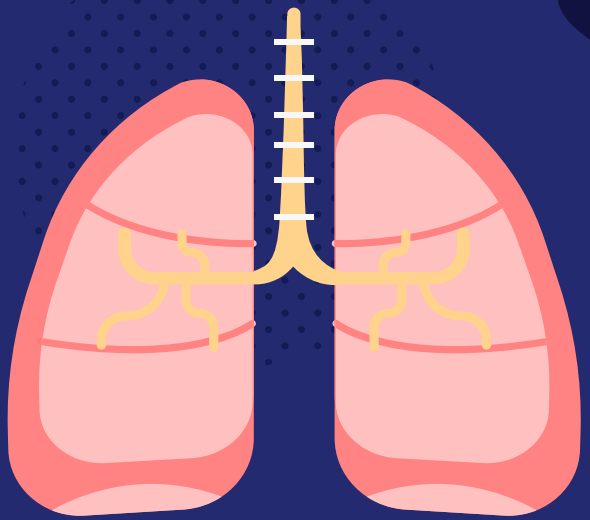


Tuberculosis Lungs

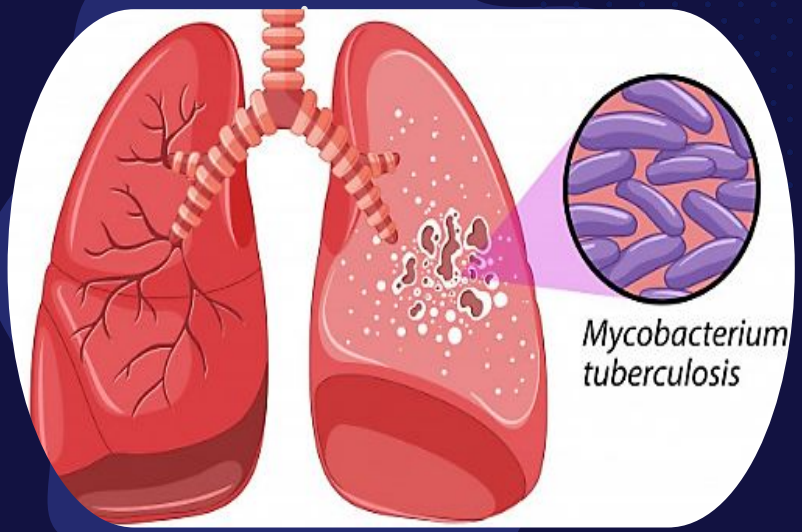
Here is where our presentation begins

CONTENTS OF THIS Dataset :

- Total of Images : **6336**
- Number of normal labels : **3500**
- Number of Tuberculosis labels : **2836**

DEFINITION

Tuberculosis (TB) is a disease caused by germs that are spread from person to person through the air. TB usually affects the lungs, but it can also affect other parts of the body, such as the brain, the kidneys, or the spine. A person with TB can die if they do not get treatment.



Getting The Dataset :

```
[3]: from google.colab import drive

     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Libraries :

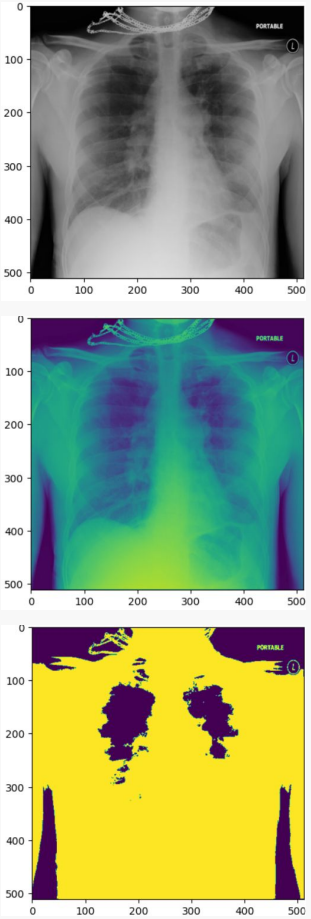
```
import numpy as np
import pandas as pd
import os
from re import search
import shutil
import natsort
from PIL import Image
import matplotlib.pyplot as plt
import cv2
from os import listdir
from matplotlib.image import *
from sklearn.model_selection import train_test_split
import keras
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense,Flatten,Dropout
from keras.layers.normalization import batch_normalization
from matplotlib.image import imread
import tensorflow as tf
# from tensorflow.keras.preprocessing.image.ImageDataGenerator
from tensorflow.keras.preprocessing import *
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

01

Preprocessing The Dataset

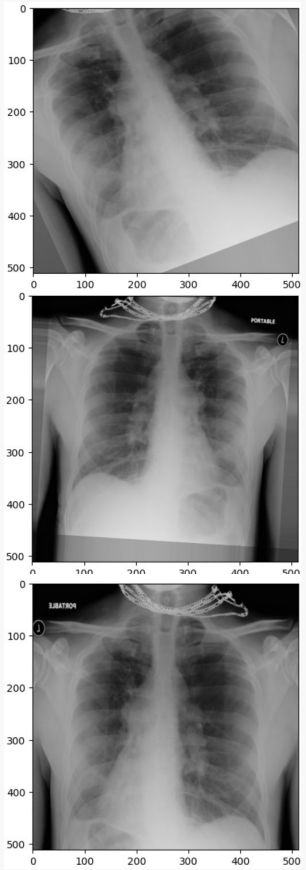
Image Preprocessing:

```
####data(images) preprocessing########
import matplotlib.pylab as plt
Train_DIR=r'/content/drive/MyDrive/'
Categories=['Normal','Tuberculosis']
for j in Categories:
    path=os.path.join(Train_DIR,j)
    for img in os.listdir(path):
        image=cv2.imread(os.path.join(path,img))
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        ret,binary_image = cv2.threshold(gray_image,70,255,0)
        plt.imshow(image)
        plt.show()
        plt.imshow(gray_image)
        plt.show()
        plt.imshow(binary_image)
        plt.show()
        print(gray_image.shape)
        break
    break
```



Balance The Number Of Images:

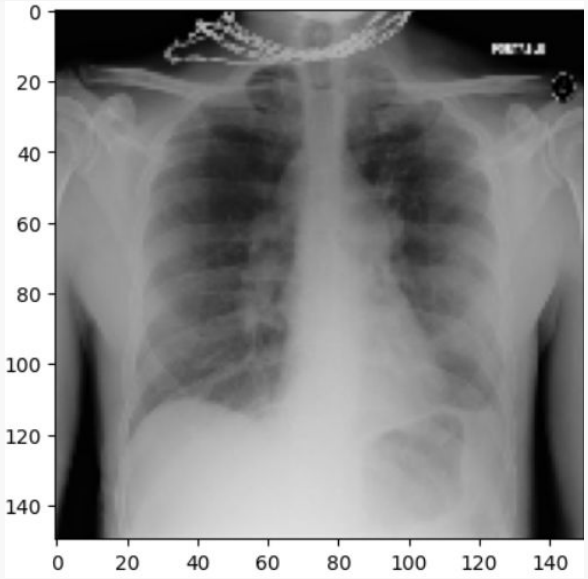
```
train_datagen = ImageDataGenerator(
            rescale=1./255,
            zoom_range=0.2,
            rotation_range = 30,
            width_shift_range = 0.1,
            horizontal_flip=True)
newImage = image.reshape((1,) + image.shape)
i=0
for batch in train_datagen.flow(newImage, batch_size=1):
        img = np.reshape(batch,(512,512,3))
        plt.imshow(img)
        plt.show()
        i += 1
        if i == 3:
                break
```



Resize The Image :

In [7]:

```
IMG_SIZE=150
new_image=cv2.resize(image,(IMG_SIZE,IMG_SIZE))
plt.imshow(new_image)
plt.show()
```



Loading Normal Images and Labels :

```
def load_normal_images_and_labels():
    img_lst=[]
    labels=[]
    j=0
    for index in os.listdir('/content/drive/MyDrive/Normal'):
        img=cv2.imread(os.path.join('/content/drive/MyDrive/Normal',index))
        #resize image to 150*150
        resized_img =cv2.resize(img,(150,150))
         # convert image to gray
        gray_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2GRAY)
        # convert image to binary image
        ret,binary_img = cv2.threshold(gray_img,70,255,0)
        img_lst.append(binary_img)
        labels.append('Normal')
        j+=1

    return img_lst, labels
images1, labels1 = load_normal_images_and_labels()
print("No. of images loaded = ",len(images1),"\nNo. of labels loaded = ",len(labels1))
print(type(images1),type(labels1))
```

```
No. of images loaded =  3500
No. of labels loaded =  3500
<class 'list'> <class 'list'>
```

Loading Tuberculosis Images and Labels :

```
def load_tuberculosis_images_and_labels():
    img_lst=[]
    labels=[]
    newImages=[]
    j=0
    for index in os.listdir('/content/drive/MyDrive/Tuberculosis'):
        img=cv2.imread(os.path.join('/content/drive/MyDrive/Tuberculosis',index))
        #resize image to 150*150
        resized_img = cv2.resize(img,(150,150))
        lab = resized_img.reshape((1,) + resized_img.shape)
        i=0
        for batch in train_datagen.flow(lab, batch_size=1):
                reshapedImage = np.reshape(batch,(150,150,3))
                gray_img = cv2.cvtColor(reshapedImage, cv2.COLOR_BGR2GRAY)
                ret,binary_img = cv2.threshold(gray_img,70,255,0)
                newImages.append(binary_img)
                labels.append('Tuberculosis')
                i += 1
                if i == 3:
                        break
        ######################
        ######################
        # convert image to gray
        gray_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2GRAY)
        # convert image to binary image
        ret,binary_img = cv2.threshold(gray_img,70,255,0)
        img_lst.append(binary_img)
        labels.append('Tuberculosis')
        j+=1

    return img_lst, labels,newImages
images2, labels2 ,newImages = load_tuberculosis_images_and_labels()
print("No. of images loaded = ",len(images2),"\nNo. of new images loaded = ",len(newImages),"\nNo. of labels loaded = ",len(labels2))
print(type(images2),type(labels2))
```

```
No. of images loaded =  709
No. of new images loaded =  2127
No. of labels loaded =  2836
<class 'list'> <class 'list'>
```

Loading Normal Images and Labels :

```
images=images1+images2+newImages
labels=labels1+labels2
images =np.array(images)
labels =np.array(labels)
print("Images shape = ",images.shape,"\nLabels shape = ",labels.shape)
print(type(images),type(labels))
```

```
Images shape =  (6336, 150, 150)
Labels shape =  (6336,)
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

Collecting Images In one List :

```
images=images1+images2+newImages
labels=labels1+labels2
images =np.array(images)
labels =np.array(labels)
print("Images shape = ",images.shape,"\nLabels shape = ",labels.shape)
print(type(images),type(labels))
```

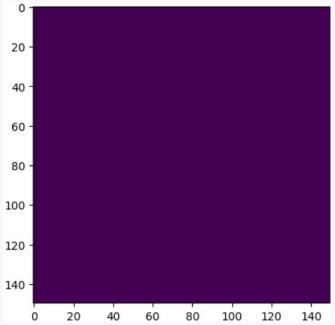
```
Images shape =  (6336, 150, 150)
Labels shape =  (6336,)
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

Shuffling The Dataset :

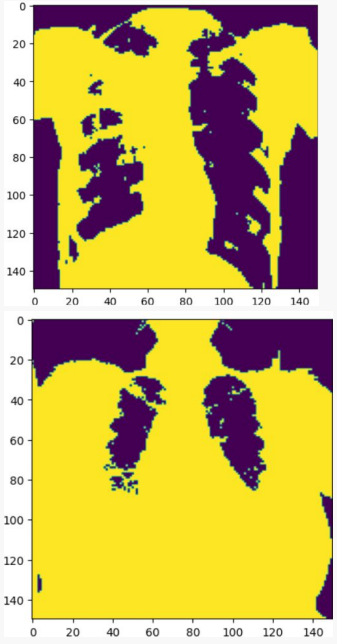
```
from sklearn.utils import shuffle
def shuffle_in_unison(a,b):
    assert len(a)==len(b)
    shuffled_a=np.empty(a.shape,dtype=a.dtype)
    shuffled_b=np.empty(b.shape,dtype=b.dtype)
    permutation=np.random.permutation(len(a))
    for old_index,new_index in enumerate(permutation):
      shuffled_a[new_index]=a[old_index]
      shuffled_b[new_index]=b[old_index]
    return   shuffled_a,shuffled_b
shuffled_a,shuffled_b=shuffle_in_unison(images,labels)
```

```
k=0
for i in shuffled_a:
   plt.imshow(i)
   plt.show()
   k+=1
   if k==10:
            break
k=0
for i in shuffled_b:
   print(i)
   k+=1
   if k==10:
            break
print(len(shuffled_a))
print(len(shuffled_b))
print(shuffled_a.shape)
print(shuffled_b.shape)
```

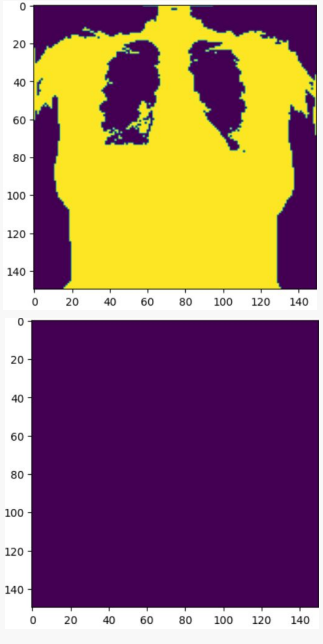
The Result :



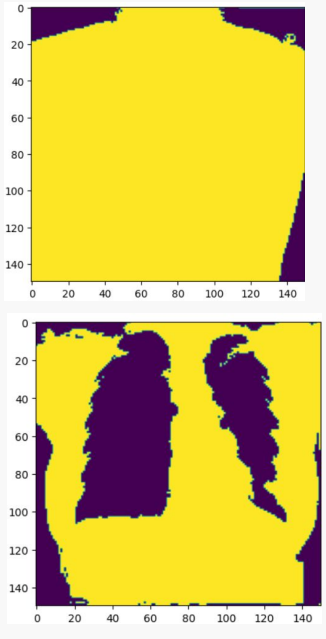
Tuberculosis
Normal



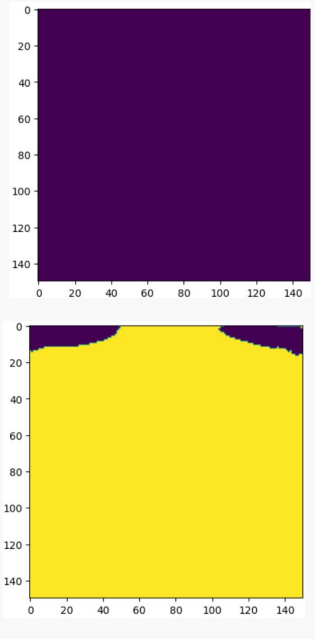
Tuberculosis
Normal



Normal
Tuberculosis



Normal
Tuberculosis



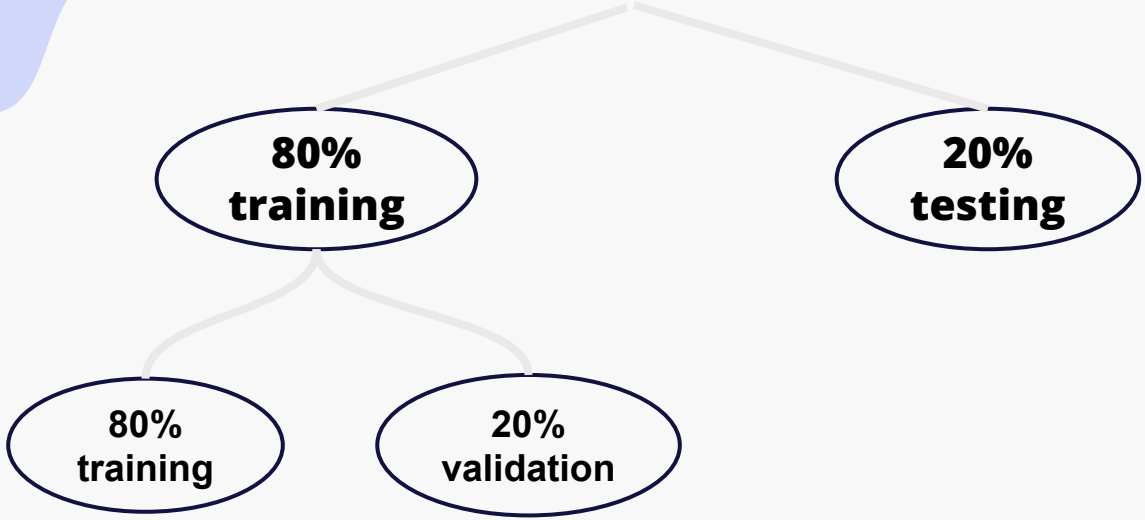
Tuberculosis
Normal

6336 6336 (6336, 150, 150) (6336,)

02

Splitting Dataset

The Dataset



Scale Of Each Feature:

```
####scale of each feature###
X_train_scaled = x_train/255
X_test_scaled = x_test/255
X_val_scaled=x_val/255
print(X_train_scaled.shape)
print(X_test_scaled.shape)
print(X_val_scaled.shape)
print(y_train.shape)
```

```
(3547, 150, 150)
(1268, 150, 150)
(1521, 150, 150)
(3547,)
```

Training Models

First CNN&KNN Model :

CNN Part

```
#building cnn model
cnn_model = Sequential()
cnn_model.add(Conv2D(16,1,padding="same", activation="relu", input_shape=(150,150,1)))
cnn_model.add(MaxPooling2D(padding="same"))

cnn_model.add(Conv2D(32, 1, padding="same", activation="relu"))
cnn_model.add(Dropout(0.5))
cnn_model.add(MaxPooling2D(padding="same"))

cnn_model.add(Conv2D(64,1,padding="same", activation="relu"))
cnn_model.add(MaxPooling2D(padding="same"))

cnn_model.add(Conv2D(128, 1, padding="same", activation="relu"))
cnn_model.add(Dropout(0.5))

cnn_model.add(MaxPooling2D(padding="same"))
cnn_model.add(Conv2D(256, 1, padding="same", activation="relu"))
cnn_model.add(Dropout(0.5))

cnn_model.add(MaxPooling2D(padding="same"))
cnn_model.add(Conv2D(512, 1, padding="same", activation="relu"))
cnn_model.add(Dropout(0.5))

cnn_model.add(MaxPooling2D(padding="same"))
cnn_model.add(Flatten())
#model.add(Dense(128,activation="relu"))
#model.add(Dense(2, activation="softmax"))

cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 16)      32

 max_pooling2d (MaxPooling2D  (None, 75, 75, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 75, 75, 32)        544

 dropout (Dropout)           (None, 75, 75, 32)        0

 max_pooling2d_1 (MaxPooling  (None, 38, 38, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 38, 38, 64)        2112

 max_pooling2d_2 (MaxPooling  (None, 19, 19, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 19, 19, 128)       8320

 dropout_1 (Dropout)         (None, 19, 19, 128)       0

 max_pooling2d_3 (MaxPooling  (None, 10, 10, 128)      0
 2D)

 conv2d_4 (Conv2D)           (None, 10, 10, 256)       33024

 dropout_2 (Dropout)         (None, 10, 10, 256)       0

 max_pooling2d_4 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 conv2d_5 (Conv2D)           (None, 5, 5, 512)         131584

 dropout_3 (Dropout)         (None, 5, 5, 512)         0

 max_pooling2d_5 (MaxPooling  (None, 3, 3, 512)        0
 2D)

 flatten (Flatten)           (None, 4608)              0
```

```
Total params: 175,616
Trainable params: 175,616
Non-trainable params: 0
```

Training For 40 Epoch :

```
early_stopping = tf.keras.callbacks.EarlyStopping(
    min_delta=0.000001,  # minimium amount of change to count as an improvement
    patience=5,  # how many epochs to wait before stopping
)
#cnn_model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'], )
cnn_model.compile(optimizer='adam',
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
#cnn_model.fit(X_train_scaled,y_train , epochs=40)
cnn_model.fit(X_train_scaled,y_train , epochs=40,  validation_data=(X_val_scaled, y_val))
```

Feature Map :

```
# Extract feature maps from the flattened layer of the CNN model
features_train = cnn_model.predict(x_train)
features_val = cnn_model.predict(x_val)
features_test = cnn_model.predict(x_test)
```

```
111/111 [==============================] - 9s 75ms/step
48/48 [==============================] - 5s 111ms/step
40/40 [==============================] - 3s 65ms/step
```

KNN Part

```
knn.fit(features_train, y_train )
```

```
KNeighborsClassifier(n_neighbors=6)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

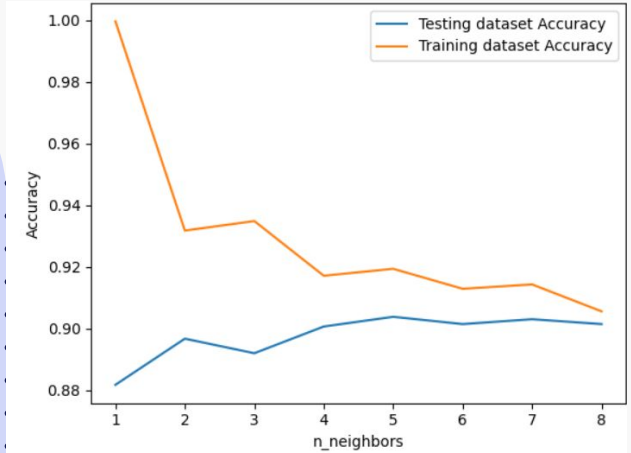
```
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

```
# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(features_train, y_train)

    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(features_train, y_train)
    test_accuracy[i] = knn.score(features_test, y_test)
```

Testing Model

CNN&KNN

```
# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```




```
# Evaluate the combination system
score = knn.score(features_test, y_test)
y_pred = knn.predict(features_test)
print("Accuracy: %.2f%%" % (score*100))
```

Accuracy: 90.14%

```python
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
cm = confusion_matrix(y_test, y_pred)
```

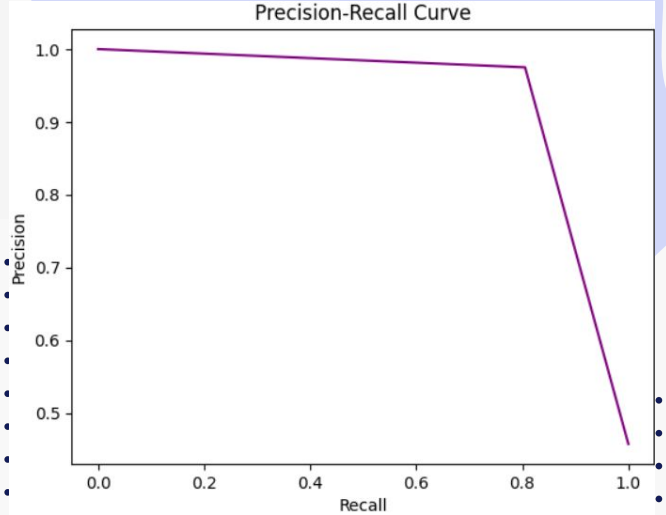
```python
# Calculating accuracy, precision, recall and f1-score
accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[0][1] + cm[1][0] + cm[1][1])
precision = cm[1][1] / (cm[1][1] + cm[0][1])    # Also called Positive Predictive Value
recall = cm[1][1] / (cm[1][1] + cm[1][0])    # Also called Sensitivity, Hit Rate or True Positive Rate
f1_score = 2 * ((precision * recall) / (precision + recall))
```

```python
# Printing the performance metrics
print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1-Score:', f1_score)
```

```
Accuracy: 0.9014195583596214
Precision: 0.9749478079331941
Recall: 0.8051724137931034
F1-Score: 0.881964117091596
```

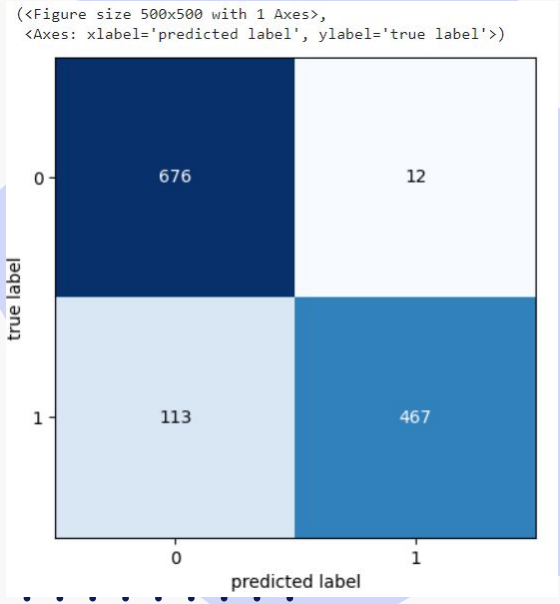
Precision Per Recall:

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)
#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()
```



Model Report :

	precision	recall	f1-score	support
0	0.86	0.98	0.92	688
1	0.97	0.81	0.88	580
accuracy			0.90	1268
macro avg	0.92	0.89	0.90	1268
weighted avg	0.91	0.90	0.90	1268



Second CNN Model :

```python
#building th full CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(150, 150,1)))
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
#model.add(Dense(2, activation='softmax'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
_________________________________
Total params: 1,742,593
Trainable params: 1,742,593
Non-trainable params: 0
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_28 (Conv2D)	(None, 150, 150, 32)	320
conv2d_29 (Conv2D)	(None, 150, 150, 32)	9248
conv2d_30 (Conv2D)	(None, 150, 150, 32)	9248
conv2d_31 (Conv2D)	(None, 150, 150, 32)	9248
max_pooling2d_14 (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_32 (Conv2D)	(None, 75, 75, 32)	9248
conv2d_33 (Conv2D)	(None, 75, 75, 32)	9248
conv2d_34 (Conv2D)	(None, 75, 75, 32)	9248
max_pooling2d_15 (MaxPooling2D)	(None, 37, 37, 32)	0
conv2d_35 (Conv2D)	(None, 37, 37, 64)	18496
conv2d_36 (Conv2D)	(None, 37, 37, 64)	36928
max_pooling2d_16 (MaxPooling2D)	(None, 18, 18, 64)	0
conv2d_37 (Conv2D)	(None, 18, 18, 128)	73856
conv2d_38 (Conv2D)	(None, 18, 18, 128)	147584
max_pooling2d_17 (MaxPooling2D)	(None, 9, 9, 128)	0
flatten_3 (Flatten)	(None, 10368)	0
dense_15 (Dense)	(None, 128)	1327232
dense_16 (Dense)	(None, 128)	16512
dense_17 (Dense)	(None, 128)	16512
dense_18 (Dense)	(None, 128)	16512
dense_19 (Dense)	(None, 128)	16512
dense_20 (Dense)	(None, 128)	16512
dense_21 (Dense)	(None, 1)	129

Training For 40 Epoch :

```
# compile model
#from keras.optimizers import adam_v2
#opt = adam_v2(lr=0.001)

early_stopping = tf.keras.callbacks.EarlyStopping(
    min_delta=0.0001,  # minimium amount of change to count as an improvement
    patience=10,  # how many epochs to wait before stopping
)
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

```
#history = model.fit(X_train_scaled, epochs=3,  validation_data=(X_val_scaled, y_val), callbacks=[early_stopping])
#history = cnn_model.fit(X_train_scaled, y_train, epochs=10 )
history = model.fit(X_train_scaled,y_train , epochs=40,  validation_data=(X_val_scaled, y_val), callbacks=[early_stopping])
```

Testing

Model

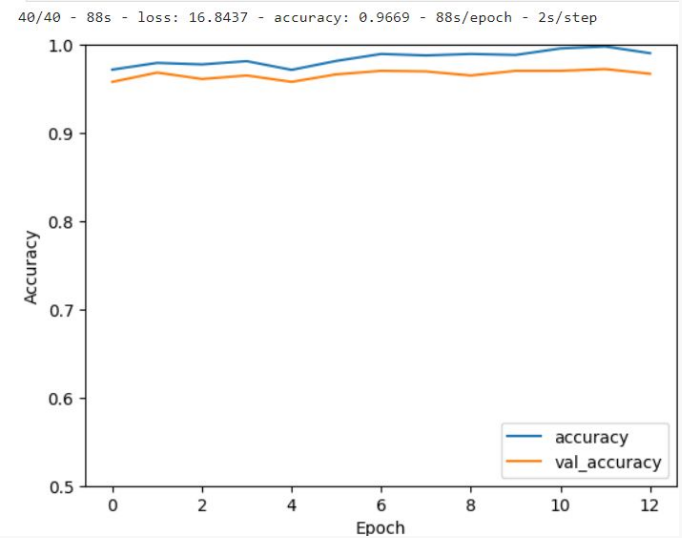
CNN

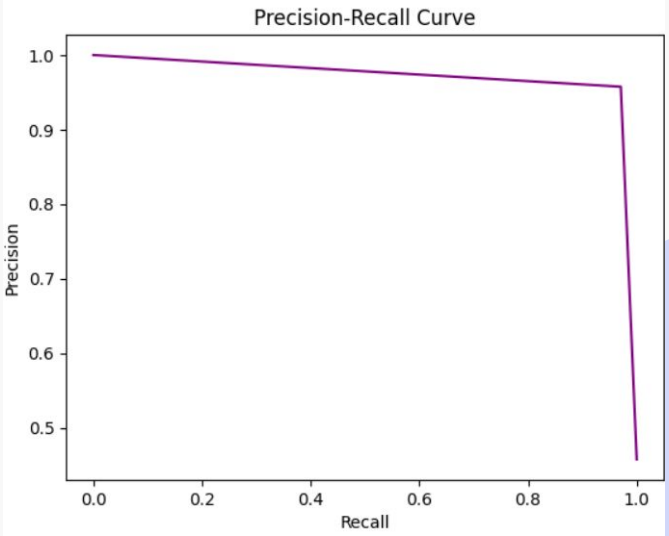
Prediction :

```
prediction = model.predict (x_test)
prediction= (prediction > 0.5)
prediction
```

```
40/40 [==============================] - 62s 2s/step
array([[ True],
       [ True],
       [False],
       ...,
       [False],
       [ True],
       [ True]])
```

Lose Curve & Precision Per Recall:



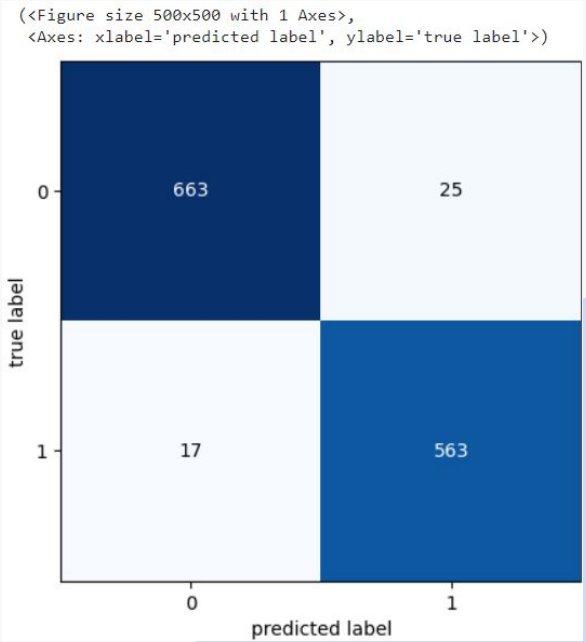


Model Report :

```
accuracy = accuracy_score(y_test, prediction)
print("Accuracy: %.f%%" % (accuracy*100))
print(classification_report(y_test, prediction))
```

Accuracy: 97%

	precision	recall	f1-score	support
0	0.97	0.96	0.97	688
1	0.96	0.97	0.96	580
accuracy			0.97	1268
macro avg	0.97	0.97	0.97	1268
weighted avg	0.97	0.97	0.97	1268



Conclusion

CNN

CNN-KNN

CNN on it's own gets better results than CNN-KNN model since KNN, output completely relies on nearest neighbors, which may or may not be good choice. Also it is sensitive to distance metrics. On the other hand, CNN extract the features from the input data. Which are very helpful for making analysis

Contrary to the research which is saying that CNN-KNN is better but I think that is because our dataset is much larger than the papers and we added more Dense layers to evaluate the model

THANKS!

Do you have any questions?

صفيه حمدي عبدالحميد
202000466

اميره اسامه محمد عبدالله
202000162

حبيبة ياسر سعيد محمد
202000262

زينب محمد متولي علي
202000366

صفاء محمود محمد
202000465

ريم اسماعيل حسن احمد
202000346