Final Project on

# AI Based Pneumonia Detection Using Chest X-Ray Images

**NOV 30, 2022**

**The University of Alabama in Huntsville**
**Name : Paridhi Parajuli**
**A# : 25324108**
**Course : CS588**

## Table of Contents

## 1. Introduction

With the increase in COVID19 pandemic, the cases of Pneumonia is also increasing. Pneumonia is an infection in lungs that can lead to severe respiratory conditions and even death. About 15% of under 5 year old children's death is due to Pneumonia [1]. It can be detected by taking an X-ray of the chest. In this project, an AI based Pneumonia Detection from X Ray images is built that can be useful to replace machines with doctors, thus saving time and money. This AI model may not replace the need for medical professionals but it can definitely support them for decision making [2]. The goal of this project are:

- To detect pneumonia using 3 classifiers and attain maximum possible values of performance metrics.
- To reduce the dimension of the dataset such that maximum variance from data is captured and the computation complexity also decreases.

Undoubtedly, it is a big data problem because of the volume, kind and nature of data it requires and the kind of value it offers. This projects fits the 5Vs of big data in the following ways:

- **Volume**: The dataset is around 1.24GB in size with around 6k images of normal and pneumatic cases. As the data is an image, it is obvious it will occupy volume as compared to tabular data.
- **Variety**: It represents the complexity of the data. X-ray images are unstructured data and their complexity increases with the dimension and number of channels of images. The images are stored in different formats and might need different kinds of pre-processing.
- **Velocity**: After this project is deployed, it will be facing a continuous input stream of images that need to be detected in real time or near to real time. As the complexity of images increases and the number of clients/hospitals increases, the incoming velocity will be very high. This project should be made resilient enough to handle such speed of data.
- **Veracity**: It represents the trustworthiness of the data. The dataset is collected from Kaggle. It could have been labeled by a machine or a person and there could have been some errors labeling it or even in the machine that takes x-ray.
- **Value**: This project allows detection of pneumonia from chest x-ray images without involving an expert/doctor. This can significantly reduce the workload on doctors, save time and money as well.

## 2. Methodology

This project is built following the big data life cycle methodologies.

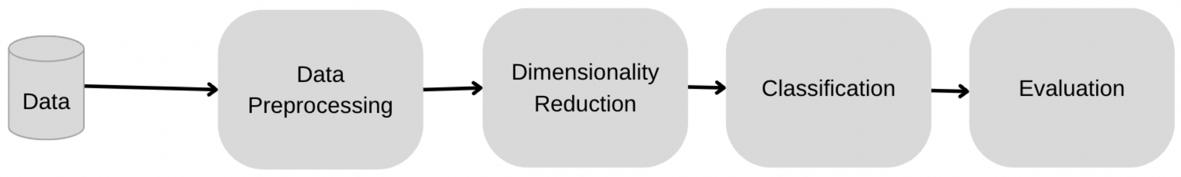


Figure 1: Methodology of Project

### 2.1 Data Identification and Extraction

The dataset that we will be using is from Kaggle [3]. This dataset has two folders : one with positive images and another with negative images. The images had varying dimensions and extensions and channels. I used the “glob” library to parse the positive and negative image file names along with its label.
The sample data looked like:

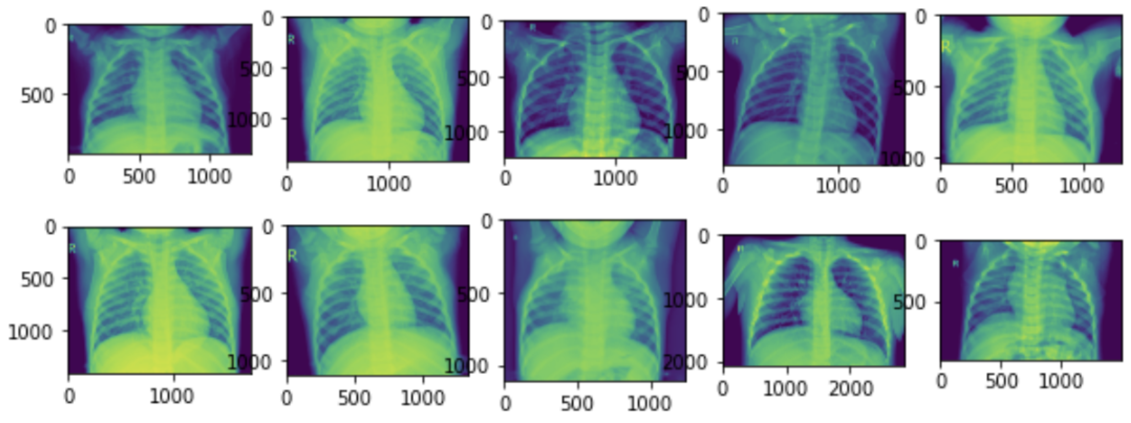


Figure 2: Raw Sample Data

And the frequency distribution of the classes is:

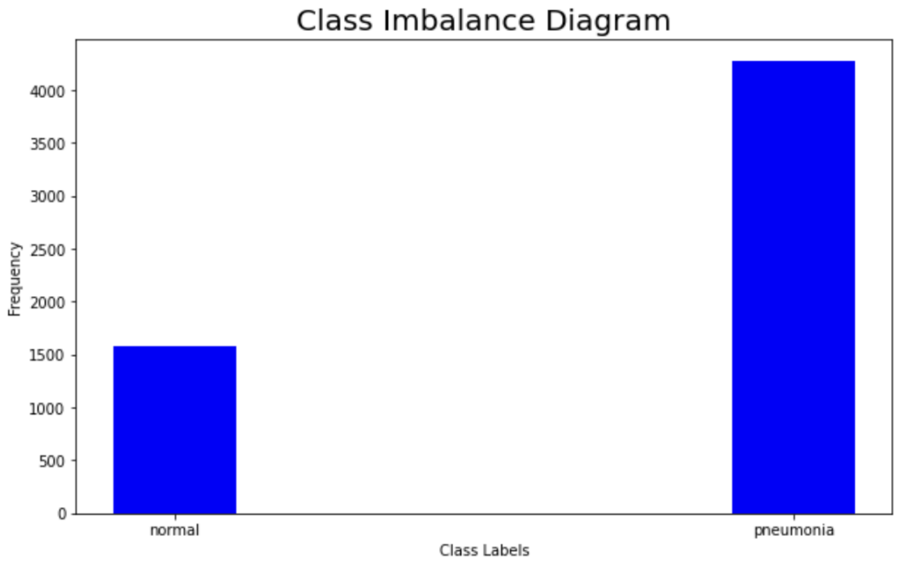


Figure 3: Bar Diagram of Class Frequencies

We can see that the data is imbalanced. This is not desirable for classification problems but still we will proceed with this dataset and see what performance metrics we will get.

## 2.2 Data Pre Processing

This process is extremely important to make the data ready to be ingested to modeling/classification algorithms. The data we have is raw, all the images do not have the same dimension and they can be very difficult to be handled by the algorithms that we are going to use. So the following pre processing steps have been used as a part of data preparation:

- Image Resize :Resizing of images has been done to make all the images consistent in terms of their dimensions. The PIL library of python has been used to do this.
- Grayscale conversion: The images have RGB channels. They were converted to grayscale using the PIL library for the simplicity of computation.
- Conversion to 1d array: Using the flatten method of numpy, the multi dimensional array of images was converted to one dimensional array [4]. Also, the individual pixel values were scaled between 0-1.
- Standardization: Before applying PCA, it was necessary to standardize the data so that all the images were translated around the mean of data using the user defined standardize() function. After this the data has a mean of 0 and a standard deviation of 1. This makes it easier to compare covariances between features [5].

After these preprocessing steps the sample data looked like:

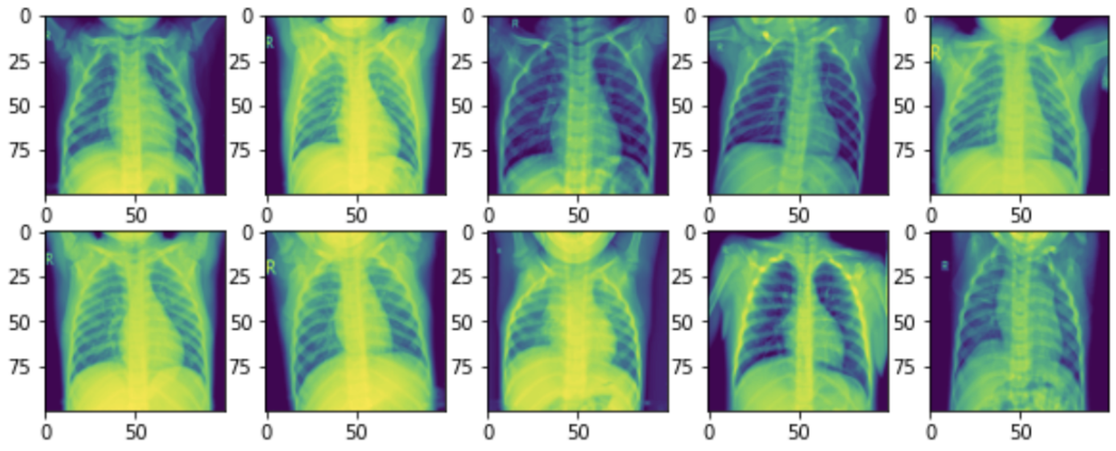


Figure 4: Sample Data after Preprocessing

## 2.3 Dimensionality Reduction

For reducing the dimension, Principal Component Analysis algorithm is used. PCA is a dimension reduction technique that works by finding the direction of maximum variance of data and projecting the data onto that direction. In order to apply PCA, we should first standardize the data by translating it around the mean of data. The maximum principal components could be equal to the number of features. But we take only the first n PCs that can sum up to our desired value of explained variance. Before PCA, the dimension of data was 5856*10000. 5856 was reduced to 323 components using PCA with preserving 95% of the information from the data. The plot of the explained variance of first 4 PCs is:

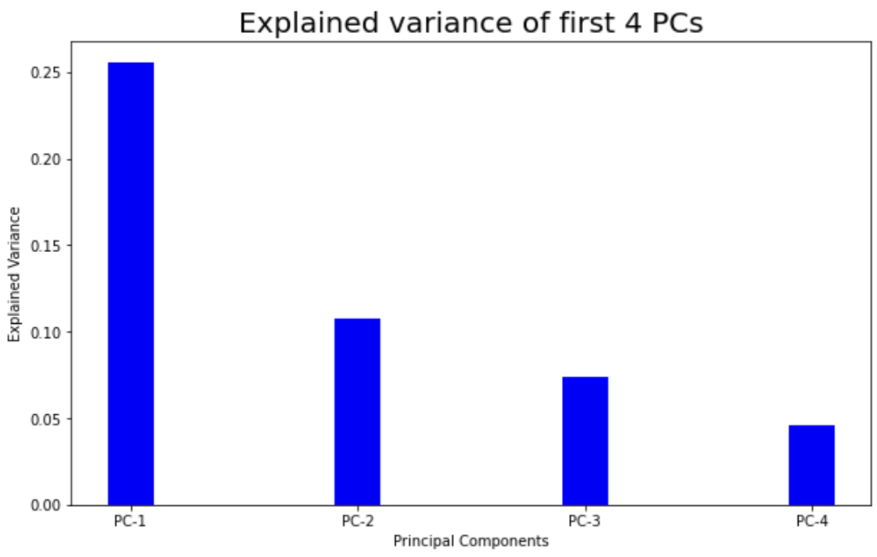


Figure 5: Bar Diagram of Explained Variance of first 4 PCs

The first PC accounts for around 25% of variance in data, the second one accounts for almost 10% and so on. After applying PCA, the dataset looks something like this:

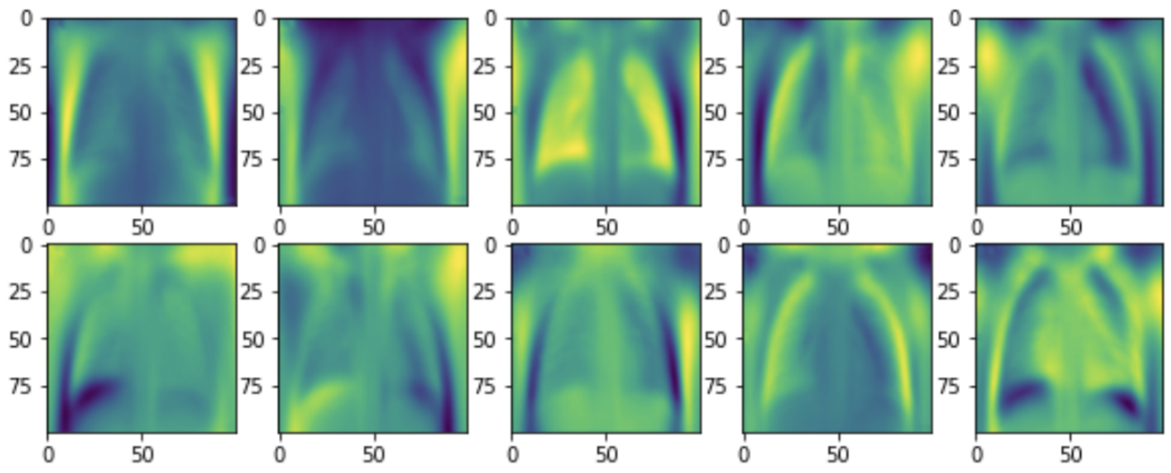


Figure 6: Sample Data after Dimension Reduction

## 2.4 Classification

Multiple classification algorithms are used to fit the data. The PCA reduced data is separated into training and testing sets using the train_test_split library from sklearn with test size of 30%. Stratification of the train and test sets is done so that equal ratio of the classes go to both of the training and test sets.

### 2.4.1 Naive Bayes

It is a classification algorithm based on the bayes theorem. It assumes that all the independent variables are independent of each other. A gaussian naive bayes classifier is used here as the likelihood of the input features is considered to follow a gaussian distribution. This algorithm is computationally fast and works well for small datasets.

### 2.4.2 Support Vector Machines (SVM)

It is a classification algorithm that works by maximizing the minimum distance between the data points and the decision boundary i.e the margin. It is a quadratic problem with linear constraints that can be solved using quadratic programming. This algorithm works well for non linear data as well. A SVM classifier with “poly” kernel and C=0.5 has been used.

### 2.4.3 Multi-layer Perceptron (MLP)

Neural networks consist of an input layer, output layer and one or many hidden layers. Each layer identifies some pattern related to the data. Input layers take the input as a flattened one dimensional array and the hidden layer finds some pattern and adjusts its weight according to the optimization function. To train the model, learning rate of 0.01 is used, sigmoid activation function is used at the output layer and binary cross entropy loss is used with adam optimizer.
The summary of model we have used is:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 fc1 (Dense)                 (None, 10)                3240

 fc2 (Dense)                 (None, 10)                110

 output (Dense)              (None, 2)                 22

=================================================================
```

Table 1: MLE model summary

The test and train accuracy of the model during the training phase is shown by the figure below and it looks like the data is being fitted properly without overfitting or underfitting [6].

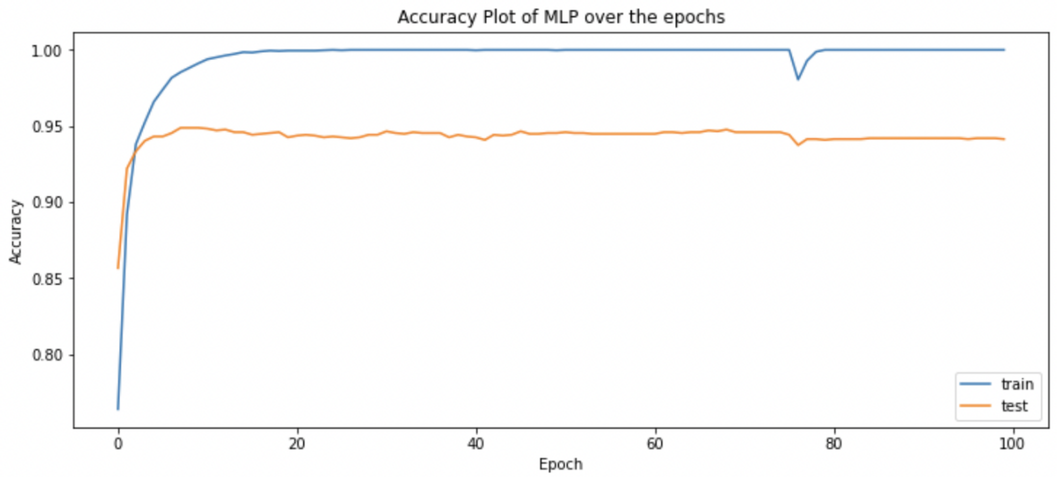


Figure 7: Training and test accuracy over the epochs of MLE

### 2.5 Predicting on New Cases

After fitting the models and choosing the best among all the classifiers, the best one will be deployed to predict new cases. The final model is saved as a h5 file which has all the weights and other parameters needed for predicting. In this phase, no training is done, only prediction is performed. A random pneumatic chest x-ray image is downloaded from the web and subjected to prediction and the model output looks something like this:

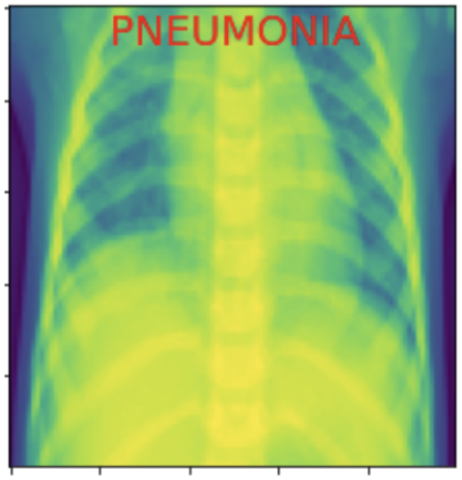


Figure 8: Prediction of test case

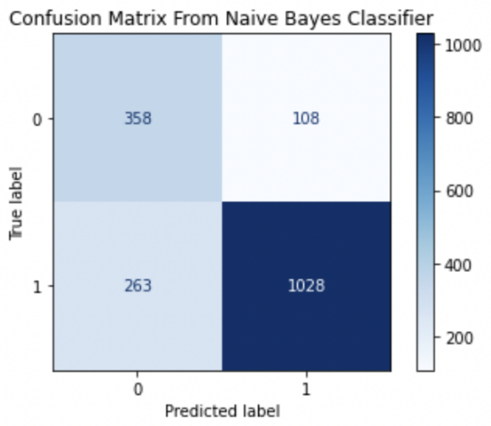
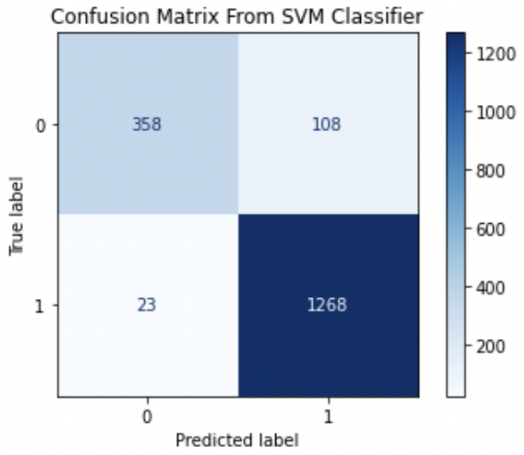
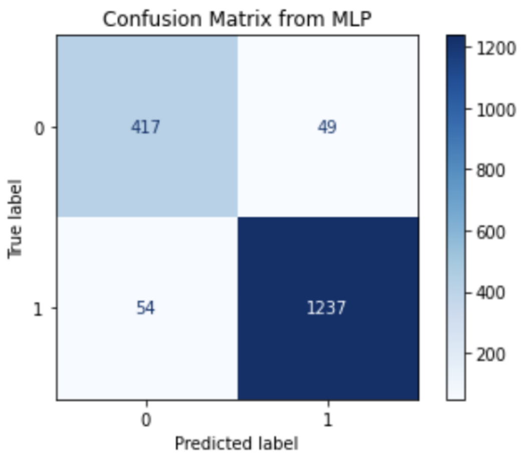
## 3. Results

### 3.1 Performance Metrics Evaluation

| **Classifier** | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **Train Accuracy** | **Time** |
|---|---|---|---|---|---|---|
| **Naive Bayes** | 78.88% | 74.07% | 78.22% | 75.29% | 77.62% | 27.6 ms |
| **SVM** | 92.54% | 93.05% | 87.52% | 89.81% | 94.77% | 1.76 s |
| **MLP** | 94.02% | 96.76% | 95.04% | 95.89% | 98.95% | 41.5 s |

Table 2: Comparison of performance metrics of different classifiers

**3.3 Confusion Matrix**

| **Classifier** | **Confusion Matrix** |
|---|---|
| **Naive Bayes** |  Figure 9: Heat map of Naive Bayes confusion matrix |
| **SVM** |  Figure 10: Heat map of SVM confusion matrix |
| **MLP** |  Figure 11: Heat map of MLP confusion matrix |

## 3.2 Discussion & Analysis

We can see that the Naive Bayes classifier did not perform well because this is quite a big dataset with many numbers of features and naive bayes doesn't not work well with a large number of features. The SVM model did fairly well because it can also work well with non linear data. During model evaluation it is equally important to look at the precision and recall because accuracy alone cannot give the best estimation of performance especially when the classes are imbalanced. If we look at precision and recall then again, MLP is the best among all. Since pneumonia is a critical health issue, false negatives are not acceptable. In that case our classifier would be saying "No" to an actual pneumonia patient. This error becomes critical so we want to have a model that can give us the maximum recall/sensitivity. In the confusion matrix heatmaps above, the more the value in the diagonals the better model it will be. Diagonal values correspond to the correct classifications whereas non diagonal values are the misclassifications. The naive bayes model has the maximum sum of non diagonal elements where the MLP classifier has the least value.

SVM model is also fairly good because it gives decent accuracy and other metrics and has relatively less computation time than MLP. When the size of the training dataset increases, the computation time for MLP will also increase. But for now around 42 seconds of training time is very practical for those values of performance metrics. Therefore we choose the MLP model although it has a higher computation time than the others.

## 4. Conclusion

Pneumonia can be fatal if not treated in its early stage. Chest X ray images are the most popular way to diagnose the presence of Pneumonia. This project aims to facilitate doctors and other medical professionals by saving their time by automatically detecting the presence of pneumonia from x-ray images without expert supervision. After pre-processing and performing PCA dimensionality reduction in the data, it was experimented with 3 classifiers, out of which the best performing classifier was the MLP model. The MLP model detects pneumonia with an accuracy of 94.02%, precision of 96.76% and recall of 95.04%. Thus this is a prototype model that can be extended to a real world deployable model with improvements as required.

## 5. References

[1] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0256630

[2] https://www.altexsoft.com/blog/image-recognition-neural-networks-use-cases/

[3] https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

[4] http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/11ImageProc/15PCA.pdf

[5] https://godzillabutnicer.com/the-why-of-principal-component-analysis-standardization-covariance

[6] https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for -deep-lear ning-models/

**6. Appendix**

```
import glob
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import
ConfusionMatrixDisplay,confusion_matrix,precision_recall_fscore_support
files0 =glob.glob('/content/drive/MyDrive/chest_xray/*/NORMAL/*')
files1 =glob.glob('/content/drive/MyDrive/chest_xray/*/PNEUMONIA/*')
plt.figure(figsize=(10,10))
for i in range(10):
   plt.subplot(5,5,i+1)
   plt.imshow(Image.open(files0[i]))
plt.figure(figsize = (10, 6))
plt.bar(['normal','pneumonia'], [len(files0),len(files1)],width=0.2, color="blue")
plt.xlabel("Class Labels")
plt.ylabel("Frequency")
plt.title("Class Imbalance Diagram", fontsize=20)
plt.show()
DIMENSIONS = 100
def load_images(files0,files1):
   all_images=[]
   all_labels=[]
   for f in files0:
      im=Image.open(f).convert('L').resize((DIMENSIONS,DIMENSIONS))
      img=np.array(im).flatten()/255
      all_images.append(img)
      all_labels.append(0)
   for f in files1:
      im=Image.open(f).convert('L').resize((DIMENSIONS,DIMENSIONS))
      img=np.array(im).flatten()/255
      all_images.append(img)
      all_labels.append(1)
   return all_images, all_labels
all_images, all_labels= load_images(files0,files1)
def standarize(x):
   mean = np.mean(x)
   std = np.std(x)
   x = (x- mean) / std
   return mean,std,x
_mean,_std,images_st=standarize(np.array(all_images))
plt.figure(figsize=(10,10))
```

```
for i in range(10):
   plt.subplot(5,5,i+1)
   plt.imshow(images_st[i].reshape(100,100))
pca= PCA(n_components=0.95)
pca=pca.fit(images_st)
img_pca=pca.transform(images_st)
var=pca.explained_variance_ratio_
plt.figure(figsize = (10, 6))
plt.bar([f"PC-{i+1}" for i in range(4)], list(var[:4]),width=0.2, color="blue")
plt.xlabel("Principal Components")
plt.ylabel("Explained Variance")
plt.title("Explained variance of first 4 PCs", fontsize=20)
plt.show()
plt.figure(figsize=(10,10))
eigen_cell=pca.components_
eigen_cell=eigen_cell.reshape(323,100,100)
for i in range(10):
   plt.subplot(5,5,i+1)
   plt.imshow(eigen_cell[i])
X_train, X_test, y_train, y_test = train_test_split(img_pca, np.array(all_labels),test_size = 0.3,
random_state = 111)
from sklearn.naive_bayes import GaussianNB
clf=GaussianNB()
%time clf.fit(X_train, y_train)
train_acc=metrics.accuracy_score(y_train, clf.predict(X_train))
test_acc=metrics.accuracy_score(y_test, clf.predict(X_test))
m=precision_recall_fscore_support(y_test, clf.predict(X_test), average='macro')
print(f"Test Accuracy : {test_acc*100}%\nTrain Accuracy : {train_acc*100}%\nPrecision:
{m[0]*100}%\nRecall : {m[1]*100}%\nFscore: {m[2]*100}%" )
com=confusion_matrix(y_test, clf.predict(X_test))
disp = ConfusionMatrixDisplay(confusion_matrix=com)
disp.plot(cmap =plt.cm.Blues)
plt.title(f"Confusion Matrix From Naive Bayes Classifier")
plt.show()
clf = svm.SVC(kernel = "poly",C=0.5, random_state=111)
%time clf.fit(X_train, y_train)
m=precision_recall_fscore_support(y_test, clf.predict(X_test), average='macro')
train_acc=metrics.accuracy_score(y_train, clf.predict(X_train))
test_acc=metrics.accuracy_score(y_test, clf.predict(X_test))
print(f"Test Accuracy : {test_acc*100}%\nTrain Accuracy : {train_acc*100}%\nPrecision:
{m[0]*100}%\nRecall : {m[1]*100}%\nFscore: {m[2]*100}%" )
com=confusion_matrix(y_test, clf.predict(X_test))
disp = ConfusionMatrixDisplay(confusion_matrix=com)
disp.plot(cmap =plt.cm.Blues)
plt.title(f"Confusion Matrix From SVM Classifier")
plt.show()
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from sklearn.preprocessing import OneHotEncoder
y_ = np.array(all_labels).reshape(-1, 1)
encoder = OneHotEncoder(sparse=False)
y__ = encoder.fit_transform(y_)
model = Sequential()
model.add(Dense(10, input_shape=(img_pca.shape[1],), activation='relu', name='fc1')) #4
model.add(Dense(10, activation='relu', name='fc2'))
model.add(Dense((len(np.unique(y_))), activation='sigmoid', name='output'))
optimizer = Adam(lr=0.001)
print(model.summary())
train_x, test_x, train_y, test_y = train_test_split(img_pca, y__, test_size=0.3,random_state = 111)
model.compile(optimizer, loss='binary_crossentropy', metrics=['accuracy'])
%time history = model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=100,
verbose=0)
plt.figure(figsize=(12,5))
plt.title('Accuracy Plot of MLP over the epochs')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
r=model.predict(test_x)
pred=[]
for i in range(len(r)):
  pred.append(np.argmax(r[i]))
pred = np.array(pred)
y_or = np.argmax(test_y, axis=1)
accuracy = accuracy_score(y_or, pred)
precision = precision_score(y_or, pred)
recall = recall_score(y_or, pred)
f1 = f1_score(y_or, pred)
print(f"Accuracy : {accuracy*100}%\nPrecision: {precision*100}%\nRecall :
{recall*100}%\nFscore: {f1*100}%" )
_, train_acc = model.evaluate(train_x, train_y, verbose=0)
print("Training accuracy:",str(train_acc*100) +"%")
com=confusion_matrix(y_or, pred)
disp = ConfusionMatrixDisplay(confusion_matrix=com)
disp.plot(cmap =plt.cm.Blues)
plt.title(f"Confusion Matrix from MLP")
plt.show()
model.save("/content/drive/MyDrive/model.h5")
from keras.models import load_model
model1 = load_model('/content/drive/MyDrive/model.h5')
```

```
im=Image.open('/content/drive/MyDrive/test.jpeg').convert('L').resize((DIMENSIONS,DIMENS
IONS))
img=np.array(im).flatten()/255
img_test = (img- _mean) / _std
img_test=img_test.reshape(1,-1)
pca_test = pca.transform(img_test)
r=model1.predict(pca_test,verbose=0)
if np.argmax(r[0])==1 :
  output="PNEUMONIA"
else:
  output="NORMAL"
plt.imshow(im)
plt.text(50, 5, output, horizontalalignment='center',
     verticalalignment='center',fontsize=20, color='r')
```