In [1]:
```
!unzip '/content/drive/My Drive/Covid19Pred/Dataset_kaggle.zip'
```

```
Archive:  /content/drive/My Drive/Covid19Pred/Dataset_kaggle.zip
   creating: Dataset_kaggle/
   creating: Dataset_kaggle/COVID/
 extracting: Dataset_kaggle/COVID/Covid (1).png
  inflating: Dataset_kaggle/COVID/Covid (10).png
  inflating: Dataset_kaggle/COVID/Covid (100).png
  inflating: Dataset_kaggle/COVID/Covid (1000).png
  inflating: Dataset_kaggle/COVID/Covid (1001).png
  inflating: Dataset_kaggle/COVID/Covid (1002).png
  inflating: Dataset_kaggle/COVID/Covid (1003).png
  inflating: Dataset_kaggle/COVID/Covid (1004).png
  inflating: Dataset_kaggle/COVID/Covid (1005).png
  inflating: Dataset_kaggle/COVID/Covid (1006).png
  inflating: Dataset_kaggle/COVID/Covid (1007).png
  inflating: Dataset_kaggle/COVID/Covid (1008).png
  inflating: Dataset_kaggle/COVID/Covid (1009).png
  inflating: Dataset_kaggle/COVID/Covid (101).png
  inflating: Dataset_kaggle/COVID/Covid (1010).png
  inflating: Dataset_kaggle/COVID/Covid (1011).png
```

In [2]:
```
import os
import cv2
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
from keras.layers.normalization import BatchNormalization
```

```
Using TensorFlow backend.
```

In [3]:
```
yes=os.listdir('/content/Dataset_kaggle/COVID')
no=os.listdir('/content/Dataset_kaggle/non-COVID')
```

In [4]:
```
data=np.concatenate([yes,no])
len(data)==len(yes)+len(no)
```

Out[4]:
```
True
```

In [5]:
```
target_x=np.full(len(yes),1)
target_y=np.full(len(no),0)
data_target=np.concatenate([target_x,target_y])
```

In [6]:
```
yes_values=os.listdir('/content/Dataset_kaggle/COVID')
no_values=os.listdir('/content/Dataset_kaggle/non-COVID')
```

In [7]:
```
X_data =[]
for file in yes_values:
    img = cv2.imread('/content/Dataset_kaggle/COVID/'+file)
    face = cv2.resize(img, (227, 227) )
    (b, g, r)=cv2.split(face)
    img=cv2.merge([r,g,b])
    X_data.append(img)
```

In [8]:
```
for file in no_values:
    img = cv2.imread('/content/Dataset_kaggle/non-COVID/'+file)
    face = cv2.resize(img, (227, 227) )
    (b, g, r)=cv2.split(face)
    img=cv2.merge([r,g,b])
    X_data.append(img)
```

In [9]:
```
X = np.squeeze(X_data)
```

In [10]:
```
X = X.astype('float32')
X /= 255
```

In [11]:
```
x_train,x_test,y_train,y_test=train_test_split(X, data_target, test_size=0.2, ra
```

In [12]:
```
#Now let us define AlexNet CNN model to train the model
model=Sequential()

#1 conv layer
model.add(Conv2D(filters=96,kernel_size=(11,11),strides=(4,4),padding="valid",ac

#1 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())

#2 conv layer
model.add(Conv2D(filters=256,kernel_size=(5,5),strides=(1,1),padding="valid",act

#2 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())

#3 conv layer
model.add(Conv2D(filters=384,kernel_size=(3,3),strides=(1,1),padding="valid",act

#4 conv layer
model.add(Conv2D(filters=384,kernel_size=(3,3),strides=(1,1),padding="valid",act

#5 conv layer
model.add(Conv2D(filters=256,kernel_size=(3,3),strides=(1,1),padding="valid",act

#3 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())


model.add(Flatten())

#1 dense layer
model.add(Dense(4096,input_shape=(227,227,3),activation="relu"))

model.add(Dropout(0.4))

model.add(BatchNormalization())

#2 dense layer
model.add(Dense(4096,activation="relu"))

model.add(Dropout(0.4))

model.add(BatchNormalization())

#3 dense layer
model.add(Dense(1000,activation="relu"))

model.add(Dropout(0.4))

model.add(BatchNormalization())
```

```
#output layer
model.add(Dense(20,activation="softmax"))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 55, 55, 96)        34944
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 27, 27, 96)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 27, 27, 96)        384
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 23, 23, 256)       614656
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 11, 256)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 11, 11, 256)       1024
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 9, 9, 384)         885120
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 384)         1327488
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 5, 256)         884992
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              4198400
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
batch_normalization_4 (Batch (None, 4096)              16384
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 4096)              16384
_________________________________________________________________
dense_3 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_3 (Dropout)          (None, 1000)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 1000)              4000
_________________________________________________________________
dense_4 (Dense)              (None, 20)                20020
=================================================================
Total params: 28,883,132
Trainable params: 28,863,532
```

```
Non-trainable params: 19,600
_________________________________________________________________
```

In [13]:
```
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=
```

In [14]:
```
history=model.fit(x_train, y_train, epochs=1000, batch_size=128,validation_data=
```

```
Train on 1984 samples, validate on 497 samples
Epoch 1/1000
1984/1984 [==============================] - 12s 6ms/step - loss: 2.2376 - ac
curacy: 0.6003 - val_loss: 586.0645 - val_accuracy: 0.0000e+00
Epoch 2/1000
1984/1984 [==============================] - 3s 2ms/step - loss: 0.8529 - acc
uracy: 0.7999 - val_loss: 474.6297 - val_accuracy: 0.4990
Epoch 3/1000
1984/1984 [==============================] - 3s 2ms/step - loss: 0.5380 - acc
uracy: 0.8387 - val_loss: 461.3928 - val_accuracy: 0.4990
Epoch 4/1000
1984/1984 [==============================] - 3s 2ms/step - loss: 0.3496 - acc
uracy: 0.8765 - val_loss: 147.5300 - val_accuracy: 0.4990
Epoch 5/1000
1984/1984 [==============================] - 3s 2ms/step - loss: 0.2611 - acc
uracy: 0.9022 - val_loss: 141.1609 - val_accuracy: 0.4990
Epoch 6/1000
1984/1984 [==============================] - 3s 2ms/step - loss: 0.2466 - acc
uracy: 0.9047 - val_loss: 85.4918 - val_accuracy: 0.4990
```

In [15]:
```
final_loss, final_acc = model.evaluate(x_test, y_test, verbose=0)
print('The final accuracy is ',final_acc)
```

```
The final accuracy is  0.9738430380821228
```

In [15]:
```

```