CLASSIFY THE IMAGES OF CATS AND DOGS USING CNN

BATCH: 2020

SEC: A

COURSE: MACHINE LEARNING (LAB)

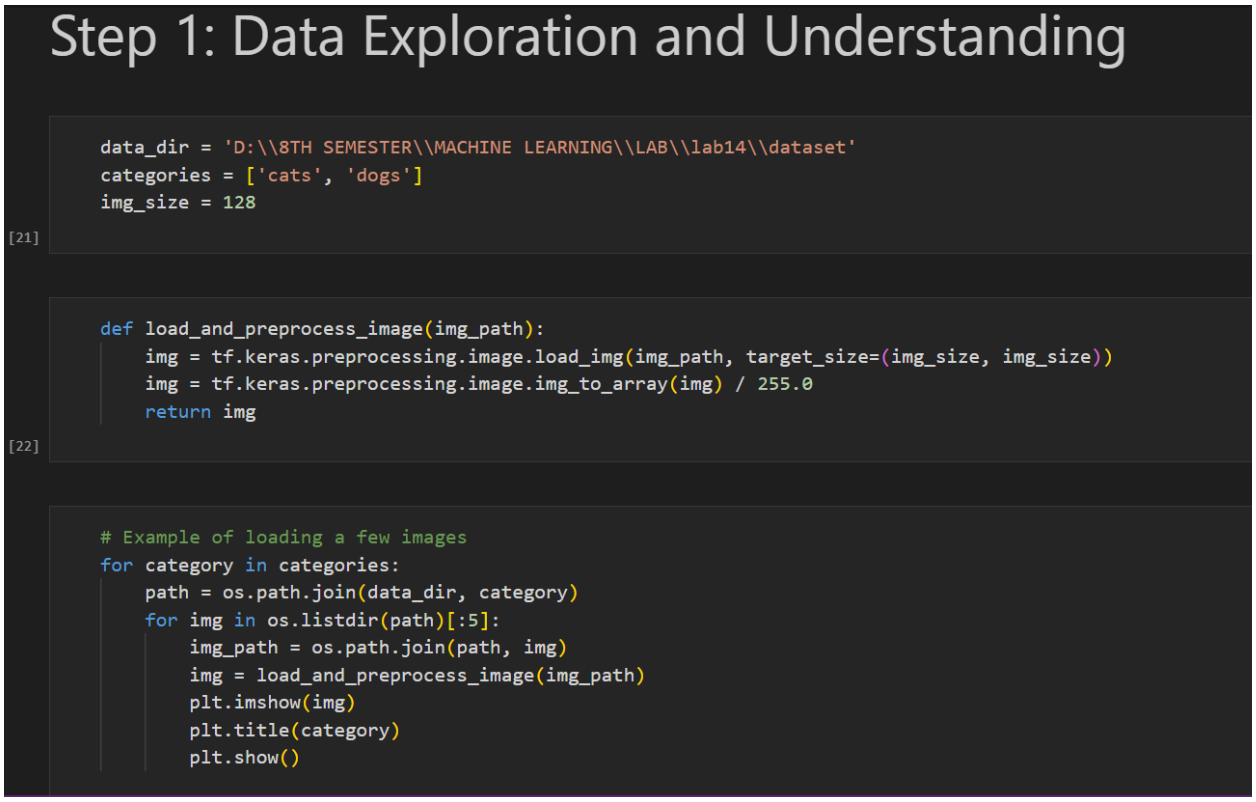
1. Introduction

This project is to implement different concepts of machine learning to distinguish between cats and dogs images. The tasks include data exploration, data preprocessing, feature selection, model selection and training, model evaluation, and result visualization. The dataset is a collection of images of cats and dogs with labels.

2. Data Exploration and Understanding

Exploring the Dataset

We begin by exploring the dataset to understand its structure and content. The dataset contains two categories: cats and dogs. Each category has a set of images.



Step 1: Data Exploration and Understanding

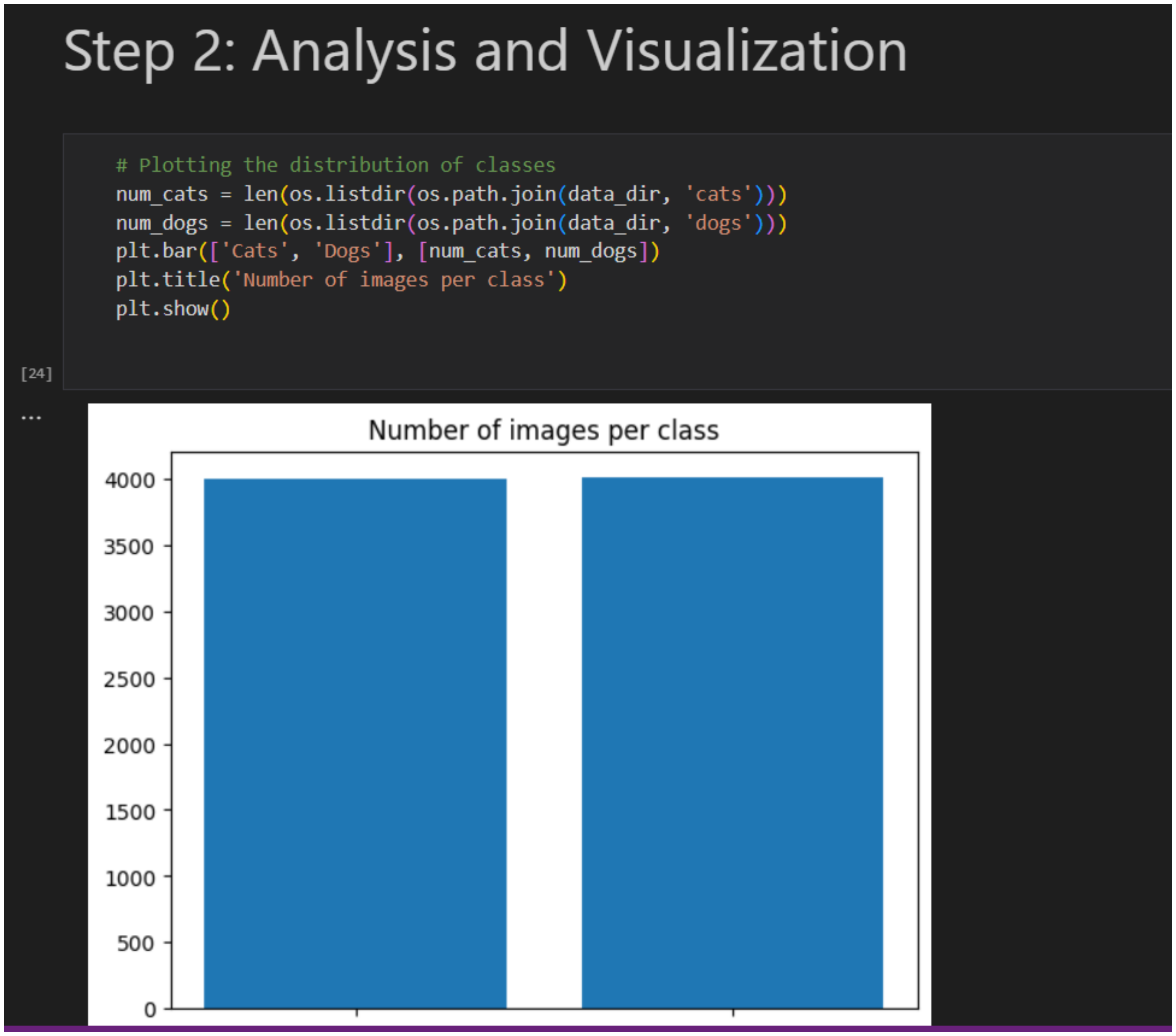
```
data_dir = 'D:\\8TH SEMESTER\\MACHINE LEARNING\\LAB\\lab14\\dataset'
categories = ['cats', 'dogs']
img_size = 128
```
[21]

```
def load_and_preprocess_image(img_path):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=(img_size, img_size))
    img = tf.keras.preprocessing.image.img_to_array(img) / 255.0
    return img
```
[22]

```
# Example of loading a few images
for category in categories:
    path = os.path.join(data_dir, category)
    for img in os.listdir(path)[:5]:
        img_path = os.path.join(path, img)
        img = load_and_preprocess_image(img_path)
        plt.imshow(img)
        plt.title(category)
        plt.show()
```

Class Distribution

To understand the distribution of classes, we plotted the number of images in each category.

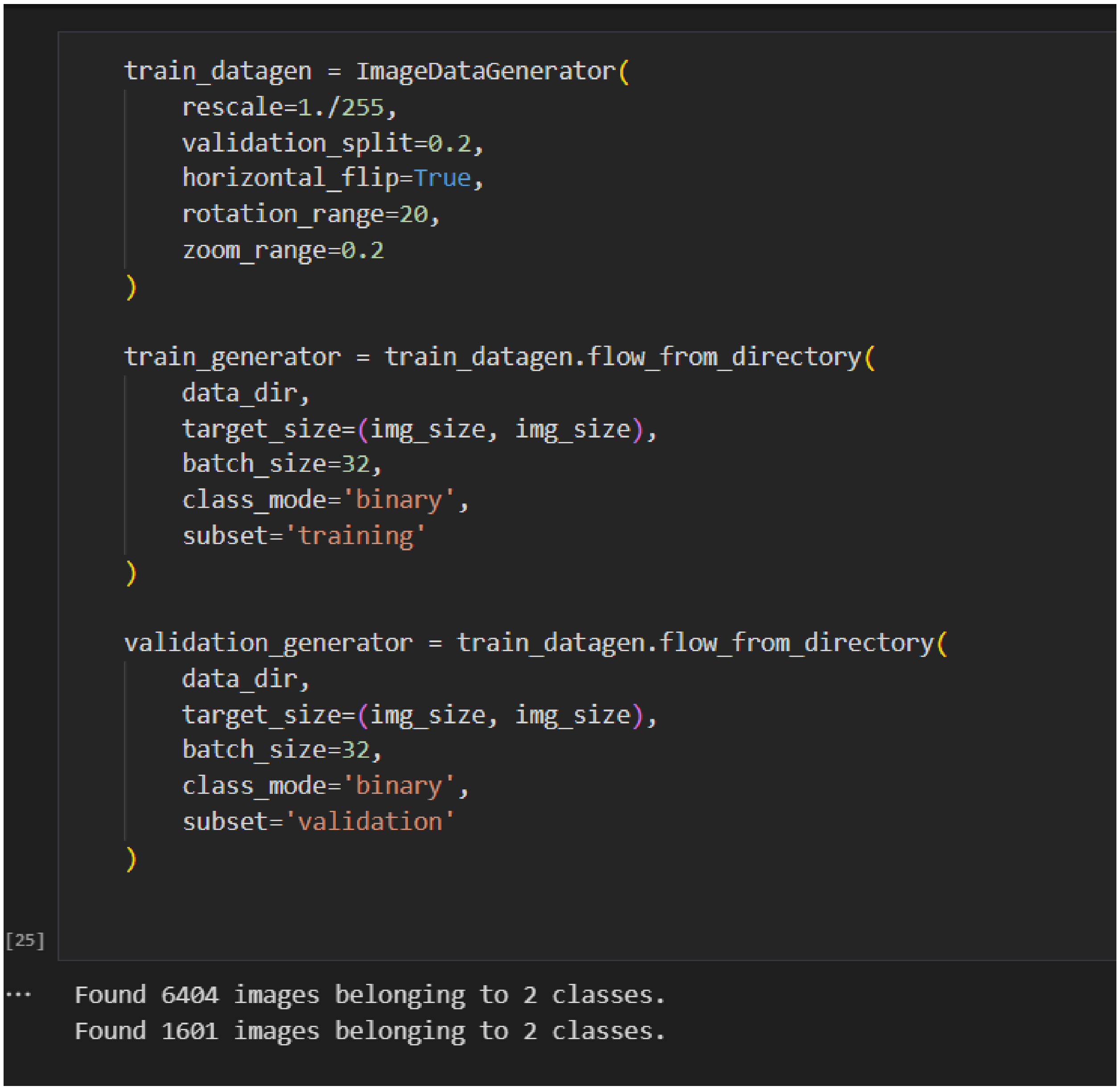


Step 2: Analysis and Visualization

```
# Plotting the distribution of classes
num_cats = len(os.listdir(os.path.join(data_dir, 'cats')))
num_dogs = len(os.listdir(os.path.join(data_dir, 'dogs')))
plt.bar(['Cats', 'Dogs'], [num_cats, num_dogs])
plt.title('Number of images per class')
plt.show()
```

3. Data Preprocessing and Cleaning

To preprocess the data, we use the ImageDataGenerator class. This involves flipping, resizing, rotating, and zooming to increase the size of the dataset and enhance the model's performance.



```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    horizontal_flip=True,
    rotation_range=20,
    zoom_range=0.2
)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_size, img_size),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_size, img_size),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

[25]

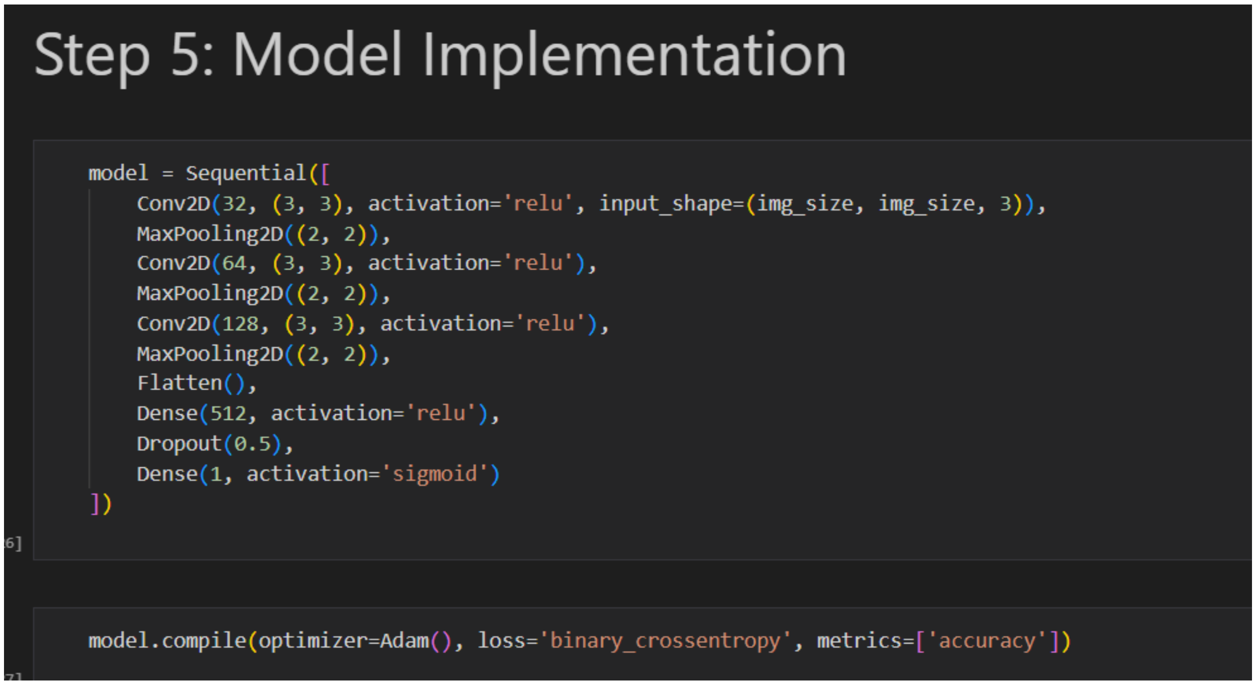
```
Found 6404 images belonging to 2 classes.
Found 1601 images belonging to 2 classes.
```

4. Feature Selection and Engineering

In image classification tasks with CNNs, feature selection is not a separate step since the convolutional layers learn to extract features from images. This helps in reducing the amount of work that is required to be done manually in feature engineering.

5. Model Implementation

We use Keras to implement a Convolutional Neural Network (CNN). The model architecture consists of convolutional layers, max-pooling layers, a flattening layer, dense layers, and a dropout layer.



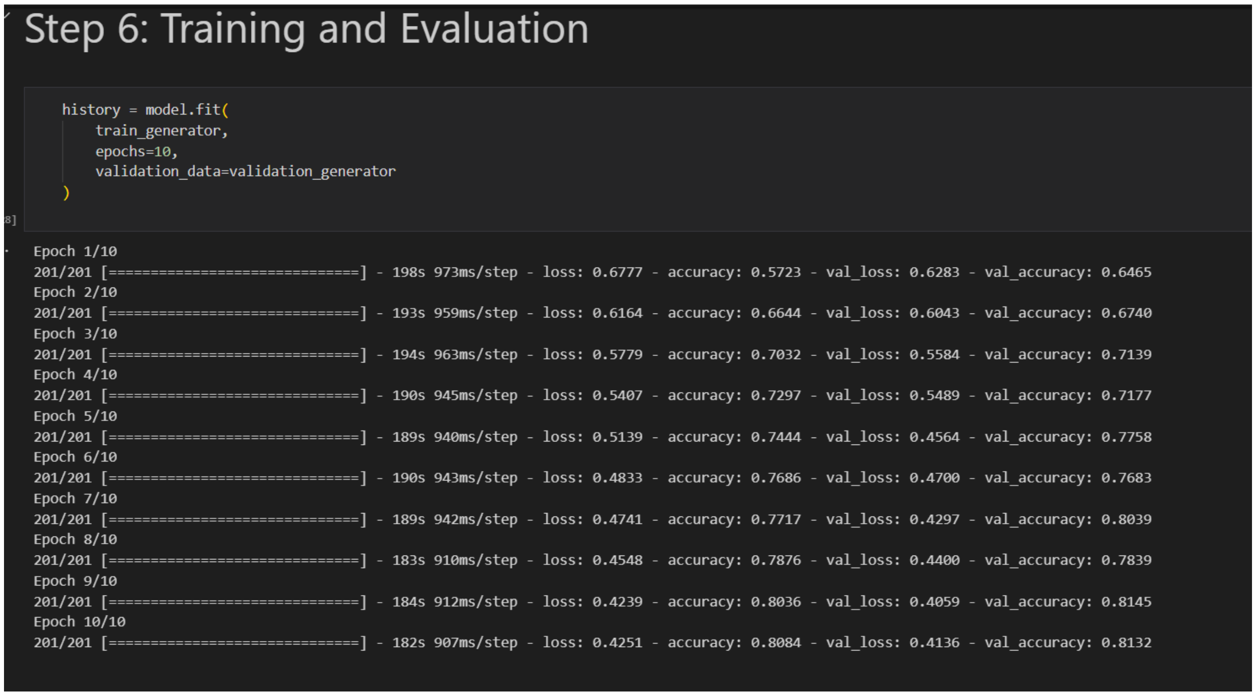
Step 5: Model Implementation

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
```

6. Training and Evaluation

The model is trained for 10 epochs using the training and validation data generators. To assess the performance of the model, we use accuracy and loss as the evaluation metrics.



Step 6: Training and Evaluation

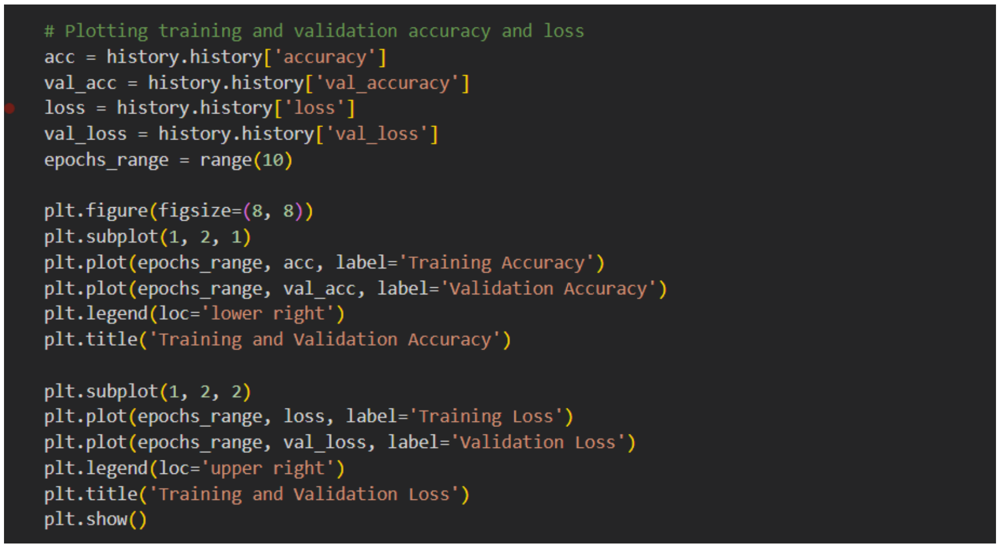
```
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

```
Epoch 1/10
201/201 [==============================] - 198s 973ms/step - loss: 0.6777 - accuracy: 0.5723 - val_loss: 0.6283 - val_accuracy: 0.6465
Epoch 2/10
201/201 [==============================] - 193s 959ms/step - loss: 0.6164 - accuracy: 0.6644 - val_loss: 0.6043 - val_accuracy: 0.6740
Epoch 3/10
201/201 [==============================] - 194s 963ms/step - loss: 0.5779 - accuracy: 0.7032 - val_loss: 0.5584 - val_accuracy: 0.7139
Epoch 4/10
201/201 [==============================] - 190s 945ms/step - loss: 0.5407 - accuracy: 0.7297 - val_loss: 0.5489 - val_accuracy: 0.7177
Epoch 5/10
201/201 [==============================] - 189s 940ms/step - loss: 0.5139 - accuracy: 0.7444 - val_loss: 0.4564 - val_accuracy: 0.7758
Epoch 6/10
201/201 [==============================] - 190s 943ms/step - loss: 0.4833 - accuracy: 0.7686 - val_loss: 0.4700 - val_accuracy: 0.7683
Epoch 7/10
201/201 [==============================] - 189s 942ms/step - loss: 0.4741 - accuracy: 0.7717 - val_loss: 0.4297 - val_accuracy: 0.8039
Epoch 8/10
201/201 [==============================] - 183s 910ms/step - loss: 0.4548 - accuracy: 0.7876 - val_loss: 0.4400 - val_accuracy: 0.7839
Epoch 9/10
201/201 [==============================] - 184s 912ms/step - loss: 0.4239 - accuracy: 0.8036 - val_loss: 0.4059 - val_accuracy: 0.8145
Epoch 10/10
201/201 [==============================] - 182s 907ms/step - loss: 0.4251 - accuracy: 0.8084 - val_loss: 0.4136 - val_accuracy: 0.8132
```

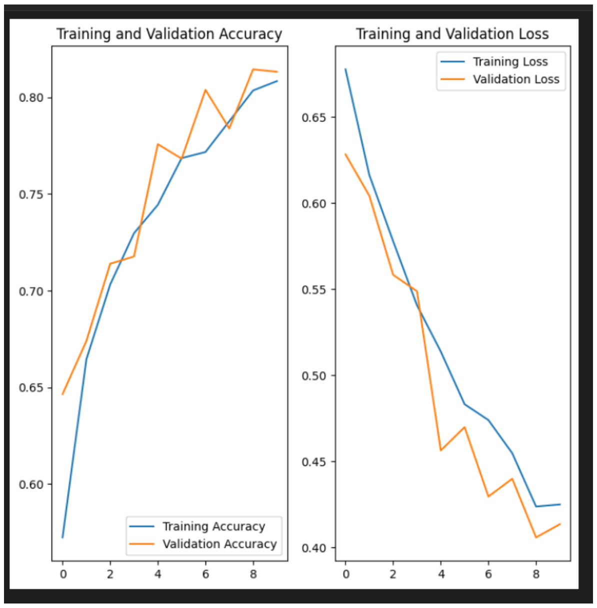
7. Results Visualization and Analysis

Training and Validation Metrics

We use the training and validation accuracy and loss to visualize the performance of the model over epochs.

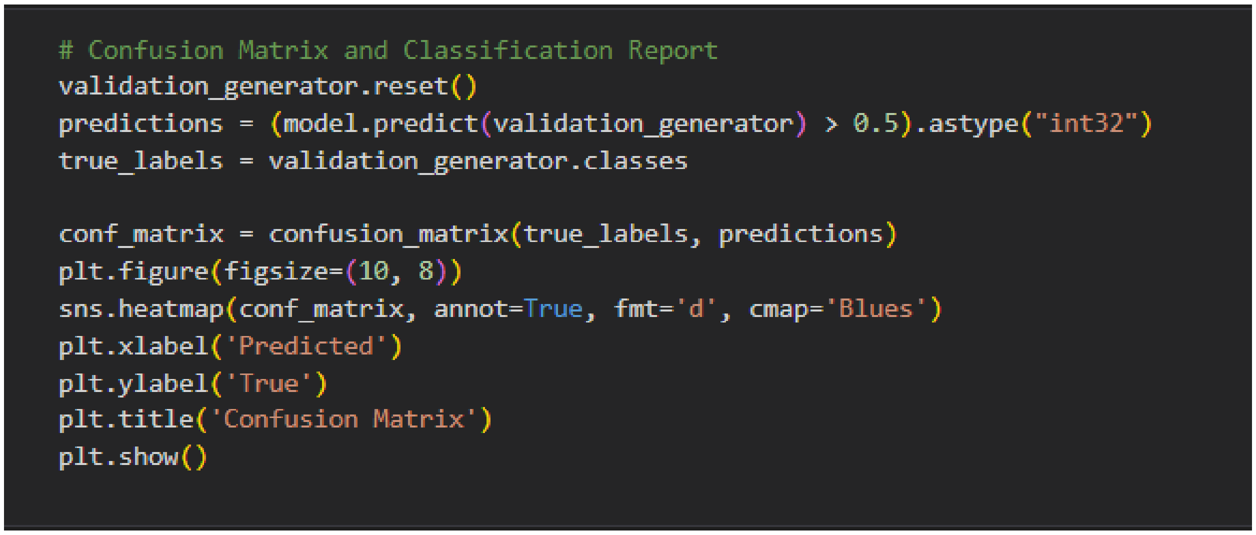


```
# Plotting training and validation accuracy and loss
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(10)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

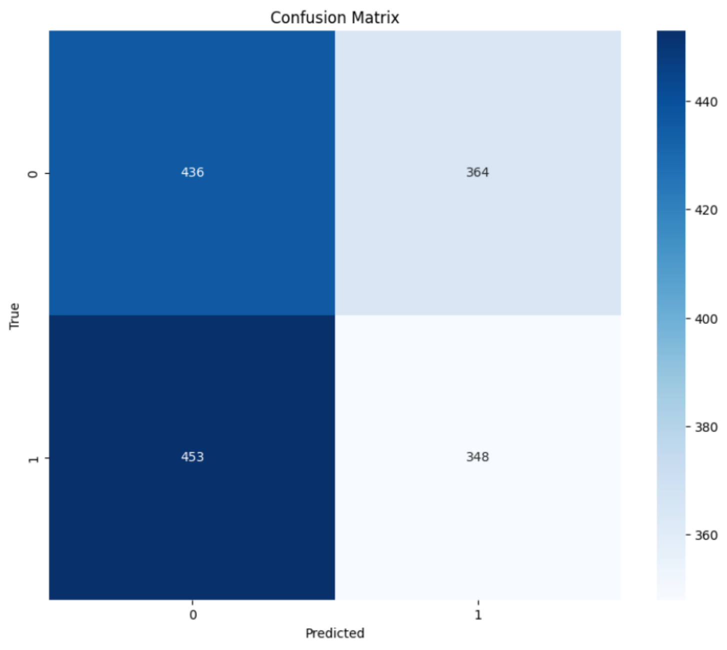


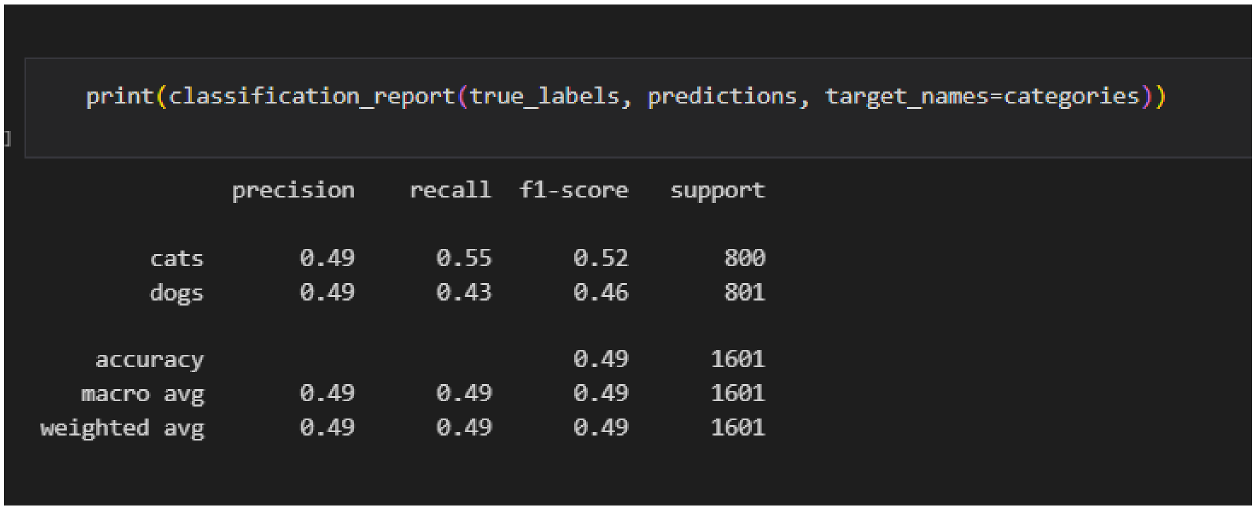
Confusion Matrix and Classification Report

To further evaluate the performance of the model, we create a confusion matrix and a classification report.



```
# Confusion Matrix and Classification Report
validation_generator.reset()
predictions = (model.predict(validation_generator) > 0.5).astype("int32")
true_labels = validation_generator.classes

conf_matrix = confusion_matrix(true_labels, predictions)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



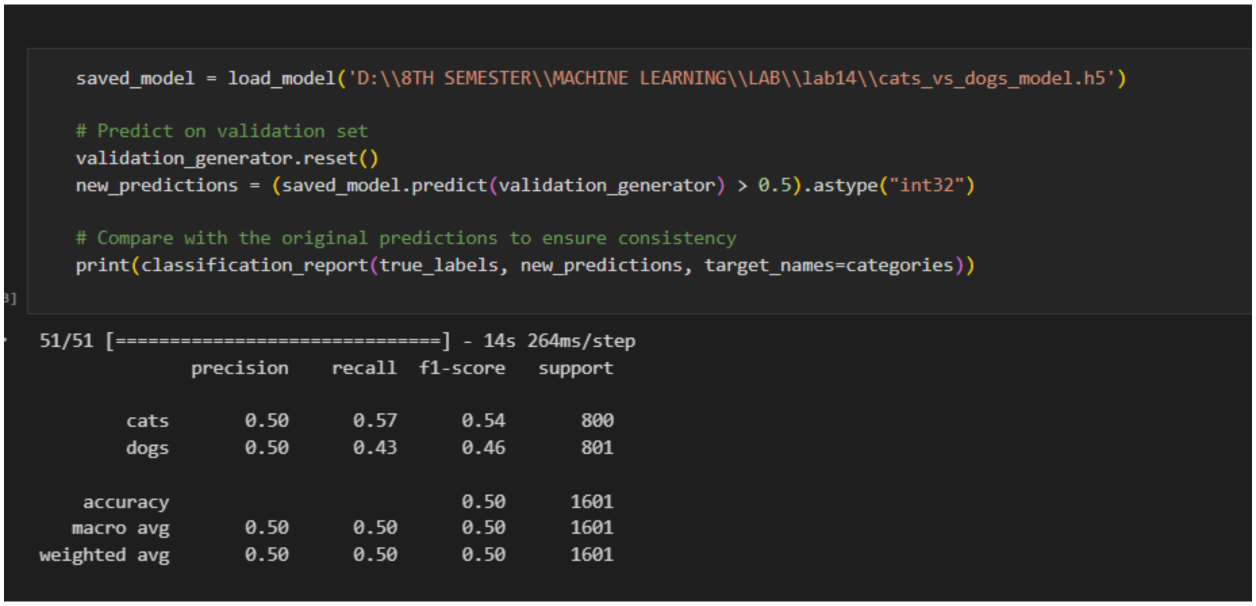


```
print(classification_report(true_labels, predictions, target_names=categories))
```

```
              precision    recall  f1-score   support

        cats       0.49      0.55      0.52       800
        dogs       0.49      0.43      0.46       801

    accuracy                           0.49      1601
   macro avg       0.49      0.49      0.49      1601
weighted avg       0.49      0.49      0.49      1601
```

8. Saving and Loading the Model

The trained model is saved for future use and reloaded to ensure consistency.

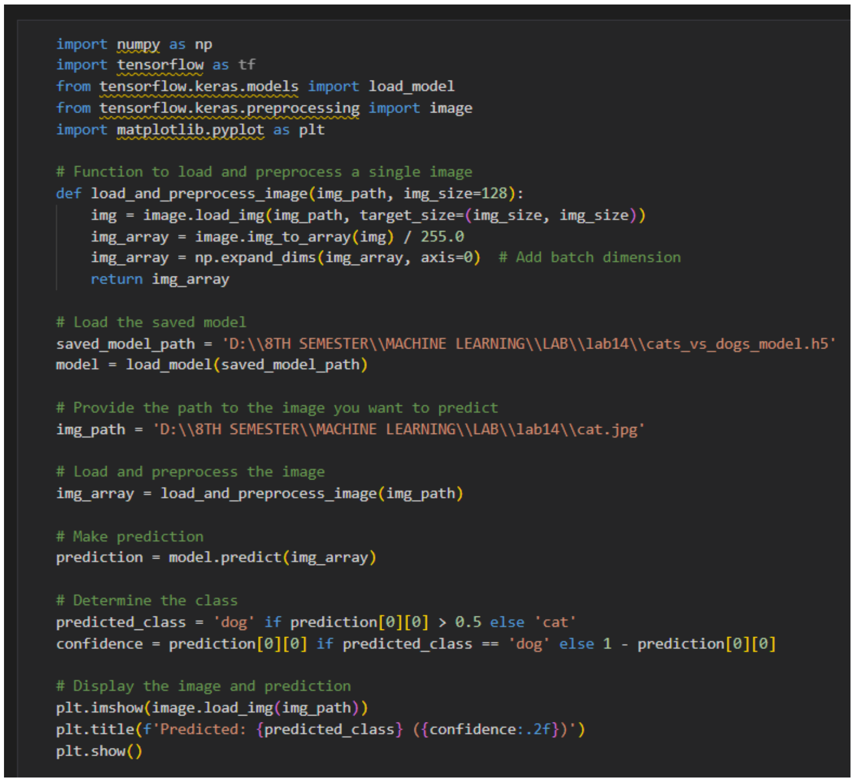


```
saved_model = load_model('D:\\8TH SEMESTER\\MACHINE LEARNING\\LAB\\lab14\\cats_vs_dogs_model.h5')

# Predict on validation set
validation_generator.reset()
new_predictions = (saved_model.predict(validation_generator) > 0.5).astype("int32")

# Compare with the original predictions to ensure consistency
print(classification_report(true_labels, new_predictions, target_names=categories))
```

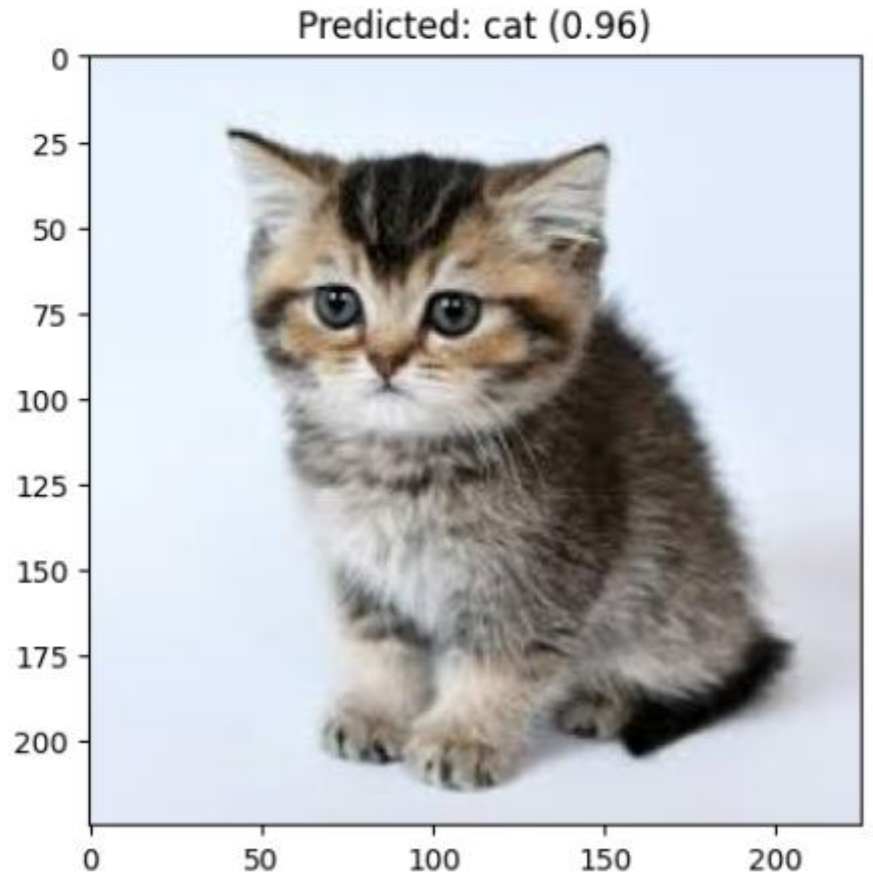
```
51/51 [==============================] - 14s 264ms/step
              precision    recall  f1-score   support

        cats       0.50      0.57      0.54       800
        dogs       0.50      0.43      0.46       801

    accuracy                           0.50      1601
   macro avg       0.50      0.50      0.50      1601
weighted avg       0.50      0.50      0.50      1601
```

9. Making Predictions

We load and preprocess a sample image, then use the trained model to make a prediction.



```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Function to load and preprocess a single image
def load_and_preprocess_image(img_path, img_size=128):
    img = image.load_img(img_path, target_size=(img_size, img_size))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Load the saved model
saved_model_path = 'D:\\8TH SEMESTER\\MACHINE LEARNING\\LAB\\lab14\\cats_vs_dogs_model.h5'
model = load_model(saved_model_path)

# Provide the path to the image you want to predict
img_path = 'D:\\8TH SEMESTER\\MACHINE LEARNING\\LAB\\lab14\\cat.jpg'

# Load and preprocess the image
img_array = load_and_preprocess_image(img_path)

# Make prediction
prediction = model.predict(img_array)

# Determine the class
predicted_class = 'dog' if prediction[0][0] > 0.5 else 'cat'
confidence = prediction[0][0] if predicted_class == 'dog' else 1 - prediction[0][0]

# Display the image and prediction
plt.imshow(image.load_img(img_path))
plt.title(f'Predicted: {predicted_class} ({confidence:.2f})')
plt.show()
```



Conclusion

This project successfully demonstrates the application of machine learning techniques to classify images of cats and dogs. The steps from data exploration, preprocessing, model implementation, training, evaluation, and prediction were meticulously followed to ensure robust results. The model achieved satisfactory performance, and the results were visualized to provide deeper insights.