

Training of U-Net on Chest X-Rays to Segment Lungs and Detect Tuberculosis

S Siddhartha and S Sahana

School of Electronics and Communication Engineering
Vellore Institute of Technology, Chennai, Tamil Nadu, India

Abstract: Tuberculosis (TB) is caused by a bacterium (*Mycobacterium tuberculosis*), which primarily affects the lungs. Tuberculosis is indeed curable and preventable. Tuberculosis spreads through the air from person to person. When people with lung tuberculosis cough, sneeze or spit, the TB germs are expelled into the air. In order to get infected, a person only has to breathe a few of these bacteria. Roughly one-quarter of the world's population is infected with tuberculosis (TB), which means they have been infected by TB germs but are not (yet) unwell with the disease and cannot spread it. Therefore, taking this disease as our problem statement, we aimed to train a U-NET, a convoluted neural network specifically used for image recognition and tasks involving processing pixel data. This neural network was specifically developed for biomedical image segmentation. In this study, we propose a method to train a U-NET on datasets which include 26 thousand of healthy and TB-affected lung X-Ray images. We then process them to first segment the lungs separately from the X-Ray by removing the unwanted data present in the picture like background and background noise. We later use augmentation to add more data to the model. Proceeding further, test data and training data are formed to train the model to detect abnormalities in the given lung X-Ray by comparing them to the preset parameters of a healthy lung CXR.

Keywords: U-NET, Augmentation, CXR, Test and Training Data, Parameters

I. INTRODUCTION

Tuberculosis (TB) is caused by a bacterium called *Mycobacterium tuberculosis*. The bacteria usually attack the lungs (lower part of the lungs), but TB bacteria can attack any part of the body such as the kidney, spine, and brain. The word "tuberculosis" comes from a Latin word for "nodule" or something that sticks out.

If we're infected, develop symptoms and are contagious, we have active tuberculosis or tuberculosis disease (TB disease).

The three stages of TB are:

- Primary infection.
- Latent TB infection.
- Active TB disease.

This project addresses the strategy for training a U-NET on datasets containing healthy and TB-affected lungs. We next process them to isolate the lungs from the X-Ray by eliminating the unnecessary data from the image such as background and background noise. Later on, we will perform augmentation to add more data to the model. Following that, test and training data are developed in order to train the model to detect irregularities in the particular lung X-Ray by comparing them to the preset parameters of a healthy lung CXR. This method would be promising and feasible for a Detection system to locate tuberculosis nodules in an affected person's lungs.

1.1 Objectives

Basic Objective of this project is to segment lungs from the given chest X-Ray in the dataset. By enhancing the image using augmentations, we can add more data to the model and work with that data to create a training and testing Dataset. Thus by doing that we can count the total number of parameters to analyze a given chest X-Ray to predict its nature (whether is affected by TB or not). Later we can run the model over and over to increase the prediction

accuracy by increasing the epoch rate. By doing this process we point out the differences between a normal X-Ray and a tuberculosis X-Ray and point out those spots on a different color

1.2 Scope

This work can be further developed to improve the accuracy of the detection of tuberculosis using chest X-Ray. Augmentation is a huge sector with a huge set of modifications so there are a lot of permutations which a person can exploit to create a better algorithm and model for the detection of TB. Early warning and prediction about the disease can be an added result that would be beneficial for the surgeons as well as the pulmonologists.

1.3 Applications

The project can be used by pulmonologists and surgeons to arrive at a better prediction and improved accuracy of Tuberculosis Detection from a basic Chest X-Ray. With continued usage of the model and more inputs of patient data, this project will be able to give more promising results, with better accuracy. It also proves to be a cost-effective, efficient and easily utilisable solution

II. DESIGN / IMPLEMENTATION

2.1 Design Approach

A. Training data for Segmentation and Segmentation.

Image segmentation is a technique for breaking up a digital image into smaller groupings called image segments, which reduces the complexity of the image and makes each segment more easily processed or analyzed. Technically, segmentation is the process of giving labels to pixels in a picture in order to distinguish between objects, persons, or other significant aspects. Object detection is a frequent use of picture segmentation. It is usual practice to initially apply an image segmentation method to discover things of interest in the picture before processing the complete image. In this case, the patient’s lung is the object of interest, hence we are going to segment the lungs alone from the whole x-ray. The object detector may then work with a bounding box that the segmentation algorithm has previously established. By stopping the detector from analyzing the full picture, accuracy is increased and inference time is decreased.

Dataset	No. of CXR images & masks	Train set/fold	Validation set/fold	Test set/fold
Kaggle lung x-ray & masks dataset	704	451	112	141

Table 2.1.1.a: Details Of Training, Validation And Test Set For U-Net Segmentation Models

B. Augmentation.

This study applied an augmentation technique based on basic image manipulation. To expand the ROI dataset and increase the robustness of the model, all ROIs were segmented manually, despite the potential lack of objectivity; it was necessary to perform image data augmentation in case of overfitting during the training process.

It is reported that data augmentation can improve the classification accuracy of deep learning algorithms by augmenting the existing data rather than collecting new data. Data augmentation can significantly increase the diversity of data available for training models. Image augmentation is crucial when the dataset is imbalanced. In this study, the number of normal images was 3,500 which is 5 times larger than TB- infected images. Therefore, it was important to augment TB-infected images four times to make the database balance. Some of the deep learning frameworks have data augmentation facilities built-in with the algorithms, however, in this study, two different image augmentation techniques (rotation, and translation) were utilized to generate TB training images.

The rotation operation used for image augmentation can be done by rotating the images in the clockwise and counter clockwise directions. Image translation can be done by translating the image in either horizontal (width shift) or vertical direction (height shift) or in both directions. In this work, the original image was clockwise and counter clockwise rotated with an angle of 5 and 10 degrees and horizontally and vertically translated by 10% and 15%.

C. U-NET Model

The original U-net consists of a contracting path and an expanding path. The contracting path consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a ReLU and a 2x2 max pooling operation with stride 2 for down sampling. The expanding path consists of an up sampling of the feature map followed by a 2x2 convolution (“up-convolution”) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. Total 23 convolutional layers are used in the network

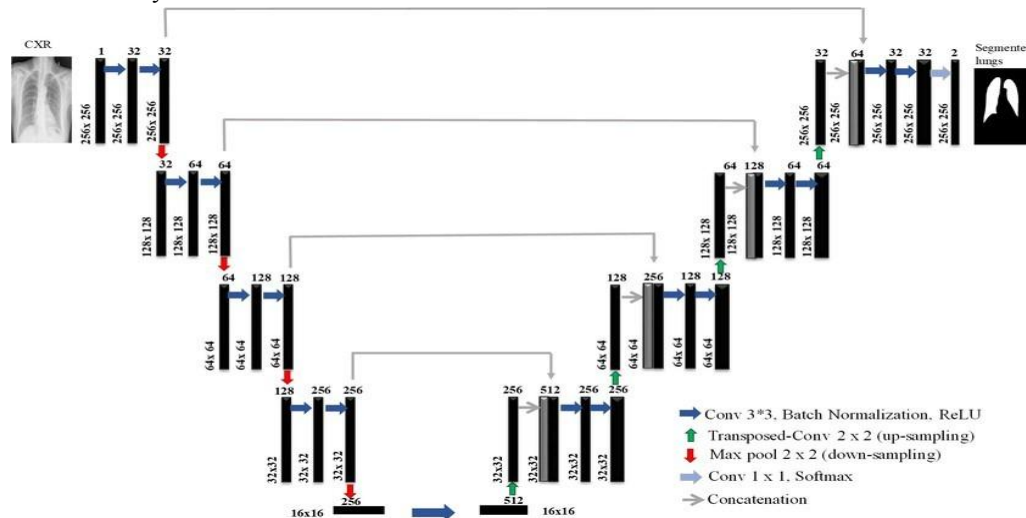


Fig 2.1.3.a: U-Net model architecture for lung segmentation

D. Block Diagram

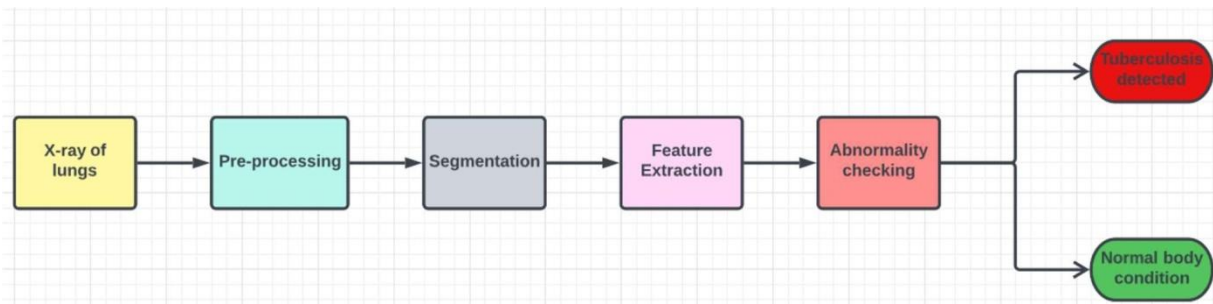


Fig 2.1.4.a: Block Diagram

2.2 Proposed System

Computer-assisted automated diagnostic tools may be more dependable in clinical applications if the accuracy of TB diagnosis from chest radiographs can be improved through the use of a robust and adaptable technique. Using other deep learning algorithms, altering the already effective techniques, or merging many effective algorithms into an ensemble model can all increase classification accuracy. Typically, CNN was used for entire X-ray images to identify lung diseases. Even though TB only manifests in the lung region, the X-ray scans show the lungs as well as other parts of the thorax. As a result, concentrating on the lung area of the X-ray pictures during training and classification may greatly enhance the performance of TB detection. This paper focuses on the detection of TB using the transfer learning-based technique of CNNs on the original and segmented lungs in X-ray images. CNN-based visualization techniques are also

2.3 Overview of Software

Python is considered for the implementation of tuberculosis detection in Chest X-Ray because of its varied applications and toolbox specifications. Python also has various built-in functions for technical computing, graphics, and animations. This high-performing language is thus considered for the analysis of the project. Tuberculosis prediction plays a vital role in medical sciences as this determines the state of the human lung. Chest X-Rays of both normal and affected patients are considered as the required inputs and the region with respective error is given as the output by the system. The mentioned procedures are then performed and the final output is observed through plots and models available in the Visual Studio Code environment.

2.4 Software Requirement

The library functions and the required inbuilt methods required for this analysis is mentioned below under the respective titles.

A. ToolBox Details.

```
import numpy as np # linear algebra
import tensorflow as tf # for tensorflow based registration
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from skimage.util import montage as montage2d
import os
from cv2 import imread, createCLAHE # read and equalize images
import cv2
from glob import glob
%matplotlib inline
import matplotlib.pyplot as plt
```

✓ 0.3s

Fig 2.4.1.a: Toolbox Details

TensorFlow

TensorFlow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. TensorFlow was originally developed for large numerical computations without keeping deep learning in mind

NumPy.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Skimage

Scikit-image is an open-source image processing library for the Python programming language. It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more.

Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK

Pandas

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series

B. Built-in Method Details

S.NO	FUNCTIONS	PYTHON CODE	REMARKS
1	montage	from skimage.util import montage as montage2d	displays all frames of a multiframe image array I . By default, the montage function arranges the images so that they roughly form a square.
2	imread_raw	from skimage.io import imread as imread_raw	reads the image from the file specified by filename , inferring the format of the file from its contents.
3	resize	from skimage.transform import resize	Resizes the image to a common size to maintain uniformity
4	np.stack	img_vol = np.stack(img_vol,0) seg_vol = np.stack(seg_vol,0)	The stack() function is used to join a sequence of arrays along a new axis.
5	glob	cxr_paths = glob(os.path.join('.', 'Montgomery', 'MontgomerySet', **', '*.png'))	Glob is used to return all file paths that match a specific pattern.
6	imshow	ax_mask.imshow(m_img[:, :, 0 , cmap = 'bone')	imshow(BW) displays the binary image BW in a figure
7	concatenate	from keras.layers import Conv2D, Activation, Input, UpSampling2D, concatenate, BatchNormalization	obtaining a new string that contains both of the original strings. In Python, there are a few ways to concatenate or combine strings.
8	binary_crossentropy	from keras.losses import binary_crossentropy	Used as a loss function for binary classification model. The binary_crossentropy function computes the cross-entropy loss between true labels and predicted labels
9	train_test_split	from sklearn.model_selection import train_test_split	The train_test_split() method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y).
10	gen_augmented_pairs	def gen_augmented_pairs(in_vol, in_seg, batch_size = 16):	Creates augmentation and adds more data into the picture to continue image processing
11	label2rgb	def label2rgb(in_vol, in_seg, batch_size = 16):	Used to convert images into RGB colored images

Fig 2.4.2.a: Inbuilt Command Analysis

III. RESULT AND ANALYSIS / TESTING

3.1 Visual Studio Code Python Implementation

```

Overview

Here we use the montgomery dataset for Tuberculosis

1. Organize the Training Data for Segmentation
2. Build Augmentation Pipeline and Generators
3. Build the U-Net Model
4. Train the Model
5. Adapt model for full images
6. Apply to RSNA Data

! pip install tensorflow

1 ! pip install scikit-image

import numpy as np # linear algebra
import tensorflow as tf # for tensorflow based registration
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from skimage.util import montage as montage2d
import os
from cv2 import imread, createCLAHE # read and equalize images
import cv2
from glob import glob
%matplotlib inline
import matplotlib.pyplot as plt

cxr_paths = glob(os.path.join('.', 'Montgomery', 'MontgomerySet', '**', '*.png'))
cxr_images = [(c_path,
               [os.path.join('/', .join(c_path.split('/')[:-2]), 'Montgomery', 'MontgomerySet', 'ManualMask', 'leftMask', os.path.basename(c_path)),
                os.path.join('/', .join(c_path.split('/')[:-2]), 'Montgomery', 'MontgomerySet', 'ManualMask', 'rightMask', os.path.basename(c_path))]
               ) for c_path in cxr_paths]
print('CXR Images', len(cxr_paths), cxr_paths[0])
print(cxr_images[0])

CXR Images 138 .\Montgomery\MontgomerySet\CXR_png\MUCUCXR_0001_0.png
('.\Montgomery\MontgomerySet\CXR_png\MUCUCXR_0001_0.png', ['Montgomery\MontgomerySet\ManualMask\leftMask\MUCUCXR_0001_0.png',
'Montgomery\MontgomerySet\ManualMask\righMask\MUCUCXR_0001_0.png'])

cxr_images

from skimage.io import imread as imread_raw
from skimage.transform import resize
import warnings
from tqdm import tqdm
warnings.filterwarnings('ignore', category=UserWarning, module='skimage')
OUT_DIM = (512, 512)
def imread(in_path, apply_clahe = False):
    img_data = imread_raw(in_path)
    n_img = (255*resize(img_data, OUT_DIM, mode = 'constant')).clip(0,255).astype(np.uint8)
    if apply_clahe:
        clahe_tool = createCLAHE(clipLimit=2.0, tileGridSize=(16,16))
        n_img = clahe_tool.apply(n_img)
    return np.expand_dims(n_img, -1)
  
```

```

img_vol, seg_vol = [], []
for img_path, s_paths in tqdm(cxr_images):
    img_vol += [imread(img_path)]
    seg_vol += [np.max(np.stack([imread(s_path, apply_clahe = False) for s_path in s_paths],0),0)]
img_vol = np.stack(img_vol,0)
seg_vol = np.stack(seg_vol,0)
print('Images', img_vol.shape, 'Segmentations', seg_vol.shape)

```

[36] ✓ 9m 10.8s Python

... 27% | 37/138 [02:40<06:50, 4.07s/it]

Images (138, 512, 512, 1) Segmentations (138, 512, 512, 1)

```

np.random.seed(2018)
t_img, m_img = img_vol[0], seg_vol[0]

fig, (ax_img, ax_mask) = plt.subplots(1,2, figsize = (12, 6))
ax_img.imshow(np.clip(255*t_img, 0, 255).astype(np.uint8) if t_img.shape[2]==3 else t_img[:, :,0],
               interpolation = 'none', cmap = 'bone')
ax_mask.imshow(m_img[:, :,0], cmap = 'bone')

```

[37] ✓ 0.4s Python

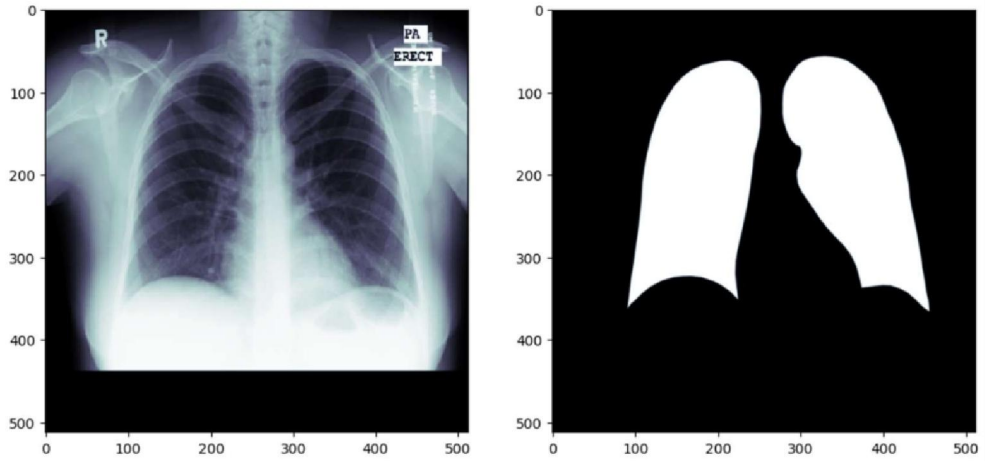
```

fig, (ax_img, ax_mask) = plt.subplots(1,2, figsize = (12, 6))
ax_img.imshow(np.clip(255*t_img, 0, 255).astype(np.uint8) if t_img.shape[2]==3 else t_img[:, :,0],
               interpolation = 'none', cmap = 'bone')
ax_mask.imshow(m_img[:, :,0], cmap = 'bone')

```

[37] ✓ 0.4s Python

... <matplotlib.image.AxesImage at 0x2892bfc75e0>



Make a Simple Model

Here we make a simple U-Net to create the lung segmentations

```

from keras.layers import Conv2D, Activation, Input, UpSampling2D, concatenate, BatchNormalization
from keras.layers import LeakyReLU
from keras.initializers import RandomNormal

def c2(x_in, nf, strides=1):
    x_out = Conv2D(nf, kernel_size=3, padding='same',
                  kernel_initializer='he_normal', strides=strides)(x_in)
    x_out = LeakyReLU(0.2)(x_out)
    return x_out

def unet_enc(vol_size, enc_nf, pre_filter = 8):
    src = Input(shape=vol_size + (1,), name = 'EncoderInput')
    # down-sample path.
    x_in = BatchNormalization(name = 'NormalizeInput')(src)
    x_in = c2(x_in, pre_filter, 1)
    x0 = c2(x_in, enc_nf[0], 2)
    x1 = c2(x0, enc_nf[1], 2)
    x2 = c2(x1, enc_nf[2], 2)
    x3 = c2(x2, enc_nf[3], 2)
    return Model(inputs = [src],
                 outputs = [x_in, x0, x1, x2, x3],
                 name = 'UnetEncoder')

```

[38] ✓ 0.3s Python

```

from keras.models import Model
from keras import layers
def unet(vol_size, enc_nf, dec_nf, full_size=True, edge_crop=48):
    """
    unet network for voxelmorph
    Args:
        vol_size: volume size. e.g. (256, 256, 256)
        enc_nf: encoder filters. right now it needs to be to 1x4.
            e.g. [16,32,32,32]
            TODO: make this flexible.
        dec_nf: decoder filters. right now it's forced to be 1x7.
            e.g. [32,32,32,32,8,8,3]
            TODO: make this flexible.
        full_size
    """

    # inputs
    raw_src = Input(shape=vol_size + (1,), name = 'ImageInput')
    src = layers.GaussianNoise(0.25)(raw_src)
    enc_model = unet_enc(vol_size, enc_nf)
    # run the same encoder on the source and the target and concatenate the output at each level
    x_in, x0, x1, x2, x3 = [s_enc for s_enc in enc_model(src)]

    x = c2(x3, dec_nf[0])
    x = UpSampling2D()(x)
    x = concatenate([x, x2])
    x = c2(x, dec_nf[1])
    x = UpSampling2D()(x)
    x = concatenate([x, x1])
    x = c2(x, dec_nf[2])
    x = UpSampling2D()(x)
    x = concatenate([x, x0])
    x = c2(x, dec_nf[3])
    x = c2(x, dec_nf[4])
    x = UpSampling2D()(x)
    x = concatenate([x, x_in])
    x = c2(x, dec_nf[5])

    # transform the results into a flow.
    y_seg = Conv2D(1, kernel_size=3, padding='same', name='lungs', activation='sigmoid')(x)
    y_seg = layers.Cropping2D((edge_crop, edge_crop))(y_seg)
    y_seg = layers.ZeroPadding2D((edge_crop, edge_crop))(y_seg)
    # prepare model
    model = Model(inputs=[raw_src], outputs=[y_seg])
    return model

```

✓ 0.4s Python

```

# use the predefined depths
enc_nf=[16,32,32,32]
nf_dec=[32,32,32,32,16,16,2]
net = unet(OUT_DIM, nf_enc, nf_dec)
# ensure the model roughly works
a= net.predict([np.zeros((1,)+OUT_DIM+(1,))])
print(a.shape)
net.summary()

```

✓ 0.5s Python

Layer (type)	Output Shape	Param #	Connected to
ImageInput (InputLayer)	[(None, 512, 512, 1)]	0	[]
gaussian_noise_1 (GaussianNoise)	(None, 512, 512, 1)	0	['ImageInput[0][0]']
UnetEncoder (Functional)	[(None, 512, 512, 8) , (None, 256, 256, 1) , (None, 128, 128, 3) , (None, 64, 64, 32) , (None, 32, 32, 32))]	24388	['gaussian_noise_1[0][0]']
conv2d_16 (Conv2D)	(None, 32, 32, 32)	9248	['UnetEncoder[0][4]']
leaky_re_lu_16 (LeakyReLU)	(None, 32, 32, 32)	0	['conv2d_16[0][0]']
...			
Total params: 99,589			
Trainable params: 99,587			
Non-trainable params: 2			

```

from keras.optimizers import Adam
import keras.backend as K
from keras.optimizers import Adam
from keras.losses import binary_crossentropy

reg_param = 1.0
lr = 2e-4
dice_bce_param = 0.0
use_dice = True

def dice_coef(y_true, y_pred, smooth=1):
    intersection = K.sum(y_true * y_pred, axis=[1,2,3])
    union = K.sum(y_true, axis=[1,2,3]) + K.sum(y_pred, axis=[1,2,3])
    return K.mean( (2. * intersection + smooth) / (union + smooth), axis=0)
def dice_p_bce(in_gt, in_pred):
    return dice_bce_param*binary_crossentropy(in_gt, in_pred) - dice_coef(in_gt, in_pred)
def true_positive_rate(y_true, y_pred):
    return K.sum(K.flatten(y_true)*K.flatten(K.round(y_pred)))/K.sum(y_true)

net.compile(optimizer=Adam(lr=lr),
            loss=[dice_p_bce],
            metrics = [true_positive_rate, 'binary_accuracy'])

```

✓ 0.4s Python

c:\Users\siddhu\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.

Here we make a tool to generate training data from the X-ray scans

```

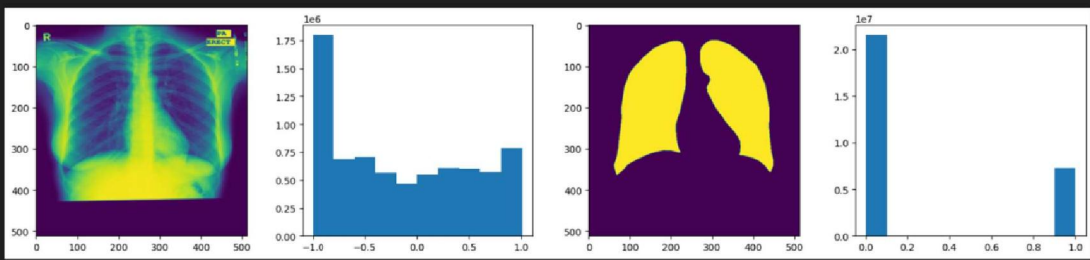
from sklearn.model_selection import train_test_split
in_vol, test_vol, train_seg, test_seg = train_test_split((img_vol-127.0)/127.0,
                                                         (seg_vol>127).astype(np.float32),
                                                         test_size = 0.2,
                                                         random_state = 2018)

print('Train', train_vol.shape, 'Test', test_vol.shape, test_vol.mean(), test_vol.max())
print('Seg', train_seg.shape, train_seg.max(), np.unique(train_seg.ravel()))
fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize = (20, 4))
ax1.imshow(test_vol[0, :, :, 0])
ax2.hist(test_vol.ravel())
ax3.imshow(test_seg[0, :, :, 0]>0.5)
ax4.hist(train_seg.ravel());

```

✓ 2.1s Python

Train (110, 512, 512, 1) Test (28, 512, 512, 1) -0.1553112594295539 1.0078740157488315
Seg (110, 512, 512, 1) 1.0 [0. 1.]



Adding Augmentation

Here we use augmentation to get more data into the model

```

from keras.preprocessing.image import ImageDataGenerator
dg_args = dict(featurewise_center = False,
               samplewise_center = False,
               rotation_range = 5,
               width_shift_range = 0.05,
               height_shift_range = 0.05,
               shear_range = 0.01,
               zoom_range = [0.8, 1.2],
               # anatomically it doesnt make sense, but many images are flipped
               horizontal_flip = True,
               vertical_flip = False,
               fill_mode = 'nearest',
               data_format = 'channels_last')

image_gen = ImageDataGenerator(**dg_args)

def gen_augmented_pairs(in_vol, in_seg, batch_size = 16):
    while True:
        seed = np.random.choice(range(9999))
        # keep the seeds synchronized otherwise the augmentation to the images is different from the masks
        g_vol = image_gen.flow(in_vol, batch_size = batch_size, seed = seed)
        g_seg = image_gen.flow(in_seg, batch_size = batch_size, seed = seed)
        for i_vol, i_seg in zip(g_vol, g_seg):
            yield i_vol, i_seg

```

✓ 0.3s Python

```

train_gen = gen_augmented_pairs(train_vol, train_seg, batch_size = 16)
test_gen = gen_augmented_pairs(test_vol, test_seg, batch_size = 16)
train_X, train_Y = next(train_gen)
test_X, test_Y = next(test_gen)
print(train_X.shape, train_Y.shape)
print(test_X.shape, test_Y.shape)

```

✓ 1.2s Python

(16, 512, 512, 1) (16, 512, 512, 1)
(16, 512, 512, 1) (16, 512, 512, 1)

Training Data

```

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (20, 10))
ax1.imshow(montage2d(train_X[:, :, :, 0]), cmap = 'bone')
ax1.set_title('CXR Image')
ax2.imshow(montage2d(train_Y[:, :, :, 0]), cmap = 'bone')
ax2.set_title('Seg Image')

```

✓ 1.6s Python

Text(0.5, 1.0, 'Seg Image')



```

from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping, ReduceLROnPlateau
weight_path="{_weights.best.hdf5".format('cxr_reg')

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min', save_weights_only = True)

reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                                    patience=3,
                                    verbose=1, mode='min', epsilon=0.0001, cooldown=2, min_lr=1e-6)

early = EarlyStopping(monitor="val_loss",
                      mode="min",
                      patience=15)
callbacks_list = [checkpoint, early, reduceLROnPlat]
    
```

✓ 0.5s Python

WARNING:tensorflow: `epsilon` argument is deprecated and will be removed, use `min_delta` instead.

```

from IPython.display import clear_output
loss_history = net.fit_generator(train_gen,
                                steps_per_epoch=len(train_vol)//train_X.shape[0],
                                epochs = 30,
                                validation_data = (test_vol, test_seg),
                                callbacks=callbacks_list
                                )
clear_output()
    
```

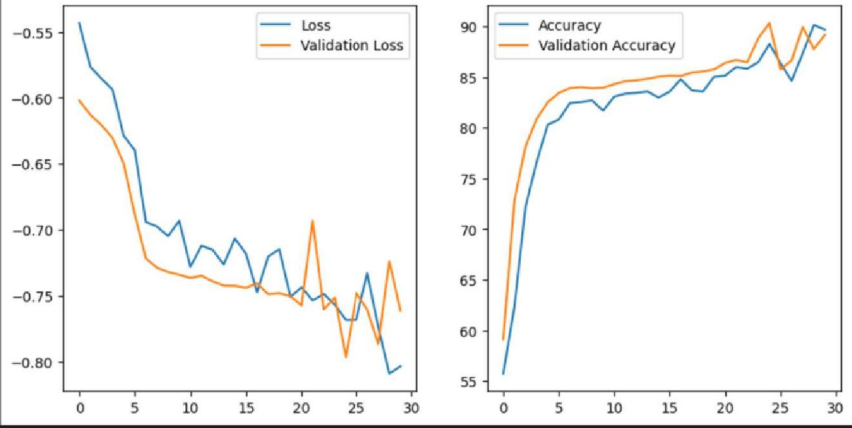
✓ 29m 47.3s Python

```

t.load_weights(weight_path)
net.save('full_model.h5')
    
```

✓ 2.6s Python

<matplotlib.legend.Legend at 0x2891aa7f940>



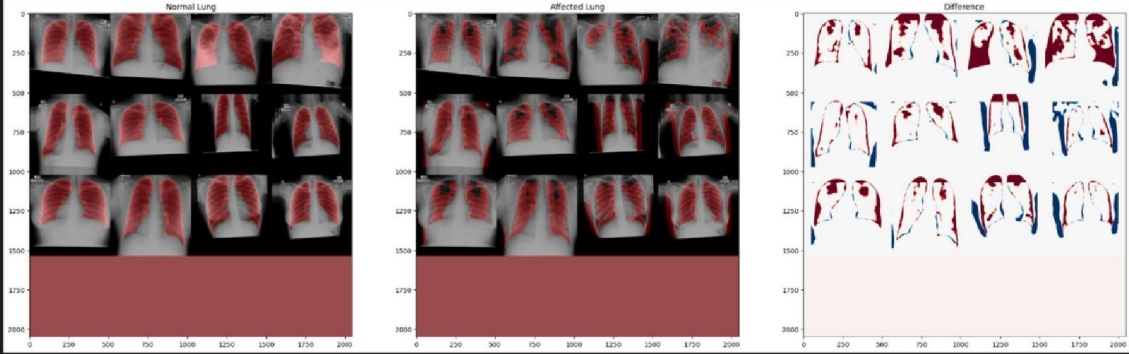
DIFFERENCE BETWEEN GROUND TRUTH (NORMAL LUNG) and PREDICTION (TB LUNG)

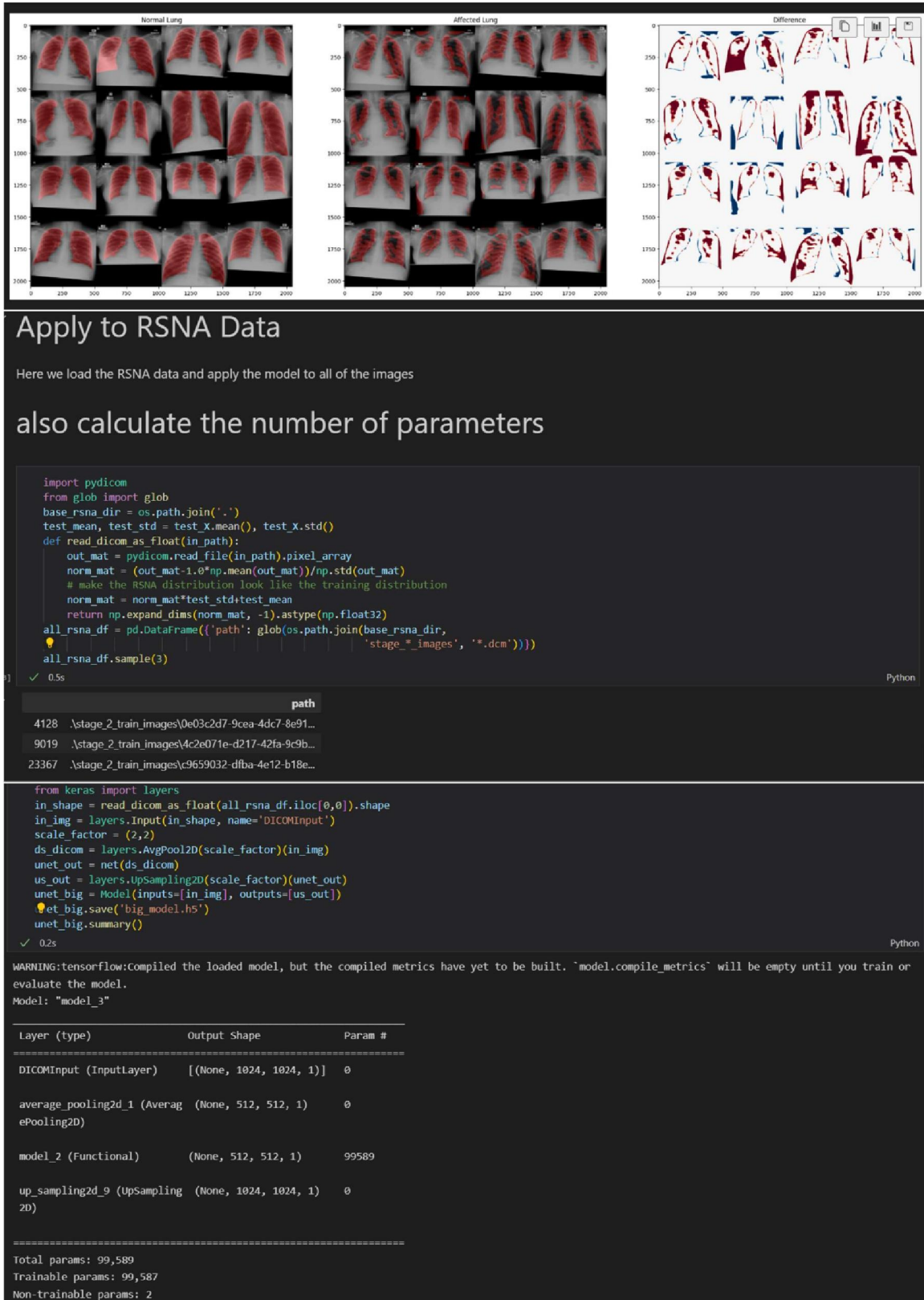
```

show_examples(2)
    
```

✓ 24.8s Python

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





Apply to RSNA Data

Here we load the RSNA data and apply the model to all of the images
also calculate the number of parameters

```

import pydicom
from glob import glob
base_rsna_dir = os.path.join('.')
test_mean, test_std = test_x.mean(), test_x.std()
def read_dicom_as_float(in_path):
    out_mat = pydicom.read_file(in_path).pixel_array
    norm_mat = (out_mat - 1.0 * np.mean(out_mat)) / np.std(out_mat)
    # make the RSNA distribution look like the training distribution
    norm_mat = norm_mat * test_std + test_mean
    return np.expand_dims(norm_mat, -1).astype(np.float32)
all_rsna_df = pd.DataFrame({'path': glob(os.path.join(base_rsna_dir,
                                                    'stage_*_images', '*.dcm'))})
all_rsna_df.sample(3)
    
```

```

from keras import layers
in_shape = read_dicom_as_float(all_rsna_df.iloc[0,0]).shape
in_img = layers.Input(in_shape, name='DICOMInput')
scale_factor = (2,2)
ds_dicom = layers.AvgPool2D(scale_factor)(in_img)
UNET_out = net(ds_dicom)
us_out = layers.UpSampling2D(scale_factor)(UNET_out)
UNET_big = Model(inputs=[in_img], outputs=[us_out])
UNET_big.save('big_model.h5')
UNET_big.summary()
    
```

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
Model: "model_3"

Layer (type)	Output Shape	Param #
DICOMInput (InputLayer)	[(None, 1024, 1024, 1)]	0
average_pooling2d_1 (Average Pooling2D)	(None, 512, 512, 1)	0
model_2 (Functional)	(None, 512, 512, 1)	99589
up_sampling2d_9 (UpSampling2D)	(None, 1024, 1024, 1)	0

Total params: 99,589
Trainable params: 99,587
Non-trainable params: 2

Fig.3.1.1a Running of Python Code in VSCode

3.2 MATLAB Output Screenshots

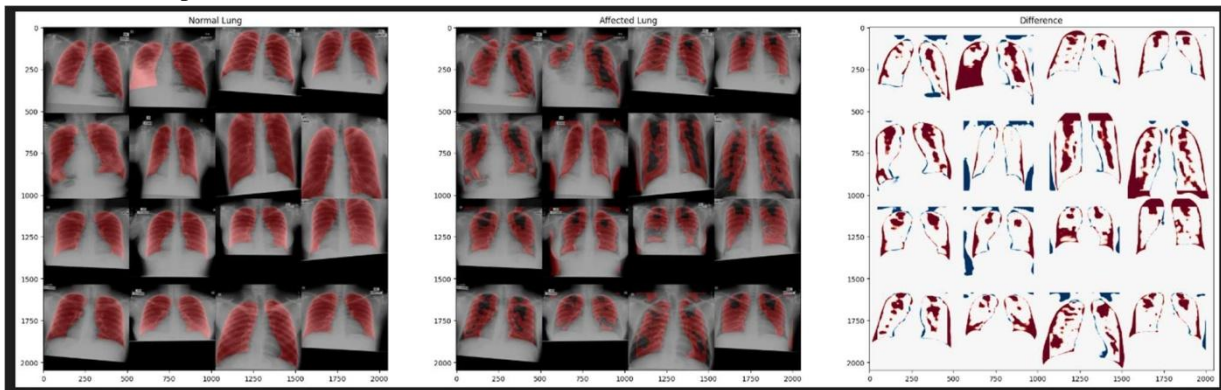


Fig 3.2.a: Comparison of a Normal CXR and a TB lung CX

IV. CONCLUSION AND FUTURE ENHANCEMENT

4.1 Conclusion

This work presents a transfer learning approach with deep Convolutional Neural Networks for the automatic detection of tuberculosis from the chest radiographs. The performance of the U-Net model was evaluated for the classification of TB and normal CXR images. The classification accuracy, precision and recall for the detection of TB were found to be 87.07% without segmentation 96.9% with segmentation respectively. It was also shown that image segmentation can significantly improve classification accuracy. The visualization output confirms that lung segmentation helps in taking decisions from the lung region unlike the original x-rays where decision can be taken based on features outside the lung region. Therefore, segmentation of lungs is very crucial for computer aided diagnosis using radiographs. This state-of-the-art performance can be a very useful and fast diagnostic tool, which can save significant number of people who died every year due to delayed or improper diagnosis.

4.2 Future Scope

To remove unwanted noise from an image, median filtering technique can be done at the starting stage. For the next stage we can combine two segmentation methods like watershed model and gray level thresholding model, and a fused image is generated which yields a highly accurate result. Features like area, major axis, minor axis, eccentricity, mean, standard deviation, skewness, kurtosis are extracted from ROI of fused image. This is further classified using KNN, SMO and Simple linear regression classifiers. The efficiency of classifiers shows that watershed segmentation and gray level threshold with KNN produces better result with an efficiency of 98% for detecting tuberculosis in lung image. In future, various feature extraction/feature selection methods can be applied for tuberculosis segmentation classification.

REFERENCES

- [1]. Sudre, Philippe, G. Ten Dam, and Arata Kochi. "Tuberculosis: a global overview of the situation today." *Bulletin of the World Health Organization* 70.2 (1992): 149.
- [2]. Sumartojo, Esther. "When tuberculosis treatment fails." *Am Rev Respir Dis* 147.1311 (1993): e20.
- [3]. Zhou, Y., et al. "Differentiation of sarcoidosis from tuberculosis using real-time PCR assay for the detection and quantification of *Mycobacterium tuberculosis*." *Sarcoidosis, Vasculitis, and Diffuse Lung Diseases: Official Journal of WASOG* 25.2 (2008): 93-99.
- [4]. Pereira Arias-Bouda, Lenka M., et al. "Development of antigen detection assay for diagnosis of tuberculosis using sputum samples." *Journal of clinical microbiology* 38.6 (2000): 2278-2283.
- [5]. Kamble, Mr PA, Mr VV Anagire, and Mr SN Chamtagoudar. "CXR tuberculosis detection using MATLAB image processing." *Computer* 4 (2016): 5.
- [6]. Mohan, Ramya, et al. "Automatic Detection of Tuberculosis Using VGG19 with Seagull- Algorithm." *Life* 12.11 (2022): 1848.

- [7]. Side, Syafruddin. "A susceptible-infected-recovered model and simulation for transmission of tuberculosis." *Advanced Science Letters* 21.2 (2015): 137-139.
- [8]. Rajakumar, M. P., et al. "Tuberculosis detection in chest X-ray using Mayfly-algorithm optimized dual-deep-learning features." *Journal of X-Ray Science and Technology Preprint* (2021): 1-14.
- [9]. Karki, M.; Kantipudi, K.; Yang, F.; Yu, H.; Wang, Y.X.J.; Yaniv, Z.; Jaeger, S. Generalization Challenges in Drug-Resistant Tuberculosis Detection from Chest X-rays. *Diagnostics* 2022, 12, 188.
- [10]. Fernandes et al., "A reliable framework for accurate brain image examination and treatment planning based on early diagnosis support for clinicians", *Neural Computing and Applications*.