Chihuahua and Muffin Image Classification

Mahnaz Taheri

May 2024

Abstract

This project explores different CNN models and adjusts their hidden layers and filters to see how they affect overfitting. Data augmentation is also used to make the training data more diverse, helping the models learn better. By testing these models, the best settings to reduce overfitting is found while keeping the models accurate. Finally, the top-performing model is validated through cross-validation and evaluated on the test set, resulting in improved performance with test accuracy of 97%.

Figure 1: First Batch of Training Set

Introduction

Convolutional Neural Networks (CNNs) have become fundamental tools in various machine learning applications, particularly in feature or object recognition in images [1]. This project investigates the effect of manipulating the number of hidden layers and filters in five different CNN architectures. Through experimentation, the project aims to find strategies to mitigate overfitting, on of the common challenges in deep learning models.
To address overfitting, our methodology incorporates image augmentation techniques, which promote generalization. By varying the architecture parameters and evaluating performance metrics, such as accuracy and loss, the project clarifies how the complexity of a model affects its ability to generalize to new data.
Furthermore, in the final phase, cross-validation has been used to test how well the optimized CNN model performs. This helps ensure the model is reliable and effective.

1 Data and Pre-processing

1.1 Dataset

The Chihuahua and Muffin dataset, available on Kaggle, consists of 4743 training images and 1184 test images. It presents the challenge of distinguishing between photos of Chihuahuas and muffins. This dataset provides an opportunity to explore how effectively computer algorithms can learn to classify diverse objects in images.

1.2 Image Transformation

An image generator is a tool for transforming images in machine learning applications. When processing images in this project, they are converted into the RGB format, represented as tensors with dimensions (150, 150, 3), where 150x150 denotes the image's height and width, and 3 represents the red, green, and blue color channels. Additionally, as part of the pre-processing step, the pixel values of the images are rescaled from the original range of [0, 255] to a normalized range of [0, 1]. This rescaling makes the data uniform and helps the machine learning models train better and faster.
In Figure 1, the visualization of the first batch of the training set can be seen.

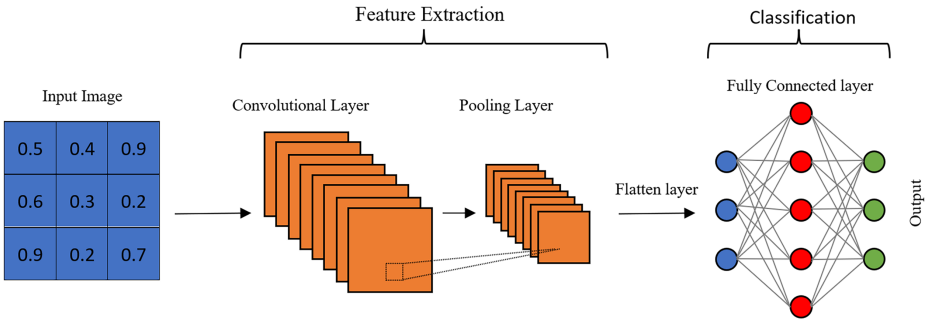

Figure 2: Architecture of a Convolutional Neural Network (CNN)

1.3 Splitting the Dataset

The dataset has been pre-divided into training and testing subsets, yet an additional validation set is constructed to aid in model evaluation and tuning. This validation set is derived from the training subset by further partitioning it, with 20 percent of the original data allocated to the validation set without being included in the training phase. By separating a portion of the data for validation, model performance can be monitored during training without being contaminated by the testing subset. This approach enables fine-tuning of hyperparameters and early detection of overfitting, enhancing the model's generalization capabilities [2].

2 CNN Architecture

In this section, the architecture and steps to construct a Convolutional Neural Network(CNN) are described. And later on, various architectures implemented on our dataset can be seen.

2.1 Layers

To begin constructing a CNN, the initial step involves defining its layers. As illustrated in Figure 2, a CNN is commonly segmented into four types of layers:

1. Convolution Layer: The convolution layer is like a filter that searches through the input data, looking for patterns and structures. Each filter is trained to recognize particular features, such as edges or textures, by comparing its weights with the input data. This comparison generates feature maps that highlight key information in the data.

2. Pooling Layer: The pooling layer reduces the spatial dimensions of the feature maps generated by the convolution layer while retaining important features. Common pooling operations include max pooling, which selects the maximum value within each pool, and average pooling, which computes the average value. By down sampling the feature maps, the pooling layer helps to decrease computational complexity and control overfitting, while preserving essential information for subsequent layers.

3. Flatten Layer: The Flatten layer in neural networks is responsible for converting multi-dimensional data, such as feature maps generated by convolutional layers, into a one-dimensional vector. This transformation enables the output of convolutional layers to be fed into fully connected layers for further processing. Basically, the Flatten layer, flattens the input data by removing all dimensions except for the batch dimension. This makes it easier to connect with the next layers in the network.

4. Fully-Connected Layer: The fully-connected layer, also referred to as the dense layer, establishes connections between every neuron in the preceding layer and every neuron in the current layer. This connectivity allows the layer to perform high-level reasoning and decision-making by receiving input from all neurons in the previous layer and applying a set of adjustable weights and biases to generate an output. This layer plays a critical role in capturing intricate relationships within the data and making predictions based on the features extracted from earlier layers [3].

2.2 Configure Architecture

Once the layers are defined, the next step is to configure the architecture of the CNN. This involves specifying the number and types of layers, as well as their respective parameters such as filter sizes, number of filters, activation functions, and any additional settings required.
The activation functions employed in this section are as follows:
1.**ReLU (Rectified Linear Activation):**

- Used for the dense layer.
- ReLU is a commonly used activation function that introduces non-linearity to the model by replacing negative values with zero.
- It helps the network learn complex patterns and accelerates convergence during training.

2.**Sigmoid:**

- Used for the output layer.
- Sigmoid is a popular activation function for binary classification tasks.
- It compresses the output values between 0 and 1, representing probabilities of belonging to one class, making it suitable for binary classification problems.

2.3 Compile Model

After configuring the architecture, the model needs to be compiled. During compilation, three key components should be specified:

1. The loss function, which measures how well the model performs on the training data.
2. The optimizer, which adjusts the model's weights to minimize the loss function.
3. Any evaluation metrics, which are used to monitor the model's performance during training.

2.3.1 Loss Function

The binary cross entropy loss function, commonly used in CNN compilation for binary classification tasks, measures the dissimilarity between the predicted probability distribution and the true binary labels. It calculates the loss by comparing each predicted probability with the corresponding true label, penalizing incorrect predictions more heavily. This loss function is well-suited for tasks where the output is binary (e.g., classifying images as belonging to one of two categories) and aims to minimize the difference between predicted probabilities and actual labels during model training.

2.3.2 Optimizer

The Adam optimizer, proposed by Kingma and Ba [4], is an adaptive learning rate optimization algorithm that incorporates elements from RMSprop and momentum techniques. It dynamically adjusts the learning rate for each parameter by considering past gradients and squared gradients, leading to faster convergence and improved performance in various scenarios. Adam's ability to automatically adapt the learning rate during training makes it particularly effective for optimizing Convolutional Neural Networks (CNNs), facilitating quicker model convergence.

2.3.3 Metric

The accuracy metric in CNN compilation measures the percentage of correctly classified samples out of the total. It's a way to assess how well the model predicts class labels. High accuracy indicates accurate predictions, while low accuracy suggests potential difficulties in classification.

3 Methodology

This section describes the different architectural configurations utilized in the study. Five distinct models were implemented, all sharing the same arguments during the fitting process. However, each of the first four models varies in terms of the number of convolution layers, hidden layers, and the quantity of filters employed. The fifth model is tested using cross-validation to check how well it works. This helps validate the best model configuration found.
All models share identical compilation arguments:
1.**Loss function**: Binary crossentropy.
2.**Optimizer**: Adam.
3.**Metrics**: Accuracy.
4.**Number of epochs**: 20

3.1 Model 1,2,3 and 4

1. Model 1: Two convolutional layers are utilized, with 32 and 64 filters, respectively, followed by a fully connected dense layer.

2. Model 2: Three convolutional layers are utilized, with 32, 64 and 128 filters, respectively, followed by a fully connected dense layers.

3. Model 3: Four convolutional layers are utilized, with 32, 64, 128 and 256 filters, respectively, followed by two fully connected dense layers.

4. Model 4: Image augmentation with four convolutional layers is utilized similarly to model 3, followed by three fully connected dense layers.

3.2 Image Augmentation

Image augmentation involves applying transformations like rotations, translations, and scaling to existing images to increase the diversity of training data. This technique helps improve the robustness and generalization of machine learning models, particularly Convolutional Neural Networks (CNNs), by mitigating overfitting and enhancing performance in tasks such as image classification and object detection [5].
In this project, the ImageDataGenerator class from Keras is performed to apply the following image augmentation techniques:

1. Zoom Range: Applies a random zoom of up to 15%, helping the model handle variations in object size.

2. Width Shift Range: Shifts images horizontally by up to 10%, making the model robust to horizontal displacements.

3. Height Shift Range: Shifts images vertically by up to 10%, aiding the model in managing vertical displacements.

4. Horizontal Flip: Randomly flips images horizontally to teach the model mirror-invariant features.

5. Validation Split: Reserves 10% of the data for validation to evaluate model performance on unseen data.

3.3 Plotting Accuracy and Loss

Plotting the accuracy and loss of training versus validation provides insight into a model's performance during training.
1: Accuracy plot of training vs validation set:

- Plotting accuracy allows to visualize how well the model is performing on both the training and validation datasets over time.
- A rising training accuracy curve indicates that the model is improving and learning from the training data.
- If the validation accuracy curve follows the training accuracy closely, it suggests that the model generalizes well to unseen data. However, if the validation accuracy plateaus or starts to decline while the training accuracy continues to rise, it indicates potential overfitting.

2: Loss plot of training vs validation set:

- Plotting loss shows how the model's performance improves in terms of minimizing errors on both the training and validation datasets.
- A decreasing training loss curve indicates that the model is reducing errors and fitting the training data better.
- Ideally, the validation loss curve should follow the training loss curve closely. However, if the validation loss starts to increase or plateau while the training loss continues to decrease, it indicates overfitting and poor generalization to new data

As depicted in Figure 3, the training accuracy curve exhibits a rising trend, eventually constant at maximum accuracy (1.0). On the other hand, the validation accuracy curve appears flat, indicating a plateau. The divergence between these two curves suggests overfitting of the model.
It can be seen in Figure 4 and 5, Models 2 and 3 appear to exhibit improved performance compared to Model 1. However, despite this improvement, the two plots of training vs validation in both models still do not closely track each other's trends.
In Figure 6, it appears that Model 4 has improved, with a reduction in overfitting observed.
In Figure 7, the training loss steadily decreases, approaching zero. However, around epoch 5, the validation loss begins to increase, signaling a possible overfitting.

It can be observed in Figure 8 and 9, that the training and validation loss show a similar trend as observed in Model 1.
While in Figure 10, it seems that the two plots closely follow each other's trends, indicating a reduction in overfitting and suggesting an improved model.

3.4 Evaluation and Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification model. It summarizes the predicted and actual classes of a dataset in a tabular format. The rows of the matrix represent the actual classes, while the columns represent the predicted classes. Each cell in the matrix contains the count of instances where the actual class was predicted correctly (true positives),

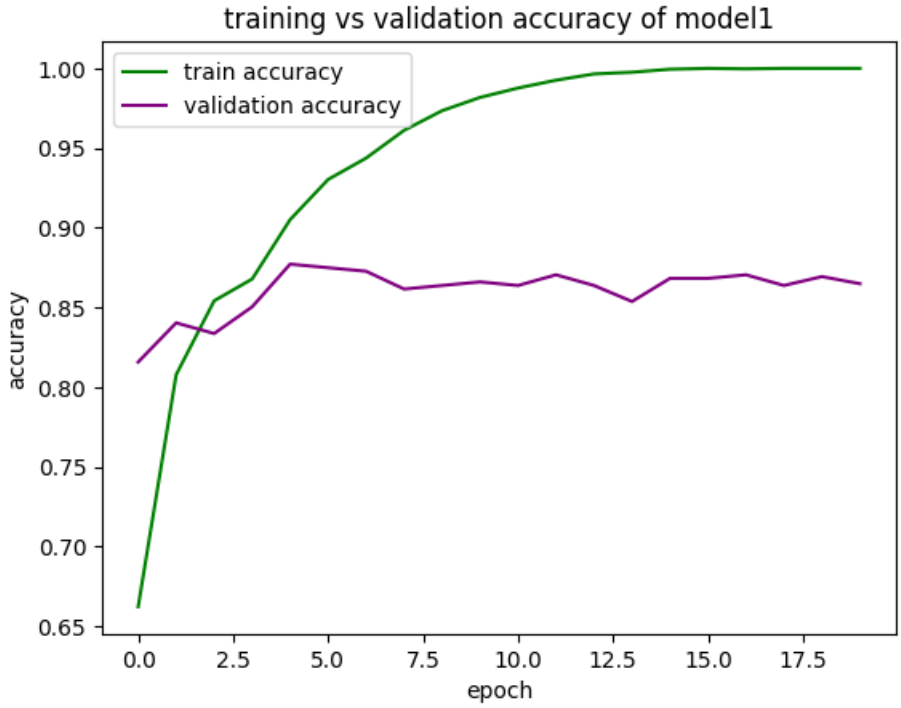

Figure 3: Training vs validation accuracy plot of model 1

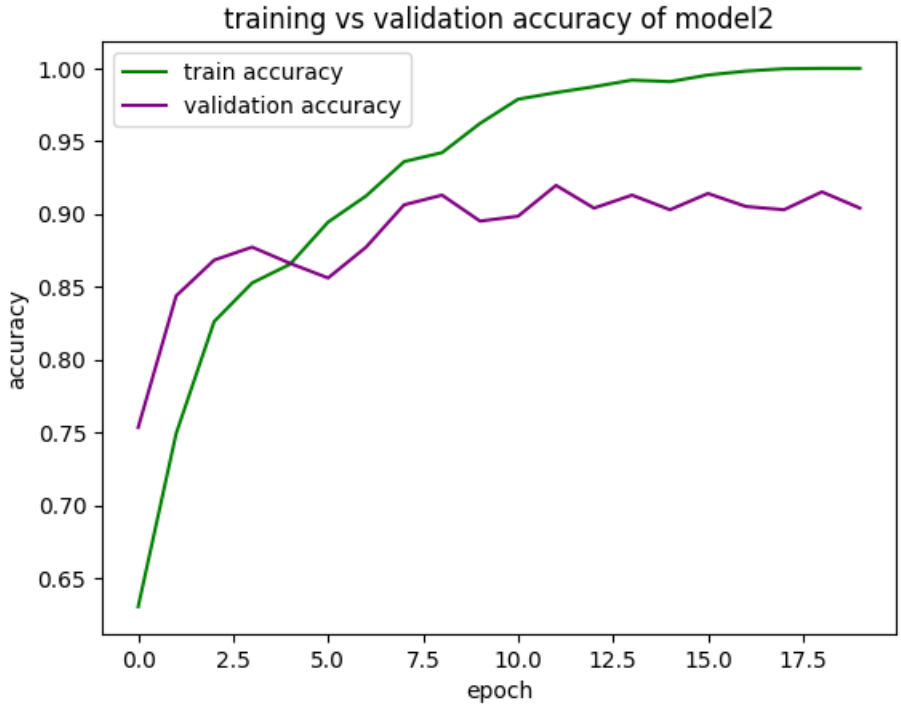

Figure 4: Training vs validation accuracy plot of model 2

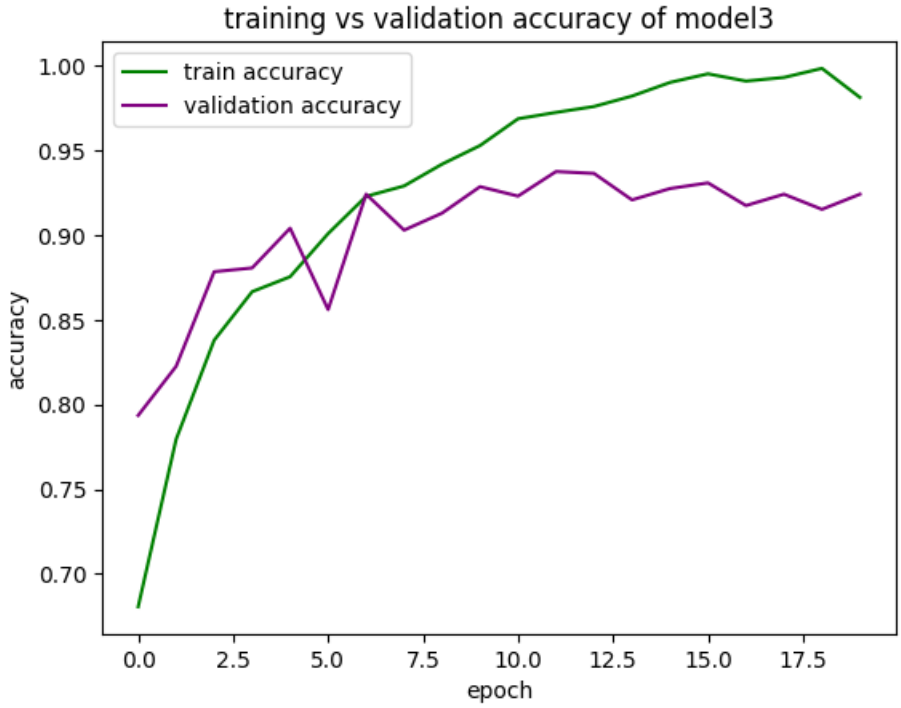

Figure 5: Training vs validation accuracy plot of model 3

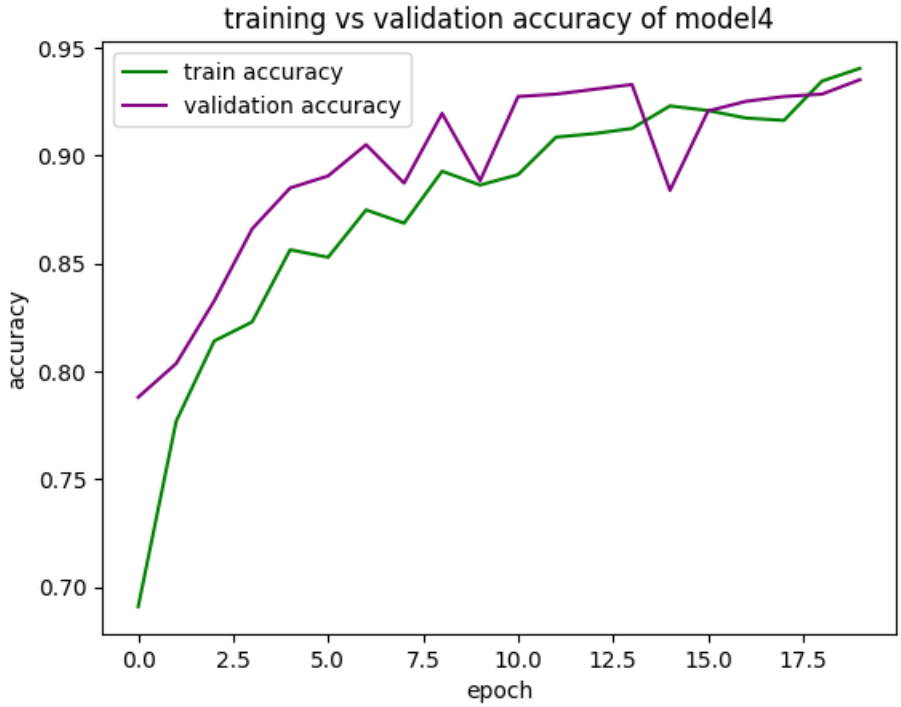

Figure 6: Training vs validation accuracy plot of model 2

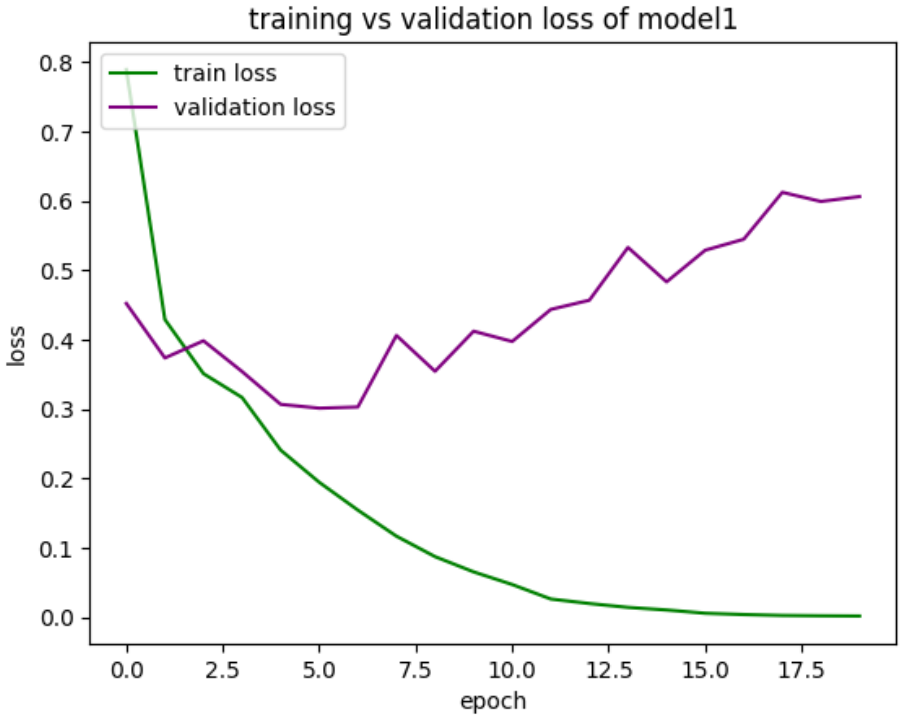

Figure 7: Training vs validation loss plot of model 1

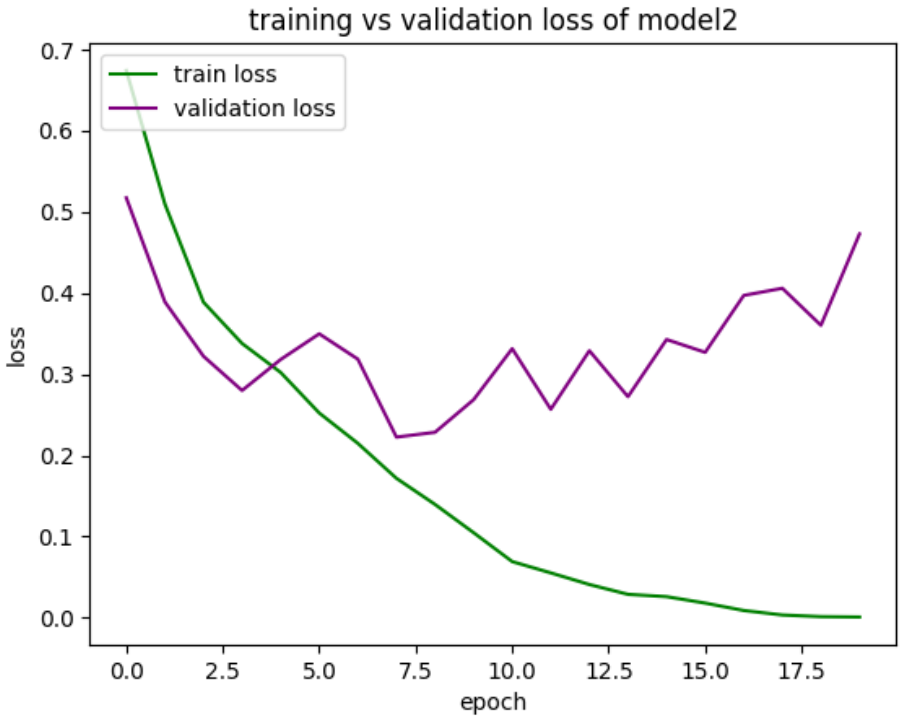

Figure 8: Training vs validation loss plot of model 2

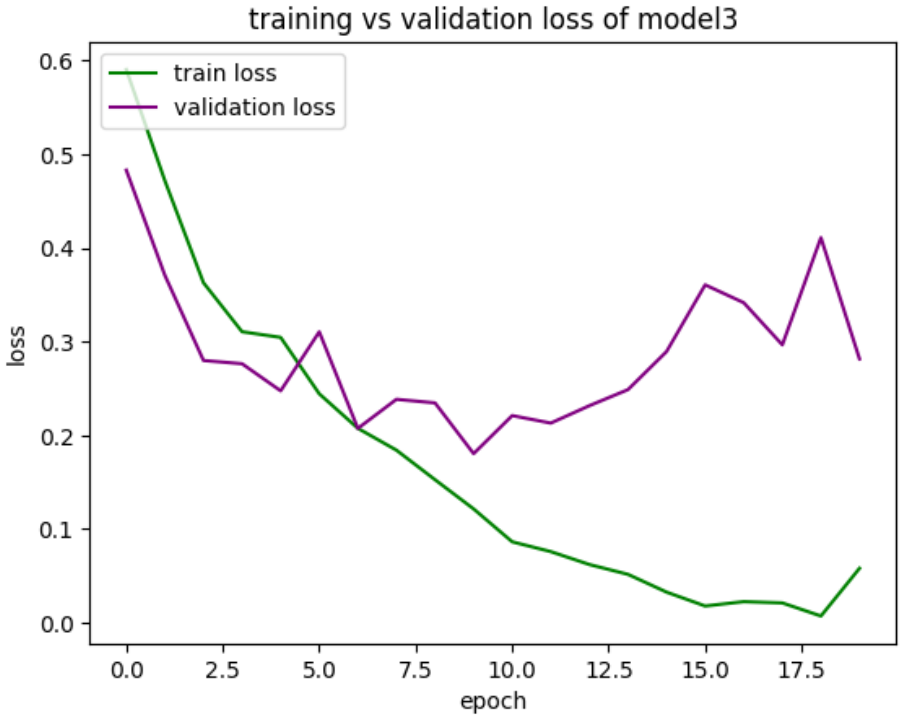

Figure 9: Training vs validation loss plot of model 3

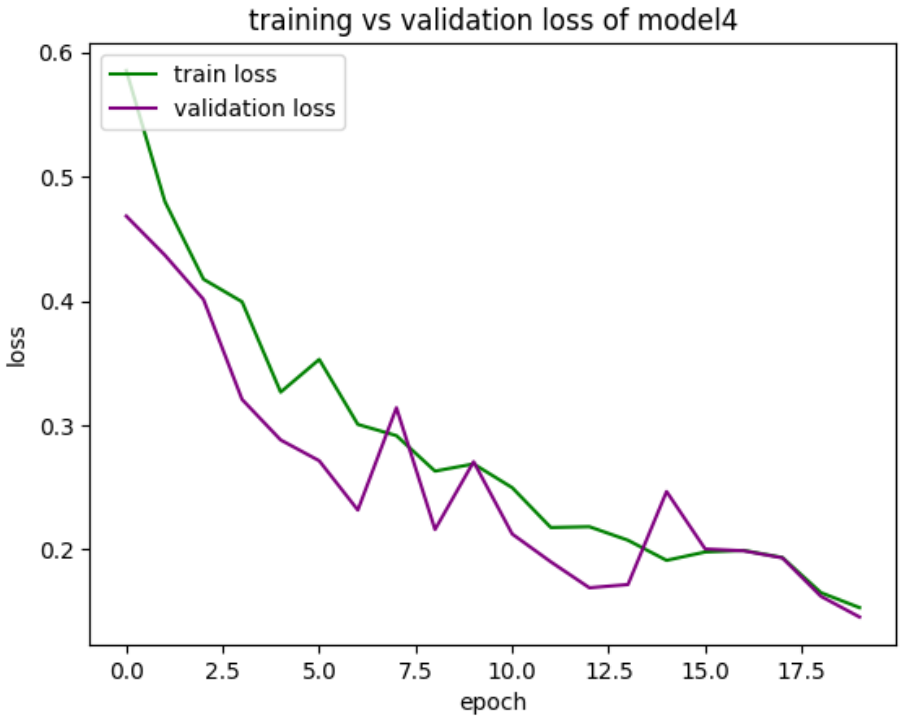

Figure 10: Training vs validation loss plot of model 4

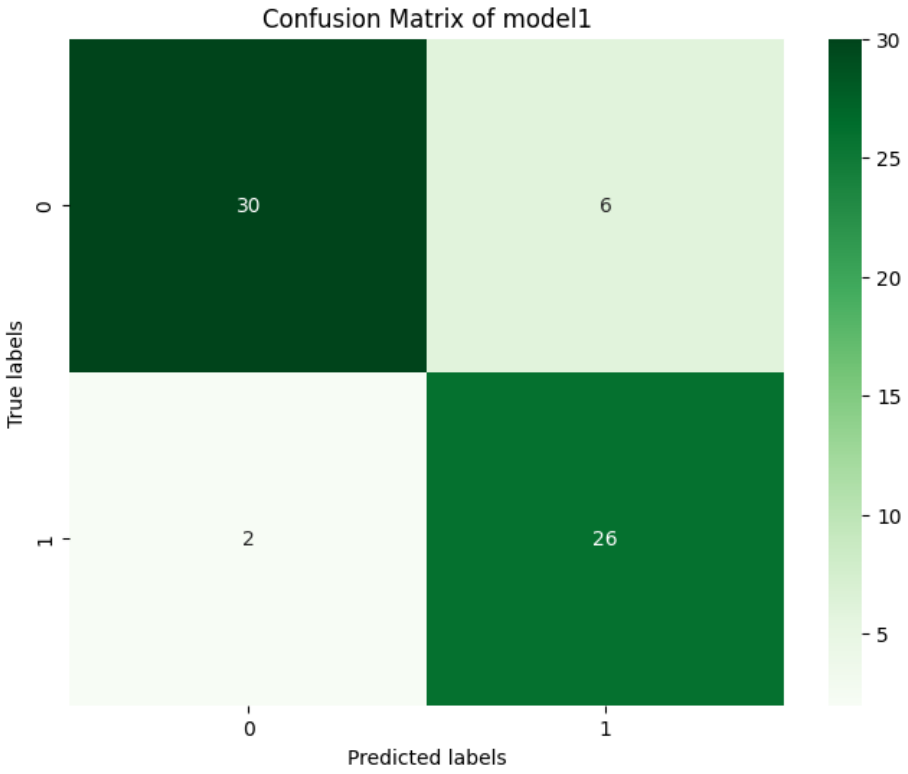

Figure 11: Confusion matrix of model 1

incorrectly (false positives), or missed (false negatives) [6].

In Figures 11, 12, and 13, we observe the confusion matrix of Models 1, 2, and 3. These plots reveal that images labeled as 0 (representing Chihuahua) and 1 (representing Muffin) are mostly classified correctly with a few misclassifications.
However, as depicted in Figure 14, the confusion matrix shows improvement, indicating that Model 4 outperforms the others.

3.5 Evaluation

In addition to analyzing accuracy and loss plots, we can assess the models' performance by testing them on unseen data, such as the test set. Ultimately, it becomes evident that incorporating additional blocks of convolution and dense layers leads to improvements in the models which can be seen in Table 1.

Table 1: Test accuracy and loss

Model	Model 1	Model 2	Model 3	Model 4
Test Accuracy	0.8598	0.8986	0.9139	0.9274
Test Loss	0.7562	0.5422	0.3031	0.1764

3.6 Cross Validation

Cross validation is a statistical technique used to evaluate and improve the performance of machine learning models by ensuring that they generalize well to unseen data [7]. The primary idea is to

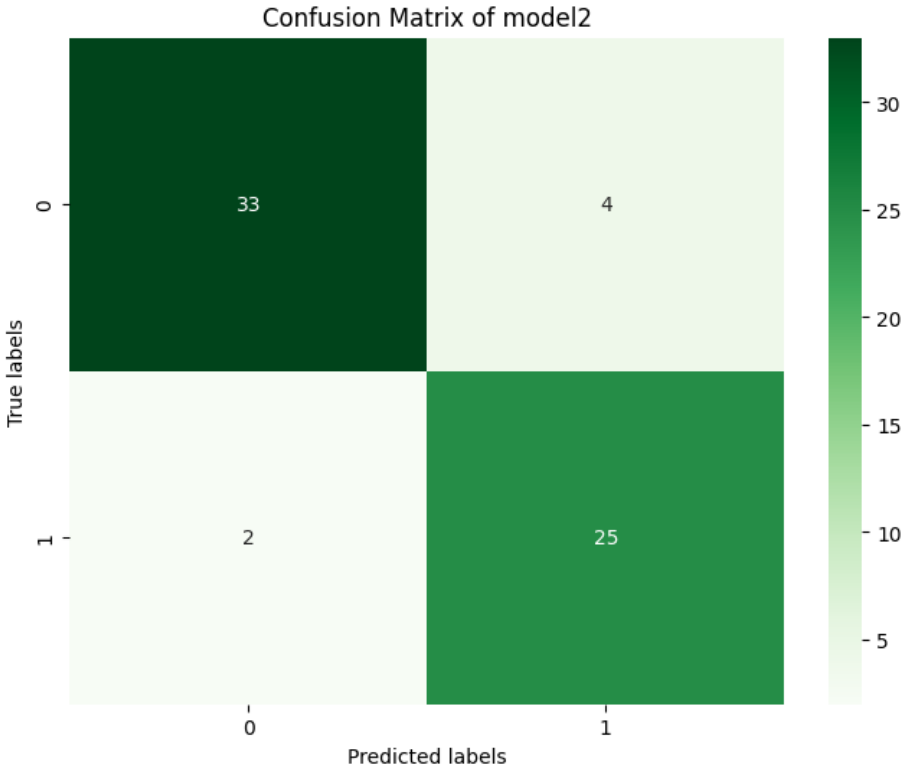

Figure 12: Confusion matrix of model 2

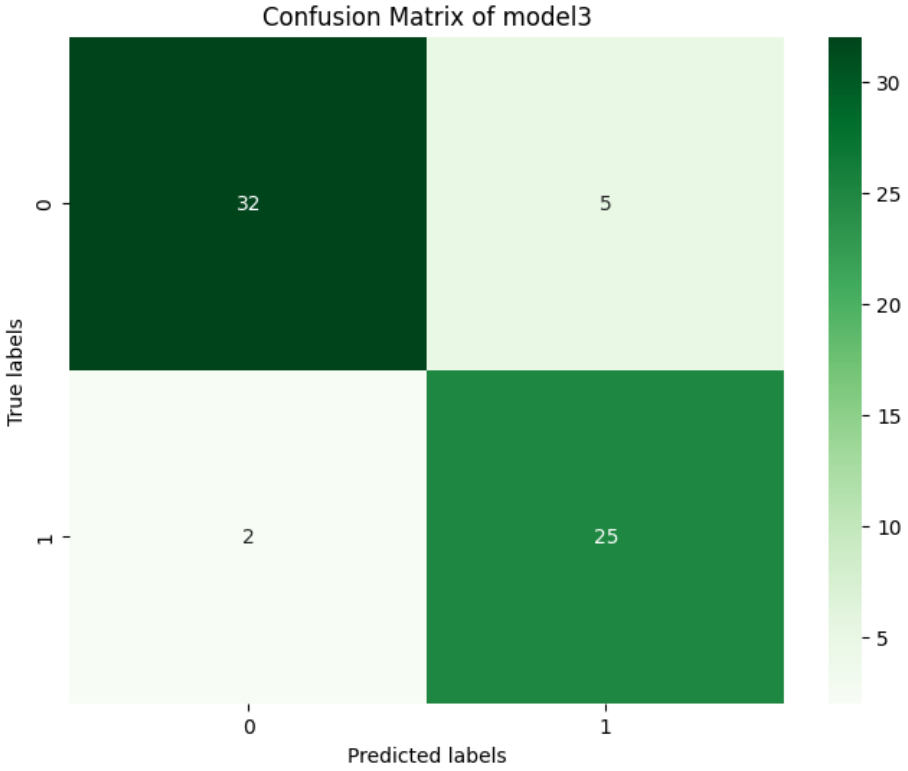

Figure 13: Confusion matrix of model 3

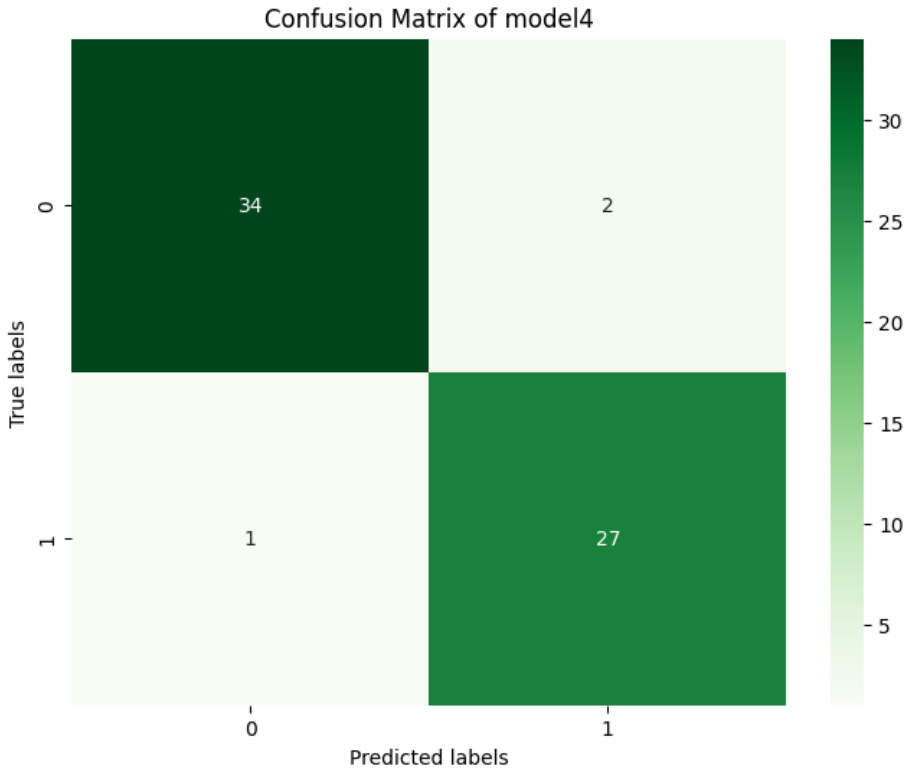

Figure 14: Confusion matrix of model 4

partition the dataset into several subsets, or "folds," and train the model on some of these folds while testing it on the remaining ones. This process is repeated multiple times, with each fold being used as the test set exactly once. The results from each iteration are then averaged to provide a robust estimate of the model's performance. This method helps in identifying overfitting, underfitting, and provides insights into the stability and reliability of the model.
When applying Convolutional Neural Networks (CNNs) in machine learning, cross validation becomes particularly crucial due to the typically large number of parameters and the risk of overfitting. CNNs, which are particularly effective for tasks involving image data, benefit from cross validation as it ensures the network's architecture and parameters are optimally tuned [8]. By partitioning the dataset into training and validation sets multiple times, cross validation provides a more accurate assessment of the CNN's ability to generalize across different data distributions. This process not only aids in hyperparameter tuning but also validates the model's robustness, ensuring that the CNN performs well not just on the training data, but also on new, unseen data.
In our case, we implemented 5-fold cross validation on the most effective model, Model4. For each fold, the validation dataset was assessed to measure the model's performance. Table 2 illustrates the validation accuracy and loss across the folds. After completing the cross-validation process, we tested the model with a separate test set, which had not been seen by the model during training. This final evaluation with the unseen data further confirmed that the model's performance had indeed improved which can be seen in Table 3.

4 Conclusion

In conclusion, our experimentation with four different model architectures, adjusting the number of convolutional and dense layers, demonstrated a clear trend of improvement in model performance.

Table 2: Folds Validation Accuracy and Loss

Folds	Accuracy	Loss
Fold 1	0.8988	0.2571
Fold 2	0.9283	0.1876
Fold 3	0.8862	0.3012
Fold 4	0.9494	0.1394
Fold 5	0.9715	0.9715
Mean	0.9268	0.1948

Table 3: Test Accuracy and Loss after CV

Test Accuracy	Test Loss
0.9527	0.1288

Gradually increasing the layers in each model resulted in better accuracy and reduced loss, concluding in the identification of the best-performing model. Applying 5-fold cross validation to this model further validated its effectiveness, showing continued improvement and robustness. Overall, these adjustments and the application of cross validation contributed to mitigating overfitting, thereby enhancing the CNN's ability to generalize well to unseen data.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[2] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 2019.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[6] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[8] Haofeng Zhang, Liang Wang, Yanyan Ma, Yang Liu, and Wei Wang. A review of recent advances in convolutional neural networks. *Neurocomputing*, 323:91–111, 2019.

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