Convolutional Neural Network

Importing the libraries

In [122…

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

In [123…

```
tf.__version__
```

Out[123…

```
'2.4.0'
```

Part 1 - Data Preprocessing

Preprocessing the Training set

In [124…

```
# augementing images by applying transformation
# recale normalizes each pixel (all values become between 0, 1)
# prenvents overfitting
train_datagen = ImageDataGenerator(
            rescale= 1./255,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True)
#import training_set
#class mode determines the type of classification
# each folder is a class
train_set = train_datagen.flow_from_directory(
    './dataset/training_set',
    target_size= (64, 64),
    batch_size=32,
    class_mode='binary')
```

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

Preprocessing the Test set

In [125…

```
# we only apply feature scaling for the test, but other transformation will not be appl
test_datagen = ImageDataGenerator(rescale= 1./255)
test_set = test_datagen.flow_from_directory(
    './dataset/test_set',
    target_size= (64, 64),
    batch_size=32,
    class_mode='binary')
```

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

Part 2 - Building the CNN

Initialising the CNN

In [126…

```
cnn = tf.keras.models.Sequential()
```

Step 1 - Convolution

In [127…

```
# number of freature detectors
# kernel_size is the size of the feature detector
# for input shape, since its rgb we'll have 3 2d arrays
# first layer needs input shape
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shap
```

Step 2 - Pooling

In [128…

```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Adding a second convolutional layer

In [129…

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Step 3 - Flattening

In [130…

```
cnn.add(tf.keras.layers.Flatten())
```

Step 4 - Full Connection

In [131…

```
# image data needs more processing power so we'll use more neurons
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

Step 5 - Output Layer

In [132…

```
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 3 - Training the CNN

Compiling the CNN

In [133…

```
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the CNN on the Training set and evaluating it on the Test set

In [134…

```
# started with 10 the accuracy was not converging
# cnn.fit(x = train_set, validation_data = test_set, epochs = 25)
# cnn.save('cnn.h5')
```

```
Epoch 1/25
250/250 [==============================] - 116s 463ms/step - loss: 0.6914 - accuracy: 0.
5528 - val_loss: 0.6143 - val_accuracy: 0.6495
Epoch 2/25
250/250 [==============================] - 50s 201ms/step - loss: 0.6142 - accuracy: 0.6
671 - val_loss: 0.5963 - val_accuracy: 0.6745
Epoch 3/25
250/250 [==============================] - 50s 202ms/step - loss: 0.5683 - accuracy: 0.7
062 - val_loss: 0.5235 - val_accuracy: 0.7415
Epoch 4/25
250/250 [==============================] - 50s 200ms/step - loss: 0.5340 - accuracy: 0.7
285 - val_loss: 0.5110 - val_accuracy: 0.7470
Epoch 5/25
250/250 [==============================] - 50s 201ms/step - loss: 0.4994 - accuracy: 0.7
560 - val_loss: 0.5220 - val_accuracy: 0.7515
Epoch 6/25
250/250 [==============================] - 51s 206ms/step - loss: 0.4767 - accuracy: 0.7
736 - val_loss: 0.5025 - val_accuracy: 0.7635
Epoch 7/25
250/250 [==============================] - 51s 202ms/step - loss: 0.4570 - accuracy: 0.7
866 - val_loss: 0.5030 - val_accuracy: 0.7715
Epoch 8/25
250/250 [==============================] - 51s 205ms/step - loss: 0.4539 - accuracy: 0.7
841 - val_loss: 0.4768 - val_accuracy: 0.7810
Epoch 9/25
250/250 [==============================] - 54s 217ms/step - loss: 0.4263 - accuracy: 0.8
074 - val_loss: 0.4665 - val_accuracy: 0.7810
Epoch 10/25
250/250 [==============================] - 50s 200ms/step - loss: 0.4034 - accuracy: 0.8
139 - val_loss: 0.5256 - val_accuracy: 0.7515
Epoch 11/25
250/250 [==============================] - 50s 201ms/step - loss: 0.3903 - accuracy: 0.8
222 - val_loss: 0.5156 - val_accuracy: 0.7730
Epoch 12/25
250/250 [==============================] - 51s 206ms/step - loss: 0.3805 - accuracy: 0.8
250 - val_loss: 0.4843 - val_accuracy: 0.7910
Epoch 13/25
250/250 [==============================] - 55s 222ms/step - loss: 0.3608 - accuracy: 0.8
389 - val_loss: 0.4547 - val_accuracy: 0.8045
Epoch 14/25
250/250 [==============================] - 50s 201ms/step - loss: 0.3435 - accuracy: 0.8
490 - val_loss: 0.4866 - val_accuracy: 0.7925
Epoch 15/25
250/250 [==============================] - 50s 200ms/step - loss: 0.3322 - accuracy: 0.8
509 - val_loss: 0.4704 - val_accuracy: 0.7930
Epoch 16/25
250/250 [==============================] - 51s 204ms/step - loss: 0.3130 - accuracy: 0.8
693 - val_loss: 0.4661 - val_accuracy: 0.8010
Epoch 17/25
250/250 [==============================] - 52s 207ms/step - loss: 0.3050 - accuracy: 0.8
671 - val_loss: 0.5540 - val_accuracy: 0.7780
Epoch 18/25
250/250 [==============================] - 50s 202ms/step - loss: 0.2761 - accuracy: 0.8
831 - val_loss: 0.4743 - val_accuracy: 0.8150
Epoch 19/25
250/250 [==============================] - 50s 201ms/step - loss: 0.2720 - accuracy: 0.8
871 - val_loss: 0.5050 - val_accuracy: 0.8095
Epoch 20/25
250/250 [==============================] - 50s 202ms/step - loss: 0.2459 - accuracy: 0.8
961 - val_loss: 0.5034 - val_accuracy: 0.8145
```

```
Epoch 21/25
250/250 [==============================] - 51s 202ms/step - loss: 0.2241 - accuracy: 0.9
068 - val_loss: 0.5210 - val_accuracy: 0.8140
Epoch 22/25
250/250 [==============================] - 54s 217ms/step - loss: 0.2080 - accuracy: 0.9
181 - val_loss: 0.5634 - val_accuracy: 0.8030
Epoch 23/25
250/250 [==============================] - 50s 201ms/step - loss: 0.2088 - accuracy: 0.9
193 - val_loss: 0.5588 - val_accuracy: 0.7940
Epoch 24/25
250/250 [==============================] - 51s 203ms/step - loss: 0.2081 - accuracy: 0.9
112 - val_loss: 0.5924 - val_accuracy: 0.7960
Epoch 25/25
250/250 [==============================] - 52s 206ms/step - loss: 0.1859 - accuracy: 0.9
255 - val_loss: 0.5838 - val_accuracy: 0.8070
```

In [163…

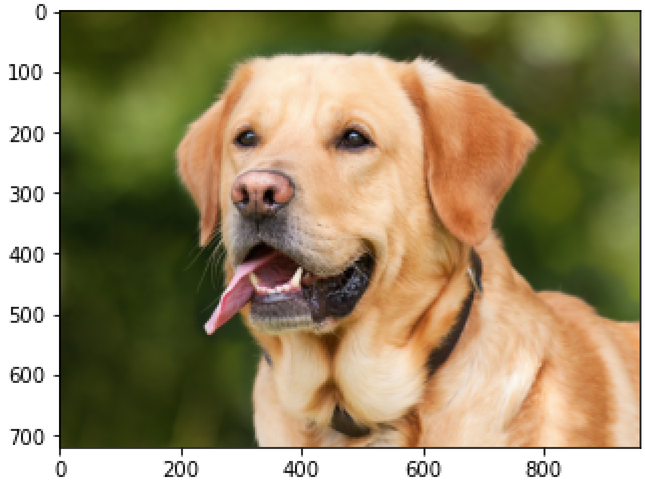
```
#load the saved model
from keras.models import load_model
cnn = tf.keras.models.load_model("cnn.h5")
```

Part 4 - Making a single prediction

In [165…

```
from keras.preprocessing import image
test_img = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg')
#original image
print('Original image:')
plt.imshow(test_img)
plt.show()

test_img = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg', target_size=(64
#convert img into array
test_img = image.img_to_array(test_img)
# add extra demension that corresponds to batch size
test_img = np.expand_dims(test_img, axis = 0)

result = cnn.predict(test_img)

# print(train_set.class_indices) cats = 0, dog = 1

if (result[0][0] > 0.5 ):
    print('dog')
else:
    print('cat')
```

```
Original image:
```



```
dog
```