
Fold 4	0.0930	0.9069
Fold 5	0.0803	0.9196
Mean	**0.0851**	**0.9149**

It is noted that the losses fall within a small interval, indicating a high level of stability in the model.