Convolutional Neural Network

Esc+M Shift+Enter

In [1]:
```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

In [2]:
```
tf.__version__
```

Out[2]:
```
'2.8.0'
```

Part 1 - Data Preprocessing

Preprocessing the Training set

In [3]:
```
train_datagen = ImageDataGenerator(rescale = 1./255, #ImageDataGenerator-->Class;tool that apply all transformation images of train set
                                   shear_range = 0.2, #rescale-->feature scaling,output value will be between 0-255
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
training_set = train_datagen.flow_from_directory('dataset/training_set', #dataset/training_set->path of the folder dataset
                                                 target_size = (64, 64), #Will be fed
                                                 batch_size = 32,
                                                 class_mode = 'binary') #class_mode-->as the output type binary
#Image augmentation has been done here->If we didn't do here after training training set accuracy would be high,overfitted and test set accuracy would be low.
```

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

Preprocessing the Test set

In [4]:
```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64), #have to be same as training set
                                            batch_size = 32,
                                            class_mode = 'binary')
```

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

Part 2 - Building the CNN

Initialising the CNN

In [5]:
```
cnn = tf.keras.models.Sequential() #will allow to create ANN as a sequence of layers.Keras->Library,Models->module
```

Step 1 - Convolution

In [6]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64, 64, 3])) #layers->module,Conv2D->Class
#input_shape-->input dataset dimension
```

Step 2 - Pooling

In [7]:
```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
#pool_size-->the size of specific frame from feature map to creat pooled featured map
#strides-->sliding/shifting size
```

Adding a second convolutional layer

In [8]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu')) #kernal_size-->feature detector dimension
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Step 3 - Flattening

In [9]:
```
cnn.add(tf.keras.layers.Flatten()) #flatting to one dim vector
```

Step 4 - Full Connection

In [10]:
```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu')) #Dense-->Class;units-->number of hidden neurons
```

Step 5 - Output Layer

In [11]:
```
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid')) #unit=1-->One output;sigmoid-->recommend for binary classsification
#softmax-->multiclass classification
```

Part 3 - Training the CNN

Compiling the CNN

In [12]:
```
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
#accuracy-->relevant way to measure the performance of classification method
```

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

In [13]:
```
cnn.fit(x = training_set, validation_data = test_set, epochs = 25) #fit()-->method;train cnn on trainning set
#validation_data-->to evaluate test set
#batch_size(32) * 250=8000 total train image;250->steps in each epoch;first accuracy->training set,second accuracy->test set
```

```
Epoch 1/25
250/250 [==============================] - 108s 429ms/step - loss: 0.6773 - accuracy: 0.5764 - val_loss: 0.6742 - val_accuracy: 0.5660
Epoch 2/25
250/250 [==============================] - 40s 161ms/step - loss: 0.6382 - accuracy: 0.6361 - val_loss: 0.5855 - val_accuracy: 0.6960
Epoch 3/25
250/250 [==============================] - 37s 149ms/step - loss: 0.5909 - accuracy: 0.6846 - val_loss: 0.6059 - val_accuracy: 0.6755
Epoch 4/25
250/250 [==============================] - 37s 146ms/step - loss: 0.5511 - accuracy: 0.7140 - val_loss: 0.5263 - val_accuracy: 0.7380
Epoch 5/25
250/250 [==============================] - 38s 153ms/step - loss: 0.5218 - accuracy: 0.7402 - val_loss: 0.5280 - val_accuracy: 0.7420
Epoch 6/25
250/250 [==============================] - 38s 151ms/step - loss: 0.4968 - accuracy: 0.7549 - val_loss: 0.4965 - val_accuracy: 0.7595
Epoch 7/25
250/250 [==============================] - 38s 153ms/step - loss: 0.4814 - accuracy: 0.7673 - val_loss: 0.4854 - val_accuracy: 0.7730
Epoch 8/25
250/250 [==============================] - 37s 150ms/step - loss: 0.4653 - accuracy: 0.7749 - val_loss: 0.4739 - val_accuracy: 0.7835
Epoch 9/25
250/250 [==============================] - 38s 152ms/step - loss: 0.4510 - accuracy: 0.7830 - val_loss: 0.4546 - val_accuracy: 0.7895
Epoch 10/25
250/250 [==============================] - 36s 144ms/step - loss: 0.4305 - accuracy: 0.7996 - val_loss: 0.4708 - val_accuracy: 0.8010
Epoch 11/25
250/250 [==============================] - 37s 147ms/step - loss: 0.4185 - accuracy: 0.8075 - val_loss: 0.4525 - val_accuracy: 0.8035
Epoch 12/25
250/250 [==============================] - 36s 145ms/step - loss: 0.4019 - accuracy: 0.8151 - val_loss: 0.4818 - val_accuracy: 0.7800
Epoch 13/25
250/250 [==============================] - 35s 138ms/step - loss: 0.3868 - accuracy: 0.8240 - val_loss: 0.4532 - val_accuracy: 0.8070
Epoch 14/25
250/250 [==============================] - 35s 141ms/step - loss: 0.3818 - accuracy: 0.8278 - val_loss: 0.4790 - val_accuracy: 0.7920
Epoch 15/25
250/250 [==============================] - 37s 146ms/step - loss: 0.3629 - accuracy: 0.8350 - val_loss: 0.4870 - val_accuracy: 0.7805
Epoch 16/25
250/250 [==============================] - 35s 140ms/step - loss: 0.3574 - accuracy: 0.8385 - val_loss: 0.5146 - val_accuracy: 0.7825
Epoch 17/25
250/250 [==============================] - 35s 138ms/step - loss: 0.3449 - accuracy: 0.8476 - val_loss: 0.5012 - val_accuracy: 0.8150
Epoch 18/25
250/250 [==============================] - 36s 146ms/step - loss: 0.3313 - accuracy: 0.8574 - val_loss: 0.4679 - val_accuracy: 0.7970
Epoch 19/25
250/250 [==============================] - 36s 145ms/step - loss: 0.3235 - accuracy: 0.8568 - val_loss: 0.4898 - val_accuracy: 0.8015
Epoch 20/25
250/250 [==============================] - 36s 144ms/step - loss: 0.3112 - accuracy: 0.8610 - val_loss: 0.5189 - val_accuracy: 0.7910
Epoch 21/25
250/250 [==============================] - 36s 143ms/step - loss: 0.2918 - accuracy: 0.8755 - val_loss: 0.4998 - val_accuracy: 0.8060
Epoch 22/25
250/250 [==============================] - 37s 149ms/step - loss: 0.2895 - accuracy: 0.8708 - val_loss: 0.4905 - val_accuracy: 0.8040
Epoch 23/25
250/250 [==============================] - 88s 354ms/step - loss: 0.2782 - accuracy: 0.8814 - val_loss: 0.5112 - val_accuracy: 0.7960
Epoch 24/25
250/250 [==============================] - 43s 174ms/step - loss: 0.2552 - accuracy: 0.8939 - val_loss: 0.5353 - val_accuracy: 0.7970
Epoch 25/25
250/250 [==============================] - 37s 148ms/step - loss: 0.2590 - accuracy: 0.8889 - val_loss: 0.5017 - val_accuracy: 0.8035
```

Out[13]:
```
<keras.callbacks.History at 0x1cfb8171390>
```

Part 4 - Making a single prediction

In [14]:
```
import numpy as np
from keras.preprocessing import image #image-->submodule
test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg', target_size = (64, 64)) #load_img-->specific function of image submodule
test_image = image.img_to_array(test_image) #img_to_array-->converting test image(pil formate) to numpy array
test_image = np.expand_dims(test_image, axis = 0) #expand_dims()-->to add extra dim corresponding to batch
result = cnn.predict(test_image)                  #axix=0-->dim of batch whice are adding image will be the first dimension
training_set.class_indices
if result[0][0] == 1: #batch index zero,only one element(test image) that also index zero
  prediction = 'dog'
else:
  prediction = 'cat'
```

In [15]:
```
print(prediction)
```

```
dog
```