Part-1 -Building the CNN

Importing Libraries and packages

In [1]:
```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

Initialising the CNN

In [2]:
```
#Naming it as classifier as we would be classifying only two persons
classifier = Sequential()
```

Steps involved in building a CNN

Step 1.Convolution step 2.max pooling step 3.flatten step 4.Full connection

Step 1:Convolution:This step consists of applying different filters/kernels onto the image and with the help of feature detector we get a reduced size of a image which is known as feature map

In [3]:
```
#add method to the object classifier:
classifier.add(Convolution2D(32,3,3,input_shape=(64,64,3),activation =
'relu'))
#Here in arguments 32 denotes 32 feature detectors and 3*3 dimensions.
#that means CNN contains 32 feature maps
#input_shape is required as the images  we use in the dataset are all o
f not same size.therefore to bring them all to a common shape we use th
is 64,64 denotes the pixel size and 3 denotes no of channels
#Activation function:Rectifier function-relu
```

```
C:\Anaconda-Spyder\lib\site-packages\ipykernel_launcher.py:2: UserWarni
ng: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (3, 3), i
nput_shape=(64, 64, 3..., activation="relu")`
```

Step 2: Max pooling

In [4]:
```
#If the size of the feature map is even then pooled feature map is divi
ded by 2 or it is divided by 2 and 1 is added if the size is odd as we
 are using a stride of size 2

classifier.add(MaxPooling2D(pool_size=(2,2)))
```

In [5]:
```
classifier.add(Convolution2D(32,3,3,activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))
```

```
C:\Anaconda-Spyder\lib\site-packages\ipykernel_launcher.py:1: UserWarni
ng: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (3, 3), a
ctivation="relu")`
  """Entry point for launching an IPython kernel.
```

Step 3: Flattening

In [6]:
```
classifier.add(Flatten())
```

Step 4: Full connection

In [7]:
```
#Dense is the function used for full connected layer
classifier.add(Dense(output_dim = 128, activation = 'relu'))
classifier.add(Dense(output_dim = 1, activation = 'sigmoid'))
```

```
C:\Anaconda-Spyder\lib\site-packages\ipykernel_launcher.py:2: UserWarni
ng: Update your `Dense` call to the Keras 2 API: `Dense(activation="rel
u", units=128)`

C:\Anaconda-Spyder\lib\site-packages\ipykernel_launcher.py:3: UserWarni
ng: Update your `Dense` call to the Keras 2 API: `Dense(activation="sig
moid", units=1)`
  This is separate from the ipykernel package so we can avoid doing imp
orts until
```

In [8]:
```
classifier.compile(optimizer = 'adam',loss= 'binary_crossentropy',metri
cs=['accuracy'])
```

Part 2: Image Preprocessing step

In [9]:
```
#imported the code from keras.io
from keras.preprocessing.image import ImageDataGenerator
```

In [10]:
```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)
```

In [11]:
```
training_set = train_datagen.flow_from_directory(
        'C://Users//HP//Desktop//DATASET//training_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')
```

```
Found 8 images belonging to 2 classes.
```

In [12]:
```
test_set = test_datagen.flow_from_directory(
        'C://Users//HP//Desktop//DATASET//test_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')
```

```
Found 4 images belonging to 2 classes.
```

In [13]:
```
classifier.fit_generator(
        training_set,
        steps_per_epoch=8,
        epochs=25,
        validation_data=test_set,
        validation_steps=2)
```

```
Epoch 1/25
8/8 [==============================] - 9s 1s/step - loss: 0.5609 - accu
racy: 0.7031 - val_loss: 0.5671 - val_accuracy: 0.5000
Epoch 2/25
8/8 [==============================] - 6s 719ms/step - loss: 0.1885 - a
ccuracy: 0.9844 - val_loss: 0.6252 - val_accuracy: 0.7500
Epoch 3/25
8/8 [==============================] - 5s 621ms/step - loss: 0.0216 - a
ccuracy: 1.0000 - val_loss: 0.7537 - val_accuracy: 0.5000
Epoch 4/25
8/8 [==============================] - 5s 651ms/step - loss: 0.0015 - a
ccuracy: 1.0000 - val_loss: 0.8963 - val_accuracy: 0.5000
Epoch 5/25
8/8 [==============================] - 5s 602ms/step - loss: 5.3231e-04
- accuracy: 1.0000 - val_loss: 1.0105 - val_accuracy: 0.5000
Epoch 6/25
```

```
8/8 [==============================] - 7s 845ms/step - loss: 1.1009e-04
- accuracy: 1.0000 - val_loss: 0.9270 - val_accuracy: 0.5000
Epoch 7/25
8/8 [==============================] - 6s 743ms/step - loss: 8.3751e-05
- accuracy: 1.0000 - val_loss: 0.9181 - val_accuracy: 0.5000
Epoch 8/25
8/8 [==============================] - 6s 773ms/step - loss: 6.7404e-05
- accuracy: 1.0000 - val_loss: 0.8298 - val_accuracy: 0.5000
Epoch 9/25
8/8 [==============================] - 7s 849ms/step - loss: 5.2757e-05
- accuracy: 1.0000 - val_loss: 0.7534 - val_accuracy: 0.5000
Epoch 10/25
8/8 [==============================] - 8s 1s/step - loss: 3.3359e-05 -
accuracy: 1.0000 - val_loss: 0.6738 - val_accuracy: 0.5000
Epoch 11/25
8/8 [==============================] - 8s 1s/step - loss: 2.1280e-05 -
accuracy: 1.0000 - val_loss: 0.6343 - val_accuracy: 0.5000
Epoch 12/25
8/8 [==============================] - 8s 976ms/step - loss: 3.6741e-05
- accuracy: 1.0000 - val_loss: 0.5857 - val_accuracy: 0.5000
Epoch 13/25
8/8 [==============================] - 8s 1s/step - loss: 7.2055e-05 -
accuracy: 1.0000 - val_loss: 0.5526 - val_accuracy: 0.7500
Epoch 14/25
8/8 [==============================] - 8s 1s/step - loss: 2.1895e-05 -
accuracy: 1.0000 - val_loss: 0.5029 - val_accuracy: 0.7500
Epoch 15/25
8/8 [==============================] - 9s 1s/step - loss: 3.9524e-05 -
accuracy: 1.0000 - val_loss: 0.5461 - val_accuracy: 0.5000
Epoch 16/25
8/8 [==============================] - 8s 993ms/step - loss: 1.2732e-05
- accuracy: 1.0000 - val_loss: 0.4673 - val_accuracy: 0.5000
Epoch 17/25
8/8 [==============================] - 8s 948ms/step - loss: 1.8677e-05
- accuracy: 1.0000 - val_loss: 0.3693 - val_accuracy: 0.7500
Epoch 18/25
8/8 [==============================] - 7s 845ms/step - loss: 1.1949e-05
- accuracy: 1.0000 - val_loss: 0.3544 - val_accuracy: 0.7500
Epoch 19/25
```

```
8/8 [==============================] - 6s 773ms/step - loss: 1.4764e-05
- accuracy: 1.0000 - val_loss: 0.3804 - val_accuracy: 0.7500
Epoch 20/25
8/8 [==============================] - 5s 671ms/step - loss: 1.0037e-05
- accuracy: 1.0000 - val_loss: 0.4044 - val_accuracy: 0.7500
Epoch 21/25
8/8 [==============================] - 6s 727ms/step - loss: 8.8998e-06
- accuracy: 1.0000 - val_loss: 0.3371 - val_accuracy: 0.7500
Epoch 22/25
8/8 [==============================] - 6s 783ms/step - loss: 6.6364e-06
- accuracy: 1.0000 - val_loss: 0.2917 - val_accuracy: 0.7500
Epoch 23/25
8/8 [==============================] - 8s 982ms/step - loss: 8.0996e-06
- accuracy: 1.0000 - val_loss: 0.2883 - val_accuracy: 0.7500
Epoch 24/25
8/8 [==============================] - 7s 834ms/step - loss: 8.1810e-06
- accuracy: 1.0000 - val_loss: 0.3009 - val_accuracy: 0.7500
Epoch 25/25
8/8 [==============================] - 7s 867ms/step - loss: 1.0678e-05
- accuracy: 1.0000 - val_loss: 0.3389 - val_accuracy: 0.7500
```

Out[13]: `<keras.callbacks.callbacks.History at 0x25b1172be10>`

We got an accuracy of 100% for training set and 75% for test set

Part 3: Making predictions

In [31]:
```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('C://Users//HP//Desktop//DATASET//single_prediction//brad_or_dicaprio1.jpg',target_size=(64, 64))
```

In [32]:
```
test_image = image.img_to_array(test_image)
```

In [33]:
```
test_image = np.expand_dims(test_image,axis = 0)
```

In [34]:
```
classifier.predict(test_image)
```

Out[34]:
```
array([[0.]], dtype=float32)
```

In [35]:
```
result = classifier.predict(test_image)
```

In [36]:
```
training_set.class_indices
```

Out[36]:
```
{'BradPitt': 0, 'DiCaprio': 1}
```

In [29]:
```
if result[0][0] ==1:
    prediction = 'DiCaprio'
else:
    prediction = 'BradPitt'
```

In [37]:
```
print(result)
```

```
[[0.]]
```

In [ ]: