In [ ]:

```
#Importing the necessary libraries

import numpy as np
import tensorflow.keras
from keras.models import Sequential
from keras.layers import Conv2D , MaxPool2D , Dropout , Flatten , Dense , Activation
import cv2
import imutils
from imutils import paths
```

In [43]:

```
#For reading a CSV file we use pandas.read_csv() but for images we use a slightly different method.
#For images, we first list the images using , list(paths.list_images()) and then read each image with cv2.imread().
#In order to avoid the batch explosion during matrices multiplication , we normalise our input matrices by dividing it by 255. As grayscal
#We will store the images matrices in a list named cat_images.

cats = list(paths.list_images("D:\Study\Deep Learning\catstrain"))
print(len(cats))
cat_imgs=[]
cat_shape=[]
for i in cats:
    cat_img=cv2.imread(i)
    cat_img=cat_img/255
    cat_imgs.append(cat_img)
    cat_shape.append(cat_img.shape)
    print(cat_img.shape)
```

```
20
(281, 300, 3)
(500, 489, 3)
(410, 432, 3)
(225, 300, 3)
(315, 500, 3)
(268, 320, 3)
(353, 406, 3)
(259, 448, 3)
(375, 500, 3)
(375, 500, 3)
(224, 320, 3)
(397, 312, 3)
(375, 500, 3)
(415, 500, 3)
(375, 500, 3)
(144, 176, 3)
(304, 400, 3)
(500, 495, 3)
(346, 461, 3)
(426, 320, 3)
```

In [30]:

```
#With exactly same procedure we import and read the dogs images as well and store them in a list named cat_images.


dogs = list(paths.list_images("D:\Study\Deep Learning\dogstrain"))
#print(dogs)
dog_imgs=[]
dog_shape=[]
for i in dogs:
    dog_img=cv2.imread(i)
    dog_img=dog_img/255
    dog_imgs.append(dog_img)
    dog_shape.append(dog_img.shape)
    print(dog_img.shape)
```

```
(500, 327, 3)
(293, 269, 3)
(102, 135, 3)
(162, 98, 3)
(428, 363, 3)
(387, 500, 3)
(375, 500, 3)
(381, 500, 3)
(336, 272, 3)
(348, 216, 3)
(225, 300, 3)
(199, 188, 3)
(333, 500, 3)
(375, 500, 3)
(288, 300, 3)
(376, 500, 3)
(488, 500, 3)
(264, 300, 3)
(500, 470, 3)
(500, 369, 3)
```

In [44]:

```
#When we give the images to the Convolutional Neural Network than the input size of images should be fixed .
#So , we need  to check wether all the images are of same size or not and if not than we will resize them to same size.

cats_avg_shape=np.array(cat_shape).sum(axis=0)/20
print(cats_avg_shape)
cat_imgs = np.array(cat_imgs)
cat_new_imgs=[]
for i in cat_imgs:
    i.resize((256,256,3) , refcheck=False)
    cat_new_imgs.append(i)
```

```
[343.35 408.95   3.  ]

C:\Users\Ankita Sharma\AppData\Local\Temp\ipykernel_17744\781627601.py:6: VisibleDeprecationWarning: Creating an ndarray fr
om ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is de
precated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  cat_imgs = np.array(cat_imgs)
```

In [45]:

```
#now lets have a look wether all the cat images are of same size or not.

for i in cat_new_imgs:
    print(i.shape)
```

```
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
```

In [46]:

```
#Similarly as before we will first check the dog images sizes and resize them.

dog_avg_shape = np.array(dog_shape).sum(axis=0)/20
print(dogs_avg_shape)
dog_new_imgs=[]
for i in dog_imgs:
    i.resize((256,256,3) , refcheck=False)
    dog_new_imgs.append(i)
    print(i.shape)
```

```
[343.   355.35   3.  ]
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
(256, 256, 3)
```

In [47]:

```
#As we don't we have the labels list , so we will create one for ourselves.

labels_cat = np.array(["Cat" for i in np.arange(20)])
labels_dog = np.array(["Dog" for i in np.arange(20)])
```

In [48]:

```
labels_cat
```

Out[48]:

```
array(['Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat',
       'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat',
       'Cat', 'Cat'], dtype='<U3')
```

In [49]:

```
#We will turn our list of images into numpy arrays so that we can concatenate them.

cat_new_imgs = np.array(cat_new_imgs)
dog_new_imgs = np.array(dog_new_imgs)
```

In [50]:

```
#Creating a single input array that contains both Cat and Dog images.

X = np.concatenate((cat_new_imgs , dog_new_imgs))
X
```

```
        [[0.61568627, 0.50588235, 0.47843137],
         [0.49019608, 0.38039216, 0.35294118],
         [0.38823529, 0.27843137, 0.25098039],
         ...,
         [0.58431373, 0.41176471, 0.34509804],
         [0.59215686, 0.41960784, 0.35294118],
         [0.59215686, 0.41960784, 0.35294118]],

        [[0.58823529, 0.41960784, 0.36078431],
         [0.6       , 0.43137255, 0.37254902],
         [0.61568627, 0.44313725, 0.39215686],
         ...,
         [0.62745098, 0.52941176, 0.60784314],
         [0.55686275, 0.4627451 , 0.53333333],
         [0.60784314, 0.51372549, 0.58431373]],

        ...,

        [[0.1254902 , 0.16862745, 0.24705882],
         [0.16470588, 0.20784314, 0.28627451],
```

In [38]:

```
#Also creating a single array for labels

labels = np.concatenate((labels_cat,labels_dog) , axis=0)
labels
```

Out[38]:

```
array(['Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat',
       'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat',
       'Cat', 'Cat', 'Dog', 'Dog', 'Dog', 'Dog', 'Dog', 'Dog', 'Dog',
       'Dog', 'Dog', 'Dog', 'Dog', 'Dog', 'Dog', 'Dog', 'Dog', 'Dog',
       'Dog', 'Dog', 'Dog', 'Dog'], dtype='<U3')
```

importing the OneHotEncoder from ScikitLearn, to convert our labels into binary matrix representing our labels with 0 and 1 ,bcz we know machine can't interpret text.

In [42]:

```
from sklearn.preprocessing import OneHotEncoder
```

In [40]:

```
#Creating the encoder object and then converting labels into binary matrix with fit_transform().
#to view our output matrix we need to turn it toarray()

encoder = OneHotEncoder()
labels=labels.reshape(-1,1)
y=encoder.fit_transform(labels)
#y1 = encoder.transform(y).toarray()
y1=y.toarray()
```

In [41]:

```
#let's have a look at our label matrix

y1
```

Out[41]:

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.]])
```

In [52]:

```
#Now it's the time to create our CNN model.

#We will create our model as prescribed to us in the guidelines

#Project Guidelines:
#  • Begin by creating the ipynb file in the same parent folder where the downloaded data set is kept. The CNN model should have the follo

#  • Input layer

#  • Convolutional layer 1 with 32 filters of kernel size[5,5]

#  • Pooling layer 1 with pool size[2,2] and stride 2

#  • Convolutional layer 2 with 64 filters of kernel size[5,5]

#  • Pooling layer 2 with pool size[2,2] and stride 2

#  •Dense layer whose output size is fixed in the hyper parameter: fc_size=32

#  • Dropout layer with dropout probability 0.4

#  •Predict the class by doing a softmax on the output of the dropout layers.


model = Sequential()
model.add(Conv2D(32 , (5,5) , input_shape=(256,256,3)))
model.add(MaxPool2D(pool_size=(2,2) , strides=2))
model.add(Conv2D(64 , (5,5)))
model.add(MaxPool2D(pool_size=(2,2) , strides=2))
model.add(Flatten())
model.add(Dense(32))
model.add(Activation(activation='relu'))
#model.add(Dropout(0.4))
model.add(Dense(16))
model.add(Activation(activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2))
model.add(Activation(activation='softmax'))
```

In [53]:

```
#Compiling our Model.

model.compile(optimizer='adam' , loss=tensorflow.losses.BinaryCrossentropy() , metrics=[tensorflow.metrics.BinaryAccuracy()])
```

In [26]:

```
#Finally training the model with our input matrices and labels.
#Firstly with epochs=100 and then in next step we will train with epochs=200.

model.fit(x=X, y=y1 , epochs=100)
```

```
Epoch 93/100
2/2 [==============================] - 3s 583ms/step - loss: 0.2409 - binary_accuracy: 0.9000
Epoch 94/100
2/2 [==============================] - 3s 587ms/step - loss: 0.2510 - binary_accuracy: 0.9000
Epoch 95/100
2/2 [==============================] - 3s 589ms/step - loss: 0.1902 - binary_accuracy: 0.9250
Epoch 96/100
2/2 [==============================] - 3s 583ms/step - loss: 0.2139 - binary_accuracy: 0.9250
Epoch 97/100
2/2 [==============================] - 3s 582ms/step - loss: 0.1966 - binary_accuracy: 0.9250
Epoch 98/100
2/2 [==============================] - 3s 575ms/step - loss: 0.1986 - binary_accuracy: 0.9250
Epoch 99/100
2/2 [==============================] - 3s 574ms/step - loss: 0.2454 - binary_accuracy: 0.9500
Epoch 100/100
2/2 [==============================] - 3s 569ms/step - loss: 0.1817 - binary_accuracy: 0.9750
```

Out[26]:

```
<keras.callbacks.History at 0x1be06f08ee0>
```

In [27]:

```
model.fit(x=X , y=y1 , epochs=200)
```

```
Epoch 193/200
2/2 [==============================] - 3s 583ms/step - loss: 0.0639 - binary_accuracy: 0.9750
Epoch 194/200
2/2 [==============================] - 3s 603ms/step - loss: 0.0375 - binary_accuracy: 1.0000
Epoch 195/200
2/2 [==============================] - 3s 582ms/step - loss: 0.0835 - binary_accuracy: 1.0000
Epoch 196/200
2/2 [==============================] - 3s 600ms/step - loss: 0.0358 - binary_accuracy: 1.0000
Epoch 197/200
2/2 [==============================] - 3s 593ms/step - loss: 0.1038 - binary_accuracy: 0.9750
Epoch 198/200
2/2 [==============================] - 3s 586ms/step - loss: 0.0958 - binary_accuracy: 0.9750
Epoch 199/200
2/2 [==============================] - 3s 602ms/step - loss: 0.1840 - binary_accuracy: 0.9750
Epoch 200/200
2/2 [==============================] - 3s 588ms/step - loss: 0.0653 - binary_accuracy: 0.9750
```

Out[27]:

```
<keras.callbacks.History at 0x1be09175520>
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: