```
# Convolutional Neural Network

# Importing the libraries
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
tf.__version__
```

```
'2.8.2'
```

```
! unzip dataset.zip
```

Data Preprocessing

Here we augment the data (images) with Keras. ImageDataGenerator helps us to rotate or augment the data in the real time while training the data

```
# Part 1 - Data Preprocessing

# Preprocessing the Training set

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

# Preprocessing the Test set
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 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

Building the CNN

In this part, we build the convolution neural network with keras. At every layer relu activation is used except in the output layer. There we have used Sigmoid activation since we have 2 class classification.

```
# Part 2 - Building the CNN

# Initialising the CNN
cnn = tf.keras.models.Sequential()

# Step 1 - Convolution
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[

# Step 2 - Pooling
cnn.add(tf.keras.layers.MaxPool2D(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.MaxPool2D(pool_size=2, strides=2))

# Step 3 - Flattening
cnn.add(tf.keras.layers.Flatten())

# Step 4 - Full Connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

# Step 5 - Output Layer
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))



# Part 3 - Training the CNN

# Compiling the CNN
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Training the CNN on the Training set and evaluating it on the Test set
cnn.fit(x = training_set, validation_data = test_set, epochs = 100)
```

```
Epoch 29/100
250/250 [==============================] - 66s 265ms/step - loss: 0.1931 - accurac
Epoch 30/100
250/250 [==============================] - 66s 265ms/step - loss: 0.1838 - accurac
Epoch 31/100
250/250 [==============================] - 66s 265ms/step - loss: 0.1749 - accurac
Epoch 32/100
250/250 [==============================] - 66s 266ms/step - loss: 0.1638 - accurac
Epoch 33/100
250/250 [==============================] - 69s 276ms/step - loss: 0.1549 - accurac
Epoch 34/100
250/250 [==============================] - 66s 265ms/step - loss: 0.1616 - accurac
Epoch 35/100
250/250 [==============================] - 67s 266ms/step - loss: 0.1365 - accurac
Epoch 36/100
250/250 [==============================] - 67s 266ms/step - loss: 0.1329 - accurac
Epoch 37/100
250/250 [==============================] - 66s 265ms/step - loss: 0.1225 - accurac
Epoch 38/100
250/250 [==============================] - 66s 265ms/step - loss: 0.1282 - accurac
Epoch 39/100
250/250 [==============================] - 67s 266ms/step - loss: 0.1137 - accurac
Epoch 40/100
```

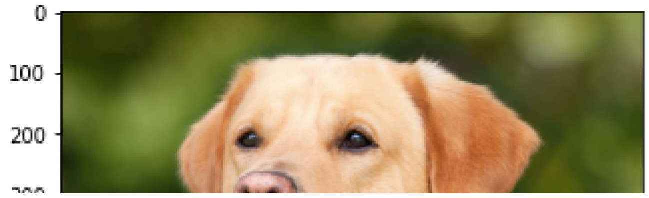
```
250/250 [==============================] - 66s 265ms/step - loss: 0.1180 - accurac
Epoch 41/100
250/250 [==============================] - 66s 265ms/step - loss: 0.1187 - accurac
Epoch 42/100
250/250 [==============================] - 67s 266ms/step - loss: 0.1006 - accurac
Epoch 43/100
250/250 [==============================] - 67s 266ms/step - loss: 0.1044 - accurac
Epoch 44/100
250/250 [==============================] - 66s 266ms/step - loss: 0.1017 - accurac
Epoch 45/100
250/250 [==============================] - 67s 267ms/step - loss: 0.1051 - accurac
Epoch 46/100
250/250 [==============================] - 66s 265ms/step - loss: 0.0856 - accurac
Epoch 47/100
250/250 [==============================] - 66s 265ms/step - loss: 0.0847 - accurac
Epoch 48/100
250/250 [==============================] - 66s 266ms/step - loss: 0.0908 - accurac
Epoch 49/100
250/250 [==============================] - 66s 265ms/step - loss: 0.0869 - accurac
Epoch 50/100
250/250 [==============================] - 67s 266ms/step - loss: 0.0779 - accurac
Epoch 51/100
250/250 [==============================] - 67s 266ms/step - loss: 0.0874 - accurac
Epoch 52/100
250/250 [==============================] - 67s 266ms/step - loss: 0.0763 - accurac
Epoch 53/100
250/250 [==============================] - 66s 265ms/step - loss: 0.0777 - accurac
Epoch 54/100
250/250 [==============================] - 66s 265ms/step - loss: 0.0786 - accurac
Epoch 55/100
250/250 [==============================] - 66s 265ms/step - loss: 0.0700 - accurac
Epoch 56/100
250/250 [==============================] - 66s 265ms/step - loss: 0.0752 - accurac
Epoch 57/100
```

Prediction

Dog

```
%pylab inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('dataset/single_prediction/cat_or_dog_1.jpg')
imgplot = plt.imshow(img)
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



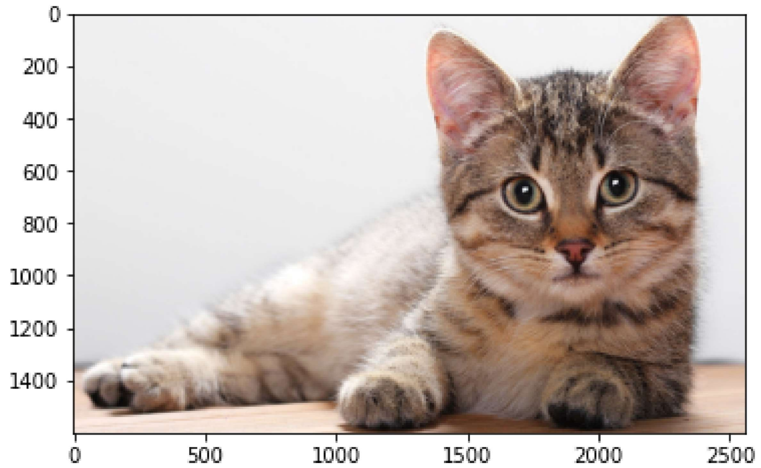
```
# Part 4 - Making a single prediction

import numpy as np
from keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg', target_size = (6
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
    prediction = 'dog'
else:
    prediction = 'cat'
print(prediction)
```

```
dog
```

Cat

```
%pylab inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('dataset/single_prediction/cat_or_dog_2.jpg')
imgplot = plt.imshow(img)
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```




```
test_image = image.load_img('dataset/single_prediction/cat_or_dog_2.jpg', target_size = (6
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
```

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

```
cat
```