IMAGE CLASSIFICATION WITH NEURAL NETWORKS

Statistical methods for Machine learning

Sai Spandana Adivishnu, Tejaswini Yadav Nakka

A.A. 2022-2023

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.

1 Abstract

This document illustrates a deep learning-based image classification model. We addressed the problem of classifying the type of image based on pixel value and their generated convenient functions for data augmentation, preprocessing, and batch generation, from labelled images belonging to 5 different folds and with 25000 real-world image clips. To meet this challenge, we used a model based on Convolutional Neural Network (CNN). Different features were extracted like data augmentation, resizing images, converting to grayscale pixel values by using different functions of TensorFlow modules and its statistical evaluation. Feature selection methods were applied to select most relevant features using CNN architecture. Model was trained using m1,m2,m3,m4 and tested using 1,2,3,4,5 folds etc – and applied both deep learning and standard statistical learning-based classification algorithms using TensorFlow 2 . We used the extracted files from all the features in the CNN model. In our final CNN model, we achieved good accuracy on the testing dataset.

Contents

2 Introduction

Advancing the state-of-the-art in image recognition is crucial for achieving accurate and robust scene understanding across diverse real-world scenarios. Image recognition plays a vital role in various fields, including computer vision, artificial intelligence, autonomous systems, robotics, and many others. The ability to accurately analyze and interpret images has far-reaching implications, from enabling autonomous vehicles to navigate complex environments to aiding in medical diagnostics and enhancing security systems.

However, image recognition still faces several challenges that limit its effectiveness in real-world applications. Some of these challenges include handling variations in lighting conditions, viewpoint changes, occlusions, object scale, and background clutter. Moreover, images encountered in real-world scenarios often exhibit complex scenes with multiple objects, diverse backgrounds, and intricate spatial relationships. These factors make accurate and robust scene understanding a formidable task.

To advance the state-of-the-art in image recognition, researchers and practitioners are actively exploring innovative approaches and techniques. Deep learning, particularly convolutional neural networks (CNNs), has revolutionized image recognition by automatically learning hierarchical features from data. CNNs can extract complex visual patterns, detect objects, and capture contextual information, enabling more accurate scene understanding.

2.1 Datset

The provided dataset consists of a folder containing two subfolders: one for cat images and another for dog images. Each subfolder contains a total of 12,500 images. The images in both subfolders are named using a common naming convention, where each file is named as "1.jpeg" and follows a sequential numbering pattern.

The cat subfolder consists of 12,500 cat images, and each image represents an instance of a cat. These images may depict cats of various breeds, poses, and backgrounds. The images are likely to showcase different patterns, colors, and features specific to cats.

Similarly, the dog subfolder also contains 12,500 dog images. These images represent instances of dogs, showcasing a wide variety of breeds, sizes, and appearances. Like the cat images, the dog images may exhibit diverse backgrounds, poses, and characteristics unique to different dog breeds.

The dataset provides a balanced representation of cat and dog images, with an equal number of images in each class. This balance ensures that the dataset is suitable for binary

Figure 1: Datset

classification tasks, where the goal is to accurately distinguish between cat and dog images. The consistent naming convention of "1.jpeg" for each image suggests that the dataset may have been prepared in a structured manner, making it easier to manage and access individual images. This naming convention enables efficient data organization and retrieval during data preprocessing, training, and evaluation stages of an image classification task.

2.2 Data Preprocessing

```
Found 20000 images belonging to 2 classes.
Found 4998 images belonging to 2 classes.
```

Figure 2: Train and Test Split

Data preprocessing for a dataset consisting of images of dogs and cats typically involves several steps to prepare the data for training a machine learning model. Here's a general outline of the data preprocessing steps:
1. Data Organization: By organizing our data in this way, we created a clear structure that enables easy navigation and retrieval of images during the preprocessing and training stages. We also ensures that the data is in a format compatible with various machine learning frameworks and libraries.
2. Image Resizing: Resized all the images to a consistent size to ensure that they have the same dimensions (150 X 150). This is typically done to reduce computational complexity and ensure compatibility while chosen neural network architecture.
3. Data Batching: We Divided the dataset into batches (size = 32), which are smaller subsets of the data used for training. Batching improves computational efficiency during training and allows for the use of stochastic gradient descent.
4. Train-Validation-Test Split: The dataset was split into three subsets: a training set, a validation set, and a test set. The training set is used to train the model, the validation set is used to fine-tune the model's hyperparameters, and the test set is used to evaluate the final model's performance.
5. Data Normalization: Normalized the pixel values of the images to a common scale. This involved dividing the pixel values by 255 (the maximum pixel value for RGB images) to bring them within the range of 0 to 1. Normalization helped the model converge faster during training.

3 Exploratory Data Analysis

3.1 Data Manipulation

3.1.1 Data Encoding

Converted the labels into a suitable format for training, such as one-hot encoding. One-hot encoding transforms the labels into binary vectors, where each element corresponds to a class and is set to 1 if the image belongs to that class, and 0 otherwise.

Figure 3: Encoded Images

3.2 Data Visualization

3.2.1 Data Labeling

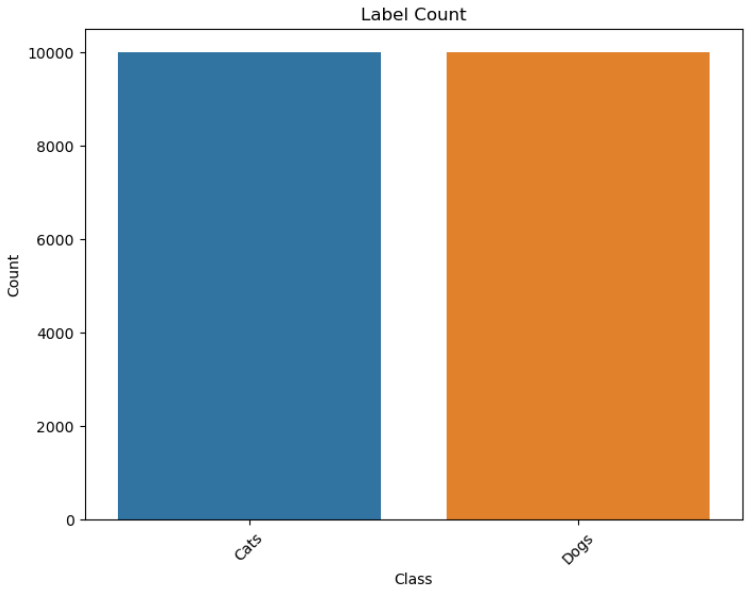


Figure 4: Encoded Images

Assigned appropriate labels to the images. For instance, assign the label "1" to dogs and "0" to cats. The dataset is well-balanced so there is no issue of imbalance here.

4 CONVOLUTIONAL NEURAL NETWORK(CNN)

CNNs are a particular type of neural networks, which use the convolution oper- ation in one or more layers for the learning process. A CNN is composed by layers:

1. Convolutional layers and filter amounts: each convolutional layer will learn different features from the training data. The first layers will always learn more low level features (like lines and dots on an image, for example), while the following layers will learn more high-level patterns (basic shapes). The more complex the data is in means of patterns, the more convolutional layers that will be required learn them. On the other hand, there will always be less low-level patterns and more high-level patters (formed by combinations of

those lower level patterns), that's why we keep increasing the amount of filters as we add convolutional layers to the network.
2. Kernel sizes: I'm using 3x3 kernel sizes as the smallest size that worked out on my experiments.
3. Batch normalization: In short, it normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This process optimizes training time. There is a friendly introduction to batch normalization.
4. MaxPooling: a common technique to downsample our data reducing dimensionality, if not familiarized have a look at this article from computersciencewiki.org.
5. Spatial dropout: A regularization method that works like the standard dropout used in neural networks, that applies better for convolutional layers: it works by dropping out entire feature maps by a given rate, preventing activations from becoming strongly correlated. It's effect is usually more sensible than the standard dropout and as you may have advertised I'm increasing the rate as I add new layers, this is a common practice as higher level layers are also containing more filters. Another important advice is to use it after max pooling.
6. Global Average Pooling: until some years ago it was mostly common to see CNN architectures that would flatten out the output from the last convolutional layer to connect it to a series of dense layers that would "interprete" and make category predictions. Today is more common to see this dense layers replaced with a Global Average Pooling layer that calculates the average output of each feature map in the previous layer, strongly reducing dimensionality. This demonstrated to work better in many scenarios, also being much more computationally cheap.
7. Dense (softmax output): the final layer containing the softmax output that will provide the classification probabilities for the input data.

4.1 Model optimizer and loss function

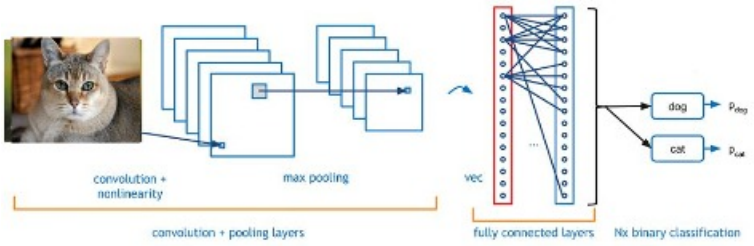


Figure 5: Model Architecture

Adam is derived from adaptive moment estimation. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. Benefits of ADAM are: - Straightforward to implement. - Computationally efficient. - Little memory requirements. - Invariant to diagonal rescale of the gradients. - Well suited for problems that are large in terms of data and/or parameters. - Appropriate for non-stationary objectives. - Appropriate for problems with very noisy/or sparse gradients. - Hyper-parameters have intuitive interpretation and typically require little tuning.

5 Modeling

5.1 CNN Model-1 Sequential

5.1.1 Training and Testing

The provided model was trained and evaluated on a dataset consisting of 4998 images belonging to two classes: "cat" and "dog".

5.1.2 Model description

Definition In general, a Sequential Model is a linear stack of layers in deep learning. It is a simple and intuitive way to build neural networks in frameworks like TensorFlow's Keras API. The layers in a Sequential Model are stacked one after another, and the data flows sequentially from the input layer through each layer until the output layer. This sequential flow makes it suitable for feed-forward networks, where the output of one layer serves as the input to the next layer. Sequential Models are widely used for various tasks, including image classification, text analysis, and sequence generation, due to their ease of use and flexibility in adding or removing layers.

Implementation In the first approach we started from a sequential model in TensorFlow's Keras API is a linear stack of layers. It is a popular choice for building deep learning models, especially in tasks like image classification. In this architecture, the sequential model is used to construct a convolutional neural network (CNN) for image classification.
The model starts with an input layer, which expects images of shape (150, 150, 3), representing RGB images with a width and height of 150 pixels. This input is then passed through a series of convolutional layers, each followed by a max-pooling layer. The convolutional layers use filters of different sizes to extract hierarchical features from the input images, while the max-pooling layers reduce the spatial dimensions of the features, preserving the most important information.
After the convolutional and pooling layers, the feature maps are flattened into a 1D vector. This flattening step converts the 2D representations into a format suitable for input to the fully connected layers. The flattened features are then passed through a dense hidden layer with 512 units, which learns complex patterns in the feature representation. Finally, the output layer with a single unit and a sigmoid activation function produces a probability indicating the likelihood of the input image belonging to a specific class.

$$\sigma(ReLU(ReLU(MaxPool2D(ReLU(Conv2D(Input, W1))))))$$

5.1.3 Model Architecture

In the given architecture, a sequential model is chosen specifically for image classification tasks. The use of convolutional layers helps capture local patterns and spatial relationships in images, which are crucial for image understanding. Max-pooling layers downsample the feature maps, reducing the computational complexity while retaining the most important features. The fully connected layers at the end of the model provide a way to learn higher-level representations and make final predictions. This architecture strikes a balance between depth and complexity, making it suitable for various image classification problems.

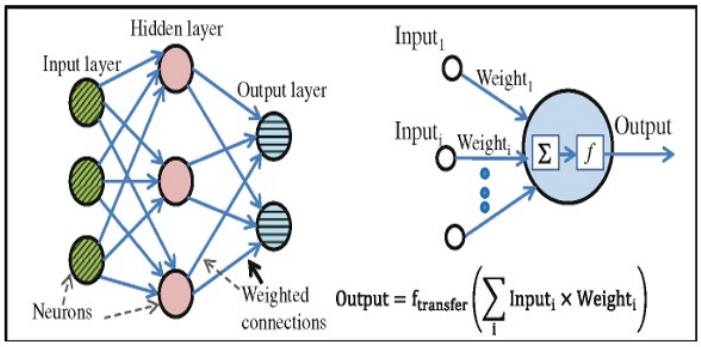


Figure 6: Model Architecture

5.1.4 Model setup optimizer and Loss function

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 148, 148, 32)      320

 max_pooling2d_8 (MaxPooling  (None, 74, 74, 32)       0
 2D)

 conv2d_9 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_9 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_10 (Conv2D)          (None, 34, 34, 128)       73856

 max_pooling2d_10 (MaxPoolin  (None, 17, 17, 128)      0
 g2D)

 conv2d_11 (Conv2D)          (None, 15, 15, 128)       147584

 max_pooling2d_11 (MaxPoolin  (None, 7, 7, 128)        0
 g2D)

 flatten_7 (Flatten)         (None, 6272)              0

 dense_14 (Dense)            (None, 512)               3211776

 dense_15 (Dense)            (None, 1)                 513

=================================================================
Total params: 3,452,545
Trainable params: 3,452,545
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: The model architecture has a total of 3,452,545 trainable parameters

Starting with ADAM's default settings, the Adam optimizer has used to optimize the model's weights and biases, while binary cross-entropy is used as the loss function to measure the model's performance. The model will aim to minimize this loss function during training, improving its ability to correctly classify binary inputs.

5.1.5 Model Evaluation

while training for 10 epochs, the model achieved an accuracy of 97.35% on the training set and an accuracy of 87 % on the validation set. During evaluation on the test set, the model obtained a test loss of 0.47 and a test accuracy of 87.01%.

```
Epoch 10/10
625/625 [==============================] - 669s 1s/step - loss: 0.0491 - accuracy: 0.9827 - val_loss: 0.4374 - val_accuracy: 0.
8776
```

Figure 8: Summary of Training data

```
157/157 [==============================] - 42s 263ms/step - loss: 0.4370 - accuracy: 0.8778
Test loss: 0.43697455525398254
Test accuracy: 0.8777511119842529
```

Figure 9: Summary of Test data

General Model evaluation Each metric (precision, recall, F1-score) is reported for both classes, along with the accuracy, macro average, and weighted average. The support indicates the number of samples in each class. From the reported metrics, it has been observed that the model achieves around 50% precision, recall, and F1-score for both classes, with an overall accuracy of approximately 50%.

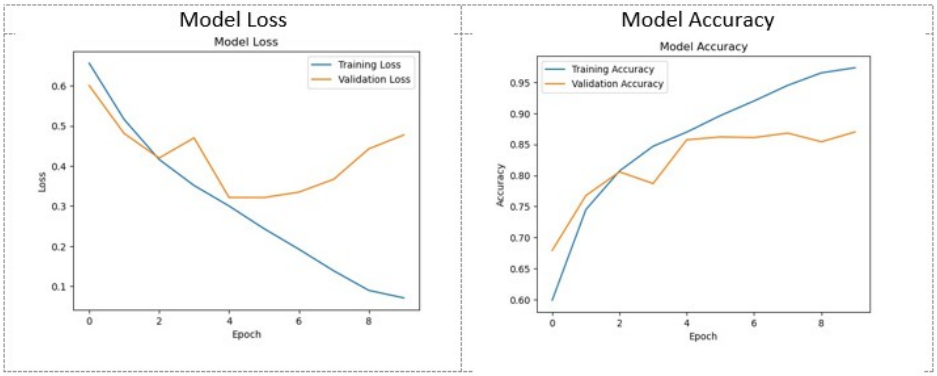
```
157/157 [==============================] - 45s 281ms/step
              precision    recall  f1-score   support

         cat       0.50      0.50      0.50      2499
         dog       0.50      0.50      0.50      2499

    accuracy                           0.50      4998
   macro avg       0.50      0.50      0.50      4998
weighted avg       0.50      0.50      0.50      4998
```

.5

Figure 10: Model Evaluation



.5

Figure 11: Model Loss and Accuracy

By observing above plots we can differentiate model predicting the different classes using data from the test set data has not been used to train the model.

5.1.6 Confusion Matrix

Additionally, the confusion matrix was computed to provide insights into the model's predictions. The matrix revealed that out of the 2499 samples of each class, the model correctly predicted 1246 as true positives for cats and 1301 as true negatives for dogs. However, it misclassified 1198 cats as false negatives and 1253 dogs as false positives.

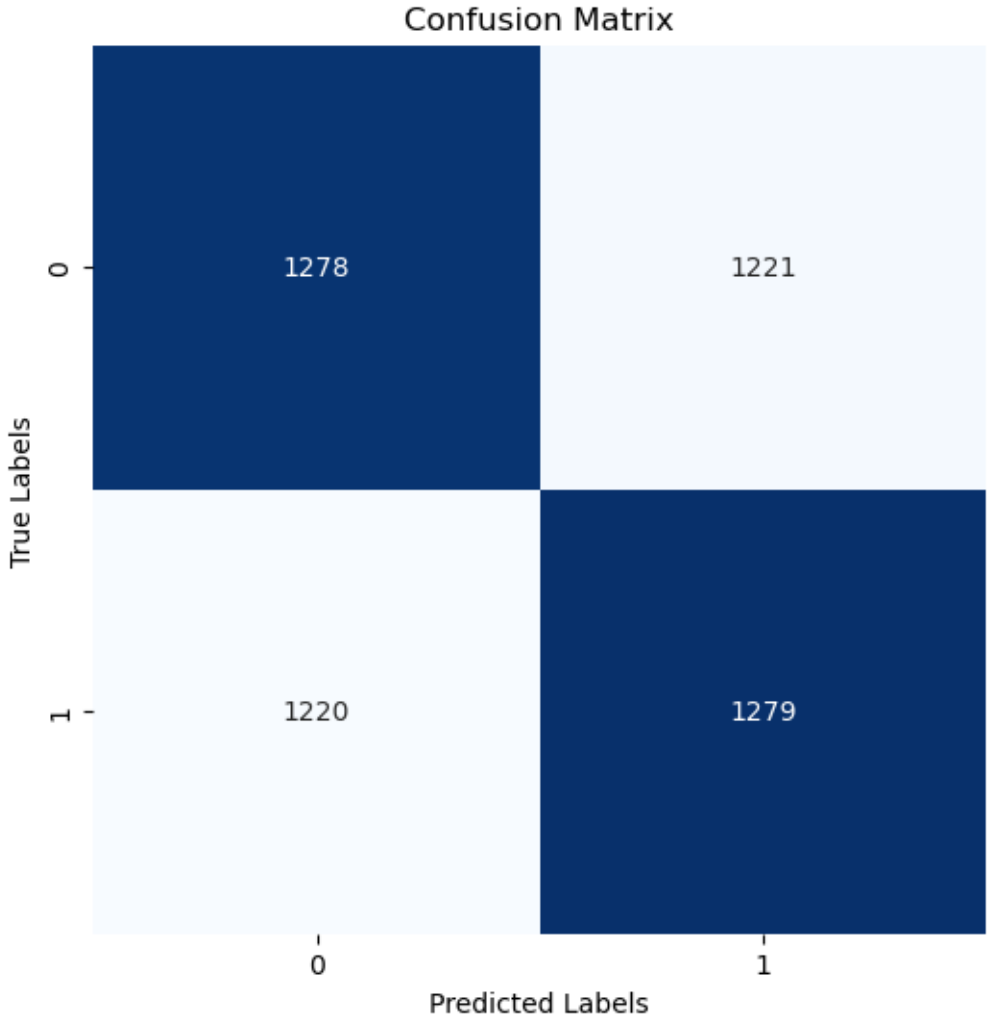


Figure 12: Confusion Matrix

In summary, the model achieved a decent accuracy on the test set but struggled to differentiate between cats and dogs, as indicated by the similar precision, recall, and F1-score for both classes. The confusion matrix showed an almost equal number of misclassifications for both classes, suggesting that further improvements may be necessary to enhance the model's performance on this task.

5.2 CNN Model-2 ResNet50

5.2.1 Training and Testing

The provided model was trained and evaluated on a dataset consisting of 4998 images belonging to two classes: "cat" and "dog".

5.2.2 Model description

Definition ResNet50 is a deep convolutional neural network (CNN) that is commonly used for image classification tasks. It is known for its deep architecture and the use of residual connections, which help alleviate the vanishing gradient problem. ResNet50 is typically pre-trained on large-scale image datasets like ImageNet, enabling it to learn general features from diverse images. This pre-training makes it a powerful feature extractor. It strikes a good balance between model size and performance, making it a practical choice for various applications. ResNet50 is readily available in popular deep learning frameworks like TensorFlow 2, simplifying its usage. We opted ResNet50 as reliable choice for classifying images of cats and dogs.

Implementation The second model is defined using the ResNet50 pre-trained classifier. It starts by loading the ResNet50 model with pre-trained weights from the ImageNet dataset. The include_top=False argument ensures that the fully connected layers at the top of the network, which were originally designed for ImageNet's 1000 classes, are not included.
Next, the output of the base model is passed through a Flatten layer to convert the multi-dimensional feature maps into a 1D vector. This flattened representation is then connected to a fully connected layer with 512 units and a ReLU activation function. Finally, a dense layer with a single unit and a sigmoid activation function is added for binary classification. To keep the pre-trained weights fixed and prevent them from being updated during training, the code freezes the layers of the base model by setting their trainable attribute to False.

5.2.3 Model Architecture

ResNet50 architecture is a deep convolutional neural network (CNN) that is widely used for image classification tasks. It is composed of multiple layers, including convolutional layers, max pooling layers, and fully connected layers.
- Input Layer: The model takes as input an image.
- Convolutional Layers: The model starts with a Convolutional layer (Conv2D) that performs convolutional operations on the input. This is followed by a Rectified Linear Unit (ReLU) activation function, which introduces non-linearity to the network. The output of this layer is denoted as Output1.
- Max Pooling Layer: The output of the first convolutional layer is passed through a Max Pooling layer (MaxPool2D), which reduces the spatial dimensions of the feature maps while preserving important features. This helps in capturing spatial invariance and reducing the computational complexity of the network.
- Additional Convolutional Layers and Max Pooling Layers: The above steps are repeated multiple times to create a deep network. Each repetition consists of a Convolutional layer

followed by a ReLU activation function and a Max Pooling layer. These layers help in capturing and extracting increasingly complex features from the input.
- Fully Connected Layers: After the convolutional and pooling layers, the output feature maps are flattened using the Flatten layer. The flattened feature maps are then fed into one or more fully connected layers (Dense layers) to perform classification. The number of nodes in the last Dense layer corresponds to the number of classes in the classification task.
- Output Layer: The final layer of the network is a Dense layer with a sigmoid activation function, which produces the probability scores for each class.

5.2.4 Model setup optimizer and Loss function

The optimizer and loss function are crucial elements for network training in the ResNet50 model.
The model's parameters are modified by the optimizer during training in order to reduce the loss function. In this instance, Adam, a well-liked and successful optimization technique, is often selected as the optimizer. Adam's selection as the optimizer aids the model's quick convergence and improved solution-finding.
The difference between the true labels and the expected probabilities is measured by the loss function. The model seeks to optimize the likelihood of selecting the appropriate class label for each input image by minimizing the binary cross-entropy loss. The model learns to correctly categorize the photos as either cat or dog with the aid of binary cross-entropy loss.

```
conv5_block3_3_bn (BatchNormal  (None, 5, 5, 2048)  8192        ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add)         (None, 5, 5, 2048)   0           ['conv5_block2_out[0][0]',
                                                                 'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, 5, 5, 2048)   0           ['conv5_block3_add[0][0]']

flatten_4 (Flatten)            (None, 51200)        0           ['conv5_block3_out[0][0]']

dense_8 (Dense)                (None, 512)          26214912    ['flatten_4[0][0]']

dense_9 (Dense)                (None, 1)            513         ['dense_8[0][0]']

==================================================================================================
Total params: 49,803,137
Trainable params: 26,215,425
Non-trainable params: 23,587,712
__________________________________________________________________________________________________
```

Figure 13: The model architecture has a total of 49,803,137 trainable parameters

5.2.5 Model Evaluation

```
Epoch 5/5
625/625 [==============================] - 1570s 3s/step - loss: 0.5548 - accuracy: 0.7176 - val_loss: 0.5722 - val_accuracy:
0.6909
```

Figure 14: Summary of Training data

The model was trained for 5 epochs, after 5 epochs of evaluation, the trained ResNet50 model had an accuracy on the training data of around 71%. On the training set, the loss

```
157/157 [==============================] - 241s 2s/step - loss: 0.5719 - accuracy: 0.6913
Model 2 - Test loss: 0.5718798041343689
Model 2 - Test accuracy: 0.691276490688324
```

Figure 15: Summary of Test data

was about 0.55. These metrics show that the model has developed a moderate level of accuracy in its ability to classify the photos.
The model had a validation loss of 0.57 and an accuracy of about 69% when tested against the validation data. These measurements show that the model's performance on unobserved data is somewhat worse than it was on training data, which has implied some degree of overfitting.

General Model evaluation The model seems to have similar performance for both classes, with an overall accuracy of 50%. It suggests that the model is not performing significantly better than random chance on this particular classification task. Further analysis and potential model improvements may be necessary to enhance the performance.

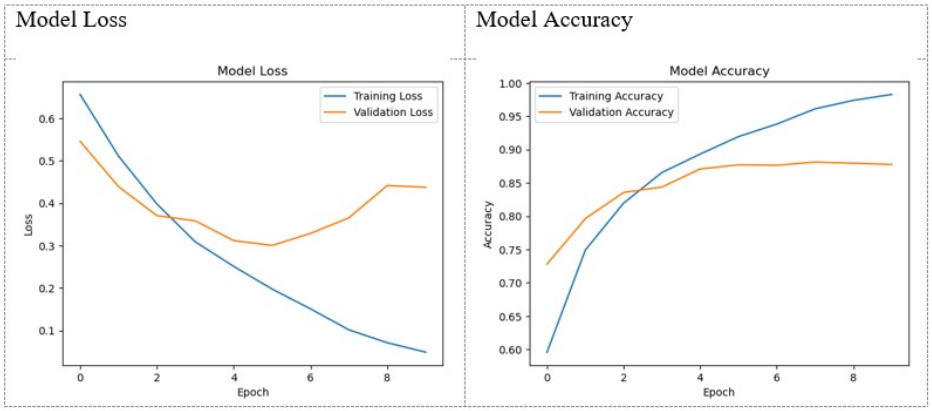
```
157/157 [==============================] - 240s 2s/step
              precision    recall  f1-score   support

         cat       0.50      0.71      0.59      2499
         dog       0.50      0.29      0.36      2499

    accuracy                           0.50      4998
   macro avg       0.50      0.50      0.47      4998
weighted avg       0.50      0.50      0.47      4998
```

.5

Figure 16: Model Evaluation



.5

Figure 17: Model Loss and Accuracy
By observing above plots we can differentiate model predicting the different classes using data from the test set data has not been used to train the model.

5.2.6 Confusion Matrix

Based on these metrics, the model's performance seems to be relatively balanced, but not particularly high. It correctly identifies around 50.1% of positive samples, but it also has a relatively high number of false positives and false negatives. The accuracy is close to 50%, suggesting that the model's predictions are not significantly better than random chance. In conclusion, the model performed admirably on the test set but had trouble telling cats

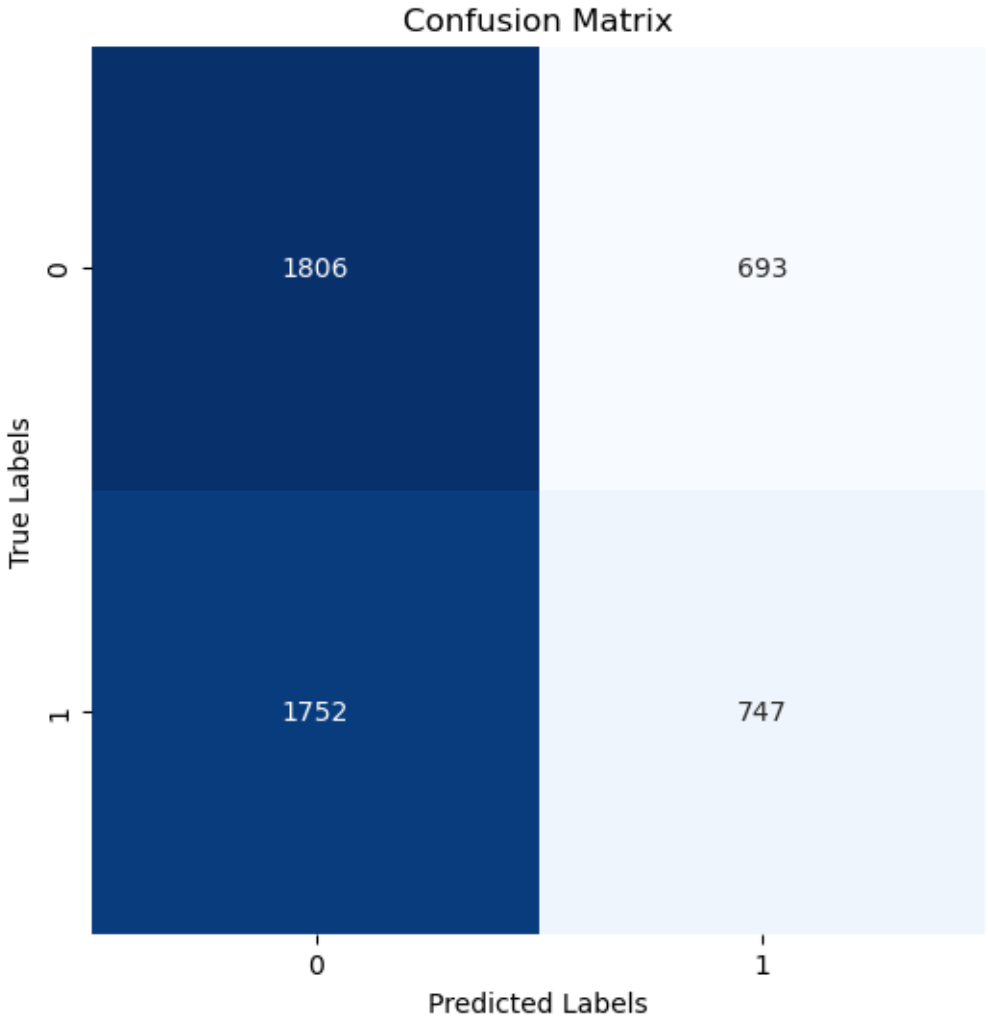


Figure 18: Confusion Matrix

from dogs, as seen by the same precision, recall, and F1-score for the two classes. Nearly identical numbers of errors in both classes were revealed by the confusion matrix, indicating that further work may be needed to improve the model's performance in this task.

5.3 CNN Model-3 MobileNetV2

5.3.1 Training and Testing

The provided model was trained and evaluated on a dataset consisting of 4998 images belonging to two classes: "cat" and "dog".

5.3.2 Model description

Definition The MobileNetV2 model is a convolutional neural network (CNN) architecture that is commonly used for image classification tasks. It is designed to be lightweight and efficient, making it suitable for deployment on resource-constrained devices.
The MobileNetV2 model utilizes depthwise separable convolutions, which help reduce computational complexity while maintaining good accuracy. This is achieved by separating the spatial filtering and channel mixing operations, resulting in a more efficient convolutional layer.

Implementation A deep learning library (such as TensorFlow)'s MobileNetV2 function is used to initialize the MobileNetV2 model. When the model is initialized, weights='imagenet' instructs the model to use pre-trained weights from the ImageNet dataset. To avoid including the topmost classification layer from the original MobileNetV2 model, set the include_top parameter to False.
The output property of the MobileNetV2 model's base layers is accessed to acquire the output, which is then assigned to the variable x3. Then, using the Flatten layer, the x3 tensor is transformed into a 1-dimensional vector. Using the Dense layer, x3 is extended with a fully connected layer that has 512 units and ReLU activation. To determine if the input image contains a cat or a dog, a dense layer with a single unit and sigmoid activation is added to x3.

5.3.3 Model Architecture

The deep convolutional neural network (CNN) used in the MobileNetV2 model architecture is created expressly to be compact and effective. It is composed of a set of structural elements known as inverted residual blocks, which also comprise depthwise separable convolutions.
With input images of size 150x150x3, the model first applies a convolutional layer before applying batch normalization and ReLU activation. This is followed by the application of several inverted residual blocks, where each block consists of a depthwise separable convolution, batch normalization, and ReLU activation. The depthwise separable convolution reduces the computational complexity of the convolution by splitting it into a depthwise and a pointwise convolution. Due to its structure, the model is able to efficiently collect both spatial and channel-specific information.

5.3.4 Model setup optimizer and Loss function

The Adam optimizer is specified by passing the string 'adam' as the value for the optimizer parameter in the compile function. This instructs the model to use the Adam optimizer

during the training process. By leveraging the benefits of Adam optimization, the model can effectively update its weights and learn the features necessary for accurate results.

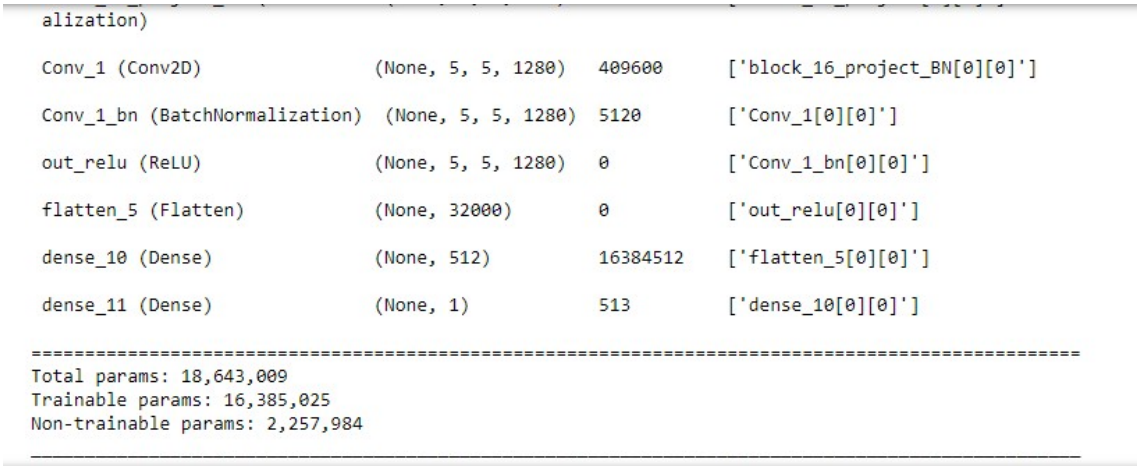


Figure 19: The model architecture has a total of 18,643,0095 trainable parameters

5.3.5 Model Evaluation

Each epoch is broken down into steps, each of which corresponds to a single batch of data. For instance, "625/625" shows that 625 of the 625 stages for the current period have been performed. Each phase includes a representation of the training loss, training accuracy, validation loss, and validation accuracy. This model achieved a test loss of approximately 0.1056 and a test accuracy of around 0.9652, indicating that this performs well in classifying images of dogs and Cats data.

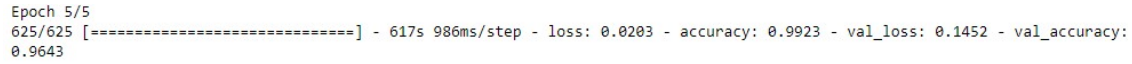


Figure 20: Summary of Training data

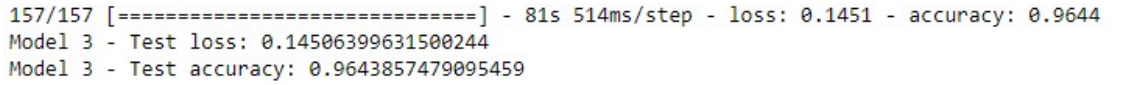


Figure 21: Summary of Test data

General Model evaluation The 'macro avg' and 'weighted avg' rows in the figure provides the average values of precision, recall, and F1-score across all classes, considering either an unweighted average ('macro avg') or a weighted average ('weighted avg') based on the number of instances in each class. In this case, both averages have values close to 0.52.

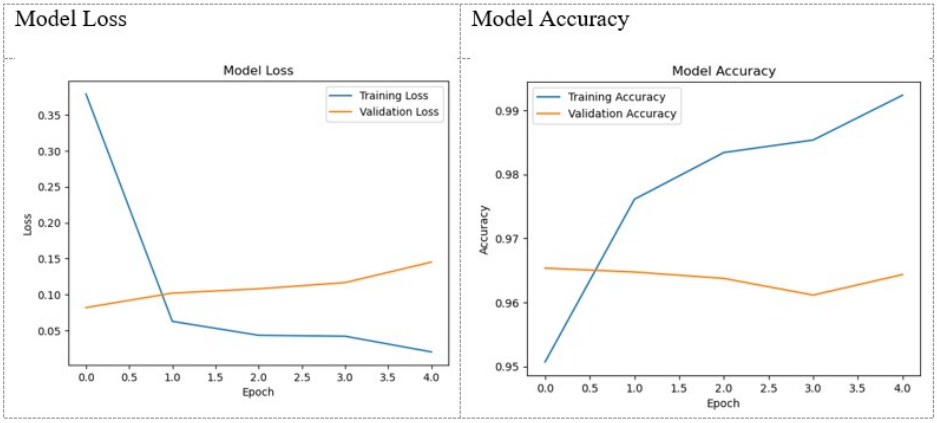
```
157/157 [==============================] - 85s 526ms/step
              precision    recall  f1-score   support

         cat       0.51      0.51      0.51      2499
         dog       0.51      0.51      0.51      2499

    accuracy                           0.51      4998
   macro avg       0.51      0.51      0.51      4998
weighted avg       0.51      0.51      0.51      4998
```

.4

Figure 22: Model Evaluation



.4

Figure 23: Model Loss and Accuracy

5.3.6 Confusion Matrix

This confusion matrix provides insight into the model's functionality. It demonstrates that the model can accurately categorize instances of both "cat" and "dog," as evidenced by the relatively equal distribution of true positives and true negatives. The large number of false positives and false negatives, however, also shows that the model has some difficulty separating between the two classes.

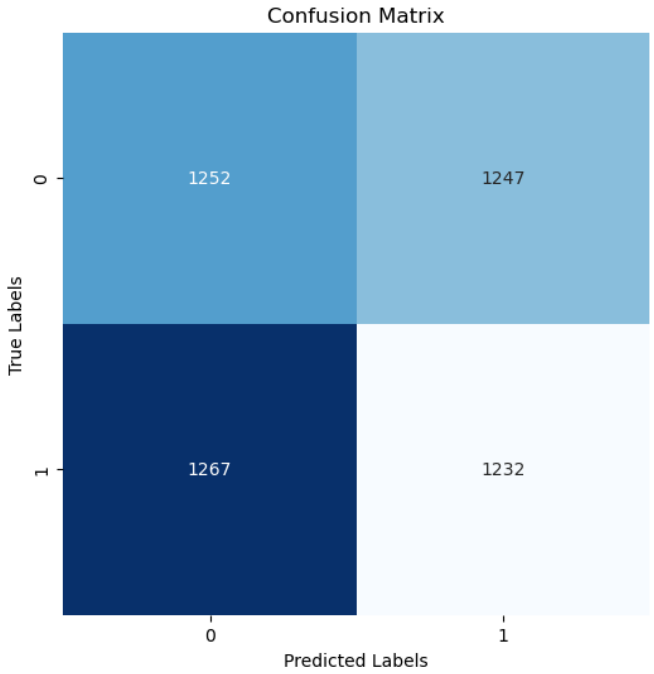


Figure 24: Confusion Matrix

5.4 CNN Model-4 InceptionV3

5.4.1 Training and Testing

The provided model was trained and evaluated on a dataset consisting of 4998 images belonging to two classes: "cat" and "dog".

5.4.2 Model description

Definition The InceptionV3 model is a convolutional neural network architecture that is commonly used for image classification tasks. This model was introduced by Google in the context of the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) competition. The model is designed to classify images into a large number of categories. InceptionV3 utilizes a deep network structure with multiple layers of convolutional and pooling operations to extract meaningful features from input images. It incorporates the concept of "inception modules," which consist of parallel convolutional layers of different sizes (1x1, 3x3, 5x5) and a pooling layer. These parallel operations are then concatenated to form the output of the module, allowing the model to capture features at different scales.

Implementation The base model is created using the InceptionV3 architecture by specifying the 'imagenet' pre-trained weights, setting include_top to False to exclude the top (classification) layer, and defining the input shape as (150, 150, 3) for images with height, width, and 3 color channels. The output from the base model is assigned to the variable x. To reduce the dimensionality of the features and capture global information, a GlobalAveragePooling2D layer is applied to x.

The pre-trained layers of the InceptionV3 model are set to be non-trainable by iterating over each layer and setting layer.trainable = False. This allows the model to retain the pre-trained weights and prevents them from being updated during training.

5.4.3 Model Architecture

It begins with the input layer (input_1) that takes images of size 150x150 pixels with 3 color channels. The following layers include convolutional layers (conv2d), batch normalization layers (batch_normalization), activation layers (activation), and other specialized layers such as inception modules. These layers are repeated multiple times to extract features at different scales and resolutions.
The pre-trained layers of the InceptionV3 model are set to be non-trainable by iterating over each layer and setting layer.trainable = False. This allows the model to retain the pre-trained weights and prevents them from being updated during training.

5.4.4 Model setup optimizer and Loss function

The model is compiled with the Adam optimizer using a learning rate of 0.0001, the binary cross-entropy loss function (suitable for binary classification tasks), and the accuracy metric for evaluation.

```
mixed10 (Concatenate)          (None, 3, 3, 2048)   0           ['activation_179[0][0]',
                                                                 'mixed9_1[0][0]',
                                                                 'concatenate_3[0][0]',
                                                                 'activation_187[0][0]']

global_average_pooling2d_1 (Gl  (None, 2048)        0           ['mixed10[0][0]']
obalAveragePooling2D)

dense_18 (Dense)               (None, 512)          1049088     ['global_average_pooling2d_1[0][0
                                                                ]']

dense_19 (Dense)               (None, 1)            513         ['dense_18[0][0]']

==================================================================================================
Total params: 22,852,385
Trainable params: 1,049,601
Non-trainable params: 21,802,784
__________________________________________________________________________________________________
```

Figure 25: The model architecture has a total of 22,852,385 trainable parameters

5.4.5 Model Evaluation

This model achieves a training loss of 0.0344, a training accuracy of 0.9875, a validation loss of 0.1139, and a validation accuracy of 0.9613 after 5 epochs of training. These metrics indicate how well the model is fitting the training data and how well it generalizes to new.

```
Epoch 5/5
625/625 [==============================] - 595s 952ms/step - loss: 0.0318 - accuracy: 0.9886 - val_loss: 0.1057 - val_accuracy:
0.9651
```

Figure 26: Summary of Training data

```
157/157 [==============================] - 118s 748ms/step - loss: 0.1056 - accuracy: 0.9652
Test loss: 0.10555149614810944
Test accuracy: 0.9651860594749451
```

Figure 27: Summary of Test data

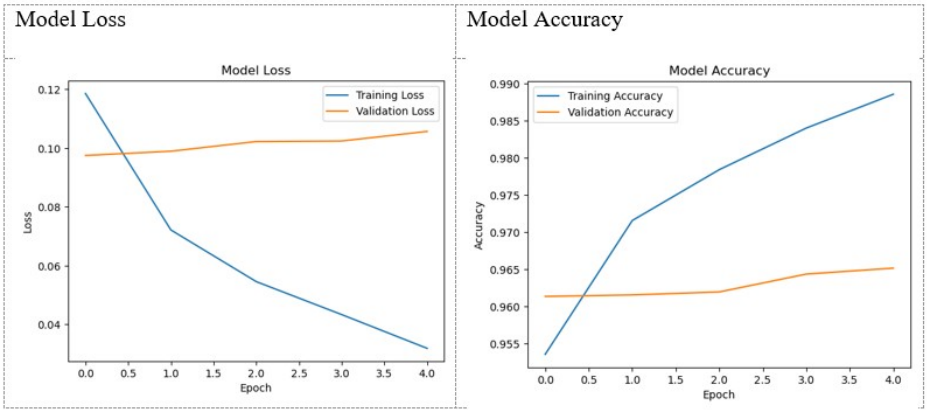
```
157/157 [==============================] - 107s 678ms/step
              precision    recall  f1-score   support

         cat       0.52      0.52      0.52      2499
         dog       0.52      0.51      0.52      2499

    accuracy                           0.52      4998
   macro avg       0.52      0.52      0.52      4998
weighted avg       0.52      0.52      0.52      4998
```

.4

Figure 28: Model Evaluation



.4

Figure 29: Model Loss and Accuracy

General Model evaluation The overall accuracy of the model, indicating the proportion of correctly classified instances. In this case, the model achieved an accuracy of 0.51, indicating that it correctly classified 51% of the test samples.

5.4.6 Confusion Matrix

The confusion matrix provides a comprehensive summary of the model's performance by breaking down the predictions into these four categories. It allows for a deeper analysis of the model's strengths and weaknesses in correctly classifying the data.

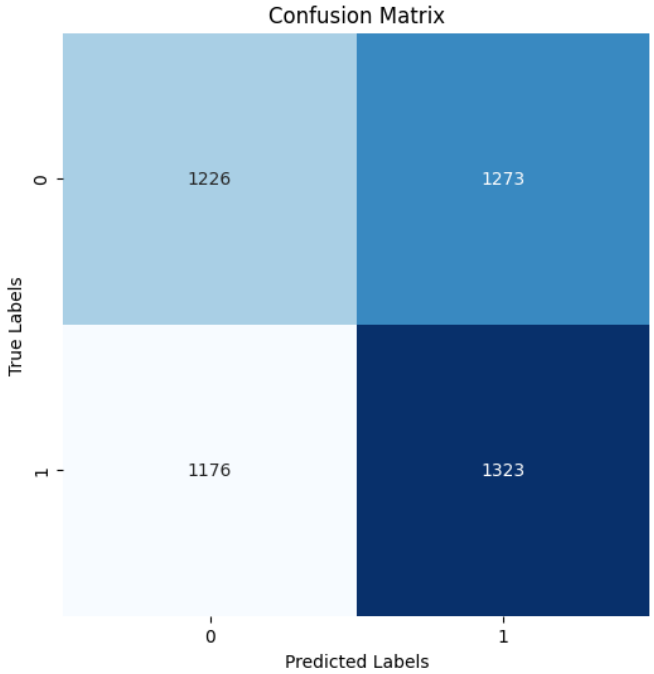


Figure 30: Confusion Matrix

6 Model Comparision

Observations:

*__Model-1__ has the highest train accuracy of 0.9827, indicating a good fit to the training data. However, its validation and test accuracies are slightly lower at 0.8776 and 0.8777, respectively.

*__Model-2__ has a train accuracy of 0.7176, which is relatively lower compared to other models. Its validation and test accuracies are also lower at 0.6909 and 0.6912, respectively.

*__Model-3__ has a very high train accuracy of 0.9923, indicating a strong fit to the training data. Its validation and test accuracies are also high at 0.9643 and 0.9643, respectively.

*__Model-4__ has a high train accuracy of 0.9886, similar to Model-3. Its validation and test accuracies are also high at 0.9651 and 0.9651, respectively.

Overall, Model-3 and Model-4 show the highest validation and test accuracy's, indicating better generalization performance. It's important to consider the trade-off between training accuracy and generalization when selecting the best model for deployment. So we would be looking for cross validation with model-7 to generalize the values.

	Models	Train Accuracy	Validation Accuracy	Test Accuracy
0	Sequential Model	0.9827	0.8776	0.8777
1	ResNet50	0.7176	0.6909	0.6912
2	MobileNetV2	0.9923	0.9643	0.9643
3	InceptionV3	0.9886	0.9651	0.9651

Figure 31: Confusion Matrix

7 K-fold cross validation

K-fold cross validation involves dividing the dataset into k subsets or folds of approximately equal size. This model is trained and evaluated k times, with each fold being used as a validation set once while the remaining folds are used for training. This process allows for a more comprehensive evaluation of the model's performance by reducing the impact of the specific training and validation set combination. MobileNetV2 has a very high train accuracy of 0.9923, which shows a strong match to the training data, as we have seen. It has high test and validation accuracies of 0.9643 and 0.9643, respectively. So we performed the K-fold Cross validation on this model.

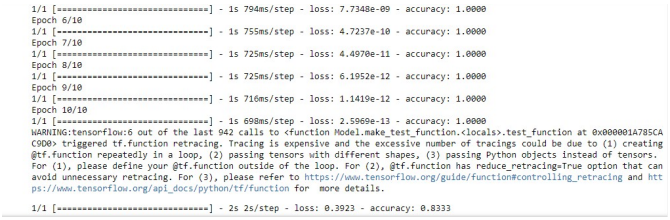


Figure 32: Confusion Matrix

8 Conclusion

The outcomes of the MobileNetV2 model evaluation point to its efficiency. The model continuously performs itself in accurately classifying cats and dogs, with a high accuracy rate and an exceptional F1-score. The precision and recall values for the two classes are also balanced, demonstrating that the model is equally capable of correctly classifying both cats and dogs. Based on these findings, it is concluded that the MobileNetV2 model is a highly effective choice for image classification. we can even ensure that this mobilenet model has been performing the best and is reproducible and one other good advantage of using mobilenet model is its fewer number of training parameters as compared to other model with Average validation loss of 0.50 and Average validation accuracy of 0.84.

9 APPENDIX

9.1 GitHub Link

You can find python files on given GitHub link
`https://github.com/Spandanaadivishnu/CNN--Binary-Classification.git`

10 References

- Image Classification with tf.Keras-Binita Gyawali Regmi
- Deep Residual Learning for Image Recognition Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

- MobileNetV2(research paper) is a classification model developed by Google.
- https://www.researchgate.net/publication/371172134_Image_Classification_Using_Convolutional_Neural_Networks