الاء محمد حامد عوض 20201378365

Report

1 – Importing Libraries

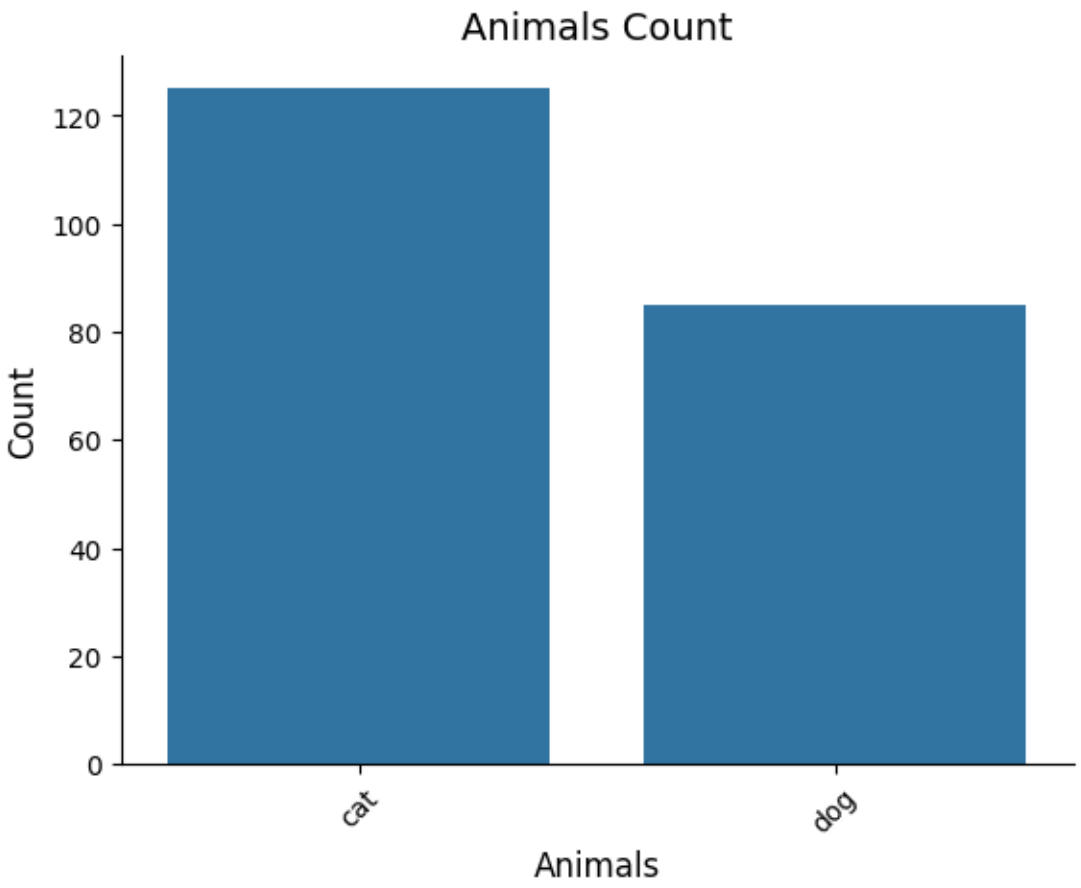
The first step of the project was to install and import the necessary libraries needed such as Librosa for loading and dealing with audio data, matplotlib for visualization, numpy for dealing with arrays and sklearn for normalizing and splitting the data.

2- Importing the Dataset

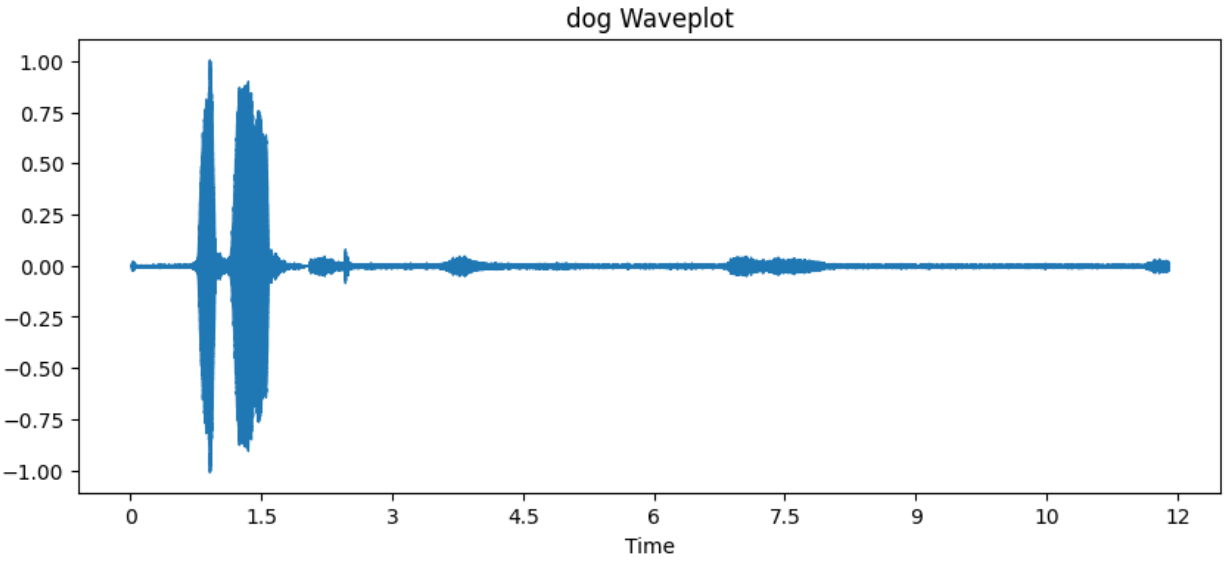
The dataset was downloaded from Kaggle which was already split into training and testing folders. Then, I looped through the training part to create lists with the names of the animals (cat or dog) and the audio paths. Later, these lists were turned into a Pandas data frame with two columns, one for animals and the other for paths and 210 rows since we have 210 audios in the training dataset each one corresponding its classification.

3 – Data Visualization

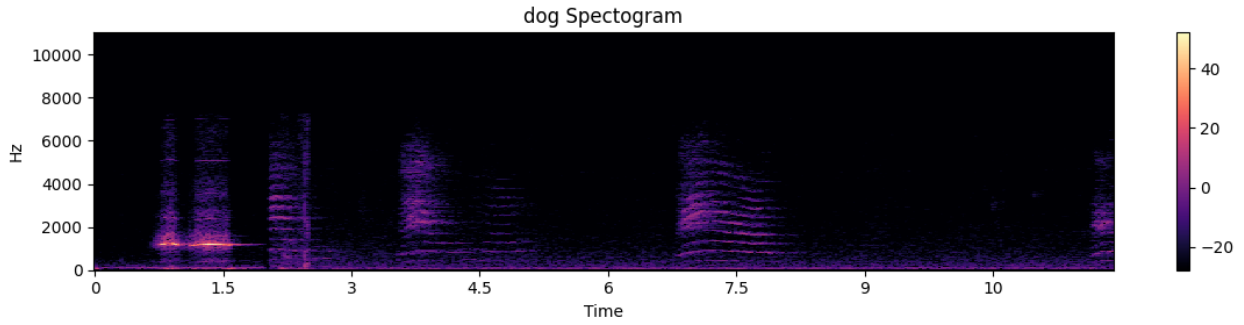
This is an important step to understand the dataset clearly. A bar chart was created to show the classes in the data and the number of instances in each class. From the figure below, we can notice that the cat class has more data than the dog class.



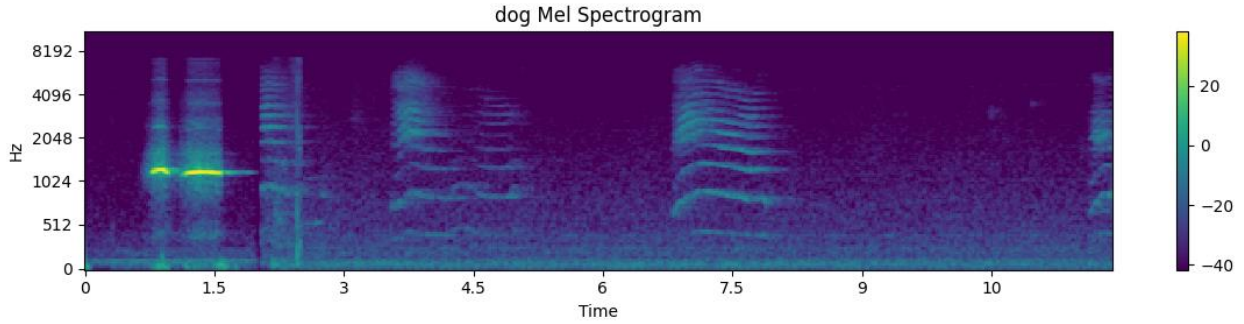
I have also generated the waveplot which captures the varying amplitudes of the signal samples over time for one of the audios to get a comprehensive understanding of the data.



Using short-time Fourier transform, which separates the signal into time windows, and applies the FFT on each time window I generated the Spectrogram which is the visualization of our signal's frequencies as it varies over time.



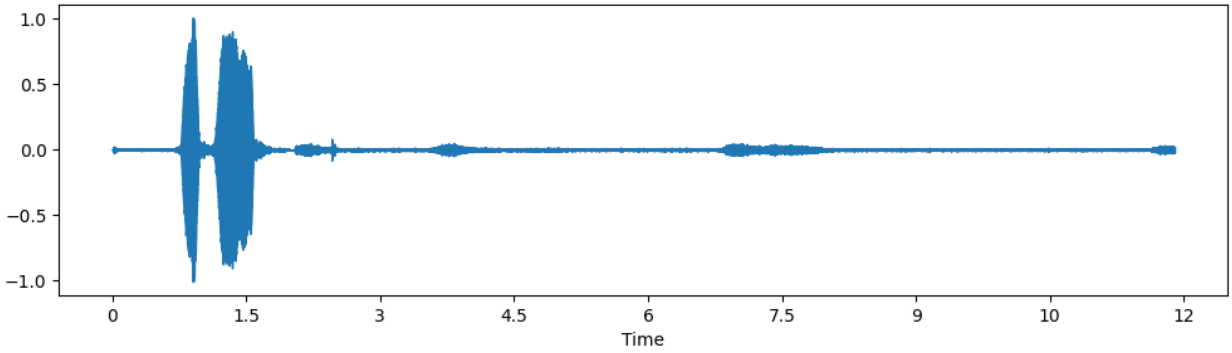
Converting the spectrogram frequencies into mel scale, the Mel Spectrogram was generated.



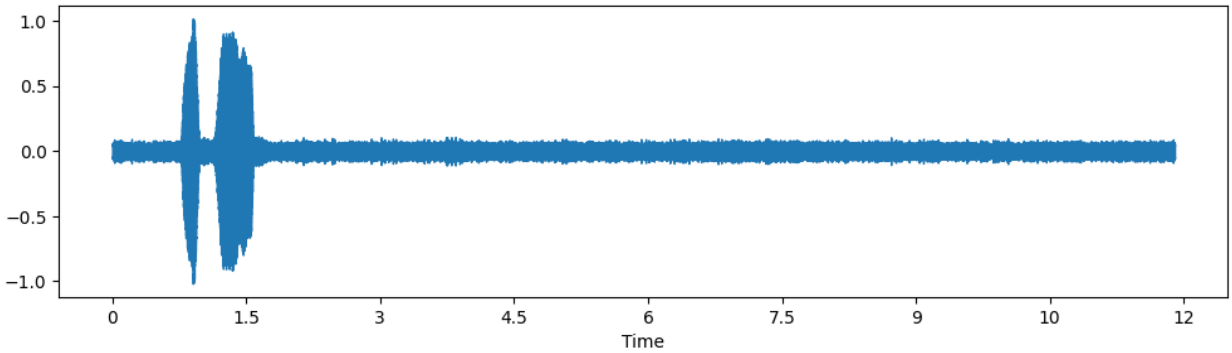
4- Data Augmentation

Since the animals' dataset is relatively small, I had decided to apply data augmentation techniques to increase its size and make it more complex for the model. First, only noise and time stretching were applied; however, after working on the model, it was notable that more data augmentation was needed so time shifting and pitch were added.

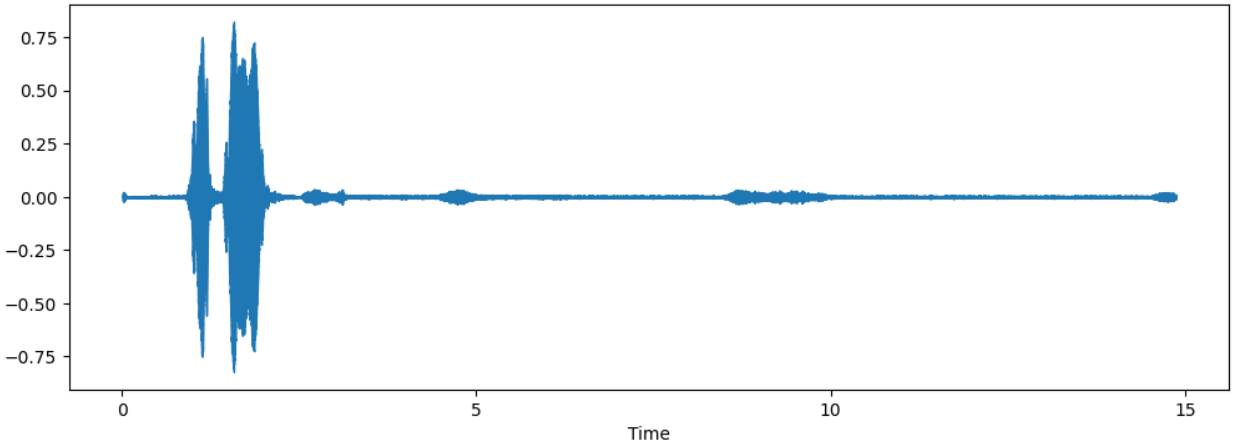
Audio without augmentation:



Audio after noise added:



Audio after time stretching:



5- Feature Extraction

- **extract_features** : This function was created to extract features from the data for the deep learning model where two features were extracted

MFCC (Mel-frequency cepstral coefficients) and Mel spectrogram. MFCC was extracted using librosa.feature.mfcc() then appended to the result array followed by the Mel spectrogram extraction using librosa.feature.melspectrogram() which was also appended to the result array.

- **get_features**: This function loads the audio files and calls the data agumentation functions as well as the extract_features function and returns a stacked array of features for the original and augmented versions of the audio.

- **Feature Extraction and Processing Loop**: This loop iterates over each audio file and its associated animal label in the data_df DataFrame, extracts features using get_features(), and stores them in lists X and Y. For each audio file, get_features(path) is called to extract features, which are then processed and added to the X list. The corresponding animal label is added to the Y list for each set of features.

6- Data Preparation

The Y labels are transformed into 0 and 1 through one-hot encoding.

7- Splitting Data Into Training and Validation

Using Sklearn the data is splitted into x_train, x_val, y_train and y_val and the validation percentage is 0.2 of the training.

For the preprocessing of the data to be used with CNNs, Standard scaler from sklearn was used to apply normalization on x_train and x_val. Then a new axis was added to the data to get the shape (samples, features, 1). The last step is done to make the data compatible with models that expect a 3D input.

8 – CNN model

Keras was imported to help with using the CNN layers' components like conv1D and max pooling.

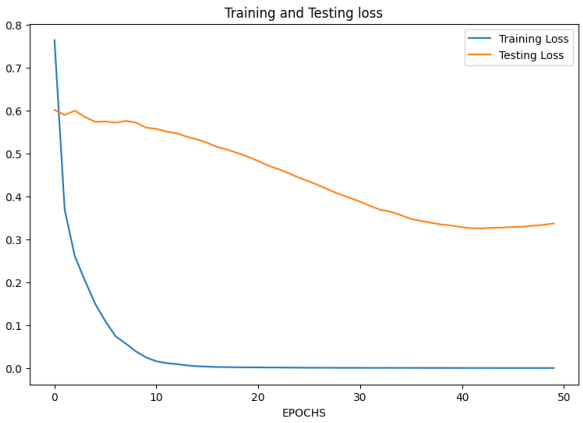
First a sequential model was created with the layers as following:

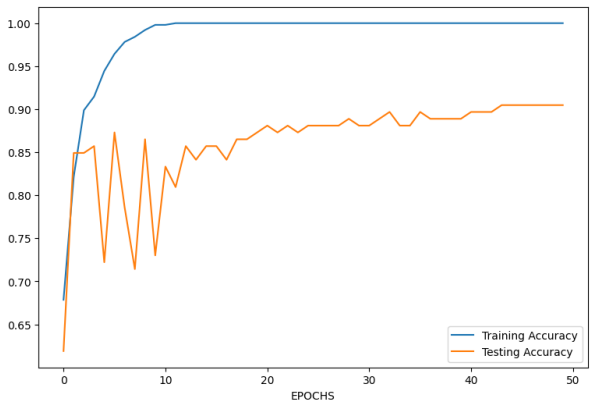
```
model.add(Conv1D(128 , kernel_size = 5 , strides = 1 , padding = 'same' , activation ='relu',
         input_shape=(x_train.shape[1],1)
         ))
model.add(BatchNormalization())
model.add(Dense(128 , activation = 'relu'))
model.add(MaxPooling1D(pool_size = 5 , strides = 2 , padding = 'same'))

model.add(Conv1D(256 , kernel_size = 5 , strides = 1 , padding = 'same' , activation ='relu',
         input_shape=(x_train.shape[1],1)
         ))
model.add(BatchNormalization())
model.add(Dense(64 , activation = 'relu'))
model.add(MaxPooling1D(pool_size = 5 , strides = 2 , padding = 'same'))

model.add(Flatten())

model.add(Dense(32 , activation = 'relu'))
model.add(Dense(2 , activation = 'softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' ,metrics = ['accuracy'])
```

Conv1D, Batch normalization, dense layer and max pooling were used and repeated with different channels numbers where the last dense layer has 2 neurons to output dog or cat (0 or 1), Adam optimizer was used.

Trial 1 with less data augmentation:

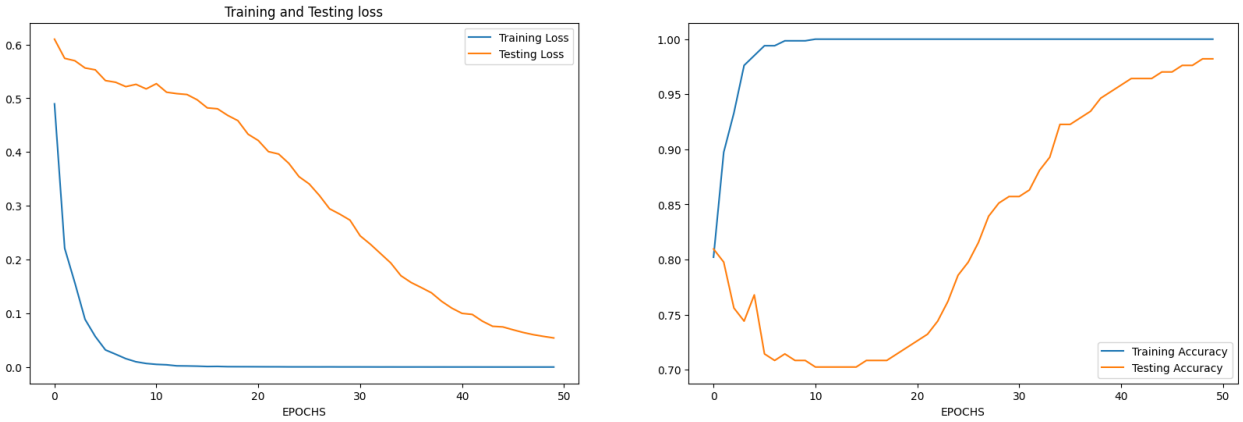
```
Epoch 50/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 210ms/step - accuracy: 1.0000 - loss: 2.4999e-04 - val_accuracy: 0.9048 - val_loss: 0.3371
```





Trial 2 with more data augmentation:

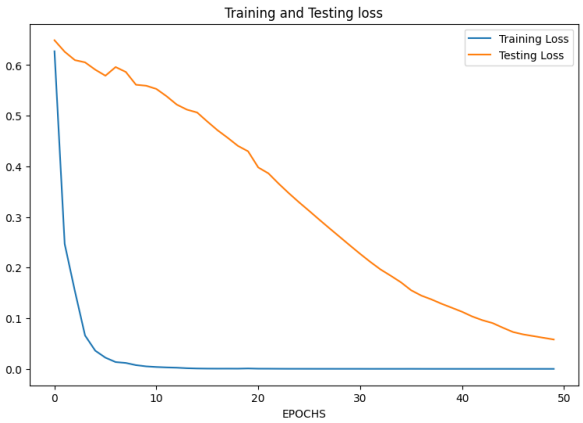
```
Epoch 50/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 202ms/step - accuracy: 1.0000 - loss: 8.7514e-05 - val_accuracy: 0.9821 - val_loss: 0.0541
```

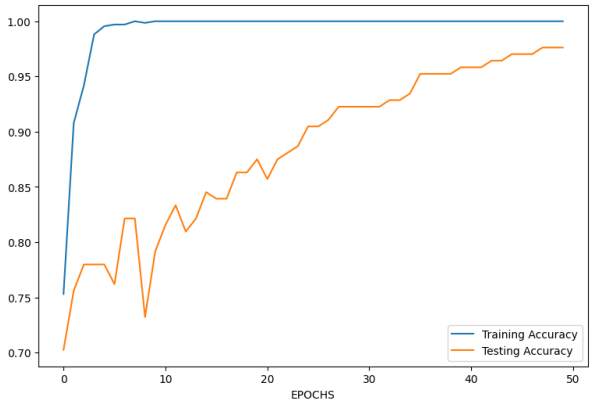


Trial 3 with more data and an additional block of layers:

```
model.add(Conv1D(128 , kernel_size = 5 , strides = 1 , padding = 'same' , activation ='relu',
         input_shape=(x_train.shape[1],1)
         ))
model.add(BatchNormalization())
model.add(Dense(128 , activation = 'relu'))
model.add(MaxPooling1D(pool_size = 5 , strides = 2 , padding = 'same'))

model.add(Conv1D(256 , kernel_size = 5 , strides = 1 , padding = 'same' , activation ='relu',
         input_shape=(x_train.shape[1],1)
         ))
model.add(BatchNormalization())
model.add(Dense(64 , activation = 'relu'))
model.add(MaxPooling1D(pool_size = 5 , strides = 2 , padding = 'same'))

model.add(Conv1D(512 , kernel_size = 5 , strides = 1 , padding = 'same' , activation ='relu',
         input_shape=(x_train.shape[1],1)
         ))
model.add(BatchNormalization())
model.add(Dense(32 , activation = 'relu'))
model.add(MaxPooling1D(pool_size = 5 , strides = 2 , padding = 'same'))


model.add(Flatten())

model.add(Dense(32 , activation = 'relu'))
model.add(Dense(2 , activation = 'softmax'))
```

```
Epoch 50/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 288ms/step - accuracy: 1.0000 - loss: 5.4696e-05 - val_accuracy: 0.9762 - val_loss: 0.0581
```





9 – Testing dataset

The testing dataset was imported and preprocess following all the same steps applied to the training and evaluation data

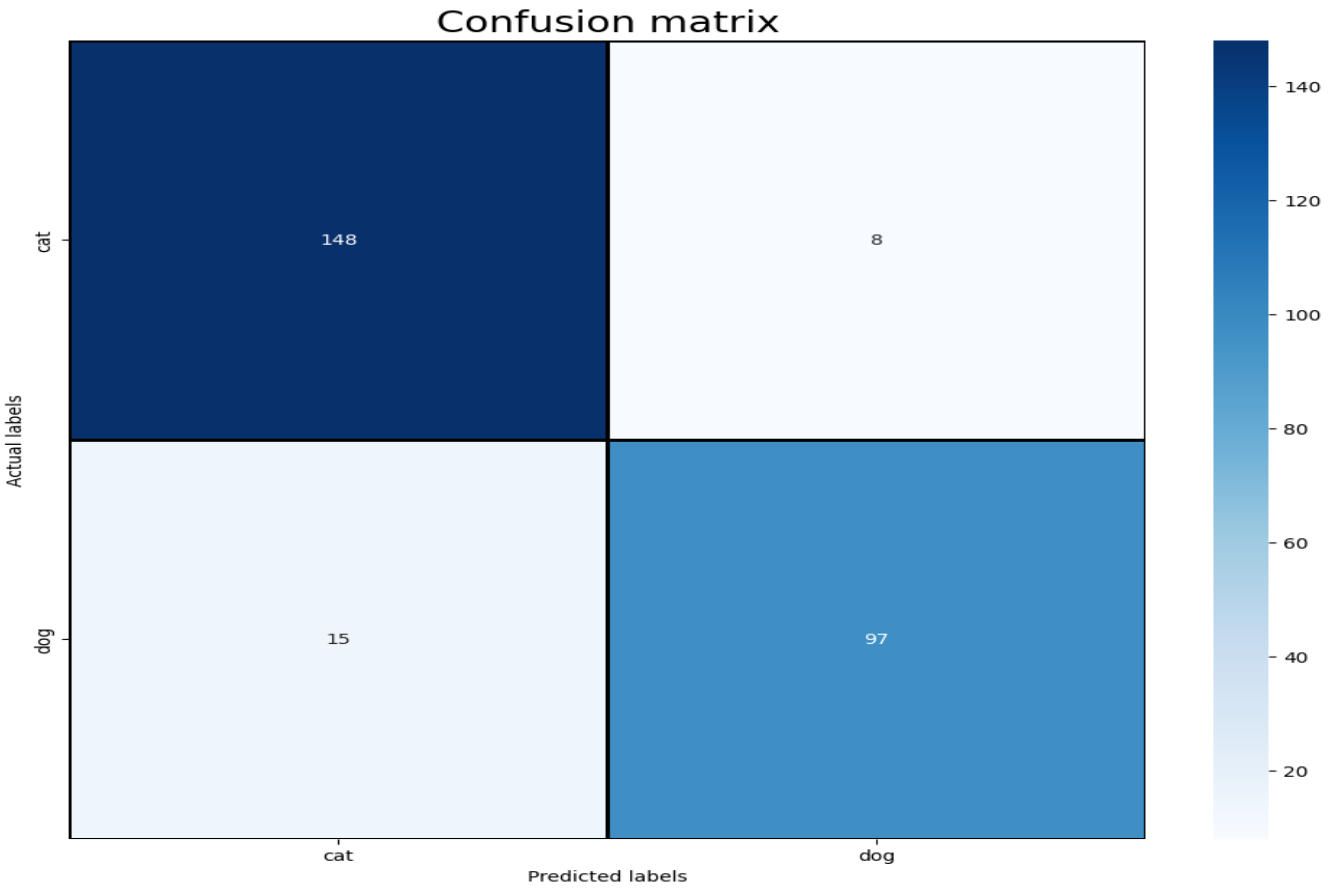
10- Evaluation

After testing the model on the data, it achieved an accuracy of 91.4%

```
[ ]  print("Accuracy of the model on the test data : " , model.evaluate(x_test,Y)[1]*100,'%')
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.8921 - loss: 0.4620
Accuracy of the model on the test data :  91.41790866851807 %
```

Confusion Matrix:


Confusion matrix
148
8
15
97
cat
dog
Actual labels
cat
dog
Predicted labels
140
120
100
80
60
40
20