Data Science and Economics
Department of Economics, Management and Quantitative Methods
Department of Computer Science "Giovanni degli Antoni"
Università degli Studi di Milano

Image classification with Machine Learning:

Recognising muffins and chihuahuas classes with Neural Networks

Rana Nosabahsalout - 963749

Soudabeh Masoudisoltani - 961506

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

Link to the repository:

Soudabeh Masoudisoltani:

https://github.com/pnightsore/ML_ImageClassification

Rana NosabahSalout:

https://github.com/Rananosabah/ML_ImageClassification/tree/main

Table of Contents:

Chapter 1: Summary

The aim of this project is to perform a binary classification task with data as images, and to find a good classifier that is able to classify images into two different classes of Chihuahua and Muffins while avoiding overfitting and minimizing test and train loss.

To achieve this goal, we have introduced 5 models built on different CNN architectures.

The first 4 models are built from scratch, and they seem average at predicting unseen values, model 4 is introduced as the tuned version of model 3 and it seems to be good at predicting the true class for unseen data.

Finally, we added a last model, model 5, that is built not from scratch but chosen among three pre-trained architectures called Xception and InceptionV3 and EfficientNetB0

We will see that the model with hyperparameter tuning as well as the one that has been built on previously trained architectures will have the highest accuracy among the ones we presented in this project.

The extremely high amount of accuracy and the very low amount of zero-one loss observed in model 5 shows that it can be our preferred choice among the 5 models.

Chapter 2: Data and preprocessing

To conduct this image classification research and to perform image classification tasks, we have used an available dataset from Kaggle

This dataset includes 5917 diverse types of images both of Chihuahua dogs and of Muffins.

Since all images in our dataset are labeled, this is of course a supervised learning task.

To make some modification and to prepare the data for our research, we have performed several pre-processing techniques that will be discussed in the following section.

2.1. Pre-processing of the data

We have converted all the images from JPEG to RGB hence input shape (224, 224, 3), and the pixels are scaled from [0, 255] to [0, 1].

We added a Data Augmentation layer to model 3 and 4, and its shape was kept as before.

There are still several changes made to the data before training that were particular only for some of the models, these manipulations will be discussed later and in each specific model's section.

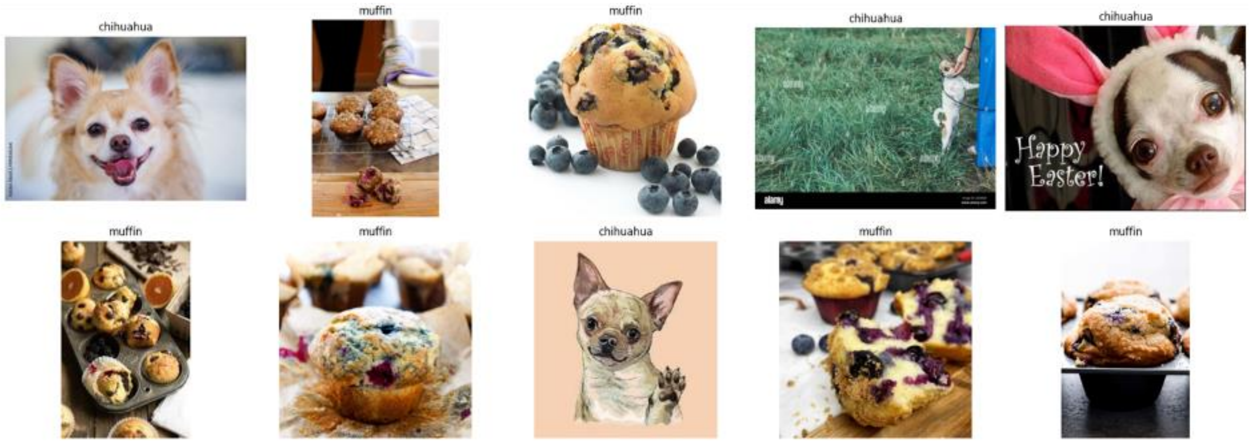


Figure 2.1 A random visualization of the train data

Chapter 3: Model components and some essential theory

To perform this project, several CNN architectures have been used, each with their own types of layers, consisting of convolutional, max pooling, dense, data augmentation flatten and dropout layers.

in this section we briefly explain each layer type and finally some ideas behind the choice of our:

- loss function
- Activation function
- Optimizer

Are provided.

3.1. Convolutional layer

This layer is a key component of CNN, used mostly for image processing and recognition tasks.

Convolutional layer takes an image and scans it with a filter (small grid of weights) to detect patterns such as shapes or edges.

Then, the filter moves across all the images creating a new set of images called feature maps. Feature maps highlight these detected patterns.

The above process works with the aim of helping CNN understand important aspects of an image, and finally allowing it to recognize patterns and objects more effectively.

After the convolution operation, the activation function is applied.

An activation function determines if a neuron should be activated or not and does this by using a mathematical operation and applying it to the input of that neuron. This allows the network to learn and model more complex patterns that are not linear.

ReLU, Tanh and Sigmoid are some common examples of activation functions.

ReLU outputs zero if the input is less than 0 and the input itself for positive values of inputs.

Sigmoid however, puts the input in range between 0 and 1.

In summary these activation functions' job is to help the network capture relationships and consequently make more accurate predictions.

In this project we have used Tanh and Relu activation functions, we also tried Sigmoid for model 1, however, as we expected it was not a good choice for this particular dataset. below is a formal representation of each activation function used:

- Tanh activation function which is mostly used in the hidden layers

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

- Relu activation function, linear for positive and 0 for negative inputs

$$\mathrm{ReLU}(x) = \max(0, x)$$

- Sigmoid activation function which ranges between 0 and 1, it is used in the output layer for binary classification

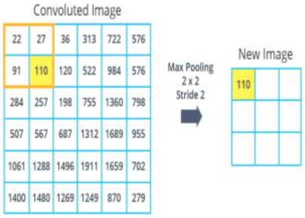
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

3.2. Max pooling layer

This layer reduces the dimension of an input by scanning it first with a small window and selecting the largest value from each region it covers , the network then becomes more efficient and overfitting lessens.

This layer plays a very useful role in CNN by discarding less relevant information while down sampling the input. This finally, helps the network become more time efficient and focused on the most important patterns in the data.

F*igure 3.1 Max pooling example*



22	27	36	313	722	576
91	110	120	522	984	576
284	257	198	755	1360	798
507	567	687	1312	1689	955
1061	1288	1496	1911	1659	702
1400	1480	1269	1249	870	279

110		

3.3. Average pooling

Average Pooling is a pooling operation that calculates the average value for patches of a feature map and uses it to create a down sampled (pooled) feature map. It is usually used after a convolutional layer. It adds a small amount of translation invariance - meaning translating the image by a small amount does not significantly affect the values of most pooled outputs. It extracts features more smoothly than Max Pooling, whereas max pooling extracts more pronounced features like edges.

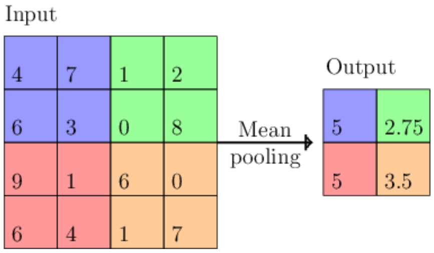


Figure 3.2 Average pooling example

3.3. Flatten layer

This layer comes to work only after the max pooling layer, and it changes the max pooling layer output's spatial dimension to a single dimension tensor. Then this single dimension tensor is used as the input for the following dense layer. So in short, the Flatten layer helps prepare the input of the Dense layer.

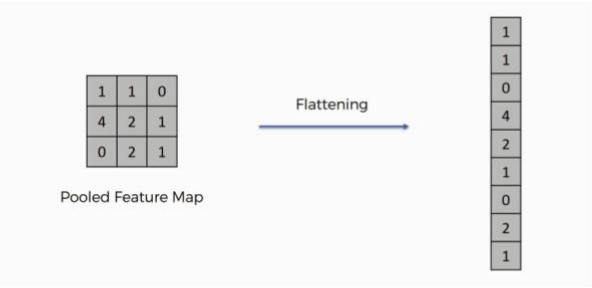


Figure 3.3 Example of Flatten layer

3.4. Dense layer

This fully connected layer is a layer where neurons are connected each and every one of them in the previous layer.

Through learned weights and biases, neurons from the previous layer are combined then after applying an activation function to produce the output.

This layer is among the last layers of the network integrating features from previous layers and using it to make final predictions.

3.5. Dropout layer

The dropout technique tries to prevent overfitting while training. In each iteration of the training process, some neurons are dropped out in a specific layer to add noise to the model.

The dropout rate takes value between 0 and 1. This number is associated with the fraction of neurons that should be dropped out in each training iteration. For example, a dropout rate equal to 0.1 means that during each training iteration 0.1 of the neurons will be dropped out randomly.

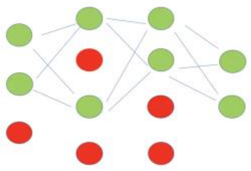


Figure 3.4 Dropout example

3.6. Data Augmentation Layer

This technique also tries to reduce overfitting, improve generalization. It does this by generating new images for the training set from each image and by adding it to the dataset.

By implementing this technique and adding it to the layers we improve the performance and robustness of the data.

Chapter 4: Architectures, Models, and their performance analysis

It’s worth pointing out the difference between architecture and models.

The architecture is the design of the network. Including the arrangements and types of layers used in the design of our network among the ones mentioned in the previous section as well as the size and number of filters. In short, the architecture defines the network structure, and the way data flows through it.

On the other hand, the model is architecture plus the trained parameters such as biases and weights. The model is created after training the network on the dataset. We then use the model to make predictions and perform tasks. As architecture is the design of the network, the model is the sum of the design as well as the trained parameters.

Before introducing the models made and used in this project, we present some very important concepts that are going to be used to explain and evaluate our models later:

- Loss function

Since we are dealing with a task of binary image classification, the binary cross entropy loss function is utilized. Below is the formal definition of this loss function:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))$$

N being the number of data points, y_i true label (0, 1), for the i^{th} data point, pi predicted probability of the class with label 1 for i^{th}data point.

- Zero-one loss:

The simplest loss function that counts the mistakes made by the classifier on the train set, for every mistake it takes 1 and 0 otherwise. This loss function is common for classification.

- Optimizer

Our goal is to minimize the value for the loss function, here we have utilized Adam (Adaptive Moment Estimation) optimizer. This is a proper choice of optimizer when dealing with many parameters. In terms of computational load and memory it is an efficient choice.

- Early stopping and patience

This is a technique to prevent overfitting. It works in this way: we monitor a metric while training and set a number as patience. If that metric does not improve after several epochs (patience) we ask to stop the training and no more epochs will be trained. So, patience determines the number of epochs we are willing to wait until we observe improvement.

4.1. Model 1

4.1.1. Model 1 architecture

We start from a simple sequential model with 64 batch sizes. This sequential model is the simplest here and does not use rescaled data. The patience of Early stopping was set to 3, meaning the training will stop after 3 epochs without improvement in validation loss.

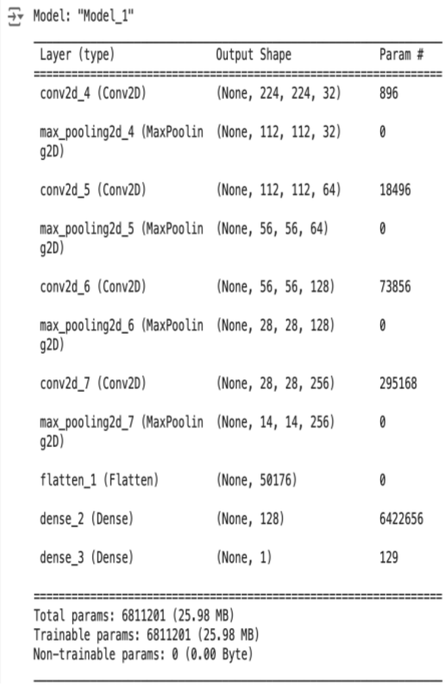
To avoid overlapping of pixels, we used the same dimension for stride and pool_size.

The max pooling layer would choose the maximum value among several pixels and by taking only one value out of several, reduce the dimension. After every convolutional layer, we added max pooling to return the most highlighted part of the image. In the last layer we always used the Sigmoid activation function because it is a binary classification task.

For this model we have made 4 convolutional blocks with respectively 32, 64, 128, 256 filters, each with one convolutional layer and for each block a max pooling layer has been used after the convolutional layer to reduce dimension.

2 by 2 pool size and same for stride to avoid overlapping. Finally in the fully connected layers we have added a dense with 128 units and tanh activation function and a final dense layer with Sigmoid as activation function and 1 unit. No dropout and no augmentation is used for this model.

Figure 4.1 Model 1 summary

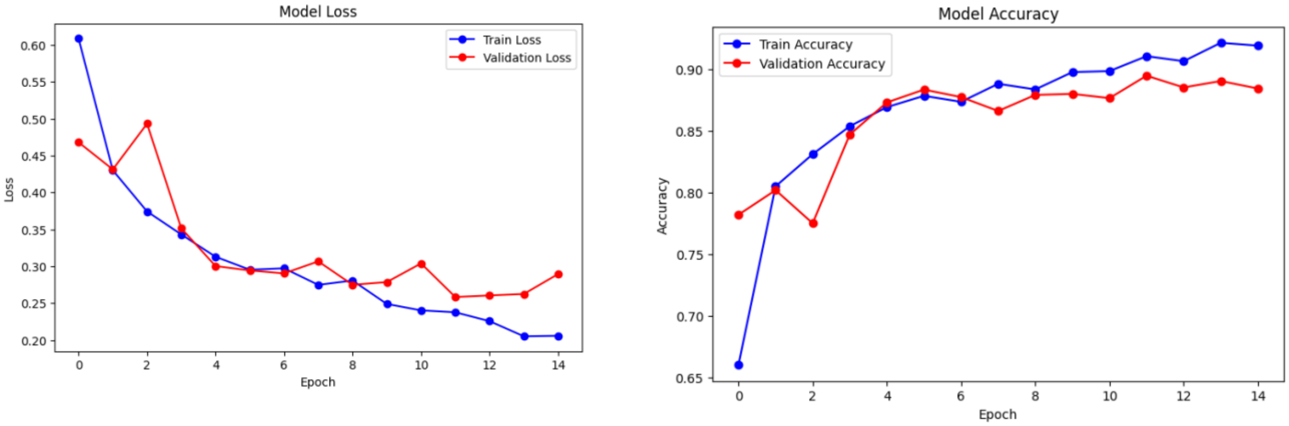


```
Model: "Model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 224, 224, 32)      896

 max_pooling2d_4 (MaxPoolin  (None, 112, 112, 32)      0
 g2D)

 conv2d_5 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_5 (MaxPoolin  (None, 56, 56, 64)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 56, 56, 128)       73856

 max_pooling2d_6 (MaxPoolin  (None, 28, 28, 128)       0
 g2D)

 conv2d_7 (Conv2D)           (None, 28, 28, 256)       295168

 max_pooling2d_7 (MaxPoolin  (None, 14, 14, 256)       0
 g2D)

 flatten_1 (Flatten)         (None, 50176)             0

 dense_2 (Dense)             (None, 128)               6422656

 dense_3 (Dense)             (None, 1)                 129

=================================================================
Total params: 6811201 (25.98 MB)
Trainable params: 6811201 (25.98 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4.1.2. Model 1 Performance analysis

	Batch size	*Learning rate*	*Optimizer*	*Activation function*	*Epochs*	*Epochs before stopping*	*metrics*
Model properties	64	0.001	Adam	Tanh, Sigmoid	30	15	Accuracy

Figure 4.2 Model 1 Loss/ Accuracy plot



Model 1 performance:

Train loss	0.2584
Train accuracy	0.8950
Test loss	0.2927
Test accuracy	0.8885

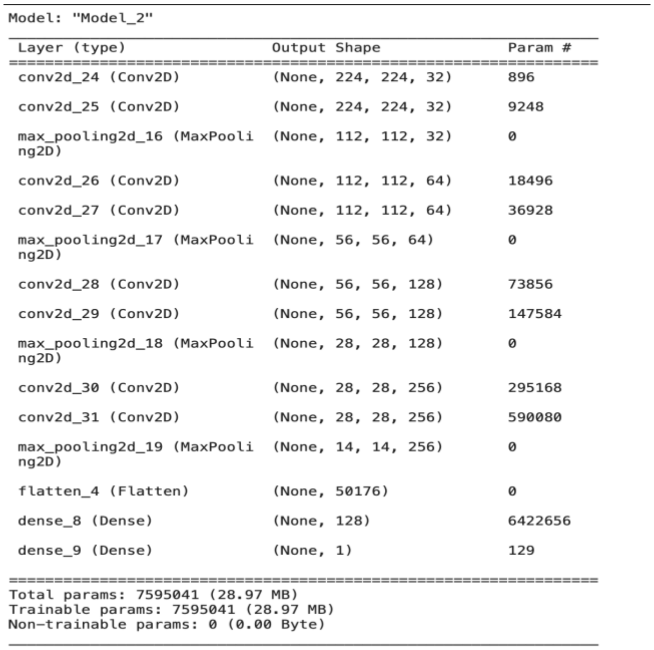
As seen above through accuracy value and graphs this is a fairly good model. We face early stopping with patience 3, so the training stops at the 15th epoch due to no improvement in loss.

The graphs show a persistent disparity among training and validation accuracies, along with an increasing trend in validation loss as training progresses. These observations strongly suggest that the model is experiencing overfitting and fails to generalize effectively to unseen data

4.2. Model 2

4.2.1. Model 2 architecture

This model’s architecture is like Model 1, but we have used Relu activation function instead of Tanh. and each convolutional block has two identical layers instead of one with the same filters. We don’t expect a big difference in the output of this model, since model 1 and model 2 are very similar. And finally, we have 2 dense layers with the same number of units as before.



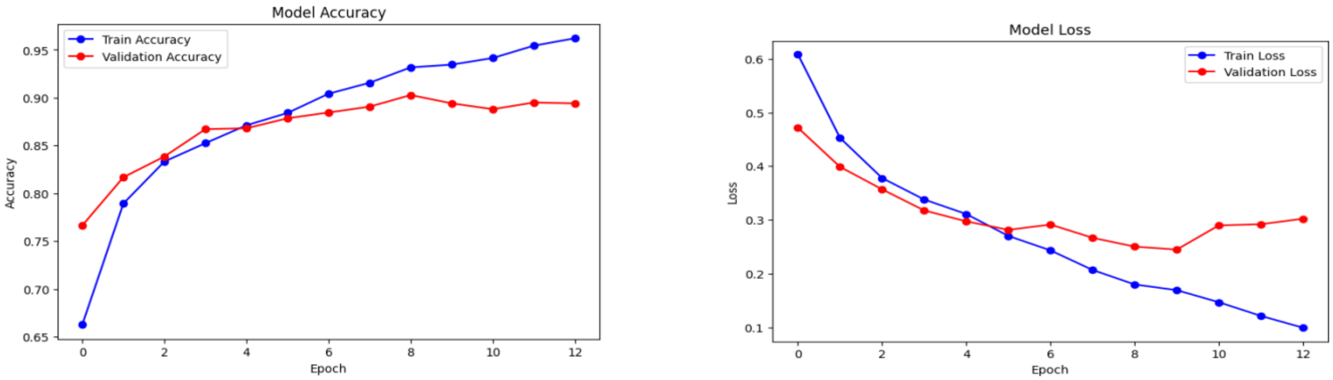
```
Model: "Model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 224, 224, 32)      896

 conv2d_25 (Conv2D)          (None, 224, 224, 32)      9248

 max_pooling2d_16 (MaxPooli  (None, 112, 112, 32)      0
 ng2D)

 conv2d_26 (Conv2D)          (None, 112, 112, 64)      18496

 conv2d_27 (Conv2D)          (None, 112, 112, 64)      36928

 max_pooling2d_17 (MaxPooli  (None, 56, 56, 64)        0
 ng2D)

 conv2d_28 (Conv2D)          (None, 56, 56, 128)       73856

 conv2d_29 (Conv2D)          (None, 56, 56, 128)       147584

 max_pooling2d_18 (MaxPooli  (None, 28, 28, 128)       0
 ng2D)

 conv2d_30 (Conv2D)          (None, 28, 28, 256)       295168

 conv2d_31 (Conv2D)          (None, 28, 28, 256)       590080

 max_pooling2d_19 (MaxPooli  (None, 14, 14, 256)       0
 ng2D)

 flatten_4 (Flatten)         (None, 50176)             0

 dense_8 (Dense)             (None, 128)               6422656

 dense_9 (Dense)             (None, 1)                 129

=================================================================
Total params: 7595041 (28.97 MB)
Trainable params: 7595041 (28.97 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 4.3 Model 2 summary

4.2.2. Model 2 Performance analysis

	Batch size	*Learning rate*	*Optimizer*	*Activation function*	*Epochs*	*Epochs before stopping*
Model properties	64	0.001	Adam	Relu, Sigmoid	20	13

Figure 4.4 Model 2 Loss/ Accuracy plots



Model 2 performance:

Train loss	0.2446
Train accuracy	0.8941
Test loss	0.2840
Test accuracy	0.8952

This model is improved in performance relative to model 1, visible by accuracy and reduced loss for both training and validation. Such improvements are a result of proper modifications in the model's architecture or training methodologies, enhancing its ability to learn and generalize from training data. However, signs of overfitting remain, though reduced, highlighting an area for potential future refinement.

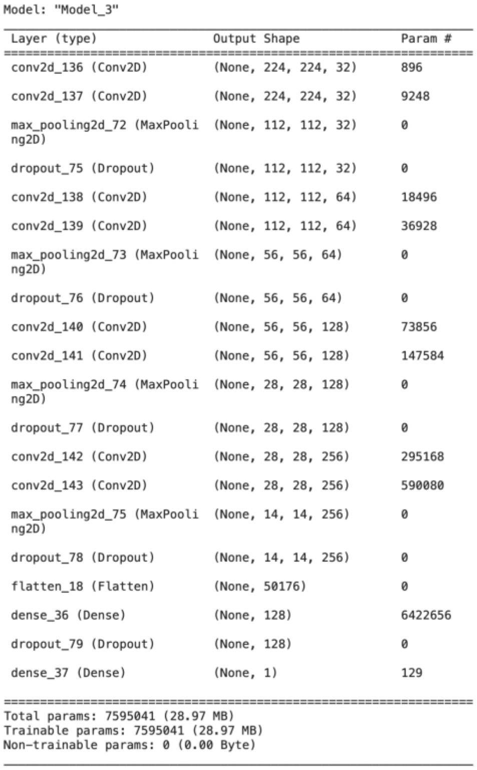
4.3. Model 3

4.3.1. Model 3 architecture

This model also consists of 4 convolutional layers with the same number of filters(32, 62, 128, 256) as the previous two models and uses relu activation function, but with the difference that after the max pooling layer in each convolutional block, we added a dropout layer with dropout rate equal to 0.2

We also added a dropout layer with the rate 0.3 in the fully connected layers.

Data augmentation layer is also added to this model that is described in the next page.



```
Model: "Model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_136 (Conv2D)         (None, 224, 224, 32)      896

 conv2d_137 (Conv2D)         (None, 224, 224, 32)      9248

 max_pooling2d_72 (MaxPooli  (None, 112, 112, 32)      0
 ng2D)

 dropout_75 (Dropout)        (None, 112, 112, 32)      0

 conv2d_138 (Conv2D)         (None, 112, 112, 64)      18496

 conv2d_139 (Conv2D)         (None, 112, 112, 64)      36928

 max_pooling2d_73 (MaxPooli  (None, 56, 56, 64)        0
 ng2D)

 dropout_76 (Dropout)        (None, 56, 56, 64)        0

 conv2d_140 (Conv2D)         (None, 56, 56, 128)       73856

 conv2d_141 (Conv2D)         (None, 56, 56, 128)       147584

 max_pooling2d_74 (MaxPooli  (None, 28, 28, 128)       0
 ng2D)

 dropout_77 (Dropout)        (None, 28, 28, 128)       0

 conv2d_142 (Conv2D)         (None, 28, 28, 256)       295168

 conv2d_143 (Conv2D)         (None, 28, 28, 256)       590080

 max_pooling2d_75 (MaxPooli  (None, 14, 14, 256)       0
 ng2D)

 dropout_78 (Dropout)        (None, 14, 14, 256)       0

 flatten_18 (Flatten)        (None, 50176)             0

 dense_36 (Dense)            (None, 128)               6422656

 dropout_79 (Dropout)        (None, 128)               0

 dense_37 (Dense)            (None, 1)                 129

=================================================================
Total params: 7595041 (28.97 MB)
Trainable params: 7595041 (28.97 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 4.5 Model 3 summary

4.3.2. Model 3 Performance analysis

In this model we performed a 5-fold cross validation. And the average of all 5 validation losses and accuracy in these 5 folds will be reported. We also added data augmentation in this model:

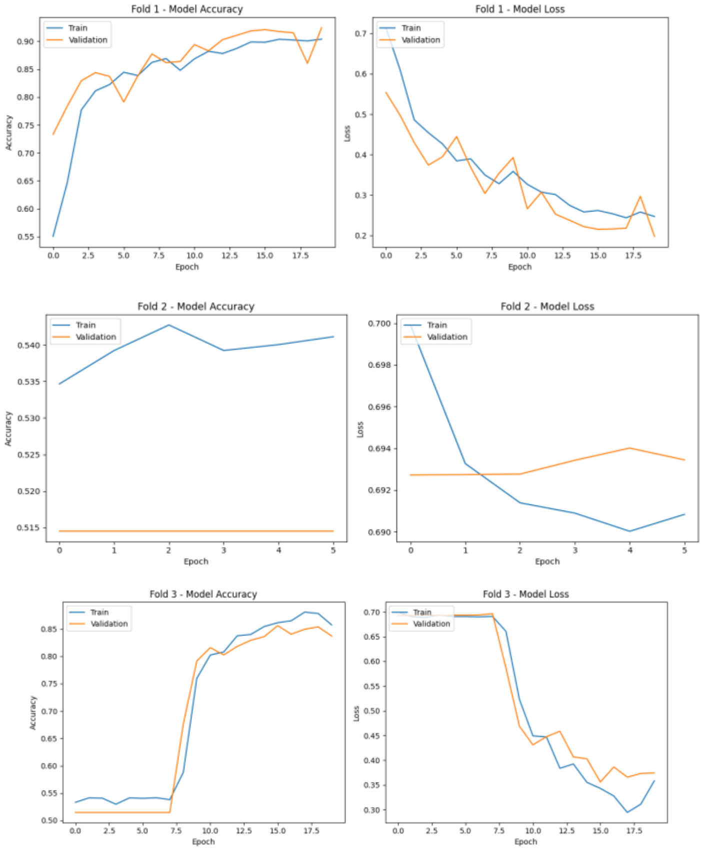
- Rescaling (0, 255)
- Rotation in a range [0, 180]
- Width (shifting horizontally) = 0.2
- Heights (shifting vertically) shift = 0.2
- Zoom range 0.2

The values we chose for the above variables are chosen randomly for this model. However, later according to their performance we'll perform hyperparameter tuning, hoping to get better results in the next model.

Since we are using 5-fold cross validation, we reduced epochs in this model to 20 to save time.

	Batch size	*Learning rate*	*Optimizer*	*Activation function*	*Epochs*	*K-fold cross validation*
Model properties	64	0.001	Adam	Relu, Sigmoid	20	Yes, K = 5

Figure 4.6 Model 3, 5 Folds Loss/ Accuracy plots



Fold 1 - Model Accuracy

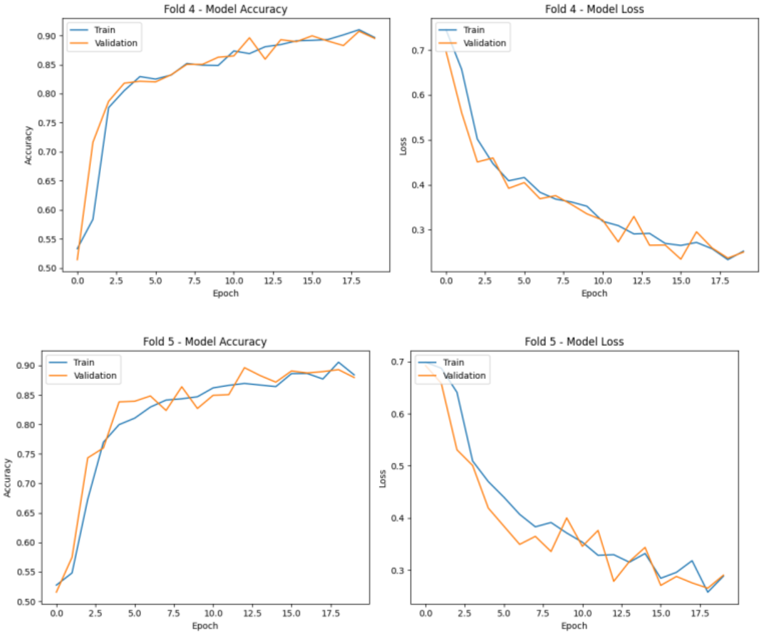
Fold 1 - Model Loss

Fold 2 - Model Accuracy

Fold 2 - Model Loss

Fold 3 - Model Accuracy

Fold 3 - Model Loss



Model 3 performance:

Average Train loss	0.2325
Average Train accuracy	0.9121
Test loss	0.2144
Test accuracy	0.9222

Across different cross-validation folds, Model 3 maintains stable accuracy levels, suggesting robustness in the training process. but there are signs of mild overfitting in some folds, as observable by the discrepancies between training and validation accuracies and rises in validation loss at the later stages of training.

4.4. Model 4

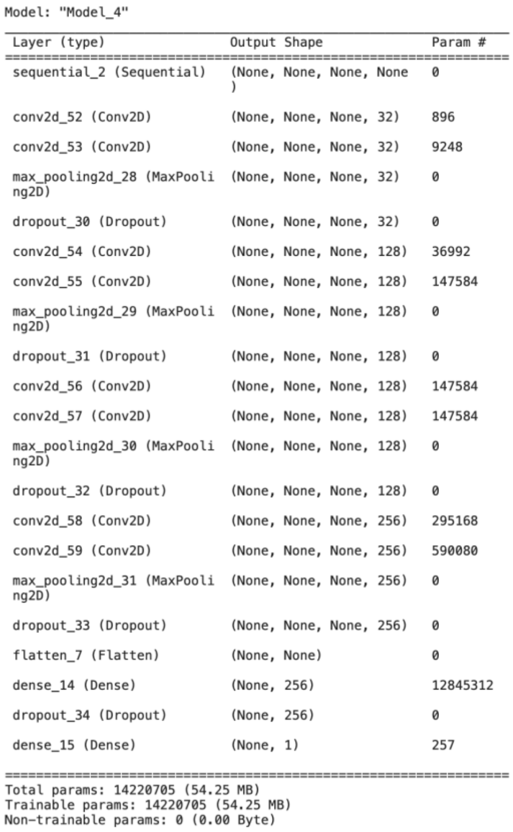
4.4.1. Model 4 architecture

This model is achieved after tuning the parameters of model 3 to get a better performance.

Using 4 convolutional blocks, each with two layers and a max pooling layer, we add a dropout layer to this model too. For the first two blocks the rate of dropout is 0.13 and for the 3rd and 4th blocks, the rate is 0.07 and 0.08 respectively. In the fully connected layers, we also have a dropout rate of 0.2 chosen among the three options, a dense layer with 256 units and one with 1 unit.

We keep on utilizing relu as activation function and sigmoid at the final layer.

And for data augmentation we have kept random flip horizontally, random rotation and random zoom are to be chosen after tuning.



```
Model: "Model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_2 (Sequential)   (None, None, None, None   0
                             )

 conv2d_52 (Conv2D)          (None, None, None, 32)    896

 conv2d_53 (Conv2D)          (None, None, None, 32)    9248

 max_pooling2d_28 (MaxPooli  (None, None, None, 32)    0
 ng2D)

 dropout_30 (Dropout)        (None, None, None, 32)    0

 conv2d_54 (Conv2D)          (None, None, None, 128)   36992

 conv2d_55 (Conv2D)          (None, None, None, 128)   147584

 max_pooling2d_29 (MaxPooli  (None, None, None, 128)   0
 ng2D)

 dropout_31 (Dropout)        (None, None, None, 128)   0

 conv2d_56 (Conv2D)          (None, None, None, 128)   147584

 conv2d_57 (Conv2D)          (None, None, None, 128)   147584

 max_pooling2d_30 (MaxPooli  (None, None, None, 128)   0
 ng2D)

 dropout_32 (Dropout)        (None, None, None, 128)   0

 conv2d_58 (Conv2D)          (None, None, None, 256)   295168

 conv2d_59 (Conv2D)          (None, None, None, 256)   590080

 max_pooling2d_31 (MaxPooli  (None, None, None, 256)   0
 ng2D)

 dropout_33 (Dropout)        (None, None, None, 256)   0

 flatten_7 (Flatten)         (None, None)              0

 dense_14 (Dense)            (None, 256)               12845312

 dropout_34 (Dropout)        (None, 256)               0

 dense_15 (Dense)            (None, 1)                 257

=================================================================
Total params: 14220705 (54.25 MB)
Trainable params: 14220705 (54.25 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 4.7 Model 4 summary

4.4.2. Hyperparameter tuning results

We have used a Baysian optimizer, our objective is validation accuracy and max trial is set to 20.

Here patience has been increased to 5.

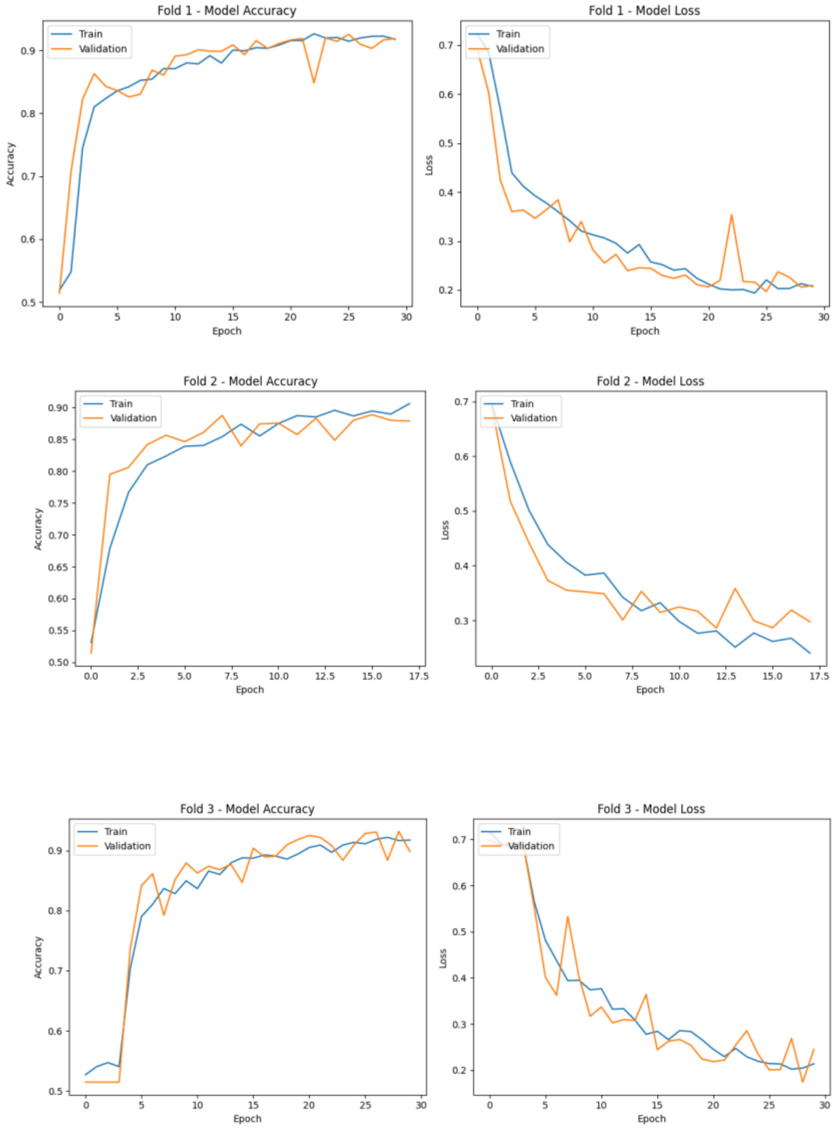
Below is the result of hyperparameter tuning performed on model 3 data augmentation to achieve model 4:

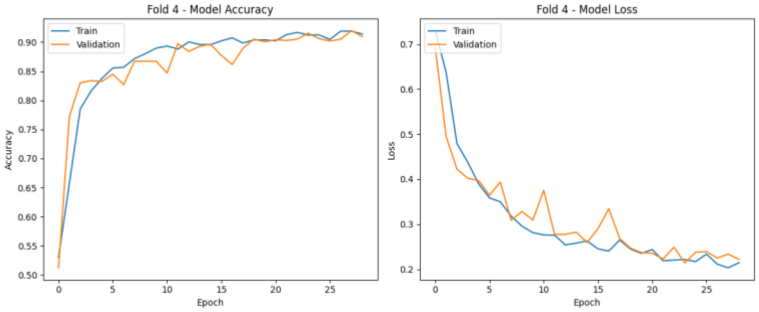
Hyperparameter tuning results	Range of searches:	Tuned value:
Parameter:		
Random Rotation	0.01 - 0.3	0.17
Random Zoom	0.01 - 0.3	0.11
Num_filter_b1	32, 64, 128, 256	32
Dropout_block_b1	0.2 - 0.4	0.13
Num_filter_b2	32, 64, 128, 256	128
Dropout_block_b2	0.2 - 0.4	0.13
Num_filter_b3	32, 64, 128, 256	0.07
Num_filter_b4	32, 64, 128, 256	256
Dropout_block_b4	0.2 - 0.4	0.08
Num_fc	128, 256, 512	256
Dropout_fc	0.2 - 0.4	0.2
Learning rate	1e-2, 1e-3, 1e-4	0.001

4.4.3. Model 4 Performance analysis

	Batch size	*Learning rate*	*Optimizer*	*Activation function*	*Epochs*	*K-fold cross validation*
Model properties	64	0.001	Adam	Relu, Sigmoid	30	Yes K = 5

Figure 4.8 Model 4's 5 Folds Loss/ Accuracy plots





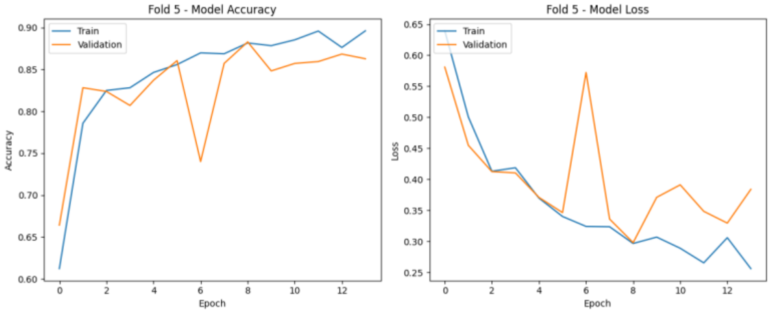
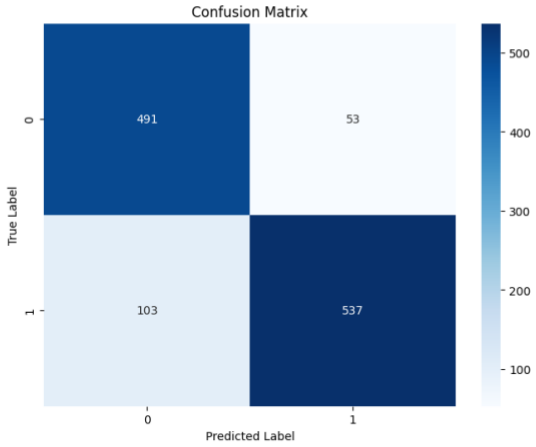


Figure 4.9 Model 4 confusion matrix



Model 4 performance:

Average Train loss	0.2701
Average Train accuracy	0.8939
Test loss	0.3499
Test accuracy	0.8682

Model 4 demonstrates little signs of overfitting. The model's performance is consistent, with small gaps between training and validation results, indicating that while there are areas for improvement, the issue of overfitting is not severe.

4.5. Model 5: Hyperparameter tuning for Xception, Inception and EfficientNetB0 pre-trained models

4.5.1 Xception architecture

It is a pre-trained CNN model designed for image classification. It mainly follows the principles of Inception architecture with some changes.

Its main properties are reduction of computation cost by reducing the number of parameters.

4.5.2 InceptionV3 architecture

This pre-trained model is part of the Inception family developed by Google.
It is known for achieving high accuracy on image classification tasks, with low computation costs. Its main characteristics are:

- Factorized convolution: breaking convolution into smaller, more manageable parts.
- Auxiliary classifier: help to mitigate gradient vanishing issue by intermediate outputs.
- Efficient grid size reduction: while remaining high accuracy, making sure to keep the model computationally efficient

4.5.3. EfficientNetB0

It is another CNN pre-trained architecture that achieves accuracy with less computation and fewer parameters while requiring lower resources. It uses:

- Compound scaling: balances increase in depth, width and resolution keeping efficiency.
- MBConv Blocks: efficient blocks using depth wise separable convolutions.
- Swish activation: this activation is expected to perform better than traditional relu.

All of these models are commonly used for transfer learning in image classification and recognition tasks since they have robust architectures, and they contain pre-trained weights from having been trained on large datasets such as ImageNe and their initial weights are optimized for a wide variety of image classification tasks.

The first two architectures use several default parameters values that are reported in the table below:

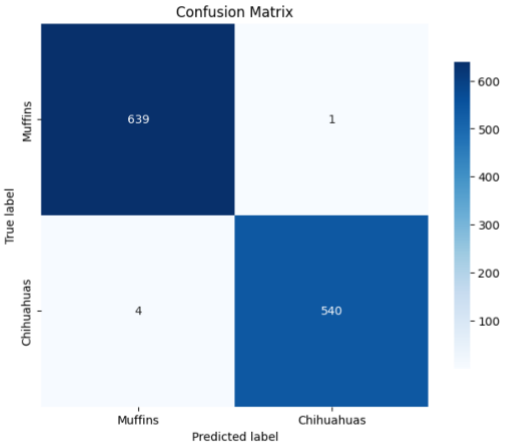
Input shape	Activation functions	Batch normaliz ation	Pooling layer	Optimizer	Learning rate	Loss function	Dropout rate
(299, 299, 3) Our input shape: (224, 224, 3)	Relu	yes	Yes	Adam	0.001	Categorical cross entropy	0.5

Rescaling the input data augmentation:	
Rescale	1./255
Rotation_range	20
Width_shift_range	0.2
Height_shift_range	0.2
Zoom_range	0.2

After 20 trials the below information has been gained:

Best Model	Activation	Dropout	Learning rate
Xception	Relu	False	0.0003

Figure 4.10 Model 5 confusion matrix



Model 5 performance:

Train loss	0.0633
Train accuracy	0.9911
Test loss	0.0456
Test accuracy	0.989

Model 5 consistently delivers high accuracy and maintains low loss across all evaluation metrics and folds, indicative of excellent performance. Despite its robustness, there are subtle variations in loss metrics across different folds, suggesting a slight sensitivity to specific subsets of validation data. These minor inconsistencies point to a potential for overfitting, though the overall impact remains low.

Chapter 5: Conclusion

After extensive model selection and model performance analysis, we have reached the following conclusions regarding the performance of our image classifier:

The performance metrics that has been used is model accuracy based on train and test values.

And we know that the higher training accuracy means that the model is learning our data better and more effectively, consequently we expect a better performance on test data and a better prediction. However sometimes overfitting occurs. Meaning that the model has learnt too well the train data but does not perform well on the test data since it has not learnt to be able to generalize. This issue has been our concern through this project. So it is very important for us to make sure that our preferred model gives a proper test and train accuracy value, because at the end we would like a model that can generalize well on the unseen data (test data), demonstrating the robustness of the model.

The confusion matrix shows that our model correctly classified that data with minimal misclassification.

For this objective we have worked on 5 different models

Model 1 and 2 are made from scratch, they are similar to each other, other than the fact that model two has two extra convolutional layers.

Model 3, also built from scratch, becomes a little bit more complicated, having an augmentation layer and using 5-fold cross validation and adding a dropout layer. it performs better.

Model 4 is achieved after performing hyperparameter tuning on model 3, and we expect to observe the effectiveness of tuning and how it shifts the model’s accuracy.

Finally, we use a 5th model, but for this one we rely on a pre-trained architecture called Xception.

We have used zero one loss to evaluate the model’s performance shown through the confusion matrix.

And it gives us the highest accuracy and if we were to choose one model to further make predictions with, this model would have been our choice.

Our chosen classifier has shown excellent performance, achieving good accuracy on both the training and test datasets.

The low misclassification rate, provided through the heatmap plot, further explains the effectiveness of our model in distinguishing between muffins and chihuahuas.

These results validate the efficacy of our approach and highlight the potential of deep learning models in complex image classification tasks.

With more refinement and fine-tuning, our model could find practical applications in various domains, ranging from animal recognition to food identification.