PULMONARY NODULES DETECTION USING WATERSHED ALGORITHM AND CNN

A Major Project Work

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

G. PRANAV	**18H61A0447**
S. LIKHITHA	**18H61A0453**
G. SHIREESHA	**19H65A0401**
I. SRAVANI	**18H61A0425**

Under the guidance of
Dr. Raghu Indrakanti
Assistant Professor
Department of ECE

Department of Electronics and Communication Engineering

ANURAG GROUP OF INSTITUTIONS

AUTONOMOUS

SCHOOL OF ENGINEERING

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad)

Venkatapur (V), Ghatkesar (M), Medchal-Malkajgiri Dist-500088

2021-2022

ANURAG GROUP OF INSTITUTIONS

AUTONOMOUS

SCHOOL OF ENGINEERING

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad

Venkatapur (V), Ghatkesar (M), Medchal-Malkajgiri Dist-500088

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the project report entitled **"Pulmonary Nodules Detection using Watershed Algorithm and CNN"** being submitted by

G. Pranav	18H61A0447
S. Likhitha	18H61A0453
G. Shireesha	19H65A0401
I. Sravani	18H61A0425

in partial fulfillment for the award of the Degree of Bachelor of Technology in Electronics & Communication Engineering to the Jawaharlal Nehru Technological University, Hyderabad is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Dr. Raghu Indrakanti
Assistant Professor

Dr. S. Satheesh Kumaran
Head of the Department
Department of ECE

External Examiner

ACKNOWLEDGEMENT

This project is an acknowledgement to the inspiration, drive and technical assistance contributed by many individuals. This project would have never seen light of this day without the help and guidance we have received. We would like to express our gratitude to all the people behind the screen who helped us to transform an idea into a real application.

It's our privilege and pleasure to express our profound sense of gratitude to **Dr. Raghu Indrakanti, Assistant Professor**, Department of ECE for his guidance throughout this dissertation work.

We express our sincere gratitude to **Dr.S. Satheesh Kumaran, Head of Department**, Electronics and Communication Engineering for his precious suggestions for the successful completion of this project. He is also a great source of inspiration to our work.

We would like to express our deep sense of gratitude to **Dr. G. Vishnu Murthy, Director,** Anurag Group of Institutions for his tremendous support, encouragement and inspiration. Lastly, we thank almighty, our parents, friends for their constant encouragement without which this assignment would not be possible. We would like to thank all the other staff members, both teaching and non- teaching, which have extended their timely help and eased my work.

BY

G. Pranav	**18H61A0447**
S. Likhitha	**18H61A0453**
G. Shireesha	**19H65A0401**
I. Sravani	**18H61A0425**

DECLARATION

We hereby declare that the result embodied in this project report entitled **“Pulmonary Nodules Detection using Watershed Algorithm and CNN”** is carried out by us during the year 2021-2022 for the partial fulfilment of the award of **Bachelor of Technology** in, **Electronics and Communication Engineering**, from ANURAG GROUP OF INSTITUTIONS. We have not submitted this project report to any other Universities / Institute for the award of any degree.

BY

G. Pranav	**18H61A0447**
S. Likhitha	**18H61A0453**
G. Shireesha	**19H65A0401**
I. Sravani	**18H61A0425**

ABSTRACT

Lung cancer is one of the most dangerous cancers in the world, causing more than 2 million deaths per year. Cancerous (malignant) and noncancerous (benign) pulmonary nodules are the small growths of cells inside the lung. Detection of malignant lung nodules at an early stage is necessary for the crucial prognosis. Computed Tomography (CT) scan can provide valuable information in the diagnosis of lung diseases. A dynamic threshold segmentation method was used to identify lung nodule in CT images. Then, an improved watershed method was used to mark suspicious areas on the CT image. The main objective of this project is to detect the pulmonary nodules from the given input CT scan image and to classify the lung cancer. To detect the pulmonary nodules, watershed algorithm and Convolutional Neural Network (CNN) model is used. The CNN architecture consists of eight layers: one input layer, three convolutional layers and two sub-sampling layers intercepted with ReLu and max-pooling for salient feature extraction, and one fully connected layer that uses sigmoid function as output layer. A total of 1658 images with an augmented image data store are used to enhance the performance of the network in the study generating high sensitivity scores with good true positives. Our proposed CNN achieved the classification accuracy of 98.1%, an average specificity of 93%, and an average sensitivity of 93.4% with reduced false positives.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CT	: Computed Tomography
MRI	: Magnetic Resonance Imaging
PET	: Positron Emission Tomography
ABF	: Adaptive Bilateral Filter
GPU	: Graphics processing Unit
TPU	: Tensor Processing Unit
CNN	: Convolutional Neural Network
ReLU	: Rectified Linear Unit
SGD	: Stochastic Gradient Descent
ROI	: Region of Interest
ROC	: Receiver Operating Characteristics
AOC	: Area under the ROC Curve

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

Cancer is one of the major public health issues spread worldwide leading to deaths with high mortality due to unclear clinical examinations. Risk factors causing cancer are due to the consumption of tobacco, smoking, biological and chemical reactions (exposure to radon gas), and environmental conditions (exposure to secondhand smoke). Radiologists recommend different imaging modalities for detecting pulmonary nodule regions, such as computed tomography (CT), magnetic resonance imaging (MRI), and positron emission tomography (PET).

Worldwide in 2020, lung cancer occurred in 2.2 million people and resulted in 1.8 million deaths. It is the most common cause of cancer-related death in both men and women. The most common age at diagnosis is 70 years. In most countries the five-year survival rate is around 10 to 20%, while in Japan it is 33%, in Israel 27%, and in the Republic of Korea 25%. Outcomes typically are worse in the developing world.

The most commonly preferred imaging modality is the CT scans by the radiologists. CT scans are popular due to their advantages with respect to cost, availability, and rapid acquisition of scans across complete lung sections. In recent years, the mortality of lung cancer has decreased by around 20% with low-dose CT images as reported by the National Lung Screening Trial. Patients with lung cancer symptoms undergo CT scans to distinguish certain abnormal growth observed in the lungs. More exceptionally, sensitively identifying the little knobs (cancerous cells) is a trivial task as the nodules may be attached to vessels or the walls of the chest or false positively considered as irregularly shaped due to noise.

The pulmonary nodules are irregularly shaped growth in the lungs with its diameter measured up to 3 mm in the chest region. Also, the pulmonary nodules are categorized based on its shape (round, irregular shaped), size (small or large), location (vascular or pleural regions), texture (solid or non-solid), and so on [1]. As the patient receives the CT scans in a clinical laboratory, the radiologists evaluate and detect the suspicious nodule from the numerous CT images. Based on the possibility of malignancy examined through the nodule information (density, morphology, texture features), their diagnosis is acknowledged with an appropriate treatment plan [2]. The task in identifying the nodules is rigorous. Inappropriate professional experience, distraction, fatigue while capturing scans, etc., may destabilize nodule detection contributing to misinterpretations of false positives with the available data.

1.1.1 LUNG CANCER

Lung cancer, also known as lung carcinoma, since about 98–99% of all lung cancers are carcinomas is a malignant lung tumour characterized by uncontrolled cell growth in tissues. Lung carcinomas derive from transformed, malignant cells that originate as epithelial cells, or from tissues composed of epithelial cells. Other lung cancers, such as the rare sarcomas of the lung, are generated by the malignant transformation of connective tissues (i.e. nerve, fat, muscle, bone), which arise from mesenchymal cells. Lymphomas and melanomas (from lymphoid and melanocyte cell lineages) can also rarely result in lung cancer.

In time, this uncontrolled growth can spread beyond the lung – either by direct extension, by entering the lymphatic circulation, or via the hematogenous, bloodborne spread – the process called metastasis – into nearby tissue or other, more distant parts of the body. Most cancers that start in the lung, known as primary lung cancers, are carcinomas. The most common symptoms are coughing (including coughing up blood), weight loss, shortness of breath, and chest pains.

The vast majority (85%) of cases of lung cancer are due to long-term tobacco smoking. About 10–15% of cases occur in people who have never smoked. These cases are often caused by a combination of genetic factors and exposure   to radon gas, asbestos, second-hand smoke, or other forms of air pollution. Lung cancer may be seen on chest radiographs and computed tomography (CT) scans. The diagnosis is confirmed by biopsy, which is usually performed by bronchoscopy or CT-guidance. The major method of prevention is the avoidance of risk factors, including smoking and air pollution. Treatment and long-term outcomes depend on the type of cancer, the stage (degree of spread), and the person's overall health. Most cases are not curable. Common treatments include surgery, chemotherapy, and radiotherapy. NSCLC is sometimes treated with surgery, whereas SCLC usually responds better to chemotherapy and radiotherapy.

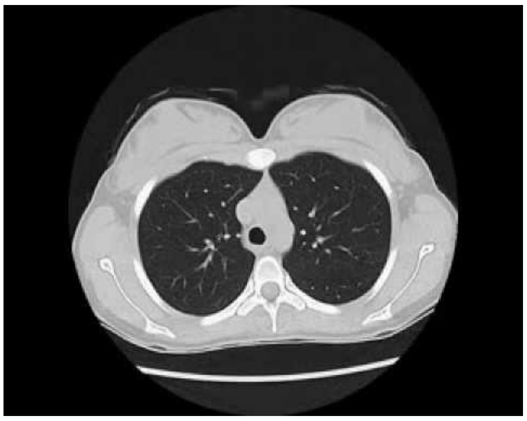


Fig.1.1.1(a) Normal lung CT scan image

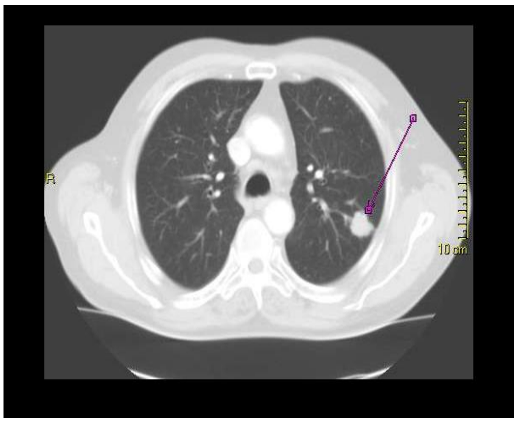


Fig.1.1.1(b) CT scan showing a cancerous tumor in the left lung

1.1.2 LUNG NODULE

A lung nodule (or mass) is a small abnormal area that is sometimes found during a CT scan of the chest. These scans are done for many reasons, such as part of lung cancer screening, or to check the lungs if you have symptoms. Most lung nodules seen on CT scans are not cancer. Most lung nodules are benign (not cancerous). Rarely, pulmonary nodules are a sign of lung cancer. About 95% of lung nodules are benign. Many things can cause benign lung nodules, including infections and scarring.

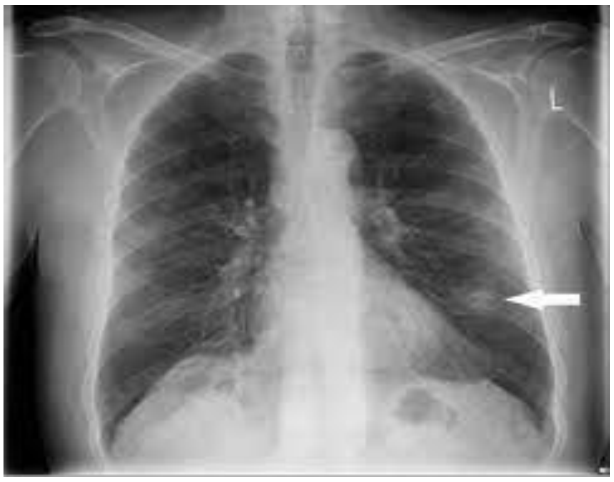


Fig1.1.2 Chest X-Ray showing lung nodule

Commonly called a “spot on the lung” or a “shadow,” a nodule is a round area that is more dense than normal lung tissue. It shows up as a white spot on a CT scan. Lung nodules are usually caused by scar tissue, a healed infection that may never have made you sick, or some irritant in the air. Sometimes, a nodule can be an early lung cancer. If you have a pulmonary nodule, your healthcare provider may want to perform additional tests to determine the cause and rule out lung cancer. Lung nodules show up on imaging scans like

X-rays or CT scans. Your healthcare provider may refer to the growth as a spot on the lung, coin lesion or shadow. When an infection or illness inflames lung tissue, a small clump of cells (granuloma) can form. Over time, a granuloma can calcify or harden in the lung, causing a noncancerous lung nodule. A neoplasm is an abnormal growth of cells in the lung. Neurofibromas are a type of noncancerous neoplasm. Types of malignant (cancerous) neoplasms include lung cancer and carcinoid tumors.

1.1.3 CAUSES

Smoking

Tobacco smoking is by far the main contributor to lung cancer. Cigarette smoke contains at least 73 known carcinogens, including benzo pyrene, NNK, 1,3-butadiene, and a radioactive isotope of polonium – polonium-210. Across the developed world, 90% of lung cancer deaths in men and 70% of those in women during 2000 were attributed to smoking. Smoking accounts for about 85% of lung cancer cases. Vaping may be a risk factor for lung cancer, but less than that of cigarettes, and further research is necessary due to the length of time it can take for lung cancer to develop following an exposure to carcinogens.

Radon gas

Radon is a colorless and odorless gas generated by the breakdown of radioactive radium, which in turn is the decay product of uranium, found in the Earth's crust. The radiation decay products ionize genetic material, causing mutations that sometimes become cancerous. Radon is the second-most common cause of lung cancer in the US, causing about 21,000 deaths each year. The risk increases 8–16% for every 100 Bq/m^3 increase in the radon concentration. Radon gas levels vary by locality and the composition of the underlying soil and rocks. About one in 15 homes in the US has radon levels above the recommended guideline of 4 picocuries per liter (pCi/l) (148 Bq/m^3).

Asbestos

Asbestos can cause a variety of lung diseases such as lung cancer. Tobacco smoking and exposure to asbestos together have synergistic effects on the development of lung cancer. In smokers who work with asbestos, the risk of lung cancer is increased 45-fold compared to the general population. Asbestos can also cause cancer of the pleura, called mesothelioma – which actually is different from lung cancer.

Air pollution

Outdoor air pollutants, especially chemicals released from the burning of fossil fuels, increase the risk of lung cancer. Fine particulates ($PM_{2.5}$) and sulfate aerosols, which may be released in traffic exhaust fumes, are associated with a slightly increased risk. For nitrogen dioxide, an incremental increase of 10 parts per billion increases the risk of lung cancer by 14%. Outdoor air pollution is estimated to cause 1–2% of lung cancers.

Tentative evidence supports an increased risk of lung cancer from indoor air pollution in relation to the burning of wood, charcoal, dung, or crop residue for cooking and heating. Women who are exposed to indoor coal smoke have roughly twice the risk, and many of the by-products of burning biomass are known or suspected carcinogens. This risk affects about 2.4 billion people worldwide, and it is believed to result in 1.5% of lung cancer deaths.

Genetics

About 8% of lung cancer cases are caused by inherited (genetic) factors. In relatives of people who are diagnosed with lung cancer, the risk is doubled, likely due to a combination of genes. Polymorphisms on chromosomes 5, 6, and 15 have been identified and are associated with an increased risk of lung cancer. Single-nucleotide polymorphisms of the genes encoding the nicotinic acetylcholine receptor (nAChR) – CHRNA5, CHRNA3, and CHRNB4 – are of those associated with an increased risk of lung cancer, as well as RGS17 – a gene regulating G-protein signaling. Newer genetic studies, have identified 18 susceptibility loci achieving genome-wide significance. These loci highlight a heterogeneity in genetic susceptibility across the histological subtypes of lung cancer, again identifying the cholinergic nicotinic receptors, e.g. CHRNA2.

1.2 AIM OF THE PROJECT

- This project aims to perform segmentation of lungs using watershed algorithm.
- To classify the lung nodules by using proposed CNN model.
- To Reduce false positives and false negatives.
- To improve accuracy with SGD and Adam optimizers.

CHAPTER 2
LITERATURE SURVEY

In the study, we incorporate an end-to-end convolutional neural network to automatically learn the features for the classification of pulmonary lung nodules as benign, malignant, or non-cancerous. The NROI is extracted, trained, and tested via deep learning using CNN. To compare its performance, CNNs are trained using images generated by GAN and are fined-tuned with actual input samples to differentiate and classify the lung nodules based on the classification strategy. In addition, CADx systems are implemented to extract the handcrafted texture, density, and morphological features. The methodology is compared with the state-of-the-art methods and traditional handcrafted methods for performance evaluation.

However, because diagnostic efficacy of SPNs with PET/CT is challenging, we propose a novel lung nodule detection method using PET/CT and an improved watershed algorithm. Compared with the existing methods, our proposed method detects SPNs with more sensitivity and differentiates malignant and benign SPNs with fewer false-positives.

In this work, first the input image is enhanced by using histogram equalization for image contrast and denoised by using Adaptive Bilateral Filter (ABF). After pre-processing, the next step is to find the lung region extraction. To extract the lung region, Artificial Bee Colony (ABC) segmentation approach is applied. The holes in the lung region are filled by using mathematical morphology technique in the ABC segmented image. After that the texture features are extracted to find the cancerous lung nodules. After finding the location of the cancerous lung nodules the next process is to classify the lung disease name and its severity based on the feature extraction. A new CNN method based on FPSO for reducing the computational complexity of CNN is proposed. FPSOCNN improves the efficiency of CNN.

CHAPTER 3
METHODOLOGY

3.1 Introduction

The overall proposed work for the lung sickness classification has been proven in Fig.1. The observe involves the layout and implementation of a convolutional neural network structure for mastering the feature maps of pics for lung nodule classification as benign, malignant, or non-cancerous. Training data (or a training dataset) is the initial data used to train machine learning models. Training datasets are fed to machine learning algorithms to teach them how to make predictions or perform a desired task. AI training data will vary depending on whether you're using supervised or unsupervised learning.

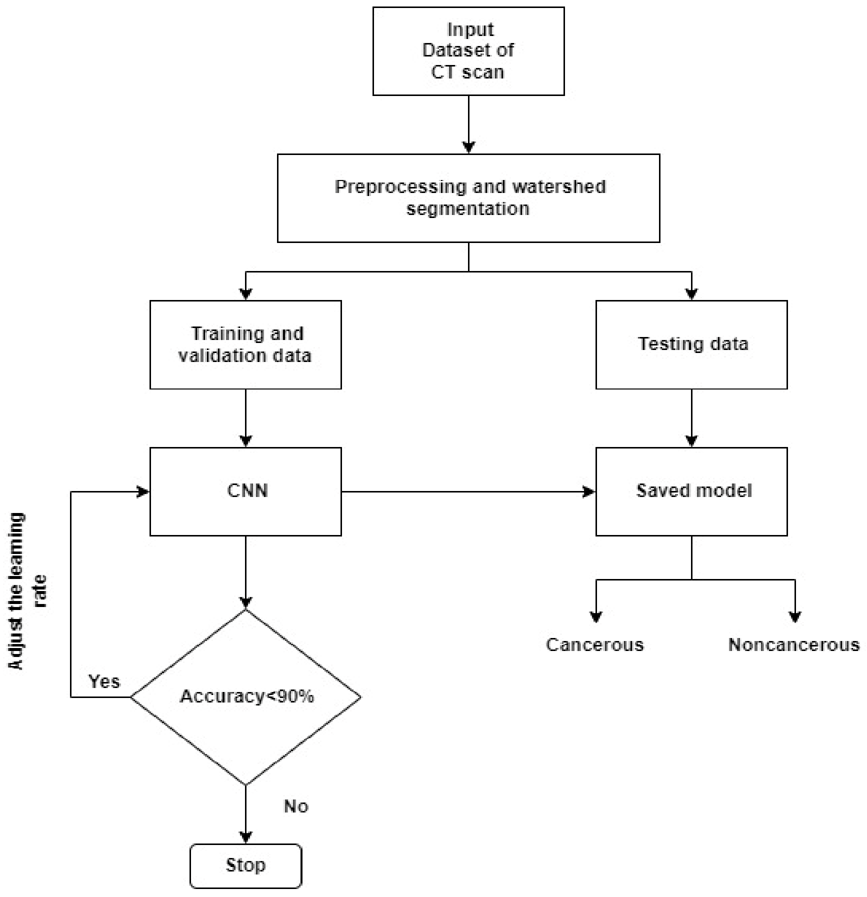


Fig.3.1 Proposed Model for Cancer Detection

Unsupervised learning uses unlabelled data. Models are tasked with finding patterns (or similarities and deviations) in the data to make inferences and reach conclusions. With supervised learning, on the other hand, humans must tag, label, or annotate the data to their criteria, in order to train the model to reach the desired conclusion (output). Labeled data is shown in the examples above, where the desired outputs are predetermined.

3.2 Pre-Processing

To begin with, for pre-processing, the assessment of input CT scan image is superior. The raw CT test pixel are preprocessed to beautify the fine and oftenly reduce the noisy artifacts that took place all through the photo acquisition process [3]. It locally recovers the evaluation of pictures through dividing the photograph into several sub areas and through remodeling the intensity values of every sub location independently to fulfill with a completely unique goal.

3.3 Watershed algorithms

Watershed algorithms are utilized in deep getting to know in the main for object segmentation, that is, for placing apart different objects in an picture. This permits for counting the objects or for further assessment of the separated objects. A watershed is a metamorphosis described on a grayscale image. The name refers metaphorically to a geological watershed, or drainage divide, which separates adjacent drainage basins. The watershed transformation treats the photograph it operates upon like a topographic map, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges.

"The topological watershed" was added by M. Couprie and G. Bertrand in 1997 starting from user-defined markers, the watershed algorithm treats pixel values as a local topography (elevation). The algorithm floods basins from the markers till basins attributed to unique markers meet on watershed lines. In many instances, markers are chosen as local minima of the photograph, from which basins are flooded. First, we extract inner and external markers from CT scan images with the assist of binary dilations and add them with a complete dark image using watershed strategies. And it gets rid of external noise from the image and gives a watershed marker of lungs and cancer cells.

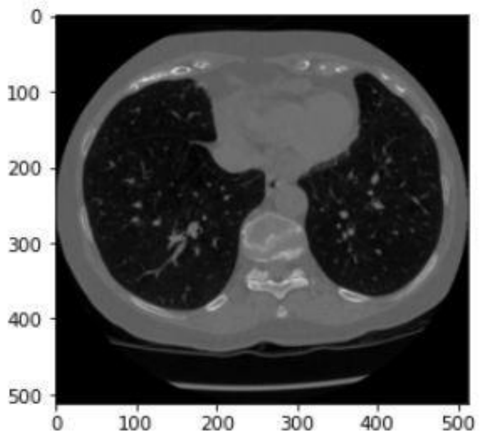


Fig.3.2(a) CT image of Lungs

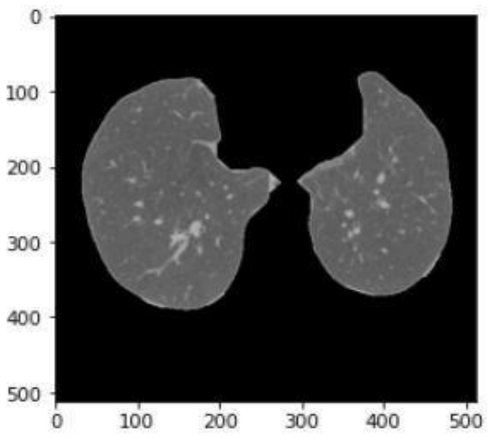


Fig.3.2(b) Segmentation of lungs using watershed algorithm

The watershed is a classical algorithm used for segmentation, that is, for separating different objects in an image. A watershed is a transformation defined on a grayscale image. The name refers metaphorically to a geological watershed, or drainage divide, which separates adjacent drainage basins. The watershed transformation treats the image it operates upon like a topographic map, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges.

Starting from user-defined markers, the watershed algorithm treats pixels values as a local topography (elevation). The algorithm floods basins from the markers until basins attributed to different markers meet on watershed lines. In many cases, markers are chosen as local minima of the image, from which basins are flooded.

First , we extract internal and external markers from CT scan images with the help of binary dilations and add them with a complete dark image using watershed methods. And it removes external noise from the image and gives a watershed marker of lungs and cancer cells. As we can see in the below figure watershed marker removes external noise and applies a binary mask on the image , black pixels in lungs represent cancer cells.

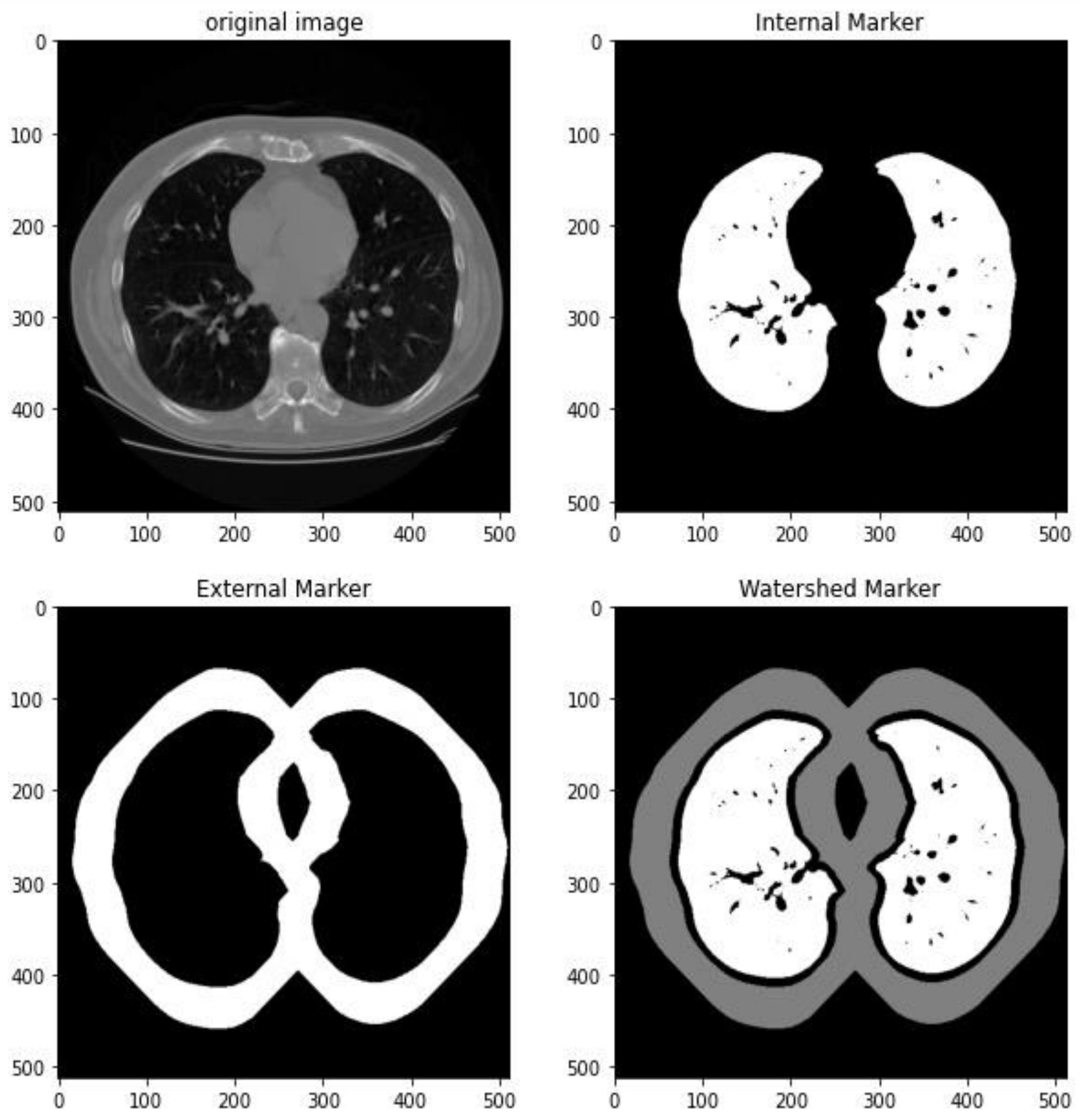


Fig 3.2(c) Different markers extracted from CT scan image using watershed algorithm

For better segmentation we integrate sobel filter with watershed algorithms. It removes the external layer of lungs. After removing the outer layer we use the internal marker and the Outline that was just created to generate the lung filter using bitwise or operations of numpy. It also removes the heart from CT scan images. Next step is to close off the lung filter with morphological operations and morphological gradients. It provides better segmented lungs than the previous process. We can see this process in the figure below.

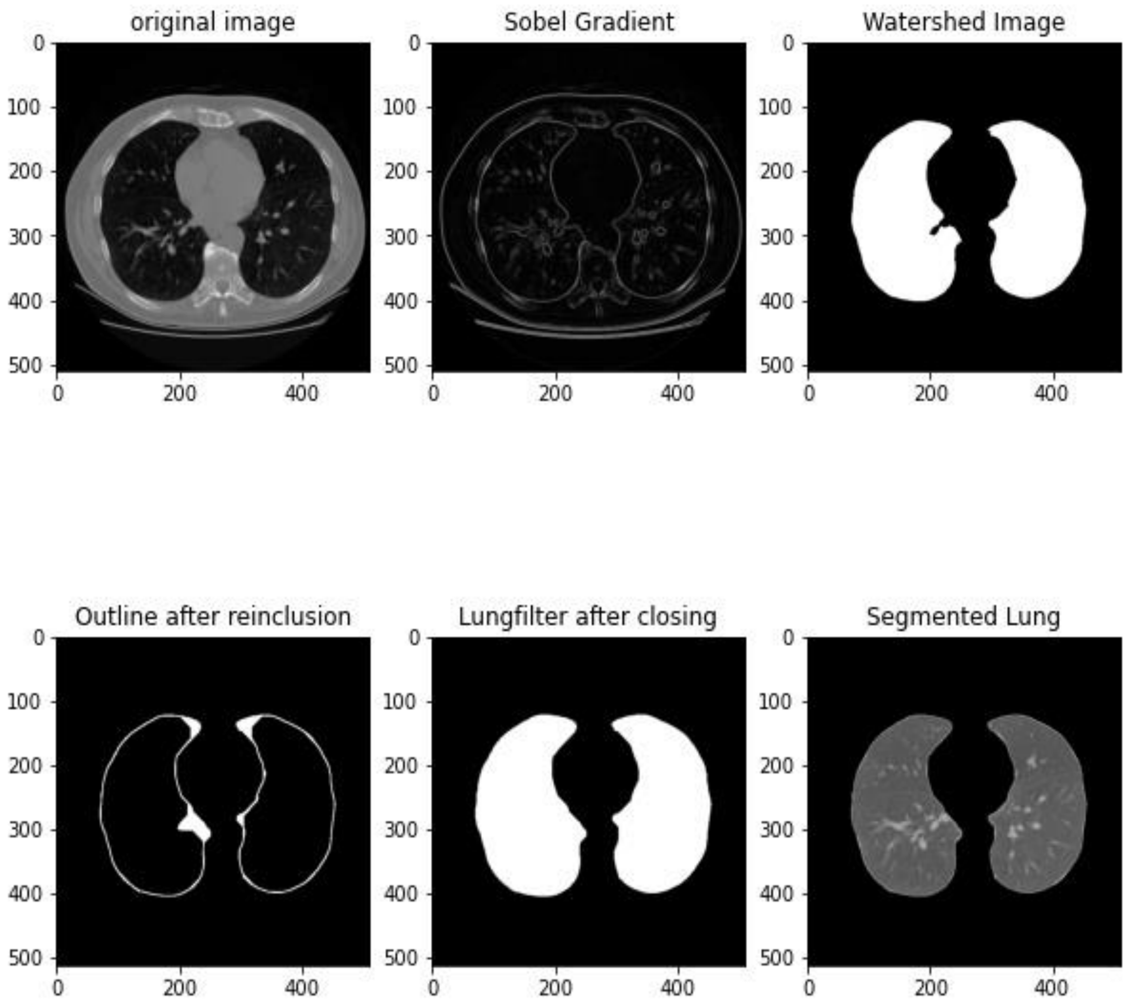


Fig 3.3 Image segmentation process visualization

By doing this we in total generated 1002 images with related labels which includes almost 12 patients CT scan data in which there are almost the same number of cancer and non-cancer patients.

3.4 Convolution Neural Network (CNN)

A CNN is a kind of Neural network commonly used for the object Detection. It helps to classify the diverse segment photos. CNN includes input layer, Hidden layer and output layer. input layer is associated with the neurons that helps the CNN for in addition processing. Hidden layer consists of Convolution layer, Pooling, completely connected layers. here the hidden layer is used for the function extraction of the phase photos. A feature map is produced through convolution layer through convolution of various sub regions of the input photograph with a found kernel. It enables in converting the records in to the 1D array.

The output layer used here is sigmoid it facilitates to classify whether cancer is present or not. CNN stocks weights inside the convolutional layer reducing the reminiscence footprint and increases the overall performance of the community. In CNN structure, typically the convolution layer and pool layer are used in a few mixture. The pooling layer typically carries out kinds of operations viz. max pooling means pooling. In advise pooling, the common community is calculated inside the feature points and in max pooling it is calculated inner a most of function factors. The segmented tumor areas are used to train CNN structure.

The Convolutional Neural Network (CNN) has shown excellent performance in many computer vision and machine learning problems. Many solid papers have been published on this topic, and quite some high quality open source CNN software packages have been made available. There are also well-written CNN tutorials or CNN software manuals. However, I believe that an introductory CNN material specifically prepared for beginners is still needed. Research papers are usually very terse and lack details. It might be difficult for beginners to read such papers. A tutorial targeting experienced researchers may not cover all the necessary details to understand how a CNN runs. This note tries to present a document that

- is self-contained. It is expected that all required mathematical background knowledge are introduced in this note itself (or in other notes for this course);
- has details for all the derivations. This note tries to explain all the necessary math in details. We try not to ignore an important step in a derivation. Thus, it should be possible for a beginner to follow (although an expert may feel this note tautological).
- ignores implementation details. The purpose is for a reader to understand how a CNN runs at the mathematical level. We will ignore those implementation details. In CNN, making correct choices for various implementation details is one of the keys to its high accuracy. After understanding, it is more advantageous to learn these implementation and design details with hands-on experience by playing with CNN programming. CNN is useful in a lot of applications, especially in image related tasks. Applications of CNN include image classification, image semantic segmentation, object detection in images, etc.

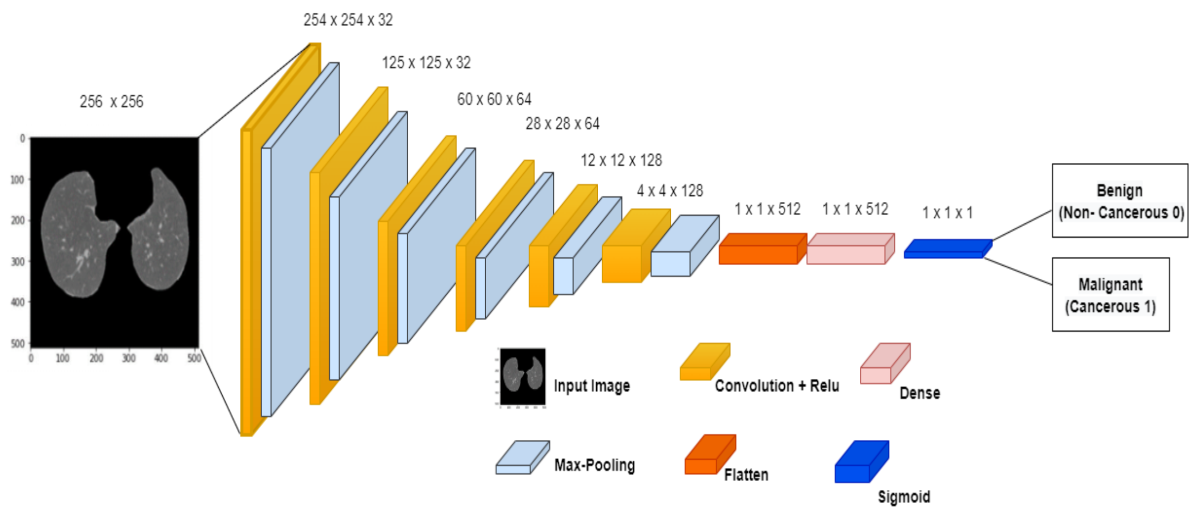


Fig.3.4 Proposed 2D-CNN Architecture for classification.

Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value. When you input an image in a ConvNet, each layer generates several activation functions that are passed on to the next layer.

The first layer usually extracts basic features such as horizontal or diagonal edges. This output is passed on to the next layer which detects more complex features such as corners or combinational edges. As we move deeper into the network it can identify even more complex features such as objects, faces, etc.

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a “class.” For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.

3.4.1 Convolutional Layer

A convolutional layer is the main building block of a CNN. It contains a set of filters (or kernels), parameters of which are to be learned throughout the training. The size of the filters is usually smaller than the actual image. Each filter convolves with the image and creates an activation map. For convolution the filter slid across the height and width of the image and the dot product between every element of the filter and the input is calculated at every spatial position.

A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image [4]. The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images.

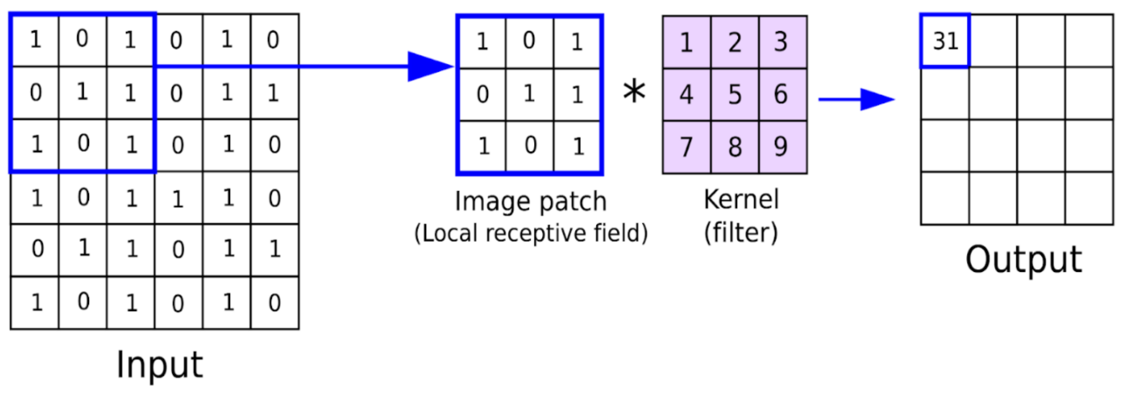


Fig 3.4.1(a) Convolution Operation

The convolutional neural network, or CNN for short, is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data. Central to the convolutional neural network is the convolutional layer that gives the network its name. This layer performs an operation called a “convolution“.

In the context of a convolutional neural network, a convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

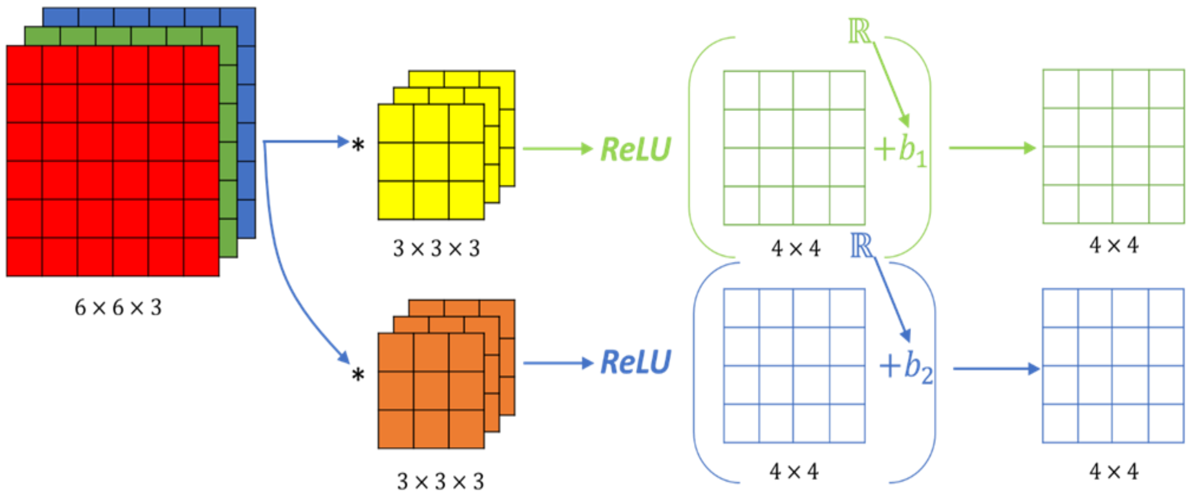


Fig 3.4.1(b) One layer of CNN

The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the “scalar product“.

3.4.2 ReLU

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input. The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

A neural network is comprised of layers of nodes and learns to map examples of inputs to outputs.For a given node, the inputs are multiplied by the weights in a node and summed together. This value is referred to as the summed activation of the node. The summed activation is then transformed via an activation function and defines the specific output or “activation” of the node.

The simplest activation function is referred to as the linear activation, where no transform is applied at all. A network comprised of only linear activation functions is very easy to train, but cannot learn complex mapping functions. Linear activation functions are still used in the output layer for networks that predict a quantity (e.g. regression problems).
Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data. Traditionally, two widely used nonlinear activation functions are the sigmoid and hyperbolic tangent activation functions.

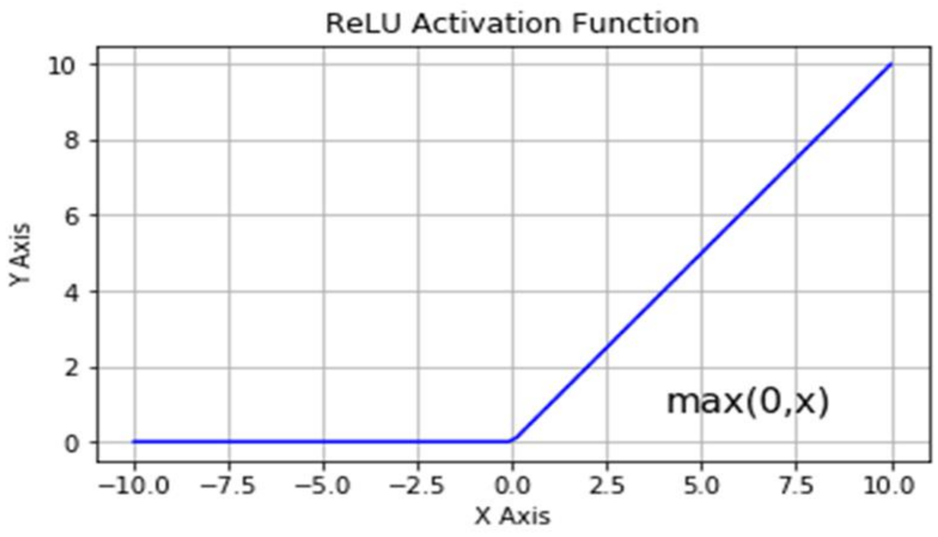


Fig.3.4.2 ReLU Activation Function

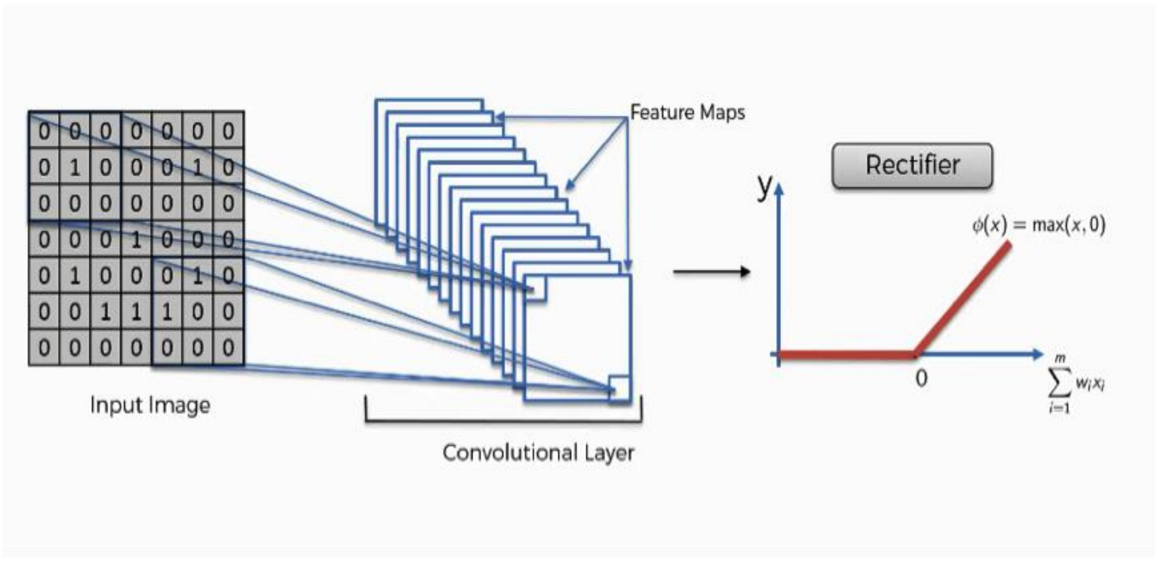


Fig.3.4.2 ReLU Activation Function

3.4.3 Drop Out

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to complex co-adaptations.

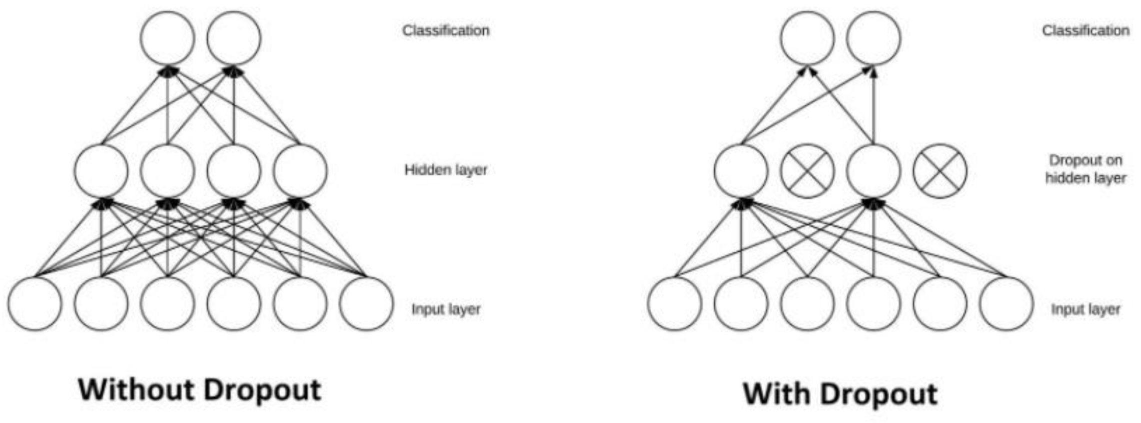


Fig.3.4.3(a) Dropout in CNN

You can imagine that if neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

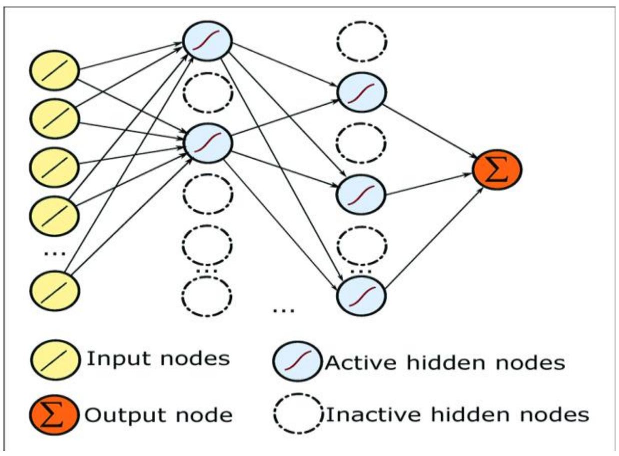


Fig.3.4.3(b) Drop Out

3.4.4 Max Pooling

A pooling layer is another building block of a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is max pooling. Max Pooling adds a bit of slight – Shift Invariance, Rotational Invariance, Scale Invariance. Slight change or shift does not cause invariance as we get max value from the 2 *2 image. This is called Shift invariance. Similarly, Max Pooling is slightly Rotational and scale-invariant.

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map. Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image. For example: in MNIST dataset, the digits are represented in white color and the background is black. So, max pooling is used.

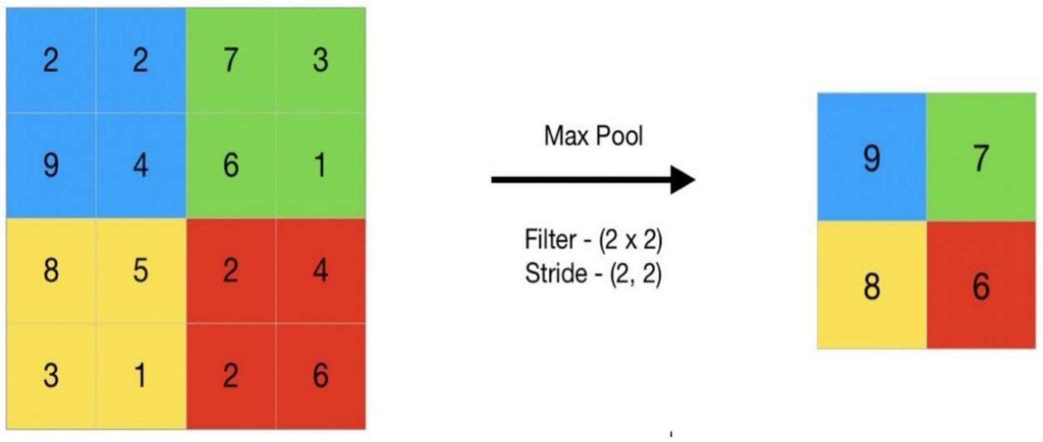


Fig.3.4.4 Max Pooling

3.4.5 Flattening

The flattening step is a refreshingly simple step involved in building a convolutional neural network. It involves taking the pooled feature map that is generated in the pooling step and transforming it into a one-dimensional vector. Here is a visual representation of what this process looks like. The reason why we transform the pooled feature map into a one-dimensional vector is because this vector will now be fed into an artificial neural network. Flatten function flattens the multi-dimensional input tensors into a single dimension, so you can model your input layer and build your neural network model, then pass those data into every single neuron of the model effectively. Is flattening layer necessary? No, this isn't specific to transfer learning. It is used over feature maps in the classification layer, that is easier to interpret and less prone to overfitting than a normal fully connected layer.

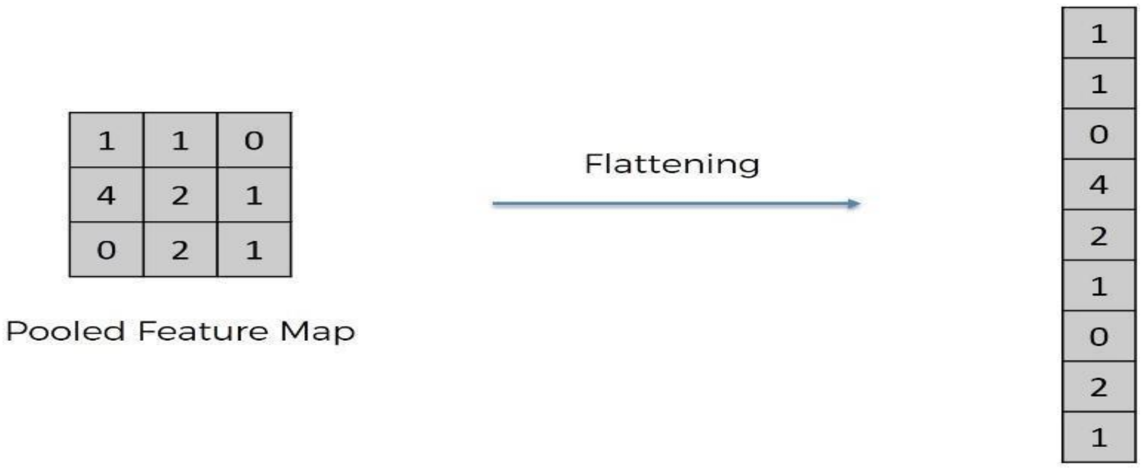


Fig.3.4.5 Conversion of Convoluted Layer to Flatten Layer.

3.4.6 Sigmoid Activation Function

Activation functions are a critical part of the design of a neural network. The choice of activation function in the hidden layer will control how well the network model learns the training dataset. The choice of activation function in the output layer will define the type of predictions the model can make. As such, a careful choice of activation function must be made for each deep learning neural network project. Activation functions are a key part of neural network design. The modern default activation function for hidden layers is the ReLU function. The activation function for output layers depends on the type of prediction problem.

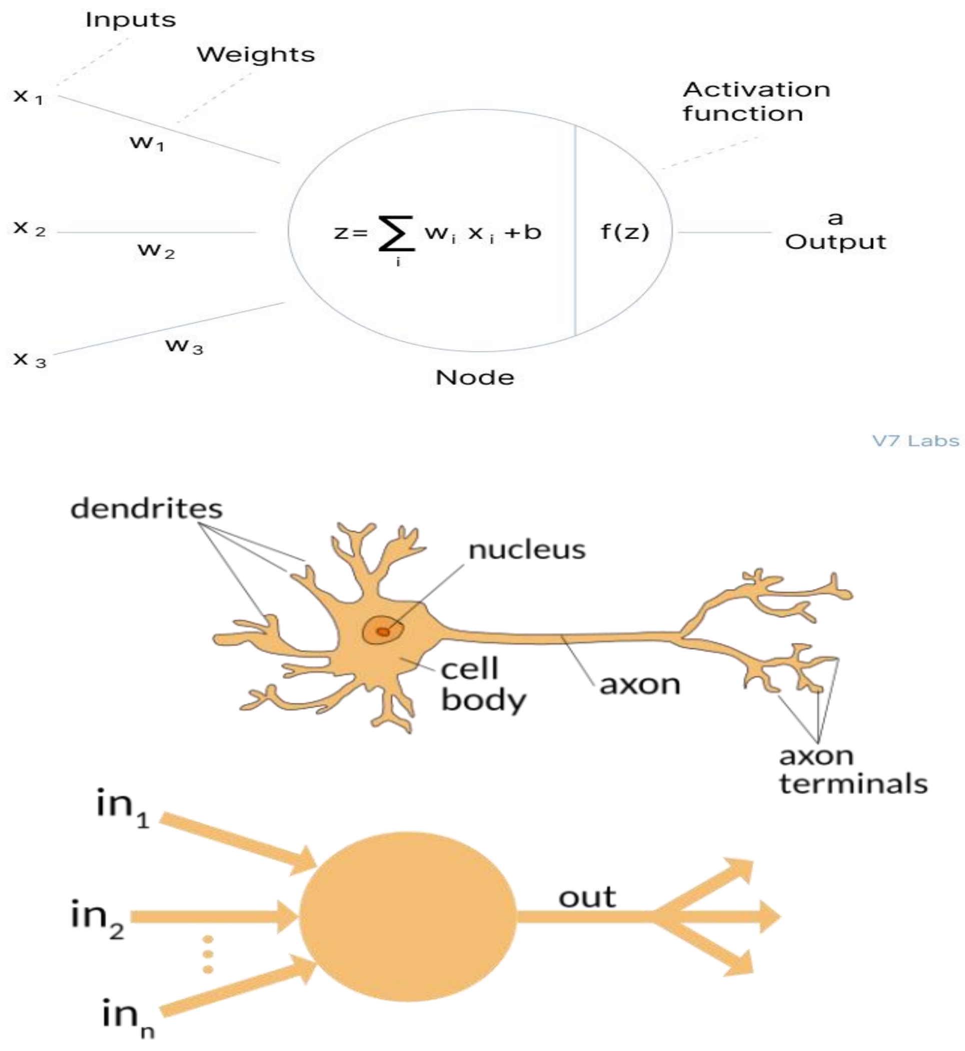


Fig.3.4.6(a) Representation of sigmoid activation function

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. Sometimes the activation function is called a “transfer function.” Many activation functions are nonlinear and may be referred to as the “nonlinearity” in the layer or the network design. The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model.

Technically, the activation function is used within or after the internal processing of each node in the network, although networks are designed to use the same activation function for all nodes in a layer. A network may have three types of layers: input layers that take raw input from the domain, hidden layers that take input from another layer and pass output to another layer, and output layers that make a prediction.

The sigmoid activation function, also called the logistic function, is traditionally a very popular activation function for neural networks. The input to the function is transformed into a value between 0.0 and 1.0 The shape of the function for all possible inputs is an S-shape from zero up through 0.5 to 1.0.The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

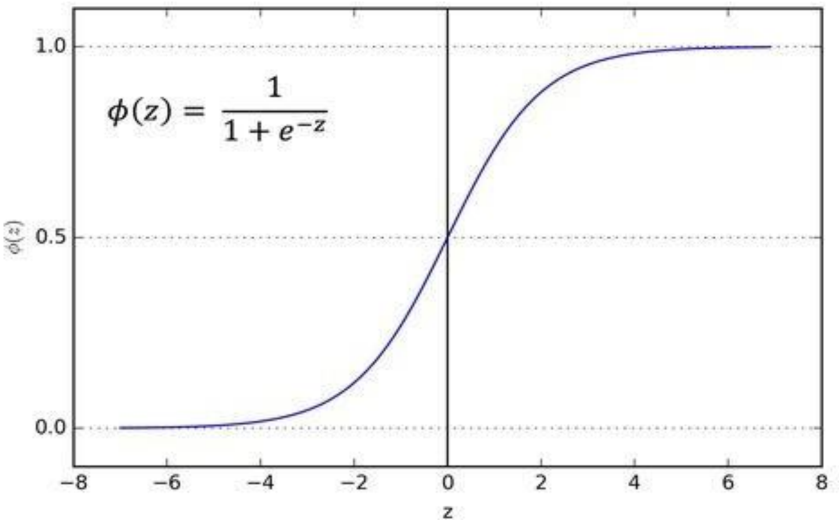


Fig.3.4.6(b) Sigmoid Activation Function

3.4.7 Optimization Method

Optimization is the process where we train the model iteratively that results in a maximum and minimum function evaluation. It is one of the most important phenomena in Machine Learning to get better results. Optimizers are algorithms or methods used to exchange the attributes of the neural network like weights and learning rate to lessen the losses.

We have trained the model with optimizers: Adam and Stochastic Gradient Descent (SGD). The learning rate for both optimizers is 0.001. Figure 4 and 5 indicates the model accuracy and model loss curve for Adam optimizer. Figure 6 and 7 indicates the model accuracy and model loss curve for SGD optimizer. Among the optimizers, Adam has the better learning curve as compared to the SGD. Adam is a famous algorithm in the field of deep learning. The results of the Adam optimizer commonly had faster computation time, and required fewer parameters for tuning. Due to all that, Adam is suggested as the default optimizer for most of the applications .

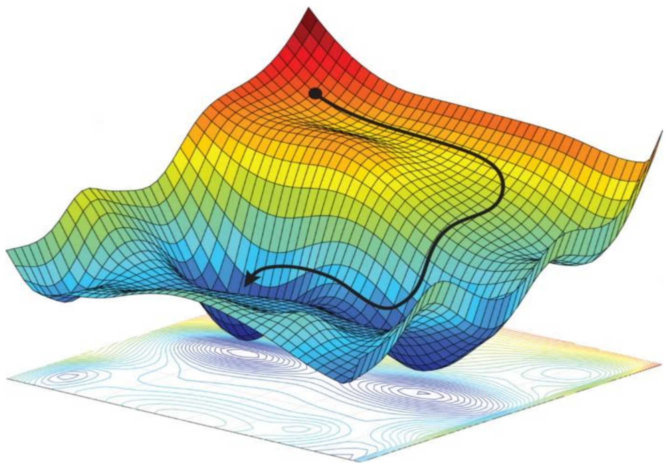


Fig.3.4.7(a) Optimization of Deep Neural Networks

Stochastic Gradient Descent

Gradient descent is the most common algorithm for model optimization for minimizing the error. In order to perform gradient descent, you have to iterate over the training dataset while re-adjusting the model. Our goal is to minimize the cost function because it means you get the smallest possible error and improve the accuracy of the model.

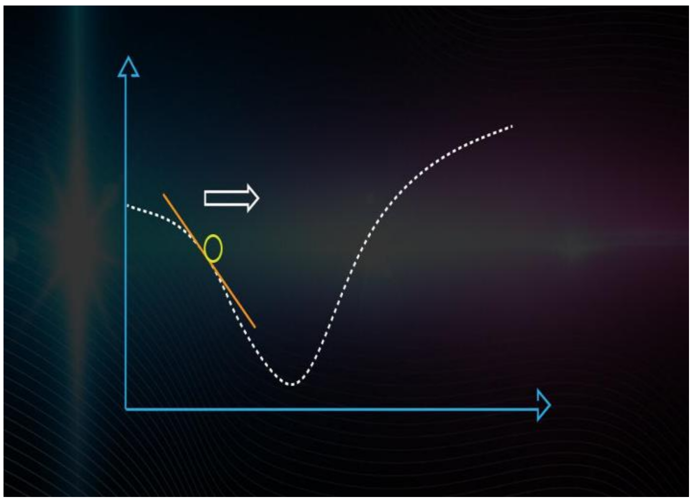


Fig.4.4.7(b) Graphical representation of gradient descent

Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). Especially in high-dimensional optimization problems this reduces the very high computational burden, achieving faster iterations in trade for a lower convergence rate.

While the basic idea behind stochastic approximation can be traced back to the Robbins–Monro algorithm of the 1950s, stochastic gradient descent has become an important optimization method in machine learning.

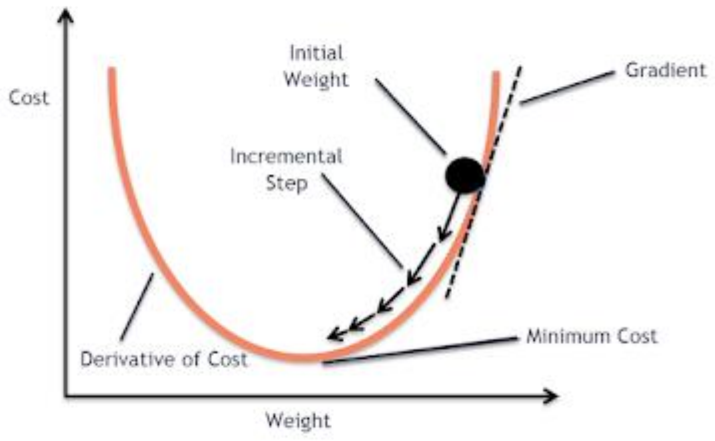


Fig.4.4.7(c) Graphical representation of stochastic gradient descent

Adam Optimization

The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

Benefits of using Adam on non-convex optimization problems, as follows:

- Straightforward to implement.
- Computationally efficient.
- Little memory requirements.
- Invariant to diagonal rescale of the gradients.
- Well suited for problems that are large in terms of data and/or parameters.
- Appropriate for non-stationary objectives.
- Appropriate for problems with very noisy/or sparse gradients.
- Hyper-parameters have intuitive interpretation and typically require little tuning.

Adam is a popular algorithm in the field of deep learning because it achieves good results fast.

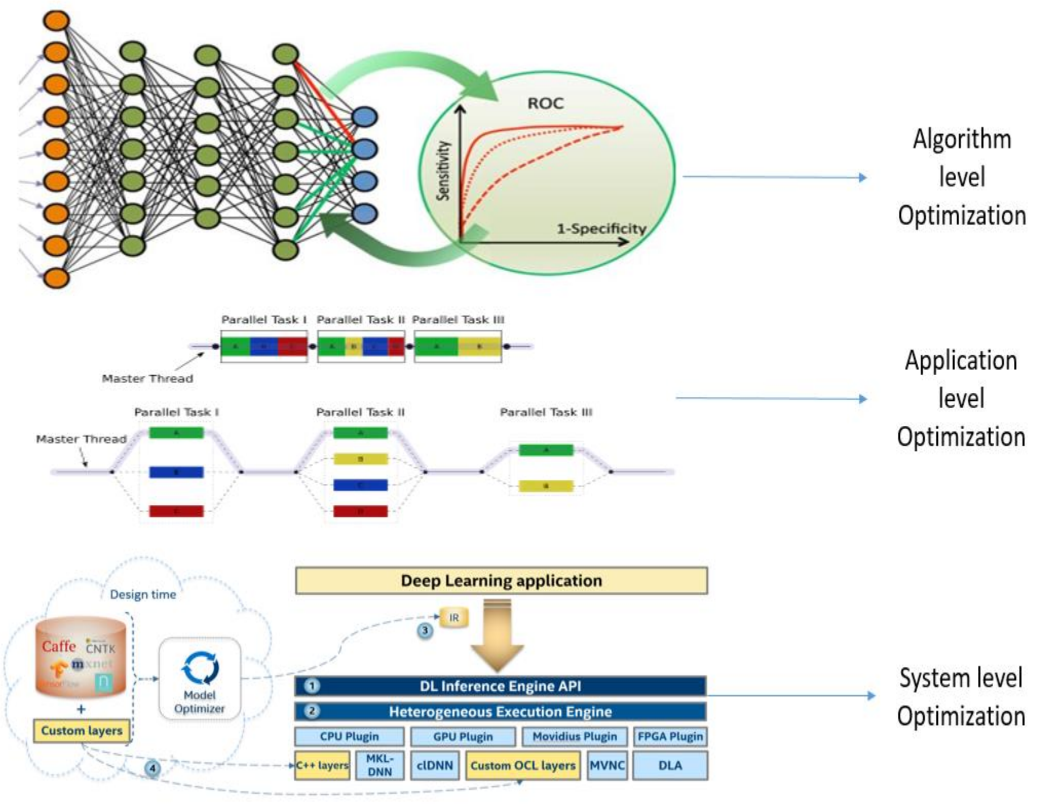


Fig.4.4.7(d) Adam Optimization Technique

Adam combines the benefits of two other stochastic gradient descent extensions and the Adaptive Gradient Algorithm (AdaGrad), which retains a learning speed per-parameter that improves performance on sparse gradients issues (e.g., natural language issues and computer vision issues). Root Mean Square Propagation (RMSProp) also preserves per-parameters learning rates adjusted to the weight based on the average of recent magnitudes. Offline and non-stationary problems, this algorithm does well.

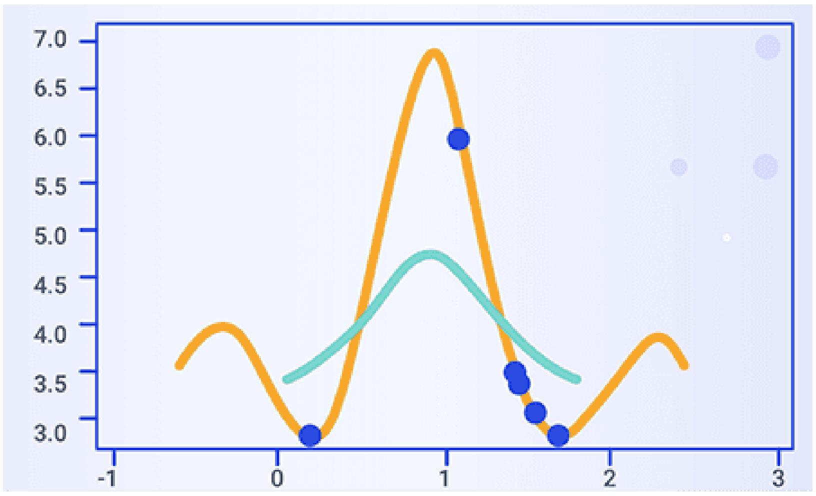


Fig.4.4.7(e) Adam Optimization Algorithm

Adam realizes the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates.

• Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models.

• Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

• Adam is relatively easy to configure where the default configuration parameters do well on most problems.

CHAPTER 4
SOFTWARE REQUIREMENTS

4.1 Jupyter Notebook

4.1.1 Introduction

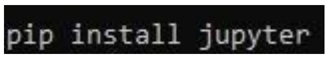
The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Its uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. Jupyter Notebooks are powerful, versatile, shareable and provide the ability to perform data visualization in the same environment. Jupyter Notebooks allow data scientists to create and share their documents, from codes to full blown reports Jupyter Book is an open-source project for building books and documents from computational material. It allows the user to construct the content in a mixture of Markdown, an extended version of Markdown called MyST, Maths & Equations using MathJax, Jupyter Notebooks, reStructuredText, the output of running Jupyter Notebooks at build time. Multiple output formats can be produced (currently single files, multipage HTML web pages and PDF files).

4.1.2 Installation Process

The best method of installing the Jupyter Notebooks is by the installation of the Anaconda package. The Jupyter Notebook and the Jupyter Lab comes pre-installed in the Anaconda package, and you don't have to install this on your own. One of the requirements here is Python, either Python 3.3 or greater or Python 2.7.

The general recommendation is that you use the Anaconda distribution to install both Python and the notebook application. The advantage of Anaconda is that you have access to over 720 packages that can easily be installed with Anaconda's conda, a package, dependency, and environment manager. However, if you are not interested in installing the Anaconda package manager, and you want to simply install the Jupyter Notebook the Pythonic way, make sure you have downloaded the latest Python Version (which is 3.9 at the time of writing of this article), and then proceed to issue the following command in the command prompt .

PowerShell you have downloaded the latest Python Version (which is 3.9 at the time of writing of this article), and then proceed to issue the following command in the command prompt or PowerShell.



```
pip install jupyter
```

With the installation of the Jupyter Notebook complete, you should be able to accessit successfully. Just type the jupyter notebook command in the command prompt or PowerShell. This should launch your Jupyter Notebook in your selected web browser with a local hosting URL, which is labeled ashttp://localhost:8888.

4.2 Python

4.2.1 Introduction

Python is a general purpose, dynamic, high level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures. Python is easy to plan yet powerful and versatile scripting language, which makes it attractive for Application Development. Python's syntax and dynamic timing with its interpreted nature make it an ideal language for scripting and rapid application development. Python supports multiple programming pattern, including object-oriented, imperative, and functional or procedural programming styles.

Python is not intended to work in a particular area, such as web programming. That is why it is known as multipurpose programming language because it can be used with web, enterprise, 3D CAD, etc. We don’t need to use data types to declare variable because it is dynamically typed so we can write a=10 to assign an integer value in an integer variable. Python makes the development and debugging fast because there is no compilation step included in Python development, and edit-test-debug cycle is very fast. Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances.

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a five-member "Steering Council" to lead the project.

Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector (in addition to reference counting) for memory management and support for Unicode. Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the utility, which automates the translation of Python 2 code to Python 3. Python 2.7's end-of- life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's end- of-life, only Python 3.6.x and later are supported. Python 3.9.2 and 3.8.8 were expedited as all versions of Python (including 2.7) had security issues, leading to possible remote code execution and web cache poisoning.

4.2.2 Libraries Required

4.2.2.1 TensorFlow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow was developed by the Google Brain team for internal Google use in research and production. The initial version was released under the Apache License 2.0 in 2015. Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019. TensorFlow can be used in a wide variety of programming languages, most notably Python, as well as JavaScript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors. Starting in 2011, Google Brain built DistBelief as a proprietary machine learning system based on deep learning neural networks. Google assigned multiple computer scientists, including Jeff Dean, to simplify and refactor the codebase of DistBelief into a faster, more robust application-grade library, which became TensorFlow. In 2009, the team, led by Geoffrey Hinton, had implemented generalized backpropagation and other improvements which allowed generation of neural networks with substantially higher accuracy, for instance a 25% reduction in errors in speech recognition.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google.

In December 2017, developers from Google, Cisco, RedHat, CoreOS, and CaiCloud introduced Kubeflow at a conference. Kubeflow allows operation and deployment of TensorFlow on Kubernetes. In March 2018, Google announced TensorFlow.js version 1.0 for machine learning in JavaScript. In Jan 2019, Google announced TensorFlow 2.0.] It became officially available in Sep 2019. In May 2019, Google announced TensorFlow Graphics for deep learning in computer graphics.

4.2.2.2 Keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. As of version 2.4, only TensorFlow is supported. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a

Google engineer. Chollet is also the author of the XCeption deep neural network model. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling. Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine.[3] It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU).

4.2.2.3 Pydicom

Pydicom is a pure Python package for working with DICOM files such as medical images, reports, and radiotherapy objects. pydicom makes it easy to read these complex files into natural pythonic structures for easy manipulation. Modified datasets can be written again to DICOM format files. Pydicom is not a DICOM server, and is not primarily about viewing images. It is designed to let you manipulate data elements in DICOM files with Python code. One limitation is that compressed pixel data (e.g. JPEG) can only be altered in an intelligent way if decompressing it first. Once decompressed, it can be altered and written back to a DICOM file the same way as initially uncompressed data.

4.2.2.4 Numpy

NumPy stands for ‘Numerical Python’. It is an open-source Python library used to perform variousmathematical and scientific tasks. It contains multi-dimensional arrays and matrices, along with many high-level mathematical functions that operate on these arrays and matrices.

NumPy library is an important foundational tool for studying Machine Learning. Many of its functions are very useful for performing any mathematical or scientific calculation. As it is known that mathematics is the foundation of machine learning, most of the mathematical tasks can be performed using NumPy.

4.2.2.5 Pandas

Pandas is an open-source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named, Numpy which provides support for multi-dimensional arrays. As one of the most popular data wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem, and is typically included in every Python distribution, from those that come with the operating system to commercial vendor distributions like Active State's Active Python.

4.2.2.6 Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

4.2.2.7 sklearn

Scikit-learn or sklearn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction. sklearn is used to build machine learning models. It should not be used for reading the data, manipulating and summarizing it. There are better libraries for that (e.g. NumPy, Pandas etc.) Scikit-learn comes loaded with a lot of features. Here are a few of them to help you understand the spread:

Supervised learning algorithms**:** Think of any supervised machine learning algorithm you might have heard about and there is a very high chance that it is part of scikit-learn. Starting from Generalized linear models (e.g., Linear Regression), Support Vector Machines (SVM), Decision Trees to Bayesian methods – all of them are part of scikit-learn toolbox. The spread of machine learning algorithms is one of the big reasons for the high usage of scikit-learn. I started using scikit to solve supervised learning problems and would recommend that to people new to scikit / machine learning as well.

Cross-validation: There are various methods to check the accuracy of supervised models on unseen data using sklearn.

Unsupervised learning algorithms: Again, there is a large spread of machine learning algorithms in the offering – starting from clustering, factor analysis, principal component analysis to unsupervised neural networks.

Various toy datasets: This came in handy while learning scikit-learn. I had learned SAS using various academic datasets (e.g., IRIS dataset, Boston House prices dataset). Having them handy while learning a new library helped a lot.

Feature extraction: Scikit-learn for extracting features from images and text (e.g. Bag of words)

4.2.3 DATASET

The Datasets used in this work is the Data science Bowl 2017 Lung cancer detection (DSB3). Inside the dataset, the target data referred to as 'stage1_labels.csv' incorporates 1398 labels and 1398 affected person id. The statistics is referred to as 'stage1' which incorporates 1595 patient folders, every folder incorporates above a hundred slices in Digital Imaging and Communications in Medicine (DICOM) layout. A DICOM record carries snap shots and statistics about the affected person's data. every CT image has a resolution of 512 x 512. In target data, 1035 patients no longer having cancer and 362 cancer patients having cancer. In this work, 1638 CT snap shots are generated with associated labels which incorporates nearly 17 patients. Every patient contains one hundred slices. The complete information is splitted into 80% of the data in the training, 10% of the data inside the validation and 10% of the data within the testing.

CHAPTER 5
RESULTS & DISCUSSION

The neural network based on convolutional and watershed segmentation has been implemented in Python and the system is trained with sample data sets for the model to understand and familiarize the lung cancer. A sample image has been fed as an input to the trained model and the model at this stage is able to tell the presence of cancer and locate the cancer spot in the sample image of a lung cancer. The process involves the feeding the input image, preprocessing, feature extraction, identifying the cancer spot and indicate the results to the user.

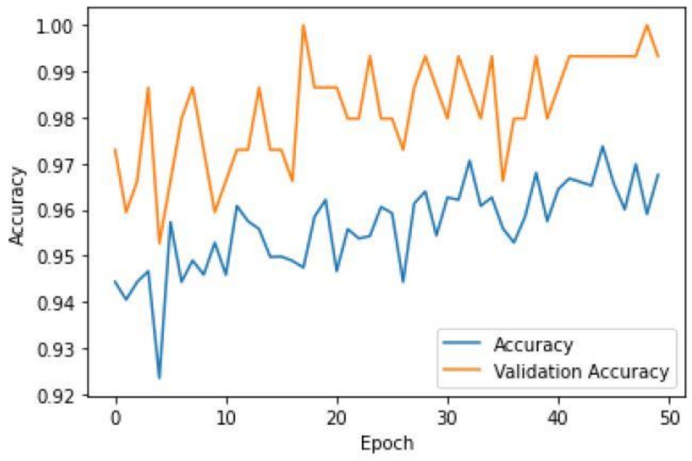


Fig.5(c) Training accuracy vs Validation accuracy using SGD

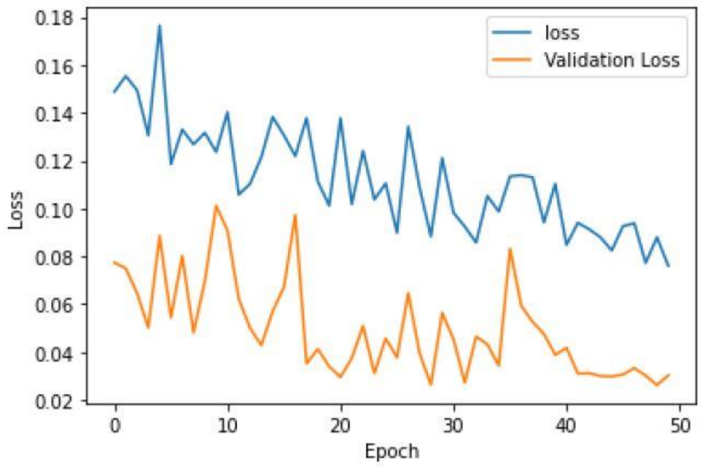


Fig.5(d) Training loss vs Validation loss using SGD

The CNN is trained with the maximum accuracy of 98.17% by using Adam optimizer. The learning curve of the CNN is shown in figure.

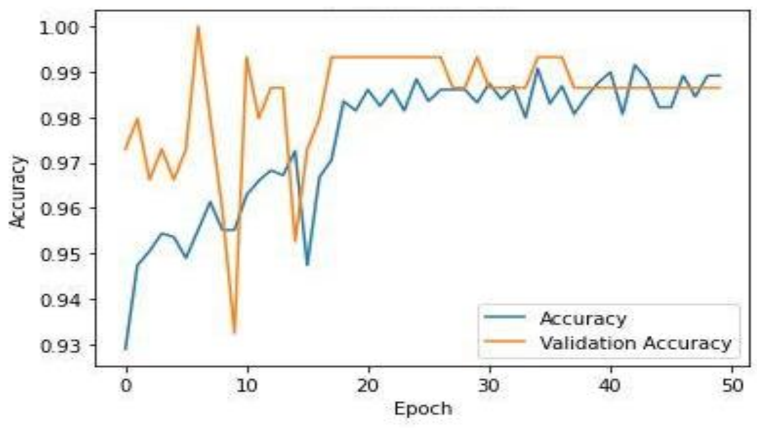


Fig.5(e) Training Accuracy Vs Validation Accuracy

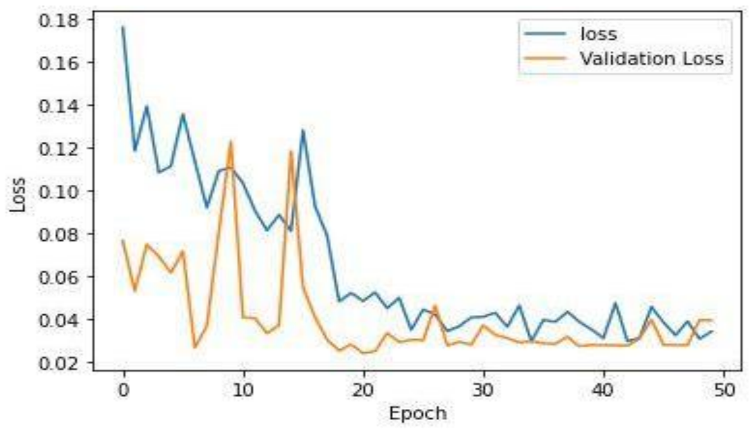


Fig.5(f) Training Loss vs Validation Loss

We proposed a deep structured algorithm to automatically extract the self-learned featured the use of an cease-to-stop learning CNN in diagnosing lung most cancers CT images. A properly-tuned CNN algorithm with optimizers exhibited promising effects when in comparison to actual ROI samples in phrases of classification accuracy (93.9%), differentiating the benign, malignant, and non-cancerous lung nodules with low false positives. The contributions of every feature extraction technique which are used in this task are evaluated. To evaluate the performance of these feature extraction processes, the overall performance metrics, specifically, accuracy measures are used. Ideally, a good feature extraction method is predicted to have a high accuracy.

Table.1 Comparison of model metrics with Adam and SGD

Optimizers	Accuracy	Sensitivity	Specificity
Adam	98.17%	97.91%	98.27%
SGD	94.5%	89.58%	96.5%

5.1 Analysis of proposed work with confusion matrix

To confirm the performance of the proposed classification approach, the confusion matrix is used as the the metric. Here there are two classes: benign nodules and malignant.

- It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.
- It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.
- With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

A confusion matrix, also known as an error matrix, is a summarized table used to assess the performance of a classification model. The number of correct and incorrect predictions are summarized with count values and broken down by each class. Below is an image of the structure of a 2x2 confusion matrix. To give an example, let’s say that there were ten instances where a classification model predicted ‘Yes’ in which the actual value was ‘Yes’. Then the number ten would go in the top left corner in the True Positive quadrant. This leads us to some key terms:

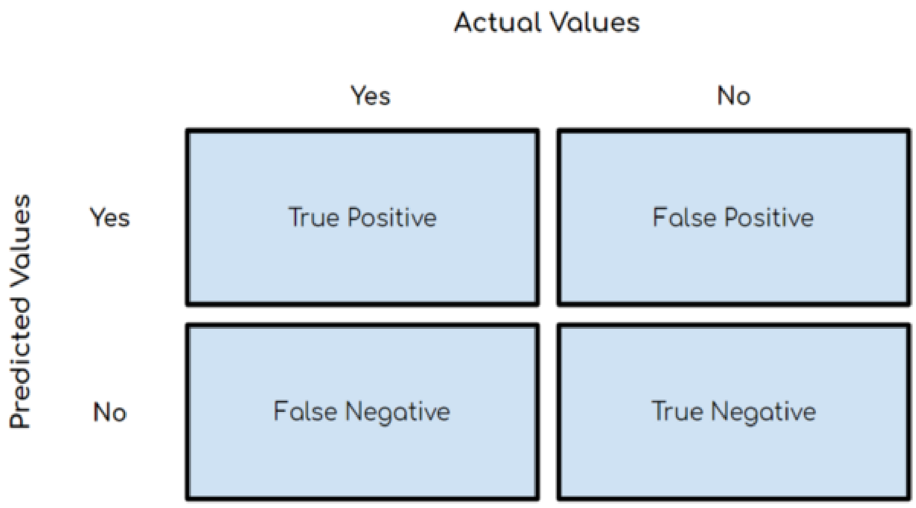


Fig.5.1 Structure for a 2x2 Confusion Matrix

- Positive (P): Observation is positive (e.g.is a dog).
- Negative (N): Observation is not positive (e.g.is not a dog).
- True Positive (TP): Outcome where the model correctly predicts the positive class.
- True Negative (TN): Outcome where the model correctly predicts the negative class.
- False Positive (FP): Also called a type 1 error, an outcome where the model incorrectly predicts the positive class when it is actually negative.
- False Negative (FN): Also called a type 2 error, an outcome where the model incorrectly predicts the negative class when it is actually positive.

5.2 Confusion Matrix Metrics

we can dive into some of the main metrics that we can calculate from a confusion matrix.

Accuracy

The most commonly used metric to judge a model and is actually not a clear indicator of the performance. The worse happens when classes are imbalanced. This is simply equal to the proportion of predictions that the model classified correctly. Take for example a cancer detection model. The chances of actually having cancer are very low. Let's say out of 100, 90 of the patients don't have cancer and the remaining 10 actually have it. We don't want to miss on a patient who is having cancer but goes undetected (false negative). Detecting everyone as not having cancer gives an accuracy of 90% straight. The model did nothing here but just gave cancer free for all the 100 predictions.

$$Accuracy = \frac{\#\, of\ correct\ predictions}{total\ \#\, of\ predictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

Precision is also known as **positive predictive value** and is the proportion of relevant instances among the retrieved instances. Here denominator is the model prediction done as positive from the whole given dataset. Take it as to find out '*how much the model is right* when it says it is right'.In other words, it answers the question "What proportion of positive identifications was actually correct?"

$$Precision = \frac{TP}{TP + FP}$$

Recall / Sensitivity

Recall, also known as the sensitivity, hit rate, or the true positive rate **(**TPR**)**, is the proportion of the total amount of relevant instances that were actually retrieved. Percentage of positive instances out of the total actual positive instances. Therefore denominator (TP + FN*)* here is the actual number of positive instances present in the dataset.

$$Recall = \frac{TP}{TP + FN}$$

Specificity

Specificity, also known as the true negative rate (TNR)**,** measures the proportion of actual negatives that are correctly identified as such. It is the opposite of recall. Percentage of negative instances out of the *total actual negative* instances. Therefore denominator (*TN + FP)* here is the *actual* number of negative instances present in the dataset. It is similar to recall but the shift is on the negative instances. *Like finding out how many healthy patients were not having cancer and were told they don't have cancer*. Kind of a measure to see how separate the classes are.

$$Specificity = \frac{TN}{TN + FP}$$

F1 score

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Support

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

ROC (Receiver Operating Characteristics) curve

ROC stands for receiver operating characteristic and the graph is plotted against TPR and FPR for various threshold values. As TPR increases FPR also increases. As you can see in the first figure, we have four categories and we want the threshold value that leads us closer to the top left corner. Comparing different predictors (here 3) on a given dataset also becomes easy as you can see in figure 2, one can choose the threshold according to the application at hand. ROC AUC is just the area under the curve, the higher its numerical value the better.

$$\textit{True Positive Rate (TPR) = RECALL} = \frac{TP}{TP+FN}$$

$$\textit{False Positive Rate (FPR) = 1} - Specificity = \frac{FP}{TN+FP}$$

A useful tool when predicting the probability of a binary outcome is the Receiver Operating Characteristic curve, or ROC curve. It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0

and 1.0. Put another way, it plots the false alarm rate versus the hit rate. The true positive rate is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive.

1 True Positive Rate = True Positives / (True Positives + False Negatives)

The true positive rate is also referred to as sensitivity.

1 Sensitivity = True Positives / (True Positives + False Negatives)

The false positive rate is calculated as the number of false positives divided by the sum of the number of false positives and the number of true negatives. It is also called the false alarm rate as it summarizes how often a positive class is predicted when the actual outcome is negative.

1 False Positive Rate = False Positives / (False Positives + True Negatives)

The false positive rate is also referred to as the inverted specificity where specificity is the total number of true negatives divided by the sum of the number of true negatives and false positives.

1 Specificity = True Negatives / (True Negatives + False Positives)

1 False Positive Rate = 1-Specificity

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

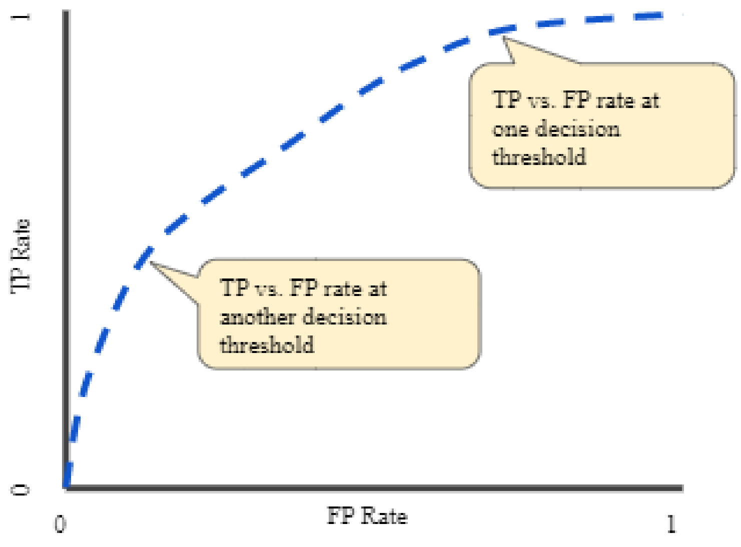


Fig.5.2(a) TP vs FP rate at different classification thresholds.

The ROC curve is a useful tool for a few reasons:

The curves of different models can be compared directly in general or for different thresholds.

The area under the curve (AUC) can be used as a summary of the model skill.

The shape of the curve contains a lot of information, including what we might care about most for a problem, the expected false positive rate, and the false negative rate.

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

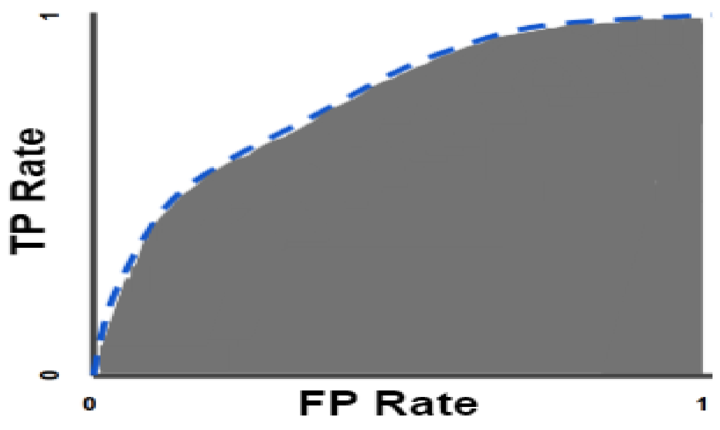


Fig.5.2(b) AUC (Area under the ROC Curve).

AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red) example.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

AUC is desirable for the following two reasons:

AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values.

AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:

Scale invariance is not always desirable. For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
Classification-threshold invariance is not always desirable. In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.
Smaller values on the x-axis of the plot indicate lower false positives and higher true negatives.
Larger values on the y-axis of the plot indicate higher true positives and lower false negatives. If you are confused, remember, when we predict a binary outcome, it is either a correct prediction (true positive) or not (false positive). There is a tension between these options, the same with true negative and false negative.
A skillful model will assign a higher probability to a randomly chosen real positive occurrence than a negative occurrence on average. This is what we mean when we say that the model has skill. Generally, skillful models are represented by curves that bow up to the top left of the plot.

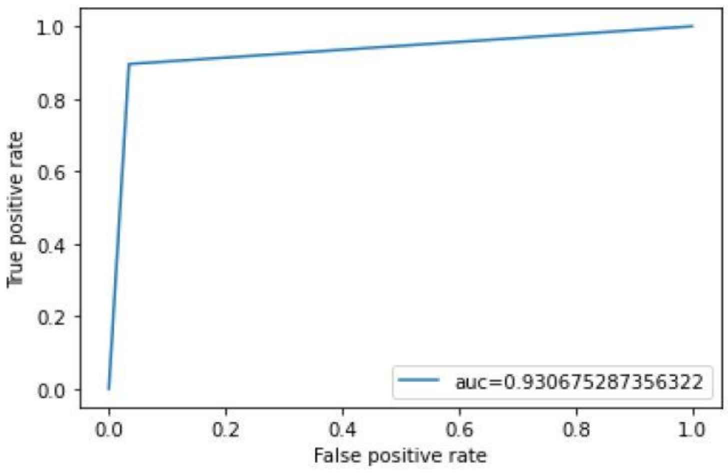


Fig.5.2(c) ROC curve of cancer prediction using SGD optimizer

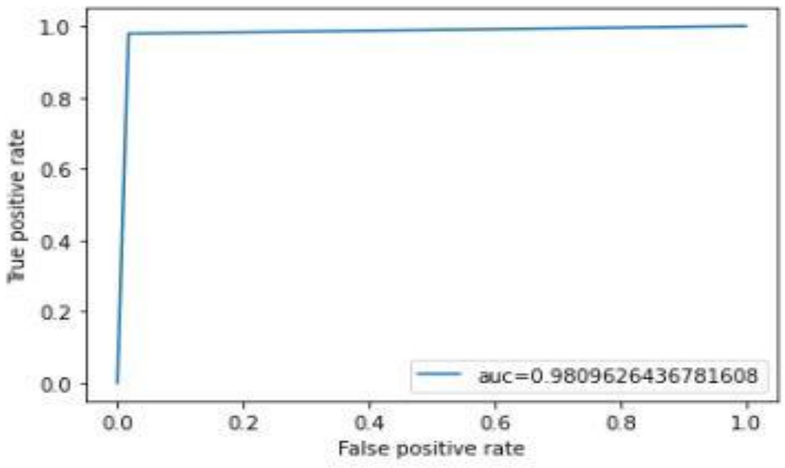


Fig.5.2(d) ROC curve of cancer prediction using Adam optimizer

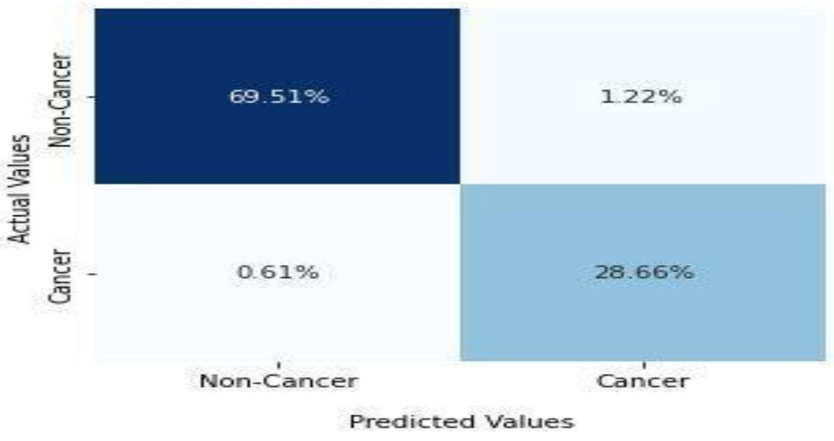


Fig.5.2(e) Confusion matrix for cancer detection using Adam optimizer

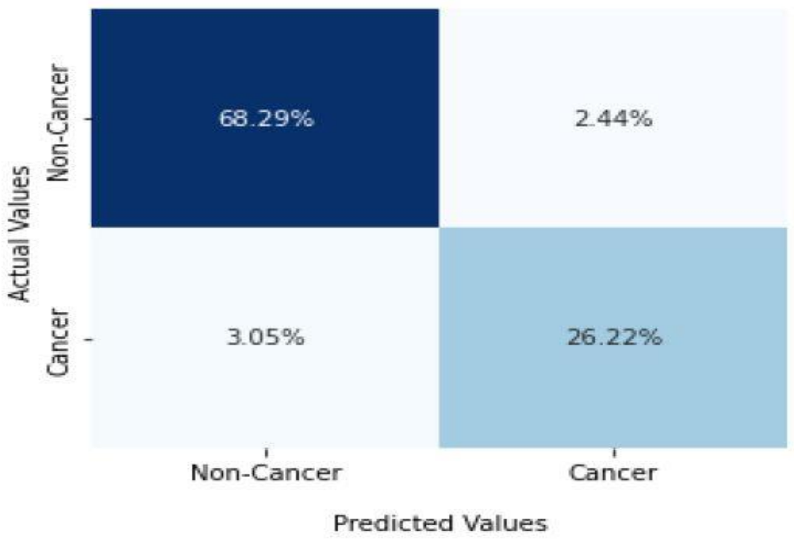
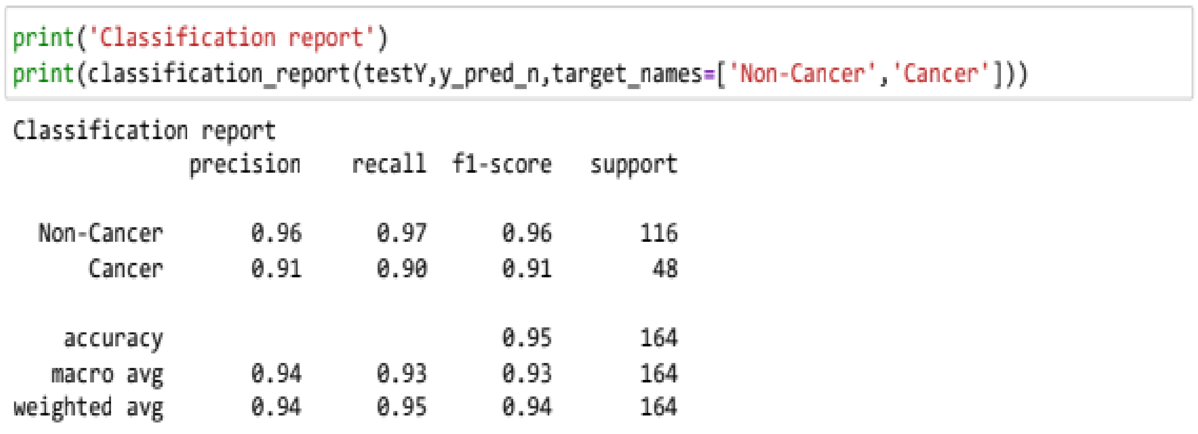


Fig.5.2(f) Confusion matrix for cancer detection using SGD optimizer

The confusion matrix is used to obtain the result of the classification or misclassification report in the form of a matrix. By using the concept of binary classification, four combinations of data category can be formed which are True-Positive (TP), True-Negative (TN), False-Positive (FP), and False-Negative (FN) as represented in Table 1 which are later used for calculation of different types of performance evaluation metrics. Where the True-Positive (TP) are more concerned samples and False-Negative (FP) are merely rejected/discarded samples.



```
print('Classification report')
print(classification_report(testY,y_pred_n,target_names=['Non-Cancer','Cancer']))
```

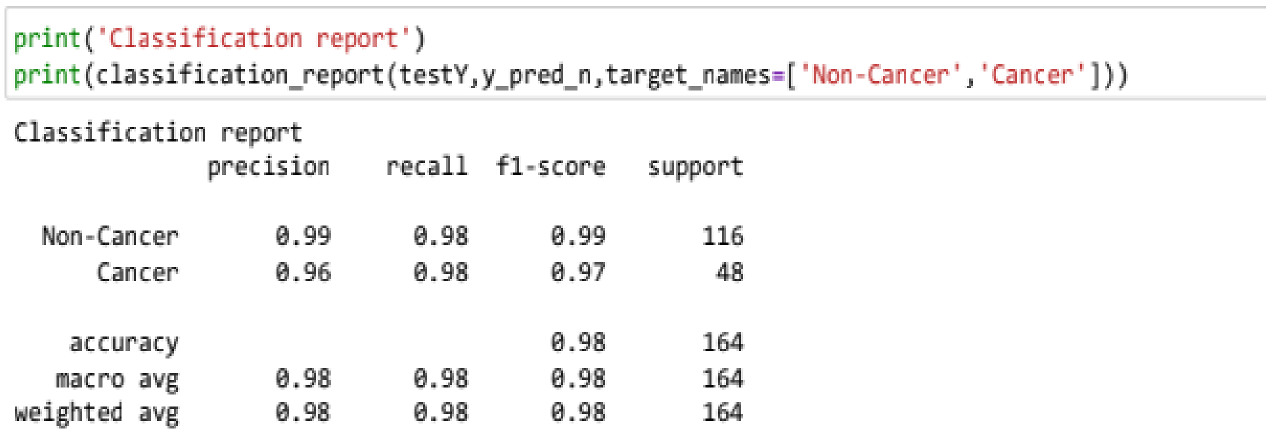
```
Classification report
              precision    recall  f1-score   support

  Non-Cancer       0.96      0.97      0.96       116
      Cancer       0.91      0.90      0.91        48

    accuracy                           0.95       164
   macro avg       0.94      0.93      0.93       164
weighted avg       0.94      0.95      0.94       164
```

Fig.5.2(g) Classification report using SGD optimizer

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.



```
print('Classification report')
print(classification_report(testY,y_pred_n,target_names=['Non-Cancer','Cancer']))
```

```
Classification report
              precision    recall  f1-score   support

  Non-Cancer       0.99      0.98      0.99       116
      Cancer       0.96      0.98      0.97        48

    accuracy                           0.98       164
   macro avg       0.98      0.98      0.98       164
weighted avg       0.98      0.98      0.98       164
```

Fig.5.2(h) Classification report using Adam optimizer

The report shows the main classification metrics precision, recall and f1-score on a per-class basis. The metrics are calculated by using true and false positives, true and false negatives. Positive and negative in this case are generic names for the predicted classes.

There are four ways to check if the predictions are right or wrong:

1. TN / True Negative: when a case was negative and predicted negative
2. TP / True Positive: when a case was positive and predicted positive
3. FN / False Negative: when a case was positive but predicted negative
4. FP / False Positive: when a case was negative but predicted positive

Precision

Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.

TP – True Positives

FP – False Positives

Precision – Accuracy of positive predictions.

Precision = TP/(TP + FP)

```
from sklearn.metrics import precision_score

print("Precision score: {}".format(precision_score(y_true,y_pred)))
```

Recall

Fraction of positives that were correctly identified.

Recall = TP/(TP+FN)

```
from sklearn.metrics import recall_score

print("Recall score: {}".format(recall_score(y_true,y_pred)))
```

F1 score

The F_1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F_1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F_1 should be used to compare classifier models, not global accuracy.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

```
from sklearn.metrics import f1_score

print("F1 Score: {}".format(f1_score(y_true,y_pred)))
```

CHAPTER 6
CONCLUSION & FUTURE SCOPE

6.1 Conclusion

In this project, a convolutional neural network-based system carried out to discover the malignancy tissues within the input lung CT photograph. Lung photograph with different form, size of the cancerous tissues has been fed on the input for training the machine. The proposed system is able to discover the presence and absence of cancerous cells with accuracy of approximately 98.17%. The accuracy of Lung cancer detection with the proposed convolutional neural network-based totally technique was compared with that acquired by using preceding works. inside the near future, the gadget may be trained with huge datasets to diagnose the kind of cancer with its size and shape.

6.2 Future Scope

Deep learning promise a radical design for lung cancer screening in the future, due to their ability to manage a vast amount of data and automatically characterize pulmonary nodules with precision. Various combination models of convolutional neural networks, machine learning, handcrafted features, computer-aided diagnosis, spectrometry, genetic and molecular changes, provided better discrimination and evaluation of a greater proportion of lung nodules with higher sensitivity, specificity, and accuracy. Besides, the deep learning model combined with spectrometry developed protein marker panels for lung cancer detection and various other models enabling non-invasive breath tests or using public health information achieves greater detection accuracy. Unlike typical algorithms, deep convolutional neural networks increased the precision of classification and characterization of lung nodules with a higher detection rate. A combination of features is always superior and combining artificial intelligence with the performance of radiologists can develop a cost-effective and time saving efficient tool for lung cancer screening. However, validation of the models is essential in the future, for their implementation in the routine healthcare system so that it can be beneficial.

CHAPTER 7
REFERENCES

[1] Supriya Suresh, Subaji - ROI-based feature learning for efficient true positive prediction using convolutional neural network for lung cancer diagnosis - Received: 12 January 2019 / Accepted: 17 February 2020 Springer-Verlag London Ltd., part of Springer Nature 2020

[2] Juanjuan Zhao, Guohua Ji, Yan Qiang, Xiaohong Han, Bo Pei, Zhenghao Shi- A New Method of Detecting Pulmonary Nodules with PET/CT Based on an Improved Watershed Algorithm College of Computer Science and Engineering, Xi'an University of Technology, 2015 Xi'an 710048, China

[3] A. Asuntha & Andy Srinivasan - Deep learning for lung Cancer detection and classification - 24 December 2018 / Revised: 8 October 2019 / Accepted: 13 October 2019 Springer Science+Business Media, LLC, part of Springer Nature 2020.

[4] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) - Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680

[5] Han H, Li L, Han F, Song B, Moore W, Liang Z - Fast and adaptive detection of pulmonary nodules in thoracic CT images using a hierarchical vector quantization scheme. IEEE J Biomed Health (2015) Inf 19(2):648–659

[6] Hansell DM, Bankier AA, MacMahon H, McLoud TC, Muller NL, Remy J Fleischner society-glossary of terms for thoracic imaging. -(2008) Radiology 246(3):697–722

[7] Hochhegger B, Zanon M, Altmayer S, Pacini GS, Balbinot F, Francisco MZ, Dalla Costa R, Watte G, Santos MK, Barros MC et al Advances in imaging and automated quantification of malignant pulmonary diseases:- (2018) a state-of-the-art review. Lung 196(6):633–642

[8] Hu J, Shen L, Sun G Squeeze-and-excitation networks - In: Proceedings of the IEEE conference on computer vision and pattern recognition, (2018) pp 7132–7141

[9] Aggarwal T, Furqan A, Kalra K Feature extraction and LDA based classification of lung nodules in chest CT scan images. IEEE, International Conference on Advances in Computing, Communications and Informatics (ICACCI), (2015) pp. 1189–1193

[10] Akram S, Javed MY, Hussain A, Riaz F, Usman Akram M Intensity-based statistical features for classification of lungs CT scan nodules using artificial intelligence techniques. Journal of Experimental & Theoretical Artificial Intelligence (2015) 27(6):737–751. https://doi.org/10.1080/0952813X.2015.1020526

[11] Alakwaa W, Nassef M, Badr A Lung Cancer detection and classification with 3D convolutional neural network (3D-CNN). International Journal of Advanced Computer Science and

Applications (2017) (IJACSA) 8(8):409–417

[12] De Sousa Costa RW, da Silva GLF, de Carvalho Filho AO, Silva AC, de Paiva Marcelo Gattass AC “Classification of malignant and benign lung nodules using taxonomic diversity index and phylogenetic distance”, springer. (2018) Med Biol Eng Comput 56(11):2125–2136

[13] Dhaware BU, Pise AC, Lung Cancer Detection Using Bayasein Classifier and FCM Segmentation. IEEE, International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT), (2016) pp. 170–174

[14] Ignatious S, Joseph R Computer Aided Lung Cancer Detection System. IEEE, Proceedings of 2015 Global Conference on Communication Technologies (GCCT 2015), (2015) pp. 555–558.

[15] Jin X-Y, Zhang Y-C, Jin Q-L Pulmonary nodule detection based on CT images using Convolution neural network. IEEE, 9th International Symposium on Computational Intelligence and Design, (2016) pp. 202– 204.

CHAPTER 8
APPENDICES

Data Preparation

```
import numpy as np
import pandas as pd
import pydicom as dicom
import os
import scipy.ndimage
import matplotlib.pyplot as plt
from skimage import measure, morphology, segmentation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import scipy.ndimage as ndimage

INPUT_FOLDER = './stage1'
patients = os.listdir(INPUT_FOLDER)
patients.sort()

def load_scan(path):
ds = []
for s in os.listdir(path):
if( s != '.DS_Store'):
ds.append(s)
slices = [dicom.read_file((path + '/' + s),force = True) for s in ds]
slices.sort(key = lambda x: int(x.InstanceNumber))
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 get_pixels_hu(scans):
image = np.stack([s.pixel_array for s in scans])
# Convert to int16 (from sometimes int16),
# should be possible as values should always be low enough (<32k)
image = image.astype(np.int16)
# Set outside-of-scan pixels to 0
# The intercept is usually -1024, so air is approximately 0
image[image == -2000] = 0
# Convert to Hounsfield units (HU)
intercept = scans[0].RescaleIntercept
slope = scans[0].RescaleSlope
if slope != 1:
image = slope * image.astype(np.float64)
image = image.astype(np.int16)
image += np.int16(intercept)
return np.array(image, dtype=np.int16)

test_patient_scans = load_scan(INPUT_FOLDER +'/'+ patients[1])
test_patient_images = get_pixels_hu(test_patient_scans)
test_patient_images.shape[0]

def generate_markers(image):
#Creation of the internal Marker
marker_internal = image < -400
marker_internal = segmentation.clear_border(marker_internal)
marker_internal_labels = measure.label(marker_internal)
areas = [r.area for r in measure.regionprops(marker_internal_labels)]
areas.sort()
if len(areas) > 2:
for region in measure.regionprops(marker_internal_labels):
if region.area < areas[-2]:
for coordinates in region.coords:
```

```
marker_internal_labels[coordinates[0], coordinates[1]] = 0
marker_internal = marker_internal_labels > 0
#Creation of the external Marker
external_a = ndimage.binary_dilation(marker_internal, iterations=10)
external_b = ndimage.binary_dilation(marker_internal, iterations=55)
marker_external = external_b ^ external_a
#Creation of the Watershed Marker matrix
marker_watershed = np.zeros((512, 512), dtype=np.int)
marker_watershed += marker_internal * 255
marker_watershed += marker_external * 128

return marker_internal, marker_external, marker_watershed

#Show some example markers from the middle
test_patient_internal, test_patient_external, test_patient_watershed =
generate_markers(test_patient_images[80])
print ("Internal Marker")
plt.imshow(test_patient_internal, cmap='gray')
plt.show()
print ("External Marker")
plt.imshow(test_patient_external, cmap='gray')
plt.show()
print ("Watershed Marker")
plt.imshow(test_patient_watershed, cmap='gray')
plt.show()

def seperate_lungs(image):
#Creation of the markers as shown above:
marker_internal, marker_external, marker_watershed = generate_markers(image)

#Creation of the Sobel-Gradient
sobel_filtered_dx = ndimage.sobel(image, 1)
sobel_filtered_dy = ndimage.sobel(image, 0)
sobel_gradient = np.hypot(sobel_filtered_dx, sobel_filtered_dy)
sobel_gradient *= 255.0 / np.max(sobel_gradient)


#Watershed algorithm
watershed = segmentation.watershed(sobel_gradient, marker_watershed)

#Reducing the image created by the Watershed algorithm to its outline
outline = ndimage.morphological_gradient(watershed, size=(3,3))
outline = outline.astype(bool)

#Performing Black-Tophat Morphology for reinclusion
#Creation of the disk-kernel and increasing its size a bit
blackhat_struct = [[0, 0, 1, 1, 1, 0, 0],
[0, 1, 1, 1, 1, 1, 0],
[1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1],
[0, 1, 1, 1, 1, 1, 0],
[0, 0, 1, 1, 1, 0, 0]]

blackhat_struct = ndimage.iterate_structure(blackhat_struct, 8)
#Perform the Black-Hat
outline += ndimage.black_tophat(outline, structure=blackhat_struct)

#Use the internal marker and the Outline that was just created to generate the lungfilter
lungfilter = np.bitwise_or(marker_internal, outline)
#Close holes in the lungfilter
#fill_holes is not used here, since in some slices the heart would be reincluded by
accident
lungfilter = ndimage.morphology.binary_closing(lungfilter, structure=np.ones((7,7)),
iterations=3)

#Apply the lungfilter (note the filtered areas being assigned -2000 HU)
segmented = np.where(lungfilter == 1, image, -2000*np.ones((512, 512)))
```

```
#### nodule
lung_nodule_1 = np.bitwise_or(marker_internal, image)
lung_nodule = np.where(lungfilter == 1, lung_nodule_1, np.zeros((512, 512)))

return segmented, lung_nodule, lungfilter, outline, watershed, sobel_gradient,
marker_internal, marker_external, marker_watershed



test_segmented, lung_nodule, test_lungfilter, test_outline, test_watershed,
test_sobel_gradient, test_marker_internal, test_marker_external, test_marker_watershed
= seperate_lungs(test_patient_images[80])

print ("Lung Nodule")
plt.imshow(lung_nodule, cmap='gray')
plt.show()
print ("Sobel Gradient")
plt.imshow(test_sobel_gradient, cmap='gray')
plt.show()
print ("Watershed Image")
plt.imshow(test_watershed, cmap='gray')
plt.show()
print ("Outline after reinclusion")
plt.imshow(test_outline, cmap='gray')
plt.show()
print ("Lungfilter after closing")
plt.imshow(test_lungfilter, cmap='gray')
plt.show()
print ("Segmented Lung")
plt.imshow(test_segmented, cmap='gray')
plt.show()



labels_df = pd.read_csv('./stage1_labels.csv',index_col=0)
```

```
labels_d = pd.read_csv('./stage1_labels.csv')
lst=labels_d["id"].tolist()
lst


data = []
labels = []
print('*'*30)
print("data in converting.......")
print('*'*30)
j = 0
for patient in patients[:18]:
if patient in lst:
test_patient_scans = load_scan(INPUT_FOLDER +'/'+ patients[j])
test_patient_images = get_pixels_hu(test_patient_scans)
path = INPUT_FOLDER +'/'+ patient
ds = []
for s in os.listdir(path):
if( s != '.DS_Store'):
ds.append(s)
slices = [dicom.read_file((path + '/' + s),force=True) for s in ds]
print("patient_number_{} : {}".format(j,patients[j]))
i = 0
for s in slices:
try:
#print(str(s.PatientID))
i += 1
if i in range(0,50):
continue
#taking 20 slices from each patient
elif i in range(50, 150):
img = test_patient_images[i]
seg_img = seperate_lungs(img)[0]
new_img = np.expand_dims(seg_img,axis = -1)
```

```
label = labels_df.at[(str(s.PatientID), 'cancer')]
data.append(new_img)
labels.append(label)

print("converted image is : "+str(len(labels)))
else:
break
except IndexError:
continue
j += 1
print("Done")

data_new = np.array(data)
data_new.shape

labels_new = np.array(labels)
labels_new.shape

from skimage.transform import  resize
import numpy as np
data_n = np.zeros((len(data_new),256,256,1))
for i in range(len(data_new)):
image = data_new[i]
image_resize =  resize(image, (256, 256), anti_aliasing=True)
data_n[i] = image_resize
data_n.shape

from numpy import save
print("saving data")
save("./data_2_256_1638.npy",data_n)
print("saving labels")
save("./labels_2_1638.npy",labels_new)

plt.imshow(data[301],cmap='gray')
```

Train and Test

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.ndimage as ndimage

from numpy import load
print("loading data")
data = load("./data_2_256_1638.npy")
print("loading labels")
labels = load("./labels_2_1638.npy")

from sklearn.model_selection import train_test_split
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.1, stratify=labels,
random_state=42,shuffle = True)

(trainX, valX, trainY, valY) = train_test_split(trainX, trainY, test_size=0.1,
stratify=trainY, random_state=42,shuffle = True)

plt.imshow(trainX[50],cmap = 'gray')
plt.show

from tensorflow.keras.preprocessing.image import ImageDataGenerator
aug_train = ImageDataGenerator(rescale= 1.0/255.,
rotation_range=20,
zoom_range=0.15,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.15,
horizontal_flip=True,
fill_mode="nearest")
```

```
aug_test=ImageDataGenerator(rescale=1.0/255)


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout

classifier_2 = Sequential()

# Step 1 - Convolution
classifier_2.add(Conv2D(32, (3, 3), input_shape = (256, 256, 1), activation = 'relu'))

# Step 2 - Pooling
classifier_2.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier_2.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier_2.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a third convolutional layer
classifier_2.add(Conv2D(64, (3, 3), activation = 'relu'))
classifier_2.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a forth convolutional layer
classifier_2.add(Conv2D(64, (3, 3), activation = 'relu'))
classifier_2.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a forth convolutional layer
classifier_2.add(Conv2D(128, (3, 3), activation = 'relu'))
classifier_2.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a forth convolutional layer
classifier_2.add(Conv2D(128, (3, 3), activation = 'relu'))
classifier_2.add(MaxPooling2D(pool_size = (2, 2)))
```

```
# Step 3 - Flattening
classifier_2.add(Flatten())

# Step 4 - Full connection
classifier_2.add(Dense(units = 512, activation = 'relu'))
classifier_2.add(Dense(units = 1, activation = 'sigmoid'))



from tensorflow.keras.optimizers import SGD,Adam
#classifier_2.compile(optimizer = Adam(lr=0.001), loss = 'binary_crossentropy', metrics
= ['acc'])
classifier_2.compile(optimizer = SGD(lr=0.001), loss = 'binary_crossentropy', metrics =
['acc'])
hist_0 = classifier_2.fit(aug_train.flow(trainX, trainY, batch_size=32),
steps_per_epoch=steps, epochs = 50, verbose = 1,
validation_data = aug_test.flow(valX, valY, batch_size=32), callbacks=callbacks_1)

# print accuracy graph
import matplotlib.pyplot as plt
plt.plot(hist_0.history["acc"])
plt.plot(hist_0.history['val_acc'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy"])

# print loss graph
plt.plot(hist_0.history['loss'])
plt.plot(hist_0.history['val_loss'])
plt.title("model loss")
plt.ylabel("Loss")
plt.xlabel("Epoch")
plt.legend(["loss","Validation Loss"])
```

```
#testing the model
import tensorflow
model_2 = tensorflow.keras.models.load_model("test_model_22.h5")
#model_2.summary()
from sklearn.metrics import confusion_matrix,classification_report
y_pred = model_2.predict(testX)
y_pred_nn=[int(p>0.5) for p in y_pred]
y_pred_n = np.array(y_pred_nn)
print('confusion matrix')
print(confusion_matrix(testY,y_pred_n))
```