Università degli Studi di Milano
Data Science for Economics

CNN for Chihuahuas vs. Muffins

Emile Rahal

Machine Learning Module

March 2023

Outline

1. Abstract
2. Introduction
3. Data Handling
4. Convolutional Neural Netwroks
5. Network Building
6. Refinement and Hyperparameter Tuning
7. Training
8. Testing
9. Cross-Validation
10. Conclusion

1 Abstract

In this project, we focus on the binary classification of Muffins and Chihuahuas using convolutional neural networks.

This report explains the process of transforming images to pixel values and exploring and understanding the choice of network architectures and the tuning of hyper-parameters.

So, after experimenting, we will be able to understand how different components and hyper-parameters of the neural network can affect its predictive performance.

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.

2 Introduction

The main goal of this project is to build a binary classifier for images that show a muffin or a chihuahua. Without a doubt, comprehensive data pre-processing and systematic exploration are the foundations of any successful machine learning project.

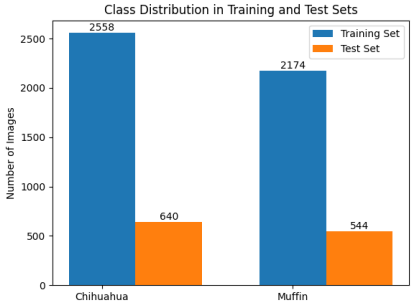
Our exploration relies on a dataset containing images featuring both detectable muffins and Chihuahuas, sourced from Kaggle, and scraped from Google images. The dataset was already split into training and test sets divided into folders for each class.

With the help of TensorFlow and the flexible Keras framework, we explore different architectures and configurations for our convolutional networks to open a discussion on their impact. Adding Hyper-parameter tuning, is an important step in enhancing model performance, with a focus on regions where performance trade-offs appear. And, the application of 5-fold cross-validation can solidify our risk estimates, providing a robust evaluation framework.

In this report, I use "we" even though it's just me to make things sound friendly and collaborative.

3 Data Handling

Data is already split into 2 sets, the training set comprises 4733 images, with 2559 Chihuahuas and 2174 Muffins, and the test set of 1184 images, there are 640 Chihuahuas and 544 Muffins.

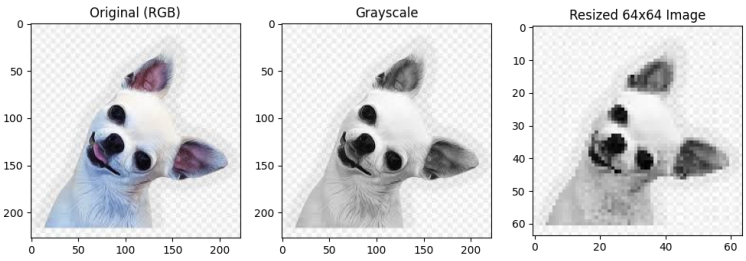


We ensured that the file extension of each image was correct (JPG), and checked for image corruption to maintain the integrity of the dataset.

3.1 Loading and Preprocessing

The data loading process involves obtaining images from specified directories for both training and test sets, without using Tensorflow and Keras to have control on each step of the process.

Using the OpenCV library, the fully-sized images are loaded in grayscale to reduce the dimensionality. Grayscale images have a single channel, making them computationally more efficient. The loaded images are then resized to 64x64 which was chosen also for computational efficiency without touching essential visual information.



The images are categorized into Chihuahuas (class 0) and Muffins (class 1). For each image, the corresponding class label is assigned, and the preprocessed image is appended to either the training or test lists where we have our image

pixel values going from 0 to 255. This step is important for collecting the necessary data for further processing.

3.2 Shuffling and Splitting

After loading the data, introducing randomness by shuffling both the training and test data is essential. Random shuffling ensures that the model doesn't learn patterns based on the order of the data.

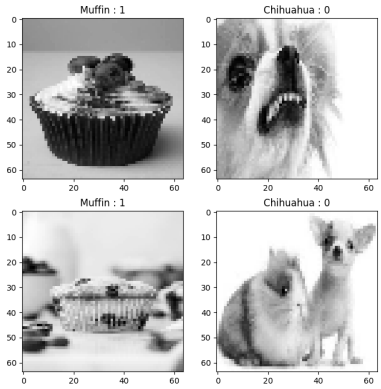


Figure 1: First 5 Samples from Training Set

Following shuffling, the data is split into features (X) and labels (y) for both the training and test sets. Features represent the preprocessed images, and labels correspond to the class labels (0 or 1).

3.3 Transformation and Normalization

Other than converting the lists to NumPy arrays, for Convolutional Neural Networks, the shape of the data is adjusted to (batch size, height, width, channels). This reshaping ensures that the data is compatible with CNN architectures. In this case, the images are reshaped to (batch size, 64, 64, 1) to represent the grayscale channel.

Normalization is then applied to the pixel values, scaling them to a range of [0, 1]. It ensures that the input values are within a consistent range. Normalizing pixel values helps prevent issues like vanishing gradients during the training process.

Lastly, the preprocessed data is saved for future use and stored as pickle files including X train, X test, y train, and y test which allows easy access without the need to repeat the preprocessing steps each time the model is trained or evaluated.

4 Convolutional Neural Netwroks

CNNs are particularly well-suited for image classification tasks, and their architecture is designed to capture patterns within images. This makes them very effective for recognizing complex features in visual data, like the unique characteristics of Chihuahuas and Muffins in our dataset.

4.1 Convolutional Layers

The base of a CNN is the convolutional layers. These layers use filters or kernels that slide across the input images, performing convolution operations. This process allows the network to extract essential features such as edges, textures, and patterns. The output of these layers will be a feature map.

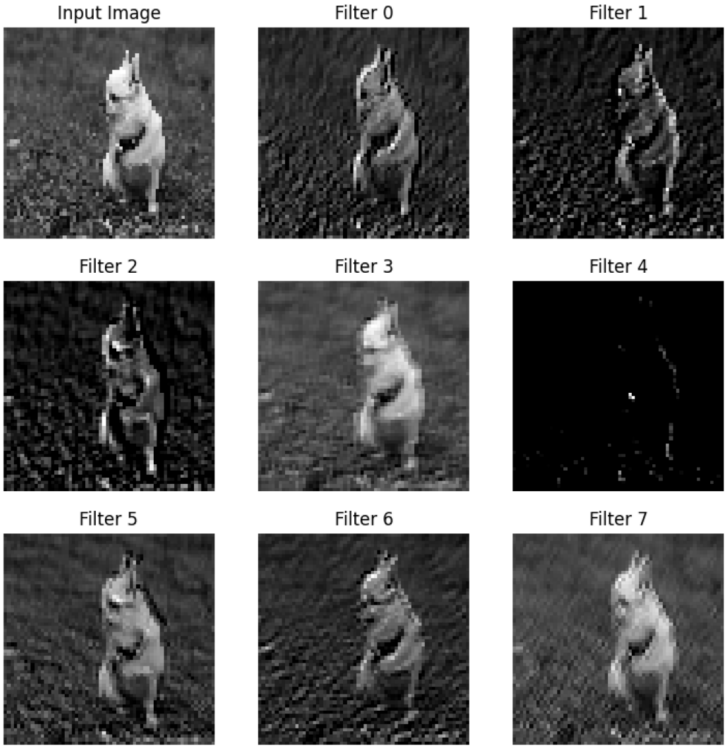
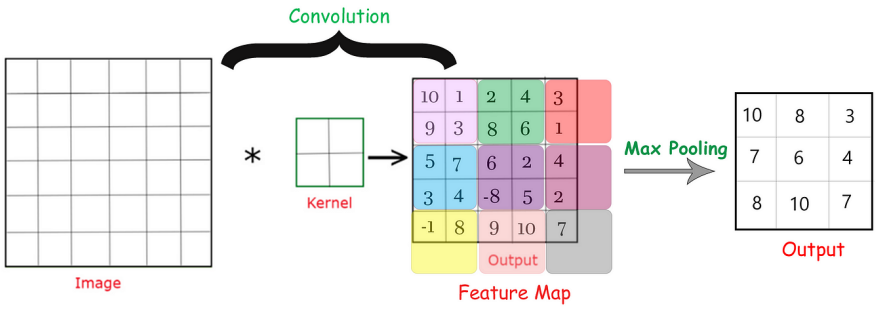


Figure 2: Some of the filters applied on an image

4.2 Pooling Layers

Pooling layers, typically in the form of max pooling, follow convolutional layers and serve to downsample the spatial dimensions of the feature map (the output of the conv). This reduces the computational load and retains the most

significant features. Max pooling selects the maximum value from a group of neighboring pixels, to care about the key information and discard less relevant details.



4.3 Activation Functions

Activation functions introduce non-linearities to the network, enabling it to learn complicated relationships within the data.

ReLU (Rectified Linear Unit):

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

ReLU is a commonly used activation function in CNNs due to its simplicity. It replaces negative values with zero, aiming for faster convergence during training (it returns the maximum between 0 and the input x)

Sigmoid(Output):

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

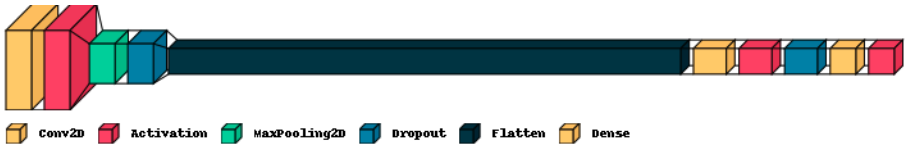
Sigmoid is used for binary classification. It transforms the output to a value between 0 and 1, making it suitable as an output layer activation in our model where the output needs to be in a probability-like range.

4.4 Flattening Layer

After the convolutional and pooling layers, a flattening layer reshapes the output into a one-dimensional vector. This step is important for transitioning from the stuff learned by previous layers to a format suitable for the fully connected layers.

4.5 Fully Connected Layers

The flattened output is then passed through fully connected layers, which operate similarly to traditional neural networks. These layers capture relationships in the data. The final fully connected layer produces the output that corresponds to the classification decision (that will be the input for the 'Sigmoid')



4.6 Loss Function

The task of this project is to make a binary classification of images showing chihuahua or muffin. Therefore, the loss function used in this project is the binary cross-entropy:

$$\mathcal{L}_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))\right)$$

Here, N is the number of images, y_i is the true label (1 for muffin, 0 for chihuahua), and $p(y_i)$ is the predicted probability of being a muffin image.

The binary cross-entropy loss penalizes bad predictions by adding to the loss. For each image labeled as a muffin ($y_i = 1$), it adds the log probability of being a muffin image ($\log(p(y_i))$), and for chihuahua images ($y_i = 0$), it adds the log probability of being a chihuahua image ($\log(1 - p(y_i))$).

4.7 Optimizer

The optimizer is responsible for minimizing the loss function during training. To achieve this goal, it is required to establish the learning rate, a hyperparameter that determines the step size of how weights need to be adjusted during training.

Adam updates the weights using a weighted sum of the first and second moments of the gradients. This adaptive learning rate technique helps in efficient convergence, ensuring that the model can discern the complicated features that distinguish Chihuahuas from Muffins.

Early Stopping

To prevent overfitting in general, we implemented early stopping with the patience of 4 epochs during the whole project implementation. This allowed the optimization process to stop if the validation loss did not improve for 4 consecutive epochs.

5 Network Building

To identify the optimal Convolutional Neural Network (CNN) for our binary classification task, we adopted a systematic approach:

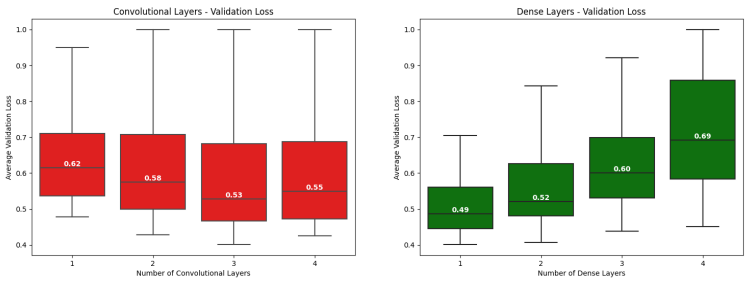
We explored a diverse set of architectures by varying the number of hidden layers (convolutional and dense) and initial values for the number of filters and neurons. The range extended from 16 to 256 (as initial value), doubling whenever a layer was added. Each architecture represents a unique combination of these values, and models were named based on their architecture for easy identification, for example:

```
"cnn_model__conv_layers_256_512_1024__dense_layers_128"
"cnn_model__conv_layers_64__dense_layers_256"
"cnn_model__conv_layers_16_32_64_128__dense_layers_128_256"
```

In total, we generated 400 potential architectures, considering all the combinations. Next, we built and trained these architectures on our original training set which provided us with a variety of trained models, each with its training history. The models were trained, with batch size = 32 and epochs = 15, for 1 time without any configurations, so everything was on the default value.

This allowed us to see the pure impact of architecture (layers and values) building on the model training while assessing the validation loss.

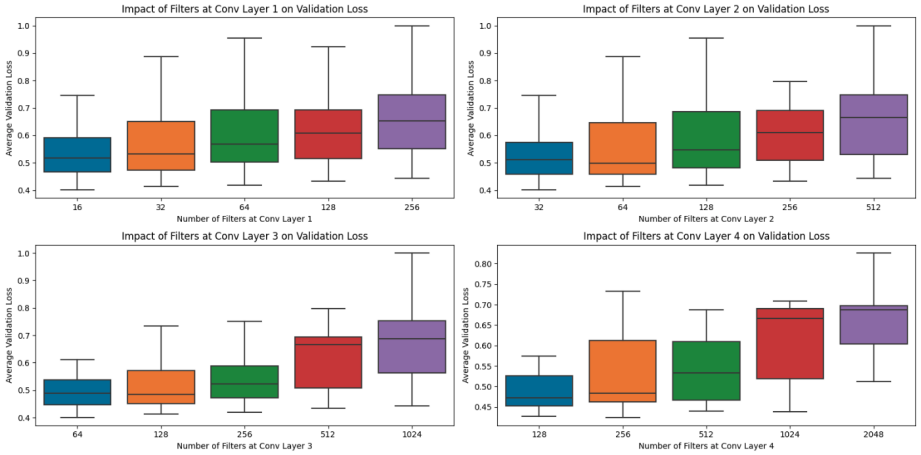
After handling the 'new' data which is the models history, having all the information needed to make inferences about the effect of these components on the average validation loss of the training of each model.



Looking at the box plots of the number of layers above, we can see that for convolutional layers, an increase in the number of layers corresponds to a consistent decrease in validation loss. This suggests that deeper convolutional

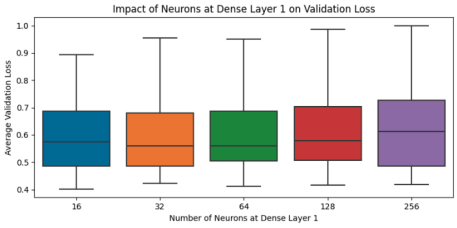
architectures generally lead to improved performance. We see that having 3 to 4 conv layers is the best choice for lower loss.

On the contrary, in dense layers, a rise in the number of layers is associated with an increase in validation loss. This indicates that, on average, deeper dense architectures tend to perform less effectively. We see that having just 1 dense layer (fully connected layer) is the best choice.



Looking at the figure above, we can notice that for each layer an increase in the number of filters corresponds to higher validation losses. The optimal configuration appears to have a range from 16 to 256 max for filters.

In summary, we can consider having a range [16-256] for incremented filter values, respectively. These incremented values will be the range for each layer, increasing the number of filters as we add layers.



Since we already mentioned that having one fully connected layer after the flattened layer is the best choice, we checked for just layer 1 insights, we noticed that nearly every value led to the same loss more or less, so we can say any value from 16 to 256 is perfect.

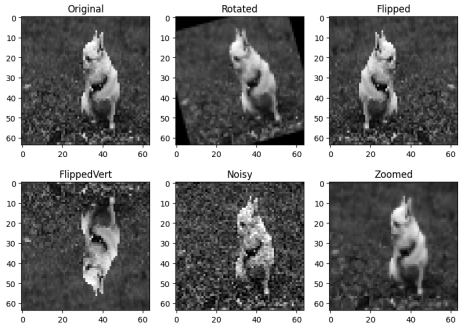
6 Refinement and Hyperparameter Tuning

Since we got to the point where we know the number of layers and the range of values for filters and neurons we can refine our model to get a lower loss.

We noticed that the majority of the models that we trained on the training data suffer from overfitting, where the validation loss goes up as the model is trained.

To avoid that we try to implement some **Data Augmentation** where we can add more diversity to our training set. This helps the model generalize better to unseen data and reduces the risk of overfitting.

Transformation	**Settings**
Rotate	-30° to 30°
Flip	Horizontal
Flip	Vertical
Noise	Gaussian
Random Zoom	0.8 to 1.2

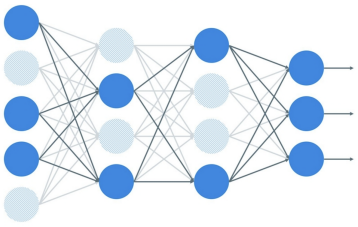


I didn't use Keras for the transformations but a custom function where I specified the settings, at the end we got around 20000 new images. That will be helpful for our model to generalize more.

In addition to data augmentation, we also introduced regularization like Dropout layers for convolutional and fully connected.

Dropout

It randomly "drops out" a fraction of neurons in the densely connected layers, preventing overfitting. This regularization encourages the network to rely on a diverse set of features, reducing the likelihood of memorizing the training data and improving its ability to generalize to unseen images.



To continue, after exploring architectures and identifying potential models, we performed **Hyperparameter Tuning** to refine the selected models and achieve better performance. The hyperparameter tuning process aimed to find the optimal configuration for parameters such as dropout rates, number of filters (based on the range specified using past insights), and number of units in dense layers.

Hyperparameter	**Range**
`dropout_hidden_layer`	10% - 30%
`dropout_flatten_layer`	20% - 50%
`convolution_1_filters`	16 - 32
`convolution_2_filters`	32 - 64
`convolution_3_filters`	64 - 128
`convolution_4_filters`	128 - 256
`dense_neurons`	64 - 256

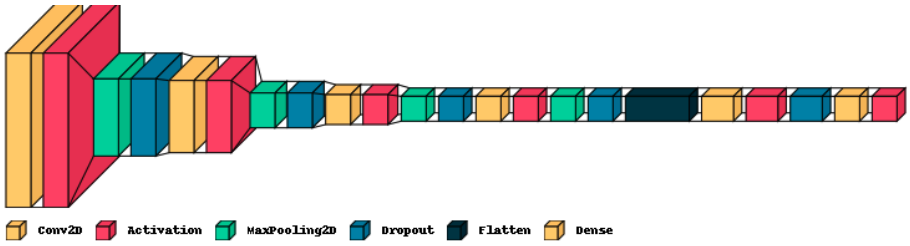
We utilized Bayesian Optimization with Keras Tuner to search for the best values that led to the highest validation accuracy.

Although all the combinations were showing decent results, looking at the results best model after 40 trials:

Best Validation Accuracy	0.8965
Elapsed Time	01h 27m 05s

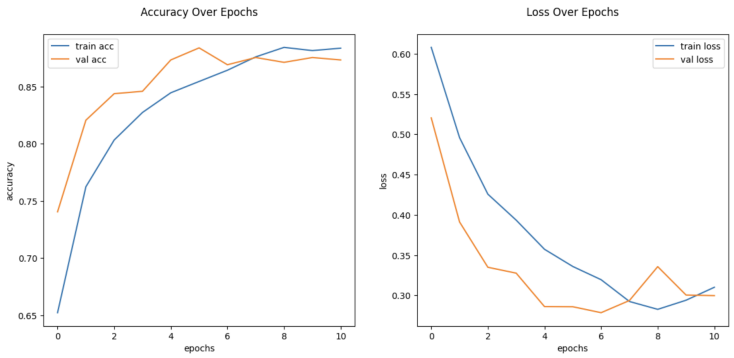
Hyperparameter	**Value**
Dropout (Hidden Layer)	25%
Dropout (Flatten Layer)	40%
Convolution 1 Filters	16
Convolution 2 Filters	64
Convolution 3 Filters	128
Convolution 4 Filters	128
Dense Neurons	64

Our model layers view would look like that:



7 Training

This best model had some instability looking at the loss curves below:



The best model showed some instability in its training, with the loss decreasing from 0.6080 to 0.3101 and accuracy improving from 65.23% to 88.37% over 11 epochs. However, fluctuations in both training and validation losses were observed. The validation loss reached a minimum of 0.2785, achieving a peak accuracy of 87.55%.

Despite these fluctuations, the model demonstrated reasonable accuracy on the validation set but can be more refined.

After that, we decided to check for the batch size and the learning rate so that maybe these fluctuations would go away and the model could be more stable.

Batch Size

The batch size is the number of training examples utilized in one iteration. A larger batch size may lead to faster convergence, but it demands more memory.

Learning Rate

The learning rate determines the size of steps taken during optimization. A higher learning rate might speed up training but risks overshooting the optimal values, while a lower rate ensures stability but might slow down convergence.

We experimented with the following values:

Batch Size
32
64
128

Learning Rate
0.0001
0.001

The best hyperparameter values that yielded optimal results are a batch size of 128 and a learning rate of 0.001, which showed a balance between efficient training and stability in convergence.

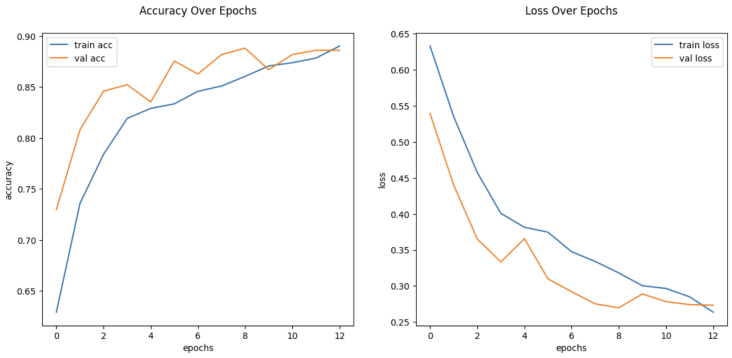


Figure 3: LR 0.001, BS 128

The selected learning rate of 0.001 and batch size of 128 contributed to model performance. The training and validation loss (above) steadily decreased over the epochs, indicating improved convergence. The accuracy metrics also showed positive trends, with training accuracy reaching 89.02% and validation accuracy peaking at 88.61%. This suggests that the model generalizes well to unseen data, reflecting a successful optimization of hyperparameters.

8 Testing

After training, we conducted a thorough evaluation of its performance on the test set. The model achieved a test loss of 0.2795 and an impressive test accuracy of 88.26%.

Classification Report

Precision	Recall	F1-Score	Accuracy
0.88	0.90	0.89	**88.26%**
0.88	0.86	0.87	

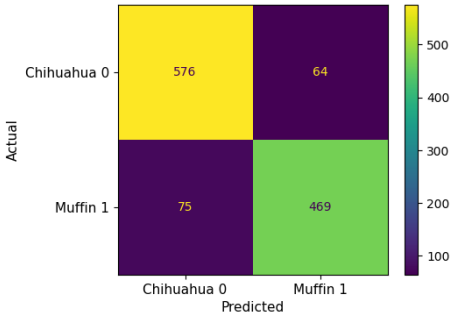
Precision is the ratio of correctly predicted positive observations to the total predicted positives. Both classes have a precision of 88%, indicating a low rate of falsely predicted positives.

Recall, or sensitivity, is the ratio of correctly predicted positive observations to all actual positives. Class 0 has a higher recall (90%) than Class 1 (86%), suggesting better identification of positive instances in Class 0.

The F1-score is the balance mean of precision and recall. High F1 scores for both classes indicate a good balance between precision and recall.

Confusion Matrix

In the Confusion Matrix, True Positives (469) are instances correctly predicted as Class 1, while True Negatives (576) are instances correctly predicted as Class 0. False Positives (64) are instances wrongly classified as positive, and False Negatives (75) are instances incorrectly classified as negative. The model demonstrates a reasonably balanced performance in distinguishing between the two classes.



9 Cross-Validation

To prepare for 5-fold cross-validation, the dataset is divided into five subsets, and in each iteration, one subset serves as the validation set while the others are used for training.

The pre-trained optimal model is loaded, and for each fold, a clone of the model is created and trained on the training subset. The model is then evaluated on the validation subset, and performance metrics such as **Zero-One** loss are recorded. The zero-one loss is a straightforward assessment of classification accuracy. Where correct predictions receive a value of 0, and incorrect predictions receive a value of 1. It quantifies the accuracy of the model by counting misclassifications. In our case, a lower zero-one loss indicates better performance.

This process is repeated for all folds, and the average metrics are calculated to assess the overall performance of the model across different validation sets.

Fold	**Validation Accuracy**	**Zero-One Loss**
1	0.8734	0.1266
2	0.8945	0.1055
3	0.9072	0.0928
4	0.8439	0.1561
5	0.8856	0.1144

Table 1: CV Results

Across the five folds, the model consistently demonstrated strong performance in classifying with an average zero-one loss of 0.119.

The average validation accuracy remained high at 88.1%, indicating reliable predictive capability across different subsets of the data.

10 Conclusion

Architecture Impact

Deeper convolutional architectures generally led to improved performance, with 3 to 4 convolutional layers identified as optimal. In contrast, deeper dense architectures tended to perform less effectively, highlighting the importance of a concise architecture, particularly with only one dense layer.

Filter Values

The analysis of filter values demonstrated its impact on validation loss. While an increase in the number of filters generally correlated with higher validation losses. We identified a range, incrementing filter values for each layer, which was the base for building effective CNN architectures.

Hyperparameter Tuning

Hyperparameter tuning, including data augmentation and early stopping, played a role in model optimization. And the adaptive learning rate helped converge the model.

Cross-Validation Performance

The application of 5-fold cross-validation reinforced risk estimates, providing a robust evaluation setting. It showed an understanding of the model's predictive performance across different validation sets.

We can say we were able to understand how we can use CNN effectively to classify images optimally.

Github Repo: CNN_Chihuahuas_vs_Muffins
Dataset: Images Dataset on Kaggle