Binary Image Classification for Muffins and Chihuahuas Images

Ali Safaei

August 2023

Abstract

Convolutional Neural Networks architectures are widely used for different tasks in computer vision. In this project, I have used six different Convolutional Neural Networks models for the binary image classification of muffins and chihuahuas images. The performance of different models with different number of convolutional layers have been analysed. Also, the effect of using dropout technique and data augmentation technique on the performance of a model have been investigated. Furthermore, I performed a hyperparameter tuning process for a model to obtain the best combination of hyperparameters to improve the performance of that model. Additionally, to apply a transfer learning approach, a pre-trained VGG16 model has been used to build a model in this project. Finally, I used 5-fold cross validation method to evaluate the performance of the highest performing model. According to the results, the highest performing model in this project has 98.23% accuracy on the test set, and 97.46% average validation accuracy obtained from 5-fold cross validation method.

1.Introduction

Computer vision is an interesting field that has made significant progress in recent years due to the advancements and developments in machine learning algorithms. Nowadays, computer vision applications can be observed in different domains like medical imaging and diagnosing, autonomous vehicles, facial recognition systems, and more. There are various kinds of tasks in the field of computer vision that are used for different systems and some important ones include image classification, object detection, object tracking, semantic segmentation, etc.

In this project, our aim is to design a binary image classifier capable of classifying images into two different categories of "muffins" and "chihuahuas". All the introduced models are built based on different Convolutional Neural Networks architectures. Six different models have been introduced. The first three models have different depths, and all are inspired from the popular VGG16 architecture. Model 3 performs the best among the first three models but unfortunately, all three models have the problem of overfitting. To create Model 4, data augmentation and dropout layers were added to the Model 3, resulting in a model with reduced overfitting. After that, I conducted a hyperparameter tuning process for the fourth model to find the optimal hyperparameters. Therefore, the fifth model is the tuned version of the fourth model. The fifth model outperformed the fourth model on the test set. Finally, for the last model, I employed transfer learning by utilizing VGG16 pretrained model. Additionally, data augmentation and dropout layers were used for the last model.

This report is organized as follows: Section 2 provides a description of the dataset and its preprocessing. Section 3 presents some of the theoretical concepts and the model components. Section 4 elaborates on all the six different models, their experimental results and analysis. Finally, Section 5 presents the project's conclusion.

2. Dataset and Preprocessing

The dataset, which is an available dataset on the Kaggle website, consists of 5917 images of both muffins and chihuahuas, and all images are labelled. For this project, all the images have been converted from JPEG to RGB and for the normalization purpose all the pixel values for each RGB channel have been scaled from [0, 255] to [0, 1]. Also, each channel in RGB has a shape of 224 pixel by 224 pixel which is the common size used for the VGG16 architecture.

3. Theoretical Concepts and Models Components

Different kinds of Convolutional Neural Network architectures have been used in this project. Each architecture consists of different types of layers, including convolutional layer, max pooling layer, flatten layer, dense layer, data augmentation layer and dropout layer. This section provides a conceptual explanation for all the mentioned types of layers. Additionally, it presents explanations for the loss function, activation functions, and optimizer utilized in this project.

3.1. Convolutional Layer

In this layer, we perform a specific operation called "convolution" on the input of the layer. Then, the convolution result will be passed through an activation function and the output of that activation function, which is the final output of this layer, is called "feature map". In convolution operation, we slide a filter over the input, and for each specific position of the input we perform the element-wise product between the filter and the input. After that, we sum up the results of these element-wise products and the result is a specific value for that specific position.

Each filter is a matrix with some learnable elements (weights). These matrix elements will be adjusted in the training process to detect different patterns in the input data. The size of each filter is an important parameter which can have different values for different architectures. In this project, all the filters have the same size equal to (3, 3). We can use different number of filters in each convolution layer, the higher the number of filters the deeper the output of the convolutional layer.

Before starting the convolution operation, we can use padding for our input. Padding is a technique which is used to prevent information loss during the convolution operation. In this technique, we add extra border pixels to our input image and by convention the padded pixels are equal to zero. If we do not use padding, the spatial dimensions of the output will be reduced compared to the input. We have two different kind of convolution operation, first one is called "Same" convolution and the second one is called "Valid" convolution. For the "Same" convolution, we use padding for the input to have an output with the same spatial dimensions as input. For the "Valid" convolution, we do not use padding and the spatial dimensions of the output will be reduced in compare with input. In this project, all the convolution operations are "Same" convolution.

Another important factor for the convolution operation is stride. Stride specifies the step size at which a filter moves across the input during a convolution operation. In this project, the stride for the convolutional layer is equal to 1.

After the convolution operation, we apply an activation function in an element-wise way to the output of each filter. By using an activation function, we add non-linearity to our computations. Different kind of activation functions can be used. In this project, ReLU activation function has been used for all the convolutional layers. The mathematical function for the ReLU activation function is described below:

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

x is the input value for this activation function.

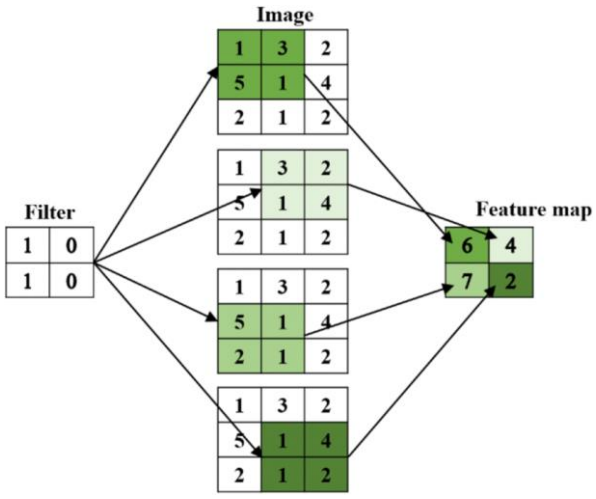

Figure 1[1]: Convolution operation example

[1] Figure 1 has been taken from this paper: Hübner, H.B., Duarte, M.A.V. & da Silva, R.B. Automatic grinding burn recognition based on time-frequency analysis and convolutional neural networks. *Int J Adv Manuf Technol* **110**, 1833–1849 (2020). https://doi.org/10.1007/s00170-020-05902-w

3.2. Max Pooling Layer

In the max pooling layer, we use a downsampling technique to reduce the spatial dimensions of a feature map. This dimensionality reduction decreases the memory usage and results in lower computational load. By applying this technique, a pooling window with specific dimensions slides over a feature map with a specific stride. At each step of this process, the pooling window selects the pixel with the highest value and discards the rest, resulting in keeping specific part of the information and discarding others. In this project, the max pooling window has a size of (2, 2) and the stride value is 2.

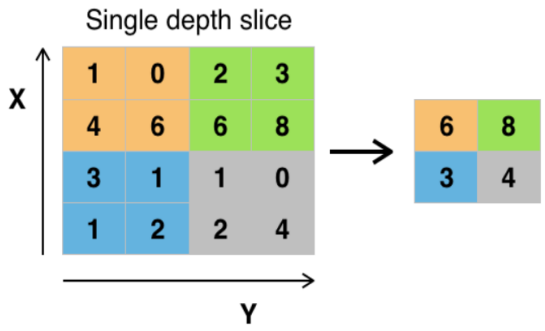

Figure 2[2]: Max pooling with a (2, 2) window and stride = 2

3.3. Flatten Layer

In a Convolutional Neural Network architecture, after having the last convolutional layer followed by a max pooling layer, we use a flatten layer to change the dimension of the max pooling layer output to a single dimension tensor. Then, the output of the flatten layer which is a single dimension tensor will be the input for a dense layer. The flatten layer is used to prepare the input for the dense layer.

3.4. Dense Layer

Dense layer, also known as fully connected layer, is a layer where each neuron is connected to every neuron from the previous layer. Dense layers are used in the Convolutional Neural Network architecture to learn features extracted by the previous layers. In this project, two dense layers are used. The first one takes its input from the flatten layer and has multiple units. Also, the kind of activation function used in this layer is the ReLU activation function. The second dense layer has one unit and uses a Sigmoid activation function to determine the result of our classification task. The mathematical function for the Sigmoid activation function is described below:

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

x is the input value for this activation function.

[2] Figure 2 is taken from this webpage: https://en.wikipedia.org/wiki/Convolutional_neural_network

The Sigmoid activation function is a non-linear function which has an output between 0 and 1. This activation function is used in the last layer of our binary image classifier to determine the probability of the input image belonging to one of the two classes.

3.5. Dropout Layer

To prevent the model from overfitting, we can apply the dropout technique for the neural network by using dropout layers. By using this technique, in each iteration of the training process, we randomly dropout some neurons in a specific layer of the neural network to add some noise to our model. The dropout rate for a specific layer in a neural network is a number between 0 and 1, determining the fraction of neurons that are supposed to be dropped out in each training iteration. As an example, for a layer in the neural network, a dropout rate equal to 0.1 means that 0.1 of the neurons will be randomly dropped out during each training iteration.

3.6. Data Augmentation Layer

To increase the diversity of our training set, we can apply different transformations to each image from this dataset and add the transformed images to it. By generating new images for the training set, we can reduce overfitting and create a model that is able to generalize better. The data augmentation layer used in this project applies different transformations including Horizontal Flip, Random Rotation (with a specific factor) and Random Zoom (with a specific factor) to the images from the training set to generate new data for the training set.

3.7. Loss Function

In this project, the task is binary image classification, so the binary cross entropy loss function has been used. A mathematical description for this loss function is presented below:

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

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

3.8. Optimizer

We should use an optimizer to find the optimal set of model parameters that minimize the loss function. For this project, the Adam (Adaptive Moment Estimation) Optimizer, which has the advantages of both RMSprop Optimizer and AdaGrad Optimizer, is used. The Adam Optimizer is a fast-converging optimizer and is widely used for problems that have large number of parameters. It performs efficiently in terms of both computational load and memory consumption. This optimizer has different parameters. In this project, excluding the learning rate parameter, the other parameters are always set to their default values.

3.9. Early Stopping

To prevent a model from overfitting, we can use a technique which is called early stopping. In this technique, we constantly monitor a specific metric in our training process and if that metric does not improve after some numbers of epochs the training process will be stopped. A parameter called "patience" determines the number of epochs the training process waits to observe an improvement for the monitored metric.

In this project, excluding section 4.7, the early stopping technique has always been used. The monitored metric is "val_loss", representing the validation loss, and the patience value is 5.

4. Models Development and Performance Analysis

4.1. Model 1

4.1.1. Architecture

This model is the simplest model introduced in this project. The order of layers for this model has been described below sequentially:

1. Two convolutional layers, each with 32 filters and ReLU activation function
2. Max pooling layer
3. Two convolutional layers, each with 64 filters and ReLU activation function
4. Max pooling layer
5. Flatten layer
6. Dense layer with 128 units and ReLU activation function
7. Dense layer with one unit and Sigmoid activation function

4.1.2. Model Training and Performance Analysis

In the training process of this model, the batch size is 32 and the learning rate for the Adam Optimizer is 0.001. The number of epochs for the training process is set to 50 but due to the use of early stopping technique the training process stopped after 7 epochs.

As you can see in figure 3 and table 1, the model performs excellently on the training set while its performance on the test set is very different, with much higher loss and lower accuracy. Consequently, the model has the overfitting problem, and it does not generalize well.

Train Loss	Train Accuracy	Test Loss	Test Accuracy
0.0302	0.9872	0.6360	0.8623

Table 1: Model 1 performance on the training set and the test set after 7 epochs

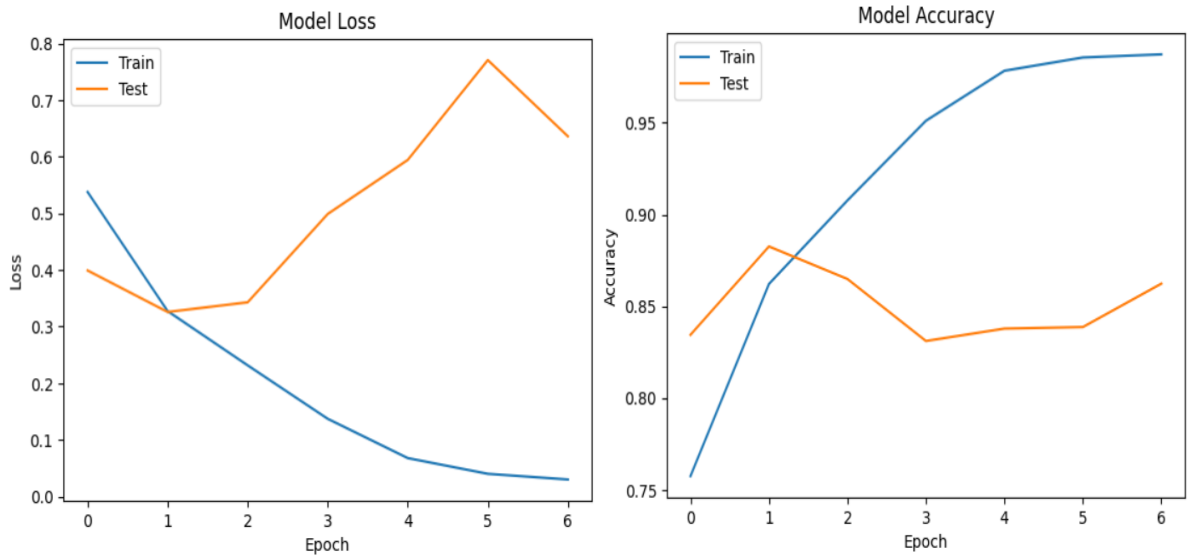

Figure 3: Model 1 performance on the training set and the test set at each epoch

4.2. Model 2

4.2.1. Architecture

To create this model, two convolutional layers, each with 128 filters, followed by one max pooling layer have been added to the Model 1. These layers were added to investigate the performance of an architecture with more convolutional layers. The order of layers for this model has been described below sequentially:

1. Two convolutional layers, each with 32 filters and ReLU activation function
2. Max pooling layer
3. Two convolutional layers, each with 64 filters and ReLU activation function
4. Max pooling layer
5. Two convolutional layers, each with 128 filters and ReLU activation function
6. Max pooling layer
7. Flatten layer
8. Dense layer with 128 units and ReLU activation function
9. Dense layer with one unit and Sigmoid activation function

4.2.2. Model Training and Performance Analysis

In the training process of this model, the batch size is 32 and the learning rate for the Adam Optimizer is 0.001. The number of epochs for the training process is set to 50 but due to the use of early stopping technique the training process stopped after 10 epochs.

The model performance has been reported in figure 4 and table 2. Based on the obtained results, the Model 2 has a better performance on the test set compared to the Model 1.

Similar to the Model 1, the Model 2 has a great performance on the training set while its performance on the test set is significantly different. As a result, we can conclude that this model has the overfitting problem.

Train Loss	Train Accuracy	Test Loss	Test Accuracy
0.0785	0.9715	0.4760	0.8792

Table 2: Model 2 performance on the training set and the test set after 10 epochs

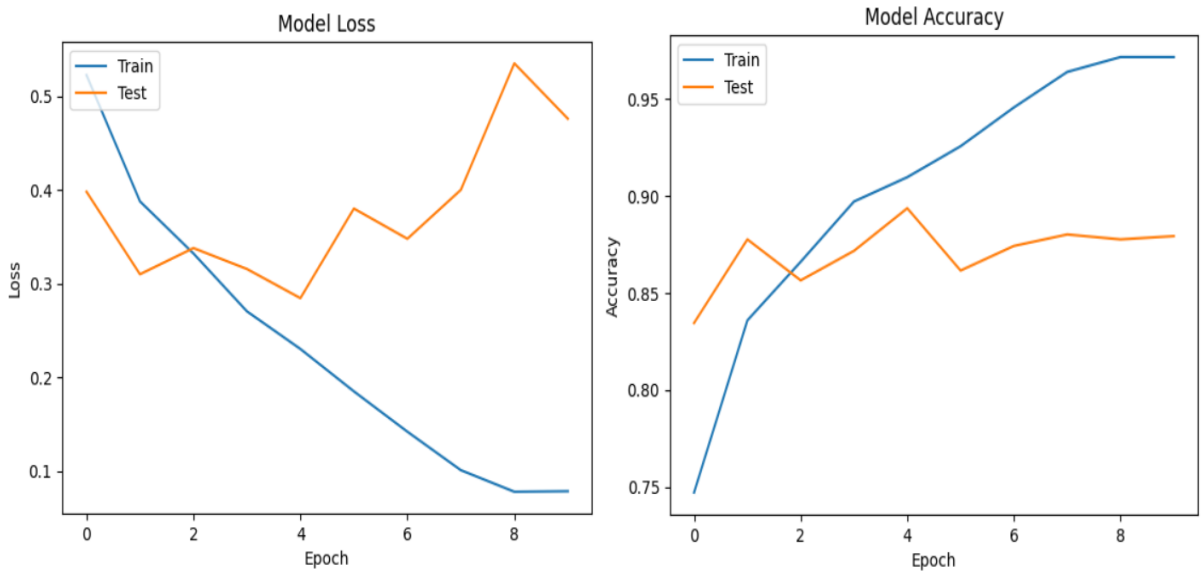

Figure 4: Model 2 performance on the training set and the test set at each epoch

4.3. Model 3

4.3.1. Architecture

Model 3 is created by adding 2 additional convolutional layers, each with 256 filters, followed by one max pooling layer. These layers were added to investigate the performance of an architecture with more convolutional layers. The order of layers for this model has been described below sequentially:

1. Two convolutional layers, each with 32 filters and ReLU activation function

2. Max pooling layer

3. Two convolutional layers, each with 64 filters and ReLU activation function

4. Max pooling layer

5. Two convolutional layers, each with 128 filters and ReLU activation function

6. Max pooling layer

7. Two convolutional layers, each with 256 filters and ReLU activation function

8. Max pooling layer

9. Flatten layer

10. Dense layer with 128 units and ReLU activation function

11. Dense layer with one unit and Sigmoid activation function

4.3.2. Model Training and Performance Analysis

In the training process of this model, the batch size is 32 and the learning rate for the Adam Optimizer is 0.001. The number of epochs for the training process is set to 50 but due to the use of early stopping technique the training process stopped after 13 epochs.

If you check out the results obtained for the performance of this model in figure 5 and table 3, you will find out that this model performs better than Model 2 on the test set, but it still has the overfitting problem.

Train Loss	Train Accuracy	Test Loss	Test Accuracy
0.1312	0.9553	0.3894	0.8851

Table 3: Model 3 performance on the training set and the test set after 13 epochs

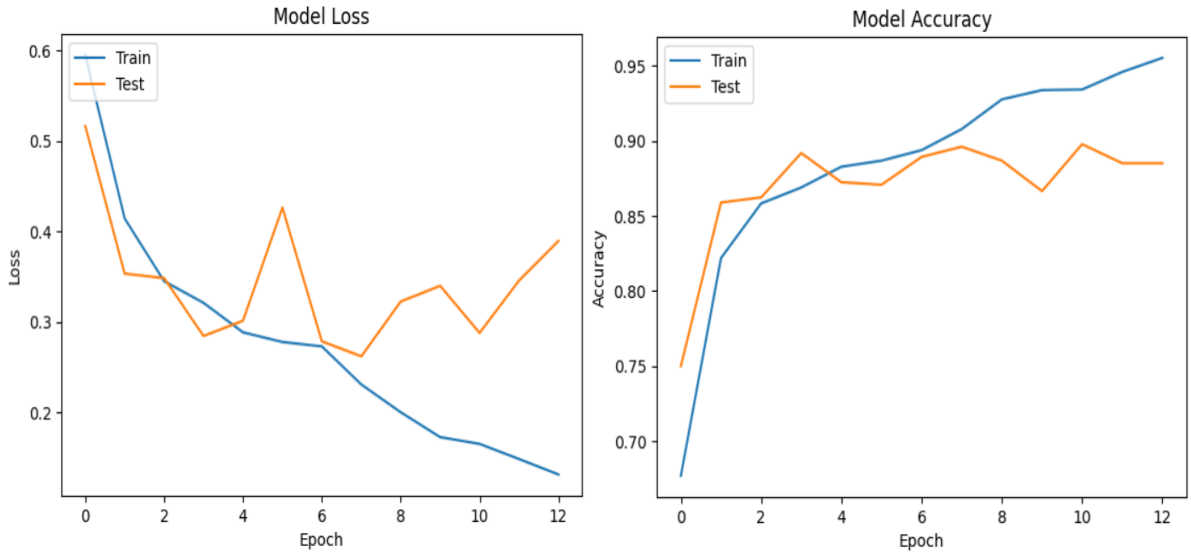

Figure 5: Model 3 performance on the training set and the test set at each epoch

4.4. Model 4

4.4.1. Architecture

Based on the previous obtained results, all the previous models have the overfitting problem, and the Model 3 has the best performance on the test set among them. To create Model 4, a data augmentation layer and some dropout layers have been added to Model 3. These layers were added to investigate their effect on the overfitting problem. The order of layers for this model has been described below sequentially:

1. Data augmentation layer with Horizontal Flip, Random Rotation with a factor of 0.05 and Random Zoom with a factor of 0.05

2. Two convolutional layers, each with 32 filters and ReLU activation function

3. Max pooling layer

4. Dropout layer with a dropout rate equal to 0.1

5. Two convolutional layers, each with 64 filters and ReLU activation function

6. Max pooling layer

7. Dropout layer with a dropout rate equal to 0.1

8. Two convolutional layers, each with 128 filters and ReLU activation function

9. Max pooling layer

10. Dropout layer with a dropout rate equal to 0.1

11. Two convolutional layers, each with 256 filters and ReLU activation function

12. Max pooling layer

13. Dropout layer with a dropout rate equal to 0.1

14. Flatten layer

15. Dense layer with 128 units and ReLU activation function

16. Dropout layer with a dropout rate equal to 0.25

17. Dense layer with one unit and Sigmoid activation function

4.4.2. Model Training and Performance Analysis

In the training process of this model, the batch size is 32 and the learning rate for the Adam Optimizer is 0.001. The number of epochs for the training process is set to 50 but due to the use of early stopping technique the training process stopped after 17 epochs.

As you can see in the results which are reported in table 4 and figure 6, the model performance on the test set is not much different from the model performance on the training set.

Based on the obtained results for the Model 3 and the Model 4, it can be concluded that the Model 4 has less variance error in compare with the Model 3. So, adding data augmentation layer and dropout layers to the Model 3, resulted in a model with reduced overfitting.

The performance of the Model 4 on the test set is better than the Model 3 but still the accuracy on the test set is not high and it is under 0.90.

Train Loss	Train Accuracy	Test Loss	Test Accuracy
0.2042	0.9190	0.2801	0.8902

Table 4: Model 4 performance on the training set and the test set after 17 epochs

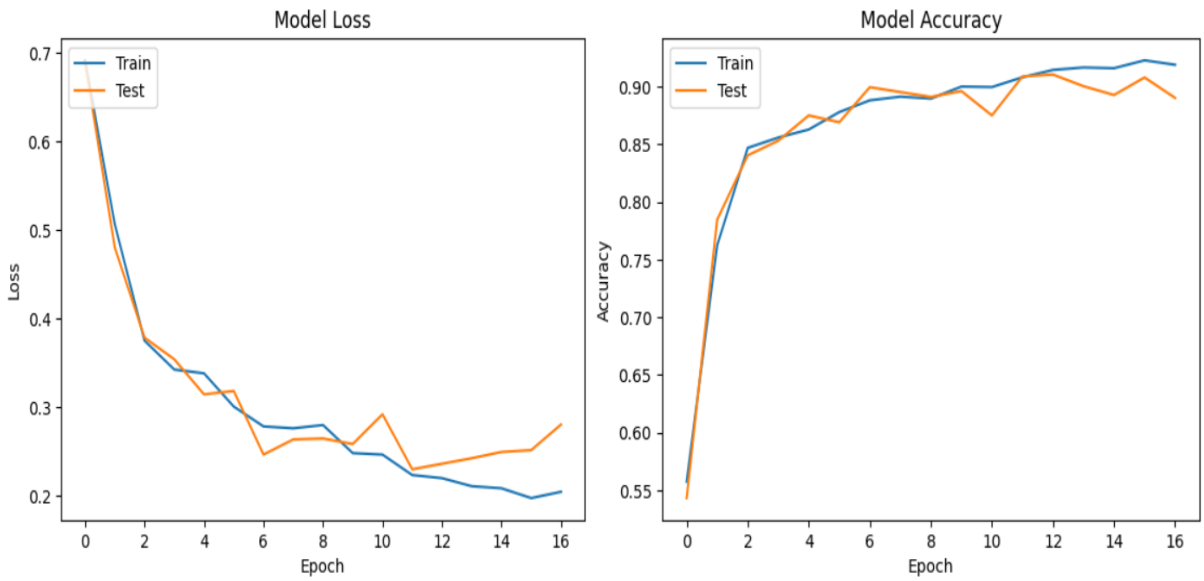

Figure 6: Model 4 performance on the training set and the test set at each epoch

4.5. Model 5

4.5.1. Hyperparameter Tuning for Model 4

To improve the performance of Model 4, we can tune some of the model hyperparameters. There are different hyperparameters in this model which can have a wide variety of values. In this section, I tried to tune some of these hyperparameters by defining a specific search space for them.

In the hyperparameter tuning process, 80% of the training set is used to train the models which have different hyperparameters and the other 20% of the training set is used to validate those models.

In the Model 4 architecture, there are 4 blocks between the data augmentation layer and the flatten layer. Each of these blocks is sequentially built by two convolutional layers with the same number of filters, one max pooling layer and one dropout layer.

In the Model 4, all the dropout layers used at the end of the mentioned blocks have the same dropout rate.

After the flatten layer, we have a dense layer which is followed by a dropout layer. This dropout layer has a different dropout rate in compare with the dropout layers which are used at the end of the previous mentioned blocks.

The model hyperparameters used in this hyperparameter tuning process and their specific search space have been described below:

1. Random Rotation factor in the data augmentation layer: Searching values between 0.01 and 0.2 with a step equal to 0.01

2. Random Zoom factor in the data augmentation layer: Searching values between 0.01 and 0.2 with a step equal to 0.01

3. Learning rate for the Adam Optimizer: Values are 0.01, 0.001, 0.0001

4. Number of filters in the convolutional layers used in the first block: Values are 32 and 64

5. Number of filters in the convolutional layers used in the second block: Values are 64 and 128

6. Number of filters in the convolutional layers used in the third block: Values are 128 and 256

7. Number of filters in the convolutional layers used in the fourth block: Values are 256 and 512

8. Number of units in the dense layer (fully connected layer): Values are 128, 256 and 512

9. Dropout rate for the dropout layer used in each block: Searching values between 0.05 and 0.15 with a step equal to 0.01

10. Dropout rate for the dropout layer used after the dense layer: Searching values between 0.2 and 0.4 with a step equal to 0.05

The best hyperparameters obtained from the hyperparameter tuning process are mentioned below:

1. Random Rotation factor in the data augmentation layer: 0.03

2. Random Zoom factor in the data augmentation layer: 0.17

3. Learning rate for the Adam Optimizer: 0.001

4. Number of filters in the convolutional layers used in the first block: 32

5. Number of filters in the convolutional layers used in the second block: 128

6. Number of filters in the convolutional layers used in the third block: 256

7. Number of filters in the convolutional layers used in the fourth block: 256

8. Number of units in the dense layer (fully connected layer): 512

9. Dropout rate for the dropout layer used in each block: 0.12

10. Dropout rate for the dropout layer used after the dense layer: 0.35

4.5.2. Architecture

Using the results obtained from the hyperparameter tuning process, the order of layers for this model has been described below sequentially:

1. Data augmentation layer with Horizontal Flip, Random Rotation with a factor of 0.03 and Random Zoom with a factor of 0.17

2. Two convolutional layers, each with 32 filters and ReLU activation function

3. Max pooling layer

4. Dropout layer with a dropout rate equal to 0.12

5. Two convolutional layers, each with 128 filters and ReLU activation function

6. Max pooling layer

7. Dropout layer with a dropout rate equal to 0.12

8. Two convolutional layers, each with 256 filters and ReLU activation function

9. Max pooling layer

10. Dropout layer with a dropout rate equal to 0.12

11. Two convolutional layers, each with 256 filters and ReLU activation function

12. Max pooling layer

13. Dropout layer with a dropout rate equal to 0.12

14. Flatten layer

15. Dense layer with 512 units and ReLU activation function

16. Dropout layer with a dropout rate equal to 0.35

17. Dense layer with one unit and Sigmoid activation function

4.5.3. Model Training and Performance Analysis

In the training process of this model, the batch size is 32 and the learning rate for the Adam Optimizer is 0.001. The number of epochs for the training process is set to 50 but due to the use of early stopping technique the training process stopped after 21 epochs.

According to the obtained results which are reported in figure 7 and table 5, the Model 5 outperforms the Model 4 on the test set, with higher accuracy and lower loss.

There is not a significant difference between the model performance on the test set and the model performance on the training set. Consequently, the model has a low variance error, and it generalizes well.

Train Loss	Train Accuracy	Test Loss	Test Accuracy
0.1497	0.9402	0.1866	0.9426

Table 5: Model 5 performance on the training set and the test set after 17 epochs

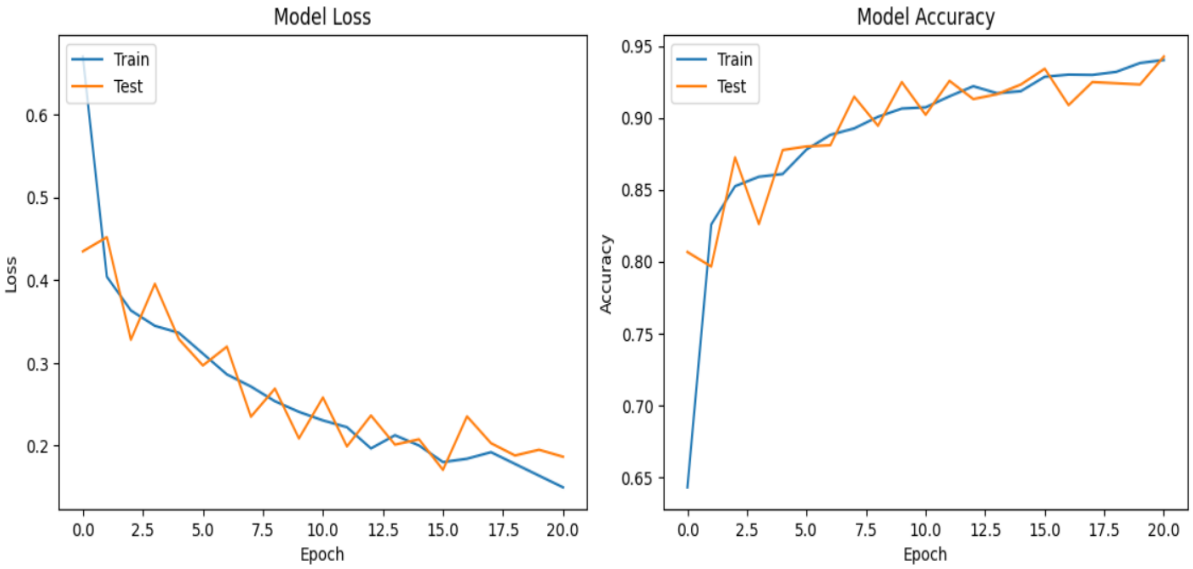

Figure 7: Model 5 performance on the training set and the test set at each epoch

4.6. Model 6

4.6.1. Pre-Trained VGG16 Model and Transfer Learning

VGG16 model is a powerful and popular model which is used for some of the computer vision applications. The order of layers for this model has been described below sequentially:

1. Two convolutional layers, each with 64 filters and ReLU activation function
2. Max pooling layer
3. Two convolutional layers, each with 128 filters and ReLU activation function
4. Max pooling layer
5. Three convolutional layers, each with 256 filters and ReLU activation function
6. Max pooling layer
7. Three convolutional layers, each with 512 filters and ReLU activation function
8. Max pooling layer
9. Three convolutional layers, each with 512 filters and ReLU activation function
10. Max pooling layer
11. Flatten layer

12. Two dense layers, each with 4096 units and ReLU activation function

13. Dense layer with 1000 units and Softmax activation function

In this project, a transfer learning approach using pre-trained VGG16 model has been used. The used pre-trained VGG16 model has been trained on a large dataset called ImageNet which contains millions of images.

First, to apply transfer learning, the flatten layer and the three fully connected layers located on the top of the pre-trained model were removed. Then, the parameters of the convolutional layers were frozen, which means that they are not trainable anymore. The obtained model with all of the mentioned changes is called the Base Model for the transfer learning.

In Model 6, the Base Model plays the role of a feature extractor which has already been trained to extract the features from images effectively. By using this approach, we don't have to train the deep convolutional layers from scratch on our dataset, and we can gain from the knowledge of the Base Model which has been trained on a much larger dataset in comparison with our dataset.

4.6.2. Architecture

The order of the components of this model has been described below sequentially:

1. Data augmentation layer with Horizontal Flip, Random Rotation with a factor of 0.05 and Random Zoom with a factor of 0.05

2. Base Model for the transfer learning

3. Flatten layer

4. Dense layer with 128 units and ReLU activation function

5. Dropout layer with a dropout rate equal to 0.25

6. Dense layer with one unit and Sigmoid activation function

4.6.3. Model Training and Performance Analysis

In the training process of this model, the batch size is 32 and the learning rate for the Adam Optimizer is 0.001. The number of epochs for the training process is set to 50 but due to the use of early stopping technique the training process stopped after 9 epochs.

Based on the results reported in figure 8 and table 6, the model performs excellently on the test set and its accuracy on the test set is more than 0.98. The results obtained for the Model 6 performance indicate that using a pre-trained model can help us to create a model with an outstanding performance on the test set. Also, the difference between the model performance on the test set and the model performance on the training set is not much and this shows that the model generalizes well.

Train Loss	Train Accuracy	Test Loss	Test Accuracy
0.0286	0.9889	0.0558	0.9823

Table 6: Model 6 performance on the training set and the test set after 9 epochs

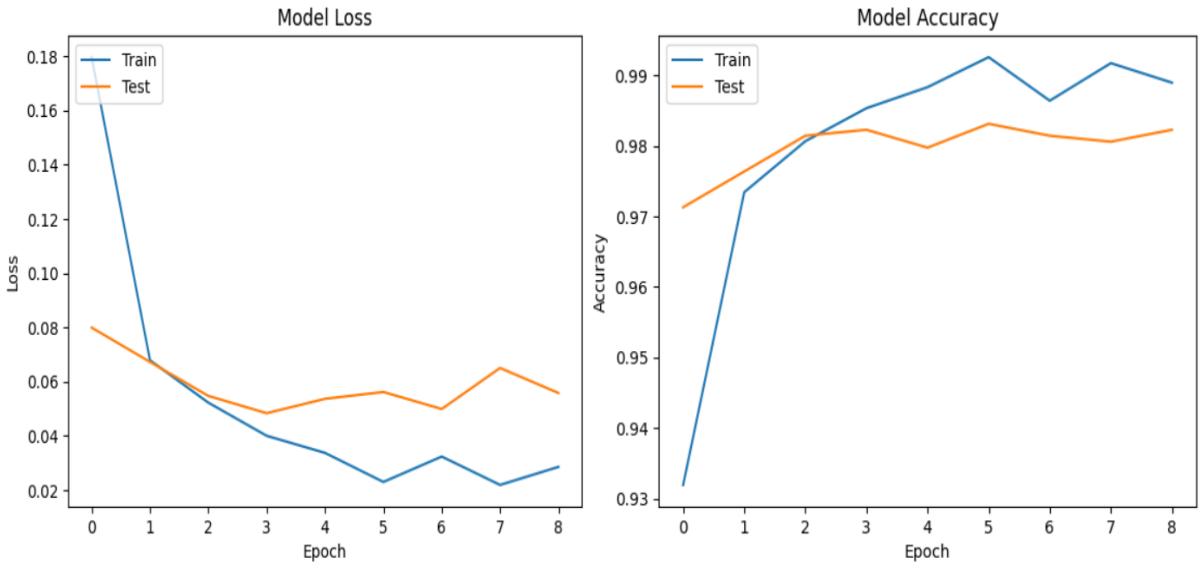

Figure 8: Model 6 performance on the training set and the test set at each epoch

4.7. 5-fold Cross Validation for Model 6

The Model 6 has the best performance among the presented models in this project. In this section, I used the 5-fold cross validation method to evaluate the performance of this model.

For the 5-fold cross validation method, we randomly divide our training set to 5 different subsets which have the same size. The process has 5 iterations. At each iteration, we use one of the 5 subsets as validation set and the other remaining 4 subsets to train the model. In the whole process, each subset will be used as validation set only once. Consequently, at this process we will have 5 different results and the final result will be the average of them.

For this method, in each training process, the batch size is 32, the learning rate for the Adam Optimizer is 0.001, and the number of epochs is set to 10. The early stopping technique has not been used in this section of the project.

You can find the obtained results of 5-fold cross validation method for the Model 6 in figure 9 and table 7.

Average Validation Loss	Average Validation Accuracy
0.0912	0.9746

Table 7: 5-fold cross validation results of Model 6

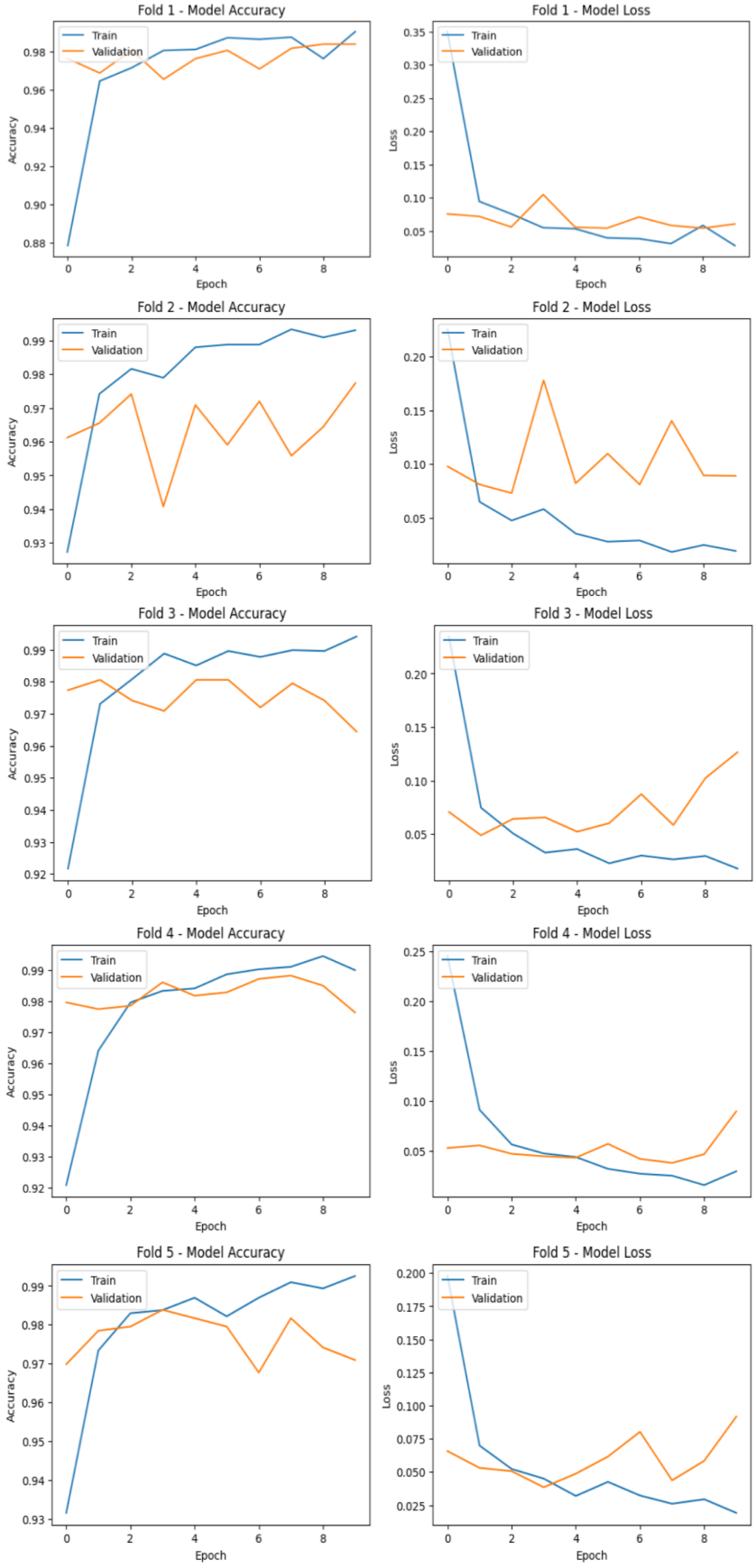

Figure 9: 5-fold cross validation results of Model 6 at each epoch

5. Conclusion

According to the presented models in this project and their corresponding results, we can conclude the following points:

1. Although adding more convolutional layers to a model can lead to better performance on the test set, without some techniques like dropout and data augmentation, we will probably end up with a model which does not generalize well and has a high variance error.

2. Using a pre-trained model like the pre-trained VGG16 can help us to create a generalized model with high performance on the test set. This approach is so effective especially when we have a small dataset, and we can gain from the knowledge obtained from a pre-trained model which has been already trained on a big dataset.