TTL PROJECT

TOPIC- Pneumonia Detection Using Deep Learning

SUBMITTED BY:
1. SARANDEEP SABUT (20051397)

SUBMITTED TO:
Prof. Mainak Bandyopadhyay

Introduction:

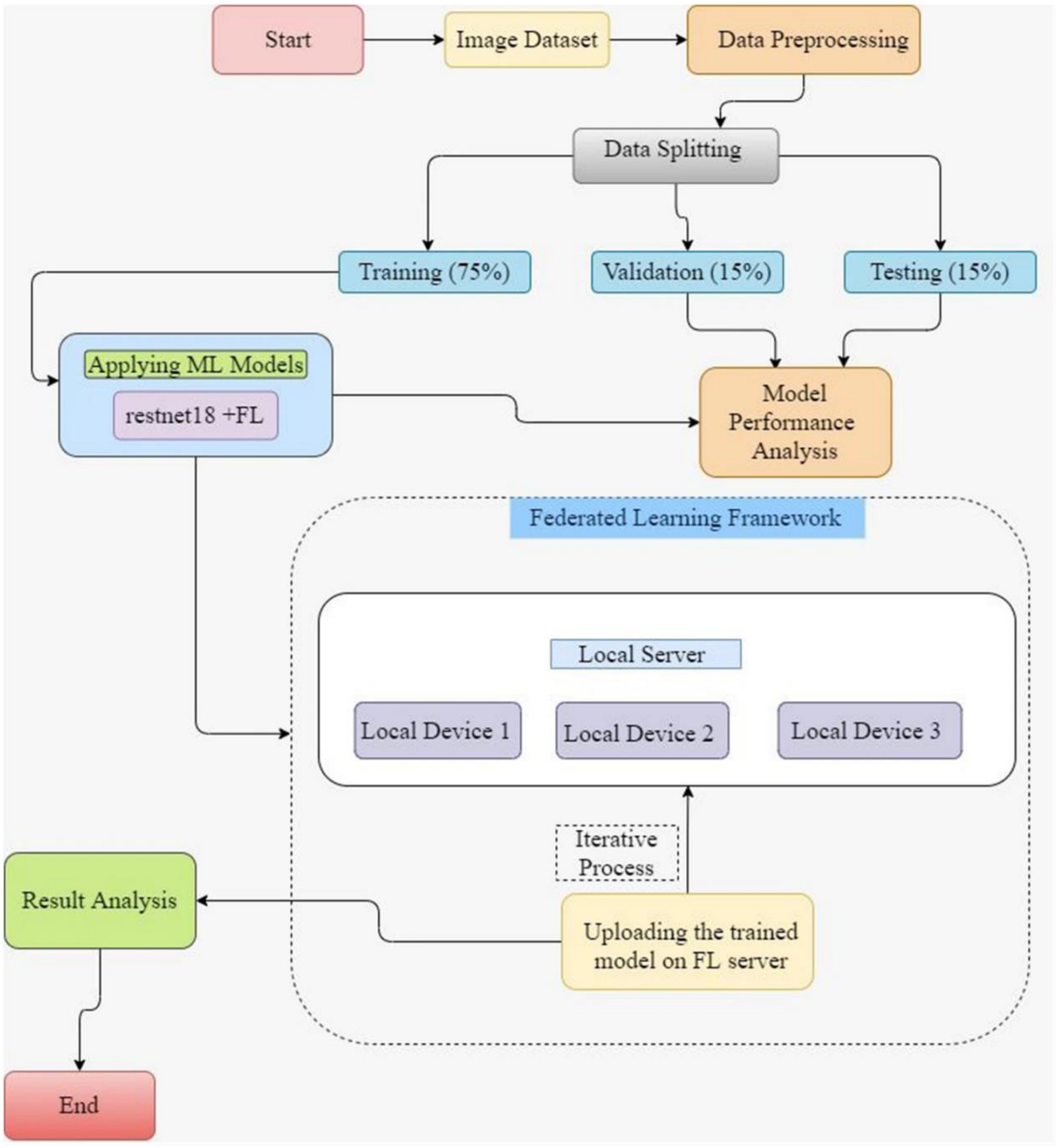
Chest X-Rays which are used to diagnose pneumonia need expert radiotherapists for evaluation. Thus, developing an automatic system for detecting pneumonia would be beneficial for treating the disease without any delay particularly in remote areas.
The experiment was performed with the CheXNet algorithm with 121 layers of CNN and chest X-ray images as inputs in diagnosing and detecting the presence of pneumonia infection.
The main advantage of CNN compared to its predecessors is that it is capable of detecting the relevant features without any human supervision. A series of convolution and pooling operations is performed on the input image, which is followed by a single or multiple fully connected layers.

Deep Learning Methods:

Medical image detection is a complicated task; therefore, an effective approach is needed. Deep learning is one of the techniques that can be used for the training of medical image datasets. In the study, deep learning model of RestNet-101 and RestNet50 was used for pneumonia detection. While considering these techniques, it has resulted in different results based on individual features. Therefore, to compensate this difference, an effective deep learner strategy was introduced that involves the combination of these techniques. In this study, dataset of 14,863 X-ray images was used and the achieved precision is 96%. Although the model output good precision, however it possesses limitations due to the complexity of combining the RestNet models that can effect the precision when larger dataset is considered in a real time scenario. The experiment was performed to demonstrate how deep learning models can diagnose diseases. In this case, the deep neural network was used to aid in diagnosing 14 diseases. The ChestXray14 database was used and trained with DenseNet and reduced pairwise error to relate their outcomes in diagnosing diseases. The architecture was developed to help in detecting and classifying diseases using multilabels. In addition, the cascade network aided in making all possible predictions by comparing several previous levels, which are used as inputs in each successive level in the Cascade network. The level-6 cascading network was used in both PWE loss and cross-entropy. The study results indicated that the Cascade network helped in increasing the performance classifiers. The use of DenseNets has produced positive outcomes that include reducing the gradient problem, reinforcing the features propagation, and reducing the parameters. However, this model is not capable of modelling the inner class.

Purposed model:

We have analysed the various work done on medical image detection in the previous section. The experiments were performed based on available datasets. It has been observed that the machine learning models effectively detects medical images when the model is fed with a larger quantity of data. The use of ML algorithms has been proven effective in detection while compared to the traditional procedures mentioned in the literature review.
ML models need a higher volume of data for effective training capable of achieving higher accuracy in detection. The lab-based datasets are always limited for effective ML model training. In the real-time medical image data, constant change in the feature variables determines the accuracy of ML models. Therefore, we need a solution that can fulfil the datasets requirements for effective training of the model. The below image shows the flow chart of our proposed model.



The above figure shows our proposed model that follows series of steps from start to end. In the proposed model, image data is followed with data processing by splitting the data at the ratio of 75%, 15% and 15% into training, validation and testing respectively. After the model is trained by the training data, then the model is used for performance analysis by testing and validation data. In the proposed architecture, the training of the model is performed in a FL framework, where the restnet18 model is sent across local devices and model is trained on individual device data. After training it comes back to central server and the process carries on as iterative to get more updates from the local device. In this framework, data is not shared, instead only the trained model is shared to the central server (FL server), therefore the privacy of the data is promised. Eventually, a fully trained model can be effectively used for various purposes for example in detecting pneumonia.

Our proposed solution involves using privacy-preserving procedures that will allow using the real-time data, which fulfil the requirements of having massive data and variant patterns of medical images for ML model training. Privacy of the data is ensured in the proposed method that involves using the Federated Learning approach. The use of FL will involve the mutual collaboration of hospitals and medical institutes to train the ML model in their local servers, and the trained model from individual entities is shared centrally and aggregated together without sharing data. The central aggregation constitutes the trained model that repeats the cycle of training periodically, which helps to attain the higher efficiency of training the model for effective medical image detection. By using this approach, the privacy of the real-time data is ensured. Deep learning is one of the effective ML models that will be aggregated together with the FL, and it will ultimately help attain the maximum feature variables pattern to produce the effective outcome for medical image detection like pneumonia.

CODES:

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models
import Sequentialfrom tensorflow.keras.layers
import Dense, Activation, Conv2D, MaxPooling2D, Flatten, Drop ut,
BatchNormalizationfrom tensorflow.keras.optimizers
import Adamfrom tensorflow.keras.callbacks
import EarlyStoppingfrom tensorflow.keras.preprocessing.image import
ImageDataGeneratorfrom sklearn.metrics
import precision_recall_curve, roc_curve, accuracy_score, confusion_matrix,
precision_score, recall_score
from sklearn.decomposition
import PCA
from sklearn.model_selection
import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns plt.style.use('fivethirtyeight')
import pickle
import os
import numpy as np
import cv2
%matplotlib inline
```

Process the images and resize them to the preferred size

```
labels = ['PNEUMONIA', 'NORMAL']
img_size = 200
def get_training_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img),
cv2.IMREAD_GRAYSCALE)
                resized_arr = cv2.resize(img_arr, (img_size, img_size))
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

Preparing the training and testing data

```
train = get_training_data('../input/chest-xray-pneumonia/chest_xray/chest_xray/train')
test = get_training_data('../input/chest-xray-pneumonia/chest_xray/chest_xray/test')
val = get_training_data('../input/chest-xray-pneumonia/chest_xray/chest_xray/val')

    pnenumonia = 0
    normal = 0

    for i, j in train:
        if j == 0:
            pnenumonia+=1
        else:
            normal+=1

    print('Pneumonia:', pnenumonia)
    print('Normal:', normal)
    print('Pneumonia - Normal:', pnenumonia-normal)
```

Pneumonia: 3875
Normal: 1341
Pneumonia - Normal: 2534

Visualize training images

```
plt.imshow(train[1][0], cmap='gray')
plt.axis('off')
print(labels[train[1][1]])
```

We are incoprating the validation data into the training data

```
X = []
y = []

for feature, label in train:
    X.append(feature)
    y.append(label)

for feature, label in test:
    X.append(feature)
    y.append(label)

for feature, label in val:
    X.append(feature)
    y.append(label)


# resize data for deep learning
X = np.array(X).reshape(-1, img_size, img_size, 1)
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=32)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.20,
random_state=32)

X_train = X_train / 255
X_test = X_test / 255
X_val = X_val / 255
```

Data augmentation

```
# good for balancing out disproportions in the dataset
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=90,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        vertical_flip=True)

datagen.fit(X_train)
```

CNN (Convolutional Neural Network)

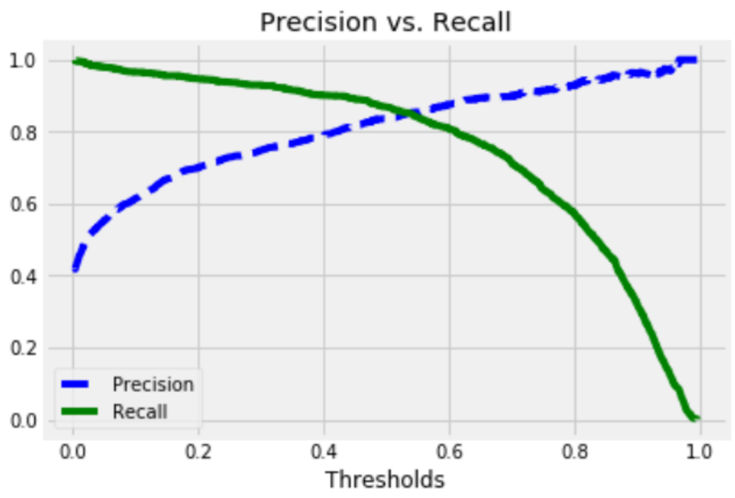
```
model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X_train.shape[1:], padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(16, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dropout(0.5))
model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

early_stop = EarlyStopping(patience=3, monitor='val_loss',
restore_best_weights=True)
adam = Adam(learning_rate=0.0001)
model.compile(loss='binary_crossentropy',optimizer=adam,metrics=['acc'])

model.summary()
history = model.fit(datagen.flow(X_train, y_train, batch_size=10),
callbacks=[early_stop], validation_data=(X_val, y_val), epochs=15)
model.evaluate(X_test, y_test)
```

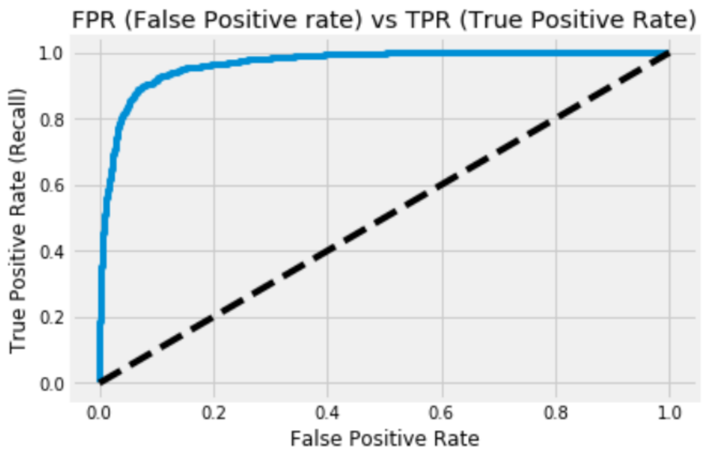
Visualizing our training progress

```
plt.figure(figsize=(16, 9))
plt.plot(history.epoch, history.history['acc'])
plt.title('Model Accuracy')
plt.legend(['train'], loc='upper left')
plt.show()

plt.figure(figsize=(16, 9))
plt.plot(history.epoch, history.history['loss'])
plt.title('Model Loss')
plt.legend(['train'], loc='upper left')
plt.show()

plt.figure(figsize=(16, 9))
plt.plot(history.epoch, history.history['val_acc'])
plt.title('Model Validation Accuracy')
plt.legend(['train'], loc='upper left')
plt.show()

plt.figure(figsize=(16, 9))
plt.plot(history.epoch, history.history['val_loss'])
plt.title('Model Validation Loss')
plt.legend(['train'], loc='upper left')
plt.show()
```

Prepare data for precision vs. recall and ROC

```
pred = model.predict(X_train)
precisions, recalls, thresholds = precision_recall_curve(y_train, pred)
fpr, tpr, thresholds2 = roc_curve(y_train, pred)

def plot_precision_recall(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], 'b--')
    plt.plot(thresholds, recalls[:-1], 'g-')
    plt.title('Precision vs. Recall')
    plt.xlabel('Thresholds')
    plt.legend(['Precision', 'Recall'], loc='best')
    plt.show()

def plot_roc(fpr, tpr):
    plt.plot(fpr, tpr)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.title('FPR (False Positive rate) vs TPR (True Positive Rate)')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate (Recall)')
    plt.show()

plot_precision_recall(precisions, recalls, thresholds)
plot_roc(fpr, tpr)
```





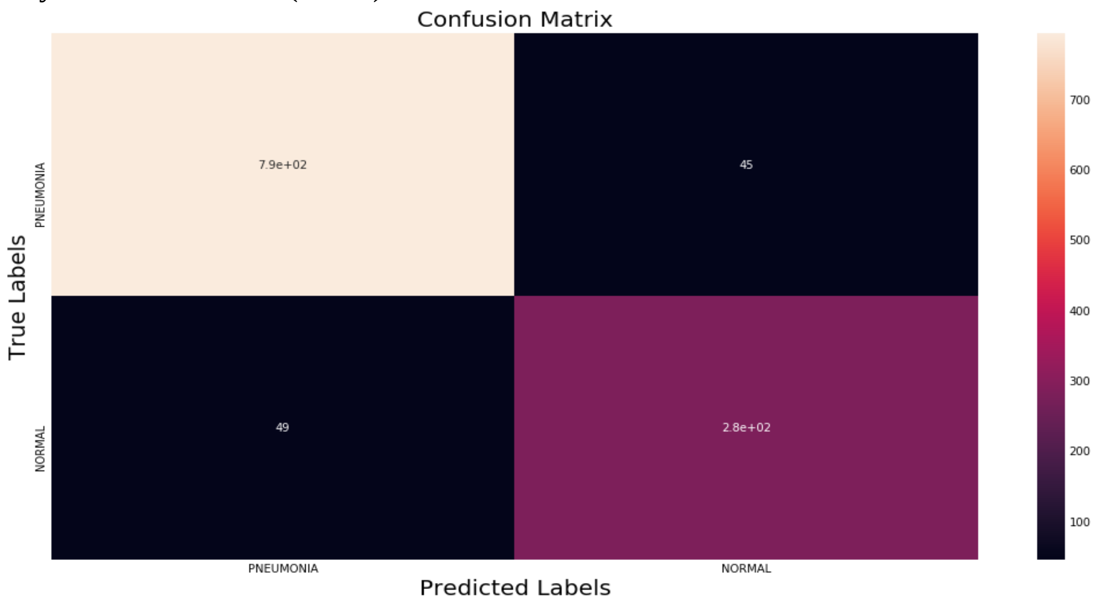
Set thresholds for our model, we want the results to be precise while not sacraficing too much recall

```
binary_predictions = []
threshold = thresholds[np.argmax(precisions >= 0.80)]
for i in predictions:
    if i >= threshold:
        binary_predictions.append(1)
    else:
        binary_predictions.append(0)

print('Accuracy on testing set:', accuracy_score(binary_predictions, y_test))
print('Precision on testing set:', precision_score(binary_predictions, y_test))
print('Recall on testing set:', recall_score(binary_predictions, y_test))
```

```
Accuracy on testing set: 0.9104095563139932
Precision on testing set: 0.9148936170212766
Recall on testing set: 0.7962962962962963
```

Plotting the confusion matrix.

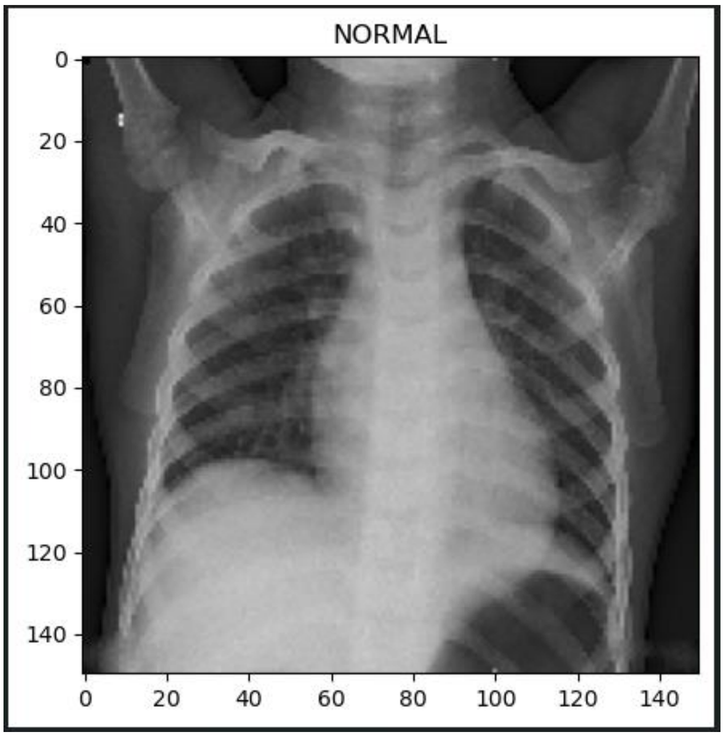
```
matrix = confusion_matrix(binary_predictions, y_test)
plt.figure(figsize=(16, 9))
ax= plt.subplot()
sns.heatmap(matrix, annot=True, ax = ax)

# labels, title and ticks
ax.set_xlabel('Predicted Labels', size=20)
ax.set_ylabel('True Labels', size=20)
ax.set_title('Confusion Matrix', size=20)
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

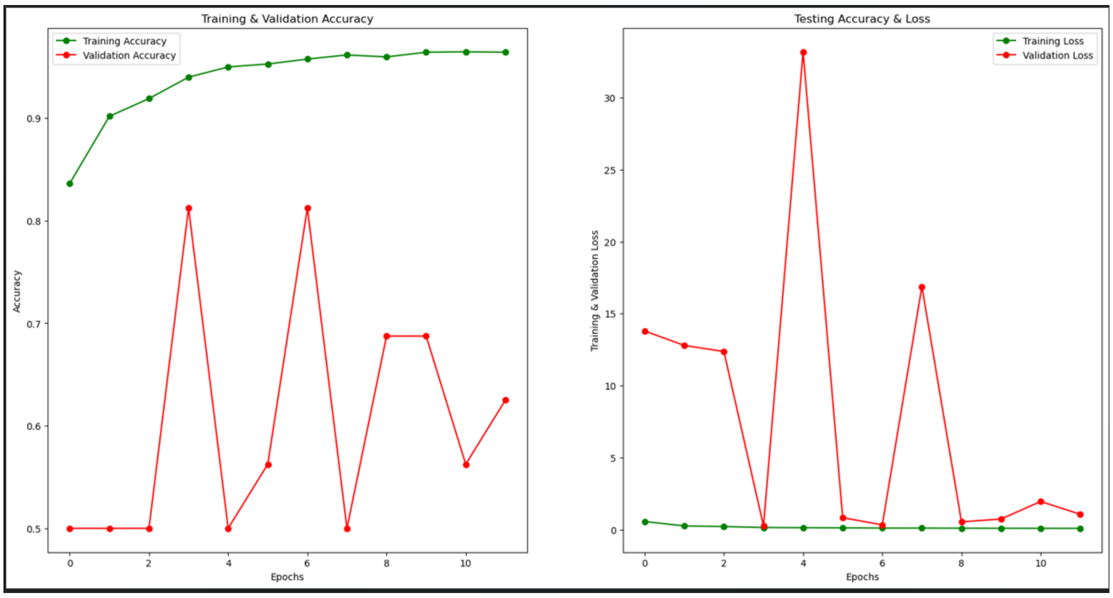


Download the model

```
model.save('pneumonia_detection_ai_version_3.h5’)
```

Output:





(accuracy of our project)

…………..XXXX………….