Welcome to my Reseach Project Code!

The topic of my research is to identify from Xray images if a person is normal, has pneumonia or covid. In this project, I have prepared the dataset, then used the Sequential Model of CNN to classify images.

I have used the X-Ray images dataset available on Kaggle for my research. It contains the X-Ray images of normal people, of people having pneumonia and of people having Covid-19.

The link for my dataset is available here :
https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database
(https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database)

Importing Packages

To download dataset directly from Kaggle if using Google Colab. This will load the dataset in the folder in Colab.

```
-> import os

-> os.environ['KAGGLE_USERNAME'] = "mrana14"

->  os.environ['KAGGLE_KEY'] = "0f0101b72d3483ba0b2684a0fc2b2a99"

->  !kaggle datasets download tawsifurrahman/covid19-radiography-da
tabase

-> !unzip covid19-radiography-database.zip
```

In [1]:
```
import os
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from numpy.core.multiarray import asarray
from sklearn.model_selection import train_test_split
import tensorflow as tf
#To install tensorflow, refer this article - https://caffeinedev.medium
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dens
```

During my Research I performed two sets of Image-Classification.

1. First iteration involved only classification of Normal and Covid-19 X-Ray Images.

2. Second iteration involved classification of all three types - Normal Covid-19 and Viral

I will show the implementation of both.

Iteration 1 - Normal and Covid-19 Image Classification Using CNN

The following two command gives count of images in Normal and COVID folder of the dataset

In [2]:
```
1 len(os.listdir('COVID-19_Radiography_Dataset/Normal/images'))
```

Out[2]: 10192

In [3]:
```
1 len(os.listdir('COVID-19_Radiography_Dataset/COVID/images'))
```

Out[3]: 3616

The next cell uses opencv command cv2.imread to read an image. This is to just visualize and understand the data we have.

In [4]:
```
1 img = cv2.imread('COVID-19_Radiography_Dataset/Normal/images/Normal-10.
```

The next cell uses matplotlib.-pyplot library to display the image

In [5]:
```
1 plt.imshow(img)
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x290860eb0>
```

This command shows the shape of the image which is 299pixels *299 pixels with color depth of 3. It has also been verified for all images that they are of the same shape. If they would not have been of the same dimensions, resizing them to the same dimension would have been necessary.

In [6]:
```
1 img.shape
```

Out[6]:
```
(299, 299, 3)
```

The below code is written to understand how I passed the urls to the loadImages function for all image categories.
'links' is used to store all the names of the files(images in our case) in that directory / folder.
'path' will store the complete path for the images.

In [7]:
```
1 links = os.listdir('COVID-19_Radiography_Dataset/COVID/images')
```

In [8]:
```
1 path = "COVID-19_Radiography_Dataset/COVID/images/" + links[0]
```

In [9]:
```
1 path
```

Out[9]:
```
'COVID-19_Radiography_Dataset/COVID/images/COVID-1.png'
```

Writing the the loadImages function

Now, to prepare the dataset for training, it shoould contain all normal and covid images. For this we need to load all the images. I have loaded each category of images separately as well as have stored the labels / targets values separately.

images[ ] is a list to all store the images and labels[ ] stores the labels for those images - whether it is 0 or 1 where:
0 -> Normal
1 -> Covid

1. We will pass the path, links/urls and target of each image to load them into the list.
2. The for loop ensures that the images are read one-by-one. And the complete image path will be stored in the variable image_path.
3. The image is now read using opencv by using the cv2.imread() function.
4. I have then performed the standardization of images by dividing it by 255.
5. The images are then stored in the images list and the targets are stored in labels.
6. Here, we have also typecasted the lists to arrays

In [10]:
```
def loadImages(path, links, target):
    images = []
    labels = []
    for i in range(len(links)):
        image_path = path + links[i]
        img = cv2.imread(image_path)
        #print("1",img)
        img = img / 255.0
        #print("2",img)
        img = cv2.resize(img, (100,100)) #if images were of different s
        images.append(img)
        labels.append(target)
    images = np.asarray(images)
    return images, labels
```

The normal_path contains the path for normal images and the normal_links contians the image urls. These two values are then passed to the load_images function which then returns the images and the targets. The step is repeated for covid images as well.

In [11]:
```
normal_path = "COVID-19_Radiography_Dataset/Normal/images/"
normal_links = os.listdir(normal_path)
normal_images, normal_targets = loadImages(normal_path, normal_links, 0
```

In [12]:
```
covid_path = "COVID-19_Radiography_Dataset/COVID/images/"
covid_links = os.listdir('COVID-19_Radiography_Dataset/COVID/images')
covid_images, covid_targets = loadImages(covid_path, covid_links, 1)
```

In [13]:
```
#covid_images = np.asarray(covid_images)
```

In [14]:
```
#len(covid_images)
```

In [15]:
```
#covid_images.shape
```

In [16]:
```
#normal_images = np.asarray(normal_images)
```

In [17]:
```
#normal_images.shape
```

In [18]:
```
#normal_images
```

np.r_ is a numpy function used to stack data row-wise. Since we have all the data stored in variables, we can simply pass those variables and store in in the data variable for images and in the targets variable for all targets.

In [19]:
```
data = np.r_[covid_images, normal_images]
```

In [20]:
```
data.shape
```

Out[20]:
```
(13808, 100, 100, 3)
```

In [21]:
```
#data[5]
```

In [22]:
```
targets = np.r_[covid_targets, normal_targets]
```

In [23]:
```
targets.shape
```

Out[23]:
```
(13808,)
```

I have now split the data into training and testing and used the 75-25 split. Since the data will be shuffled anyways, hence stacking them is not a problem.

In [24]:
```
x_train, x_test, y_train, y_test = train_test_split(data, targets, test
```

In [25]:
```
#print(x_train)
```

Now, for model building, I have already imported tensorflow as well as the Sequenctial model from keras.
In the Sequential model, the layers are simply passed one-by-one and are stacked one-by-one.

Now I created a Sequential object and stored it in a model variable. And I will now passing the layers.
The first layer I passed is the Conv2D. In Conv2D, the paramters passed are - the number of filters, I have chosen 32. One can choose 16, 32, 64, etc. The next paramter that I have passed is th kernel_size of 3.

I have then passed the imnput_shape of our data which is 100*100*3 in our case.
Additionally, I have used the activation type as "relu" which is usually the recommended one. I have

Now, my next layer is the MaxPool2D. In this by default, a pool_size of (2,2) will be considered. I have then added the Conv2D and MaxPool2D 2 more times. We do not need to pass the input_shape again.

Now, I will add the Flatten() layer which will add the input layer which will basically contain the neurons.
Post this, I have added the hidden layer and have passed 512 neurons with activation type as 'relu'. For the output layer, as number of outputs will be 2, in that case the activation type will be "softmax". In my case, I wanted only one output and wanted to use the sigmoid activation type.

I have additionally added one more dense layer to add more neurons to the input.

In [ ]:
```
1
```

In [26]:
```
model = Sequential([
    Conv2D(32, 3, input_shape=(100,100,3), activation='relu'),
    MaxPooling2D(),
    Conv2D(16, 3, activation='relu'),
    MaxPooling2D(),
    Conv2D(16, 3, activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

```
Metal device set to: Apple M1
```

I have added three functions to calculate the F1-score , Recall and Precision to as the metrics for model performance

In [27]:

```
from keras.callbacks import Callback,ModelCheckpoint
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout
from keras.wrappers.scikit_learn import KerasClassifier
import keras.backend as K

def get_f1(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision
```

In [28]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 32)        896

 max_pooling2d (MaxPooling2D  (None, 49, 49, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 47, 47, 16)        4624

 max_pooling2d_1 (MaxPooling  (None, 23, 23, 16)       0
 2D)

 conv2d_2 (Conv2D)           (None, 21, 21, 16)        2320

 max_pooling2d_2 (MaxPooling  (None, 10, 10, 16)       0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 512)               819712

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 1)                 257

=================================================================
Total params: 959,137
Trainable params: 959,137
Non-trainable params: 0
_________________________________________________________________
```

Now, it is important to compile the model and for this, I have added the optimizer as 'adam' - one of the commonly used optimizers in Deep Learning.

The loss function here I have used is Binary Crossentropy as I have 2 outputs.

The metrics I wanted to focus on was accuracy, hence I have passed that.

In [29]:
```
le(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy(),metrics=['ac
```

Now, to train the model, I used the model.fit() method and passed the training data. I passed the default batch size of 32 and an epoch value of 5 to train 5 epochs, to begin with. I have passed the test data for now in validation_data.

As we see, the accuracy has improved with epochs, as well as in comparison to the training accuracy, the test accuracy looks good.

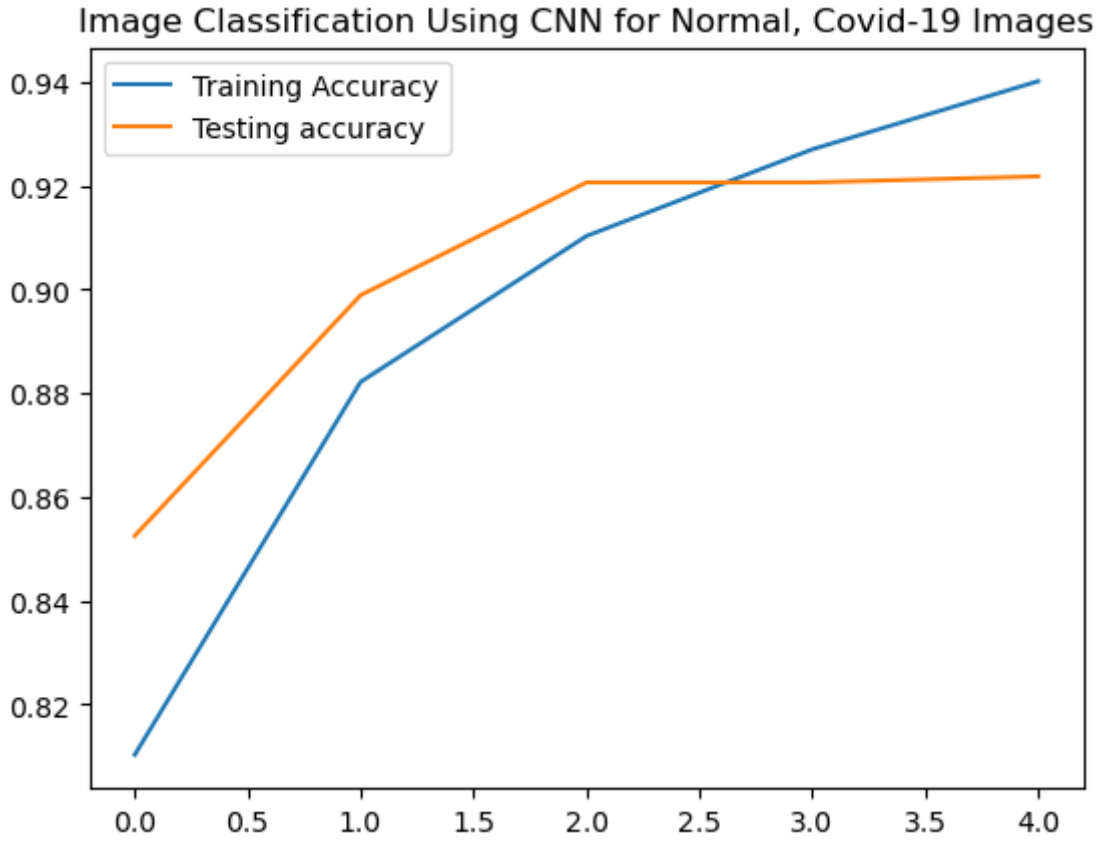
In [30]:
```
model.fit(x_train, y_train,batch_size=32,epochs=5,validation_data=(x_te
```

```
Epoch 1/5

2023-06-12 15:23:53.058282: W tensorflow/tsl/platform/profile_utils/cpu_u
tils.cc:128] Failed to get CPU frequency: 0 Hz

324/324 [==============================] - 11s 31ms/step - loss: 0.3959 -
accuracy: 0.8104 - mse: 0.1291 - get_f1: 0.5117 - recall_m: 0.4978 - prec
ision_m: 0.5996 - val_loss: 0.3174 - val_accuracy: 0.8525 - val_mse: 0.10
13 - val_get_f1: 0.6138 - val_recall_m: 0.5039 - val_precision_m: 0.8636
Epoch 2/5
324/324 [==============================] - 9s 29ms/step - loss: 0.2773 -
accuracy: 0.8822 - mse: 0.0850 - get_f1: 0.7636 - recall_m: 0.7638 - prec
ision_m: 0.7994 - val_loss: 0.2310 - val_accuracy: 0.8989 - val_mse: 0.07
13 - val_get_f1: 0.7823 - val_recall_m: 0.7779 - val_precision_m: 0.8179
Epoch 3/5
324/324 [==============================] - 9s 29ms/step - loss: 0.2174 -
accuracy: 0.9103 - mse: 0.0654 - get_f1: 0.8191 - recall_m: 0.8223 - prec
ision_m: 0.8428 - val_loss: 0.1921 - val_accuracy: 0.9206 - val_mse: 0.05
78 - val_get_f1: 0.8253 - val_recall_m: 0.7984 - val_precision_m: 0.8786
Epoch 4/5
324/324 [==============================] - 9s 29ms/step - loss: 0.1810 -
accuracy: 0.9270 - mse: 0.0540 - get_f1: 0.8536 - recall_m: 0.8564 - prec
ision_m: 0.8736 - val_loss: 0.1905 - val_accuracy: 0.9206 - val_mse: 0.05
79 - val_get_f1: 0.8412 - val_recall_m: 0.8977 - val_precision_m: 0.8066
Epoch 5/5
324/324 [==============================] - 10s 30ms/step - loss: 0.1517 -
accuracy: 0.9401 - mse: 0.0445 - get_f1: 0.8820 - recall_m: 0.8846 - prec
ision_m: 0.8934 - val_loss: 0.1898 - val_accuracy: 0.9218 - val_mse: 0.05
80 - val_get_f1: 0.8358 - val_recall_m: 0.8332 - val_precision_m: 0.8558
```

Out[30]:
```
<keras.callbacks.History at 0x291bcbb20>
```

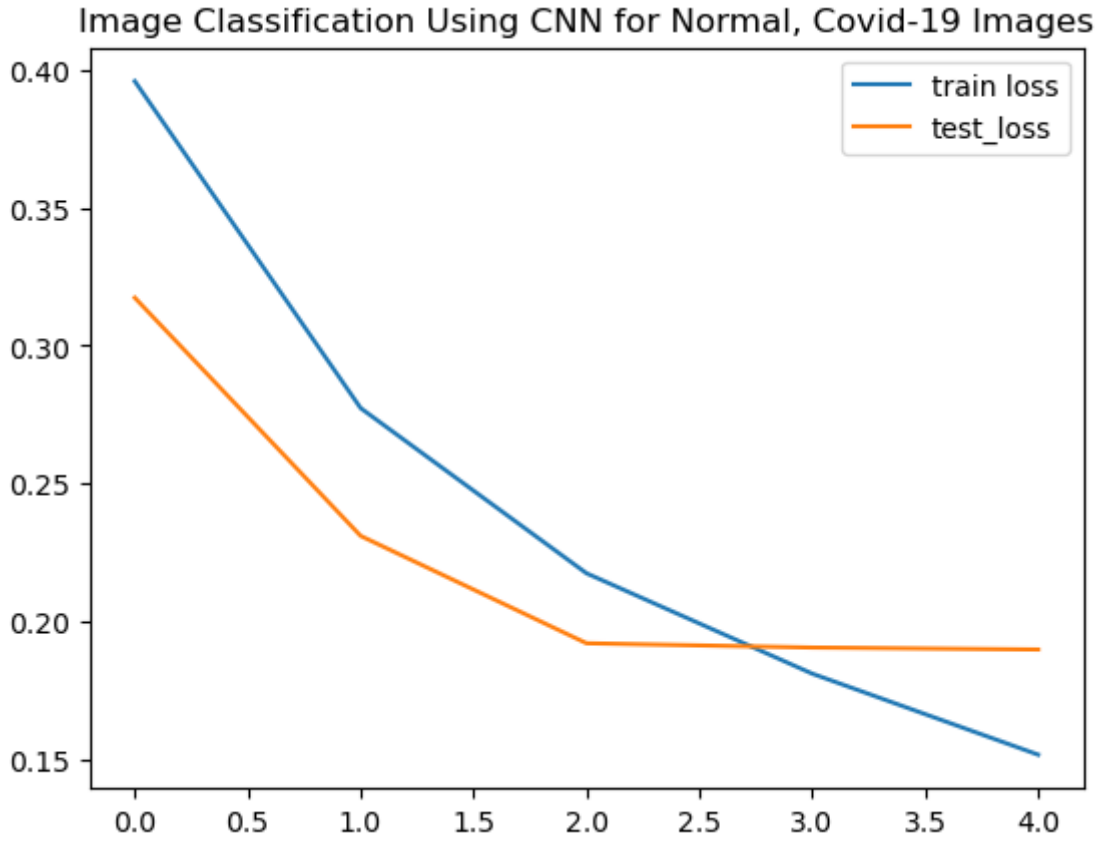
I have also tried to visualize the accuracy for training data.

In [31]:
```
plt.plot(model.history.history['accuracy'], label = 'Training Accuracy'
plt.plot(model.history.history['val_accuracy'],label = 'Testing accurac
plt.title("Image Classification Using CNN for Normal, Covid-19 Images")
plt.legend()
plt.show()
```



I have also plotted for the test data and hence we can see the loss.

In [32]:
```
plt.plot(model.history.history['loss'], label = 'train loss')
plt.plot(model.history.history['val_loss'],label = 'test_loss')
plt.title("Image Classification Using CNN for Normal, Covid-19 Images")
plt.legend()
plt.show()
```



In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Iteration 2 - Normal, Covid-19 and Viral Pneumonia Image Classification Using CNN

The same steps have been repeated as done above, but this time, I have also considered the Pneumonia images.

The following three command gives count of images in Normal, COVID and Pnemonia folder of the dataset

In [33]:
```
1 len(os.listdir('COVID-19_Radiography_Dataset/Normal/images'))
```

Out[33]: 10192

In [34]:
```
1 len(os.listdir('COVID-19_Radiography_Dataset/COVID/images'))
```

Out[34]: 3616

In [35]:
```
1 len(os.listdir('COVID-19_Radiography_Dataset/Viral Pneumonia/images'))
```

Out[35]: 1345

Writing the the loadImages function

Now, to prepare the dataset for training, it shoould contain all normal and covid images. For this we need to load all the images. I have loaded each category of images separately as well as have stored the labels / targets values separately.

images[ ] is a list to all store the images and labels[ ] stores the labels for those images - whether it is 0 or 1 where:
0 -> Normal
1 -> Covid

In [36]:
```
1  def loadImages(path, links, target):
2      images = []
3      labels = []
4      for i in range(len(links)):
5          image_path = path + links[i]
6          img = cv2.imread(image_path)
7          img = img / 255.0
8          img = cv2.resize(img, (100,100)) #if images were of different s
9          images.append(img)
10         labels.append(target)
11     images = np.asarray(images)
12     return images, labels
```

In [37]:
```
1 normal_path = "COVID-19_Radiography_Dataset/Normal/images/"
2 normal_links = os.listdir(normal_path)
3 normal_images, normal_targets = loadImages(normal_path, normal_links, 0
```

In [38]:
```
1 covid_path = "COVID-19_Radiography_Dataset/COVID/images/"
2 covid_links = os.listdir('COVID-19_Radiography_Dataset/COVID/images')
3 covid_images, covid_targets = loadImages(covid_path, covid_links, 1)
```

In [39]:
```
1 pneumonia_path = "COVID-19_Radiography_Dataset/Viral Pneumonia/images/"
2 pneumonia_links = os.listdir('COVID-19_Radiography_Dataset/Viral Pneumo
3 pneumonia_images, pneumonia_targets = loadImages(pneumonia_path, pneumo
```

In [40]:
```
1 covid_images.shape
```

Out[40]: (3616, 100, 100, 3)

In [41]:
```
1 normal_images.shape
```

Out[41]: (10192, 100, 100, 3)

In [42]:
```
1 pneumonia_images.shape
```

Out[42]: (1345, 100, 100, 3)

In [43]:
```
1 data = np.r_[covid_images, normal_images, pneumonia_images]
```

In [44]:
```
1 data.shape
```

Out[44]: (15153, 100, 100, 3)

In [45]:
```
1 targets = np.r_[covid_targets, normal_targets,pneumonia_targets]
```

In [46]:
```
1 targets.shape
```

Out[46]: (15153,)

In [47]:
```
1 x_train, x_test, y_train, y_test = train_test_split(data, targets, test
```

In [48]:
```
1  model = Sequential([
2      Conv2D(32, 3, input_shape=(100,100,3), activation='relu'),
3      MaxPooling2D(),
4      Conv2D(16, 3, activation='relu'),
5      MaxPooling2D(),
6      Conv2D(16, 3, activation='relu'),
7      MaxPooling2D(),
8      Flatten(),
9      Dense(512, activation='relu'),
10     Dense(256, activation='relu'),
11     Dense(1, activation='sigmoid')
12 ])
```

In [49]:
```
1 model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 98, 98, 32)        896

 max_pooling2d_3 (MaxPooling  (None, 49, 49, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 47, 47, 16)        4624

 max_pooling2d_4 (MaxPooling  (None, 23, 23, 16)       0
 2D)

 conv2d_5 (Conv2D)           (None, 21, 21, 16)        2320

 max_pooling2d_5 (MaxPooling  (None, 10, 10, 16)       0
 2D)

 flatten_1 (Flatten)         (None, 1600)              0

 dense_3 (Dense)             (None, 512)               819712

 dense_4 (Dense)             (None, 256)               131328

 dense_5 (Dense)             (None, 1)                 257

=================================================================
Total params: 959,137
Trainable params: 959,137
Non-trainable params: 0
_________________________________________________________________
```

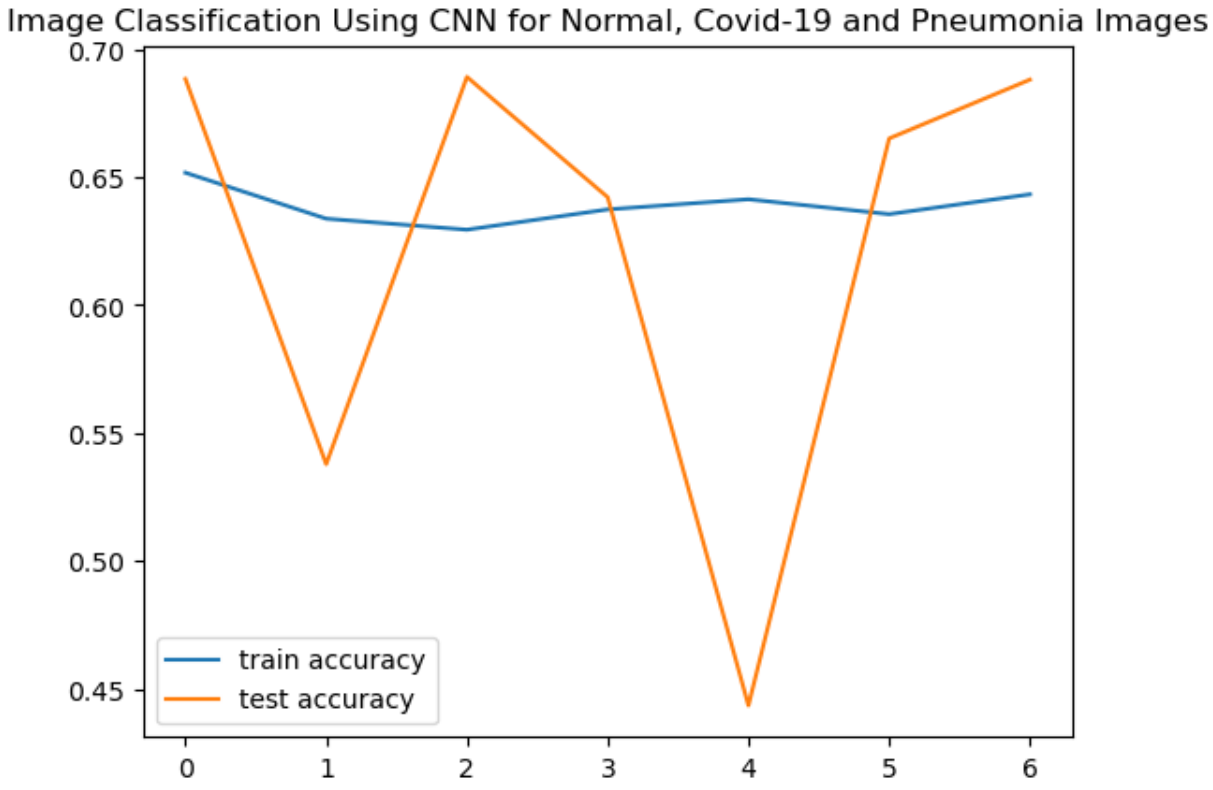
In [50]:
```
1 model.compile(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy
```

In [51]:
```
model.fit(x_train, y_train,batch_size=32,epochs=7,validation_data=(x_te
```

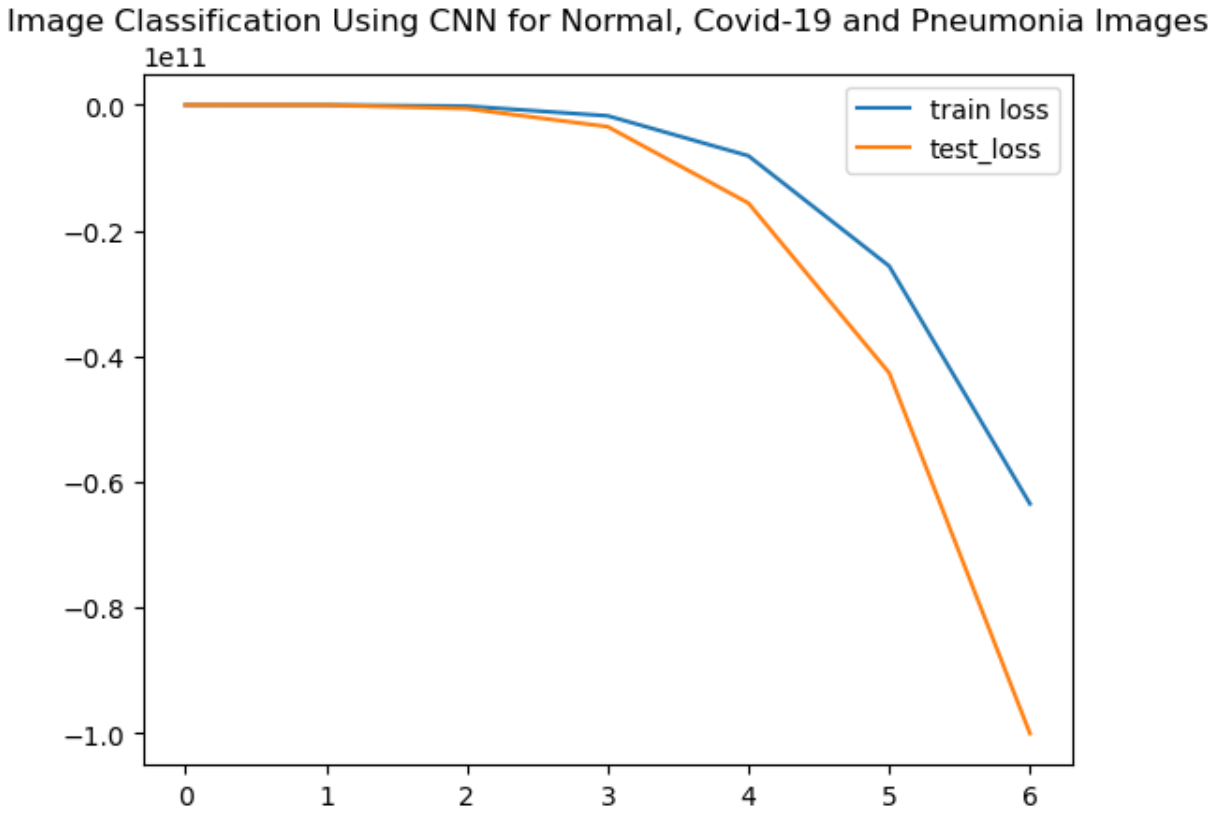
```
Epoch 1/7
356/356 [==============================] - 12s 32ms/step - loss: -1056.70
03 - accuracy: 0.6517 - mse: 0.3370 - get_f1: 0.6159 - recall_m: 0.6767 -
precision_m: 1404494.3750 - val_loss: -16658.3535 - val_accuracy: 0.6883
- val_mse: 0.3149 - val_get_f1: 0.6196 - val_recall_m: 0.5933 - val_preci
sion_m: 0.6733
Epoch 2/7
356/356 [==============================] - 11s 32ms/step - loss: -299320
0.0000 - accuracy: 0.6338 - mse: 0.3715 - get_f1: 0.6053 - recall_m: 0.68
75 - precision_m: 0.5914 - val_loss: -16070190.0000 - val_accuracy: 0.537
9 - val_mse: 0.4621 - val_get_f1: 0.5962 - val_recall_m: 0.8815 - val_pre
cision_m: 0.4601
Epoch 3/7
356/356 [==============================] - 11s 30ms/step - loss: -1821188
64.0000 - accuracy: 0.6294 - mse: 0.3758 - get_f1: 0.5989 - recall_m: 0.6
857 - precision_m: 0.5805 - val_loss: -582554624.0000 - val_accuracy: 0.6
891 - val_mse: 0.3133 - val_get_f1: 0.6636 - val_recall_m: 0.7086 - val_p
recision_m: 0.6384
Epoch 4/7
356/356 [==============================] - 10s 29ms/step - loss: -1727202
560.0000 - accuracy: 0.6374 - mse: 0.3666 - get_f1: 0.6088 - recall_m: 0.
6947 - precision_m: 0.5851 - val_loss: -3453549568.0000 - val_accuracy:
0.6421 - val_mse: 0.3595 - val_get_f1: 0.5842 - val_recall_m: 0.6116 - va
l_precision_m: 0.5799
Epoch 5/7
356/356 [==============================] - 10s 29ms/step - loss: -8097061
888.0000 - accuracy: 0.6413 - mse: 0.3645 - get_f1: 0.6150 - recall_m: 0.
6987 - precision_m: 0.5843 - val_loss: -15620661248.0000 - val_accuracy:
0.4437 - val_mse: 0.5563 - val_get_f1: 0.5624 - val_recall_m: 0.9538 - va
l_precision_m: 0.4056
Epoch 6/7
356/356 [==============================] - 11s 30ms/step - loss: -2564721
8688.0000 - accuracy: 0.6355 - mse: 0.3703 - get_f1: 0.6110 - recall_m:
0.7018 - precision_m: 0.5822 - val_loss: -42654388224.0000 - val_accurac
y: 0.6651 - val_mse: 0.3389 - val_get_f1: 0.6156 - val_recall_m: 0.6419 -
val_precision_m: 0.6117
Epoch 7/7
356/356 [==============================] - 11s 30ms/step - loss: -6350192
2304.0000 - accuracy: 0.6433 - mse: 0.3625 - get_f1: 0.6165 - recall_m:
0.7012 - precision_m: 0.5793 - val_loss: -100073406464.0000 - val_accurac
y: 0.6880 - val_mse: 0.3222 - val_get_f1: 0.6155 - val_recall_m: 0.5825 -
val_precision_m: 0.6766
```

Out[51]: <keras.callbacks.History at 0x291ad7880>

In [52]:
```
plt.plot(model.history.history['accuracy'], label = 'train accuracy')
plt.plot(model.history.history['val_accuracy'],label = 'test accuracy')
plt.title("Image Classification Using CNN for Normal, Covid-19 and Pneu
plt.legend()
plt.show()
```



In [53]:
```
plt.plot(model.history.history['loss'], label = 'train loss')
plt.plot(model.history.history['val_loss'],label = 'test_loss')
plt.title("Image Classification Using CNN for Normal, Covid-19 and Pneu
plt.legend()
plt.show()
```



In [ ]:
```

```