Today we are predicting animals in a given folder if they are a cat or dog. Below is the script to compile and the final prediction results

In [ ]:

```
# Step 1: Import necessary libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import os

# Step 2: Data preprocessing
# Define paths
train_data_path = 'training_set'
test_data_path = 'test_set'
single_prediction_path = 'single_prediction'

# Image Data Generator for training set with augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

# Image Data Generator for test set without augmentation
test_datagen = ImageDataGenerator(rescale=1./255)

# Training set
training_set = train_datagen.flow_from_directory(
    train_data_path,
    target_size=(64, 64),
    batch_size=32,
    class_mode='binary'
)

# Test set
test_set = test_datagen.flow_from_directory(
    test_data_path,
    target_size=(64, 64),
    batch_size=32,
    class_mode='binary'
)

# Step 3: Build the CNN model
cnn = tf.keras.models.Sequential()

# Convolution
```

```python
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_
# Pooling
cnn.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2))

# Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2))

# Flattening
cnn.add(tf.keras.layers.Flatten())

# Full connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# Step 4: Compile the CNN
cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 5: Train the CNN
cnn.fit(x=training_set, validation_data=test_set, epochs=25)

# Step 6: Make predictions on the single_prediction images
def predict_image(image_path):
    test_image = image.load_img(image_path, target_size=(64, 64))
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = cnn.predict(test_image)
    if result[0][0] == 1:
        prediction = 'dog'
    else:
        prediction = 'cat'
    return prediction

# Predicting the images in single_prediction folder
image1_path = os.path.join(single_prediction_path, 'cat_or_dog_1.jpg')
image2_path = os.path.join(single_prediction_path, 'cat_or_dog_2.jpg')

print(f'cat_or_dog_1.jpg is a {predict_image(image1_path)}')
print(f'cat_or_dog_2.jpg is a {predict_image(image2_path)}')
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

```
C:\Users\joelr\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfr
a8p0\LocalCache\local-packages\Python311\site-packages\keras\src\layers\convolutiona
l\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument t
o a layer. When using Sequential models, prefer using an `Input(shape)` object as th
e first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/25
```

```
C:\Users\joelr\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfr
a8p0\LocalCache\local-packages\Python311\site-packages\keras\src\trainers\data_adapt
ers\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `supe
r().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_m
ultiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they w
ill be ignored.
  self._warn_if_super_not_called()
```

```
250/250 ━━━━━━━━━━━━━━━━━━━━ 44s 167ms/step - accuracy: 0.5851 - loss: 0.6659 - val_
accuracy: 0.7070 - val_loss: 0.5788
Epoch 2/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 10s 40ms/step - accuracy: 0.6983 - loss: 0.5791 - val_a
ccuracy: 0.7355 - val_loss: 0.5330
Epoch 3/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 41ms/step - accuracy: 0.7274 - loss: 0.5459 - val_a
ccuracy: 0.7440 - val_loss: 0.5184
Epoch 4/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 42ms/step - accuracy: 0.7482 - loss: 0.5149 - val_a
ccuracy: 0.7115 - val_loss: 0.5757
Epoch 5/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 42ms/step - accuracy: 0.7437 - loss: 0.5109 - val_a
ccuracy: 0.7365 - val_loss: 0.5736
Epoch 6/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 44ms/step - accuracy: 0.7691 - loss: 0.4822 - val_a
ccuracy: 0.7635 - val_loss: 0.4839
Epoch 7/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.7681 - loss: 0.4743 - val_a
ccuracy: 0.7850 - val_loss: 0.4649
Epoch 8/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 12s 45ms/step - accuracy: 0.7802 - loss: 0.4565 - val_a
ccuracy: 0.7855 - val_loss: 0.4687
Epoch 9/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 13s 50ms/step - accuracy: 0.8001 - loss: 0.4218 - val_a
ccuracy: 0.7865 - val_loss: 0.4480
Epoch 10/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 44ms/step - accuracy: 0.7973 - loss: 0.4239 - val_a
ccuracy: 0.7410 - val_loss: 0.5466
Epoch 11/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 42ms/step - accuracy: 0.8117 - loss: 0.4125 - val_a
ccuracy: 0.7695 - val_loss: 0.4963
Epoch 12/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.8149 - loss: 0.3965 - val_a
ccuracy: 0.7855 - val_loss: 0.4762
Epoch 13/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 44ms/step - accuracy: 0.8211 - loss: 0.3835 - val_a
ccuracy: 0.7865 - val_loss: 0.4703
Epoch 14/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.8313 - loss: 0.3712 - val_a
ccuracy: 0.7930 - val_loss: 0.4590
Epoch 15/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.8388 - loss: 0.3645 - val_a
ccuracy: 0.8015 - val_loss: 0.4575
Epoch 16/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 42ms/step - accuracy: 0.8436 - loss: 0.3548 - val_a
ccuracy: 0.8000 - val_loss: 0.4534
Epoch 17/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.8443 - loss: 0.3435 - val_a
ccuracy: 0.8180 - val_loss: 0.4331
Epoch 18/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 42ms/step - accuracy: 0.8624 - loss: 0.3243 - val_a
ccuracy: 0.7965 - val_loss: 0.4676
Epoch 19/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.8646 - loss: 0.3053 - val_a
ccuracy: 0.7870 - val_loss: 0.5267
```

```
Epoch 20/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 41ms/step - accuracy: 0.8752 - loss: 0.2950 - val_a
ccuracy: 0.8065 - val_loss: 0.4931
Epoch 21/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 10s 41ms/step - accuracy: 0.8761 - loss: 0.2948 - val_a
ccuracy: 0.8085 - val_loss: 0.5255
Epoch 22/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.8820 - loss: 0.2694 - val_a
ccuracy: 0.8035 - val_loss: 0.5153
Epoch 23/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 44ms/step - accuracy: 0.8919 - loss: 0.2655 - val_a
ccuracy: 0.7980 - val_loss: 0.5298
Epoch 24/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.8982 - loss: 0.2398 - val_a
ccuracy: 0.8060 - val_loss: 0.4879
Epoch 25/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.9007 - loss: 0.2396 - val_a
ccuracy: 0.8165 - val_loss: 0.5265
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
cat_or_dog_1.jpg is a dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
cat_or_dog_2.jpg is a cat
```

In [ ]: