UNIVESITY OF MILAN
DATA SCIENCE AND ECONOMICS

Statistical Methods for Machine Learning
Course Project

Convolutional Neural Networks

Cats and Dogs Image Classification

Student: Uladzislau Luksha 964000
E-mail: Uladzislau.luksha@studenti.unimi.it

Department of Economics, Management and Quantitative Methods
January 2023

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Abstract

KEYWORDS: CONVOLUTIONAL NEURAL NETWORKS (CNN), IMAGE CLASSIFICATION, VISUAL GEOMETRY GROUP (VGG), HYPERPARAMETER TUNING

In this project, our goal was to classify images of cats and dogs with help of various types of Neural Networks. In order to preprocess the images, we downscale them to 100x100 or 150x150 pixels depending on the implemented model as well as normalized the data by dividing the pixel values by 255. The models under study are a simple convolutional neural network (CNN) as well as visual geometry group (VGG). We consider two variations of each model, for color and grayscale images. We selected the best color model based on its binary accuracy and binary cross-entropy, and then fine-tune its hyperparameters. The final model is validated using 5-fold cross-validation and evaluated on a testing set. Our experiments have shown that both VGG models used in the project achieved good results. Nevertheless, we chose to cross-validate the model with color input data because we believe that color information can increase classification accuracy on third-party data. The evaluation of our final model on test set showed the binary accuracy of 0.90 and binary cross-entropy of 0.33.

1 Introduction

In recent years, neural networks have gained significant attention for their ability to perform image classification tasks with high accuracy. Neural networks are algorithms that are designed to mimic the functioning of the human brain. They are made up of interconnected neurons, which serve as the network's basic building blocks. The number and type of neurons, as well as their connections, differ depending on the network type. Among the most varieties of neural networks (convolutional, recurrent, fully connected) our project focuses exclusively on the use of convolutional networks. Convolutional neural networks (CNNs) are a type of neural network that is designed specifically to process pixel data, such as images. They are made up of convolutional layers that extract features from images and pooling layers that reduce the spatial dimensions of the feature maps. CNNs are frequently used for image classification, object detection, and image segmentation. In this paper, we present a study on the use of neural networks for the classification of images of dogs and cats.

2 Data preprocessing and computation environment preparation

2.1 Dataset

The dataset used for the classification problem contains 25000 pictures of cats and dogs. The size of the dataset is 812.79MB. Images are stored in two folders with equal number of documents for cats and dogs. To load the dataset into the Google Collab programming environment, two options were chosen: unzipping the preloaded archive from Google Drive, as well as direct downloading of data from the source[1]. The latter option is needed for implementation without access to Google Drive, which is crucial for code replication

2.2 Computation environment

It is important to note that before we start assembling image classification models, we change the standard Google Collab computational environment to an environment using a hardware acceleration. A GPU (graphics processing unit) is a specialized processor that speeds up the processing of graphics and image data. A GPU has thousands of cores that can perform parallel computations, allowing it to outperform a CPU. The training and inference of a CNN model can be significantly faster with a GPU than with a CPU alone. In addition, to automatically optimize the performance of the data pipeline used for training, validation and test datasets, *'tensorflow.data.AUTOTUNE'* is used. Not all models or problems benefit from GPU and Autotune, so it is always a good idea to experiment and see how the model performs with and without these improvements. Looking ahead, these techniques are effective for our data because the average learning time per epoch has dropped from more than 3 minutes to only 20 seconds.

2.3 Corrupted images

Right from the start of the data processing, we ran into the problem of corrupted images. They caused a crashes of data augmentation. To detect corrupted images, we run the algorithm that opens every image file in our dataset in binary mode and reads the first 10 bytes of the file using the "peek()" method. It then checks if the image is in JPEG format by verifying if the string "JFIF" is present in the image's header. If it does not, the image is deemed corrupted, and the algorithm removes the image from the dataset. The number of corrupted images found is 1590. Now the remaining images (23410) are ready for the data augmentation.

[1] https://unimibox.unimi.it/index.php/s/eNGYGSYmqynNMqF

2.4 Data augmentation

Data augmentation is a technique for increasing the size of a dataset artificially by applying various types of transformations to the original images. Rotation, zooming, flipping, and cropping are several of the transformations available. This technique is used to prevent model overfitting by providing the model with more diverse data, as well as to make the model more robust to data variations. It can also be used to balance the dataset by adding more examples of under-represented classes. The main reasons for using data augmentation in this project is the assumption that the images of dogs and cats in the dataset may be biased towards specific poses. For example, it is likely that the owners of these pets will photograph them from similar angles, resulting in a lack of diversity in the dataset.

For this project, we have applied techniques of zooming - 20% of the processed image size in or out, rotation - 20% left or right, and horizontal flipping. We do not use vertical flip and brightness enhancement techniques, as this would further increase the computational cost of the model. Moreover, pictures of upside-down animals are not common in the real world. The example of the data augmentation result is shown in the Figure 1.

Figure 1: Example of the data augmentation application

2.5 Data preparation

In this paper, we used two models, a sequential Convolutional Neural Network (CNN) composed by the author and the Visual Geometry Group (VGG), on two different types of data: colored and grayscale. This is because color information in an image can play an important role in image classification tasks. However, color information can introduce noise into the dataset and increase the model's complexity. We hope to evaluate the impact of color information on model performance by using grayscale data. Depending on the models and colors, the format of the input data will change. For CNN models we reduce the picture size to 100x100 pixels, for VGG we downscale the picture size to 150x150 pixels. We cannot afford the bigger size of the input because of the computational intensity and the significant increase in model training time. Also, depending on the model variant (color

or grayscale) the input data shape will vary. For gray images the color will have just one dimension, while for color images the number of dimensions will be 3, one for each RGB channel.

The next step is to generate datasets. For grayscale images the dataset is stored in the list with each element of the list containing the resized image and its corresponding class number (0 for the cats and 1 for the dogs). Then we create separate arrays for features and labels for training, validation and test datasets in the proportions of 70%, 20%, 10%. For each array of features, we apply normalization by dividing their pixel values by 255. Thus, their normalized values will range from 0 to 1. After all manipulations with the data, the training set contains 16387 images (8230 images of cats and 8157 of dogs), the validation set contains 4682 images (2317 cats and 2365 dogs), and test set contains 2341 images (1194 cats and 1147 dogs).

For color images we use TensorFlow to create the dataset from a directory. The function $keras.preprocessing.image\_dataset\_from\_directory$ shuffles the data and divides it into training and validation sets. After that, we create a test set by separating it from the validation set. Thus, the data structure is the same as in the grayscale models - 70%, 20%, 10% for the training, validation, and test sets, respectively. Data normalization for colored images is performed within the implemented models. In datasets with color images, the number of documents is slightly different from those in grayscale. The data structure there is as follows: training set – 16387 images (8205 cats and 8182 dogs), validation set – 4687 (2316 cats and 2371 dogs), and test set – 2336 images (1206 cats and 1130 dogs).

It can be stated that even after applying preprocessing and removing corrupted pictures, the classes in the sets are sufficiently balanced that we can apply the accuracy and the binary loss measures to evaluate the electiveness of classification.

3 Models implemenation

Before describing in detail the architecture of the models used, it is appropriate to mention the basic blocks that make up neural networks. They are **convolutional layers, pooling layers, and fully connected layers**.

Convolutional layers are in charge of identifying patterns and features in the input image. They generate a feature map , also known as a neuron, by applying a set of learnable filters to the input image. These filters are small matrices that are applied to a specific region of the input image. **Pooling layers** are used to reduce the spatial size of the feature maps, which helps the model reduce the number of parameters and computations required. Max pooling and average pooling are two common pooling techniques. Max Pooling Layer shrinks feature maps by downsampling them to the

maximum value within a pooling window. This aids in avoiding overfitting. **Fully connected layers** are connected to all the neurons in the previous layer. They aid in the prediction of features learned by the convolutional and pooling layers. The final fully connected layer is frequently used to output the model's final predictions [1].

CNNs frequently include **normalization layers** (such as batch normalization) and non-linear activation functions (such as Rectified Linear Unit) in addition to these building blocks to introduce non-linearity and improve model performance. In our model we use only ReLU and sigmoid (for output data) activation functions. Auxiliary layers are also important: the **dropout laye**r randomly sets a fraction of input units to 0 at each update during training time, which aids in breaking feature co-adaptation. This makes the model less sensitive to the specific weights of neurons, which can help to reduce overfitting. **Flatten layer** is used to convert a multidimensional tensor to a one-dimensional tensor. It is commonly used prior to passing the output of convolutional layers to fully connected layers. The flatten layer aids in converting the outputs of the convolutional layers into a format that the fully connected layers can use. Because the fully connected layers are unable to process the convolutional layers' multi-dimensional tensor format, this step is required.

3.1 Sequential Convolutional Neural Network (CNN)

The first model used in this project is a simple sequential CNN. It consists of the following blocks:

1. Data augmentation

2. Rescaling (only for colored images, already performed for the grayscale data)

3. 2D Convolutional (64 filters, kernel size: (3,3), relu activation)

4. Max Pooling 2D (pool size: (2,2))

5. 2D Convolutional (64 filters, kernel size: (3,3), relu activation)

6. Max Pooling 2D (pool size: (2,2))

7. Flatten

8. Dense (64 output values, relu activation)

9. Dense (2 output values, sigmoid activation)

A detailed sequential model architecture used in the project (Figure 2).

This model uses an image of 100x100 pixels for input data. The batch size is 32. The binary cross-entropy loss function, the Adam optimizer, and the binary accuracy metric are being used to compile the model. The loss function is used to assess the model's performance during training. The binary cross-entropy loss function calculates the difference between the predicted and true probability distributions. The Adam optimizer adapts the learning rate for each parameter during training. We use a default value of Adam optimizer which is 0.001. The binary accuracy metric is used to

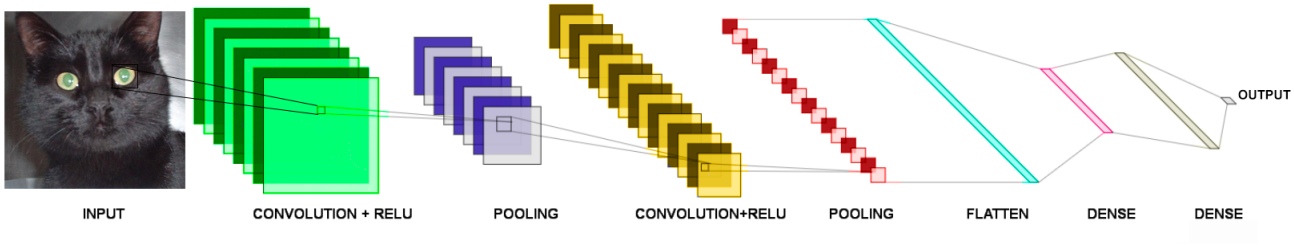

Figure 2: Convolutional Neural Network model architecture

evaluate the model's accuracy and performance. The number of epochs to train the model is set to 20. For all models used in the project, we use an early stop function with a patience threshold of 3. It means that the training process will be stopped if the validation loss does not decrease for 3 consecutive epochs, which will help to somewhat optimize the training process.

The performance of this CNN model on gray and color pictures are shown in Figures 3 and 4.

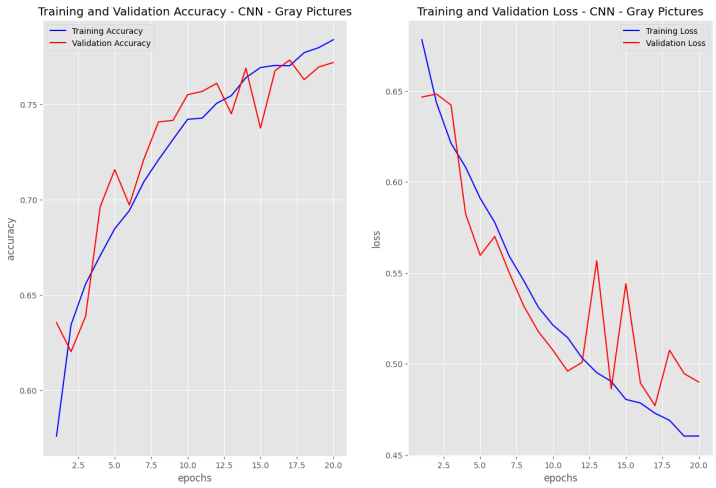

Figure 3: Training and validation binary accuracy and loss – CNN – gray images

The evaluation of the models:

Model	Training Accuracy	Validation Accuracy
CNN-gray	**0.8061**	**0.7719**
CNN-color	0.7619	0.7546

Model	Training Loss	Validation Loss
CNN-gray	**0.4297**	**0.4900**
CNN-color	0.4999	0.5045

According to the results of the first model, the neural network with grayscale pictures performed better than the CNN with colored ones. The training of the color model stopped already at the 15th epoch due to the absence of improvement. Moreover, it began to overfit from the 3rd epoch. As

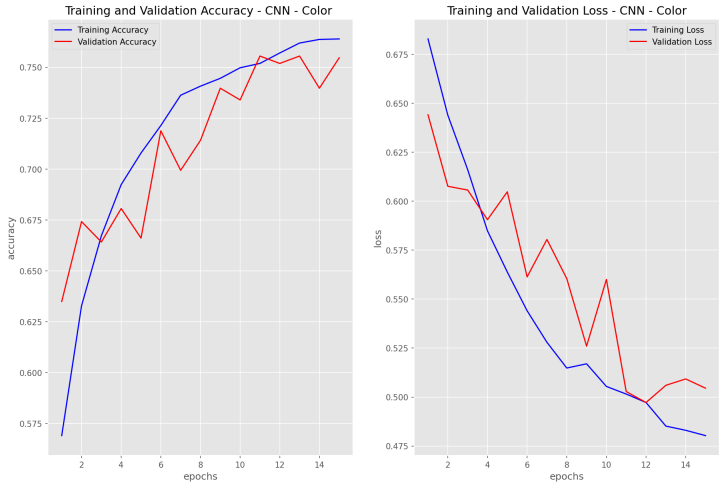

Figure 4: Training and validation binary accuracy and loss – CNN – color images

can be seen in Figure 3, the grayscale model also begins to overfit, starting at about the 11th epoch. These results are quite realistic because grayscale models may outperform color models in simple neural networks since they have less information to process and consequently are less susceptible to overfitting.

Now we will make the neural network architecture a bit more complex using a VGG model and repeat the comparison.

3.2 VGG

The Visual Geometry Group is a convolutional neural network that is known for producing cutting-edge results in image classification tasks. This model is considered a deep neural network [2], in this project, in addition to the 16 layers due to the architecture of this network, we purposely add a layer of dat augmentation, as well as re-scaling one. Hence, our VGG model should be perceived as a modified one. The model consists of the following blocks:

1. Data augmentation

2. Rescaling (only for colored images, already performed for the grayscale data)

3. 2D Convolutional (32 filters, kernel size: (3,3), stride = 1, relu activation)

4. Max Pooling 2D (pool size: (2,2))

5. 2D Convolutional (64 filters, kernel size: (3,3), stride = 1, relu activation)

6. Max Pooling 2D (pool size: (2,2))

7. 2D Convolutional (128 filters, kernel size: (3,3), stride = 1, relu activation)

8. Max Pooling 2D (pool size: (2,2))

9. 2D Convolutional (256 filters, kernel size: (3,3), stride = 1, relu activation)

10. Max Pooling 2D (pool size: (2,2))

11. Flatten

12. Dense (128 output values, relu activation)

13. Dropout (rate: 0.1)

14. Dense (128 output values, relu activation)

15. Dropout (rate: 0.1)

16. Dense (128 output values, relu activation)

17. Dropout (rate: 0.1)

18. Dense (2 output values, sigmoid activation)

This model uses an image of 150x150 pixels for input data. The batch size is 64. Exactly as in the first model we use the binary cross-entropy loss function, the default value (0.001) of the Adam optimizer, and the binary accuracy metric to compile the model. The number of epochs to train the model is set to 20. Stride is a neural network filter parameter that controls the amount of movement across the image. In this model we set its value to 1.

The performance of this VGG model on gray and color pictures is shown in Figures 5 and 6.

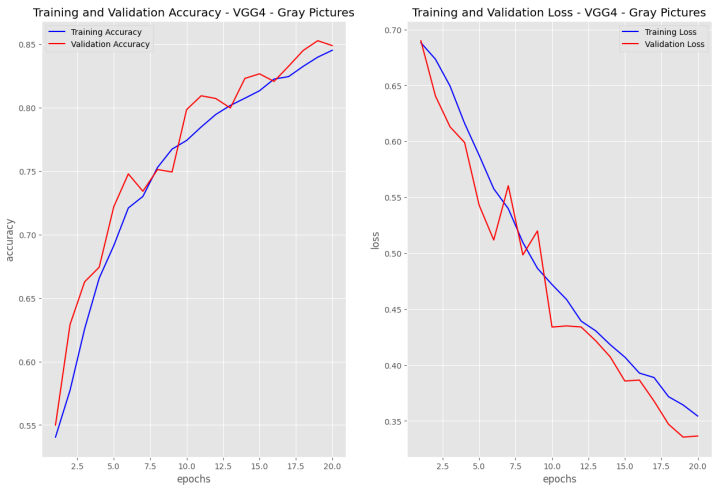

Figure 5: Training and validation binary accuracy and loss – VGG – gray images

The evaluation of the models:

Model	Training Accuracy	Validation Accuracy
VGG-gray	**0.8713**	**0.8488**
VGG-color	0.8343	0.8321

Model	Training Loss	Validation Loss
VGG-gray	**0.2984**	**0.3365**
VGG-color	0.3591	0.3783

Our second model performed much better for both grayscale and color pictures. Nevertheless, the unstable training and validation curves for both models may indicate their tendency to overfitting.

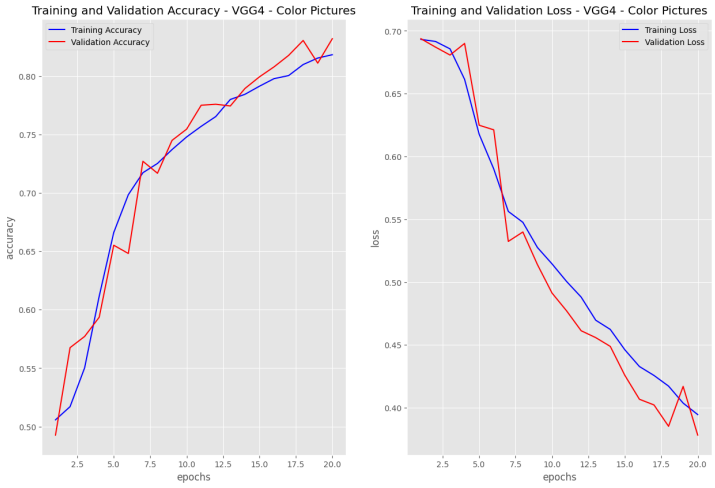

Figure 6: Training and validation binary accuracy and loss – VGG – color images

As in the previous group of CNN models, the best results are shown by the model with the input data of grayscale images. However, for further hyperparameters tuning and cross-validation, we will use a model with colored pictures. We believe that color information may be of a value for defining a certain pattern of a cat or dog. In addition, using color images in cross-validation allows for a more comprehensive evaluation of the model's performance because it is tested on a broader range of input data. It can also help to ensure that the model generalizes well to real-world images, which are usually in color.

Thus, we will use the color version of VGG model to tune the hyperparameters and perform cross-validation.

4 Hyperparameters tuning

Hyperparameter tuning is the process of systematically searching for the best combination of hyperparameters to optimize a machine learning model's performance. There are plenty of parameters to be tuned in case of convolutional neural networks but the most common parameters for tuning are the number of layers, the number of filters in each layer, the size of the filters, the stride, the padding, and the type of activation function. There are also several approaches for tuning. The most used algorithms are manual tuning, grid search, and random search. Manual tuning entails changing one hyperparameter at a time and assessing the model's performance after each change. Grid search entails specifying a value range for each hyperparameter and evaluating all possible combinations. Random search entails sampling random hyperparameter value combinations and evaluating the model's performance. Techniques such as Bayesian optimization are another popular method for tuning CNN's hyperparameters. Bayesian optimization forecasts the performance of

various hyperparameter settings using a probabilistic model.

We did not change the architecture of the neural network model and chose learning rate, number of neurons in the NN part, and dropout layers percentage as hyperparameters for tuning. Initially, we also wanted to test the activator function for the NN part, having tried Hyperbolic Tangent (Tanh) and ReLU but after noticing no visible improvements in the model, we decided to reduce hyperparameter space, leaving only the ReLU activator. Here are our tunable parameters and their ranges:

1. **Dropout layers percentage** (in the NN part): min testing value is 0.1, max testing value is 0.3, step is 0.1. [3 values];

2. **Learning rate**: min testing value is 0.0004, max testing value is 0.001, step is 0.0002 [4 values];

3. **Number of neurons in the NN part:** min testing value is 32, max testing value is 512, step is 32. [16 values].

Thus, the number of possible combinations of hyperparameter values is 192. For the tuning process, we use the random search algorithm which is built in *keras_tuner* module. We set the number of search trials equal to 15 with 15 epochs for each trial. After executing the random search algorithm, the model with 480 neurons in the NN part, a dropout of 20%, and a learning rate of 0.0004 was considered to be the best. Using these parameters, we proceed to the cross-validation step.

5 Cross-validation

Cross-validation is a technique used to evaluate the performance of a model by training it on different subsets of the data and evaluating it on the remaining subsets. To perform 5-fold cross-validation, we implemented an algorithm from scratch. First, we merged the existing training and validation data for the color images, thus forming a single training dataset. Our algorithm starts a loop that runs 5 times. In each iteration, the data is split into three subsets: the training_left, validation and training_right. The validation set is the fold that will be used for validation during this iteration. The training set is the concatenation of the training_left and training_right subsets. The train and validation are cached and prefetched for better performance. Then model is compiled with the binary cross-entropy loss function, the Adam optimizer with a learning rate from hyperparameters tuning, and the binary accuracy metric. Also for each fold we calculate a zero-one loss.

Cross-validation showed the following results in each of the folds:

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	mean	std.dev.
Binary accuracy	0.8499	0.8868	0.9292	0.9189	0.9237	0.9017	0.03
Binary cross entropy	0.3235	0.2644	0.1788	0.1987	0.1980	0.2327	0.06
Zero-one loss	0.1501	0.1132	0.0708	0.0810	0.0763	0.0983	0.03

Despite the somewhat significant scatter in the values of the binary cross entropy, we can note that, in general, the behavior of the model in each of the fold is fairly consistent. This indicates that the chosen model performs really well on the validation set, and its performance is not just a 'lucky split'. Of course, a 10-fold cross-validation or even a nested 10-fold cross-validation will be more preferable to use. However, this would increase the computation time considerably.

6 Final model performance

At the end of the project, we train the model with optimized hyperparameters using a test set. This is done to assess the model's performance on unseen data and to ensure that the model generalizes well to new data. For these purposes, we kept the test set separate from the training and validation sets, and it was not used for hyperparameter tuning.

The performance of the final model is shown in Figure 7. Hyperparameters tuning showed that after the 18th epoch, the model shows no improvement in validation accuracy, so we train the final model with 18 epochs.

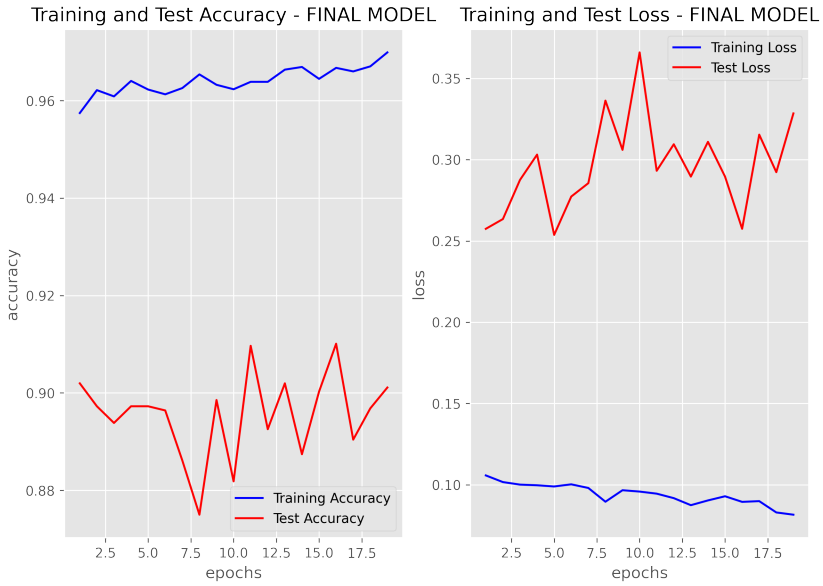

Figure 7: Training and test binary accuracy and loss – Final model

Evaluation of the final model performance:

	Binary accuracy	Binary cross entropy	Zero-one loss
Training	0.9440	0.1466	0.0560
Validation	0.9179	0.2166	0.0821
Test	0.9003	0.3254	0.0997

From the results of the model, the binary accuracy on the test set is 0.9003 and binary cross

entropy is 0.3254 which means the model is making confident predictions and not overfitting to the training data. Overall, the model's performance on the test set confirms that the optimized hyperparameters and chosen architecture are suitable for the task of classifying cats and dogs images though is always important to remember that test scores may not always reflect the true performance of the model on unseen data. It is possible that the test set is not representative of the entire population. Therefore, an important step, which we have not considered in this paper, is to test our models on other sets of real data.

7 Conclusion

The results of the experiments showed that the selected neural networks quite well cope with the task of classifying pictures of cats and dogs. However, it should be noted that the neural network architectures used leave a huge manoeuvring room for improving the models. Future work may include but is not limited to performing cross-validation and hyperparameter tuning for those models that use gray-scale images as input data, exploring different architectures, such as ResNet, incorporating more data through data augmentation techniques, fine-tuning pre-trained models, and implementing techniques such as transfer learning to improve overall model performance. Experimenting with different optimization algorithms and hyperparameter tuning methods may result in improved accuracy and loss. Incorporating other types of data, such as audio or video, in addition to image data, may also result in better results when classifying cats and dogs [3].

8 References

1. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet classification with deep convolutional neural networks.* In Advances in neural information processing systems (pp. 1097-1105).

2. Simonyan, K.,& Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition.* arXiv preprint arXiv:1409.1556.

3. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). *Going deeper with convolutions.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-9).