LUNG CANCER NODULE DETECTION AND CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

A thesis submitted in the partial fulfillment of the requirements for the award of degree of

B. Tech

In

Electronics and Communication Engineering

By

Manoj Kumar (108114053)

Nagaraj Archak (108114060)

Vijay Ravi (108114104)

ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

TIRUCHIRAPPALLI – 620015

MAY 2018

BONAFIDE CERTIFICATE

This is to certify that the project titled **LUNG CANCER DETECTION AND CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS** is a bonafide record of the work done by

Manoj Kumar (108114053)
Nagaraj Archak (108114060)
Vijay Ravi (108114104)

In partial fulfillment of the requirements for the award of the degree of **Bachelors of Technology** in **Electronics and Communication Engineering** of the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHARAPPALLLI,** during the year 2017-2018.

DR. R. MALMATHANRAJ	**DR. G. LAKSHMI NARAYANAN**
Project Guide	Head of Department

Project Viva-voce held on ______________

INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We are grateful to the department of Electronics and Communication Engineering, NIT Trichy for providing us the opportunity to do **Lung Cancer Detection and Classification using Convolutional Neural Networks** as our final year project.

We wish to extend my heartiest thanks to **Dr. R. Malmathanraj,** Assistant Professor, NIT Trichy, for the stepwise guidance that he provided. His constant encouragement and faith in us enabled us to complete the project work efficiently in the stipulated time interval.

We would also like to express our gratitude to **Dr. G. Lakshmi Narayanan**, Head of the Department, Electronics and Communication Engineering for taking time amidst his work, to help us for successful completion of the project.

Our sincere Thanks to **Dr. Mini Shaji Thomas**, Director NIT Tiruchirapalli for having provided all facilities for carrying out this project work.

Last but not the least, our sincere thanks to one and all whom we have missed to acknowledge individually.

Manoj Kumar

Nagaraj Archak

Vijay Ravi

ABSTRACT

Cancer is one of the most common, and dangerous diseases nowadays. Although many measures have been adopted to combat cancer, there are a large number of deaths around the world due to this fatal illness. Lung cancer is one of the most common forms of cancer, and it is one of the deadlier ones, claiming around 1.59 million deaths around the world. The main issue with lung cancer is its detection. Usually patients diagnosed with lung cancer are already in the late stages of cancer invasion, and the chances for survival are very slim. The cost of diagnosis and treatment post diagnosis is also an enormous amount. This project aims to develop an intelligent computer aided diagnosis (CAD) system to help doctors detect lung cancer in its early stages itself. This increases the survival chances, and also reduces the diagnosis costs. The computer aided diagnosis system is based on deep learning using convolutional neural networks, and uses CT scans of the lungs taken from patients to provide a result. The computer aided diagnosis system also uses image preprocessing techniques such as segmentation to obtain regions where lung cancer nodules are present. These nodules have a chance to develop into fully fledged lung cancer tumors. It was seen that the neural network system was not only successful in identifying cancer, and nodule regions, but also operated with a high level of accuracy.

Keywords: Lung Cancer, Convolutional Neural Networks, Segmentation, Nodule

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Cancer is an internal cell malfunction, due to a mutation in the genetic code of the cell. Normal programmed cells follow an orderly path of growth, division and death. However, cancer cells, do not experience this death, and instead continuously grow and divide. This leads to a mass of abnormal cells that grows out of control, which leads to the formation of a tumor. Lung cancer occurs when a lung cell's gene mutation makes the cell unable to correct DNA damage and unable to commit suicide.

Lung cancer also known as lung carcinoma, is the uncontrolled growth of abnormal cells in one or both of the lungs. This growth is usually witnessed in the epithelial cells that line the bronchial tubes. These abnormal cells divide rapidly, and instead of forming new healthy lung tissue form tumors. As the tumors become larger, and more numerous, they undermine the lung's ability to provide the bloodstream with oxygen.

Lung cancer is classified into two main types, non-small cell lung cancer and small cell lung cancer. Non-Small Cell lung cancer accounts for 80% of all the lung cancer cases, while Small Cell lung cancer accounts for the remaining 20%. Common forms of Non-Small Cell lung cancer include Squamous Cell Carcinoma, Adenocarcinoma, and Bronchioalveolar Carcinoma. Small Cell lung cancer consists of small cells that multiply quickly, and form large tumors that travel throughout the body. Small cell lung cancer is predominately caused by smoking.

Lung cancer has been one of the most common forms of cancer in the world for several decades. According to GLOBOCAN 2012, 70,000 new cases of lung cancer have been diagnosed, and there is an estimated mortality rate of 64,000 in India in the year 2012 alone. Lung cancer constitutes 6.9% of all new cancer cases and 9.3% of all cancer related deaths in India.

Lung cancer detection and diagnosis using the current technology is done using the following test options. Imaging tests, which may use X-Ray technology to reveal an abnormal mass or nodule can be done, a CT scan can reveal small lesions in your lungs that might not be detected on an X-Ray. Sputum cytology, which involves laboratory analysis of sputum, may reveal the presence of lung cancer cells.
Biopsy, in which the suspected tissue cells are directly collected and analyzed is also done. The stage of cancer can be done with technology like CT scans, MRI scans, and Positron Emission Tomography (PET).

The main reason behind the large number of deaths lung cancer has caused is the late detection of the cancer. By the time the cancer has been detected, it is usually in its late stages, and at a point where the chances of survival are slim. While many forms of cancer have a recommended screening test designed for their detection, like for example the mammogram for breast cancer, PSA blood test for Prostate cancer, and colonoscopy for colon cancer, there is no recommended screening test for lung cancer. Most patients experience a long period of delay usually between 5-6 months between their first diagnostic test for lung cancer and a definitive diagnosis, and majority were diagnosed at advanced stages of disease. The average total health care costs in US Dollars for a single patient was approximately $2400/month to get the diagnosis and approximately $16,000/month after the diagnosis.

The aim of this project, Lung Cancer Nodule Detection and Classification using Deep Convolution Neural Networks attempts to resolve the late diagnosis, cost and human error in the detection and diagnosis of lung cancer. The spiral CT scan which provides a 3D image of the patients lungs is required. This spiral CT scan is then preprocessed using image preprocessing techniques such as resampling, normalization, zero centering, and segmentation. The preprocessed image is then used as the input for the CNN network. A CNN network has been used here, as it is the most successful Neural Network form for analyzing visual imagery. The project also advocates the use of CNN networks for CT scan analysis based on its accuracy and economical approach.

Chapter 2 of the thesis involves discussions regarding previous and current research trends in Lung Cancer diagnosis using Computational Medical Imaging processes. Chapter 3 contains in depth discussion regarding the image preprocessing techniques, and the design of the CNN network. Chapter 4, contains the results of the Neural Network, as well as slices of the Python code used, and images of the CT scans used to test the design. The final chapter concludes the project work done, and the thesis.

CHAPTER 2: LITERATURE REVIEW

Lung cancer is the most common forms of cancer in men and third most common form of cancer in women. In 2012, there were 1.8 million lung cancer cases worldwide, with 3.9% of those from India and the highest occurrences of 35.8% from China. Cancerous tumors can be accurately detected by radiologists, however due to high infrastructure costs, these facilities are usually not affordable by lower and middle classes of the society, thereby leading to delayed cancer diagnosis and lower survival rates. The survival rate of a lung cancer patient depends on whether cancer detected is at an early stage or at a later developed stage. Therefore to determine if the patient has early stage cancer or not, the system will have to detect a small nodule. This project proposes to build a computer aided system that uses convolutional neural network (CNN) architecture to facilitate classification of lung cancer into presence or absence of cancer and further estimate the position of cancer nodules present in the computed tomography scans.

2.1 Hopfield Neural Network and Fuzzy Clustering

The early detection of the lung cancer is a challenging problem, due to the structure of the cancer cells. The follow two segmentation methods, Hopfield Neural Network (HNN) and a Fuzzy C-Mean (FCM) clustering algorithm have been developed, for segmenting sputum color images to detect the lung cancer in its early stages. The manual analysis of the sputum samples is time consuming, inaccurate and requires intensive trained person to avoid diagnostic errors. The segmentation results will be used as a base for a Computer Aided Diagnosis (CAD) system for early detection of lung cancer which will improves the chances of survival for the patient.

2.1.1 Hopfield Neural Network

Hopfield Neural Network (HNN) is one of the artificial neural networks, which has been proposed for segmenting both gray-level and color images. The HNN is very sensitive to intensity variation and it can detect the overlapping cytoplasm classes. HNN is considered as unsupervised learning. The neural network structure consists of a grid of N x M neurons with each column representing a cluster and each row representing a pixel.

The algorithm could segment 97% of the images successfully in nuclei, cytoplasm regions and clear background.

2.1.2 Fuzzy Clustering

Clustering is the process of dividing the data into homogenous regions based on the similarity of objects; information that is logically similar is physically stored together, in order to increase the efficiency in the database system and to minimize the number of disk access. The most widely used algorithm is the Fuzzy C-Mean algorithm (FCM), it uses reciprocal distance to compute fuzzy weights. This algorithm has as input a predefined number of clusters, which is the k from its name. Means stands for an average location of all the members of particular cluster and the output is a partitioning of k cluster on a set of objects. The objective of the FCM cluster is to minimize the total weighted mean square error. The algorithm segments the images into nuclei, cytoplasm regions and clear background, however, the FCM is not sensitive to intensity variation, therefore, the cytoplasm regions are detected as one cluster when the cluster number is fixed to three, four, five and six. Moreover, FCM failed in detecting the nuclei; as it detected only part of it.

It was found that the HNN segmentation results are more accurate and reliable than FCM clustering in all cases. The HNN succeeded in extracting the nuclei and cytoplasm regions. However FCM failed in detecting the nuclei, instead it detected only part of it. In addition to that, the FCM is not sensitive to intensity variations as the segmentation error at convergence is larger with FCM compared to that with HNN.

2.2 Image Thresholding and AlexNet Architecture

Pulmonary nodules are round or oval, completely wrapped in the lung parenchyma. According to the size of nodules can be divided into small nodules and large nodules. CT is currently a good imaging method for detecting pulmonary nodules. CT images can often reach hundreds of layers. Numerous CT images and smaller nodules are prone to fatigue, thus lead to missed diagnosis. The computer-aided diagnosis system has a great advantage in solving the above problems.

2.2.1 Image Segmentation

Image thresholding segmentation is a traditional and most commonly used image segmentation method. It is the most basic and the most widely used segmentation technology because of its simple implementation, small amount of computation and stable performance. In many cases, it is a necessary image preprocessing process for image analysis, feature extraction and patter recognition. The purpose of the image thresholding is to divide the set of pixels according to the gray scale and each subset is formed to have an area corresponding to the real scene.

Image segmentation includes region-based method, edge-based method and so on. Region-based method is still widely used in lung CT images and also has good performance. Threshold method mainly includes iterative threshold method, maximum interclass variance method, and entropy method.

2.2.2 AlexNet CNN Architecture

There are eight layers in AlexNet, the first five layers are convolutional layers and the latter three layers are all connected layers. ReLU activation function and pooling are carried out after the basic convolution data are obtained. In order to reduce the number of weights needed to train, other optimization methods are used like weight sharing. Both AlexNet and ResNet are selected to train and test the data set in the experiment. AlexNet came out with a training result of 0.76, whereas ResNet came out with a training result of only 0.58.

2.3 U-Net Nodule Candidate Detection and Classification

The proposed CAD system starts with preprocessing the 3D CT scans using segmentation, normalization, downsampling, and zero-centering. The initial approach was to simply input the preprocessed 3D CT scans into 3D CNNs, but the results were poor. So an additional preprocessing was performed to input only regions of interests into the 3D CNNs. To identify regions of interest, a U-Net was trained for nodule candidate detection. Then input regions around nodule candidates detected by the U-Net was fed into 3D CNNs to ultimately classify the CT scans as positive or negative for lung cancer.

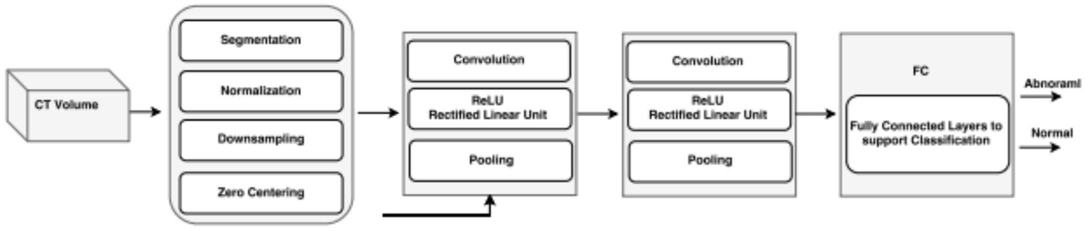


Figure 1: Input Regions around Nodules fed to CNN Architecture

2.3.1 Preprocessing and Segmentation

For each patient, pixel values were first converted in each image to Hounsfield units (HU), a measurement of radiodensity, and 2D slices are stacked into a single 3D image. Because tumors form on lung tissue, segmentation is used to mask out the bone, outside air, and other substances that would make data noisy, and leave only lung tissue information for the classifier. A number of segmentation approaches were tested, including thresholding, clustering (Kmeans and Meanshift), and Watershed. K-means and Meanshift allow very little supervision and did not produce good qualitative results. Watershed produced the best qualitative results, but took too long to run. Ultimately, thresholding was used. After segmentation, the 3D image is normalized by applying the linear scaling to squeezed all pixels of the original unsegmented image to values between 0 and 1. Spline interpolation downsamples each 3D image by a scale of 0.5 in each of the three dimensions. Finally, zero-centering is performed on data by subtracting the mean of all the images from the training set.

2.3.2 Nodule Candidate Detection

Feeding the entire segmented lungs into malignancy classifiers made results very poor. It was likely the case that the entire image was too large search space. Thus feeding smaller regions of interest instead of the entire segmented 3D image is more convenient. This was achieved by selecting small boxes containing top cancerous nodule candidates. To find these top nodule candidates, a modified version of the U-Net was trained. UNet is a 2D CNN architecture that is popular for biomedical image segmentation. During training, the modified U-Net takes as input 256 × 256 2D CT slices, and labels are provided (256 × 256 mask where nodule pixels are 1, rest are 0). The model is trained to output images of shape 256 × 256 were each pixels of the output has a value between 0 and 1 indicating the probability the pixel belongs to a nodule.

2.3.3 Convolutional Neural Network Classifier

U-Net generates more suspicious regions than actual nodules, the top 8 nodule candidates are located (32 × 32 × 32 volumes) by sliding a window over the data and saving the locations of the 8 most activated sectors. To prevent the top sectors from simply being clustered in the brightest region of the image, the 8 sectors were not permitted to overlap with each other. Then these sectors are combined into a single 64x64x64 image, which will serve as the input to classifiers, which assign a label to the image. A 3D CNN is used as linear classifier. It uses weighted softmax cross entropy loss (weight for a label is the inverse of the frequency of the label in the training set) and Adam Optimizer, and the CNNs use ReLU activation and droupout after each convolutional layer during training. Convolutional neural network consists of some number of convolutional layers, followed by one or more fully connected layers and finally an output layer. Convolutional layers preserve the spatial structure of the inputs, and as more layers are used, build up more and more complex representations of the input. The output of the convolutional layers is then used as input to a fully connected network layer. The activation function is chosen to be a Rectified Linear Unit (ReLU) with max(0, a). The last fully connected layer is used as input to the output layer. Accuracy of model is 86.6%, false classification rate is 13.4%, False positive rate is 11.9%, and False Negative is 14.7%. Almost all patients are classified correctly. Additionally, there is an enhancement on accuracy due to efficient U-Net architecture and segmentation.

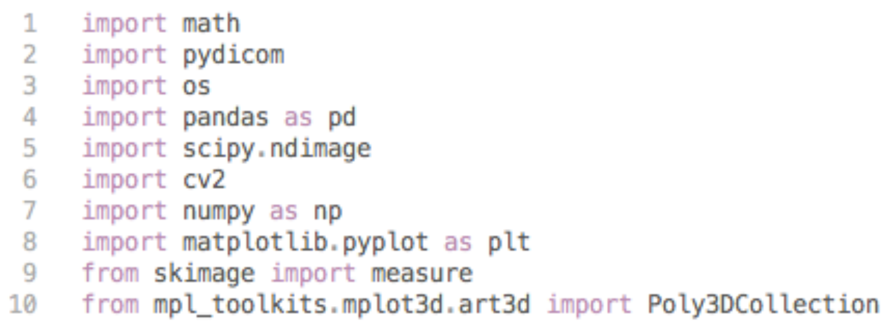
CHAPTER 3: CT SCAN IMAGE PREPROCESSING

3.1 LANGUAGE AND LIBRARIES USED

The language used for the design and architecture of the Convolutional Neural Network, Image Preprocessing and Cancer Nodule Detection was Python 2.7. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Python's simple syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports majority extensive support libraries a lot of which are utilized throughout the project. Python's high-level built in data structures, combined with dynamic typing and dynamic binding, make it very advantageous for image preprocessing, cancer nodule detection and CNN architecture development. Python has no compilation step, therefore the edit-test-debug cycle is incredibly fast. This helps in the debugging process of Python application development, as a bug or bad input will not cause a segmentation fault, instead when the interpreter discovers an error, an exception is raised. Python also consists of many extensive libraries for different applications. Before the concept of libraries can be understood with respect to Python programming, the terms Module, and Packages must be defined and understood. A module in python is .py file that defines one or more function or classes which the programmer intends to reuse in different sections of the program. A python package refers to the directory that contains a collection of Python modules. A library is therefore, a collection of precompiled routines that once imported, can be made use of by the program. The routines also called as modules are stored in object format. Therefore libraries provide certain functions that can be used directly in the application program without explicit coding by the programmer.

3.1.1 Python Dependencies

The libraries used for preprocessing CT scan images are math, pydicom, os, pandas, scipy.ndimage, cv2, numpy, matplotlib.pyplot, measure and morphology from skimage and Poly3DCollection from mpl_toolkits.mplot3d.art3d.



```python
import math
import pydicom
import os
import pandas as pd
import scipy.ndimage
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import measure
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
```

Figure 2: Python dependencies required for CT Scans Image Preprocessing

3.1.1.1 Pydicom

Pydicom is the Python library required for working with Digital Imaging and Communications in Medicine (DICOM) images files. DICOM is the standard for storing and transmitting medical images, including CT scans taken of the lungs. Therefore pydicom is used for parsing the DICOM format CT scans into natural pythonic data structures, on which typical python functions can be applied.

3.1.1.2 OS

OS library provides a portable way of using operating system dependent functionality. This library is mainly used to access the directory containing the test patient's CT scans, and concerned labels and parameters.

3.1.1.3 PANDAS

Python Data Analysis Library (PANDAS) is an open source, licensed library providing high-performance, easy to use data structures and data analysis tools.

3.1.1.4 Scipy.ndimage

Scipy.ndimage submodule is used primarily for all image processing tasks. The package contains various functions for multi-dimensional image processing. Image processing and analysis are generally seen as operations on two-dimensional arrays of values. However in medical images, the images are of higher dimensionality. The scipy.ndimage package provides a number of general image processing and analysis functions designed to operate with arrays of multiple dimensions.

3.1.1.5 OpenCV

OpenCV is a library written in python to solve computer vision problems. It makes use of numpy which is a highly optimized library for numerical operations in python. All the OpenCV array structures are converted to and from numpy arrays. This also makes it easier to integrate with other libraries that use numpy such as SciPy and Matplotlib.

3.1.1.6 Matplotlib.pyplot

Matplotlib.pyplot is used for plot generation, and to display the CT scans after the preprocessing methods are applied.

3.1.1.7 Measure from Skimage

Measure is used to label connected components of a binary image, using the dedicated skimage.measure.label function

3.1.1.8 Poly3DCollection from mpl_toolkits.mplot3d.art3d

Poly3DCollection is used to plot three dimensional images.

3.1.1.9 Numpy

Numpy, or Numerical Python is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using numpy, mathematical and logical operations on arrays can be performed.

3.1.2 Frameworks

TensorFlow is an open source software library released in 2015 by Google to make it easier for developers to design, build and train deep learning models. It does numerical computation using data-flow graphs. TensorFlow is cross-platform, it runs on nearly everything, GPUs and CPUs including mobile and embedded platforms. It provides toolkits like TensorBoard which is a suite of visualization tools that makes it easier to understand, debug, optimize and visualize TensorFlow programs.

3.2 CT SCANS AND KAGGLE DATA SCIENCE DATASET

3.2.1 Computed Tomography

A computed tomography (CT or CAT) scan allows doctors to see inside the body. It uses a combination of X-rays and a computer to create pictures of the organs, bones, and other tissues. It shows more detail than a regular X-ray. It uses a narrow X-ray beam that circles around one part of the body. This provides a series of images from many different angles. A computer uses this information to create a cross-sectional picture. This two-dimensional scan shows a “slice” of the inside of the body. This process is repeated to produce a number of slices. The computer stacks these scans one on top of the other to create a three-dimensional image.

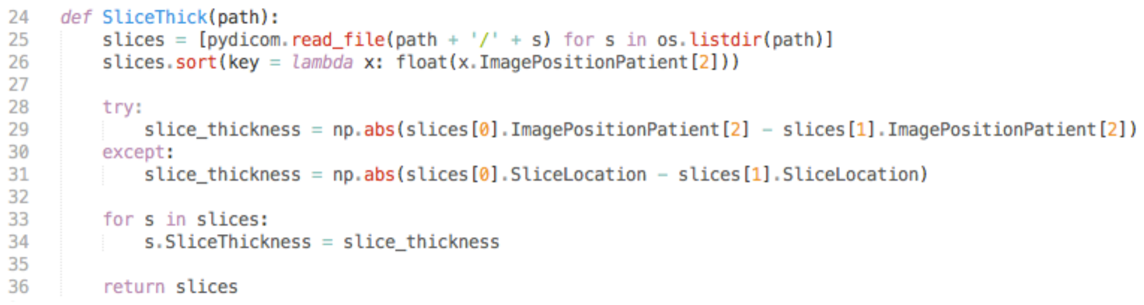
3.2.2 Kaggle Data Science Dataset

The neural network was trained and tested using the dataset available from the Kaggle Data Science Bowl. The dataset consists of 1595 low-dose CT images from high-risk patients in DICOM format. Each image contains a series with multiple axial slices of the chest cavity, where the number of slices vary from 180 to 250. The DICOM files have a header or metadata that contains the necessary information about the patient ID, as well as scan parameters such as the slice dimensions. A labels file for each patient is also provided which provides the actual result, whether the patient had cancer or not. Based on this dataset and the labels file the convolutional neural network will train itself.

3.3 CT SCANS IMAGE PREPROCESSING TECHNIQUES

3.3.1 Slice Thickness

First the DICOM CT Scans of each patient has to be loaded and read into the Python code. The DICOM files contain a lot of metadata, such as pixel size which is an indication of the size of the pixel in every dimension in reality. This pixel size/coarseness of the scan differs from scan to scan (e.g. the distance between slices may differ), which can hurt performance of CNN approaches. However this problem is later solved with isomorphic resampling. Below is the code snippet to load a scan, which consists of multiple slices, which is saved in a Python list. Every folder in the dataset consists of the complete scans of one patient. The pixel size in the Z direction, which is the slice thickness, is calculated, and added to the metadata.



```
def SliceThick(path):
    slices = [pydicom.read_file(path + '/' + s) for s in os.listdir(path)]
    slices.sort(key = lambda x: float(x.ImagePositionPatient[2]))

    try:
        slice_thickness = np.abs(slices[0].ImagePositionPatient[2] - slices[1].ImagePositionPatient[2])
    except:
        slice_thickness = np.abs(slices[0].SliceLocation - slices[1].SliceLocation)

    for s in slices:
        s.SliceThickness = slice_thickness

    return slices
```

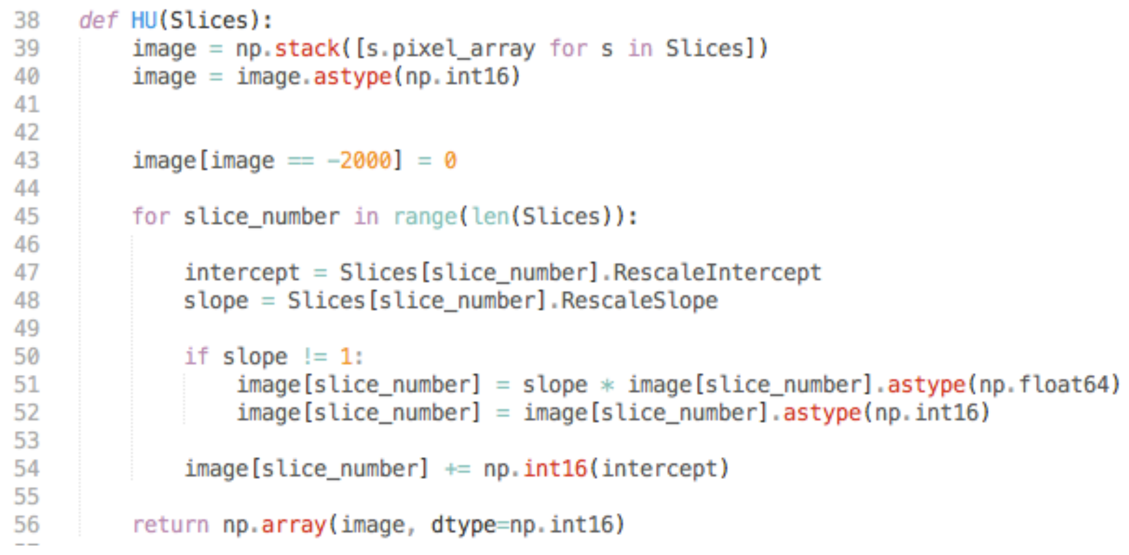
Figure 3: Calculating Slice Thickness

3.3.2 Conversion to Hounsfield Unit

The Hounsfield scale is defined as the attenuation value of the X-ray beam, minus the attenuation of water, divided by the attenuation of water, multiplied by 1000. It is also called CT Numbers and it is a quantitative scale for describing radio density. CT Scanners have cylindrical scanning bounds but the image that they output is a square. The pixels that fall outside this bound have a fixed -2000 value.

Substance	HU
Air	-1000
Lung	-500
Fat	-100 to -50
Water	0
CSF	15
Kidney	30
Blood	+30 to +45
Muscle	+10 to +40
Grey Matter	+37 to +45
White Matter	+20 to +30
Liver	+40 to +60
Soft Tissue, Contrast	+100 to +300
Bone	+700 (cancellous bone) to +3000(cortical bone)

Table 1: Hounsfield Unit Values

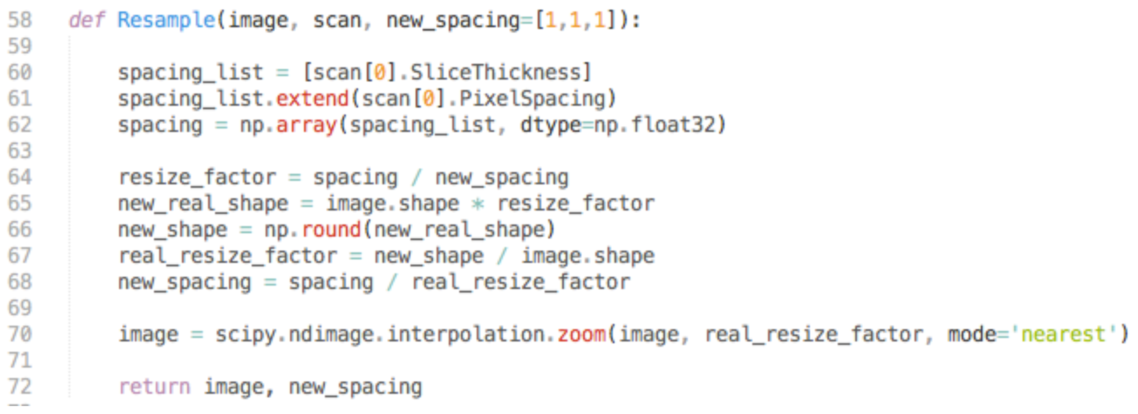


```python
def HU(Slices):
    image = np.stack([s.pixel_array for s in Slices])
    image = image.astype(np.int16)


    image[image == -2000] = 0

    for slice_number in range(len(Slices)):

        intercept = Slices[slice_number].RescaleIntercept
        slope = Slices[slice_number].RescaleSlope

        if slope != 1:
            image[slice_number] = slope * image[slice_number].astype(np.float64)
            image[slice_number] = image[slice_number].astype(np.int16)

        image[slice_number] += np.int16(intercept)

    return np.array(image, dtype=np.int16)
```

Figure 4: Converting to Hounsfield Units

3.3.3 Resampling

A scan may have a pixel spacing of [2.5, 0.5, 0.5], which means that the distance between slices is 2.5 millimeters. The second and third quantities represent the distance between the centers of adjacent pixels in the x and y direction respectively. For a different scan this may be [1.5, 0.725, 0.725], and this will be problematic for automatic analysis using neural networks. A common method of dealing with this is resampling the full dataset to a certain isotropic resolution. An isotropic resolution is an image in which the resolution is kept constant in all directions. Therefore the entire dataset is resampled to 1mmx1mmx1mm pixels so that automatic analysis using neural networks can be done without an issue.

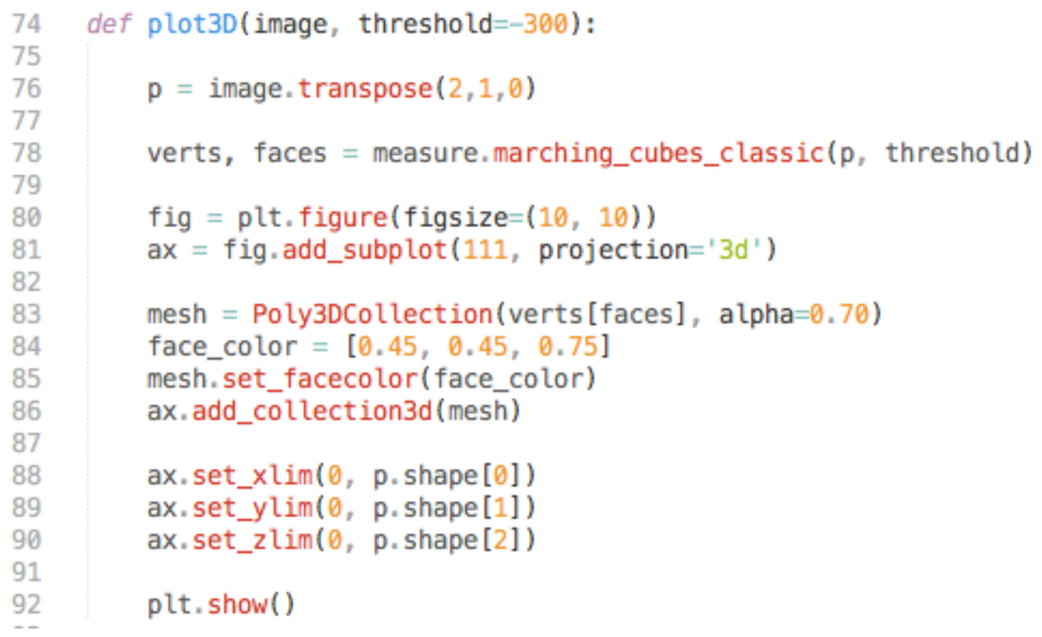


```python
def Resample(image, scan, new_spacing=[1,1,1]):

    spacing_list = [scan[0].SliceThickness]
    spacing_list.extend(scan[0].PixelSpacing)
    spacing = np.array(spacing_list, dtype=np.float32)

    resize_factor = spacing / new_spacing
    new_real_shape = image.shape * resize_factor
    new_shape = np.round(new_real_shape)
    real_resize_factor = new_shape / image.shape
    new_spacing = spacing / real_resize_factor

    image = scipy.ndimage.interpolation.zoom(image, real_resize_factor, mode='nearest')

    return image, new_spacing
```

Figure 5: Resampling the CT Scans

3.3.4 3D Plotting

All images as a part of the Kaggle Data Science Dataset are three dimensional scans of the lungs. Each 3D scan consists of multiple slices each of dimension 512 x 512 pixels. The following is a function written in python in order to visualize the images in three dimensions.

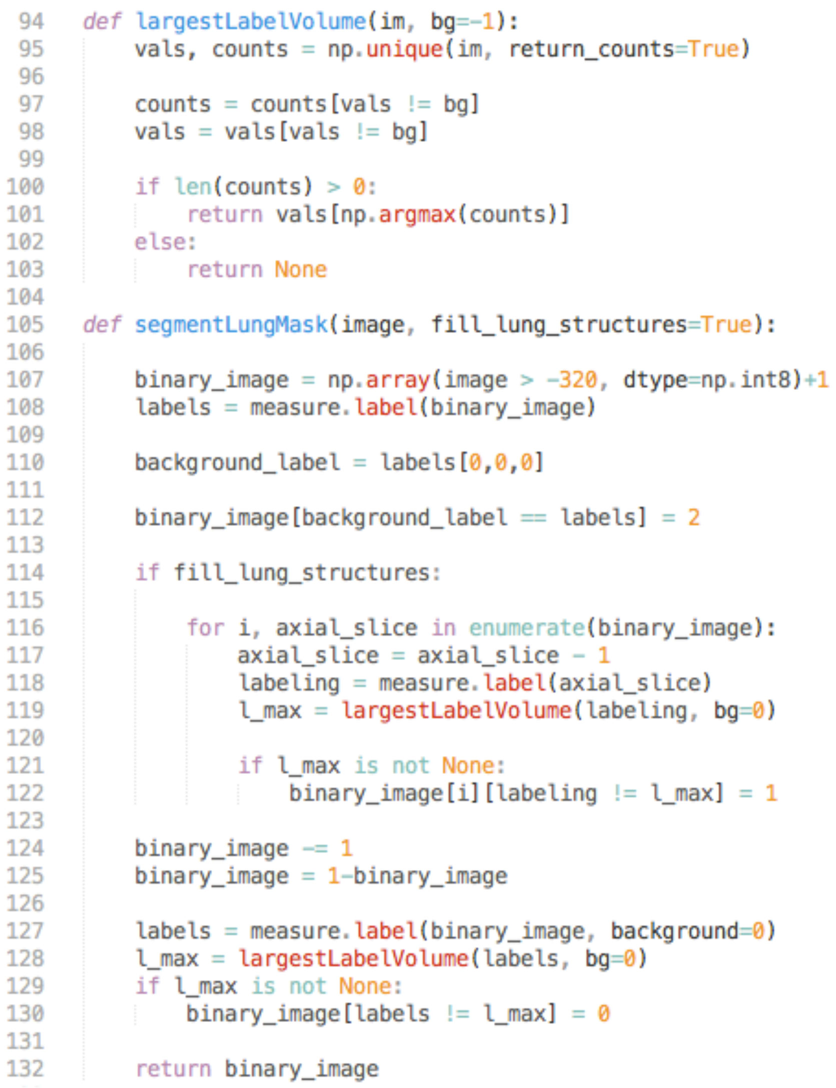


```
def plot3D(image, threshold=-300):

    p = image.transpose(2,1,0)

    verts, faces = measure.marching_cubes_classic(p, threshold)

    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')

    mesh = Poly3DCollection(verts[faces], alpha=0.70)
    face_color = [0.45, 0.45, 0.75]
    mesh.set_facecolor(face_color)
    ax.add_collection3d(mesh)

    ax.set_xlim(0, p.shape[0])
    ax.set_ylim(0, p.shape[1])
    ax.set_zlim(0, p.shape[2])

    plt.show()
```

Figure 6: 3D Plotting of CT Scans

3.3.5 Lung Segmentation

Lung segmentation is done in order to reduce the problem space and close in onto to the Region of Interest (ROI). It consists of applications of region growing and morphological operations and we have used connected component analysis for lung segmentation. The following are a few steps involved in lung segmentation such as thresholding the image at -320 HU. Applying connected components to determine label of air around person and fill it with 1s in the binary image. For every axial slice in the scan, largest solid connected component - the body and air around the person is determined, and set others to 0. This fills the structures in the lungs in the mask. Only the largest air pocket is kept since the human body has other pockets of air here and there. The following is the code snippet for lung segmentation.

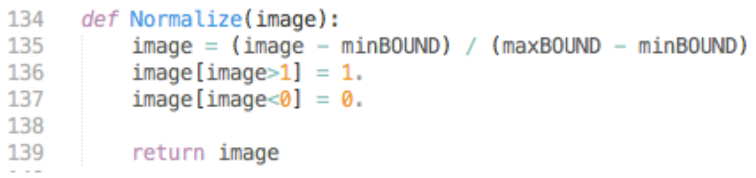


```python
def largestLabelVolume(im, bg=-1):
    vals, counts = np.unique(im, return_counts=True)

    counts = counts[vals != bg]
    vals = vals[vals != bg]

    if len(counts) > 0:
        return vals[np.argmax(counts)]
    else:
        return None

def segmentLungMask(image, fill_lung_structures=True):

    binary_image = np.array(image > -320, dtype=np.int8)+1
    labels = measure.label(binary_image)

    background_label = labels[0,0,0]

    binary_image[background_label == labels] = 2

    if fill_lung_structures:

        for i, axial_slice in enumerate(binary_image):
            axial_slice = axial_slice - 1
            labeling = measure.label(axial_slice)
            l_max = largestLabelVolume(labeling, bg=0)

            if l_max is not None:
                binary_image[i][labeling != l_max] = 1

    binary_image -= 1
    binary_image = 1-binary_image

    labels = measure.label(binary_image, background=0)
    l_max = largestLabelVolume(labels, bg=0)
    if l_max is not None:
        binary_image[labels != l_max] = 0

    return binary_image
```

Figure 7: Lung CT Scans Segmentation

3.3.6 Normalisation

Normalisation is a process that changes the range of pixel intensity values. The current pixel Hounsfield Units may range from -1000 to 2000. They will have to be normalized with a upper limit of 400 as anything above that is merely bones and unwanted noise.

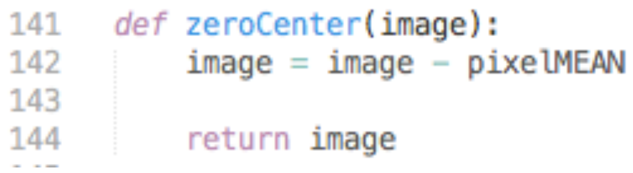


```python
def Normalize(image):
    image = (image - minBOUND) / (maxBOUND - minBOUND)
    image[image>1] = 1.
    image[image<0] = 0.

    return image
```

Figure 8: Normalising CT Scan Images

3.3.7 Zero Centering

In this preprocessing step, involves zero centering. It is advisory to zero center your data so that your mean value is 0. This is done by subtracting the mean pixel value from all pixels. Zero centering makes the learning process of the neural network much more easier.

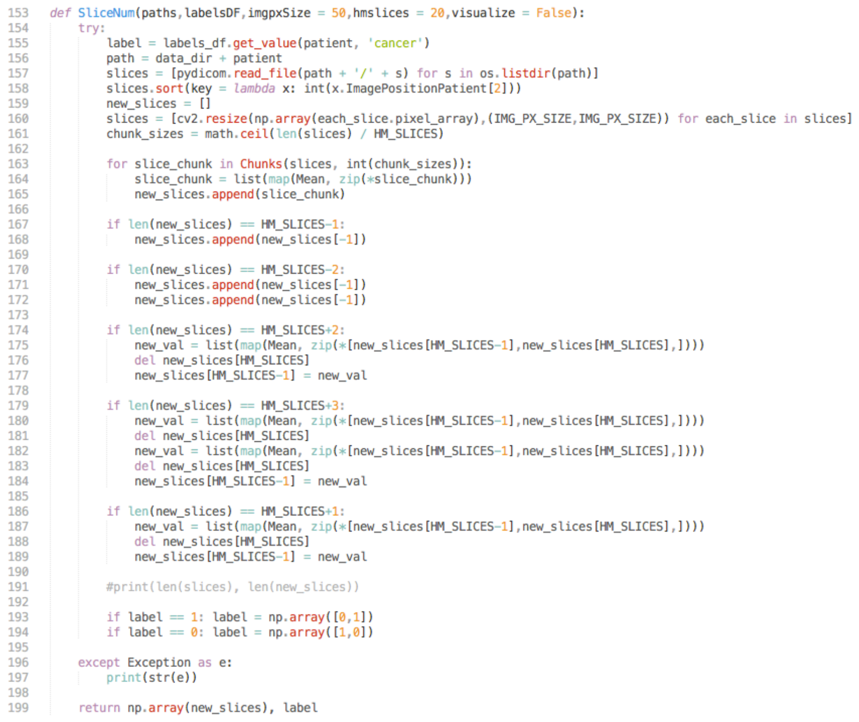


```python
def zeroCenter(image):
    image = image - pixelMEAN

    return image
```

Figure 9: Zero Centering CT Scan Images

3.3.8 Slice Count

This is the final preprocessing step. Since the number of slice in the CT scans varies from one patient to another, the number of slice will have to be made uniform across all patients in order for the CNN architecture to be able to read and process the dataset. The following code snippet creates a uniformity across all patients by condensing the variable number of slices to 20.



```python
def SliceNum(paths,labelsDF,imgpxSize = 50,hmslices = 20,visualize = False):
    try:
        label = labels_df.get_value(patient, 'cancer')
        path = data_dir + patient
        slices = [pydicom.read_file(path + '/' + s) for s in os.listdir(path)]
        slices.sort(key = lambda x: int(x.ImagePositionPatient[2]))
        new_slices = []
        slices = [cv2.resize(np.array(each_slice.pixel_array),(IMG_PX_SIZE,IMG_PX_SIZE)) for each_slice in slices]
        chunk_sizes = math.ceil(len(slices) / HM_SLICES)

        for slice_chunk in Chunks(slices, int(chunk_sizes)):
            slice_chunk = list(map(Mean, zip(*slice_chunk)))
            new_slices.append(slice_chunk)

        if len(new_slices) == HM_SLICES-1:
            new_slices.append(new_slices[-1])

        if len(new_slices) == HM_SLICES-2:
            new_slices.append(new_slices[-1])
            new_slices.append(new_slices[-1])

        if len(new_slices) == HM_SLICES+2:
            new_val = list(map(Mean, zip(*[new_slices[HM_SLICES-1],new_slices[HM_SLICES],])))
            del new_slices[HM_SLICES]
            new_slices[HM_SLICES-1] = new_val

        if len(new_slices) == HM_SLICES+3:
            new_val = list(map(Mean, zip(*[new_slices[HM_SLICES-1],new_slices[HM_SLICES],])))
            del new_slices[HM_SLICES]
            new_val = list(map(Mean, zip(*[new_slices[HM_SLICES-1],new_slices[HM_SLICES],])))
            del new_slices[HM_SLICES]
            new_slices[HM_SLICES-1] = new_val

        if len(new_slices) == HM_SLICES+1:
            new_val = list(map(Mean, zip(*[new_slices[HM_SLICES-1],new_slices[HM_SLICES],])))
            del new_slices[HM_SLICES]
            new_slices[HM_SLICES-1] = new_val

        #print(len(slices), len(new_slices))

        if label == 1: label = np.array([0,1])
        if label == 0: label = np.array([1,0])

    except Exception as e:
        print(str(e))

    return np.array(new_slices), label
```

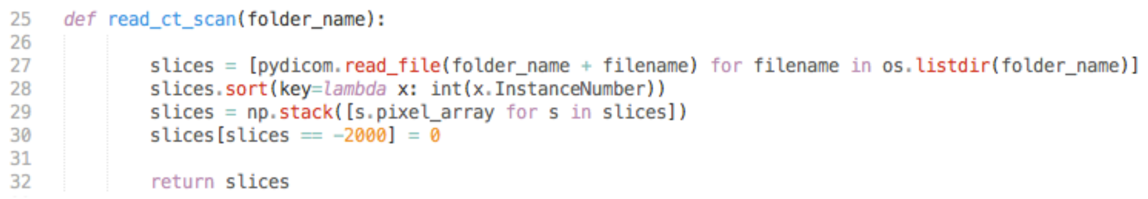
Figure 10: Condensing to 20 Slices for all patients

CHAPTER 4: CANCER NODULE CANDIDATE DETECTION

The CT scans images of the Kaggle Data Science Dataset consist of CT scans for 1595 patients and each 3D scans consists of multiple slices. The CT scans consist of not only imaging of the lung but also the surrounding noise from other tissues, bones and blood vessels. The following code snippets use several lung segmentation techniques in order to extract the Region of Interest (ROI) and highlight the regions in the lung CT scan which have high probability of containing cancer nodules. Cancer nodule candidate detection is a two step process, where the lungs are first segmented to obtain the ROI and then further isolated from the low intensity regions in the CT scans.

4.1 READING CT SCANS

The following code snipped illustrates the process of reading CT scan DICOM images using OS directory handling and pydicom python modules.



```
25  def read_ct_scan(folder_name):
26
27          slices = [pydicom.read_file(folder_name + filename) for filename in os.listdir(folder_name)]
28          slices.sort(key=lambda x: int(x.InstanceNumber))
29          slices = np.stack([s.pixel_array for s in slices])
30          slices[slices == -2000] = 0
31
32          return slices
```

Figure 11: Reading CT Scans

Line 27, reads each slice of the CT scan of a patient stored on the drive. Line 28 is used to sort and order each slice in their respective position in the 3D CT scan stack of the patient. Line 30, any intensity value that is -2000 lies outside the scanner bounds and is thus set to zero.

4.2 PLOTTING CT SCANS

To visualize the CT scan slices, they are plotted with the help of *matplotlib* module. subplots() function is used in order to display multiple images or CT slices in a single window.

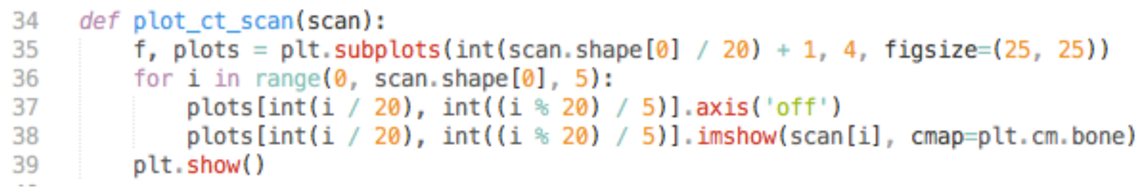

```
def plot_ct_scan(scan):
    f, plots = plt.subplots(int(scan.shape[0] / 20) + 1, 4, figsize=(25, 25))
    for i in range(0, scan.shape[0], 5):
        plots[int(i / 20), int((i % 20) / 5)].axis('off')
        plots[int(i / 20), int((i % 20) / 5)].imshow(scan[i], cmap=plt.cm.bone)
    plt.show()
```

Figure 12: Plotting CT Scans

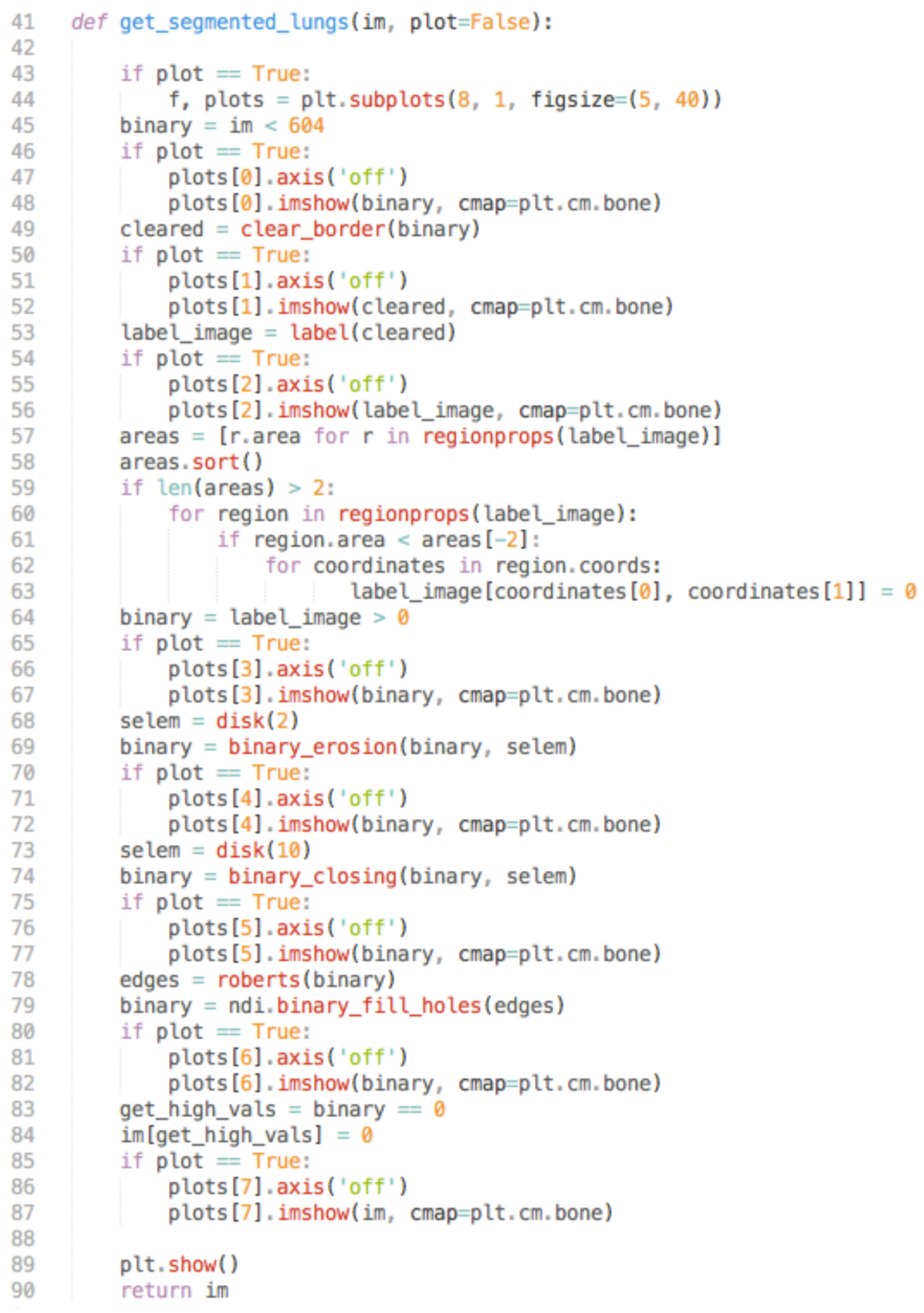
4.3 SEGMENTATION OF LUNG SCANS

After reading the CT Scan, the first step in preprocessing is the segmentation of lung structures because it is obvious that the regions of interests lies inside the lungs. It is visible that the lungs are the darker regions in the CT Scans. The bright region inside the lungs are the blood vessels or air. A threshold of -400 HU is used at all places. We segment lung structures from each slice of the CT Scan image and try not to loose the possible region of interests attached to the lung wall. There are some nodules which may be attached to the lung wall.

The CT scan images are first converted to binary images. A binary image is a type of image where the pixels have only two values which usually result in black or white.
clear_border from *skimage.segmentation* is then used to clear the objects attached to the border of the binary image.
label from *skimage.measure* is then used to label all connected regions of the binary image. Two pixels are said to be connected when the have the same pixel value.
regionprops() is used to measure the properties of the labeled image regions. *regionprops()* helps us obtain the *area* which is the number of pixels of the region. From the code snippet below, labels with the 2 largest areas are kept. The input to *regionprops()* is a labeled binary image and the labels with value zero are ignored.
binary_erosion is used with a disk of radius 2 which is known as the structuring element. The basic idea of binary erosion is to probe an image with a pre-defined shape, drawing conclusions on how the shape fits or misses the shapes in the image.
binary_closing is used with a disk of radius 10 as the structuring element. This operation tends to enlarge the boundaries of the foreground or bright regions and shrink background

color holes in such regions. This operation is done in order to keep the nodules attached to the lung walls.

binary_fill_holes is used to fill small holes inside the binary image.

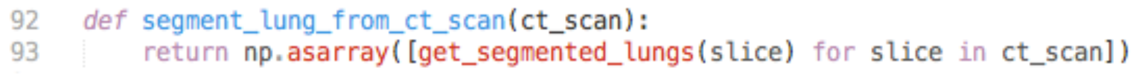


```python
def get_segmented_lungs(im, plot=False):

    if plot == True:
        f, plots = plt.subplots(8, 1, figsize=(5, 40))
    binary = im < 604
    if plot == True:
        plots[0].axis('off')
        plots[0].imshow(binary, cmap=plt.cm.bone)
    cleared = clear_border(binary)
    if plot == True:
        plots[1].axis('off')
        plots[1].imshow(cleared, cmap=plt.cm.bone)
    label_image = label(cleared)
    if plot == True:
        plots[2].axis('off')
        plots[2].imshow(label_image, cmap=plt.cm.bone)
    areas = [r.area for r in regionprops(label_image)]
    areas.sort()
    if len(areas) > 2:
        for region in regionprops(label_image):
            if region.area < areas[-2]:
                for coordinates in region.coords:
                       label_image[coordinates[0], coordinates[1]] = 0
    binary = label_image > 0
    if plot == True:
        plots[3].axis('off')
        plots[3].imshow(binary, cmap=plt.cm.bone)
    selem = disk(2)
    binary = binary_erosion(binary, selem)
    if plot == True:
        plots[4].axis('off')
        plots[4].imshow(binary, cmap=plt.cm.bone)
    selem = disk(10)
    binary = binary_closing(binary, selem)
    if plot == True:
        plots[5].axis('off')
        plots[5].imshow(binary, cmap=plt.cm.bone)
    edges = roberts(binary)
    binary = ndi.binary_fill_holes(edges)
    if plot == True:
        plots[6].axis('off')
        plots[6].imshow(binary, cmap=plt.cm.bone)
    get_high_vals = binary == 0
    im[get_high_vals] = 0
    if plot == True:
        plots[7].axis('off')
        plots[7].imshow(im, cmap=plt.cm.bone)

    plt.show()
    return im
```

Figure 13: Lung Segmentation to obtain ROI

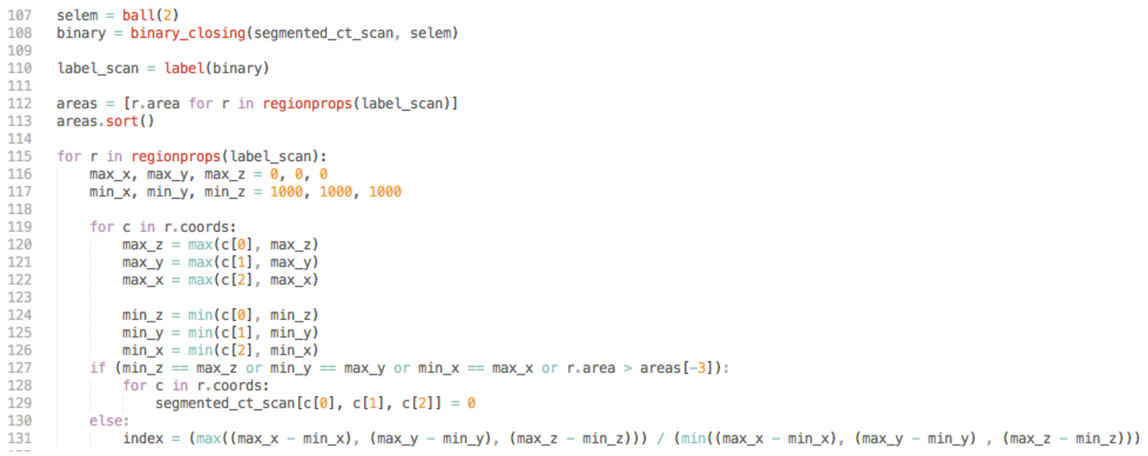
4.4 NODULE CANDIDATE GENERATION

After segmenting the lung structures from the CT Scanned images, our task is to find the candidate regions with nodules since the search space is very large. It was found in literature review that all the region of interests have intensity -400 HU. So, we used this threshold to filter the darker regions. This reduces the number of candidates by a large number and preserves all the important regions. The following code snippet is used to remove the noise associated with the blood vessels which is accomplished by removing the two largest connected components.



```
def segment_lung_from_ct_scan(ct_scan):
    return np.asarray([get_segmented_lungs(slice) for slice in ct_scan])
```

Figure 14: Nodule Candidate Generation



```
selem = ball(2)
binary = binary_closing(segmented_ct_scan, selem)

label_scan = label(binary)

areas = [r.area for r in regionprops(label_scan)]
areas.sort()

for r in regionprops(label_scan):
    max_x, max_y, max_z = 0, 0, 0
    min_x, min_y, min_z = 1000, 1000, 1000

    for c in r.coords:
        max_z = max(c[0], max_z)
        max_y = max(c[1], max_y)
        max_x = max(c[2], max_x)

        min_z = min(c[0], min_z)
        min_y = min(c[1], min_y)
        min_x = min(c[2], min_x)
    if (min_z == max_z or min_y == max_y or min_x == max_x or r.area > areas[-3]):
        for c in r.coords:
            segmented_ct_scan[c[0], c[1], c[2]] = 0
    else:
        index = (max((max_x - min_x), (max_y - min_y), (max_z - min_z))) / (min((max_x - min_x), (max_y - min_y) , (max_z - min_z)))
```

Figure 15: Removal of Noise associated with Blood Vessels

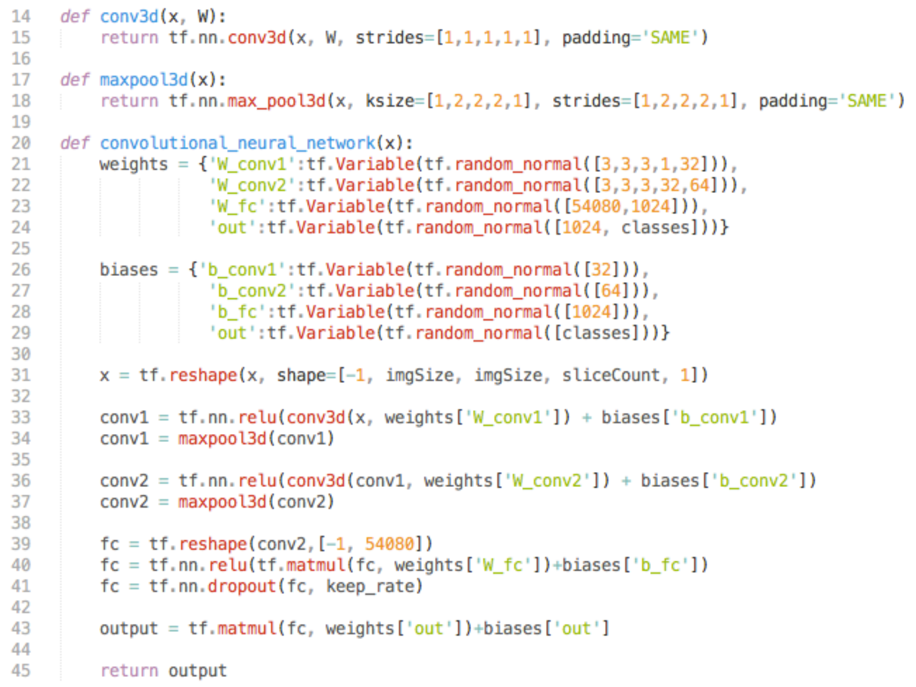
CHAPTER 5: CONVOLUTIONAL NEURAL NETWORK

CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function. CNNs have wide applications in image and video recognition, recommender systems and natural language processing. Unlike neural networks, where the input is a vector, here the input is a multi-channeled image depending upon whether the images are RGB or grayscale images.

The CNN architecture is typically made up of three components, namely, convolutional layer, pooling layer and the fully connected layer. In the convolutional layer, the n*n*n*c image is convoluted with a filter of dimension f*f*f*c. The output of the convolutional layer is then fed to the pooling layer where either max pooling or average pooling can be performed. A filter of size k*k*k is defined and in case of max pooling, the element with the maximum value in the k*k*k elements is calculated and kept while the rest of the elements are discarded. The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network.

5.1 CNN ARCHITECTURE

The Convolutional Neural Network architecture used for the prediction of presence or absence of cancer nodules candidates in the CT scans consists of two convolutional layers each of which are followed by a max pooling layer and is finally fed to a fully connected layer which is then connected to the output, where we can obtain the prediction from. Both convolutional layers use a filter of size 3 with stride 1 and max pooling is achieved with the help of filter size 2 with stride 2. In order to reduce over-fitting, dropout regularization technique is applied to the fully connected layer with a keep probability of 0.8. In the CNN architecture, the convolutional and max pooling layers are used for feature learning and the fully connected layer is used for output prediction. The code snippet below illustrated the CNN architecture.

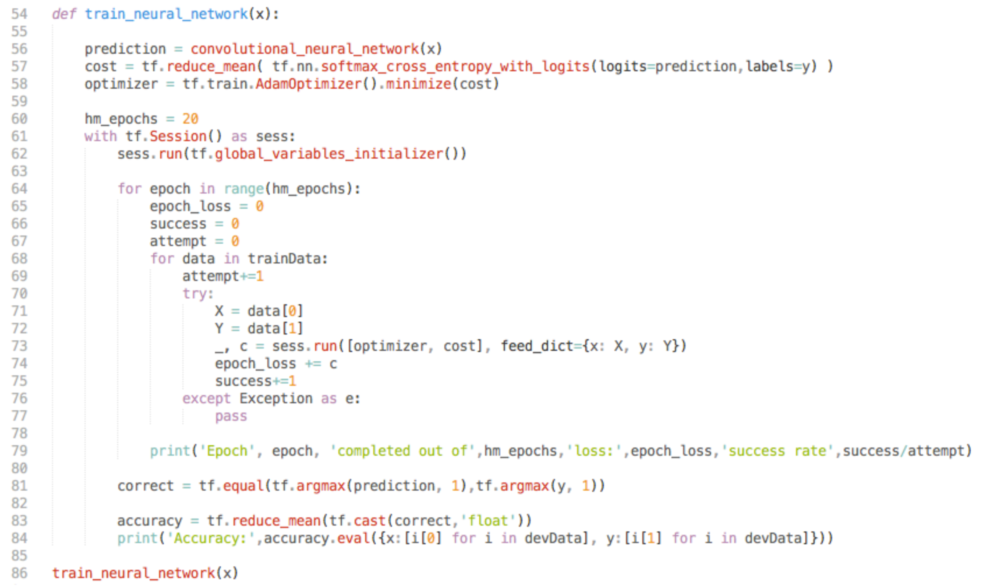


```python
def conv3d(x, W):
    return tf.nn.conv3d(x, W, strides=[1,1,1,1,1], padding='SAME')

def maxpool3d(x):
    return tf.nn.max_pool3d(x, ksize=[1,2,2,2,1], strides=[1,2,2,2,1], padding='SAME')

def convolutional_neural_network(x):
    weights = {'W_conv1':tf.Variable(tf.random_normal([3,3,3,1,32])),
               'W_conv2':tf.Variable(tf.random_normal([3,3,3,32,64])),
               'W_fc':tf.Variable(tf.random_normal([54080,1024])),
               'out':tf.Variable(tf.random_normal([1024, classes]))}

    biases = {'b_conv1':tf.Variable(tf.random_normal([32])),
              'b_conv2':tf.Variable(tf.random_normal([64])),
              'b_fc':tf.Variable(tf.random_normal([1024])),
              'out':tf.Variable(tf.random_normal([classes]))}

    x = tf.reshape(x, shape=[-1, imgSize, imgSize, sliceCount, 1])

    conv1 = tf.nn.relu(conv3d(x, weights['W_conv1']) + biases['b_conv1'])
    conv1 = maxpool3d(conv1)

    conv2 = tf.nn.relu(conv3d(conv1, weights['W_conv2']) + biases['b_conv2'])
    conv2 = maxpool3d(conv2)

    fc = tf.reshape(conv2,[-1, 54080])
    fc = tf.nn.relu(tf.matmul(fc, weights['W_fc'])+biases['b_fc'])
    fc = tf.nn.dropout(fc, keep_rate)

    output = tf.matmul(fc, weights['out'])+biases['out']

    return output
```

Figure 16: Convolutional Neural Network Architecture

5.2 TRAINING CNN

The CNN cost is computed with the help of a softmax function and Adam Optimizer is used for optimizing the cost. The dataset set is divided partially into the training set and the test set. The dataset is iterated through 20 epochs, where an epoch represents one forward propagation and one back propagation through the entire training dataset.



```python
def train_neural_network(x):

    prediction = convolutional_neural_network(x)
    cost = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(logits=prediction,labels=y) )
    optimizer = tf.train.AdamOptimizer().minimize(cost)

    hm_epochs = 20
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        for epoch in range(hm_epochs):
            epoch_loss = 0
            success = 0
            attempt = 0
            for data in trainData:
                attempt+=1
                try:
                    X = data[0]
                    Y = data[1]
                    _, c = sess.run([optimizer, cost], feed_dict={x: X, y: Y})
                    epoch_loss += c
                    success+=1
                except Exception as e:
                    pass

            print('Epoch', epoch, 'completed out of',hm_epochs,'loss:',epoch_loss,'success rate',success/attempt)

        correct = tf.equal(tf.argmax(prediction, 1),tf.argmax(y, 1))

        accuracy = tf.reduce_mean(tf.cast(correct,'float'))
        print('Accuracy:',accuracy.eval({x:[i[0] for i in devData], y:[i[1] for i in devData]}))

train_neural_network(x)
```

Figure 17: Training CNN Architecture

CHAPTER 6: RESULT AND DISCUSSIONS

6.1 PREPROCESSED IMAGES

The CT scan images were preprocessed before being fed to the Convolutional Neural Network architecture as mentioned in Chapter 3. The following are sample CT scan slices after each image preprocessing step.

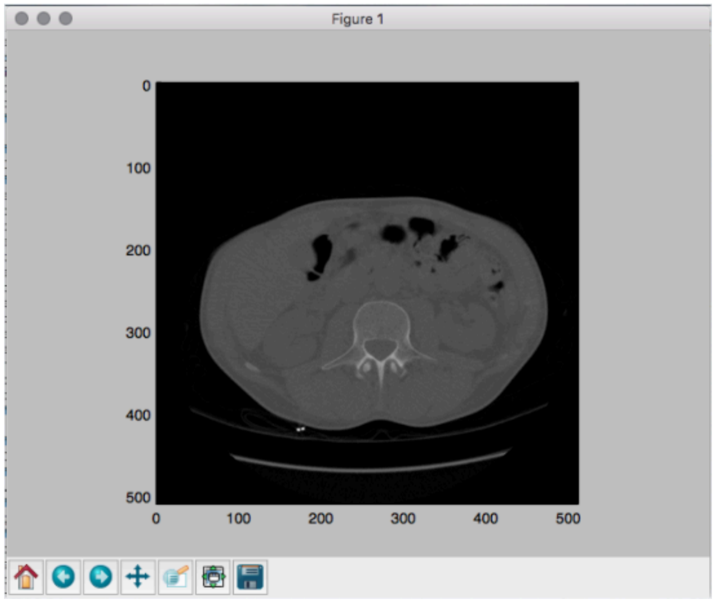


Figure 18: Sample Slice from a Patient after conversion to Hounsfield Units

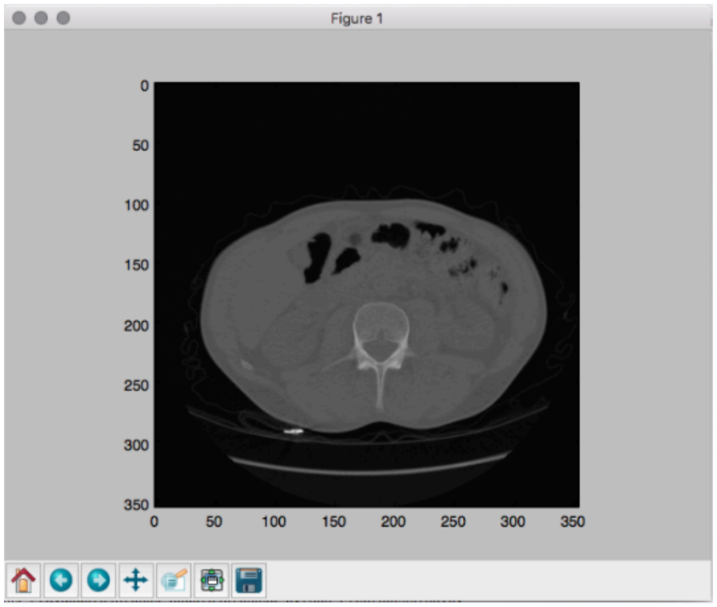


Figure 19: Sample Slice from a Patient after Resampling the CT Scans

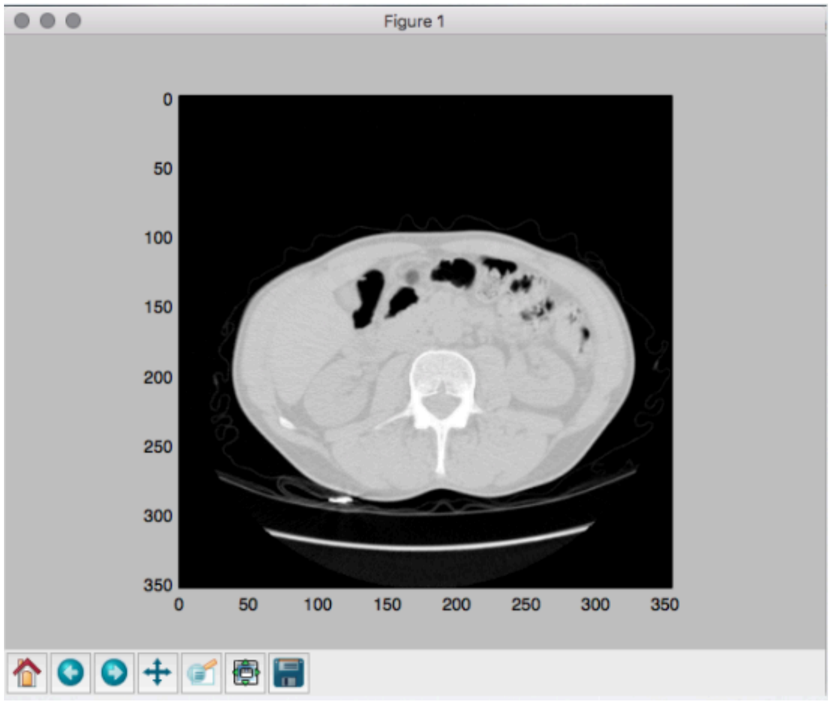


Figure 20: Sample Slice from a Patient after Normalising CT Scan Images

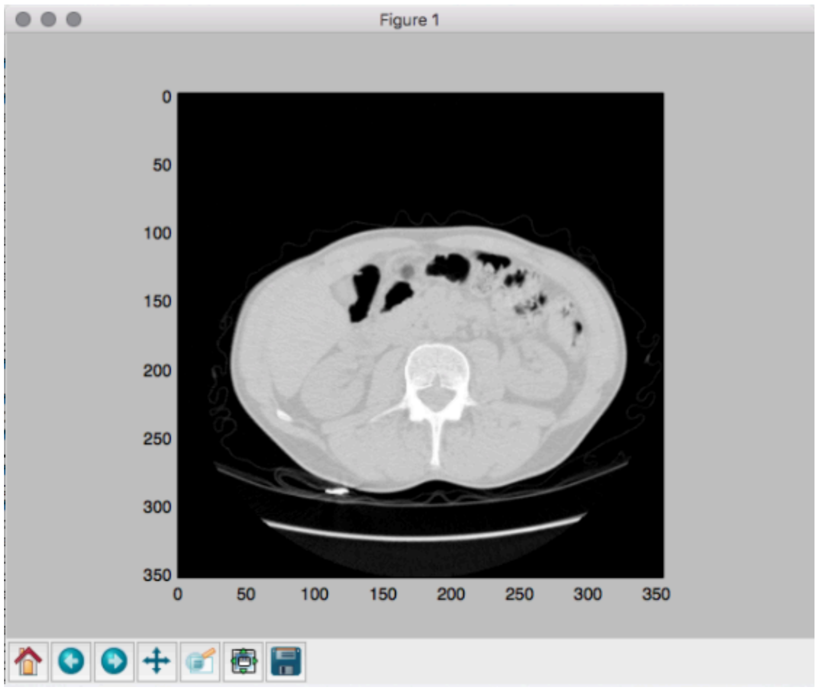


Figure 21: Sample Slice from a Patient after Zero Centering CT Scan Images

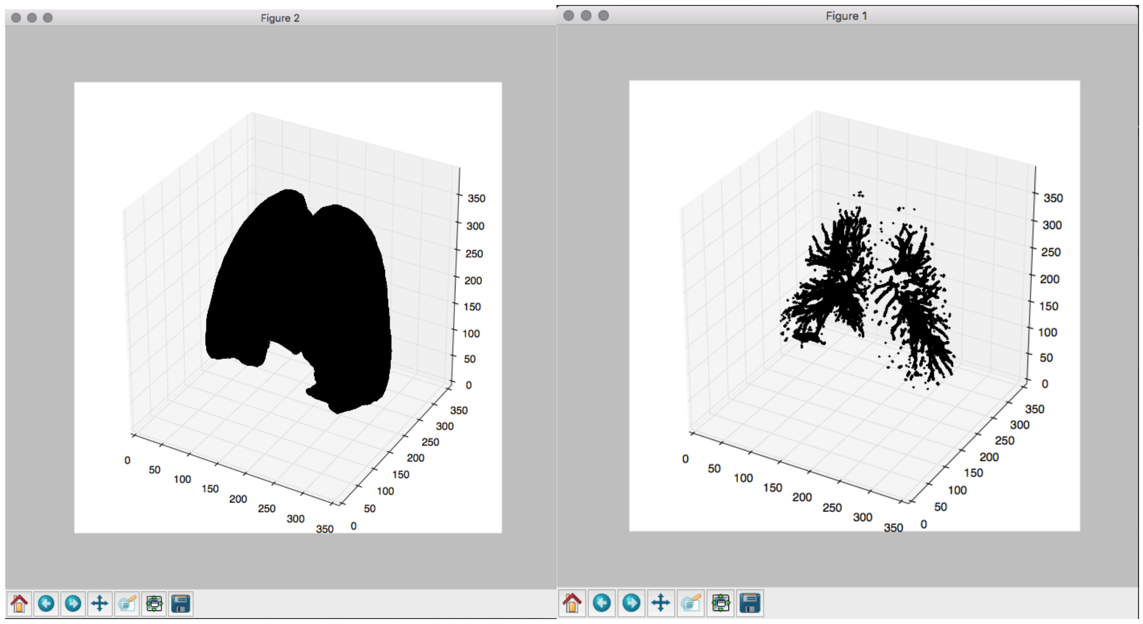


Figure 22: 3D Plots of CT Scan from a Patient after Segmenting CT Scan Images

6.2 LUNG CANCER NODULE CANDIDATES DETECTION

The following are CT scan images obtained after each step in lung segmentation and nodule detection. Finally, the possible regions of cancer nodule candidates has been 3D plotted.

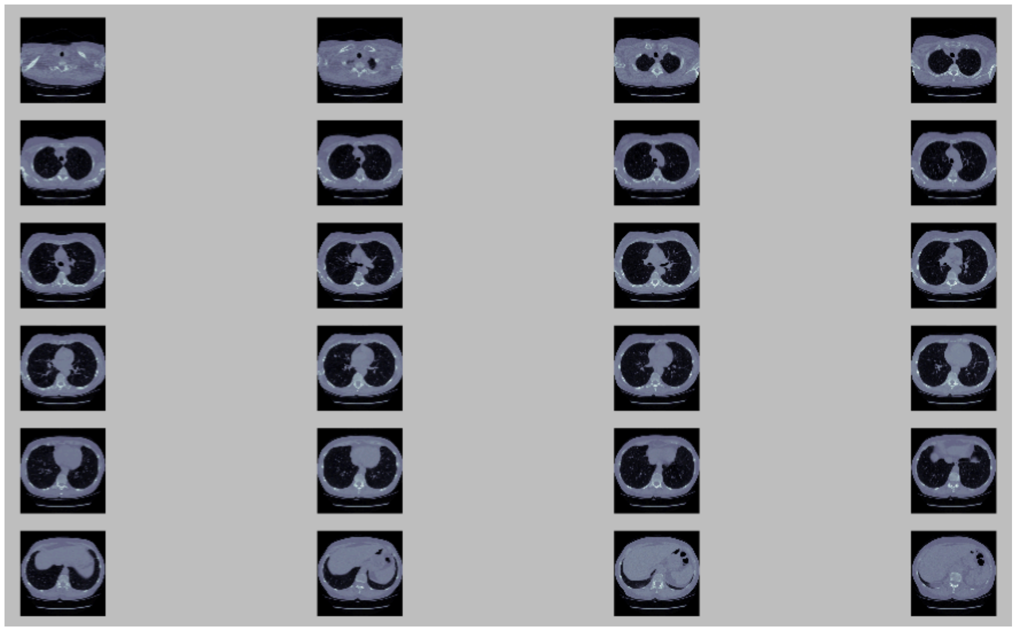


Figure 23: CT Scan Images from Dataset before Segmentation

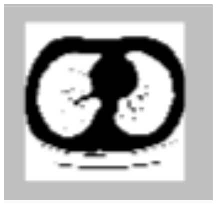


Figure 24: Converting CT Scans to Binary Images

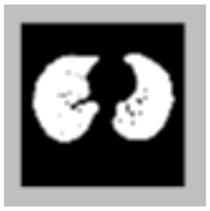


Figure 25: Binary Image after *clear_border* Operation

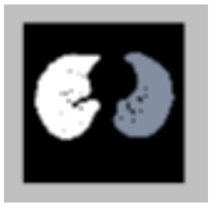


Figure 26: Binary Image after *label* Operation

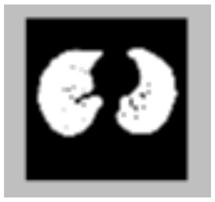


Figure 27: Binary Image after *regionprops* Operation

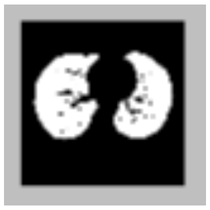


Figure 28: Binary Image after *binary_erosion* Operation

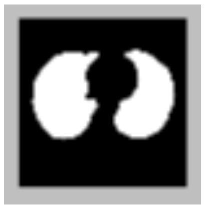


Figure 29: Binary Image after *binary_closing* Operation

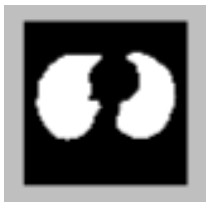


Figure 30: Binary Image after *binary_fill_holes* Operation

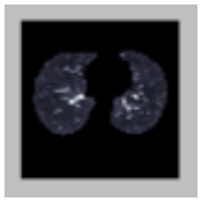


Figure 31: Binary Image after Superimposing Binary Mask on Input Image

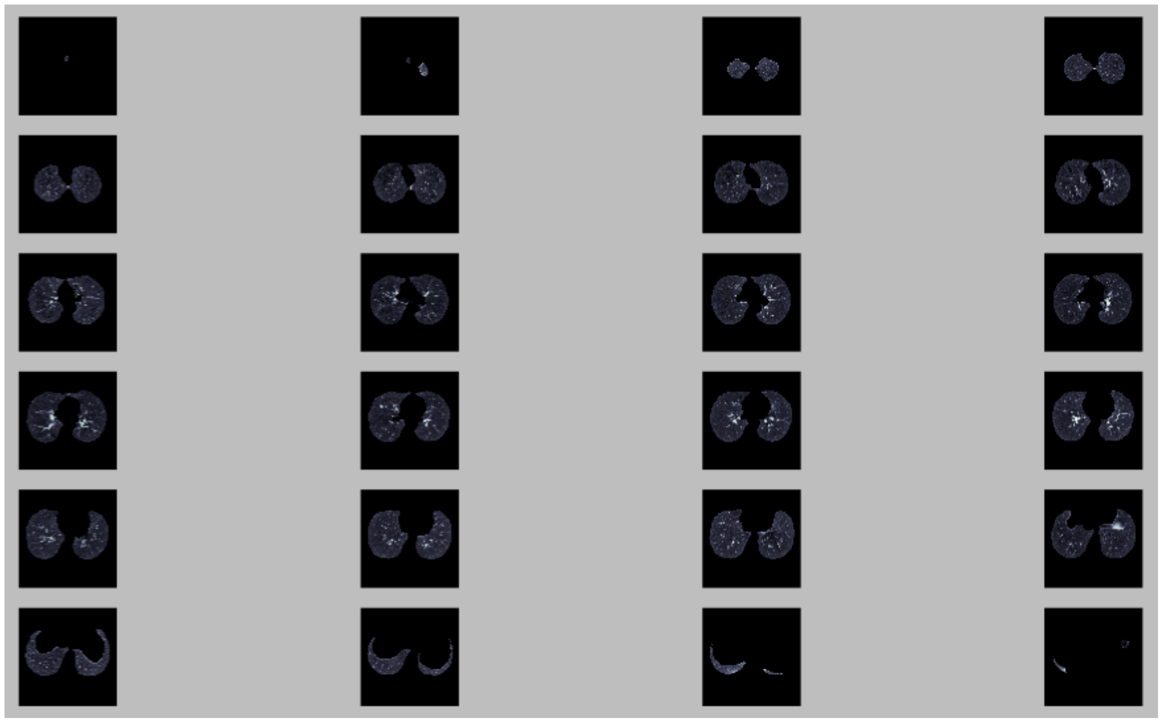


Figure 32: After Segmenting CT Scans

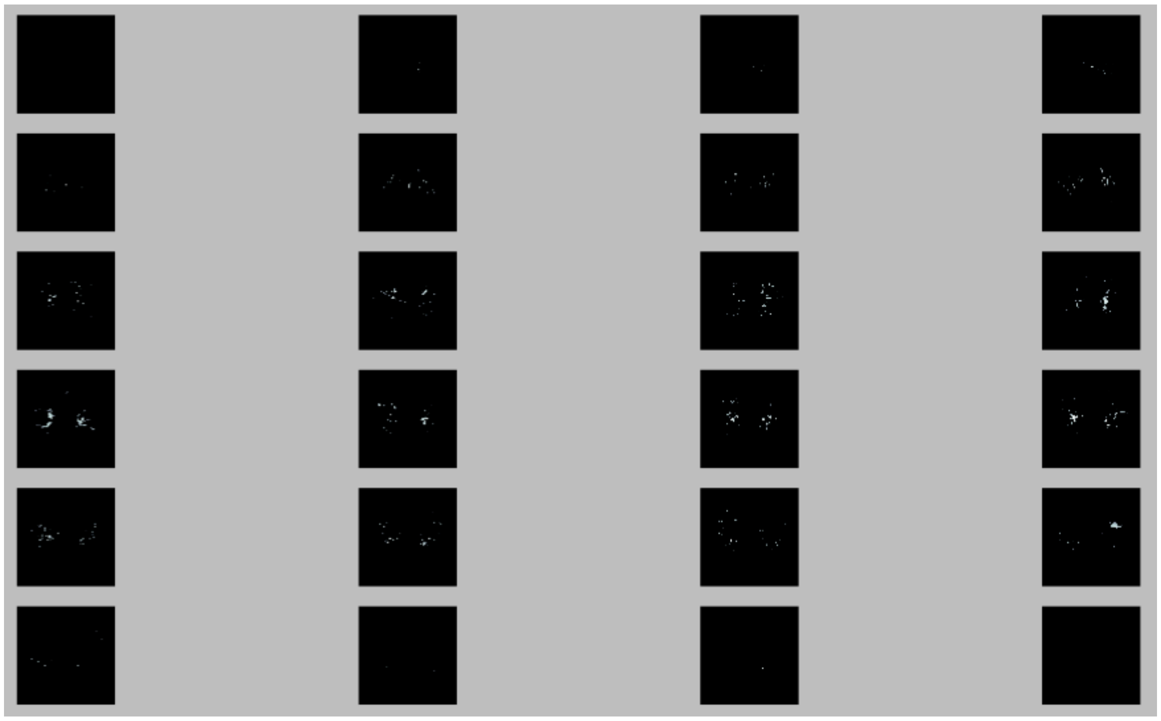


Figure 33: Nodule Candidates

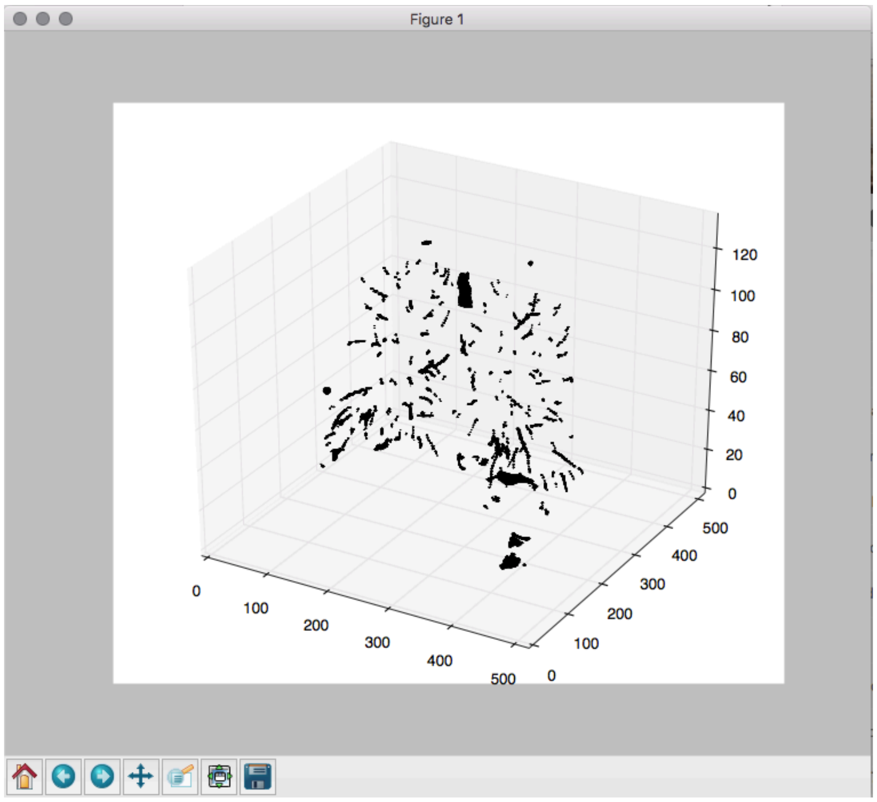


Figure 34: Nodule Candidates in 3D Scans

6.3 CNN TRAINING OUTCOME

The CNN architecture consists of two convolutional layers followed by two pooling layers where max pooling was performed. The CNN consists of one fully connect (FC) layer with dropout regularization of keep probability 0.8. The CNN architecture was trained through 20 epochs and with three different optimizers namely, Gradient Descent, RMSProp and Adam. The following table illustrates the accuracy obtained on the test data.

Gradient Descent	RMSProp	Adam
0.29	0.45	0.52

Table 2: Accuracies for Three Different Optimizers

CHAPTER 7: CONCLUSION

7.1 SUMMARY

The design, architecture and working of the convolutional neural network were presented. It was concluded that among the three optimizers, Adam Optimizer provided the highest test data accuracy. The image processing technique for nodule detection in the lungs for possible tumor sites was also explained. The existing literature concerning computational analysis of medical imaging was reviewed. The system was designed using Python 2.7 and using an open deep learning framework TensorFlow provided by Google. The dataset of CT scans of lungs of various patients to train and test the convolutional neural network were available thanks to Kaggle.

7.2 FUTURE WORK

Lung cancer, one of the most common forms of cancer, is an area where continuous research is being done, in order to improve the diagnosis efficiency, early detection, and proper stage classification. Future work that can be done to improve upon the project is listed below.

7.1.1 Classification of lung nodules

The candidate regions generated still has a lot of noise. Thus we need a classifier to classify the candidates as either Nodule or Non-Nodule. This is also referred to as False Positive Reduction step.

7.1.2 Classification of lung cancer into stages of development

Further design, and training of the neural network with more detailed datasets that contain labels regarding the stage of cancer as well, can be used to create an automated system that detects not only the presence of cancer, but also the stage to which it has progressed.

7.1.3 Classification of malignancy of lung nodules

The classification of nodules detected into malignant or benign can also be done to further improve the scope of the system.

Considering all the benefits of an automated lung cancer detection system using neural networks, it is clear that such systems will have a huge impact in the medical industry in the near future.

REFERENCES

1. Taher, F. and Sammouda, R., 2011, February. Lung cancer detection by using artificial neural network and fuzzy clustering methods. In *GCC Conference and Exhibition (GCC), 2011 IEEE* (pp. 295-298). IEEE.

2. Wang, Z., Xu, H. and Sun, M., 2017, December. Deep Learning Based Nodule Detection from Pulmonary CT Images. In *Computational Intelligence and Design (ISCID), 2017 10th International Symposium on* (Vol. 1, pp. 370-373). IEEE.

3. Alakwaa, W., Nassef, M. and Badr, A., 2017. Lung Cancer Detection and Classification with 3D Convolutional Neural Network (3D-CNN). *Lung Cancer*, *8*(8).

4. Stewart, B.W.K.P. and Wild, C.P., 2017. World cancer report 2014. Health.

5. Rao, P., Pereira, N.A. and Srinivasan, R., 2016, December. Convolutional neural networks for lung cancer screening in computed tomography (CT) scans. In Contemporary Computing and Informatics (IC3I), 2016 2nd International Conference on (pp. 489-493). IEEE.

6. Chon, A., Balachandar, N. and Lu, P., 2017. Deep convolutional neural networks for lung cancer detection. tech. rep., Stanford University.

7. Dandıl, E., Çakiroğlu, M., Ekşi, Z., Özkan, M., Kurt, Ö.K. and Canan, A., 2014, August. Artificial neural network-based classification system for lung nodules on computed tomography scans. In Soft computing and pattern recognition (soCPar), 2014 6th international conference of (pp. 382-386). IEEE.

APPENDIX

APPENDIX A

CODE ATTACHMENTS

The following is a code attachment for the code used for image preprocessing.

```
import math
import pydicom
import os
import pandas as pd
import scipy.ndimage
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import measure
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

minBOUND = -1000.0
maxBOUND = 400.0
pixelMEAN = 0.25
IMG_PX_SIZE = 50
HM_SLICES = 20

data_dir = 'FYP/stage1/'
patients = os.listdir(data_dir)
labels_df = pd.read_csv('FYP/stage1_labels.csv', index_col=0)

much_data = []

def SliceThick(path):
    slices = [pydicom.read_file(path + '/' + s) for s in os.listdir(path)]
    slices.sort(key = lambda x: float(x.ImagePositionPatient[2]))

    try:
        slice_thickness = np.abs(slices[0].ImagePositionPatient[2] - slices[1].ImagePositionPatient[2])
    except:
        slice_thickness = np.abs(slices[0].SliceLocation - slices[1].SliceLocation)

    for s in slices:
        s.SliceThickness = slice_thickness

    return slices

def HU(Slices):
    image = np.stack([s.pixel_array for s in Slices])
    image = image.astype(np.int16)

    image[image == -2000] = 0

    for slice_number in range(len(Slices)):

        intercept = Slices[slice_number].RescaleIntercept
        slope = Slices[slice_number].RescaleSlope
        if slope != 1:
            image[slice_number] = slope * image[slice_number].astype(np.float64)
            image[slice_number] = image[slice_number].astype(np.int16)

        image[slice_number] += np.int16(intercept)

    return np.array(image, dtype=np.int16)

def Resample(image, scan, new_spacing=[1,1,1]):

    spacing_list = [scan[0].SliceThickness]
    spacing_list.extend(scan[0].PixelSpacing)
    spacing = np.array(spacing_list, dtype=np.float32)

    resize_factor = spacing / new_spacing
    new_real_shape = image.shape * resize_factor
    new_shape = np.round(new_real_shape)
    real_resize_factor = new_shape / image.shape
    new_spacing = spacing / real_resize_factor

    image = scipy.ndimage.interpolation.zoom(image, real_resize_factor, mode='nearest')

    return image, new_spacing

def plot3D(image, threshold=-300):

    p = image.transpose(2,1,0)

    verts, faces = measure.marching_cubes_classic(p, threshold)

    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')

    mesh = Poly3DCollection(verts[faces], alpha=0.70)
    face_color = [0.45, 0.45, 0.75]
    mesh.set_facecolor(face_color)
    ax.add_collection3d(mesh)

    ax.set_xlim(0, p.shape[0])
    ax.set_ylim(0, p.shape[1])
    ax.set_zlim(0, p.shape[2])

    plt.show()

def largestLabelVolume(im, bg=-1):
    vals, counts = np.unique(im, return_counts=True)

    counts = counts[vals != bg]
    vals = vals[vals != bg]

    if len(counts) > 0:
        return vals[np.argmax(counts)]
    else:
        return None
```

```
def segmentLungMask(image, fill_lung_structures=True):

    binary_image = np.array(image > -320, dtype=np.int8)+1
    labels = measure.label(binary_image)
    background_label = labels[0,0,0]
    binary_image[background_label == labels] = 2

    if fill_lung_structures:

        for i, axial_slice in enumerate(binary_image):
            axial_slice = axial_slice - 1
            labeling = measure.label(axial_slice)
            l_max = largestLabelVolume(labeling, bg=0)

            if l_max is not None:
                binary_image[i][labeling != l_max] = 1

    binary_image -= 1
    binary_image = 1-binary_image

    labels = measure.label(binary_image, background=0)
    l_max = largestLabelVolume(labels, bg=0)
    if l_max is not None:
        binary_image[labels != l_max] = 0

    return binary_image

def Normalize(image):
    image = (image - minBOUND) / (maxBOUND - minBOUND)
    image[image>1] = 1.
    image[image<0] = 0.

    return image

def zeroCenter(image):
    image = image - pixelMEAN

    return image

def Chunks(l, n):
    for i in range(0, len(l), n):
        yield l[i:i + n]

def Mean(l):
    return sum(l) / len(l)

def SliceNum(paths,labelsDF,imgpxSize = 50,hmslices = 20,visualize = False):
    try:
        label = labels_df.get_value(patient, 'cancer')
        path = data_dir + patient
        slices = [pydicom.read_file(path + '/' + s) for s in os.listdir(path)]
        slices.sort(key = lambda x: int(x.ImagePositionPatient[2]))
        new_slices = []
        slices = [cv2.resize(np.array(each_slice.pixel_array),(IMG_PX_SIZE,IMG_PX_SIZE)) for each_slice
in slices]
        chunk_sizes = math.ceil(len(slices) / HM_SLICES)
        for slice_chunk in Chunks(slices, int(chunk_sizes)):
            slice_chunk = list(map(Mean, zip(*slice_chunk)))
            new_slices.append(slice_chunk)

        if len(new_slices) == HM_SLICES-1:
            new_slices.append(new_slices[-1])

        if len(new_slices) == HM_SLICES-2:
            new_slices.append(new_slices[-1])
            new_slices.append(new_slices[-1])

        if len(new_slices) == HM_SLICES+2:
            new_val = list(map(Mean, zip(*[new_slices[HM_SLICES-1],new_slices[HM_SLICES],])))
            del new_slices[HM_SLICES]
            new_slices[HM_SLICES-1] = new_val

        if len(new_slices) == HM_SLICES+3:
            new_val = list(map(Mean, zip(*[new_slices[HM_SLICES-1],new_slices[HM_SLICES],])))
            del new_slices[HM_SLICES]
            new_val = list(map(Mean, zip(*[new_slices[HM_SLICES-1],new_slices[HM_SLICES],])))
            del new_slices[HM_SLICES]
            new_slices[HM_SLICES-1] = new_val

        if len(new_slices) == HM_SLICES+1:
            new_val = list(map(Mean, zip(*[new_slices[HM_SLICES-1],new_slices[HM_SLICES],])))
            del new_slices[HM_SLICES]
            new_slices[HM_SLICES-1] = new_val

        #print(len(slices), len(new_slices))

        if label == 1: label = np.array([0,1])
            if label == 0: label = np.array([1,0])

    except Exception as e:
        print(str(e))

    return np.array(new_slices), label

def saveData(numcycle,pat):

    if numcycle % 100 == 0:
            print(num)
    try:
        img_data,label = SliceNum(pat,labels_df,imgpxSize=IMG_PX_SIZE, hmslices=HM_SLICES)
        much_data.append([img_data,label])
    except UnboundLocalError as e:
        print('This is unlabeled data!')

for num,patient in enumerate(patients[:70]):
            print len(much_data)

            X = data_dir + patient

            Slices = SliceThick(X)
            HUImage = HU(Slices)
            plt.subplot(231)
            plt.imshow(HUImage[0])

            ResampledImg,NewSpace = Resample(HUImage,Slices)
            plt.subplot(232)
            plt.imshow(ResampledImg[0])

            NormImg = Normalize(ResampledImg)
            plt.subplot(233)
            plt.imshow(NormImg[0])

            ZCImg = zeroCenter(NormImg)
            plt.subplot(234)
            plt.imshow(ZCImg[0])

            segmentedLungs = segmentLungMask(ResampledImg, False)

            segmentedLungsFill = segmentLungMask(ResampledImg, True)

            saveData(num,segmentedLungs)

            print ("%i Patients Preprocessed." %(num+1))

            plot3D(segmentedLungs, 0)
            plot3D(segmentedLungsFill, 0)
            plot3D(segmentedLungsFill - segmentedLungs, 0)

            plt.show()

np.save('preprocessedimages-{}-{}-{}.npy'.format(IMG_PX_SIZE,IMG_PX_SIZE,HM_SLICES),
much_data)

newData = np.load('preprocessedimages-50-50-20.npy')
print len(newData)
```

The following is a code attachment for the code used for nodule detection.

```
import numpy as np
import pandas as pd
import skimage, os
from skimage.morphology import ball, disk, dilation, binary_erosion, remove_small_objects, erosion,
closing, reconstruction, binary_closing
from skimage.measure import label,regionprops, perimeter
from skimage.morphology import binary_dilation, binary_opening
from skimage.filters import roberts, sobel
from skimage import measure, feature
from skimage.segmentation import clear_border
from skimage import data
from scipy import ndimage as ndi
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import pydicom
import scipy.misc
from subprocess import check_output


def read_ct_scan(folder_name):

    slices = [pydicom.read_file(folder_name + filename) for filename in os.listdir(folder_name)]
    slices.sort(key=lambda x: int(x.InstanceNumber))
    slices = np.stack([s.pixel_array for s in slices])
    slices[slices == -2000] = 0

    return slices

def plot_ct_scan(scan):
  f, plots = plt.subplots(int(scan.shape[0] / 20) + 1, 4, figsize=(25, 25))
  for i in range(0, scan.shape[0], 5):
    plots[int(i / 20), int((i % 20) / 5)].axis('off')
    plots[int(i / 20), int((i % 20) / 5)].imshow(scan[i], cmap=plt.cm.bone)
  plt.show()

def get_segmented_lungs(im, plot=False):

  if plot == True:
    f, plots = plt.subplots(8, 1, figsize=(5, 40))
  binary = im < 604
  if plot == True:
    plots[0].axis('off')
    plots[0].imshow(binary, cmap=plt.cm.bone)
  cleared = clear_border(binary)
  if plot == True:
    plots[1].axis('off')
    plots[1].imshow(cleared, cmap=plt.cm.bone)
  label_image = label(cleared)
  if plot == True:
    plots[2].axis('off')
    plots[2].imshow(label_image, cmap=plt.cm.bone)
  areas = [r.area for r in regionprops(label_image)]
  areas.sort()
```

```
    if len(areas) > 2:
        for region in regionprops(label_image):
            if region.area < areas[-2]:
                for coordinates in region.coords:
                       label_image[coordinates[0], coordinates[1]] = 0
    binary = label_image > 0
    if plot == True:
        plots[3].axis('off')
        plots[3].imshow(binary, cmap=plt.cm.bone)
    selem = disk(2)
    binary = binary_erosion(binary, selem)
    if plot == True:
        plots[4].axis('off')
        plots[4].imshow(binary, cmap=plt.cm.bone)
    selem = disk(10)
    binary = binary_closing(binary, selem)
    if plot == True:
        plots[5].axis('off')
        plots[5].imshow(binary, cmap=plt.cm.bone)
    edges = roberts(binary)
    binary = ndi.binary_fill_holes(edges)
    if plot == True:
        plots[6].axis('off')
        plots[6].imshow(binary, cmap=plt.cm.bone)
    get_high_vals = binary == 0
    im[get_high_vals] = 0
    if plot == True:
        plots[7].axis('off')
        plots[7].imshow(im, cmap=plt.cm.bone)

    plt.show()
    return im

def segment_lung_from_ct_scan(ct_scan):
    return np.asarray([get_segmented_lungs(slice) for slice in ct_scan])

ct_scan = read_ct_scan('FYP/sample_images/00cba091fa4ad62cc3200a657aeb957e/')

plot_ct_scan(ct_scan)

get_segmented_lungs(ct_scan[71], True)

segmented_ct_scan = segment_lung_from_ct_scan(ct_scan)
plot_ct_scan(segmented_ct_scan)

segmented_ct_scan[segmented_ct_scan < 604] = 0
plot_ct_scan(segmented_ct_scan)

selem = ball(2)
binary = binary_closing(segmented_ct_scan, selem)

label_scan = label(binary)

areas = [r.area for r in regionprops(label_scan)]
areas.sort()
```

```
for r in regionprops(label_scan):
    max_x, max_y, max_z = 0, 0, 0
    min_x, min_y, min_z = 1000, 1000, 1000

    for c in r.coords:
        max_z = max(c[0], max_z)
        max_y = max(c[1], max_y)
        max_x = max(c[2], max_x)

        min_z = min(c[0], min_z)
        min_y = min(c[1], min_y)
        min_x = min(c[2], min_x)
    if (min_z == max_z or min_y == max_y or min_x == max_x or r.area > areas[-3]):
        for c in r.coords:
            segmented_ct_scan[c[0], c[1], c[2]] = 0
    else:
        index = (max((max_x - min_x), (max_y - min_y), (max_z - min_z))) / (min((max_x - min_x), (max_y
- min_y) , (max_z - min_z)))

def plot_3d(image, threshold=-300):

    p = image.transpose(2,1,0)
    p = p[:,:,::-1]

    verts, faces = measure.marching_cubes_classic(p, threshold)

    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')

    mesh = Poly3DCollection(verts[faces], alpha=0.1)
    face_color = [0.5, 0.5, 1]
    mesh.set_facecolor(face_color)
    ax.add_collection3d(mesh)

    ax.set_xlim(0, p.shape[0])
    ax.set_ylim(0, p.shape[1])
    ax.set_zlim(0, p.shape[2])

    plt.show()

plot_3d(segmented_ct_scan, 604)
```

The following is a code attachment for the code used for CNN architecture.

```
import tensorflow as tf
import numpy as np

imgSize = 50
sliceCount = 20
classes = 2

x = tf.placeholder('float')
y = tf.placeholder('float')

keep_rate = 0.8
keep_prob = tf.placeholder(tf.float32)

def conv3d(x, W):
    return tf.nn.conv3d(x, W, strides=[1,1,1,1,1], padding='SAME')

def maxpool3d(x):
    return tf.nn.max_pool3d(x, ksize=[1,2,2,2,1], strides=[1,2,2,2,1], padding='SAME')

def convolutional_neural_network(x):
    weights = {'W_conv1':tf.Variable(tf.random_normal([3,3,3,1,32])),
               'W_conv2':tf.Variable(tf.random_normal([3,3,3,32,64])),
               'W_fc':tf.Variable(tf.random_normal([54080,1024])),
               'out':tf.Variable(tf.random_normal([1024, classes]))}

    biases = {'b_conv1':tf.Variable(tf.random_normal([32])),
               'b_conv2':tf.Variable(tf.random_normal([64])),
               'b_fc':tf.Variable(tf.random_normal([1024])),
               'out':tf.Variable(tf.random_normal([classes]))}

    x = tf.reshape(x, shape=[-1, imgSize, imgSize, sliceCount, 1])

    conv1 = tf.nn.relu(conv3d(x, weights['W_conv1']) + biases['b_conv1'])
    conv1 = maxpool3d(conv1)

    conv2 = tf.nn.relu(conv3d(conv1, weights['W_conv2']) + biases['b_conv2'])
    conv2 = maxpool3d(conv2)

    fc = tf.reshape(conv2,[-1, 54080])
    fc = tf.nn.relu(tf.matmul(fc, weights['W_fc'])+biases['b_fc'])
    fc = tf.nn.dropout(fc, keep_rate)

    output = tf.matmul(fc, weights['out'])+biases['out']

    return output

muchData = np.load('preprocessedimages-50-50-20.npy')
print len(muchData)
trainData = muchData[:-2]
devData = muchData[-2:]
print len(trainData)
print len(devData)

def train_neural_network(x):
    prediction = convolutional_neural_network(x)
    cost = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(logits=prediction,labels=y) )
    optimizer = tf.train.AdamOptimizer().minimize(cost)

    hm_epochs = 20
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        for epoch in range(hm_epochs):
            epoch_loss = 0
            success = 0
            attempt = 0
            for data in trainData:
                attempt+=1
                try:
                    X = data[0]
                    Y = data[1]
                    _, c = sess.run([optimizer, cost], feed_dict={x: X, y: Y})
                    epoch_loss += c
                    success+=1
                except Exception as e:
                    pass

            print('Epoch', epoch, 'completed out of',hm_epochs,'loss:',epoch_loss,'success rate',success/attempt)

        correct = tf.equal(tf.argmax(prediction, 1),tf.argmax(y, 1))

        accuracy = tf.reduce_mean(tf.cast(correct,'float'))
        print('Accuracy:',accuracy.eval({x:[i[0] for i in devData], y:[i[1] for i in devData]}))

train_neural_network(x)
```