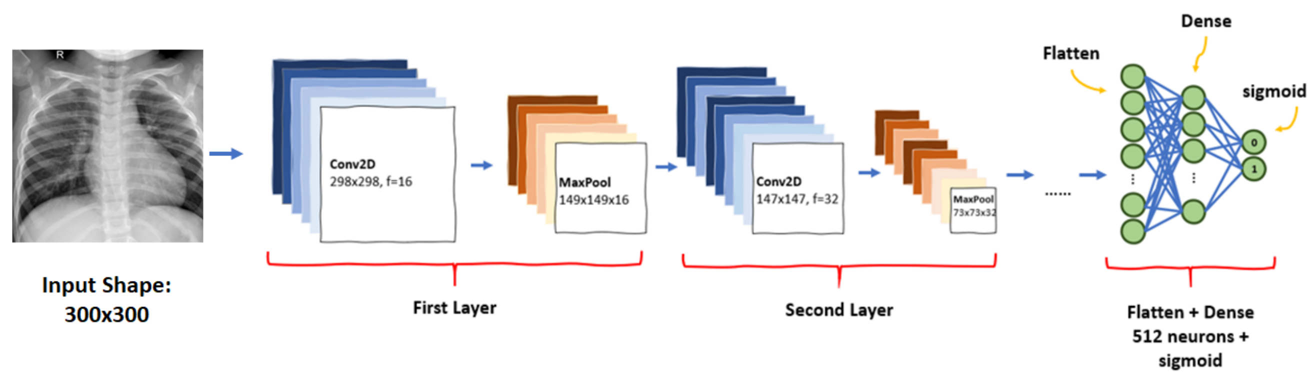
PNEUMONIA DETECTION USING CONVOLUTIONAL NEURAL NETWORKS

Pneumonia Detection using Convolutional Neural Network (CNN)



IMPORTING NECESSARY LIBRARIES AND PACKAGES

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from 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
```

```
import warnings
warnings.filterwarnings("ignore")
```

RESIZING THE IMAGES ACCORDING TO THE PREFRRED 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 DATA SETS

```
train = get_training_data("/content/drive/MyDrive/COVID19_XRAYIMAGES/xray_dataset_covid19/test")
test = get_training_data("/content/drive/MyDrive/COVID19_XRAYIMAGES/xray_dataset_covid19/train")
val = get_training_data("/content/drive/MyDrive/Pneumonia DataSet/chest_xray/val")
```

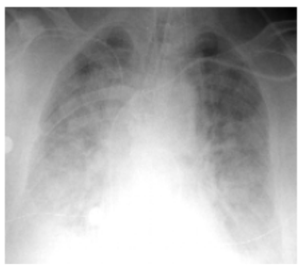
```
pnenumonia = 0
normal = 0
for i, j in train:
    if j == 0:
        pnenumonia+=1
    else:
        normal+=1

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

```
Pneumonia: 20
Normal: 20
Pneumonia - Normal: 0
```

SAMPLE IMAGE

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

PNEUMONIA



INCORPORATING VALIDATION DATA AND RESIZING THE DATA FOR DEEP LEARNING

```
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)


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
```

DATA AUGMENTATION - for balancing the disporpotion

```
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)
```

```
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()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 200, 200, 256)     2560      
                                                                 
 activation (Activation)     (None, 200, 200, 256)     0         
                                                                 
 max_pooling2d (MaxPooling2D  (None, 100, 100, 256)    0         
 )                                                               
                                                                 
 batch_normalization (BatchN  (None, 100, 100, 256)    400       
 ormalization)                                                   
                                                                 
 conv2d_1 (Conv2D)           (None, 100, 100, 64)      147520    
                                                                 
 activation_1 (Activation)   (None, 100, 100, 64)      0         
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 50, 50, 64)       0         
 2D)                                                             
                                                                 
 batch_normalization_1 (Batc  (None, 50, 50, 64)       200       
 hNormalization)                                                 
                                                                 
 conv2d_2 (Conv2D)           (None, 50, 50, 16)        9232      
                                                                 
 activation_2 (Activation)   (None, 50, 50, 16)        0         
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 25, 25, 16)       0         
 2D)                                                             
                                                                 
 batch_normalization_2 (Batc  (None, 25, 25, 16)       100       
 hNormalization)                                                 
                                                                 
 flatten (Flatten)           (None, 10000)             0         
                                                                 
 dropout (Dropout)           (None, 10000)             0         
                                                                 
 dense (Dense)               (None, 64)                640064    
                                                                 
 activation_3 (Activation)   (None, 64)                0         
                                                                 
 dropout_1 (Dropout)         (None, 64)                0         
                                                                 
 dense_1 (Dense)             (None, 1)                 65        
                                                                 
 activation_4 (Activation)   (None, 1)                 0         
                                                                 
=================================================================
Total params: 800,141
Trainable params: 799,791
Non-trainable params: 350
_________________________________________________________________
```

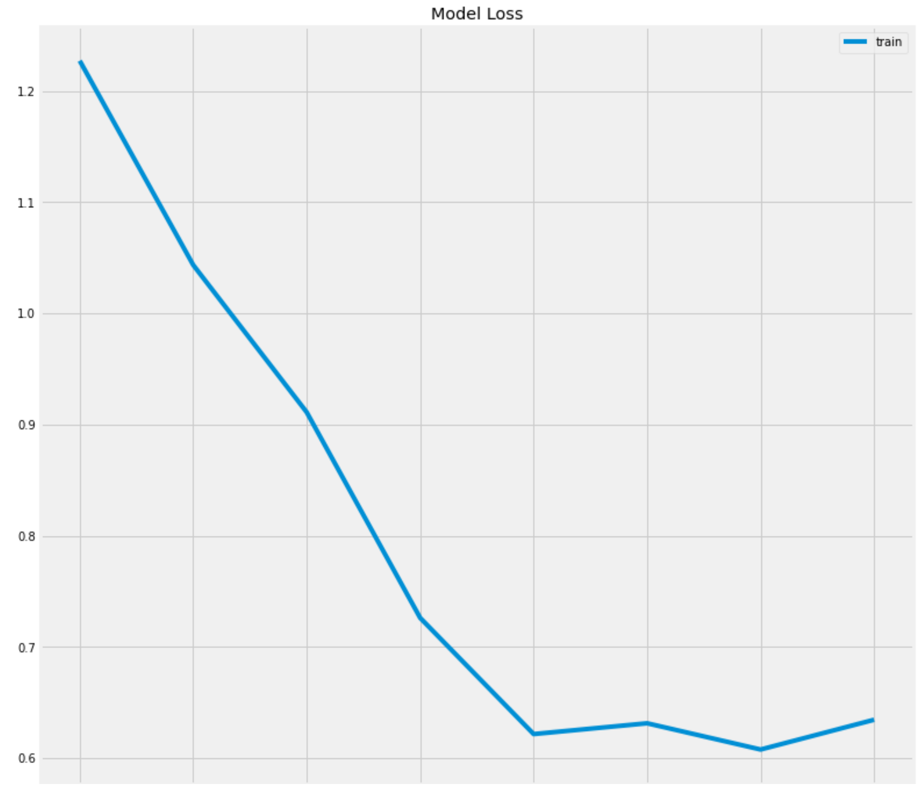
```
history = model.fit(datagen.flow(X_train, y_train, batch_size=10), callbacks=[early_stop], validation_data=(X_val, y_val), epochs=15)
```

```
Epoch 1/15
13/13 [==============================] - 92s 6s/step - loss: 1.2272 - acc: 0.5538 - val_loss: 13.6263 - val_acc: 0.3636
Epoch 2/15
13/13 [==============================] - 81s 6s/step - loss: 1.0435 - acc: 0.5308 - val_loss: 4.0438 - val_acc: 0.4545
Epoch 3/15
13/13 [==============================] - 78s 6s/step - loss: 0.9111 - acc: 0.5077 - val_loss: 3.2751 - val_acc: 0.4848
Epoch 4/15
13/13 [==============================] - 78s 6s/step - loss: 0.7262 - acc: 0.5923 - val_loss: 0.8086 - val_acc: 0.7879
Epoch 5/15
13/13 [==============================] - 78s 6s/step - loss: 0.6217 - acc: 0.7000 - val_loss: 0.7129 - val_acc: 0.8485
Epoch 6/15
13/13 [==============================] - 78s 6s/step - loss: 0.6314 - acc: 0.6538 - val_loss: 1.1506 - val_acc: 0.7879
Epoch 7/15
13/13 [==============================] - 87s 7s/step - loss: 0.6078 - acc: 0.6846 - val_loss: 3.0190 - val_acc: 0.6061
Epoch 8/15
13/13 [==============================] - 78s 6s/step - loss: 0.6346 - acc: 0.6308 - val_loss: 6.4357 - val_acc: 0.4242
```

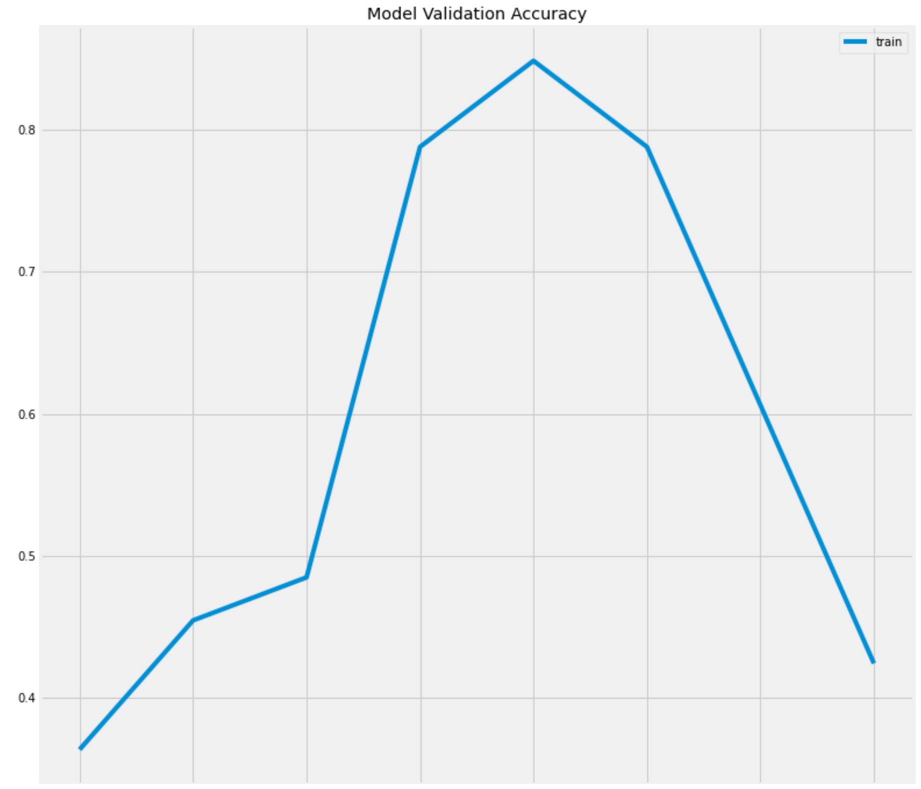
```
model.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 7s 2s/step - loss: 0.8245 - acc: 0.3171
[0.8244539499282837, 0.31707316637039185]
```

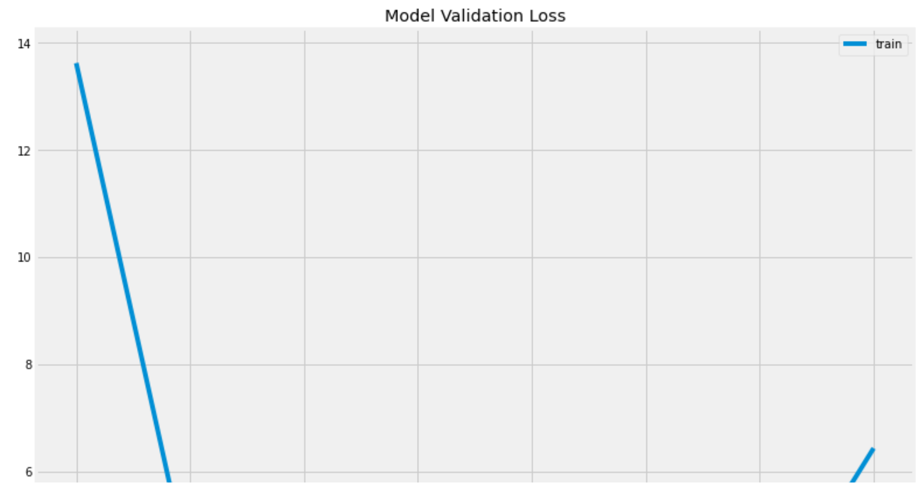
```
plt.figure(figsize=(12,12))
plt.plot(history.epoch, history.history['acc'])
plt.title('Model Accuracy')
plt.legend(['train'], loc='upper right')
plt.show()
plt.figure(figsize=(12,12))
plt.plot(history.epoch, history.history['loss'])
plt.title('Model Loss')
plt.legend(['train'], loc='upper right')
plt.show()
```



```
plt.figure(figsize=(12,12))
plt.plot(history.epoch, history.history['val_acc'])
plt.title('Model Validation Accuracy')
plt.legend(['train'], loc='upper right')
plt.show()
```



```
plt.figure(figsize=(12,12))
plt.plot(history.epoch, history.history['val_loss'])
plt.title('Model Validation Loss')
plt.legend(['train'], loc='upper right')
plt.show()
```

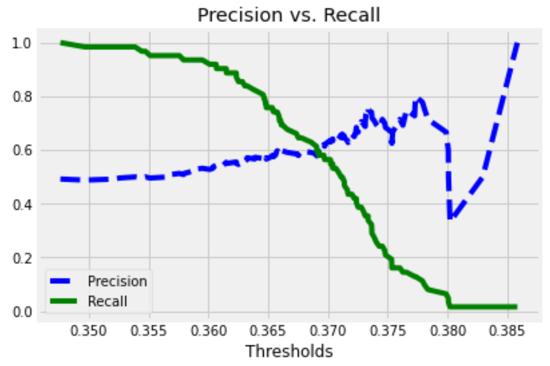


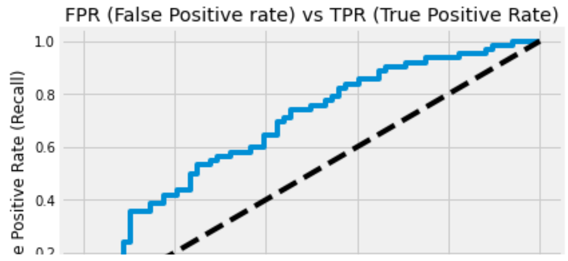
Prepare data for precision vs.recall and ROC

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

```
5/5 [==============================] - 19s 4s/step
```

```
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)
```





```
predictions = model.predict(X_test)
```

```
2/2 [==============================] - 8s 1s/step
```

SETTING THRESHOLDS

```
binary_predictions = []
threshold = thresholds[np.argmax(precisions >=0.70)]
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))
```

```
Accuracy on testing set: 0.6829268292682927
```

```
print('Precision on testing set:', precision_score(binary_predictions, y_test))
```

```
Precision on testing set: 1.0
```

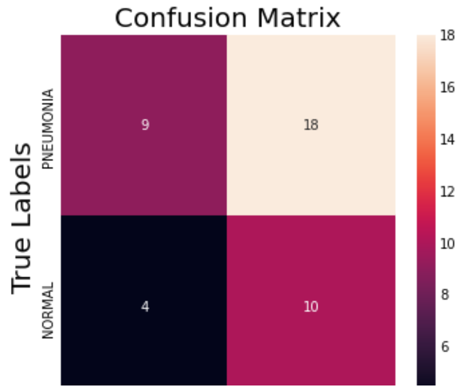
```
print('Recall on testing set:', recall_score(binary_predictions, y_test))
```

```
Recall on testing set: 0.6829268292682927
```

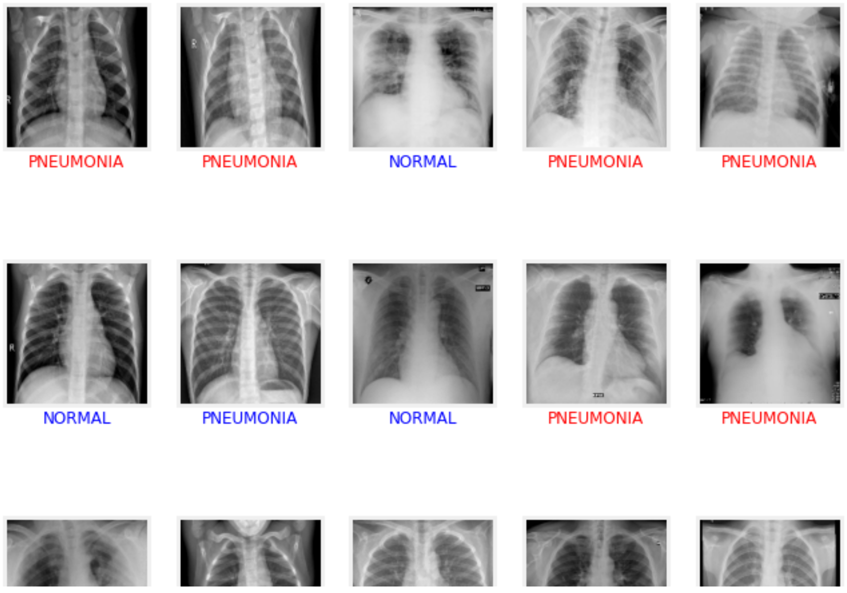
VISUALIZATION OF CONFUSION MATRIX

```
matrix = confusion_matrix(binary_predictions, y_test)
plt.figure(figsize=(5,5))
ax= plt.subplot()
sns.heatmap(matrix, annot=True, ax = ax)
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)
```

```
[Text(0, 0.5, 'PNEUMONIA'), Text(0, 1.5, 'NORMAL')]
```



```
plt.figure(figsize=(10,10))
for i in range(15):
    plt.subplot(3,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train.reshape(-1, img_size, img_size)[i], cmap='gray')
    if(binary_predictions[i]==y_test[i]):
        plt.xlabel(labels[binary_predictions[i]], color='blue')
    else:
        plt.xlabel(labels[binary_predictions[i]], color='red')
plt.show()
```



✓ 2s completed at 4:33 PM