UNIVERSITÀ DEGLI STUDI DI MILANO

BINARY IMAGE CLASSIFICATION USING NEURAL NETWORK

MAYANK PRATAP SINGH
14383A
mayankpratap.singh@studenti.unimi.it

KHUSHBOO CHAUHAN
14380A
khushboo.chauhan@studenti.unimi.it

ABSTRACT

The project intensively deals with the binary image classification using fully connected neural networks (FCNN), convolutional neural networks (CNN) and Dense Network (Dense Nets). Images of chihuahuas and Muffins from Kaggle are being used here as the dataset upon which we experimented with different CNNs and Dense Nets and investigated the performance (loss, accuracy) of our model. Further we trained the hyperparameters and finally, the risk estimation with 5-fold cross validation technique.

Contents

INTRODUCTION

Binary classification is used in the machine learning domain commonly. It is the simplest way to classify the input into one of the two possible categories. Here we have the two images to be classified either as chihuahua and muffins. The goal is to find the best architecture that classify the images by labelling it correctly.Further, we need to implement 5-fold cross validation to compute the risk estimates. It is indeed interesting, since muffin and chihuahuas are difficult to distinguish at a low resolution.

In the report, we have used CNN, DenseNets and fully connected network. And have reported the prediction results on the best model and tuned model.

TRAINING AND DATA PRE-PROCESSING

We used Google Colab to train the model which is a cloud computing service and an easy-to-setup Python environment. Upon importing the dataset, which comprises of almost 6,000 coloured JPEG images, we checked for class imbalance by counting the number of images. There was a total of 2,718 muffin images and 3,199 Chihuahua images.

In order to improve the accuracy of the networks, we applied a series of transformation to the images:

1. Data rescaling – It rescales the pixel values of the images from a range of [0,255] to [0,1], which helps the model to train faster and perform better.

2. Random transformation such as rotation, zoom, horizontal flip, width shift etc. These transformations change the original images in a dataset in a random manner during the data augmentation.

3. To ensure uniformity in preparation for subsequent processing, we resized all images to a standardized dimension of 244x244 pixels and converted them to the RGB format.

Moreover, the dataset was methodically partitioned into training, validation, and test sets, to ensure robust model training, validation, and evaluation phases.

Finally, to gain insights into the efficiency of the preprocessing steps and data augmentation techniques, visualizations were generated to showcase the transformed images alongside their respective binary classifications, where Chihuahua images were designated as class 0 and muffin images as class 1, facilitating subsequent classification tasks.

Figure 1: Normalized training images with their labels

METHODS AND MODELS

FULLY CONNECTED NEURAL NETWORK

Fully connected neural networks (FCNNs) are a type of artificial neural network where the architecture is such that all the nodes, or neurons, in one layer are connected to the neurons in the next layer.

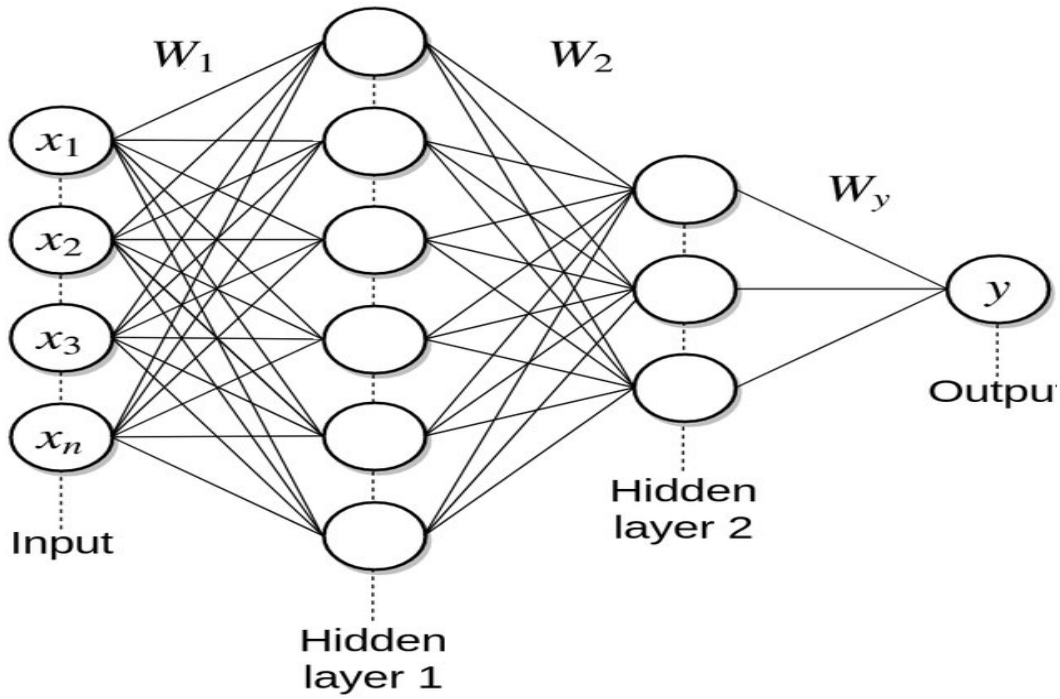

Figure 2: Fully connected neural network with two hidden layers.

However, it's not typically used for image classification tasks due to its limitations in handling the high-dimensional nature of image data. Instead, convolutional neural networks (CNNs) are commonly used for this purpose because they can capture spatial hierarchies in images more effectively, that's why we used convolutional neural network as well for training the model. The results are mentioned in the later part of the report.

MODEL 1:

We used the sequential model which adds the layers sequentially. The input layer is flattened by converting 2D image input into 1D array, specifying the height=0, width=1 and RGB colour channels. The next layer is the fully connected layer with 64 units using RELU as activation function as it is the most suitable for binary classification. To configure the learning process before training we have model.compile. This has the following specifications:

- optimizer = "adam" as it adapts the learning rate for each parameter and due to its adaptive nature, it converges faster potentially giving better results.

- loss= "binary_crossentropy" which is a loss function that penalize the incorrect prediction encouraging the model to output the probabilities close to the true labels.

- metrics= "accuracy" which specifies the performance of model using accuracy i.e. a ratio of correctly predicted instances to total instances.

During training, train_generator provides batches of images for the model to learn from, while val_generator assesses the model's performance on validation data after each epoch. Additionally, stop_early is a callback that halts training if the model's performance on the validation set doesn't improve for a set number of epochs, preventing overfitting.

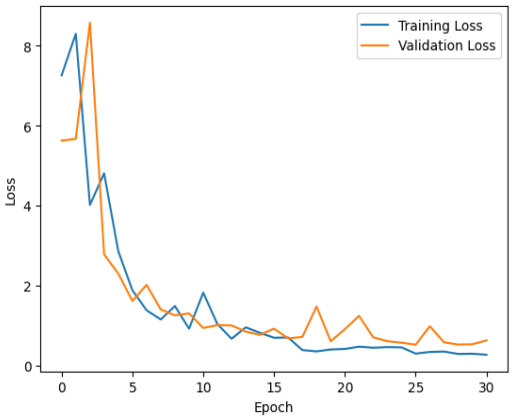

Figure 3: training and validation loss

Figure 4: training and validation accuracy

The model demonstrated good performance on both training and validation sets, the decreasing loss and increasing accuracy depicts that the model learned effectively. However, the fluctuation in validation metrics suggests potential overfitting, which we wanted to address with dropout. Hence, in model 1.2 we further added a dense layer of 128 units along with the layer of 64 units each having a dropout rate of 0.25, which resulted in the following graph:

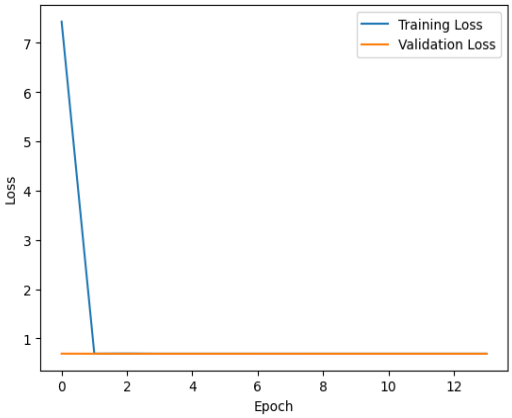

Figure 5: training and validation loss

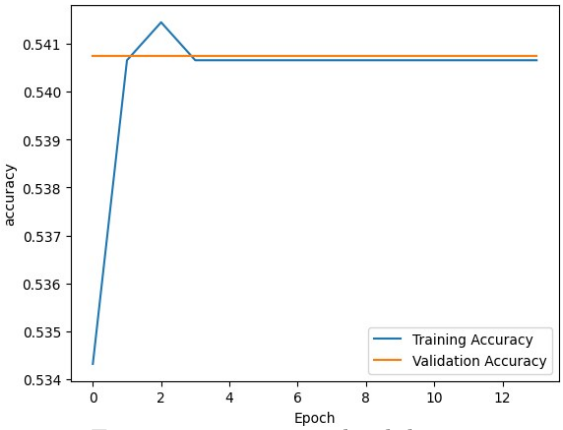

Figure 6: training and validation accuracy

While the modifications were intended to improve performance, the resulting graph indicates that the model is overfitting to training data as both training and validation accuracy are low and the rapidly stabilized loss curve shows underfitting, this means that the model is not learning from the training data. Hence, we used Densenets for a better image classification task.

DENSE NET

A DenseNet(Dense Convolutional Network) is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers.

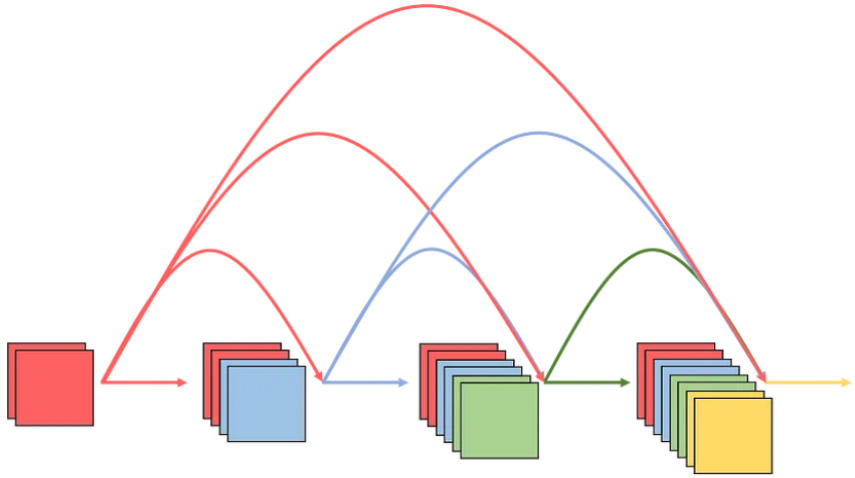

Figure 7: DenseNets

MODEL 3:

This model comprises of dense blocks, each consisting of multiple convolutional layers, and transition blocks for downscaling feature maps. The model is built using Keras with customizable hyperparameters like input shape(represented as a tuple of height, weight, colour channels), number of blocks where each dense block has multiple convolutional layers, layers per block which determines the convolutional layer within the dense layer (more layer per block allow to learn complex features) , and growth rate which defines the number of filters at each convolutional layer within a dense block. Convolutional layers with ReLU activation are used for feature extraction, followed by global average pooling (used to aggregate features learned across all dense blocks into a single vector representation) and a sigmoid output layer for binary classification. The model is compiled with Adam optimizer and binary cross-entropy loss. During training, early stopping is employed to prevent overfitting. The training history is stored in a DataFrame for analysis. This architecture offers a powerful and efficient approach for image classification tasks.

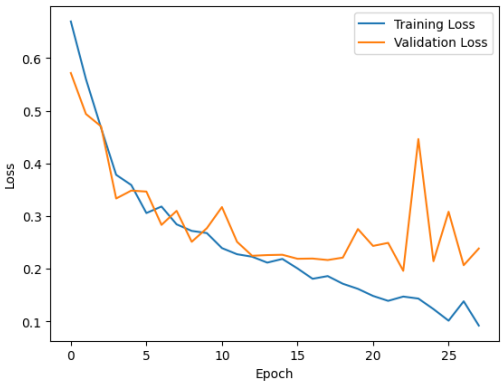

Figure 8: training and validation loss

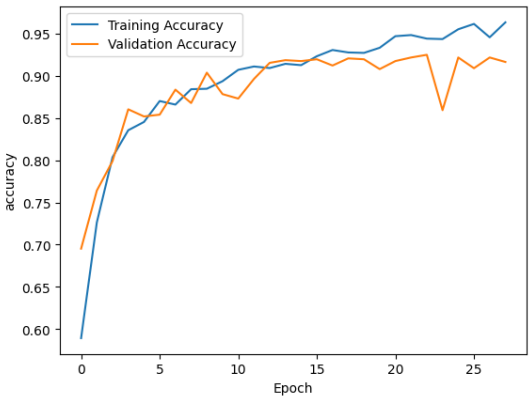

Figure 9: training and validation accuracy

CONVOLUTIONAL NEURAL NETWORK

CNNs are a class of neural networks which are quite effective in image recognition and classification. Each image in CNN passes through multiple convolutional layers with kernels (filters).

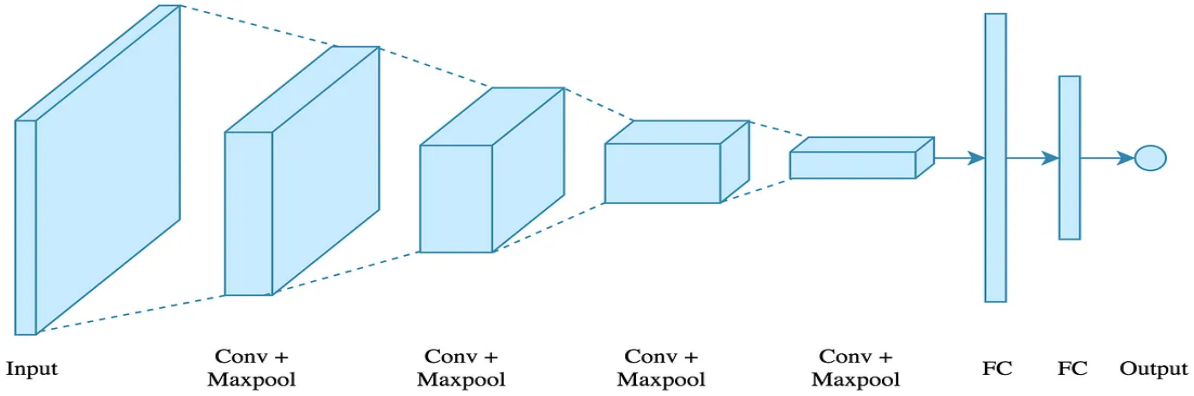

Figure 10: CNN ARCHITECTURE

CONVOLUTIONAL LAYERS

Convolution is a process of adding each element of the image to its local neighbours, weighted by the kernel. In convolution, we use various kinds of filters for extracting features from a given image.

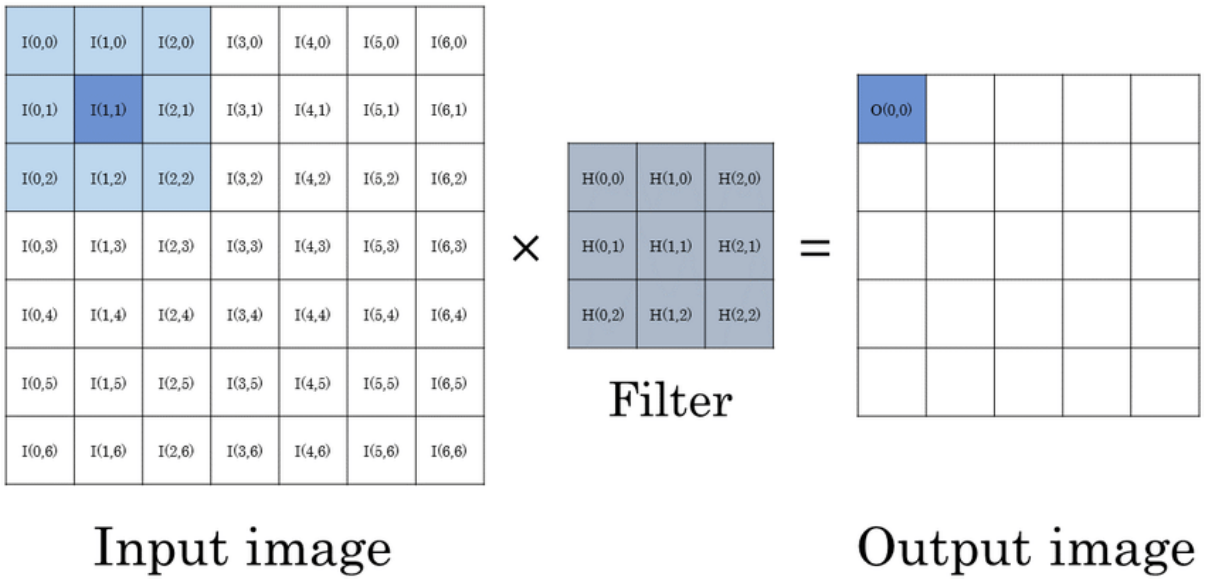

Figure 11: extracting features using filter in convolutional layer.

POOLING LAYER:

It is used to down sample the image i.e. reduces the dimensionality of each feature map but retains the most important information. Max pooling is a type of pooling that extracts only those features which have the highest value.

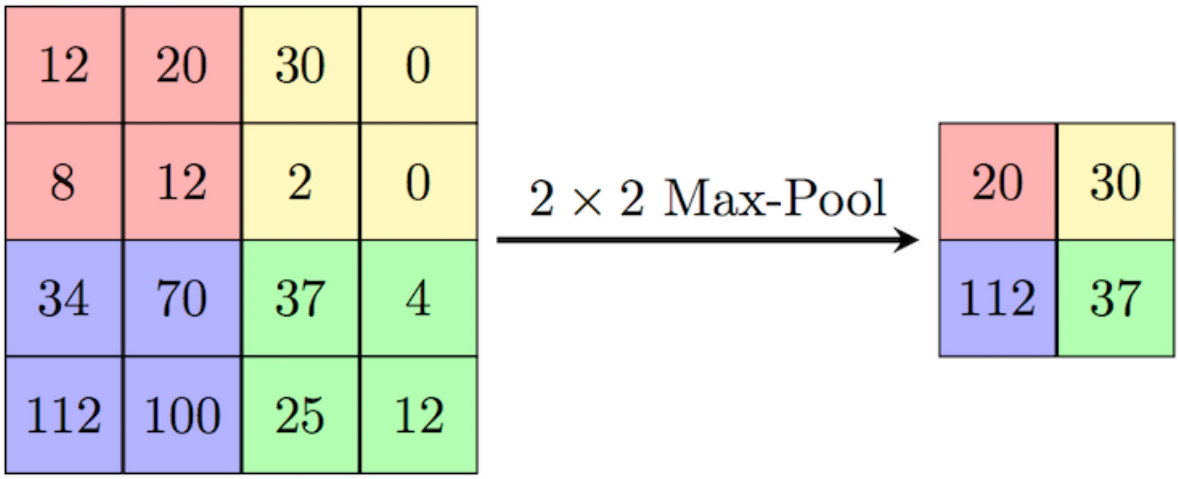

Figure 12: Max-pooling

In the above image, max Pooling is choosing maximum value features which means max pooling is giving more importance to those features which have a higher value.

The average pooling layer takes the average value of the subsections and sum pooling takes the sum of the values of the subsection.

The convolutional layer + pooling layer is considered as one layer as pooling only reduces the size of an image by giving importance to the import features and it does not produce any weights.

THE FLATTEN LAYER AND FULLY CONNECTED LAYER

The Flatten layer is used to convert the 2D output array from Pooling Layer or Convolutional layer to 1D array (Flattening the input) before feeding it to the fully connected layers.

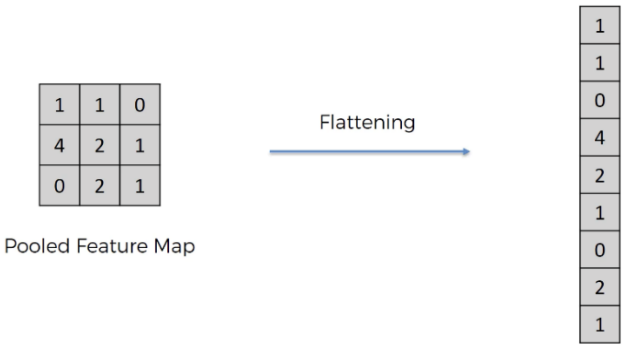

Figure 13: Flattening

The fully connected layers are a network of serially connected dense layers that would be used for classification. In a fully connected network, every neuron from layer1 is connected to every neuron in layer2. Usually for Computer Vision Application, the dense layers have large numbers of neurons (256, 128 etc.) for achieving higher accuracy in our predictions.

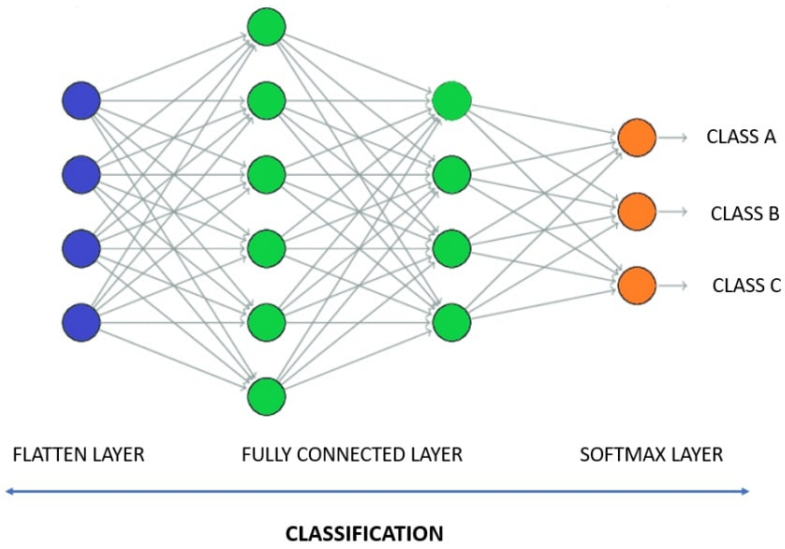

Figure 14

MODEL 2:

Using multiple CNN layers in a model can often lead to improved performance in tasks like image classification, object detection, and segmentation. That's what we did. We started from a very simple CNN model and kept experimenting with the addition of CNN layers and the dropout.

1. Model 1 (model2):

This model consists of two convolutional layers followed by max-pooling and fully connected layers. The training loss consistently decreases and the training accuracy increases showing the model is training well. The validation loss initially decreases but starts increasing after few epochs and also the validation accuracy plateaus start decreasing or fluctuating showing a sign of overfitting.

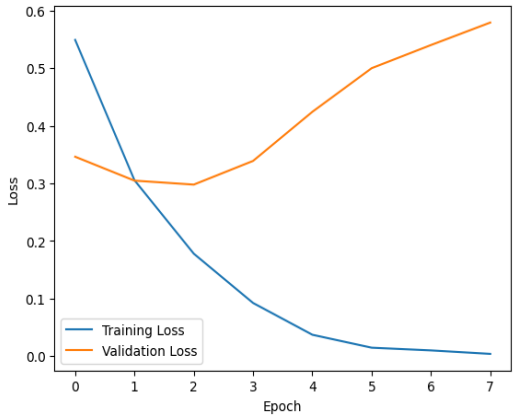

Figure 15: training and validation loss

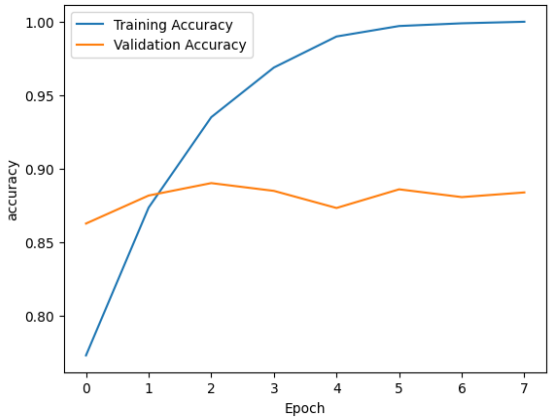

Figure 16: training and validation accuracy

2. Model 2 (model22):

In This model we added a dropout layer after the fully connected layer, helping prevent overfitting. The initial performance is lower compared to Model 1, likely due to the added regularization. However, over epochs, the model's performance improves significantly, with both training and validation accuracies increasing consistently. This indicates that the regularization introduced by the dropout layer helps the model generalize better to unseen data.

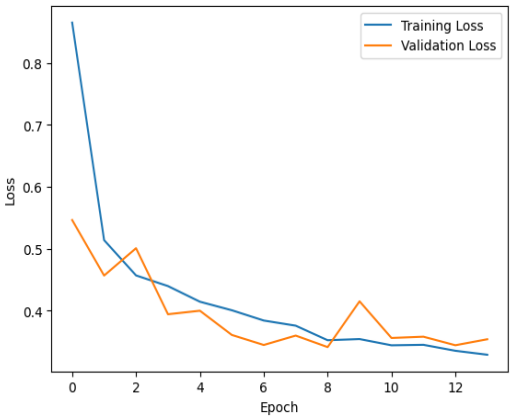

Figure 17: training and validation loss

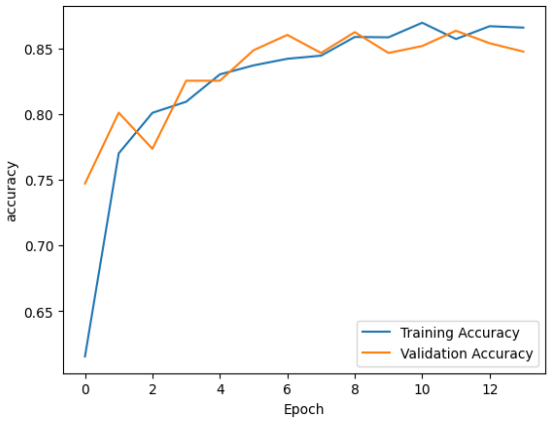

Figure 18: training and validation accuracy

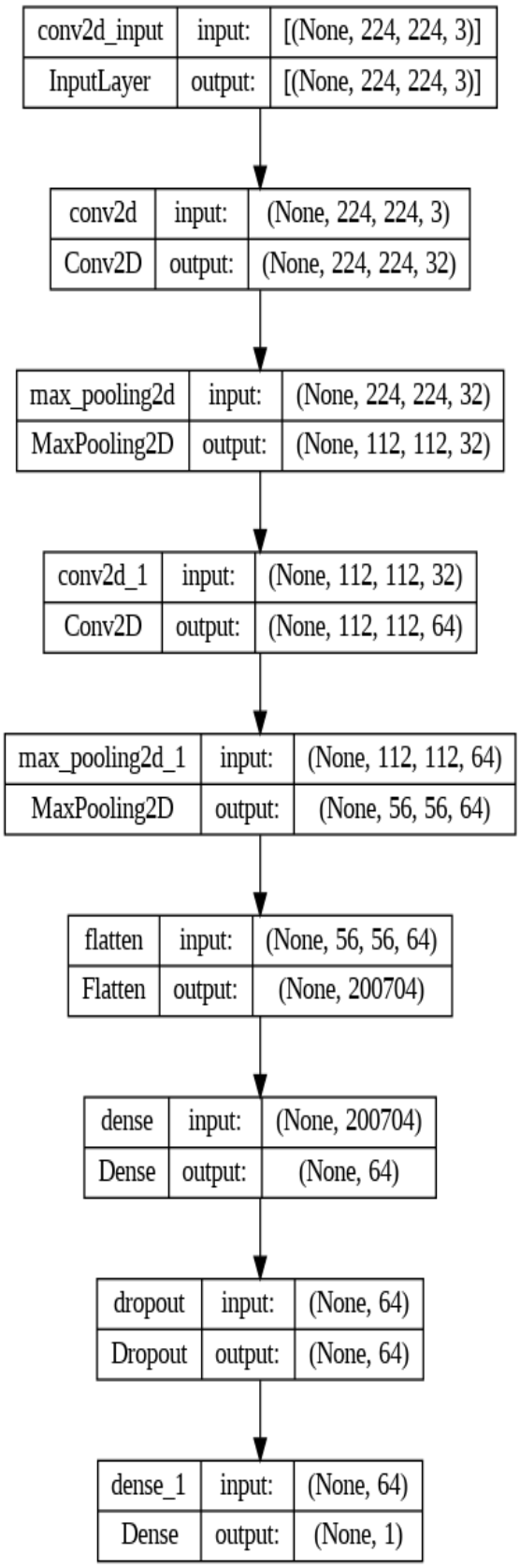

Figure 19: model22 architecture

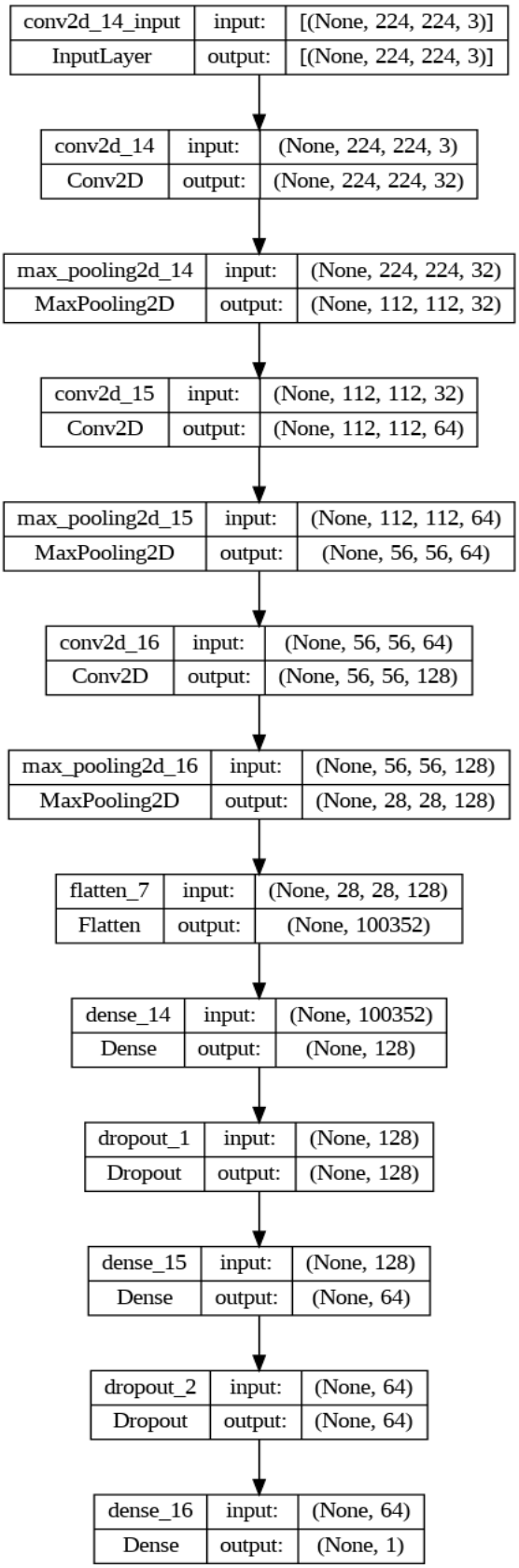

Figure 20: model23 architecture

3. Model 3 (model23):

This model further increases the depth of the network by adding an additional convolutional layer with increased filter depth. Dropout is applied after each fully connected layer to mitigate overfitting.

The initial performance is again lower compared to Model 2 but improves steadily over epochs. The validation accuracy continues to increase, indicating that the deeper architecture can learn more complex features from the data.

Showing, Deeper architectures can capture more intricate features in the data, leading to better generalization and higher accuracy on unseen data.

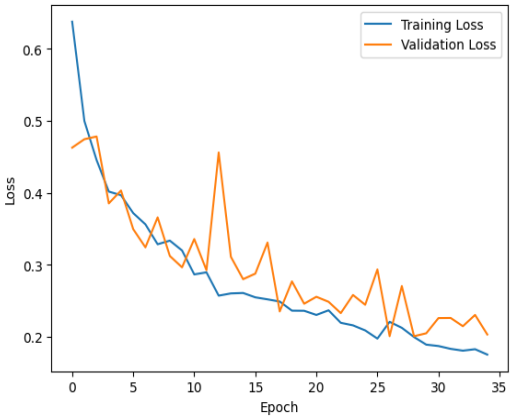

Figure 21: training and validation loss

Figure 22: training and validation accuracy

HYPERPARAMETER TUNING

Performing hyperparameter tuning using techniques like Bayesian optimization was likely done to systematically search for the best combination of hyperparameters for convolutional neural network (CNN) model.

Hyperparameters such as the number of filters, kernel sizes, dropout rates, and learning rates significantly impact the performance of a CNN model. Systematically searching through various combinations of hyperparameters can help identify configurations that lead to better model performance. The goal of hyperparameter tuning is to improve the model's performance metrics such as accuracy, precision, recall, or F1-score on unseen data.

The figure shows the best combination of parameters for CNN model. The resulted graph for training validation loss and accuracy shows the better generalization. The fluctuations in the validation accuracy indicates inconsistency across different batches for which we considered k-fold cross validation in the next step.

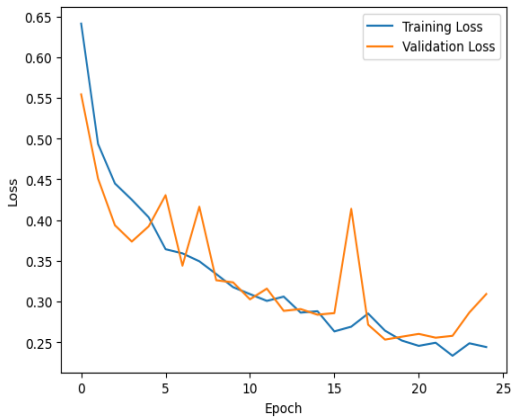

Figure 23: training and validation loss

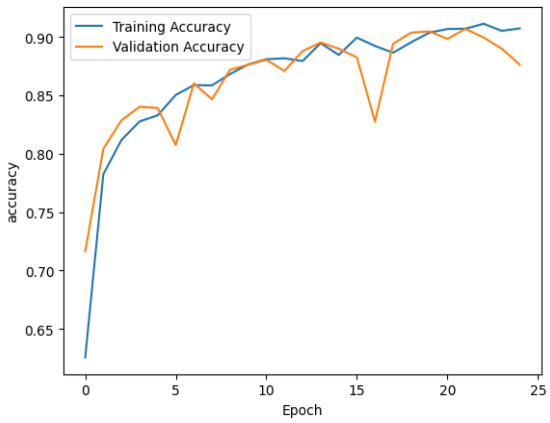

Figure 24: training and validation accuracy

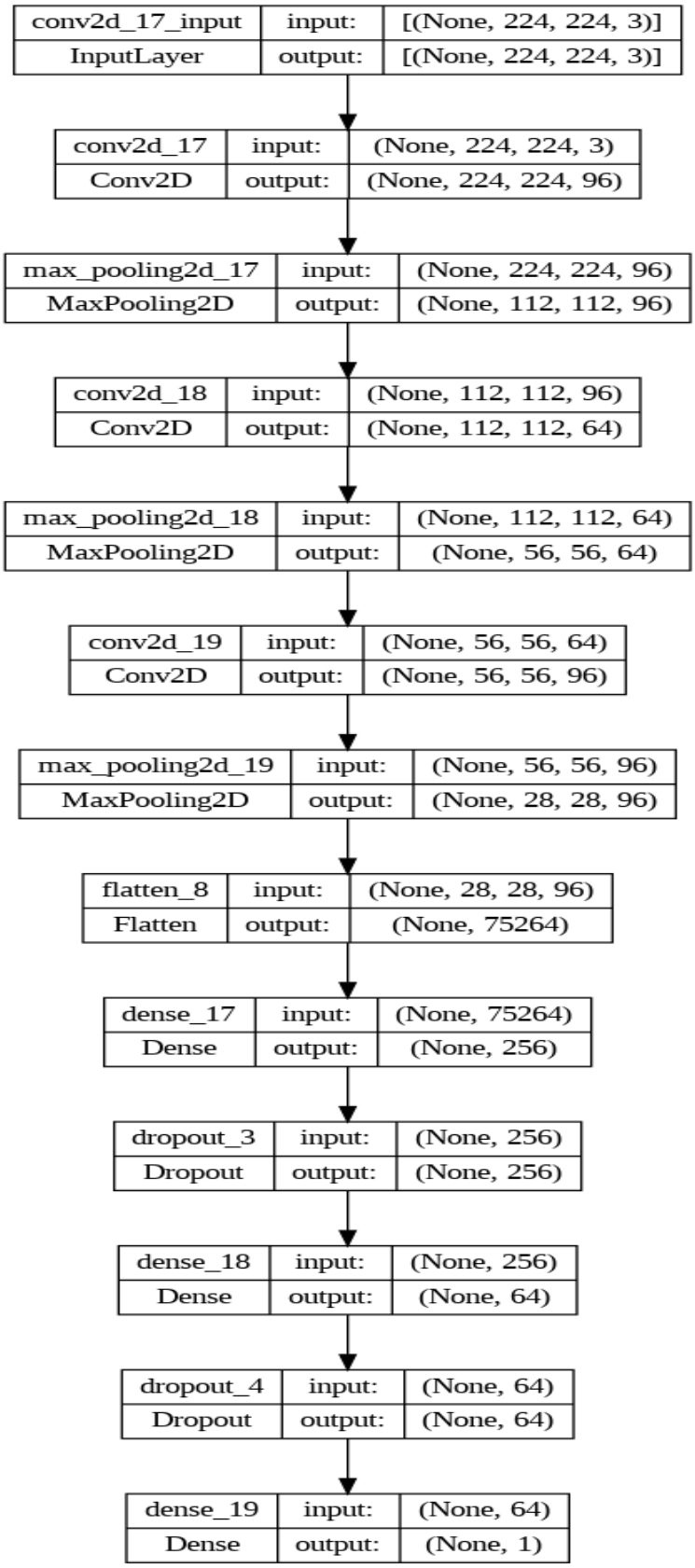

Figure25: Tuned model architecture

PREDICTION RESULTS

Using CNN model23

The prediction done on the test set of images (1184) provided an accuracy of 92%. The accuracy report and the confusion matrix (figure:26) is provided below.

	precision	Re-call	F1-score	support
Chuhuahua	0.93	0.93	0.93	640
Muffin	0.91	0.92	0.91	544
Accuracy			0.92	1184
Macro avg	0.92	0.92	0.92	1184
Weighted avg	0.92	0.92	0.92	1184

Table1: accuracy report of model23 on test set

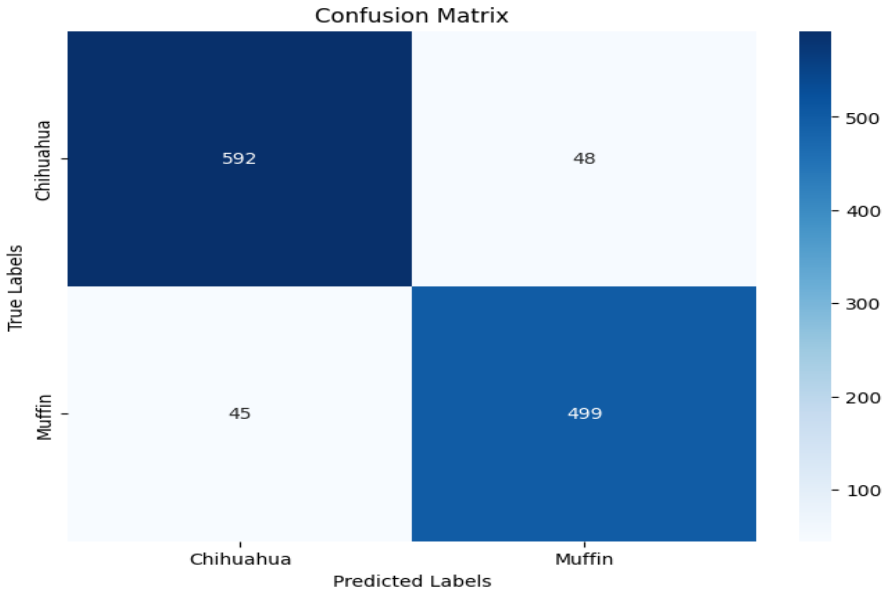

Figure 26:Confusion matrix

Using Tunned Model

We further checked the prediction done on the test set of 1184 images using the tunned model (after the hyperparameter were fine tuned), which provided an accuracy of 90%. The accuracy report and the confusion matrix (figure:27) is provided below.

	precision	Re-call	F1-score	support
Chihuahua	0.85	0.98	0.91	640
Muffin	0.97	0.80	0.88	544
Accuracy			0.92	1184
Macro avg	0.91	0.89	0.90	1184
Weighted avg	0.91	0.90	0.90	1184

Table2: accuracy report of tuned model on test set

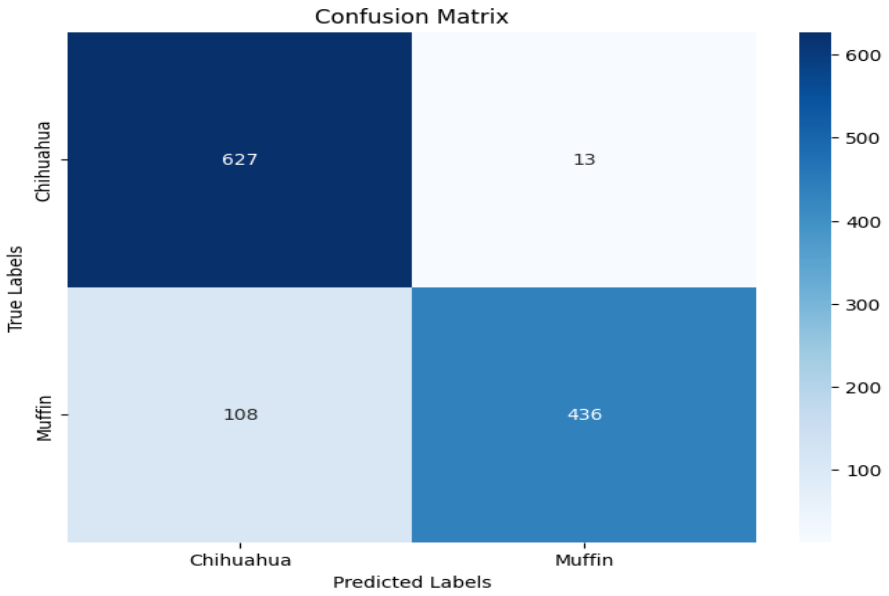

Figure 27:Confusion matrix

While the accuracy of the model after hyperparameter tuning is slightly lower than the model with randomly selected layers, the difference is not significant (90% vs. 92%).

The evaluation metrics such as loss and accuracy also show similar trends. The model with randomly selected layers has a lower loss and higher accuracy compared to the model after hyperparameter tuning.

Both models achieve high precision, recall, and F1-score for both classes (chihuahua and muffin). There are slight differences in these metrics between the two models, but they generally perform well for both classes.

5-FOLD CROSS-VALIDATION

Cross-validation serves as a crucial step in evaluating the robustness and generalization capability of machine learning models, particularly in binary classification tasks like distinguishing between images of muffins and chihuahuas. By partitioning the dataset into multiple subsets and iteratively training and evaluating the model on different combinations of these subsets, cross-validation provides insights into the model's performance across various data distributions. This process helps mitigate issues such as overfitting to specific subsets or biases introduced by a single train-test split.

In our project, we employed 5-fold cross-validation to rigorously assess the performance of our convolutional neural network (CNN) tuned-model. The obtained zero-one loss, ranging from 0.114 to 0.459 across folds with an average of 0.194, indicates the model's ability to generalize well to unseen data. While slight variations in performance were observed between folds, the overall consistency and effectiveness of the model in classifying images of muffins and chihuahuas were validated.

FOLD	**ZERO-ONE LOSS**
FOLD 1	0.11486488580703735
FOLD 2	0.12584459781646729
FOLD 3	0.14527028799057007
FOLD 4	0.45945948362350464
FOLD 5	0.1241554021835327
AVERAGE	0.1939189314842224

Table 3: ZERO-ONE LOSS

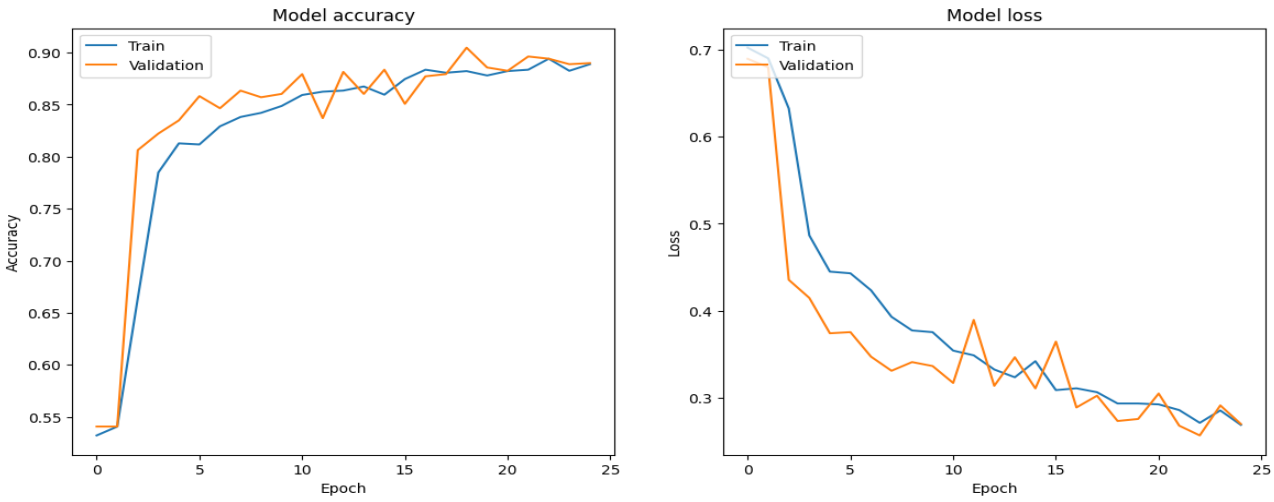

Figure 28: model accuracy and loss

CONCLUSION

The main aim of the work was to try different architectures to perform binary classification on the images of muffins and chihuahuas.

Different architectures showed different accuracies. The model23 with 3 convolutional and max pooling layers, 2 dropout layers with dropout rate of 0.3 resulted with the best accuracy and it also performed well during the prediction done on the test data, clearly indicating that the deeper architecture with additional convolutional layers and dropout regularization led to better performance.

Also, the cross validation makes the accuracy more stable providing with better results.

The experiment also drew the conclusion, that while hyperparameter tuning aims to optimize model performance, it's essential to consider trade-offs between complexity and performance. Sometimes, models with randomly selected hyperparameters might generalize better to unseen data. It's crucial to strike a balance between model complexity and performance when selecting hyperparameters. Further experimentation and fine-tuning may be necessary to find the optimal configuration for your specific task and dataset.

DECLARATION OF ORIGINALITY

We declare that this material, which we now submit for assessment, is entirely our 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 our work. We understand that plagiarism, collusion, and copying

are grave and serious offences in the university and accept the penalties that would be imposed should we engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study.