Importing the libraries

In [1]:
```
import tensorflow as tf
```

Part 1 - Data Preprocessing

In [2]:
```
from keras.preprocessing.image import ImageDataGenerator
```

Preprocessing the Training set

In [3]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
```

In [4]:
```
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

```
Found 8048 images belonging to 2 classes.
```

Preprocessing the Test set

In [5]:
```
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 [6]:
```
# Initialising the CNN
from keras.models import Sequential
classifier = Sequential()
```

Step 1 - Convolution

```
In [7]: from keras.layers import Conv2D
        classifier.add(Conv2D(filters=32, kernel_size=3, activation='relu', input_shap
        e=[64, 64, 3]))
```

Step 2 - Pooling

```
In [8]: from keras.layers import MaxPooling2D
        classifier.add(MaxPooling2D(pool_size=2, strides=2))
```

Adding a second convolutional layer

```
In [9]: classifier.add(Conv2D(filters=32, kernel_size=3, activation='relu'))
        classifier.add(MaxPooling2D(pool_size=2, strides=2))
```

Step 3 - Flattening

```
In [10]: from keras.layers import Flatten
         classifier.add(Flatten())
```

Step 4 - Full Connection

```
In [11]: from keras.layers import Dense
         classifier.add(Dense(units = 128, activation = 'relu'))
```

Step 5 - Output Layer

```
In [12]: classifier.add(Dense(units = 1, activation = 'sigmoid'))
```

Part 3 - Training the CNN

Compiling the CNN

```
In [13]: classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
         ['accuracy'])
```

In [14]:

```
classifier.fit(x = training_set, validation_data = test_set, epochs = 25)
```

```
Epoch 1/25
252/252 [==============================] - 360s 1s/step - loss: 0.7000 - accu
racy: 0.5260 - val_loss: 0.6462 - val_accuracy: 0.6380
Epoch 2/25
252/252 [==============================] - 58s 229ms/step - loss: 0.6347 - ac
curacy: 0.6485 - val_loss: 0.5973 - val_accuracy: 0.6960
Epoch 3/25
252/252 [==============================] - 54s 213ms/step - loss: 0.6000 - ac
curacy: 0.6746 - val_loss: 0.6131 - val_accuracy: 0.6725
Epoch 4/25
252/252 [==============================] - 53s 212ms/step - loss: 0.5292 - ac
curacy: 0.7402 - val_loss: 0.5229 - val_accuracy: 0.7425
Epoch 5/25
252/252 [==============================] - 55s 217ms/step - loss: 0.5172 - ac
curacy: 0.7419 - val_loss: 0.4901 - val_accuracy: 0.7635
Epoch 6/25
252/252 [==============================] - 54s 213ms/step - loss: 0.5094 - ac
curacy: 0.7458 - val_loss: 0.5237 - val_accuracy: 0.7500
Epoch 7/25
252/252 [==============================] - 56s 224ms/step - loss: 0.4790 - ac
curacy: 0.7719 - val_loss: 0.4928 - val_accuracy: 0.7580
Epoch 8/25
252/252 [==============================] - 57s 228ms/step - loss: 0.4578 - ac
curacy: 0.7819 - val_loss: 0.5098 - val_accuracy: 0.7580
Epoch 9/25
252/252 [==============================] - 55s 219ms/step - loss: 0.4455 - ac
curacy: 0.7885 - val_loss: 0.4755 - val_accuracy: 0.7780
Epoch 10/25
252/252 [==============================] - 57s 225ms/step - loss: 0.4310 - ac
curacy: 0.7920 - val_loss: 0.4606 - val_accuracy: 0.7970
Epoch 11/25
252/252 [==============================] - 55s 217ms/step - loss: 0.3976 - ac
curacy: 0.8223 - val_loss: 0.5142 - val_accuracy: 0.7480
Epoch 12/25
252/252 [==============================] - 55s 218ms/step - loss: 0.3951 - ac
curacy: 0.8181 - val_loss: 0.4517 - val_accuracy: 0.7965
Epoch 13/25
252/252 [==============================] - 57s 226ms/step - loss: 0.3650 - ac
curacy: 0.8364 - val_loss: 0.4783 - val_accuracy: 0.7955
Epoch 14/25
252/252 [==============================] - 56s 222ms/step - loss: 0.3520 - ac
curacy: 0.8433 - val_loss: 0.5034 - val_accuracy: 0.7735
Epoch 15/25
252/252 [==============================] - 55s 217ms/step - loss: 0.3387 - ac
curacy: 0.8505 - val_loss: 0.4414 - val_accuracy: 0.8080
Epoch 16/25
252/252 [==============================] - 57s 225ms/step - loss: 0.3366 - ac
curacy: 0.8534 - val_loss: 0.5188 - val_accuracy: 0.7725
Epoch 17/25
252/252 [==============================] - 57s 224ms/step - loss: 0.3176 - ac
curacy: 0.8637 - val_loss: 0.4589 - val_accuracy: 0.7975
Epoch 18/25
252/252 [==============================] - 55s 219ms/step - loss: 0.2916 - ac
curacy: 0.8773 - val_loss: 0.4450 - val_accuracy: 0.8075
Epoch 19/25
252/252 [==============================] - 56s 222ms/step - loss: 0.2785 - ac
curacy: 0.8822 - val_loss: 0.4976 - val_accuracy: 0.7980
```

```
Epoch 20/25
252/252 [==============================] - 56s 224ms/step - loss: 0.2582 - ac
curacy: 0.8866 - val_loss: 0.5053 - val_accuracy: 0.7970
Epoch 21/25
252/252 [==============================] - 54s 215ms/step - loss: 0.2483 - ac
curacy: 0.8951 - val_loss: 0.5091 - val_accuracy: 0.7980
Epoch 22/25
252/252 [==============================] - 58s 230ms/step - loss: 0.2238 - ac
curacy: 0.9109 - val_loss: 0.5420 - val_accuracy: 0.7960
Epoch 23/25
252/252 [==============================] - 57s 224ms/step - loss: 0.2189 - ac
curacy: 0.9132 - val_loss: 0.5457 - val_accuracy: 0.8035
Epoch 24/25
252/252 [==============================] - 55s 220ms/step - loss: 0.2078 - ac
curacy: 0.9158 - val_loss: 0.5489 - val_accuracy: 0.7990
Epoch 25/25
252/252 [==============================] - 56s 223ms/step - loss: 0.1947 - ac
curacy: 0.9210 - val_loss: 0.5761 - val_accuracy: 0.7930
```

Out[14]:

```
<tensorflow.python.keras.callbacks.History at 0x1febabd3c10>
```

Part 4 - Making a single prediction

In [15]:

```
import numpy as np
from keras.preprocessing import image
```

In [16]:

```
test_image = image.load_img('dataset/single_prediction/cat_or_dog_2.jpg', targ
et_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
```

In [17]:

```
result = classifier.predict(test_image)
training_set.class_indices
```

Out[17]:

```
{'cats': 0, 'dogs': 1}
```

In [18]:

```
if result[0][0] == 1:
  prediction = 'dog'
else:
  prediction = 'cat'
```

In [19]:

```
print(prediction)
```

```
cat
```