B M S COLLEGE OF ENGINEERING

(An Autonomous Institution Affiliated to VTU, Belagavi)

Post Box No.: 1908, Bull Temple Road, Bengaluru – 560 019

DEPARTMENT OF MACHINE LEARNING

Academic Year: 2023-2024 (Session: Nov 2023 - Mar 2024)

INTRODUCTION TO NEURAL NETWORKS (23AM3PCINN)

ALTERNATIVE ASSESSMENT TOOL (AAT)

MEDICAL IMAGE SEGMENTATION

Submitted by

Student Name:	**Chetna Mundra**	**Aryaman Sharma**
USN:	**1BM21AI036**	**1BM21AI027**
Date:	**10-01-2024**	
Semester & Section:	**V-A**	
Total Pages:	**10**	
Student Signature:		

Valuation Report **(to be filled by the faculty)**

Score:	
Comments:	
Faculty In-charge:	**Dr. Seemanthini K**
Faculty Signature: with date	

INDEX

1. INTRODUCTION

In recent years, medical image segmentation has emerged as a pivotal tool in the field of diagnostic radiology, playing a crucial role in the accurate and timely identification of various diseases. Among these, pneumonia is a leading respiratory ailment with significant global health implications. Rapid and precise detection of pneumonia in chest X-rays is imperative for timely intervention and effective patient care. This report delves into the advancements and challenges associated with medical image segmentation techniques employed in the diagnosis of pneumonia through the analysis of X-rayed lungs.

The conventional methods of pneumonia detection often rely on visual examination by radiologists, a process that is not only time-consuming but also subject to human error. The integration of artificial intelligence and image segmentation techniques has paved the way for more efficient and reliable diagnosis. By isolating and highlighting specific regions of interest within chest X-rays, these technologies facilitate the identification of pneumonia-related anomalies, thereby assisting healthcare professionals in making informed decisions and expediting patient treatment.

This report explores the current state-of-the-art in medical image segmentation for pneumonia diagnosis, encompassing the methodologies, challenges, and potential applications. Through an in-depth examination of the various techniques and technologies employed in this domain, we aim to shed light on the transformative impact of image segmentation on the accuracy and efficiency of pneumonia detection in chest X-rays. As we navigate through the intricacies of these cutting-edge approaches, a comprehensive understanding will be developed, laying the groundwork for improved diagnostic tools and enhanced patient outcomes.

2. METHODOLOGY

2.1 Dataset Preprocessing & Visualization

This section details the preparation and exploration of a chest X-ray dataset for pneumonia diagnosis. The dataset undergoes an 80%, 15%, 5% split for training, testing, and validation, respectively. Notably imbalanced, the dataset contains 1224 normal and 3418 pneumonia cases. Image loading and preprocessing involve resizing, converting to grayscale, transforming to RGB, and normalization. The resulting arrays, paired with labels (0 for normal, 1 for pneumonia), are created. Occurrence counts reveal the class imbalance in the training set. Visualizations provide insight into the preprocessed images, offering a representative glimpse of the dataset's diversity across training, testing, and validation sets. These steps lay the groundwork for subsequent model development, addressing imbalances and ensuring robust analysis in pneumonia detection.

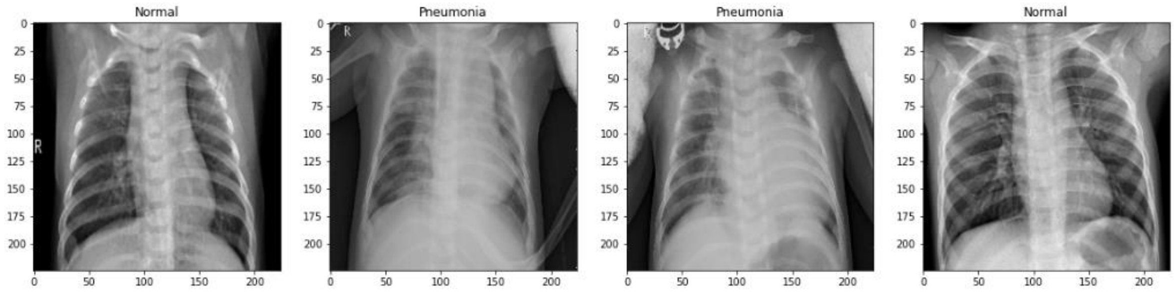


Fig 1. Refined Dataset

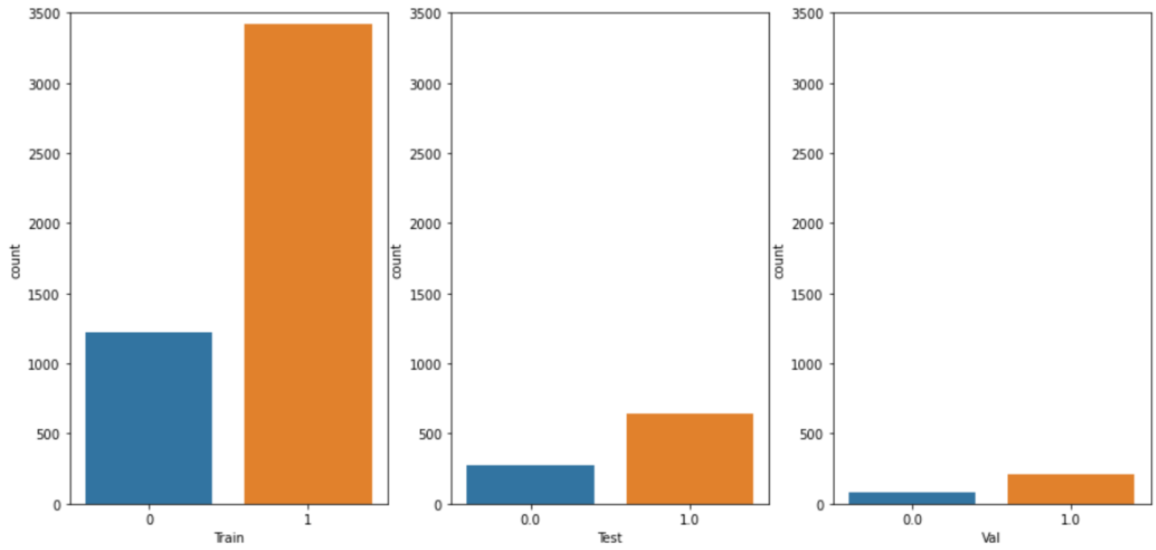


Fig 2. Graph Depicting Number of Images Alloted

2.2 Dealing with class imbalance

To tackle the class imbalance, we adopt a pragmatic approach by assigning weights to each class, ensuring equitable learning for the Convolutional Neural Network (CNN). Utilizing sklearn's `compute_class_weight` function, class weights {0: 1.896, 1: 0.679} are computed, prioritizing the minority class (pneumonia). Additionally, to optimize memory usage, a balanced subset of images is created for training. The resulting datasets for training, testing, and validation, along with their respective shapes, are established. These measures strategically address the class imbalance, providing a foundation for a more effective CNN model training, where each class contributes proportionately to the learning process.

2.3 Training

The training phase commences with a batch size of 32, chosen for its efficiency and memory optimization. Prior to training, memory is cleared to enhance computational resources. The dataset lengths for training and validation are determined, and image augmentation techniques are employed to artificially expand the dataset, preventing overfitting.

A MobileNet architecture, a pre-trained Convolutional Neural Network (CNN), is utilized to expedite training. The model is compiled with binary crossentropy loss, Adam optimizer, and relevant evaluation metrics. To enhance the model's robustness, a class weight parameter is incorporated during training.

Training is conducted for 64 epochs using a generator-based approach, with image augmentation applied in real-time. The process leverages the power of transfer learning from MobileNet, showcasing its adaptability for pneumonia detection in chest X-rays. The training history is recorded for subsequent analysis, ensuring the model's efficacy in pneumonia classification.

2.4 Plotting validation accuracy and loss

The graph depicts the training and validation accuracy of a machine learning model over 60 epochs. The red line represents the training accuracy, which starts at 0.7 and increases to 0.93 by epoch 60. The blue line represents the validation accuracy, which starts at 0.7 and increases to 0.88 by epoch 60. The gap between the training and validation accuracy suggests that the model may be overfitting the training data.

In addition to the accuracy graph, the image also shows a graph of the training and validation loss. The training loss starts at 0.5 and decreases to 0.03 by epoch 60. The validation loss starts at 0.5 and decreases to 0.12 by epoch 60. The smaller gap between the training and validation loss compared to the accuracy suggests that the model may be underfitting the data.

Overall, the graphs suggest that the model is making progress on the training data, but it may not be generalizing well to unseen data. This could be due to overfitting or underfitting.

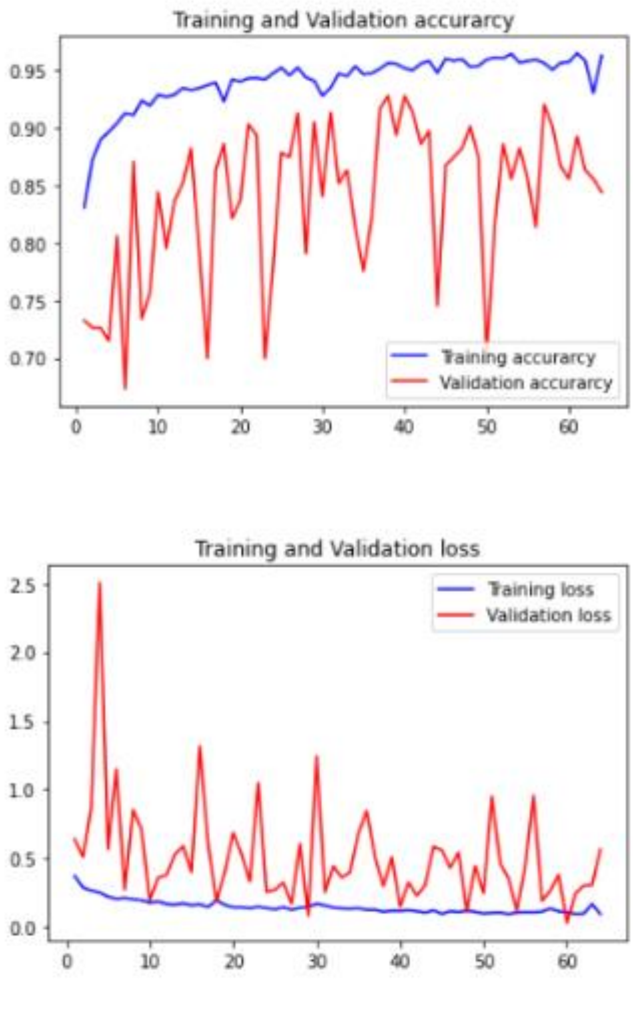


Fig 3. Accuracy and Loss

2.5 Estimation of classification performance

The model's classification performance is assessed using various metrics, including Accuracy, Precision, Recall, and F1-score. For the test set, the model achieves an Accuracy of 86.07%, Precision of 83.88%, Recall of 99.06%, and an F1-score of 90.84%. The confusion matrix reveals 156 true negatives, 122 false positives, 6 false negatives, and 635 true positives. In the training phase, the model achieves an accuracy of 96.27%. The visual representation of the confusion matrix further illustrates the model's effectiveness in distinguishing between pneumonia and normal chest X-ray images.

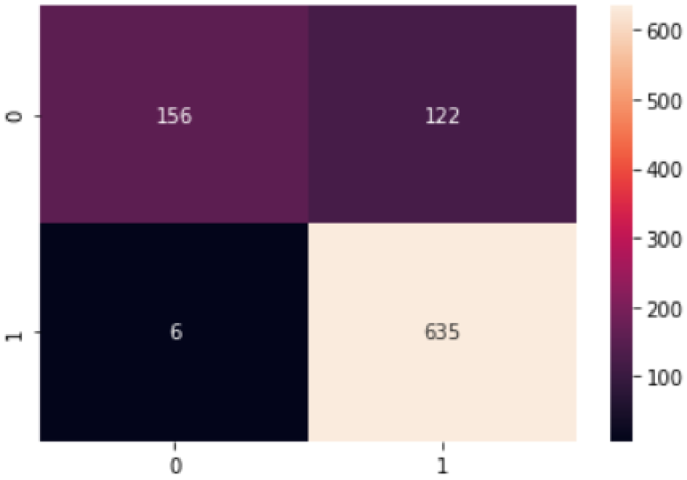


Fig 4. Heatmap

3. FLOWCHART

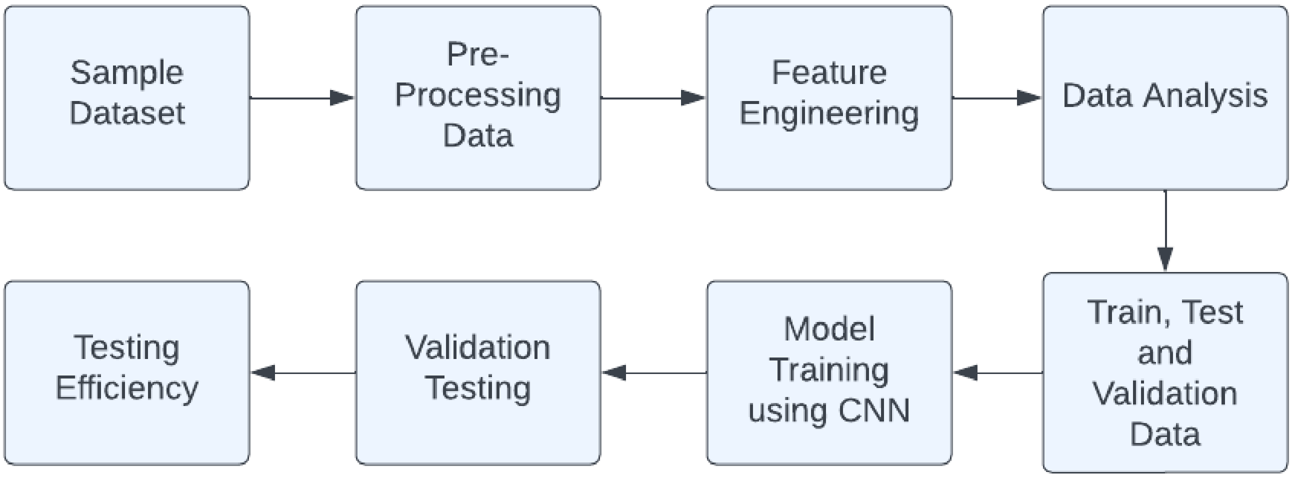


Fig 5. Flowchart

4. **RESULTS**

ROC AUC Score: 0.96, indicating excellent model discrimination between pneumonia and normal chest X-ray images.

- Model Performance:
- Accuracy (test set): 86.07%
- Precision (test set): 83.88%
- Recall (test set): 99.06%
- F1-score (test set): 90.84%
- Accuracy (training set): 96.27%

```
AUC Score: 0.9572385773128768
```

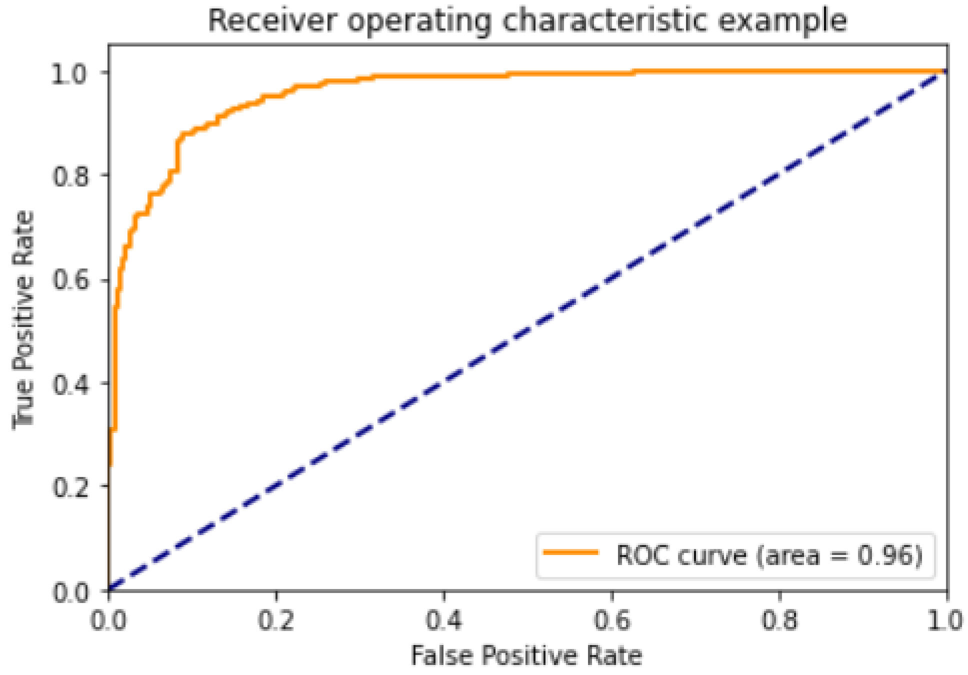


Fig 6. ROC Curve

The ROC curve shows the trade-off between correctly identifying diseased patients (true positive rate) and incorrectly identifying healthy patients (false positive rate) at different threshold levels.

The AUC score of 0.96 indicates excellent performance of the model in distinguishing between pneumonia and normal chest X-ray images.

5. APPLICATIONS

The CNN model for pneumonia detection in chest X-rays has several valuable applications in the medical field:

1. Early Diagnosis: The model aids in the early detection of pneumonia, enabling prompt medical intervention and improving patient outcomes.

2. Efficient Triage: Emergency rooms and healthcare facilities can use the model to quickly triage patients with potential pneumonia, prioritizing those who require immediate attention.

3. Resource Optimization: Hospitals can optimize resources by swiftly identifying pneumonia cases, streamlining patient management, and allocating medical staff and equipment effectively.

4. Telemedicine Support: In remote or underserved areas, the model facilitates telemedicine by providing preliminary pneumonia assessments through chest X-ray scans, bridging the gap in healthcare access.

5. Clinical Decision Support: Healthcare professionals can use the model as a decision support tool, enhancing diagnostic accuracy and aiding in treatment planning.

6. Research and Epidemiology: The model contributes to large-scale studies and epidemiological research by efficiently analyzing chest X-rays, helping to understand the prevalence and trends of pneumonia.

7. Educational Tool: The model serves as an educational tool for medical students and practitioners, offering insights into image interpretation and enhancing diagnostic skills in pneumonia identification.

6. CONCLUSION

The model demonstrates a strong ability to distinguish between pneumonia and normal chest X-ray images, achieving a high ROC AUC score of 0.96 and generally favorable performance metrics across accuracy, precision, recall, and F1-score. While exhibiting a slight decrease in accuracy from training to test sets, it maintains a high recall, suggesting a low rate of false negatives (misclassifying pneumonia cases as normal). This model shows significant potential for aiding in the accurate diagnosis of pneumonia.

.

In [1]:

```python
import os
import numpy as np
import pandas as pd
import pathlib
import imageio
```

In [2]:

```python
# Exploring dataset
base_dir = '../input/chest-xray-pneumonia/chest_xray/'

train_pneumonia_dir = base_dir+'train/PNEUMONIA/'
train_normal_dir=base_dir+'train/NORMAL/'

test_pneumonia_dir = base_dir+'test/PNEUMONIA/'
test_normal_dir = base_dir+'test/NORMAL/'

val_normal_dir= base_dir+'val/NORMAL/'
val_pnrumonia_dir= base_dir+'val/PNEUMONIA/'

train_pn = [train_pneumonia_dir+"{}".format(i) for i in os.listdir(train_pneumonia_
train_normal = [train_normal_dir+"{}".format(i) for i in os.listdir(train_normal_di

test_normal = [test_normal_dir+"{}".format(i) for i in os.listdir(test_normal_dir)]
test_pn = [test_pneumonia_dir+"{}".format(i) for i in os.listdir(test_pneumonia_dir

val_pn= [val_pnrumonia_dir+"{}".format(i) for i in os.listdir(val_pnrumonia_dir) ]
val_normal= [val_normal_dir+"{}".format(i) for i in os.listdir(val_normal_dir) ]

print ("Total images:",len(train_pn+train_normal+test_normal+test_pn+val_pn+val_nor
print ("Total pneumonia images:",len(train_pn+test_pn+val_pn))
print ("Total Nomral images:",len(train_normal+test_normal+val_normal))
```

```
Total images: 5856
Total pneumonia images: 4273
Total Nomral images: 1583
```

Dataset Preprocessing & Visualization

In [3]:

```python
# Gathering all pneumina and normal chest X-ray in two python list
pn = train_pn + test_pn + val_pn
normal = train_normal + test_normal + val_normal

# Spliting dataset in train set,test set and validation set.

train_imgs = pn[:3418]+ normal[:1224]  # 80% of 4273 Pneumonia and normal chest X-r
test_imgs = pn[3418:4059]+ normal[1224:1502]
val_imgs = pn[4059:] + normal[1502:]

print("Total Train Images %s containing %s pneumonia and %s normal images"
      % (len(train_imgs),len(pn[:3418]),len(normal[:1224])))
print("Total Test Images %s containing %s pneumonia and %s normal images"
      % (len(test_imgs),len(pn[3418:4059]),len(normal[1224:1502])))
print("Total validation Images %s containing %s pneumonia and %s normal images"
      % (len(val_imgs),len(pn[4059:]),len(normal[1502:])))



import random

random.shuffle(train_imgs)
random.shuffle(test_imgs)
random.shuffle(val_imgs)
```

```
Total Train Images 4642 containing 3418 pneumonia and 1224 normal images
Total Test Images 919 containing 641 pneumonia and 278 normal images
Total validation Images 295 containing 214 pneumonia and 81 normal images
```

Loading each image and their label into array

In [5]:

```
import cv2
img_size = 224

def preprocess_image(image_list):

    X = []
    y = []
    count=0

    for image in image_list:

        try:

            img = cv2.imread(image,cv2.IMREAD_GRAYSCALE)

            img=cv2.resize(img,(img_size,img_size),interpolation=cv2.INTER_CUBIC)

            #convert image to 2D to 3D
            img = np.dstack([img, img, img])

            #convrt greyscale image to RGB
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            # Normalalize Image
            img = img.astype(np.float32)/255.

            count=count+1

            X.append(img)


        except:
            continue
        #get the labels
        if 'NORMAL' in image:
            y.append(0)

        elif 'IM' in image:
            y.append(0)

        elif 'virus' or 'bacteria' in image:
            y.append(1)


    return X, y
```

In [6]:

```
X, y = preprocess_image(train_imgs)
```

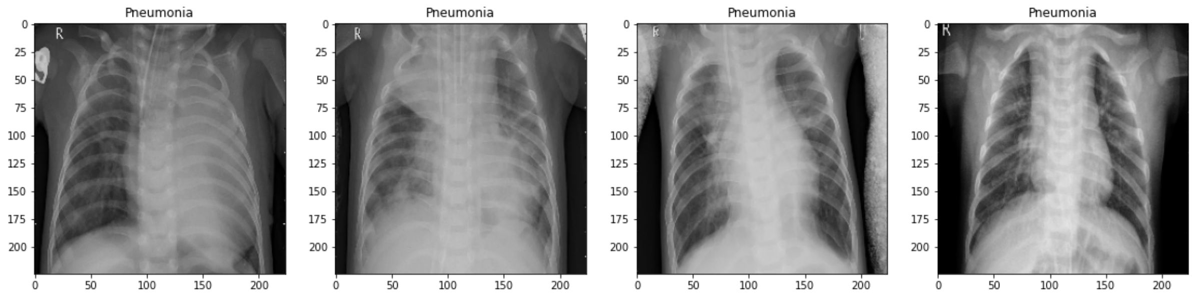
In [7]:

```
arr=y
uniqueValues, occurCount = np.unique(arr, return_counts=True)

print("Unique Values : " , uniqueValues)
print("Occurrence Count : ", occurCount)
```

```
Unique Values :  [0 1]
Occurrence Count :  [1224 3418]
```

In [8]:

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(20, 5))
k=1
for i in range(4):
    a = fig.add_subplot(1, 4, k)
    if (y[i]==0):
        a.set_title('Normal')
    else:
        a.set_title('Pneumonia')

    plt.imshow(X[i])
    k=k+1;
```



In [9]:

```
P, t = preprocess_image(test_imgs)
```

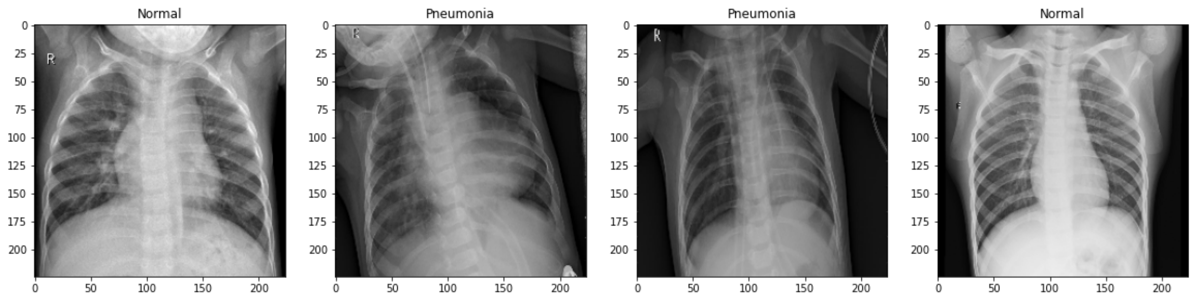
In [10]:

```
arr=t
uniqueValues, occurCount = np.unique(arr, return_counts=True)

print("Unique Values : " , uniqueValues)
print("Occurrence Count : ", occurCount)
```

```
Unique Values :  [0 1]
Occurrence Count :  [278 641]
```

In [11]:

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(20, 5))
k=1
for i in range(4):
    a = fig.add_subplot(1, 4, k)
    if (t[i]==0):
        a.set_title('Normal')
    else:
        a.set_title('Pneumonia')

    plt.imshow(P[i])
    k=k+1;
```



In [12]:

```
K, m = preprocess_image(val_imgs)
```

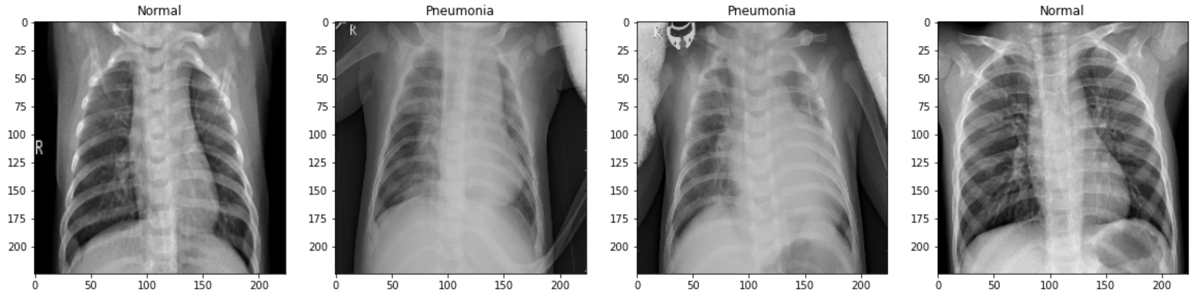
In [13]:

```
arr=m
```

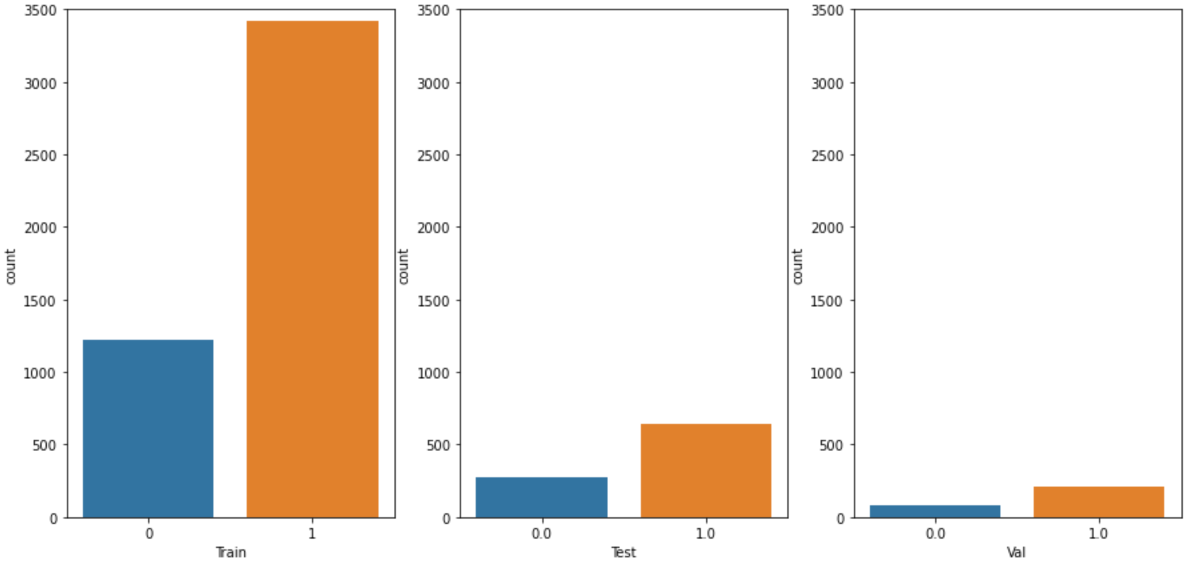
```
uniqueValues, occurCount = np.unique(arr, return_counts=True)

print("Unique Values : " , uniqueValues)
print("Occurrence Count : ", occurCount)
```

```
Unique Values :  [0 1]
Occurrence Count :  [ 81 214]
```

In [14]:
```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(20, 5))
k=1
for i in range(4):
    a = fig.add_subplot(1, 4, k)
    if (m[i]==0):
        a.set_title('Normal')
    else:
        a.set_title('Pneumonia')

    plt.imshow(K[i])
    k=k+1;
```



In [15]:
```
import seaborn as sns

df=pd.DataFrame()
df['Train']=y
df['Test']=pd.Series(t)
df['Val']=pd.Series(m)


fig, ax =plt.subplots(1,3,figsize=(15,7))
sns.countplot(df['Train'], ax=ax[0])
ax[0].set(ylim=(0, 3500))

sns.countplot(df['Test'], ax=ax[1])
ax[1].set(ylim=(0, 3500))

sns.countplot(df['Val'], ax=ax[2])
ax[2].set(ylim=(0, 3500))

fig.show()
```



In [16]:
```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(y), # here, y contains t
                                                  y)
class_weights = dict(enumerate(class_weights))
print(class_weights)
```

```
{0: 1.8962418300653594, 1: 0.679052077238151}
```

In [17]:
```
import seaborn as sns
import gc

train_imgs = train_pn[:3875]+ train_normal[:1341]
del train_imgs
gc.collect()

X_train = np.array(X)
y_train = np.array(y)
X_test = np.array(P)
y_test = np.array(t)
X_val = np.array(K)
y_val = np.array(m)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
print(X_val.shape)
print(y_val.shape)
```

```
(4642, 224, 224, 3)
(4642,)
(919, 224, 224, 3)
(919,)
(295, 224, 224, 3)
(295,)
```

Training

In [18]:
```
# clear memory
del X
del y
gc.collect()
```

```
#get the length of the train and validation data
ntrain = len(X_train)
nval = len(X_val)

batch_size = 32
```

In [19]:
```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(  rotation_range=7,
                                     width_shift_range=0.05,
                                     height_shift_range=0.05,
                                     shear_range=0.2,
                                     zoom_range=0.45,
                                     horizontal_flip=True)

val_datagen = ImageDataGenerator(zoom_range=0.45)
```

```
Using TensorFlow backend.
```

In [20]:
```
train_generator = train_datagen.flow(X_train, y_train, batch_size=batch_size)
val_generator = val_datagen.flow(X_val, y_val, batch_size=batch_size)
```

In [21]:
```
img_size =224
```

In [22]:
```
from keras import layers
from keras import models
from keras import optimizers
from keras.applications import *
from keras.layers import Dense, GlobalAveragePooling2D
from keras.preprocessing.image import img_to_array, load_img
from keras.models import Model
from keras import backend as K

base_model = MobileNet(weights=None, include_top=False,input_shape=(img_size, img_s

x = base_model.output

x = GlobalAveragePooling2D()(x)

predictions = Dense(1, activation="sigmoid")(x)

model = Model(inputs=base_model.input, outputs=predictions)
# Compile model
model.compile(optimizer='adam', loss = 'binary_crossentropy',
                           metrics = ['binary_accuracy', 'mae'])
```

In [24]:
```
history = model.fit_generator(train_generator,
                              steps_per_epoch=ntrain // batch_size,
                              epochs=64,
                              validation_data=val_generator,
                              validation_steps=nval // batch_size,
                              class_weight =class_weights,
)
```

```
Epoch 1/64
145/145 [==============================] - 82s 563ms/step - loss: 0.3733 - binary_
accuracy: 0.8308 - mae: 0.2146 - val_loss: 0.6394 - val_binary_accuracy: 0.7326 -
val_mae: 0.3425
Epoch 2/64
145/145 [==============================] - 70s 485ms/step - loss: 0.2884 - binary_
accuracy: 0.8722 - mae: 0.1633 - val_loss: 0.5110 - val_binary_accuracy: 0.7262 -
val_mae: 0.3222
Epoch 3/64
145/145 [==============================] - 70s 481ms/step - loss: 0.2651 - binary_
accuracy: 0.8898 - mae: 0.1498 - val_loss: 0.8732 - val_binary_accuracy: 0.7262 -
val_mae: 0.2890
Epoch 4/64
145/145 [==============================] - 70s 481ms/step - loss: 0.2506 - binary_
accuracy: 0.8965 - mae: 0.1382 - val_loss: 2.5055 - val_binary_accuracy: 0.7148 -
val_mae: 0.2860
Epoch 5/64
145/145 [==============================] - 70s 484ms/step - loss: 0.2218 - binary_
accuracy: 0.9039 - mae: 0.1311 - val_loss: 0.5667 - val_binary_accuracy: 0.8061 -
val_mae: 0.2099
Epoch 6/64
145/145 [==============================] - 69s 475ms/step - loss: 0.2055 - binary_
accuracy: 0.9128 - mae: 0.1179 - val_loss: 1.1499 - val_binary_accuracy: 0.6730 -
val_mae: 0.3128
Epoch 7/64
145/145 [==============================] - 69s 474ms/step - loss: 0.2121 - binary_
accuracy: 0.9108 - mae: 0.1212 - val_loss: 0.2709 - val_binary_accuracy: 0.8707 -
val_mae: 0.1499
Epoch 8/64
145/145 [==============================] - 70s 481ms/step - loss: 0.2036 - binary_
accuracy: 0.9236 - mae: 0.1151 - val_loss: 0.8545 - val_binary_accuracy: 0.7338 -
val_mae: 0.2577
Epoch 9/64
145/145 [==============================] - 69s 478ms/step - loss: 0.1978 - binary_
accuracy: 0.9191 - mae: 0.1107 - val_loss: 0.7145 - val_binary_accuracy: 0.7567 -
val_mae: 0.2580
Epoch 10/64
145/145 [==============================] - 69s 479ms/step - loss: 0.1802 - binary_
accuracy: 0.9286 - mae: 0.1018 - val_loss: 0.1946 - val_binary_accuracy: 0.8441 -
val_mae: 0.1726
Epoch 11/64
145/145 [==============================] - 69s 474ms/step - loss: 0.1884 - binary_
accuracy: 0.9269 - mae: 0.1037 - val_loss: 0.3602 - val_binary_accuracy: 0.7951 -
val_mae: 0.2373
Epoch 12/64
145/145 [==============================] - 72s 494ms/step - loss: 0.1685 - binary_
accuracy: 0.9286 - mae: 0.0983 - val_loss: 0.3782 - val_binary_accuracy: 0.8365 -
val_mae: 0.1905
Epoch 13/64
145/145 [==============================] - 68s 472ms/step - loss: 0.1632 - binary_
accuracy: 0.9345 - mae: 0.0922 - val_loss: 0.5275 - val_binary_accuracy: 0.8517 -
val_mae: 0.1582
Epoch 14/64
145/145 [==============================] - 70s 482ms/step - loss: 0.1727 - binary_
accuracy: 0.9328 - mae: 0.0956 - val_loss: 0.5903 - val_binary_accuracy: 0.8821 -
val_mae: 0.1404
Epoch 15/64
145/145 [==============================] - 70s 486ms/step - loss: 0.1587 - binary_
accuracy: 0.9343 - mae: 0.0919 - val_loss: 0.4016 - val_binary_accuracy: 0.7947 -
val_mae: 0.2206
Epoch 16/64
145/145 [==============================] - 71s 489ms/step - loss: 0.1652 - binary_
accuracy: 0.9369 - mae: 0.0946 - val_loss: 1.3163 - val_binary_accuracy: 0.6996 -
val_mae: 0.3074
```

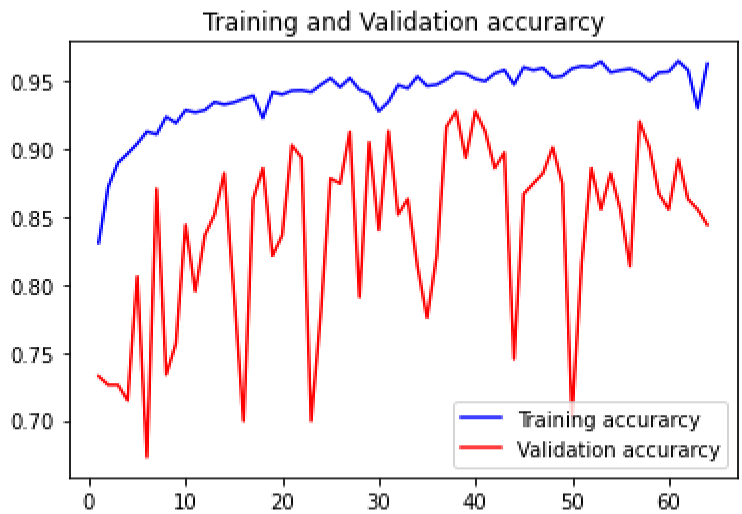
```
Epoch 17/64
145/145 [==============================] - 70s 480ms/step - loss: 0.1470 - binary_
accuracy: 0.9393 - mae: 0.0848 - val_loss: 0.5998 - val_binary_accuracy: 0.8631 -
val_mae: 0.1444
Epoch 18/64
145/145 [==============================] - 68s 471ms/step - loss: 0.2005 - binary_
accuracy: 0.9228 - mae: 0.1095 - val_loss: 0.1908 - val_binary_accuracy: 0.8859 -
val_mae: 0.1341
Epoch 19/64
145/145 [==============================] - 69s 474ms/step - loss: 0.1618 - binary_
accuracy: 0.9418 - mae: 0.0873 - val_loss: 0.4009 - val_binary_accuracy: 0.8213 -
val_mae: 0.2007
Epoch 20/64
145/145 [==============================] - 67s 462ms/step - loss: 0.1474 - binary_
accuracy: 0.9402 - mae: 0.0838 - val_loss: 0.6864 - val_binary_accuracy: 0.8365 -
val_mae: 0.2042
Epoch 21/64
145/145 [==============================] - 69s 478ms/step - loss: 0.1453 - binary_
accuracy: 0.9429 - mae: 0.0820 - val_loss: 0.5333 - val_binary_accuracy: 0.9028 -
val_mae: 0.1253
Epoch 22/64
145/145 [==============================] - 70s 481ms/step - loss: 0.1367 - binary_
accuracy: 0.9432 - mae: 0.0759 - val_loss: 0.3326 - val_binary_accuracy: 0.8935 -
val_mae: 0.1294
Epoch 23/64
145/145 [==============================] - 68s 468ms/step - loss: 0.1486 - binary_
accuracy: 0.9419 - mae: 0.0828 - val_loss: 1.0472 - val_binary_accuracy: 0.6996 -
val_mae: 0.3262
Epoch 24/64
145/145 [==============================] - 66s 458ms/step - loss: 0.1381 - binary_
accuracy: 0.9473 - mae: 0.0773 - val_loss: 0.2562 - val_binary_accuracy: 0.7795 -
val_mae: 0.2285
Epoch 25/64
145/145 [==============================] - 67s 464ms/step - loss: 0.1289 - binary_
accuracy: 0.9521 - mae: 0.0717 - val_loss: 0.2720 - val_binary_accuracy: 0.8783 -
val_mae: 0.1314
Epoch 26/64
145/145 [==============================] - 68s 469ms/step - loss: 0.1469 - binary_
accuracy: 0.9456 - mae: 0.0790 - val_loss: 0.3243 - val_binary_accuracy: 0.8745 -
val_mae: 0.1518
Epoch 27/64
145/145 [==============================] - 68s 472ms/step - loss: 0.1263 - binary_
accuracy: 0.9522 - mae: 0.0714 - val_loss: 0.1715 - val_binary_accuracy: 0.9125 -
val_mae: 0.1201
Epoch 28/64
145/145 [==============================] - 69s 478ms/step - loss: 0.1447 - binary_
accuracy: 0.9437 - mae: 0.0775 - val_loss: 0.6071 - val_binary_accuracy: 0.7909 -
val_mae: 0.2516
Epoch 29/64
145/145 [==============================] - 69s 474ms/step - loss: 0.1481 - binary_
accuracy: 0.9408 - mae: 0.0822 - val_loss: 0.0827 - val_binary_accuracy: 0.9049 -
val_mae: 0.1166
Epoch 30/64
145/145 [==============================] - 68s 468ms/step - loss: 0.2016 - binary_
accuracy: 0.9278 - mae: 0.1022 - val_loss: 1.2444 - val_binary_accuracy: 0.8403 -
val_mae: 0.1715
Epoch 31/64
145/145 [==============================] - 69s 472ms/step - loss: 0.1574 - binary_
accuracy: 0.9347 - mae: 0.0912 - val_loss: 0.2534 - val_binary_accuracy: 0.9132 -
val_mae: 0.1156
Epoch 32/64
145/145 [==============================] - 72s 495ms/step - loss: 0.1464 - binary_
accuracy: 0.9471 - mae: 0.0801 - val_loss: 0.4451 - val_binary_accuracy: 0.8517 -
val_mae: 0.1649
```

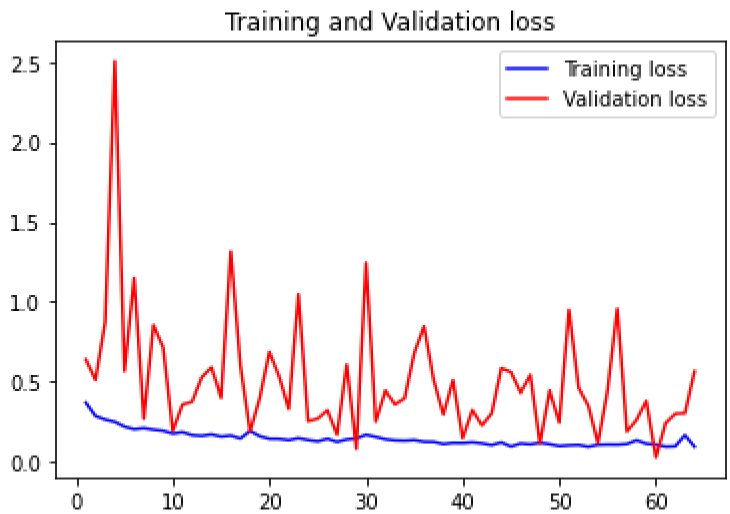
```
Epoch 33/64
145/145 [==============================] - 68s 471ms/step - loss: 0.1351 - binary_
accuracy: 0.9447 - mae: 0.0759 - val_loss: 0.3611 - val_binary_accuracy: 0.8631 -
val_mae: 0.1611
Epoch 34/64
145/145 [==============================] - 69s 473ms/step - loss: 0.1332 - binary_
accuracy: 0.9534 - mae: 0.0740 - val_loss: 0.3995 - val_binary_accuracy: 0.8137 -
val_mae: 0.2037
Epoch 35/64
145/145 [==============================] - 70s 481ms/step - loss: 0.1371 - binary_
accuracy: 0.9464 - mae: 0.0782 - val_loss: 0.6809 - val_binary_accuracy: 0.7757 -
val_mae: 0.2259
Epoch 36/64
145/145 [==============================] - 69s 477ms/step - loss: 0.1270 - binary_
accuracy: 0.9474 - mae: 0.0717 - val_loss: 0.8468 - val_binary_accuracy: 0.8213 -
val_mae: 0.1749
Epoch 37/64
145/145 [==============================] - 68s 472ms/step - loss: 0.1257 - binary_
accuracy: 0.9512 - mae: 0.0690 - val_loss: 0.5111 - val_binary_accuracy: 0.9163 -
val_mae: 0.1042
Epoch 38/64
145/145 [==============================] - 67s 464ms/step - loss: 0.1124 - binary_
accuracy: 0.9562 - mae: 0.0637 - val_loss: 0.2978 - val_binary_accuracy: 0.9278 -
val_mae: 0.1044
Epoch 39/64
145/145 [==============================] - 67s 464ms/step - loss: 0.1197 - binary_
accuracy: 0.9555 - mae: 0.0666 - val_loss: 0.5098 - val_binary_accuracy: 0.8935 -
val_mae: 0.1225
Epoch 40/64
145/145 [==============================] - 69s 476ms/step - loss: 0.1185 - binary_
accuracy: 0.9516 - mae: 0.0672 - val_loss: 0.1492 - val_binary_accuracy: 0.9278 -
val_mae: 0.1012
Epoch 41/64
145/145 [==============================] - 69s 476ms/step - loss: 0.1237 - binary_
accuracy: 0.9499 - mae: 0.0685 - val_loss: 0.3255 - val_binary_accuracy: 0.9132 -
val_mae: 0.1150
Epoch 42/64
145/145 [==============================] - 71s 492ms/step - loss: 0.1169 - binary_
accuracy: 0.9556 - mae: 0.0648 - val_loss: 0.2300 - val_binary_accuracy: 0.8859 -
val_mae: 0.1278
Epoch 43/64
145/145 [==============================] - 70s 486ms/step - loss: 0.1055 - binary_
accuracy: 0.9581 - mae: 0.0596 - val_loss: 0.3041 - val_binary_accuracy: 0.8973 -
val_mae: 0.1139
Epoch 44/64
145/145 [==============================] - 67s 464ms/step - loss: 0.1230 - binary_
accuracy: 0.9475 - mae: 0.0685 - val_loss: 0.5850 - val_binary_accuracy: 0.7452 -
val_mae: 0.2874
Epoch 45/64
145/145 [==============================] - 68s 469ms/step - loss: 0.1049 - binary_
accuracy: 0.9601 - mae: 0.0563 - val_loss: 0.5603 - val_binary_accuracy: 0.8669 -
val_mae: 0.1388
Epoch 46/64
145/145 [==============================] - 68s 467ms/step - loss: 0.1162 - binary_
accuracy: 0.9580 - mae: 0.0606 - val_loss: 0.4321 - val_binary_accuracy: 0.8745 -
val_mae: 0.1485
Epoch 47/64
145/145 [==============================] - 70s 480ms/step - loss: 0.1100 - binary_
accuracy: 0.9596 - mae: 0.0622 - val_loss: 0.5426 - val_binary_accuracy: 0.8821 -
val_mae: 0.1264
Epoch 48/64
145/145 [==============================] - 70s 480ms/step - loss: 0.1250 - binary_
accuracy: 0.9527 - mae: 0.0654 - val_loss: 0.1165 - val_binary_accuracy: 0.9011 -
val_mae: 0.1217
```

```
Epoch 49/64
145/145 [==============================] - 68s 467ms/step - loss: 0.1117 - binary_
accuracy: 0.9537 - mae: 0.0640 - val_loss: 0.4459 - val_binary_accuracy: 0.8745 -
val_mae: 0.1709
Epoch 50/64
145/145 [==============================] - 66s 457ms/step - loss: 0.0992 - binary_
accuracy: 0.9592 - mae: 0.0567 - val_loss: 0.2475 - val_binary_accuracy: 0.7034 -
val_mae: 0.3188
Epoch 51/64
145/145 [==============================] - 66s 457ms/step - loss: 0.1039 - binary_
accuracy: 0.9610 - mae: 0.0563 - val_loss: 0.9502 - val_binary_accuracy: 0.8160 -
val_mae: 0.1892
Epoch 52/64
145/145 [==============================] - 68s 470ms/step - loss: 0.1064 - binary_
accuracy: 0.9603 - mae: 0.0589 - val_loss: 0.4620 - val_binary_accuracy: 0.8859 -
val_mae: 0.1388
Epoch 53/64
145/145 [==============================] - 66s 455ms/step - loss: 0.0956 - binary_
accuracy: 0.9642 - mae: 0.0535 - val_loss: 0.3536 - val_binary_accuracy: 0.8555 -
val_mae: 0.1537
Epoch 54/64
145/145 [==============================] - 66s 452ms/step - loss: 0.1080 - binary_
accuracy: 0.9564 - mae: 0.0607 - val_loss: 0.1186 - val_binary_accuracy: 0.8821 -
val_mae: 0.1505
Epoch 55/64
145/145 [==============================] - 65s 451ms/step - loss: 0.1089 - binary_
accuracy: 0.9579 - mae: 0.0603 - val_loss: 0.4492 - val_binary_accuracy: 0.8555 -
val_mae: 0.1603
Epoch 56/64
145/145 [==============================] - 65s 447ms/step - loss: 0.1086 - binary_
accuracy: 0.9590 - mae: 0.0599 - val_loss: 0.9572 - val_binary_accuracy: 0.8137 -
val_mae: 0.1917
Epoch 57/64
145/145 [==============================] - 64s 444ms/step - loss: 0.1168 - binary_
accuracy: 0.9562 - mae: 0.0637 - val_loss: 0.1908 - val_binary_accuracy: 0.9202 -
val_mae: 0.1060
Epoch 58/64
145/145 [==============================] - 65s 448ms/step - loss: 0.1481 - binary_
accuracy: 0.9503 - mae: 0.0767 - val_loss: 0.2648 - val_binary_accuracy: 0.9011 -
val_mae: 0.1337
Epoch 59/64
145/145 [==============================] - 63s 437ms/step - loss: 0.1160 - binary_
accuracy: 0.9563 - mae: 0.0644 - val_loss: 0.3824 - val_binary_accuracy: 0.8669 -
val_mae: 0.1462
Epoch 60/64
145/145 [==============================] - 65s 448ms/step - loss: 0.1093 - binary_
accuracy: 0.9569 - mae: 0.0614 - val_loss: 0.0294 - val_binary_accuracy: 0.8555 -
val_mae: 0.1441
Epoch 61/64
145/145 [==============================] - 65s 448ms/step - loss: 0.0968 - binary_
accuracy: 0.9646 - mae: 0.0523 - val_loss: 0.2439 - val_binary_accuracy: 0.8924 -
val_mae: 0.1300
Epoch 62/64
145/145 [==============================] - 67s 459ms/step - loss: 0.0979 - binary_
accuracy: 0.9581 - mae: 0.0544 - val_loss: 0.3027 - val_binary_accuracy: 0.8631 -
val_mae: 0.1454
Epoch 63/64
145/145 [==============================] - 64s 441ms/step - loss: 0.1866 - binary_
accuracy: 0.9301 - mae: 0.0979 - val_loss: 0.3057 - val_binary_accuracy: 0.8555 -
val_mae: 0.1643
Epoch 64/64
145/145 [==============================] - 64s 443ms/step - loss: 0.0954 - binary_
accuracy: 0.9627 - mae: 0.0532 - val_loss: 0.5651 - val_binary_accuracy: 0.8441 -
val_mae: 0.1734
```

Plot how validation accuracy and loss are increasing against training accuracy and loss.

In [25]:
```
# Lets plot the train and val curve
# Get the details form the history object
acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()

plt.show()
```





In [26]:

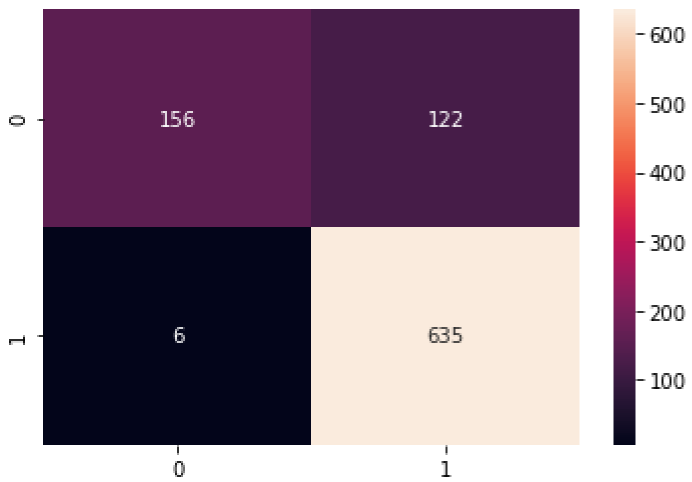
```
from sklearn.metrics import accuracy_score, confusion_matrix

preds = model.predict(X_test)

acc = accuracy_score(y_test, np.round(preds))*100
cm = confusion_matrix(y_test, np.round(preds))

tn, fp, fn, tp = cm.ravel()

print('CONFUSION MATRIX ------------------')
print(cm)

print('\n============TEST METRICS=============')
precision = tp/(tp+fp)*100
recall = tp/(tp+fn)*100
print('Accuracy: {}%'.format(acc))
print('Precision: {}%'.format(precision))
print('Recall: {}%'.format(recall))
print('F1-score: {}'.format(2*precision*recall/(precision+recall)))

print('\nTRAIN METRIC ----------------------')
print('Train acc: {}'.format(np.round((history.history['binary_accuracy'][-1])*100,
```

```
CONFUSION MATRIX ------------------
[[156 122]
 [  6 635]]

============TEST METRICS=============
Accuracy: 86.07181719260065%
Precision: 83.88375165125495%
Recall: 99.06396255850234%
F1-score: 90.84406294706724

TRAIN METRIC ----------------------
Train acc: 96.27
```

In [27]:

```
import seaborn as sns
sns.heatmap(cm, annot=True, fmt="d",)
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4b74f8c290>



In [28]:

```
from sklearn.metrics import roc_curve,roc_auc_score
from sklearn.metrics import auc

fpr , tpr , thresholds = roc_curve ( y_test , preds)
auc_keras = auc(fpr, tpr)
print("AUC Score:",auc_keras)
plt.figure()
```

```python
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % auc_keras)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC Score: 0.9572385773128768
```