Machine Learning Module

Final Project

Binary Classification of Muffins and Chihuahuas

"By Using Keras-Based Neural Networks"

Ashkan Samavatian

Matriculation number: 965235

(Student in Data Science and Economics)

Declaration:

"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."

1) Introduction

1-1) Project Overview

The current project focuses on using Keras to analyze the "Muffin vs Chihuahua" dataset, available on Kaggle, to classify images of Muffins and Chihuahuas using deep learning models.

Considering that the dataset is divided into the training and test sets from the beginning, and each set is categorized into the labels of "Muffin" and "Chihuahua", the methodology of the project contains the implementation of a 5-fold Cross-Validation technique on five different deep learning network architectures (respectively, The Custom Convolutional Neural Network (CNN), EfficientNet, MobileNet, ResNet50, and, DenseNet121 models) to compute risk estimates and applying fine-tuning hyperparameters to achieve optimal performances of the models and then applying the tuned models on the test set and evaluating the performance of the models using the Zero-One Loss, Accuracy, Precision, Recall, F1 score, Confusion Matrix, and AUC-ROC analysis by performing Python codes on the "Google Colab" service.

1-2) Literature (Background) Discussion

Binary image classification is one of the most important applications of deep learning. It involves categorizing images into one of two classes or categories. The goal is to develop a model that can automatically determine the class of an input image based on its features. The algorithm identifies these features, uses them to differentiate between different images, and assigns labels to them.

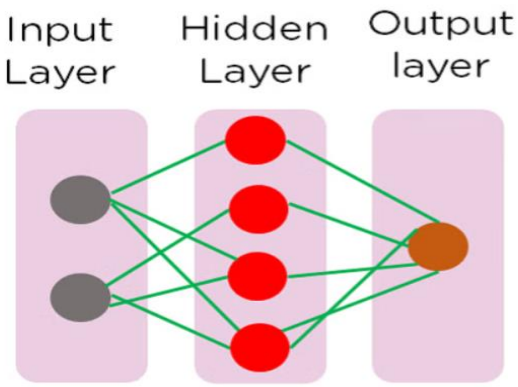


Figure 1-1: Neural Network

Image classification is done with the help of neural networks. They contain multiple layers of neurons that perform prediction, classification, etc. The output of each neuron is fed to the neurons in the next layer, which helps fine-tune the output until it reaches the final output layer. The different layers which are present in a neural network are:

Input Layer: This is the layer through which the input is given to the neural network.

Hidden Layer: This layer contains various neurons that process the input received from the input layer.

Output Layer: The final layer in the network that processes the data one last time and gives the output.

Deep Learning is a subset of Neural Networks that specifically refers to neural networks with many layers (hence the term "deep"). Deep learning models are designed to capture complex patterns in data. The architectures that were used in this project (Custom CNN, EfficientNet, MobileNet, ResNet50, DenseNet121) are all examples of deep neural networks and consist of layers of neurons (nodes) that process input data to perform classification tasks.

2) Goal of the Analysis and Methodology

2-1) Research Questions

The main questions in this project that should be addressed are:

1) Monitoring the effect of implementing five different deep learning architectures and hyperparameter tuning on the performance of image classification models in distinguishing between images of Muffins and Chihuahuas.
2) Using a 5-fold cross-validation to compute the risk estimate of each model.

2-2) Description of the Dataset

In this project, the "Muffin vs Chihuahua" dataset was used to perform training and testing procedures. This dataset is available on the Kaggle website and consists of 5917 images of Muffins and Chihuahuas. Each image is labeled as either a Muffin (1) or a Chihuahua (0). The dataset is divided into training and test sets with detailed information as below:

```
Found 4733 images belonging to 2 classes.
Found 1184 images belonging to 2 classes.
Training images:4733
Test images:1184
There are 640 images in muffin_vs_chihuahua_dataset/test/chihuahua
There are 544 images in muffin_vs_chihuahua_dataset/test/muffin
There are 2559 images in muffin_vs_chihuahua_dataset/train/chihuahua
There are 2174 images in muffin_vs_chihuahua_dataset/train/muffin
```

Figure 2-1: Details of the Dataset

```
chihuahua:0/number=2559
muffin:1/number=2174
```

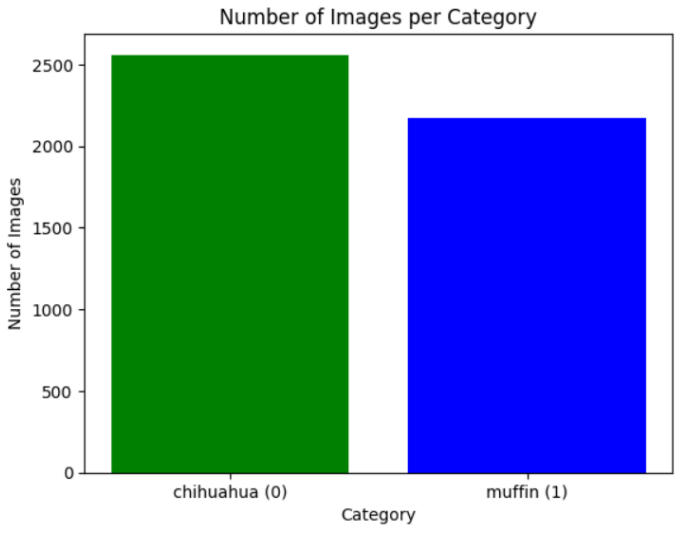


Chart 2-1: Distribution of the images in the training set

```
chihuahua:0/number=640
muffin:1/number=544
```

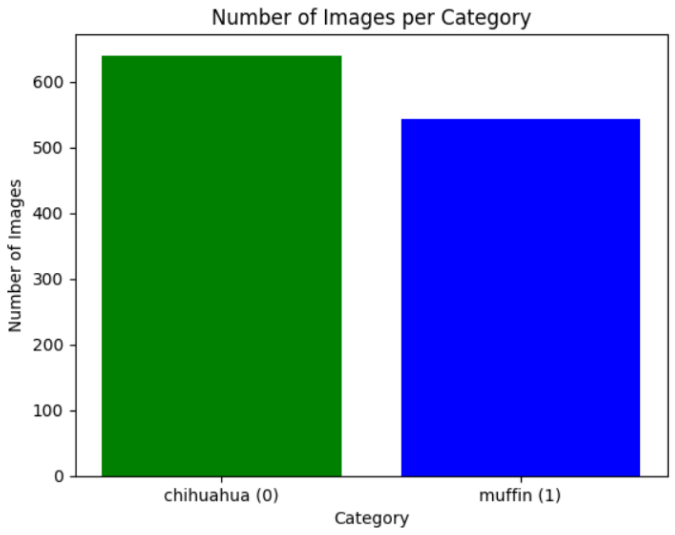


Chart 2-2: Distribution of the images in the test set

2-3) Methodology

Since the "Muffin vs Chihuahua" dataset is an image dataset and the project's goal is binary image classification, I decided to apply different deep learning network architectures to address the research question. To compute risk estimates, I performed the 5-fold Cross-Validation technique on five different deep learning models (respectively, The Custom Convolutional Neural Network (CNN), EfficientNet, MobileNet, ResNet50, and DenseNet121 models). Then, I applied hyperparameter tuning to achieve the models' optimal performance. Finally, the tuned models were applied to the test set, and I evaluated the performance of the models using the Zero-One Loss, Accuracy, Precision, Recall, F1 score, Confusion Matrix, and AUC-ROC analysis. To perform this project, I employed the following steps:

2-3-1) Data Acquisition, Handling, Exploration, and Preprocessing

After importing the necessary libraries for the project, preparing the environment for the deterministic behavior of the models, and setting the global seed to ensure the reproducibility of the results, I used my "Kaggle API" ("Kaggle username" and "Kaggle key") to download the dataset directly from Kaggle to the Google Colab service. The dataset's identifier is samuelcortinhas/muffin-vs-chihuahua-image-classification, and it was downloaded in a compressed format. Then, I unzipped the dataset in a directory with the name "muffin_vs_chihuahua_dataset" for the upcoming process.

After unzipping the dataset, I performed basic exploratory data analysis to understand its structure, such as the diversity within the training and test sets and their labels. I then defined and applied the "display_sample_images" function to monitor some sample images from both training and test sets.

Since the images had to be scaled down, I defined a function to find the minimum size of the images in each category. According to the results of this function, the image size equal to (128,128) was chosen for scaling down the images.

```
Minimum image size in muffin_vs_chihuahua_dataset/train/muffin is (102, 138)
Images with width<128: 1
Images with height<128: 0
Minimum image size in muffin_vs_chihuahua_dataset/train/chihuahua is (150, 129)
Images with width<128: 0
Images with height<128: 0
Minimum image size in muffin_vs_chihuahua_dataset/test/muffin is (181, 136)
Images with width<128: 0
Images with height<128: 0
Minimum image size in muffin_vs_chihuahua_dataset/test/chihuahua is (168, 138)
Images with width<128: 0
Images with height<128: 0
```

Figure 2-2: Results of the minimum image size finder function

For efficient computation and memory usage during the project, the batch size was assigned to 32, and all the images were transformed from JPG to RGB pixel values. Other data augmentation techniques, such as rotation, width/height shift, shear, zoom, and horizontal flip, were applied to the training set to enhance model generalization. After this step, directories for each category were defined to organize the training data.

The image file paths and corresponding labels were collected from the specified directories, and the labels were mapped using "class_indices" from the "train_generator". Then I defined a custom "load_images" function to load images from given paths, resize them to a target size (128x128), and convert the loaded images (in PIL format) into a NumPy array, which was essential because machine learning models typically require input data in NumPy array format.

After the data preprocessing steps, a stratified sampling approach was used to extract a representative 20% sample of the training data, to use in the training process of the models to reduce the computation time of the project in the iterations.

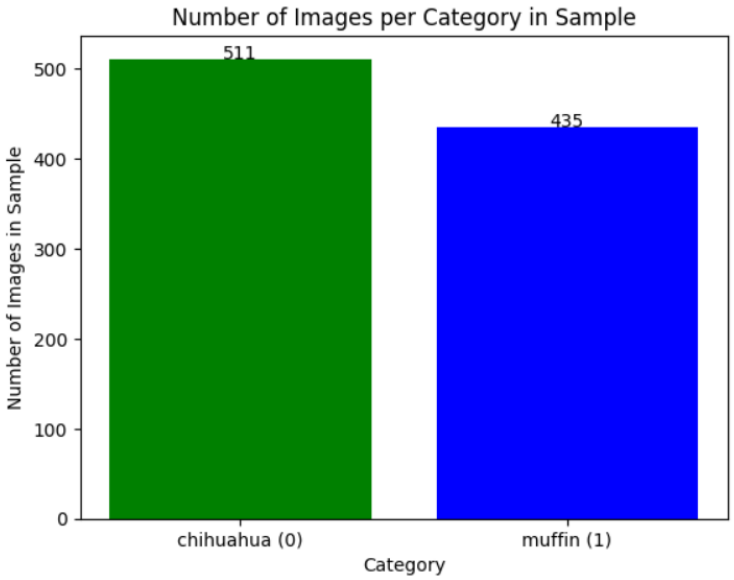


Chart 2-3: Distribution of the images in the training sample

2-3-2) Model Training and Validation

To ensure the robustness of the models and generalize well to unseen data, a 5-fold Cross-Validation technique was employed on five different deep learning models (respectively, The Custom Convolutional Neural Network (CNN), EfficientNet, MobileNet, ResNet50, and DenseNet121 models) to compute the risk estimates. An "EarlyStopping" callback was defined with "patience=3" to prevent overfitting and optimize the model performances. It monitored the validation loss parameter to ensure that the models improved not just on the training data but also on the unseen validation data. The "display_training_history" and "plot_training_history" functions were defined to display the "training and validation accuracies and losses tables", and the "training and validation accuracies and losses plots" for each model and each epoch. Then, I built and compiled the five different neural network architectures. Each model was constructed with specific layers and configurations to optimize their performance for the images' binary classification task. Finally, the cross-validated estimates were evaluated and reported based on the zero-one loss metric across multiple folds.

2-3-2-1) Convolutional Neural Network (CNN)

The CNN model consists of multiple convolutional layers followed by pooling, flattening, dense, and dropout layers. The model is designed to capture spatial features from images.

Layers: Convolutional, MaxPooling, Flatten, Dense, Dropout

Activation Function: ReLU for convolutional and dense layers due to its ability to mitigate the vanishing gradient problem and speed up convergence, Sigmoid for the output layer to provide a probability score for binary classification

Initialization: GlorotUniform initializer with a fixed seed for reproducibility

Optimizer: Adam for its adaptive learning rates and ability to handle sparse gradients and noisy data

Loss Function: Binary Cross-Entropy

Metrics: Accuracy

Layer parameters such as the number of filters, filter sizes, and pool sizes were selected to effectively capture and down-sample spatial features, while dense layers with ReLU and dropout help learn robust representations and prevent overfitting.

2-3-2-2) EfficientNet

EfficientNet is a pre-trained model on ImageNet that is known for its efficiency and performance. It improves accuracy and efficiency by combining depth, width, and resolution scaling.

Base Model: EfficientNetB0 pre-trained on ImageNet, with the top layers removed

Pooling Layer: Global Average Pooling to reduce the dimensionality of the feature maps while retaining spatial information in a compact form

Output Layer: Dense layer with Sigmoid activation to output a probability score for binary classification

Optimizer: Adam for its adaptive learning rate and robustness in handling different types of data

Loss Function: Binary Cross-Entropy

Metrics: Accuracy

The Base Model was frozen by "base_model.trainable=False" to ensure that the pre-trained weights were not updated, reducing computational load and preventing overfitting.

2-3-2-3) MobileNet

MobileNet is designed for mobile and embedded vision applications, providing efficient models with good accuracy.

Base Model: MobileNet pre-trained on ImageNet, with the top layers removed

Pooling Layer: Global Average Pooling to reduce the dimensionality of feature maps, helping to prevent overfitting while retaining essential spatial information

Dropout Layers: Dropout layers to introduce non-linearity and prevent overfitting

Output Layer: Dense layer with Sigmoid activation to provide a probability score for binary classification

Optimizer: Adam for its adaptive learning rate and robustness in handling different types of data

Loss Function: Binary Cross-Entropy

Metrics: Accuracy

Freezing the base model Kept the pre-trained weights fixed to leverage pre-learned features while training new layers specific to the classification task.

2-3-2-4) ResNet50

ResNet50 is a residual network that introduces skip connections to solve the vanishing gradient problem, allowing deeper networks to be trained.

Base Model: ResNet50, a deep residual network with 50 layers, leveraging pre-trained weights on ImageNet to provide a strong feature extraction foundation, with the top layers removed

Pooling Layer: Global Average Pooling to reduce the dimensionality of feature maps, helping to prevent overfitting while retaining essential spatial information

Dense Layers: Additional dense layers with ReLU activation to introduce non-linearity before the output layer

Output Layer: Dense layer with Sigmoid activation to provide a probability score for binary classification

Optimizer: Adam for its adaptive learning rate and robustness in handling different types of data

Loss Function: Binary Cross-Entropy

Metrics: Accuracy

The frozen base model prevented the pre-trained weights from being updated during the training.

2-3-2-5) DenseNet121

DenseNet121 connects each layer to every other layer in a feed-forward fashion, which improves the flow of information and gradients throughout the network.

Base Model: DenseNet121, a densely connected convolutional network with 121 layers, leveraging pre-trained weights on ImageNet to provide a strong feature extraction foundation, with the top layers removed

Pooling Layer: Global Average Pooling to reduce the dimensionality of feature maps, helping to prevent overfitting while retaining essential spatial information

Dense Layers: Fully connected layers with ReLU activation to introduce non-linearity before the output layer

Output Layer: Dense layer with Sigmoid activation to provide a probability score for binary classification

Optimizer: Adam for its adaptive learning rate and robustness in handling different types of data

Loss Function: Binary Cross-Entropy

Metrics: Accuracy

2-3-3) Hyperparameter Tuning

Hyperparameter tuning was performed to optimize each model's performance. The parameters tuned include the number of filters, kernel size, optimizer (for the CNN model), dense units, and learning rate (for the other four models). The tuning process involved training the models with different combinations of these hyperparameters and evaluating their performances on the validation set to identify the best configurations that yield the highest validation accuracies and lowest zero-one losses for all models. By iterating over various combinations of the parameters, the models were optimized for their performances. The use of early stopping ensured that the models did not overfit the training data, and the best parameters were selected based on validation performance.

3) Experimental Results

3-1) Training and Validation Results

3-1-1) Evaluation Metrics

3-1-1-1) Training and Validation Accuracy

Accuracy (either for training or validation) is the proportion of correctly classified samples. (both for training and validation) During the training and validation, the model evaluates its performance on the data after each epoch. The number of correct predictions is divided by the total number of samples to get accuracy. (both for training and validation)

$$\text{Training Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Training Samples}} \qquad \text{Validation Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Validation Samples}}$$

Figure 3-1: Formulas of Training and Validation Accuracy

3-1-1-2) Training and Validation loss

The loss or error (either for training or validation) indicates the model's performance. The loss (both for training and validation) is computed using the specified loss function (in this project, binary cross-entropy) after each batch and averaged over all batches in an epoch.

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

Figure 3-2: Formulas of Training and Validation Loss

3-1-1-3) Zero-One Loss

Zero-one loss is the proportion of misclassified samples in the validation data. After predictions are made on the validation data, the zero-one loss is calculated as the fraction of incorrectly predicted samples.

$$\text{Zero-One Loss} = \frac{\text{Number of Incorrect Predictions}}{\text{Total Validation Samples}}$$

Figure 3-3: Formulas of Zero-One Loss

3-1-2) CNN Training and Validation Results

Training and Validation Metrics for CNN Model

	Fold	Epoch	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0	1	1	0.578042	0.610526	18.755264	0.640257
1	1	2	0.665344	0.642105	0.649750	0.748166
2	1	3	0.750000	0.715789	0.570066	0.612658
3	1	4	0.779101	0.715789	0.505038	0.756236
4	1	5	0.767196	0.663158	0.565997	0.732838
5	2	1	0.601057	0.814815	23.151430	0.520502
6	2	2	0.686922	0.783069	0.626951	0.510663
7	2	3	0.747688	0.687831	0.522331	0.588057
8	2	4	0.702774	0.730159	0.602565	0.511097
9	2	5	0.780713	0.798942	0.471509	0.429013
10	3	1	0.593131	0.703704	16.258696	0.608642
11	3	2	0.741083	0.751323	0.565836	0.534173
12	3	3	0.762219	0.677249	0.569736	0.592805
13	3	4	0.828269	0.740741	0.413172	0.565772
14	3	5	0.887715	0.761905	0.313959	0.540370
15	4	1	0.606341	0.682540	13.339339	0.657473
16	4	2	0.700132	0.666667	0.591917	0.624456
17	4	3	0.796565	0.724868	0.494192	0.645400
18	4	4	0.774108	0.740741	0.563312	0.679390
19	4	5	0.796565	0.793651	0.503078	0.653175
20	5	1	0.632761	0.714286	9.877760	0.615180
21	5	2	0.758256	0.767196	0.550866	0.536511
22	5	3	0.782034	0.735450	0.477808	0.564921
23	5	4	0.871863	0.820106	0.325275	0.450447
24	5	5	0.865258	0.846561	0.307256	0.462800

Figure 3-4: CNN Training and Validation Accuracies and Losses

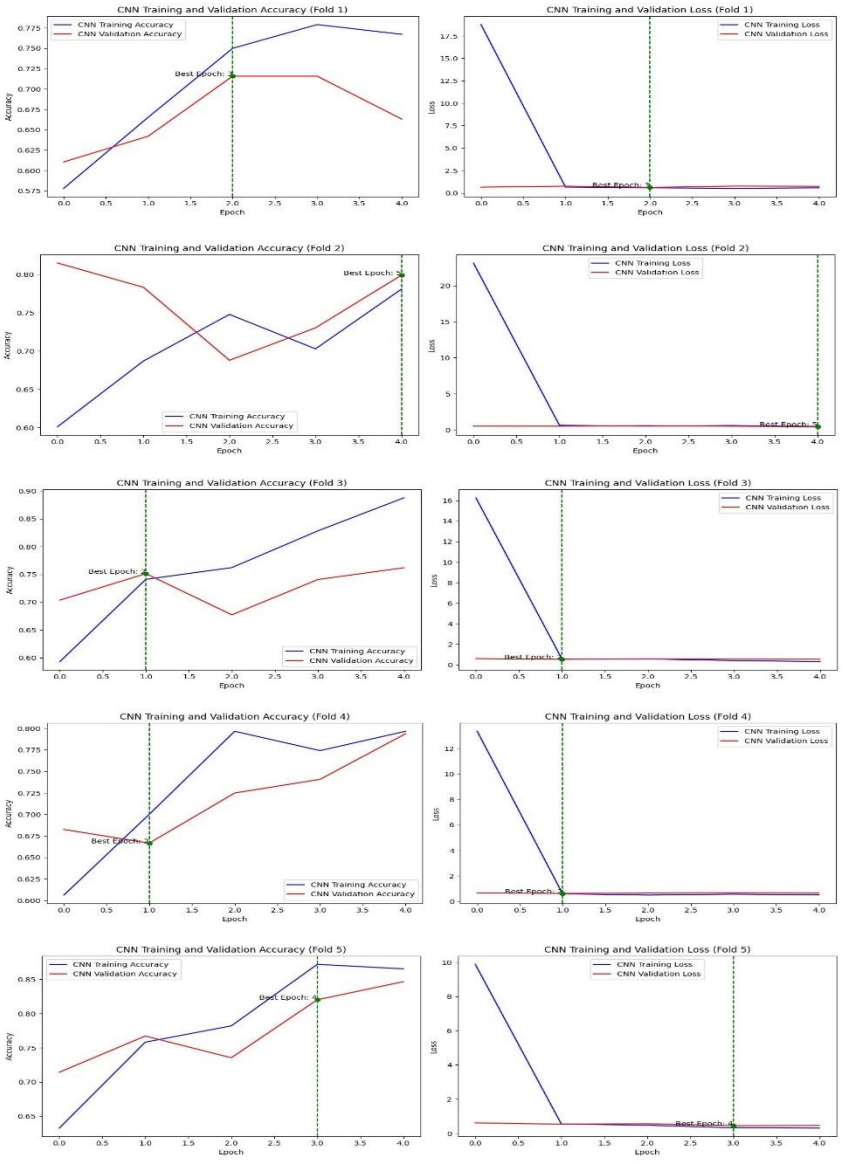


Chart 3-1: CNN Training and Validation Accuracies and Losses

The results show improvement in training accuracy across all folds. However, the validation accuracy fluctuated, indicating potential overfitting in some folds.

The significant drop in training loss within the first epoch suggests that the model learned quickly. While also decreasing, the validation loss did not match the training loss's decline, highlighting overfitting concerns.

N.B: The best Epoch in the charts is the point with the minimum validation loss among all the epochs. (For all the models)

3-1-3) EfficientNet Training and Validation Results

Training and Validation Metrics for EfficientNet Model

	Fold	Epoch	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0	1	1	0.883598	0.963158	0.376544	0.212451
1	1	2	0.969577	0.978947	0.167376	0.124893
2	1	3	0.976190	0.978947	0.107806	0.096978
3	1	4	0.984127	0.978947	0.087885	0.083012
4	1	5	0.990741	0.978947	0.074416	0.073696
5	2	1	0.900925	0.968254	0.384845	0.207897
6	2	2	0.973580	0.978836	0.166208	0.121170
7	2	3	0.981506	0.994709	0.112620	0.091677
8	2	4	0.986790	0.994709	0.088485	0.075524
9	2	5	0.986790	0.994709	0.079976	0.065663
10	3	1	0.907530	0.915344	0.366897	0.259736
11	3	2	0.982827	0.936508	0.151264	0.184321
12	3	3	0.986790	0.952381	0.097050	0.157017
13	3	4	0.990753	0.947090	0.072998	0.142954
14	3	5	0.992074	0.957672	0.065957	0.134378
15	4	1	0.869221	0.984127	0.385924	0.186369
16	4	2	0.968296	0.994709	0.169830	0.104148
17	4	3	0.980185	0.994709	0.116463	0.075883
18	4	4	0.981506	0.994709	0.094851	0.061525
19	4	5	0.982827	0.994709	0.079498	0.052464
20	5	1	0.886394	0.978836	0.384702	0.193717
21	5	2	0.973580	0.973545	0.165960	0.116063
22	5	3	0.978864	0.978836	0.115906	0.090313
23	5	4	0.989432	0.978836	0.088076	0.075913
24	5	5	0.989432	0.978836	0.075003	0.067389

Figure 3-5: EfficientNet Training and Validation Accuracies and Losses

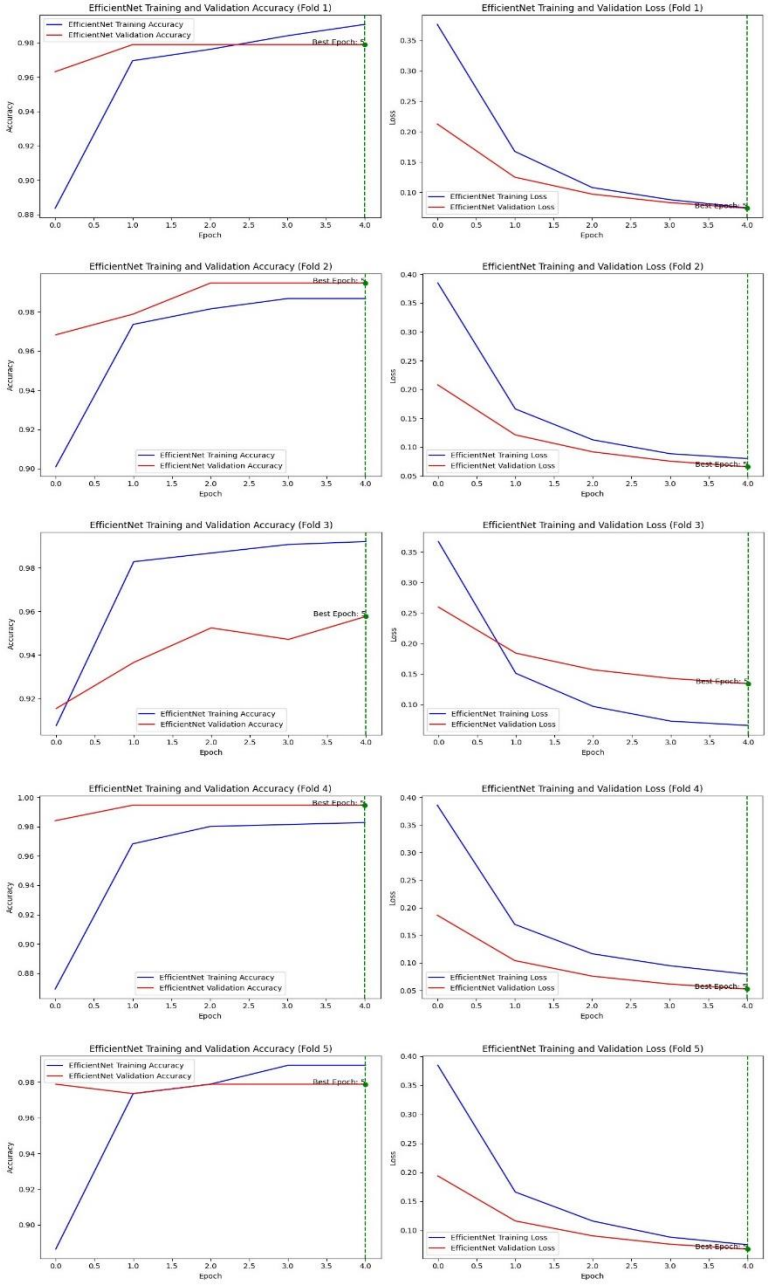


Chart 3-2: EfficientNet Training and Validation Accuracies and Losses

The results show that training accuracy consistently improved over epochs, indicating the model's learning capability. Validation accuracy also showed improvement, though it slightly fluctuated due to the validation set's variability. The model achieved high accuracy early, stabilizing around 98-99%.

Training loss decreased steadily, demonstrating effective learning. Validation loss also decreased, suggesting that the model generalized well to unseen data. Low validation loss values (around 0.05 to 0.26) indicate good model performance.

3-1-4) MobileNet Training and Validation Results

Training and Validation Metrics for MobileNet Model

	Fold	Epoch	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0	1	1	0.706349	0.821053	0.743521	0.375908
1	1	2	0.792328	0.868421	0.480388	0.312372
2	1	3	0.808201	0.836842	0.448700	0.346653
3	1	4	0.839947	0.821053	0.398179	0.365583
4	1	5	0.828042	0.847368	0.381293	0.326723
5	2	1	0.647292	0.788360	0.795633	0.410354
6	2	2	0.776750	0.846561	0.485703	0.338209
7	2	3	0.820343	0.825397	0.414263	0.297475
8	2	4	0.820343	0.857143	0.372346	0.292792
9	2	5	0.825628	0.862434	0.367097	0.284647
10	3	1	0.696169	0.761905	0.682073	0.522297
11	3	2	0.811096	0.798942	0.466932	0.446313
12	3	3	0.833553	0.793651	0.417184	0.526168
13	3	4	0.824306	0.804233	0.406718	0.389699
14	3	5	0.830912	0.820106	0.351037	0.386926
15	4	1	0.684280	0.878307	0.762509	0.303361
16	4	2	0.793923	0.862434	0.497310	0.309592
17	4	3	0.819022	0.894180	0.422584	0.274548
18	4	4	0.819022	0.883598	0.398095	0.282810
19	4	5	0.830912	0.862434	0.371366	0.298195
20	5	1	0.734478	0.846561	0.661051	0.364071
21	5	2	0.800528	0.862434	0.485142	0.294913
22	5	3	0.821664	0.873016	0.431832	0.295342
23	5	4	0.837516	0.878307	0.367022	0.295115
24	5	5	0.842801	0.878307	0.366729	0.293649

Figure 3-6: MobileNet Training and Validation Accuracies and Losses

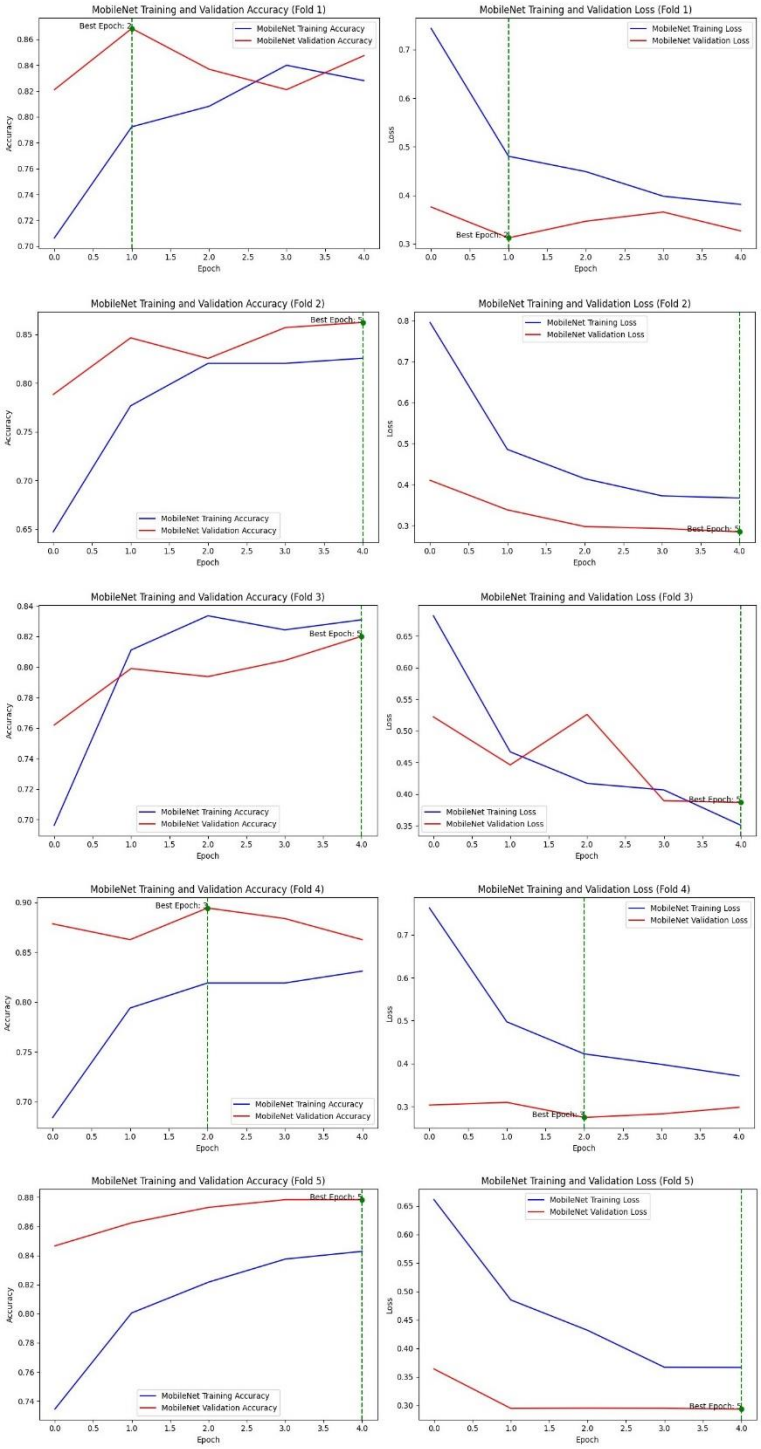


Chart 3-3: MobileNet Training and Validation Accuracies and Losses

The MobileNet model's training accuracy generally showed an upward trend, indicating that the model was learning from the data. The validation accuracy also showed an upward trend, although it varied more

significantly than the training accuracy. This variation was expected due to the different validation datasets in each fold.

Both the training and validation losses decreased over the epochs, which was a good indication of the model's learning process. The validation loss generally mirrored the training loss, although it could sometimes be higher due to overfitting or data variability.

3-1-5) ResNet50 Training and Validation Results

Training and Validation Metrics for ResNet50 Model

	Fold	Epoch	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0	1	1	0.900794	0.952632	0.285317	0.233791
1	1	2	0.970899	0.963158	0.075017	0.136475
2	1	3	0.993386	0.957895	0.020050	0.172347
3	1	4	0.997355	0.957895	0.010287	0.125751
4	1	5	0.998677	0.963158	0.008074	0.156327
5	2	1	0.927345	0.962963	0.205156	0.100412
6	2	2	0.986790	0.962963	0.031864	0.095104
7	2	3	1.000000	0.968254	0.007621	0.098426
8	2	4	1.000000	0.973545	0.004417	0.098079
9	2	5	1.000000	0.973545	0.002740	0.093113
10	3	1	0.879789	0.931217	0.344432	0.250569
11	3	2	0.972259	0.936508	0.066563	0.194135
12	3	3	0.988111	0.936508	0.030881	0.180867
13	3	4	0.997358	0.931217	0.015337	0.190570
14	3	5	1.000000	0.941799	0.006150	0.187869
15	4	1	0.912814	0.994709	0.233982	0.038714
16	4	2	0.973580	0.962963	0.070078	0.073422
17	4	3	0.992074	0.962963	0.025505	0.064998
18	4	4	0.996037	0.989418	0.013679	0.036239
19	4	5	1.000000	0.994709	0.006235	0.031894
20	5	1	0.908851	0.957672	0.254892	0.069745
21	5	2	0.989432	0.973545	0.032621	0.060234
22	5	3	0.998679	0.968254	0.010009	0.055595
23	5	4	0.998679	0.973545	0.005797	0.056083
24	5	5	1.000000	0.973545	0.003546	0.070163

Figure 3-7: ResNet50 Training and Validation Accuracies and Losses

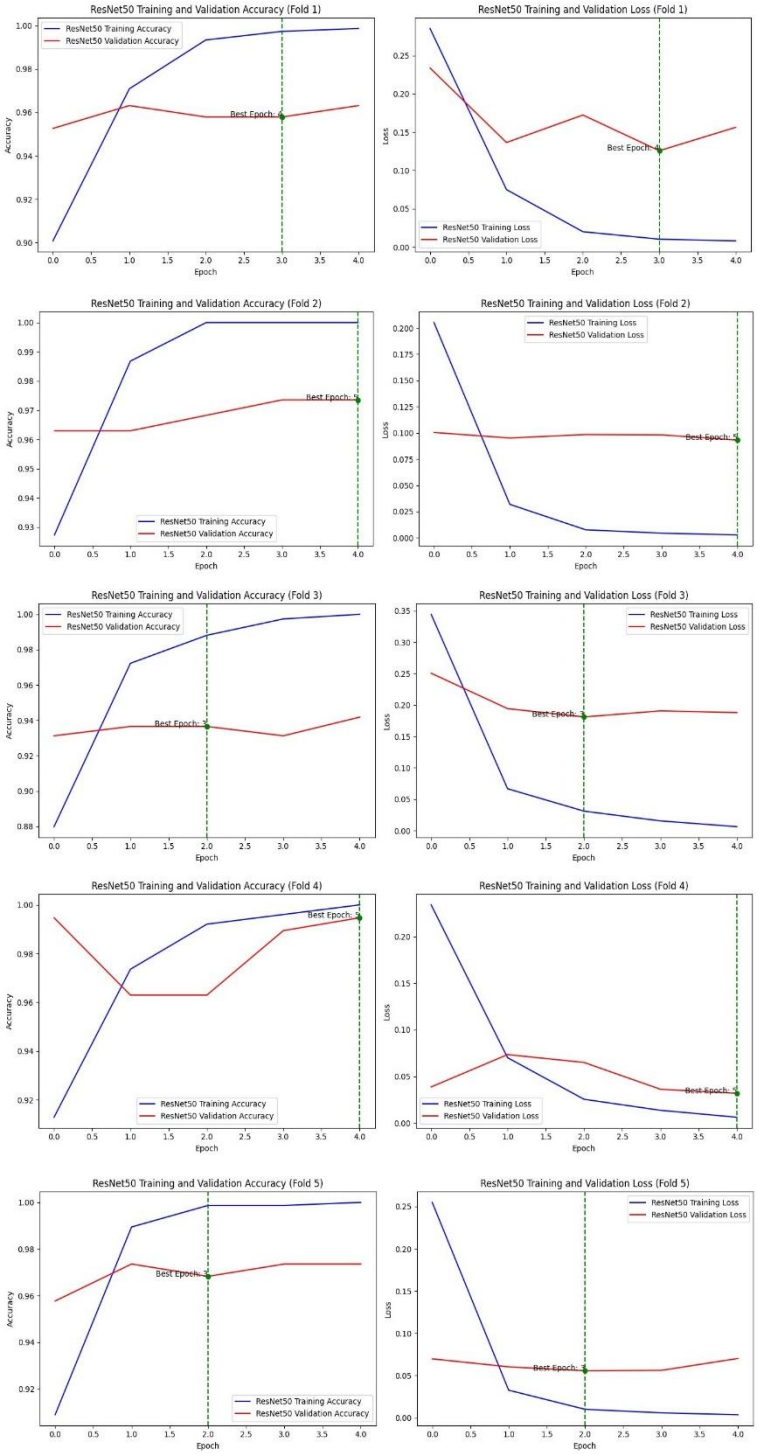


Chart 3-4: ResNet50 Training and Validation Accuracies and Losses

According to the results, the training accuracy increased steadily across epochs, often reaching very high values close to one. However, the validation accuracy did not show the same level of improvement, indicating potential overfitting.

The training loss decreased significantly, demonstrating effective learning during training. While lower than the training loss, the validation loss did not decrease as smoothly, further indicating overfitting.

3-1-6) DenseNet121 Training and Validation Results

Training and Validation Metrics for DenseNet Model

	Fold	Epoch	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0	1	1	0.723545	0.778947	1.978857	1.099150
1	1	2	0.855820	0.826316	0.458215	0.936152
2	1	3	0.886243	0.836842	0.333482	0.798172
3	1	4	0.929894	0.800000	0.165039	0.955569
4	1	5	0.932540	0.831579	0.154258	0.776161
5	2	1	0.701453	0.756614	1.889802	0.857892
6	2	2	0.845443	0.740741	0.670128	0.861347
7	2	3	0.859974	0.857143	0.449670	0.360067
8	2	4	0.922061	0.851852	0.210756	0.394784
9	2	5	0.948481	0.883598	0.148098	0.365252
10	3	1	0.721268	0.841270	1.255154	0.573921
11	3	2	0.854690	0.777778	0.624234	0.646508
12	3	3	0.899604	0.841270	0.309707	0.420567
13	3	4	0.906209	0.841270	0.239797	0.672941
14	3	5	0.932629	0.851852	0.191342	0.456724
15	4	1	0.762219	0.798942	1.188124	0.602548
16	4	2	0.861295	0.867725	0.485363	0.526285
17	4	3	0.875826	0.798942	0.448756	0.775372
18	4	4	0.892999	0.894180	0.329187	0.295888
19	4	5	0.924703	0.888889	0.220114	0.351049
20	5	1	0.745046	0.867725	1.162798	0.595770
21	5	2	0.829590	0.894180	0.688948	0.563776
22	5	3	0.867900	0.899471	0.404262	0.368045
23	5	4	0.927345	0.899471	0.218232	0.411917
24	5	5	0.947160	0.894180	0.126095	0.329317

Figure 3-8: DenseNet121 Training and Validation Accuracies and Losses

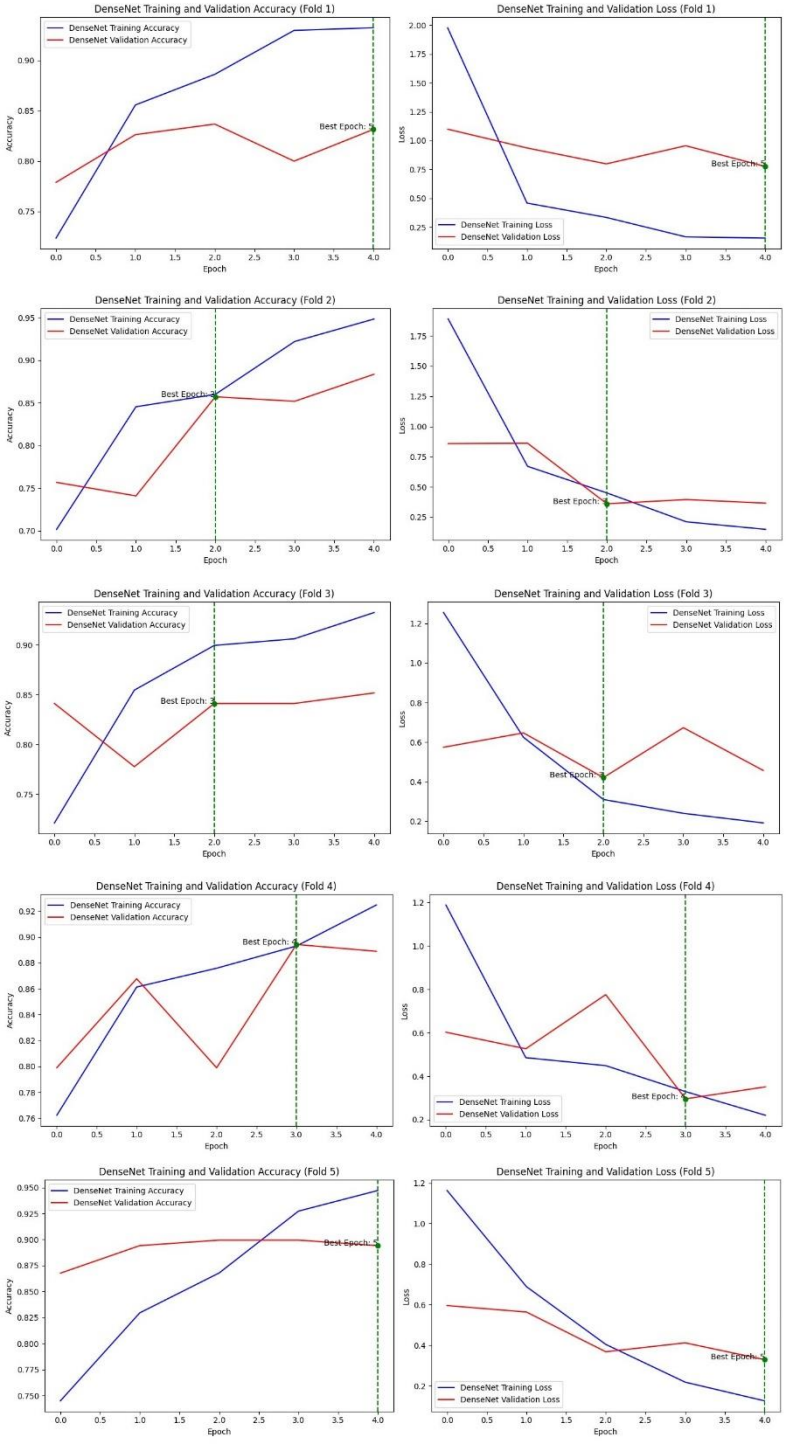


Chart 3-5: DenseNet121 Training and Validation Accuracies and Losses

The results illustrate how the model's performance varied across folds in the training and validation accuracies. Training accuracy generally improved with epochs, while validation accuracy fluctuated, indicating the balance between learning and overfitting.

The results highlight the reduction in training loss over epochs. Validation loss plots show a decline followed by a rise, suggesting overfitting.

3-1-7) Comparison of Models' Training and Validation Performance

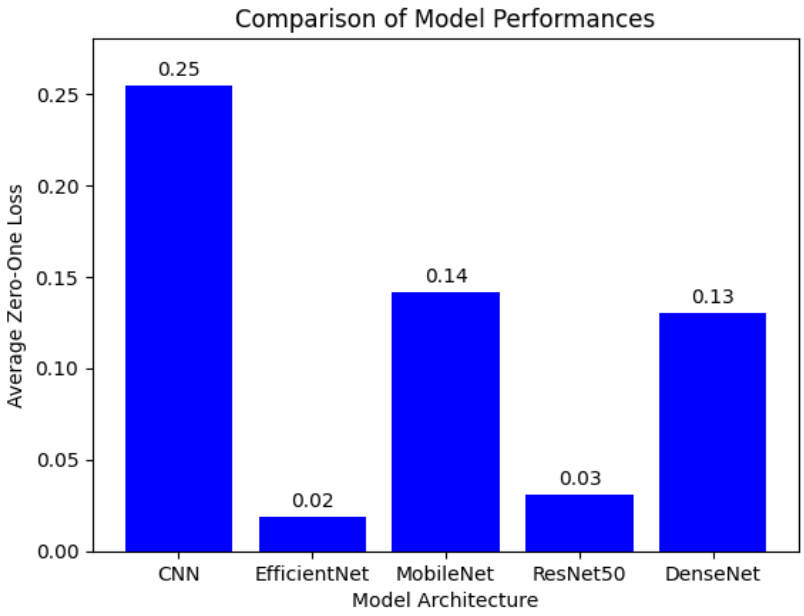


Chart 3-6: Comparison of Models' Training and Validation Performances

According to the results, the EfficientNet model was the top-performing model for the "Muffin vs Chihuahua" classification task, with the lowest zero-one loss value of around 0.02. On the other hand, the CNN model had the highest misclassification rate with a zero-one loss of around 0.25.

3-2) Hyperparameter Tuning Results

Validation accuracy and corresponding zero-one loss were used as evaluation metrics for tuning the models' hyperparameters. The tuned parameters, their corresponding configurations, and the achieved results are presented below:

3-2-1) CNN Hyperparameters Tuning Results

Parameters and Configurations: {"num_filters": 32, "kernel_size": 3, "optimizer": "adam"}

{"num_filters": 64, "kernel_size": 3, "optimizer": "adam"}

{"num_filters": 32, "kernel_size": 5, "optimizer": "sgd"}

{"num_filters": 64, "kernel_size": 5, "optimizer": "sgd"}

Best CNN Parameters: {“num_filters”: 32, “kernel_size”: 3, “optimizer”: “adam”}

Best Validation Accuracy: 0.7460317611694336

Corresponding Zero-One Loss: 0.25396825396825395

3-2-2) EfficientNet Hyperparameters Tuning Results

Parameters and Configurations: {“dense_units”: 128, “learning_rate”: 0.001}

{“dense_units”: 256, “learning_rate”: 0.001}

{“dense_units”: 128, “learning_rate”: 0.01}

{“dense_units”: 256, “learning_rate”: 0.01}

Best EfficientNet Parameters: {“dense_units”: 256, “learning_rate”: 0.001}

Best Validation Accuracy: 0.9894179701805115

Corresponding Zero-One Loss: 0.010582010582010581

3-2-3) MobileNet Hyperparameters Tuning Results

Parameters and Configurations: {“dense_units”: 128, “learning_rate”: 0.001}

{“dense_units”: 256, “learning_rate”: 0.001}

{“dense_units”: 128, “learning_rate”: 0.01}

{“dense_units”: 256, “learning_rate”: 0.01}

Best MobileNet Parameters: {“dense_units”: 256, “learning_rate”: 0.01}

Best Validation Accuracy: 0.8730158805847168

Corresponding Zero-One Loss: 0.12698412698412698

3-2-4) ResNet50 Hyperparameters Tuning Results

Parameters and Configurations: {“dense_units”: 128, “learning_rate”: 0.001}

{“dense_units”: 256, “learning_rate”: 0.001}

{“dense_units”: 128, “learning_rate”: 0.01}

{“dense_units”: 256, “learning_rate”: 0.01}

Best ResNet50 Parameters: {“dense_units”: 128, “learning_rate”: 0.001}

Best Validation Accuracy: 0.9735449552536011

Corresponding Zero-One Loss: 0.026455026455026454

3-2-5) DenseNet121 Hyperparameters Tuning Results

Parameters and Configurations: {"dense_units": 128, "learning_rate": 0.001}

{"dense_units": 256, "learning_rate": 0.001}

{"dense_units": 128, "learning_rate": 0.01}

{"dense_units": 256, "learning_rate": 0.01}

Best DenseNet121 Parameters: {"dense_units": 256, "learning_rate": 0.01}

Best Validation Accuracy: 0.8941798806190491

Corresponding Zero-One Loss: 0.10582010582010581

3-2-6) Comparison of Models' Performances Before and After Tuning

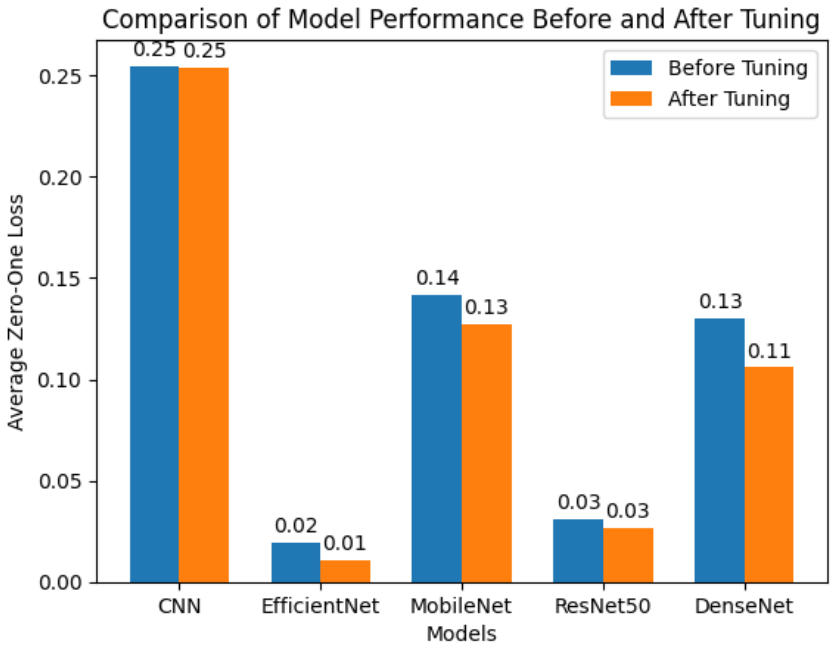


Chart 3-7: Comparison of Models' Performances Before and After Tuning

The results show that hyperparameter tuning had varying degrees of impact on different models. While EfficientNet, MobileNet, and DenseNet121 models benefited slightly from tuning, the CNN and ResNet50 models showed no significant changes. The EfficientNet model had the best performance, with the lowest zero-one loss value around 0.01 after hyperparameter tuning.

3-3) Prediction Results

For the evaluation phase with the test set, the test image file paths and corresponding test labels were collected from the specified directories, and the test labels were mapped using "class_indices" from the "test_generator". Then I defined a custom "load_test_images" function to load test images from given paths,

resize them to a target size (128x128), and convert the loaded images (in PIL format) into a NumPy array, which was essential because machine learning models typically require input data in NumPy array format.

After these steps, a stratified sampling approach was used to extract a representative 20% sample of the test data to use in the evaluation process of the models to reduce the computation time of the project in the iterations.

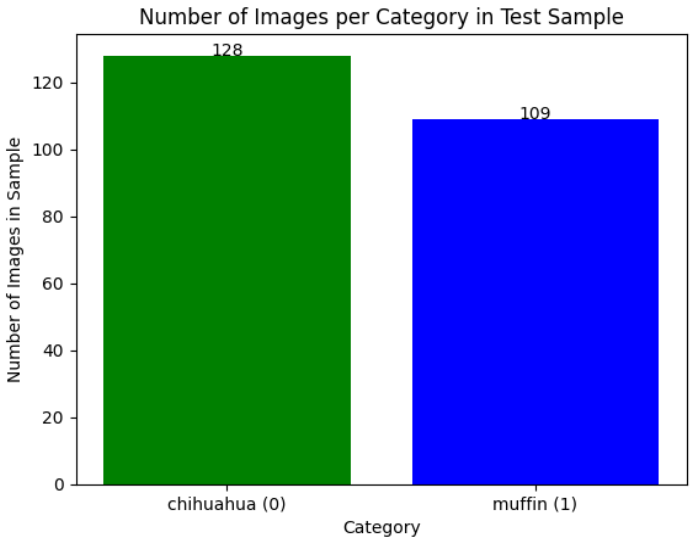


Chart 3-8: Distribution of the images in the test sample

To evaluate the models with the unseen data, I built and trained models with the best parameters of each model, gained from the hyperparameter tuning process. The "evaluate_model", "plot_confusion_matrix", and "plot_roc_curve" functions were defined to display the "Classification Report", "Confusion Matrix", and "ROC Curve". Then, the models' performances were evaluated on the test sample, and the evaluation results were visualized to provide a comprehensive understanding of each model's performance.

3-3-1) Evaluation Metrics for the prediction results

After performing the prediction phase, the models' performances were evaluated based on the "Test Loss", "Test Accuracy", the corresponding "Zero_One Loss", "Precision", "Recall", "F1-Score", "AUC-ROC", and "Confusion Matrix".

3-3-1-1) Test loss

The loss or error indicates the model's performance. It is computed using the specified loss function (in this project, binary cross-entropy)

3-3-1-2) Test Accuracy

Accuracy is the proportion of correctly classified samples. To calculate accuracy, the number of correct predictions is divided by the total number of samples.

3-3-1-3) Zero-One Loss

Zero-one loss is the proportion of misclassified samples in the test data. After predictions are made on the test data, the zero-one loss is calculated as the fraction of incorrectly predicted samples.

N.B: The formulas for "Test Loss", "Test Accuracy", and "Zero_One Loss" are calculated similarly to "Training and Validation Loss", "Training and Validation Accuracy", and the corresponding "Zero-One Loss".

3-3-1-4) Precision, Recall, F1-Score (Classification Report)

In the Classification Report, we have three different metrics that are explained below:

Precision is the ratio of correctly predicted positive observations to the total predicted positives. $\text{Precision} = \frac{TP}{TP + FP}$

Recall is the ratio of correctly predicted positive observations to all observations in the actual class. $\text{Recall} = \frac{TP}{TP + FN}$

F1-Score is the weighted average of precision and recall. $\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

In fact, the F1-score tries to find the balance between precision and recall. We then have the metrics' Macro Average and Weighted Average, which show each metric's average respectively without and considering each class's support. Thus, these two items present an insight into the data balance.

3-3-1-5) Confusion Matrix

Confusion Matrix is a table that is used to describe the performance of a classification model on a set of data for which the true values are known. Confusion Matrix has four different parts as below:

True Negative (TN): The actual is negative, and the prediction is also negative.

True Positive (TP): The actual is positive, and the prediction is also positive.

False Positive (FP): The actual is negative, but the prediction is positive. (Type 1 error)

False Negative (FN): The actual is positive, but the prediction is negative. (Type 2 error)

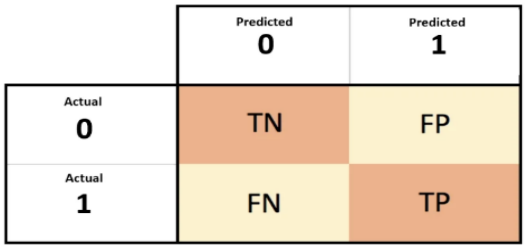


Figure 3-9: Confusion Matrix

3-3-1-6) AUC-ROC (Area Under Curve – Receiver Operating Characteristic) Analysis

AUC-ROC, or the Area Under the ROC Curve, provides a measure of how well the model can correctly classify instances with the positive label (label=1) while minimizing false positives. The AUC-ROC is a

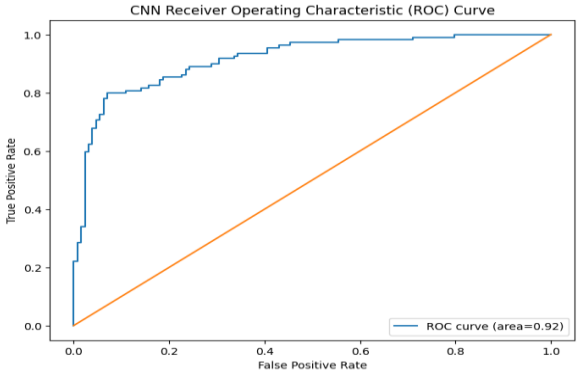
threshold-independent measure. It is a single scalar value that summarizes the performance of the binary classifiers. A value of 1 represents a perfect classifier, while a value of 0.5 represents a random classifier.

3-3-2) CNN Prediction Results

```
CNN Accuracy: 0.8143
CNN ROC-AUC: 0.9163
CNN Classification Report:
              precision    recall  f1-score   support

Chihuahua(0)       0.86      0.78      0.82       128
   Muffin(1)       0.77      0.85      0.81       109

    accuracy                           0.81       237
   macro avg       0.82      0.82      0.81       237
weighted avg       0.82      0.81      0.81       237
```




```
CNN Test Loss: 0.3900853395462036
CNN Test Accuracy: 0.8143460154533386
CNN Zero-One Loss on Test Set: 0.18565400843881857
```

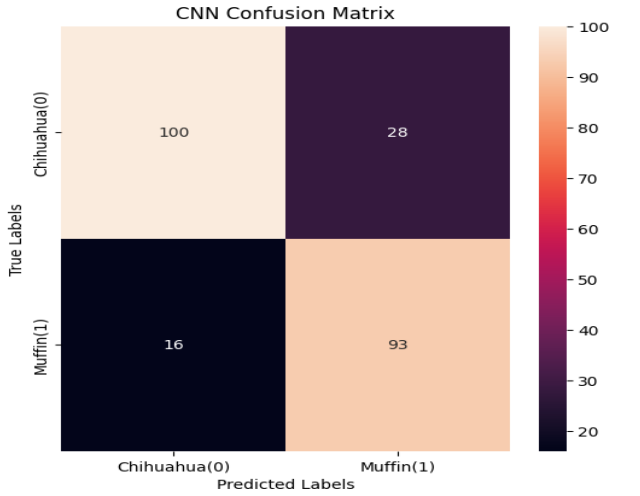


Figure 3-10: CNN Model Prediction Results

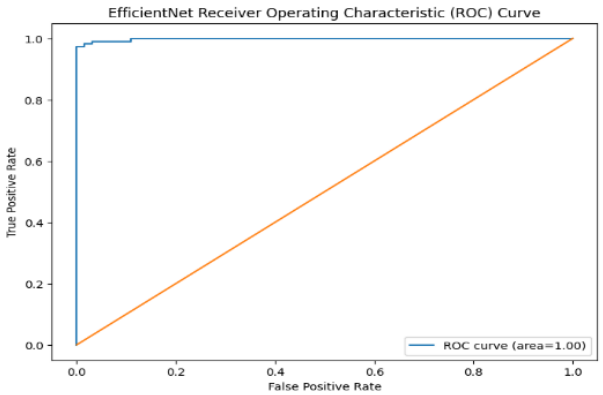
The CNN model performs strongly on the test set, with an overall accuracy of 81.43% and the ROC-AUC score of 91.63%. The precision and recall metrics suggest that the model effectively balances false positives and false negatives. The confusion matrix reveals that the model better identifies "Chihuahua" than "Muffin" images.

3-3-3) EfficientNet Prediction Results

```
EfficientNet Accuracy: 0.9873
EfficientNet ROC-AUC: 0.9986
EfficientNet Classification Report:
              precision    recall  f1-score   support

Chihuahua(0)       0.98      1.00      0.99       128
   Muffin(1)       1.00      0.97      0.99       109

    accuracy                           0.99       237
   macro avg       0.99      0.99      0.99       237
weighted avg       0.99      0.99      0.99       237
```




```
EfficientNet Test Loss: 0.04049794003367424
EfficientNet Test Accuracy: 0.9873417615890503
EfficientNet Zero-One Loss on Test Set: 0.012658227848101266
```

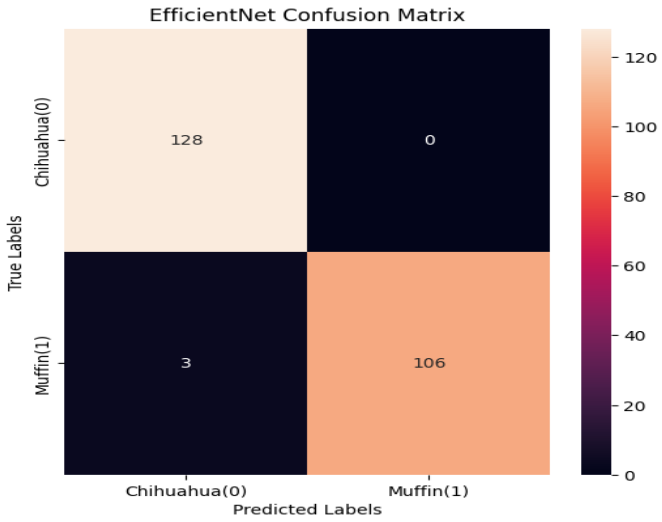


Figure 3-11: EfficentNet Model Prediction Results

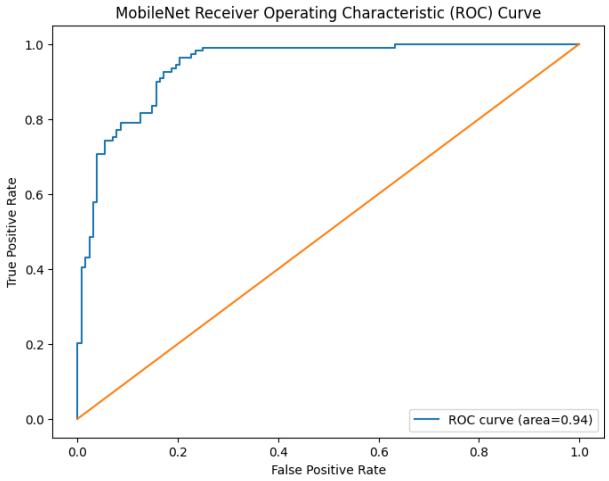
The EfficientNet model demonstrates exceptional performance on the test set, with a high accuracy of 98.73%, a low zero-one loss of 1.26%, precision, recall, F1-score, and ROC-AUC. The confusion matrix confirms the model's ability to distinguish between "Muffin" and "Chihuahua" with minimal errors. This performance suggests that EfficientNet is highly suitable for the binary classification, providing reliable and accurate results.

3-3-4) MobileNet Prediction Results

```
MobileNet Accuracy: 0.8650
MobileNet ROC-AUC: 0.9427
MobileNet Classification Report:
              precision    recall  f1-score   support

Chihuahua(0)       0.91      0.83      0.87       128
   Muffin(1)       0.82      0.91      0.86       109

    accuracy                           0.86       237
   macro avg       0.87      0.87      0.86       237
weighted avg       0.87      0.86      0.87       237
```




```
MobileNet Test Loss: 0.33914703130722046
MobileNet Test Accuracy: 0.8649789094924927
MobileNet Zero-One Loss on Test Set: 0.1350210970464135
```

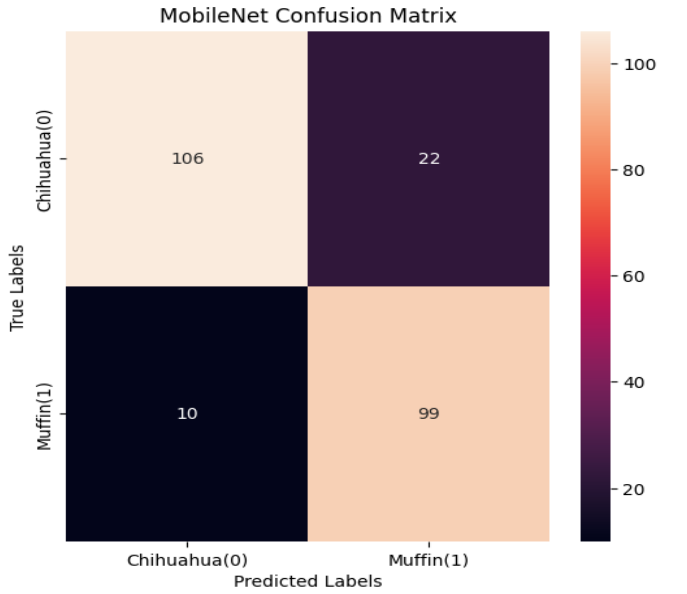


Figure 3-12: MobileNet Model Prediction Results

The confusion matrix for the MobileNet model indicates that it performs well but has some misclassifications. The ROC curve shows a significant area under the curve (AUC) of 94.27%, indicating good performance in distinguishing between the two classes.

The model has good precision and recall and, with a zero-one loss of 13.50%, shows that it correctly classifies approximately 86.5% of the images in the test set.

3-3-5) ResNet50 Prediction Results

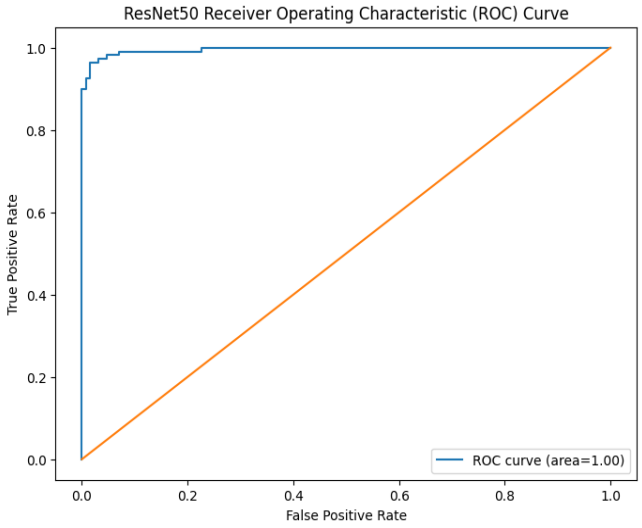
The ResNet50 model demonstrates outstanding performance on the test set with an accuracy of 96.62% and a low zero-one loss of 3.38%. The high precision, recall, and F1-scores for both classes indicate that the model effectively distinguishes between "Chihuahua" and "Muffin" images. The ROC curve further confirms the model's excellent capability in classifying the test images correctly. The confusion matrix shows minimal misclassifications, highlighting the model's robustness and reliability.

```
ResNet50 Accuracy: 0.9662
ResNet50 ROC-AUC: 0.9958
ResNet50 Classification Report:
              precision    recall  f1-score   support

Chihuahua(0)       0.97      0.97      0.97       128
   Muffin(1)       0.96      0.96      0.96       109

    accuracy                           0.97       237
   macro avg       0.97      0.97      0.97       237
weighted avg       0.97      0.97      0.97       237
```

```
ResNet50 Test Loss: 0.10078302770853043
ResNet50 Test Accuracy: 0.9662446975708008
ResNet50 Zero-One Loss on Test Set: 0.03375527426160337
```



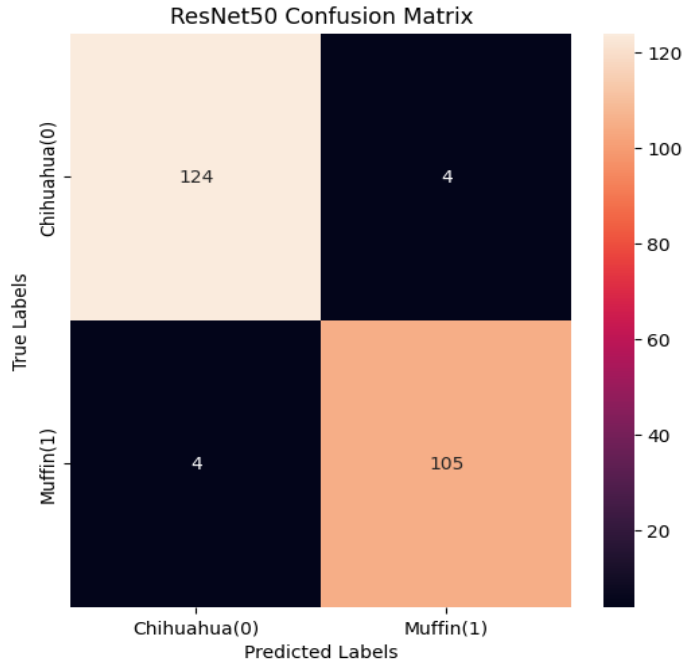


Figure 3-13: ResNet50 Model Prediction Results

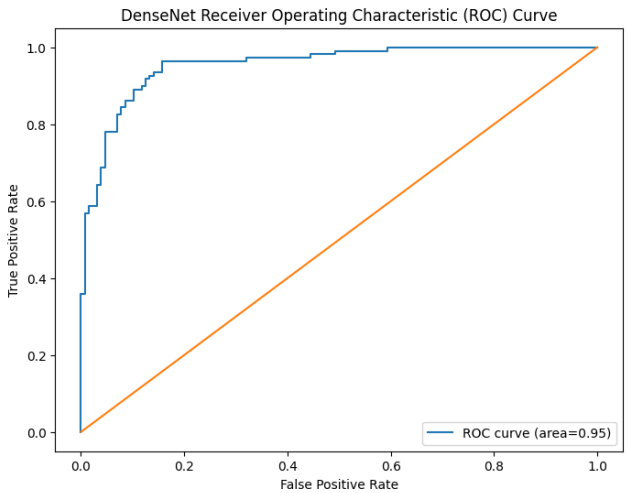
3-3-6) DenseNet121 Prediction Results

```
DenseNet Accuracy: 0.8903
DenseNet ROC-AUC: 0.9541
DenseNet Classification Report:
              precision    recall  f1-score   support

Chihuahua(0)       0.89      0.91      0.90       128
   Muffin(1)       0.90      0.86      0.88       109

    accuracy                           0.89       237
   macro avg       0.89      0.89      0.89       237
weighted avg       0.89      0.89      0.89       237
```

```
DenseNet Test Loss: 0.27646389603614807
DenseNet Test Accuracy: 0.8902953863143921
DenseNet Zero-One Loss on Test Set: 0.10970464135021098
```



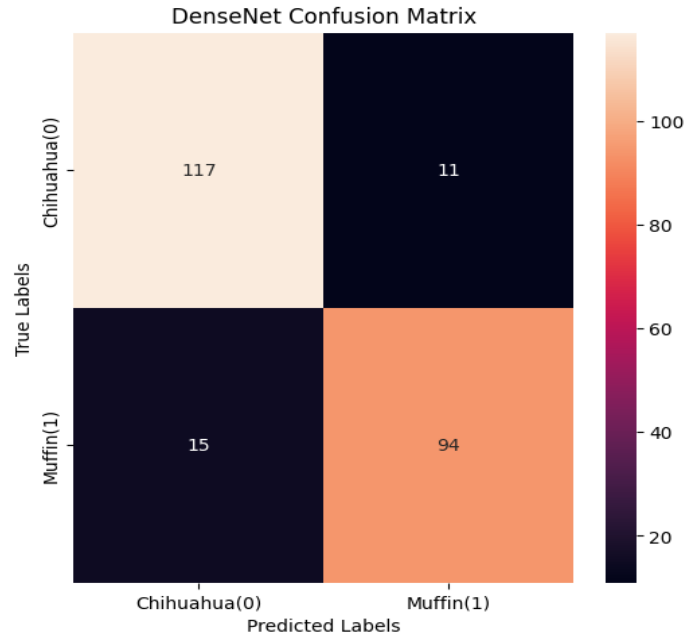


Figure 3-14: DenseNet121 Model Prediction Results

The DenseNet121 model performs well on the test set, achieving an accuracy of 89%, a high ROC-AUC score, a zero-one loss of 10.97%, and balanced precision, recall, and F1-scores for both classes. Despite its robust performance, the confusion matrix indicates that there are still some misclassifications.

3-3-7) Comparison of Models' Zero-one Losses

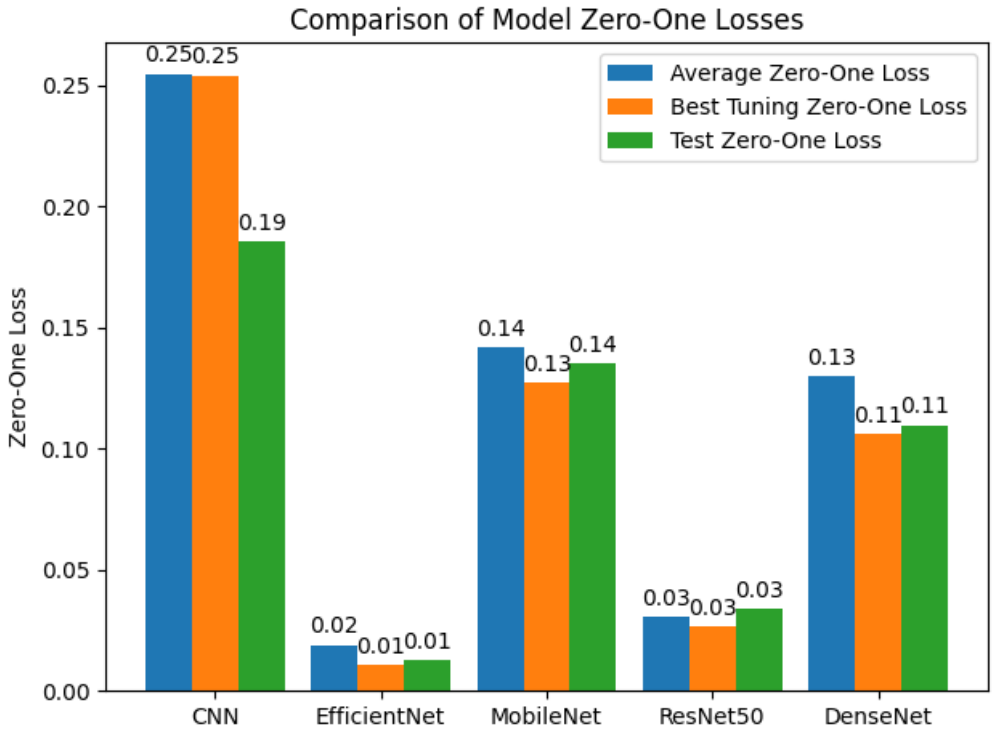


Chart 3-9: Comparison of Models' Zero-one Losses

The results show that the EfficientNet model had the best performance, with the lowest zero-one loss value around 0.01, in the evaluation process of the models. This metric underscores the model's minimal misclassification rate, reinforcing its status as the best performer.

4) Conclusions (Discussion on the Results)

In this project, I compared the performance of five different deep learning models (respectively, The Custom Convolutional Neural Network (CNN), EfficientNet, MobileNet, ResNet50, and DenseNet121 models) for the task of binary image classification in the "Muffin vs Chihuahua" dataset. A comprehensive comparison of performance metrics for these five models was done and reported as below:

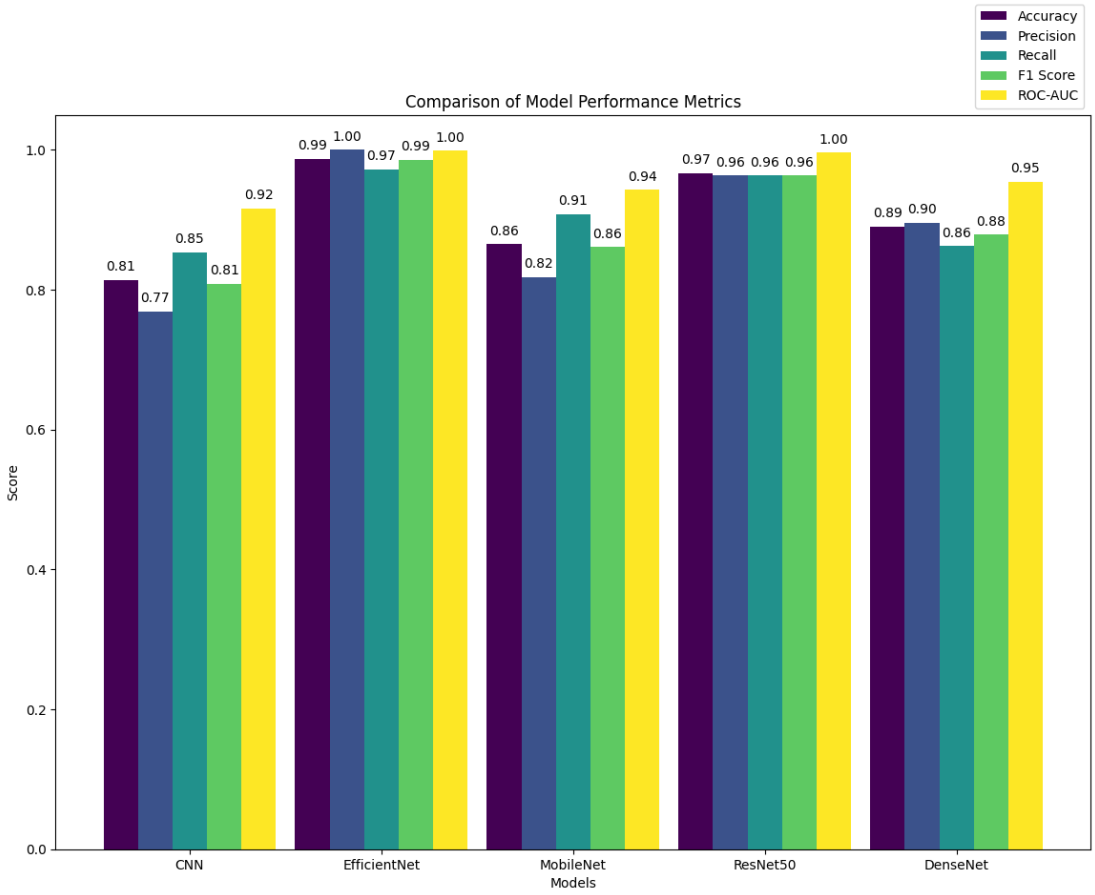


Chart 4-1: Comparison of Model Performance Metrics

The CNN model exhibited moderate performance across all metrics, with an accuracy of 81% and a ROC-AUC of 92%. Precision and recall were relatively balanced but lower compared to other models.

The EfficientNet model significantly outperformed other models with nearly perfect scores in all metrics. It achieved an accuracy of 99% and a ROC-AUC of 1.00, indicating its excellent ability to distinguish between classes.

The MobileNet model also showed strong performance, particularly in recall (91%) and ROC-AUC (94%). However, precision was slightly lower at 82%, resulting in an overall accuracy of 86%.

The ResNet50 model performed exceptionally well, with a high accuracy of 97% and a perfect ROC-AUC of 1.00. The precision and recall were both strong at 96%, leading to a balanced F1-Score of 97%.

The DenseNet121 model showed solid performance, with an accuracy of 89% and a ROC-AUC of 95%. Precision and recall were high, at 90% and 86%, respectively, indicating a good balance in performance metrics.

For the **Key Observations,** it can be concluded that the EfficientNet and ResNet50 models were top-performing, making them suitable choices for tasks requiring high accuracy and discriminative power. The MobileNet and DenseNet121 models also performed well and could be considered based on specific requirements such as computational efficiency or resource constraints. The basic CNN model, while competent, did not match the performance of the more sophisticated architectures in this comparison.

References

1) *https://www.kaggle.com/datasets/samuelcortinhas/muffin-vs-chihuahua-image-classification*
2) *Structured Binary Neural Networks for Accurate Image Classification and Semantic Segmentation by Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, Ian Reid*
3) *https://blog.aiensured.com/deep-learning-approach-to-binary-image-classification/*
4) *Ultimate Guide to Building Powerful Keras Image Classification Models by Kartik Menon*