In [1]:

```
#Import into Colab
```

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

In [2]:

```
#Imporing libraries
```

In [3]:

```
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, Dropo
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_ma
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
```

In [3]:

```
#Labeling and sizing
```

In [4]:

```
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)
```

In [4]:

```
#Training data
```

In [5]:

```
train = get_training_data('/content/drive/MyDrive/Colab Notebooks/Kaggle/chest_xray/train')
test = get_training_data('/content/drive/MyDrive/Colab Notebooks/Kaggle/chest_xray/test')
val = get_training_data('/content/drive/MyDrive/Colab Notebooks/Kaggle/chest_xray/val')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15: VisibleDepr
ecationWarning: Creating an ndarray from ragged nested sequences (which is a
list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shape
s) is deprecated. If you meant to do this, you must specify 'dtype=object' w
hen creating the ndarray
  from ipykernel import kernelapp as app
```

In [7]:

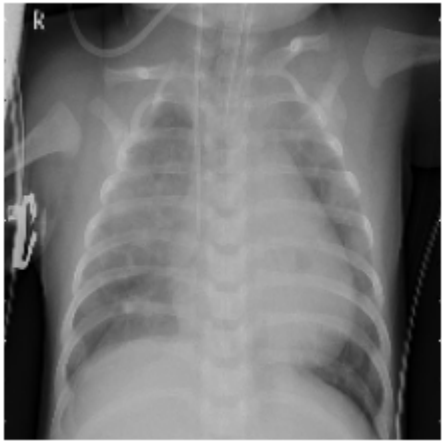
```
pnenumonia = 0
normal = 0

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
```

In [8]:

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

```
PNEUMONIA
```



In [5]:

```
#Labelling feature
```

In [9]:

```
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_
```

In [10]:

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

In [6]:

```
#ImageDataGenerator
```

In [11]:

```
# 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)
```

In [7]:

```
#Optimizers for model
```

In [12]:

```
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'])
```

In [13]:

```
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
_________________________________________________________________
```

In [ ]:

```
y = model.fit(datagen.flow(X_train, y_train, batch_size=10), callbacks=[early_stop], valida
```

```
Epoch 1/15
375/375 [==============================] - 1690s 5s/step - loss: 0.6110 -
acc: 0.7128 - val_loss: 0.6822 - val_acc: 0.7407
Epoch 2/15
375/375 [==============================] - 1735s 5s/step - loss: 0.4881 -
acc: 0.7427 - val_loss: 0.4381 - val_acc: 0.7449
Epoch 3/15
375/375 [==============================] - 1726s 5s/step - loss: 0.4171 -
acc: 0.7966 - val_loss: 0.4585 - val_acc: 0.7150
Epoch 4/15
375/375 [==============================] - 1720s 5s/step - loss: 0.3807 -
acc: 0.8217 - val_loss: 0.4891 - val_acc: 0.6894
Epoch 5/15
375/375 [==============================] - 1718s 5s/step - loss: 0.3396 -
acc: 0.8548 - val_loss: 0.2462 - val_acc: 0.9007
Epoch 6/15
375/375 [==============================] - 1732s 5s/step - loss: 0.3215 -
acc: 0.8551 - val_loss: 0.2492 - val_acc: 0.8986
Epoch 7/15
375/375 [==============================] - 1736s 5s/step - loss: 0.3127 -
acc: 0.8620 - val_loss: 0.2851 - val_acc: 0.8762
Epoch 8/15
```

```
265/375 [====================>.........] - ETA: 7:57 - loss: 0.3056 - acc:
0.8751
```

In [8]:

```
#Evaluation
```

In [15]:

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

```
37/37 [==============================] - 147s 4s/step - loss: 0.2403 - acc:
0.9036
```

Out[15]:

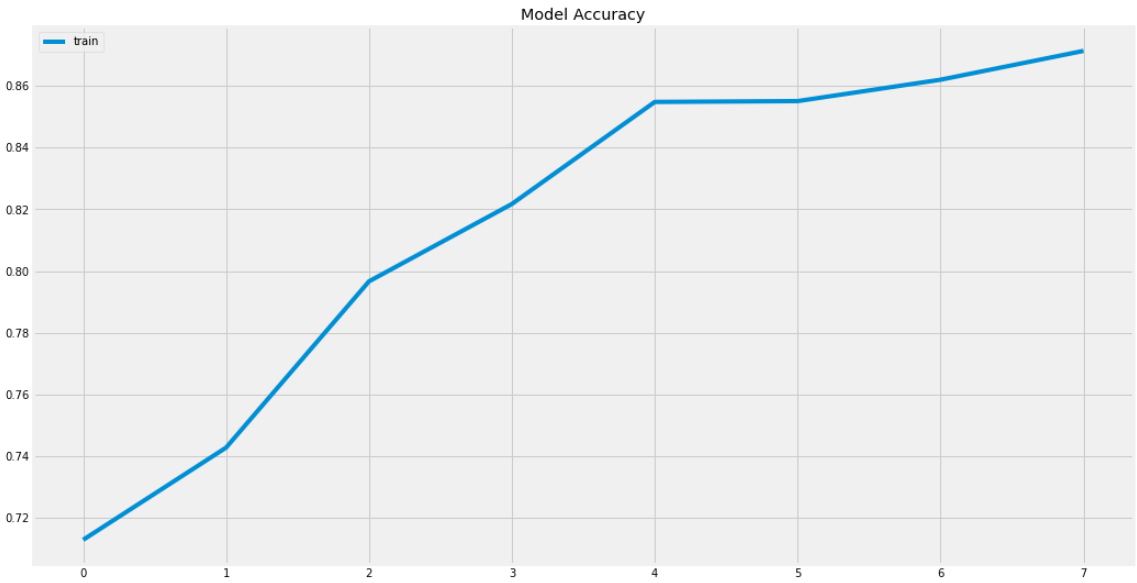
```
[0.24025382101535797, 0.9035836458206177]
```

In [9]:

```
#Graph showing accuracy and loss
```

In [16]:

```
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()
```

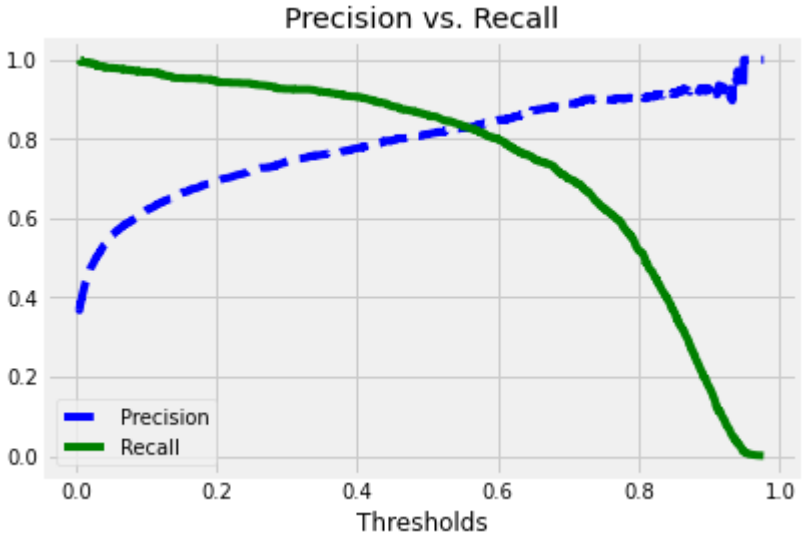


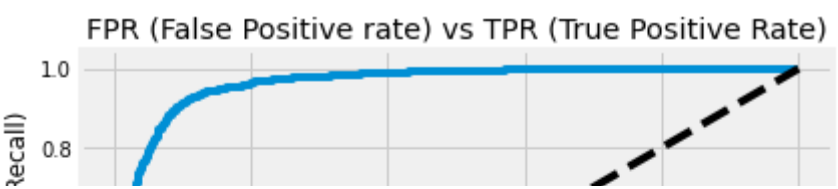
In [17]:

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

In [18]:

```
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)
```


Precision vs. Recall
1.0
0.8
0.6
0.4
0.2
0.0
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
Thresholds



In [10]:

```
#Predictions
```

In [19]:

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

In [20]:

```
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)
```

In [21]:

```
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)
```

In [22]:

```
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: 0.9001706484641638
Precision on testing set: 0.8693009118541033
Recall on testing set: 0.7944444444444444
```

In [11]:

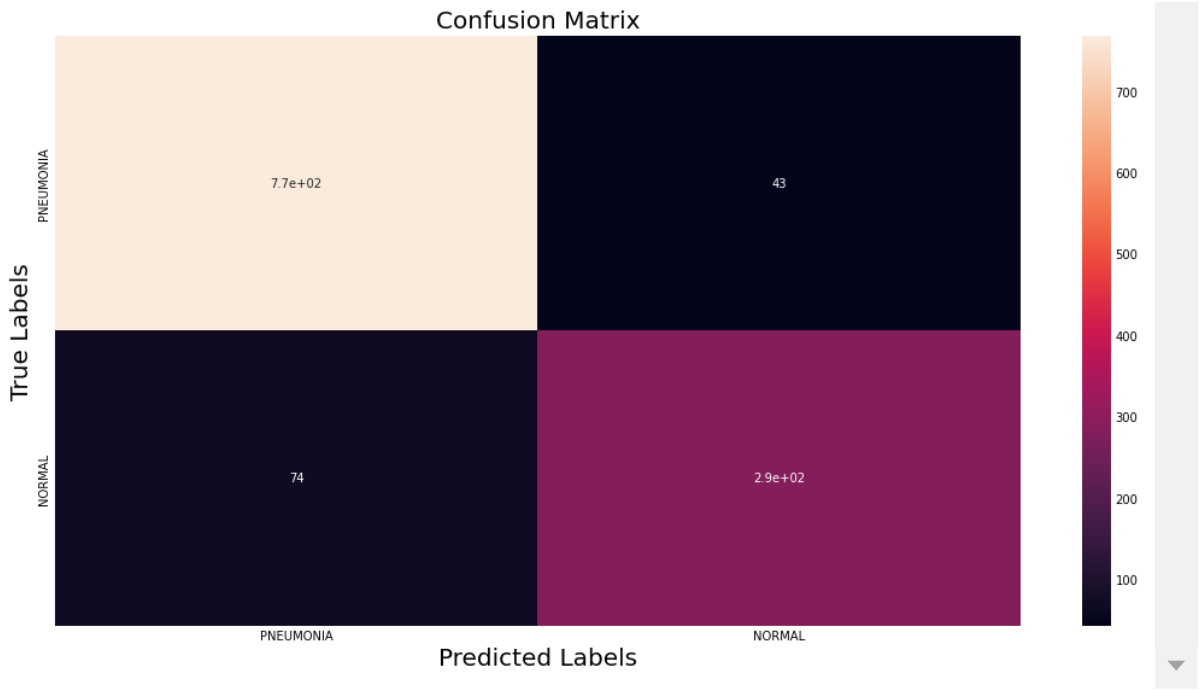
```
#Confusion matrix
```

In [23]:

```
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)
```

Out[23]:

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

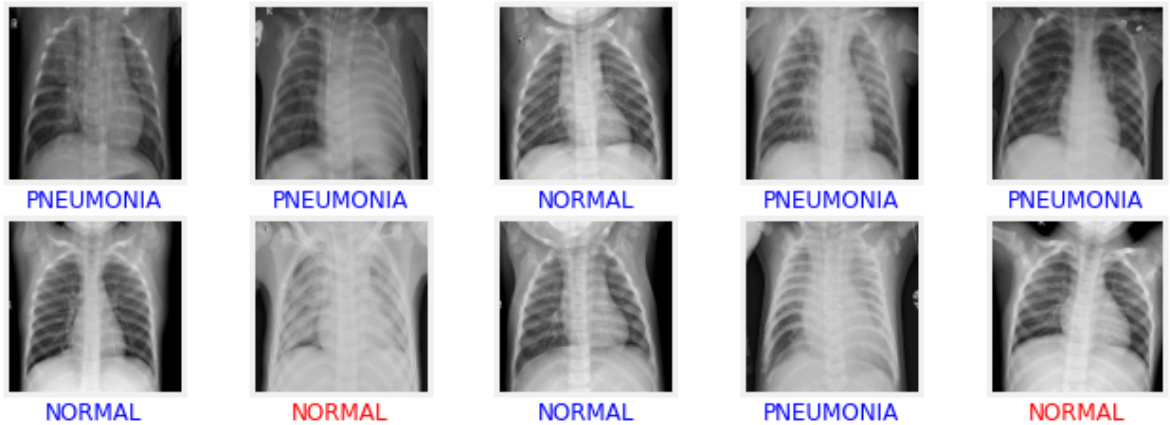


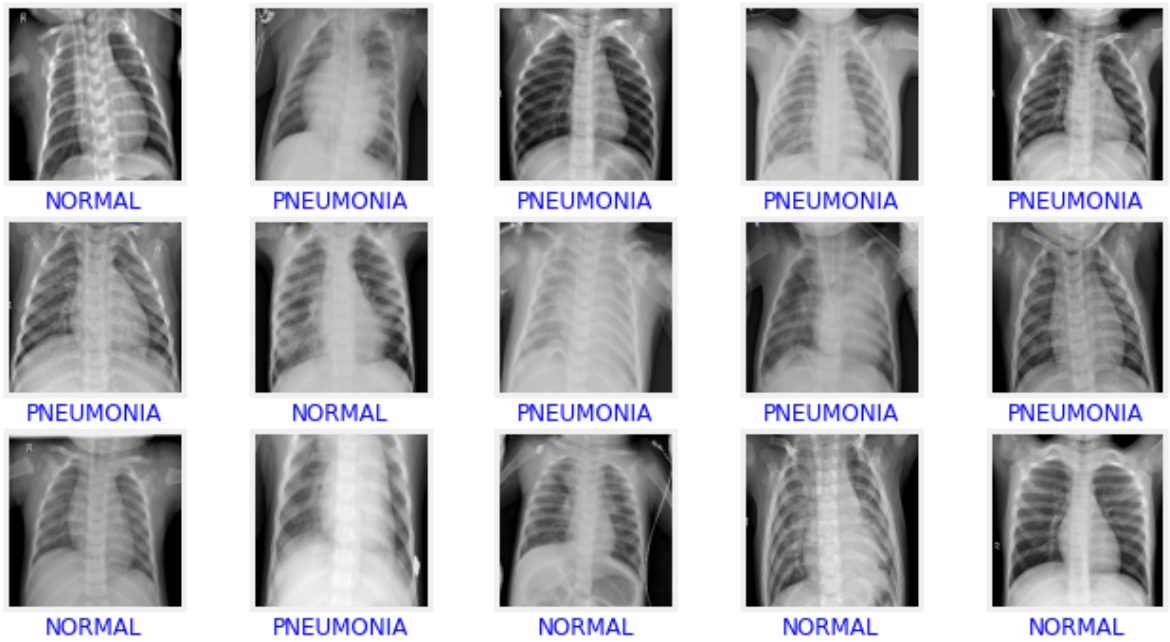
In [12]:

```
#Results
```

In [24]:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,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()
```




NORMAL
PNEUMONIA
PNEUMONIA
PNEUMONIA
PNEUMONIA
PNEUMONIA
NORMAL
PNEUMONIA
PNEUMONIA
PNEUMONIA
NORMAL
PNEUMONIA
NORMAL
NORMAL
NORMAL

In [ ]: