**LUNG SEGMENTATION USING MACHINE LEARNING**

**Submitted by**

MAAN VEER MAURYA (RA2011047010085) NISHTHA BAHIRAT (RA2011047010108) SIDDHARTH GHOSH (RA2011047010089) AVIRAL SIROTIYA (RA2011047010151) AAYUSHI AGRAWAL (RA2011047010124)

Under the guidance of

**Dr. Rakesh Kumar M**

(Assistant Professor, Department of Computational Intelligence)

in partial fulfillment for the award of the degree

Of

**BACHELOR OF TECHNOLOGY**

in

**COMPUTATIONAL INTELLIGENCE**

of

**FACULTY OF ENGINEERING AND TECHNOLOGY**

****

**S.R.M. Nagar, Kattankulathur, Chengalpattu District**

**JUNE 2022**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

##### BONAFIDE CERTIFICATE

Certified that this project report titled **“LUNG SEGMENTATION”** is the Bonafede work of **“**MAAN VEER MAURYA (RA2011047010085), NISHTHA BAHIRAT (RA2011047010108), SIDDHARTH GHOSH (RA2011047010089), AVIRAL SIROTIYA (RA2011047010151), AAYUSHI AGRAWAL (RA2011047010124)**”**, who

carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**SIGNATURE SIGNATURE**

Dr. RAKESH KUMAR M Dr. ANNIE UTHRA

**GUIDE HEAD OF THE DEPARTMENT**

Assistant Professor

Dept. of Computational Intelligence Dept. of Computational Intelligence

Signature of the Internal Examiner Signature of the External Examiner

### ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide, **Dr. Rakesh Kumar M**, his valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing the project. All through the work, in spite of his busy schedule, he has extended cheerful and cordial support to me for completing this project work.

MAAN VEER MAURYA (RA2011047010085) NISHTHA BAHIRAT (RA2011047010108) SIDDHARTH GHOSH (RA2011047010089) AVIRAL SIROTIYA (RA2011047010151) AAYUSHI AGRAWAL (RA2011047010124)

### ABSTRACT

Lung CT image segmentation is a necessary initial step for lung image analysis, it is a prerequisite step to provide an accurate lung CT image analysis such as lung cancer detection. In this work, we propose a lung CT image segmentation using the U-net architecture, one of the most used architectures in deep learning for image segmentation. The architecture consists of a contracting path to extract high-level information and a symmetric expanding path that recovers the information needed. This network can be trained end-to- end from very few images and outperforms many methods.

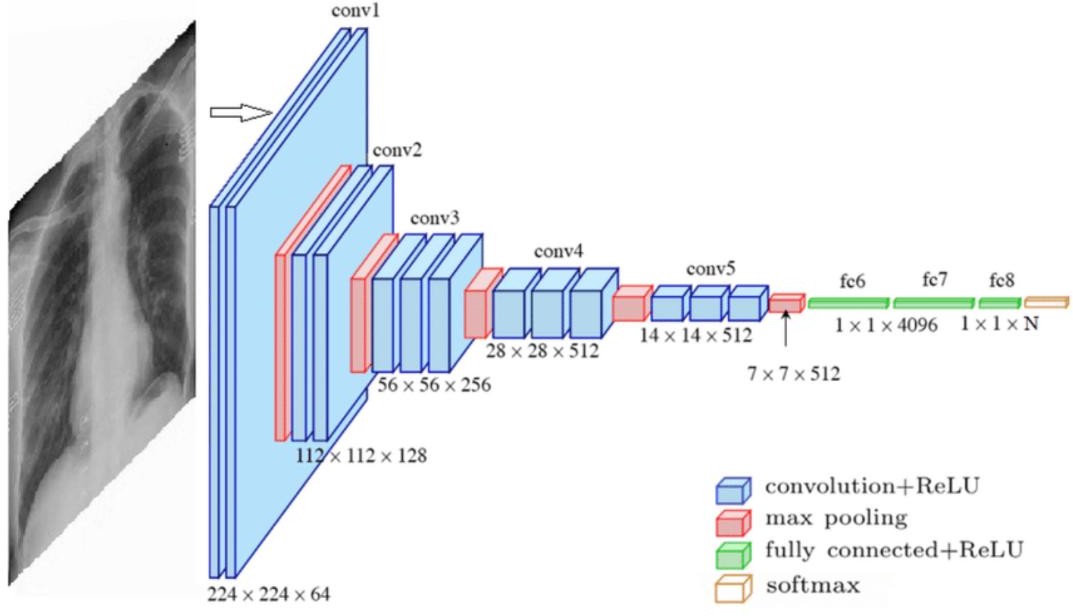
**CHAPTER 1**

### INTRODUCTION

Lung cancer is a lethal lung disease that causes more than one million of deaths yearly. It is one of the most common medical conditions in the world. By definition, lung cancer is a malignant lung tumor that is characterized by uncontrollable growth in the lung tissue. Early detection of lung cancer could reduce the mortality rate and increase the patient’s survival rate when the treatment is more likely curative. Computed tomography (CT) imaging is an efficient medical screening test used for lung cancer diagnosis and detection. The physician uses the obtained CT images to analyze and diagnose the lung tissues.

However, in many frequent cases, it is difficult for the physician to obtain an accurate diagnosis without the help of additional tool known as Computed Aided Diagnosis (CAD) System.

Computer Aided Diagnosis (CAD) system is an efficient medical diagnosis tool and a prerequisite for today’s medical imaging practicality. The physician uses the CAD system to provide an additional second opinion in order to obtain an accurate diagnosis. It is widely useful to improve the effectiveness of the treatment. For Many CAD systems, an accurate segmentation process of the target organ is always needed. It is a prerequisite initial step for an efficient quantitative lung CT image analysis. However, designing an effective lung segmentation method is a challenging problem, especially for abnormal lung parenchyma tissue, where the nodules and blood vessels need to be segmented with the lung parenchyma. Moreover, the lung parenchyma needs to be separated from the bronchus regions that are often confused with the lung tissue.



### Convolutional [Neural Network](https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/) (CNN)

A convolutional neural network, or CNN, is a deep learning neural network sketched for processing structured arrays of data such as portrayals.

CNN are very satisfactory at picking up on design in the input image, such as lines, gradients, circles, or even eyes and faces.

This characteristic that makes convolutional neural network so robust for computer vision.

CNN can run directly on a underdone image and do not need any pre- processing.

A convolutional neural network is a feed forward neural network, seldom with up to 20.

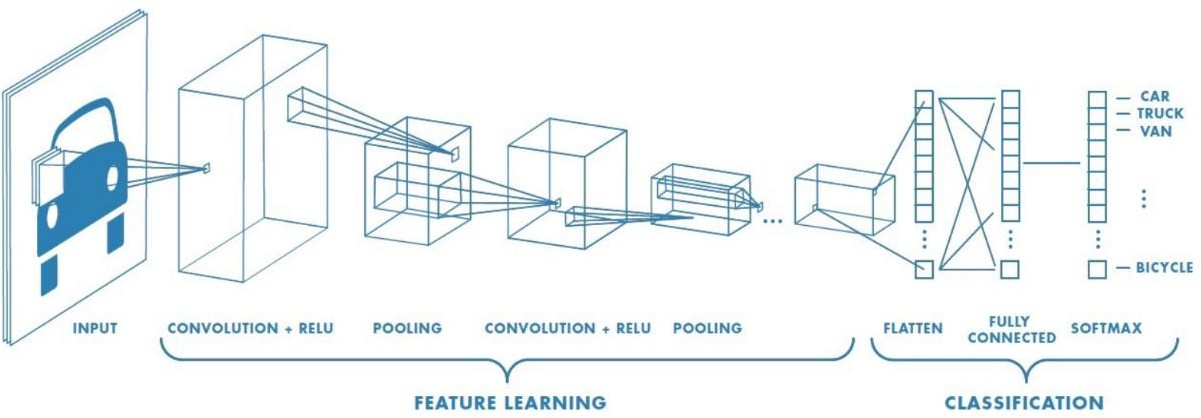
The strength of a convolutional neural network comes from a particular kind of layer called the convolutional layer.

CNN contains many convolutional layers assembled on top of each other, each one competent of recognizing more sophisticated shapes.

With three or four convolutional layers it is viable to recognize handwritten digits and with 25 layers it is possible to differentiate human faces.

The agenda for this sphere is to activate machines to view the world as humans do, perceive it in a alike fashion and even use the knowledge for a multitude of duty such as image and video recognition, image inspection and classification, media recreation, recommendation systems, natural language processing, etc.

A regular day example is given bellow:



### 1.1 OBJECTIVE:

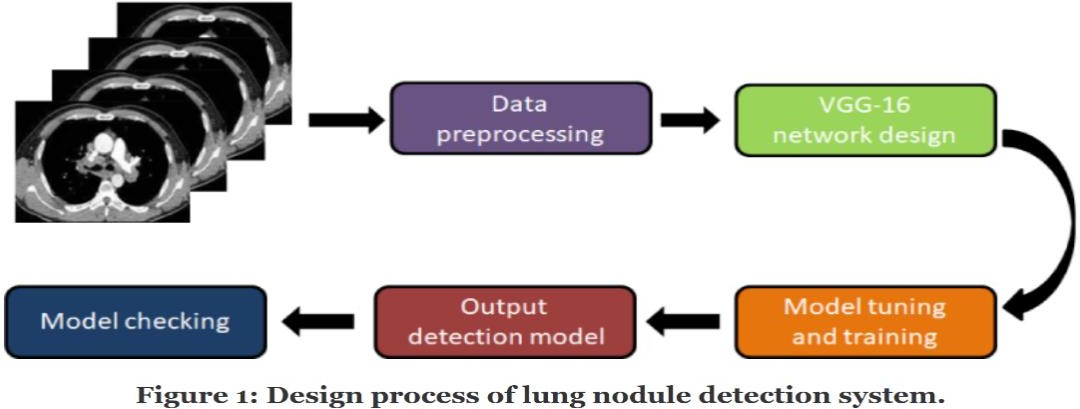
### Target detection algorithm based on CNN

Since CT images of the lungs are sequence images, most of the existing algorithms for segmenting lung parenchyma are two- dimensional segmentation processing for each frame in the CT sequence images, without considering the correlation between the images, and some researchers are engaged in sequence. In the research work of image segmentation algorithm, Gang et al. [[8](https://dl.acm.org/doi/fullHtml/10.1145/3483207.3483215#BibPLXBIB0008)]used the three-dimensional region growing method to segment the lung parenchyma, and then used the Otus threshold algorithm to extract multiple regions of interest. However, the sequence algorithm has shortcomings such as long processing time, low efficiency, and poor scalability.

Existing lung parenchymal segmentation methods need to manually select seed points, and the segmentation effect of the part that is adhered to other organs at the edge of the lung lobe is not ideal, especially the lungs of patients with lung diseases are more difficult to segment.

Aiming at the above shortcomings, this paper adopts a deep learning algorithm, adds a dilated convolution based on the VGG network, and uses the super-column feature of pixels at the same time, and finally classifies the pixels to realize the segmentation of lung parenchyma.

The research of lung nodule detection algorithm is currently the research hotspot of domestic and foreign scholars. The huge challenge encountered in the research process is to reduce the detection of false positive nodules as much as possible under the conditions of ensuring fast detection speed, simple process and high detection rate. In the context of growing maturity of deep learning technology driven by big data, the field of smart healthcare has ushered in new opportunities. This chapter is based on the improvement of the VGG- 16 network and proposes a new method for lung nodule detection.



**1.2 PROBLEM STATEMENT:**

Computer Tomography (CT) has been considered as the most sensitive imaging technique for early detection of lung cancer. On the other hand, there is a requirement for automated methodology to make use of large amount of data obtained CT images. Computer Aided Diagnosis (CAD) can be used efficiently for early detection of Lung Cancer. The usage of existing CAD system for early detection of lung cancer with the help of CT images has been unsatisfactory because of its low sensitivity and False Positive Rates (FPR). This study presents a CAD system which can automatically detect the lung cancer nodules with reduction in false positive rates. In this study, different image processing techniques are applied initially in order to obtain the lung region from the CT scan chest images. Then the segmentation is carried with the help of Fuzzy Possibility C Mean (FPCM) clustering algorithm.

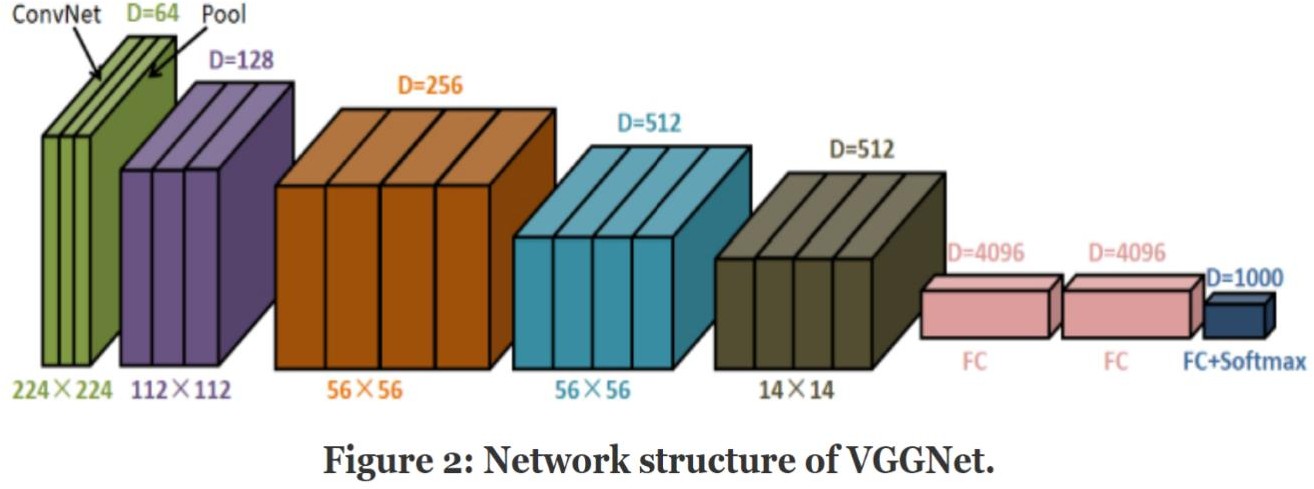
**1.3 PROPOSED SOLUTION:**

### VGG-16

**Network structure:** The VGGNet network structure was proposed in which mainly studied the relationship between depth and performance in Convolutional Neural Network (CNNs). CNNs is a class of deep

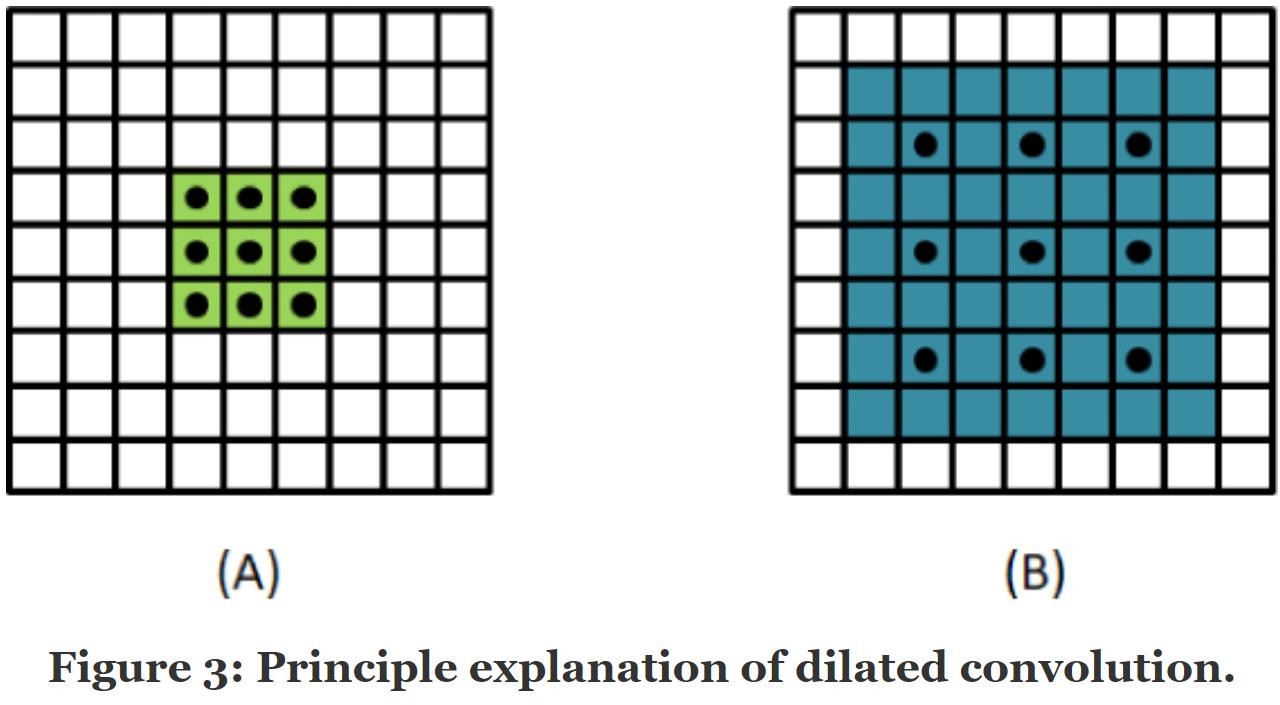
neural network, most commonly applied to detect and segment target in medical images [[3](https://dl.acm.org/doi/fullHtml/10.1145/3483207.3483215#BibPLXBIB0003), [9](https://dl.acm.org/doi/fullHtml/10.1145/3483207.3483215#BibPLXBIB0009), [13](https://dl.acm.org/doi/fullHtml/10.1145/3483207.3483215#BibPLXBIB0013), [23](https://dl.acm.org/doi/fullHtml/10.1145/3483207.3483215#BibPLXBIB0023), [24](https://dl.acm.org/doi/fullHtml/10.1145/3483207.3483215#BibPLXBIB0024)]. According to current standards, this network is not very deep, but when VGGNet was proposed, it had twice the number of layers than the commonly used network at the time, which proved that on the basis of feasible training, the deeper the network, the better the performance. And the more powerful, the better the result. In the task of image classification, the size of the input image must be fixed, because the network has a fully connected layer that requires a fixed length of input.

Before the fully connected layer, the network usually needs to convert the output feature of the convolution of the last layer into a one- dimensional vector. Through experiments, it is proved that using a small convolution kernel and increasing the network depth can also improve the effect of the network model, and VGGNet also has good generalization ability.



**Dilated convolution:** Although the pooling operation in the CNN can increase the receptive field and improve the performance of the network model, the pooling operation will also reduce the resolution. Enlarging the feature image during the up-sampling process will lose some image information. Therefore, pooling operation is not the best method in semantic segmentation network.

The concept of dilated convolution has solved this problem. Compared with the ordinary convolution method, in addition to the parameter of the convolution kernel size, the dilated convolution also has a dilated coefficient, which is mainly used to indicate the size of the dilated. This allows the dilated convolution to increase the network parameters without reducing the network.



The calculation method for the size of the cavity convolution receptive field is:

v=((ksizes+1) × (rrate−1) + ksizes) v = ((ksizes+1) × (rrate−1) + ksizes)

In the above Equation, *ksizes* represents the size of the convolution kernel, and *rrate* represents the size of the dilated coefficient.

**CHAPTER 2**

**LITERATURE REVIEW**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No** | **Project Name** | **Publishing Year** | **Journal Name** | **Author Name** |
| **1.** | **Pneumonia Detection on Chest X-Ray Using Machine Learning Paradigm** | **2019** | **Springer** | **Tej Bahadur Chandra, Kesari Verma** |
| **2.** | **ResNet-50 vs VGG-19 vs**  **training from scratch: A comparative analysis of the segmentation and classification of Pneumonia from chest X-ray images** | **2021** | **KeAi** | [**A.Victor**](https://www.sciencedirect.com/science/article/pii/S2666285X21000558) **Ikechukwu,S.Murali, R.Deepu, R.C.Shivamurthy** |
| **3.** | **An effective approach for CT lung segmentation using mask region-based convolutional neural networks** | **2020** | **ELSEVIER** | [**Qinhua Hu, Luís**](https://www.sciencedirect.com/science/article/pii/S0933365719305871?casa_token=vNS44FXXVvYAAAAA%3AicAigMZSCVojqqHXOxrkwPKiahy3wGhVZLHmlZwXqCMjHhyhnizXnwzoQksi-zUr2d2pvL7f) **Fabrício de F. Souza, Gabriel Bandeira Holanda, Shara S.A.Alves, Francisco Hércules dos S. Silva, Tao Han, Pedro P. Rebouças Filhob.** |
| **4.** | **A Deep Learning Method for Lung** | **2019** | **IEEE** | **Hieu Trung Huynh, Vo Nguyen Nhat Anh** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Segmentation on Large Size Chest X-Ray Image** |  |  |  |
| **5.** | **Enhanced lung image segmentation using deep learning** | **2022** | **Springer** | [**Shilpa Gite**](https://link.springer.com/article/10.1007/s00521-021-06719-8)**,** [**Abhinav Mishra**](https://link.springer.com/article/10.1007/s00521-021-06719-8) **&** [**Ketan Kotecha**](https://link.springer.com/article/10.1007/s00521-021-06719-8) |
| **6.** | **Survey on image segmentation techniques.** | **2017** | **Procedia Computer Science** | **Zaitoun, N. M., and M. J. Aqel.** |
| **7.** | **Colour image segmentation** | **2010** | **AMS** | **Osman, M. K.** |
| **8.** | **” Automated segmentation procedure** | **2016** | **CITSM** | **Riza, Bob Subhan** |
| **9.** | **Contour Detection and Completion for Inpainting and Segmentation Based on Topological Gradient and Fast Marching Algorithms** | **2011** | ***IEEE***  ***Transactions on Pattern Analysis and Machine Intelligence*** | **Didier Auroux** |

#### **CHAPTER 3**

#### **PROPOSED METHODOLOGY**

#### **3.1 Architecture diagram:**

#### 

#### **3.2 Description of proposed model:**

VGGNet-16 consists of 16 convolutional layers and is very appealing because of its very uniform Architecture. Similar to AlexNet, it has only 3x3 convolutions, but lots of filters. It can be trained on 4 GPUs for 2–3 weeks. It is currently the most preferred choice in the community for extracting features from images. The weight configuration of the VGGNet is publicly available and has been used in many other applications and challenges as a baseline feature extractor.

However, VGGNet consists of 138 million parameters, which can be a bit challenging to handle. VGG can be achieved through transfer Learning. In which the model is pretrained on a dataset and the parameters are updated for better accuracy and you can use the parameters values.

#### 

**Modified VGG-16 network:**

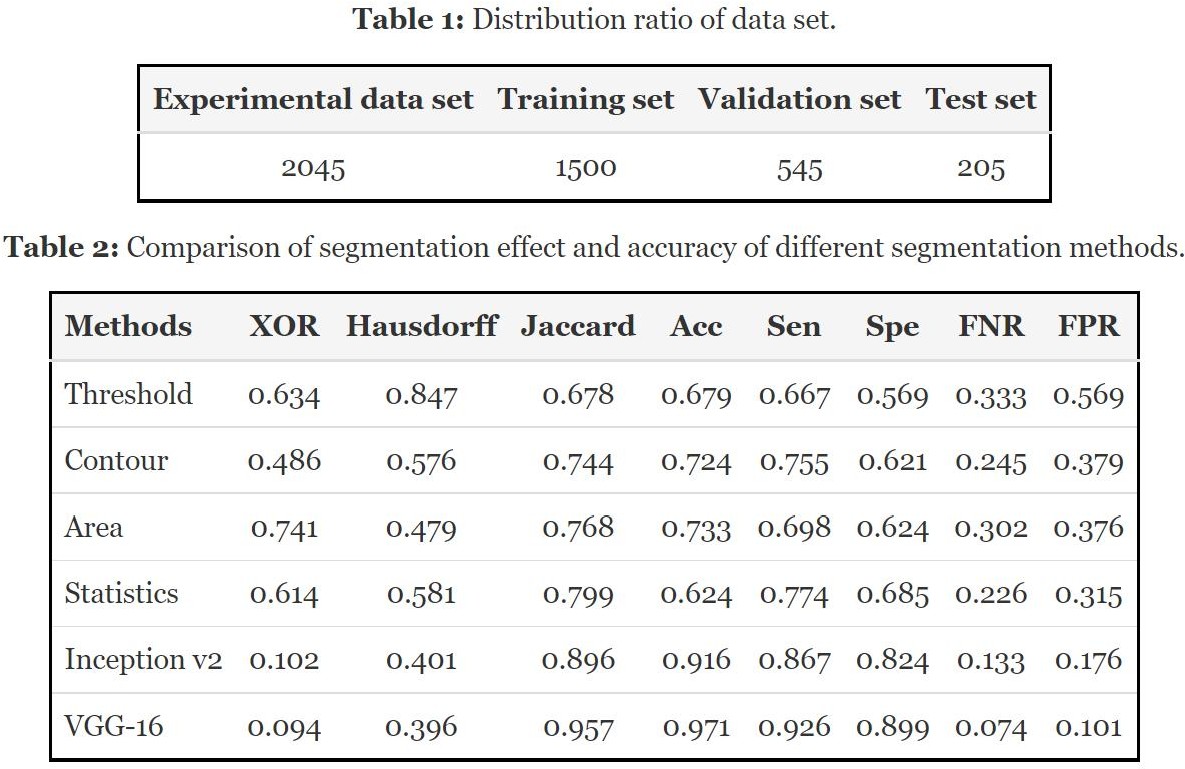
We start from the VGG-16 network, which originally designed for large-scale natural image classification. VGG-16 has 13 convolutional layers and 3 FC (fully connected) layers. The convolutional layers are denoted as conv-{11, 12, 21, 22, 31, 32, 33, 41, 42, 43, 51, 52, 53}. In this, the target dataset is comparatively small and the pre-trained VGG-16 is powerful in many segmentation tasks (The pre-trained model is obtained from the training of the ImageNet large-scale dataset). Therefore, the transfer learning is used in the training in our paper. Because we have modified the VGG-16 network, we only learn conv-{11, 12, 21, 22, 31, 32, 33, 41, 42, 43}, convolution kernel is 3 × 3, and use maxpooling. In this network, we fine-tuned the VGG-16 network. We changed the convolution of conv-{51, 52, 53} to dilated convolution, convolution kernel is 3 × 3, dilated rate is 2, and the pooling layer after cov-43 and cov-53 is canceled. We converted the last two FC layers into convolution filters, renamed cov-6 and cov-7, convolution kernel is 7 × 7, dilated rate is 4, and added them to feature sets that can be aggregated into our multi-scale hypercolumn descriptors. Following, we build predictor based on multiscale features extracted from multiple layers. Because of a strong correlation between adjacent layers, actually, there is no need to consider all the layers. We use skip-connections to extract hypercolumn features from {12, 22, 33, 43, 53, 7} with on-demand interpolation. Next, we learned about a nonlinear predictor for classifying pixels, which is implemented as a multilayer perceptron (MLP) defined on a hypercolumn features. We use MLP, which can be implemented as a series of “Fully Connected” layers, followed by the ReLU activation function.

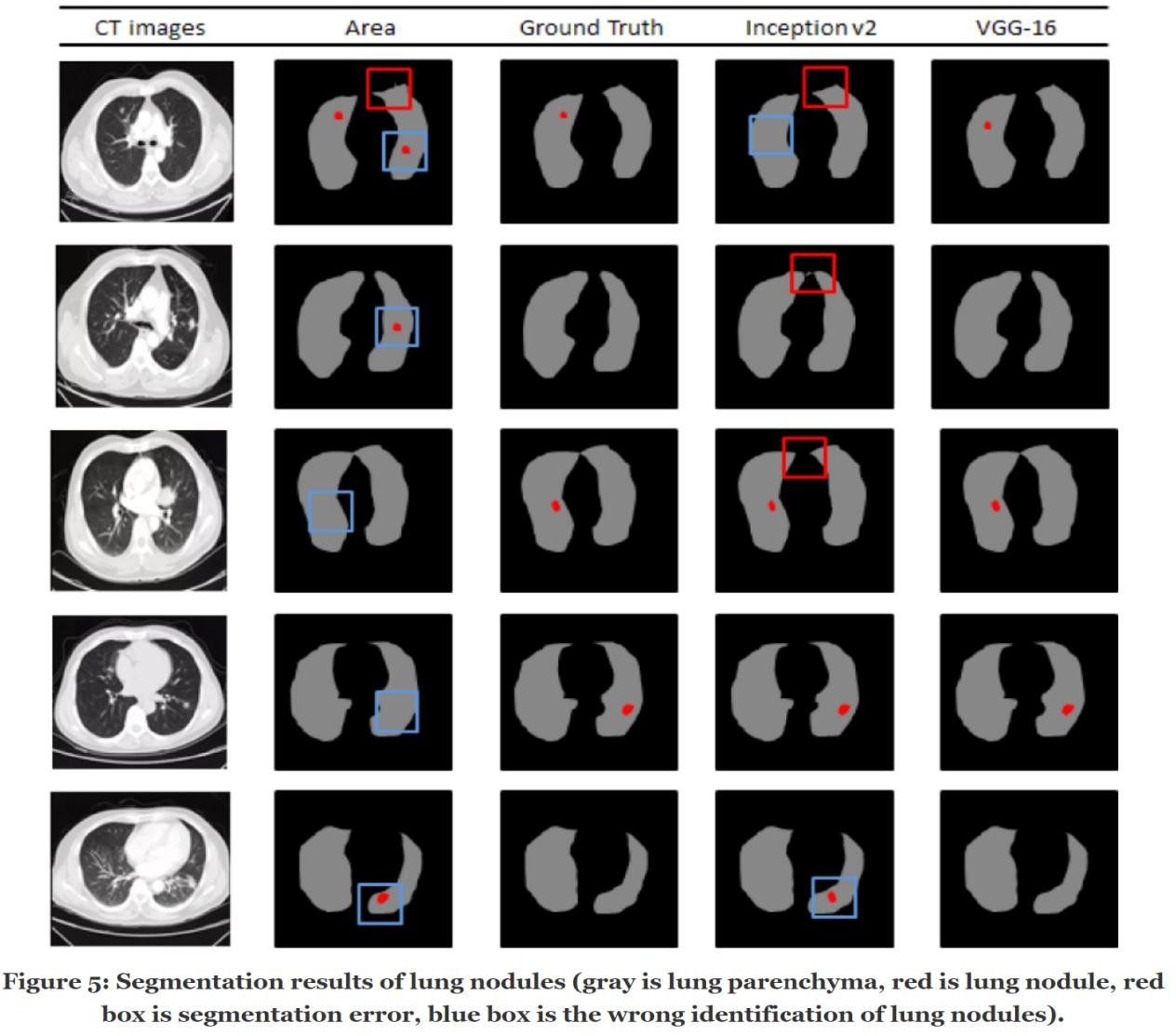
#### 

#### **CHAPTER 4**

#### **TOOLS AND SOFTWARE USED**

### 4.1 Dataset description (example)





#### **4.2 Tools description**

**OPEN CV**

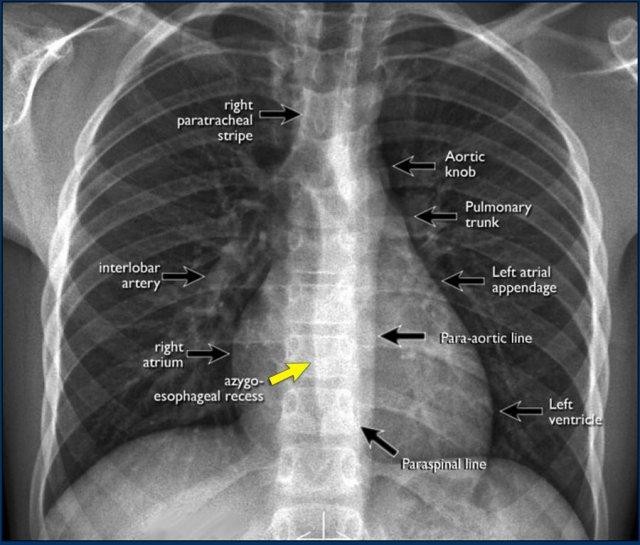
## **Computer Vision:**

It is a process by which we can understand the images and videos how they are stored and how we can manipulate and retrieve data from them. Computer Vision is the base or mostly used for Artificial Intelligence. Computer- Vision is playing a major role in self-driving cars, robotics as well as in photo correction apps.

## **OpenCV:**

It is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today’s systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

## **Chest X-ray:**

Chest x-ray uses a very small dose of ionizing radiation to produce pictures of the inside of the chest. It is used to evaluate the lungs, heart and chest wall and may be used to help diagnose shortness of breath, persistent cough, fever, chest pain or injury. It also may be used to help diagnose and monitor treatment for a variety of lung conditions such as pneumonia, emphysema and cancer. Because chest x-ray is fast and easy, it is particularly useful in emergency diagnosis and treatment.

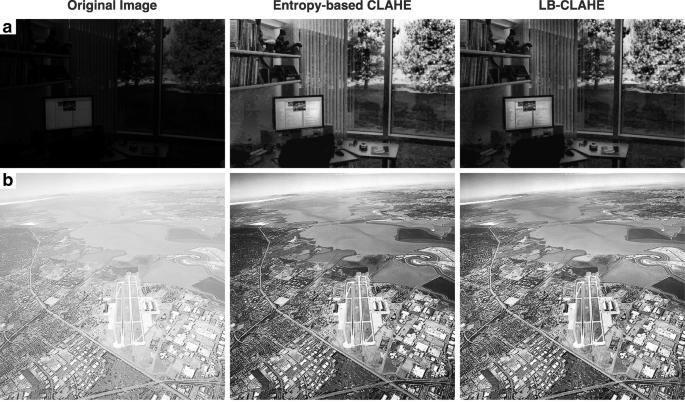
## **CLAHE Histogram Equalization**

CLAHE is a variant of Adaptive histogram equalization (AHE) which takes care of over-amplification of the contrast. CLAHE operates on small regions in the image, called tiles,

rather than the entire image. The neighboring tiles are then

combined using bilinear interpolation to remove the artificial boundaries. This algorithm can be applied to improve the contrast of images.

We can also apply CLAHE to color images, where usually it is applied on the luminance channel and the results after equalizing only the luminance channel of an HSV image are much better than equalizing all the channels of the BGR image.



## **Jakkard Index**

The Jaccard similarity index (sometimes called the Jaccard similarity *coefficient*) compares members for two sets to see which members are shared and which are distinct. It’s a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. Although it’s easy to interpret, it is extremely sensitive to small samples sizes and may give erroneous results, especially with very small samples or data sets with missing observations.



## **How to Calculate the Jaccard Index:**

The formula to find the Index is:

**Jaccard Index =**

**(The number in both sets) / (the number in either set) \* 100**

The same formula in notation is:

##### J (X, Y) = |X∩Y| / |X∪Y|

In Steps, that’s:

1. Count the number of members which are shared between both sets.
2. Count the total number of members in both sets (shared and un-shared).
3. Divide the number of shared members (1) by the total number of members (2).
4. Multiply the number you found in (3) by 100.

This percentage tells you how similar the two sets are -

1. Two sets that share all members would be 100% similar. the closer to 100%, the more similarity (e.g. 90% is more similar than 89%).
2. If they share no members, they are 0% similar.
3. The midway point — 50% — means that the two sets share half of the members.

## **Harris Corner Detector in OpenCV**

OpenCV has the function [cv.cornerHarris()](https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html) for this purpose. Its arguments are:

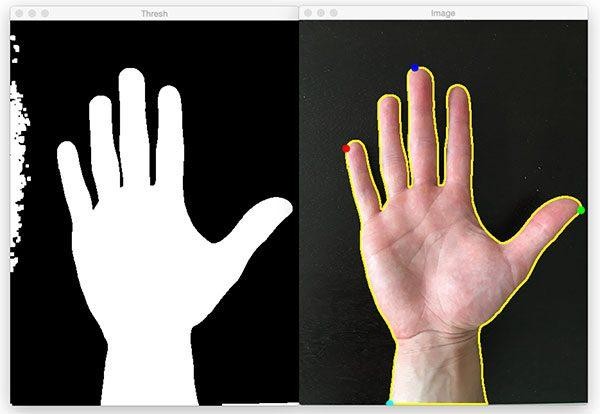
* 1. img - Input image. It should be grayscale and float32 type.
  2. blockSize - It is the size of neighborhood considered for corner detection
  3. ksize - Aperture parameter of the Sobel derivative used. 4)k - Harris detector free parameter in the equation.



## **Contour Detection**

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

1. For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
2. Since OpenCV 3.2, [findContours()](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html) no longer modifies the source image but returns a modified image as the first of three return parameters.
3. In OpenCV, finding contours is like finding white object from black background. So, remember, object to be found should be white and background should be black.



## **Black Hat**

In morphology and digital image processing, top-hat and black-hat transform are operations that are used to extract small elements and details from given images. These two types of transforms in which, the top-hat transform is defined as the difference between the input image and its opening by some structuring element, while the black-hat transform is defined as the difference between the closing and the input image. These transforms are used for various image processing tasks, such as feature extraction, background equalization, image enhancement, and others.

#### **CHAPTER 5**

#### **RESULT AND DISCUSSION**

#### **5.1 Code implementation:**

1. Using the model VGG16:

import os import shutil

os.listdir("/content/drive/MyDrive/dataset/train") os.listdir("/content/drive/MyDrive/dataset/v") import os

import cv2

import matplotlib.pyplot as plt from PIL import Image

import tensorflow as tf

from keras import backend as K

from keras.models import load\_model

from keras.preprocessing.image import img\_to\_array from tensorflow.keras.optimizers import Adam, RMSprop

from tensorflow.keras.callbacks import ReduceLROnPlateau

from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMG\_SHAPE = 224

batch\_size = 32

from tensorflow import keras

base\_model = keras.applications.VGG16(

weights='imagenet', # Load weights pre-trained on ImageNet. input\_shape=(224, 224, 3),

include\_top=False)

base\_model.summary()

base\_model.trainable = False

inputs = keras.Input(shape=(224, 224, 3))

# Separately from setting trainable on the model, we set training to False

x = base\_model(inputs, training=False)

x = keras.layers.GlobalAveragePooling2D()(x)

# A Dense classifier with a single unit (binary classification)

outputs = keras.layers.Dense(1)(x)

model = keras.Model(inputs, outputs)

model.summary()

# Important to use binary crossentropy and binary accuracy as we now have a binary classification problem

model.compile(loss=keras.losses.BinaryCrossentropy(from\_logits=Tr ue), metrics=[keras.metrics.BinaryAccuracy()])

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# create a data generator

datagen = ImageDataGenerator(

samplewise\_center=True, # set each sample mean to 0

rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)

zoom\_range = 0.1, # Randomly zoom image

width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)

height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)

horizontal\_flip=True, # randomly flip images

vertical\_flip=False) # we don't expect Bo to be upside- down so we will not flip vertically

# load and iterate training dataset

train\_it = datagen.flow\_from\_directory('/content/drive/MyDrive/dataset/train ',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='binary',

batch\_size=8)

# load and iterate validation dataset

valid\_it = datagen.flow\_from\_directory('/content/drive/MyDrive/dataset/v',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='binary',

batch\_size=8)

h1= model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, workers=10, epochs=20)

# Unfreeze the base model

base\_model.trainable = True

# It's important to recompile your model after you make any changes

# to the `trainable` attribute of any inner layer, so that your changes

# are taken into account

model.compile(optimizer=keras.optimizers.RMSprop(learning\_rate

= .000001), # Very low learning rate

loss=keras.losses.BinaryCrossentropy(from\_logits=True),

metrics=[keras.metrics.BinaryAccuracy()])

history = model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, workers=10, epochs=20)

# list all data in history

print(history.history.keys())

# summarize history for accuracy

plt.plot(history.history['binary\_accuracy'])

plt.plot(history.history['val\_binary\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

#### Using the model VGG19:

import os import shutil

os.listdir("/content/drive/MyDrive/dataset/train") os.listdir("/content/drive/MyDrive/dataset/v") import os

import cv2

import matplotlib.pyplot as plt from PIL import Image

import tensorflow as tf

from keras import backend as K

from keras.models import load\_model

from keras.preprocessing.image import img\_to\_array from tensorflow.keras.optimizers import Adam, RMSprop

from tensorflow.keras.callbacks import ReduceLROnPlateau

from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMG\_SHAPE = 224

batch\_size = 32

from tensorflow import keras

base\_model = keras.applications.VGG19(

weights='imagenet', # Load weights pre-trained on ImageNet. input\_shape=(224, 224, 3),

include\_top=False)

base\_model.summary()

base\_model.trainable = False

inputs = keras.Input(shape=(224, 224, 3))

# Separately from setting trainable on the model, we set training to False

x = base\_model(inputs, training=False)

x = keras.layers.GlobalAveragePooling2D()(x)

# A Dense classifier with a single unit (binary classification)

outputs = keras.layers.Dense(1)(x)

model = keras.Model(inputs, outputs)

model.summary()

# Important to use binary crossentropy and binary accuracy as we now have a binary classification problem

model.compile(loss=keras.losses.BinaryCrossentropy(from\_logits=Tr ue), metrics=[keras.metrics.BinaryAccuracy()])

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# create a data generator

datagen = ImageDataGenerator(

samplewise\_center=True, # set each sample mean to 0

rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)

zoom\_range = 0.1, # Randomly zoom image

width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)

height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)

horizontal\_flip=True, # randomly flip images

vertical\_flip=False) # we don't expect Bo to be upside- down so we will not flip vertically

# load and iterate training dataset

train\_it = datagen.flow\_from\_directory('/content/drive/MyDrive/dataset/train ',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='binary',

batch\_size=8)

# load and iterate validation dataset

valid\_it = datagen.flow\_from\_directory('/content/drive/MyDrive/dataset/v',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='binary',

batch\_size=8)

h1= model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, workers=10, epochs=20)

# Unfreeze the base model

base\_model.trainable = True

# It's important to recompile your model after you make any changes

# to the `trainable` attribute of any inner layer, so that your changes

# are taken into account

model.compile(optimizer=keras.optimizers.RMSprop(learning\_rate

= .000001), # Very low learning rate

loss=keras.losses.BinaryCrossentropy(from\_logits=True),

metrics=[keras.metrics.BinaryAccuracy()])

history = model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, workers=10, epochs=20)

# list all data in history

print(history.history.keys())

# summarize history for accuracy

plt.plot(history.history['binary\_accuracy'])

plt.plot(history.history['val\_binary\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

#### Using the model ResNet50:

import os import shutil

os.listdir("/content/drive/MyDrive/dataset/train") os.listdir("/content/drive/MyDrive/dataset/v") import pandas as pd

import numpy as np import keras

from keras.layers import Dense, GlobalAveragePooling2D, Dropout,

Flatten

from tensorflow.keras.optimizers import Adam, RMSprop

from tensorflow.keras.applications.resnet50 import ResNet50 from keras.preprocessing import image

from keras.models import Model import os

import cv2

import matplotlib.pyplot as plt from PIL import Image

import tensorflow as tf

from keras import backend as K

from keras.models import load\_model

from keras.preprocessing.image import img\_to\_array from tensorflow.keras.optimizers import Adam, RMSprop

from tensorflow.keras.callbacks import ReduceLROnPlateau

from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMG\_SHAPE = 224

batch\_size = 32

from tensorflow import keras

base\_model = keras.applications.ResNet50(

weights='imagenet', # Load weights pre-trained on ImageNet.

input\_shape=(224, 224, 3),

include\_top=False)

base\_model.summary()

base\_model.trainable = False

inputs = keras.Input(shape=(224, 224, 3))

# Separately from setting trainable on the model, we set training to False

x = base\_model(inputs, training=False)

x = keras.layers.GlobalAveragePooling2D()(x)

# A Dense classifier with a single unit (binary classification)

outputs = keras.layers.Dense(1)(x)

model = keras.Model(inputs, outputs)

model.summary()

# Important to use binary crossentropy and binary accuracy as we now have a binary classification problem

model.compile(loss=keras.losses.BinaryCrossentropy(from\_logits=Tr ue), metrics=[keras.metrics.BinaryAccuracy()])

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# create a data generator

datagen = ImageDataGenerator(

samplewise\_center=True, # set each sample mean to 0

rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)

zoom\_range = 0.1, # Randomly zoom image

width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)

height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)

horizontal\_flip=True, # randomly flip images

vertical\_flip=False) # we don't expect Bo to be upside- down so we will not flip vertically

# load and iterate training dataset

train\_it = datagen.flow\_from\_directory('/content/drive/MyDrive/dataset/train ',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='binary',

batch\_size=8)

# load and iterate validation dataset

valid\_it = datagen.flow\_from\_directory('/content/drive/MyDrive/dataset/v',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='binary',

batch\_size=8)

h1= model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, workers=10, epochs=20)

# Unfreeze the base model

base\_model.trainable = True

# It's important to recompile your model after you make any changes

# to the `trainable` attribute of any inner layer, so that your changes

# are taken into account

model.compile(optimizer=keras.optimizers.RMSprop(learning\_rate

= .000001), # Very low learning rate

loss=keras.losses.BinaryCrossentropy(from\_logits=True),

metrics=[keras.metrics.BinaryAccuracy()])

history = model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, workers=10, epochs=50)

# list all data in history

print(history.history.keys())

# summarize history for accuracy

plt.plot(history.history['binary\_accuracy'])

plt.plot(history.history['val\_binary\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

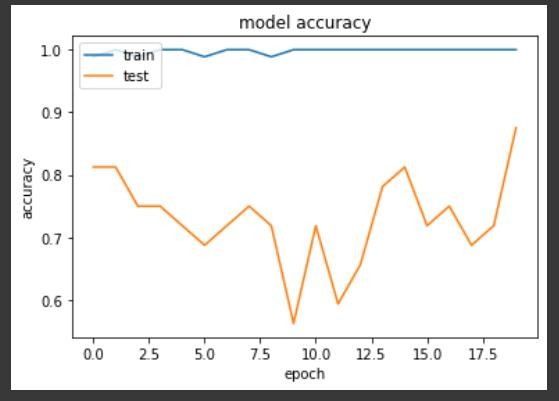
plt.ylabel('loss') plt.xlabel('epoch')

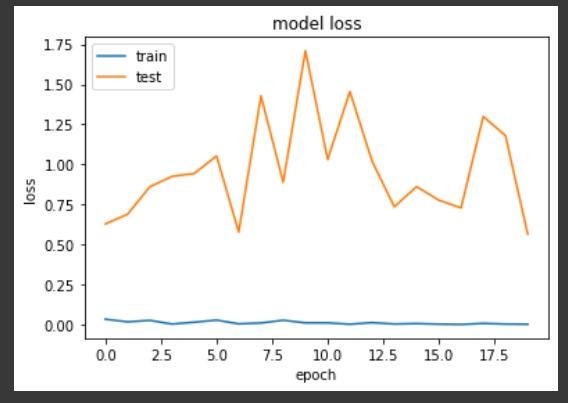
plt.legend(['train', 'test'], loc='upper left') plt.show()

**5.2 Performance evaluation:**

**OUTPUT:**

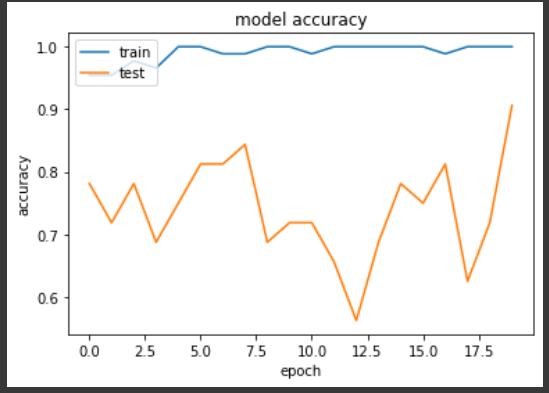
#### For VGG16:

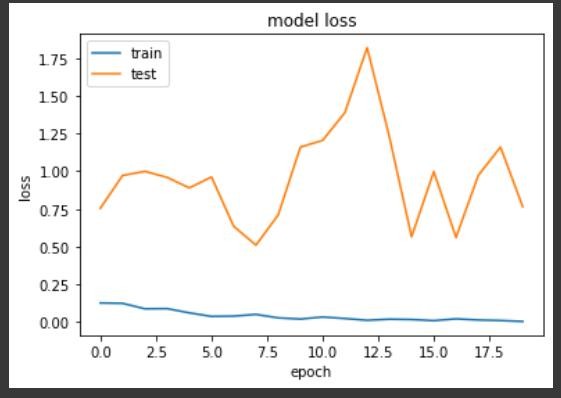




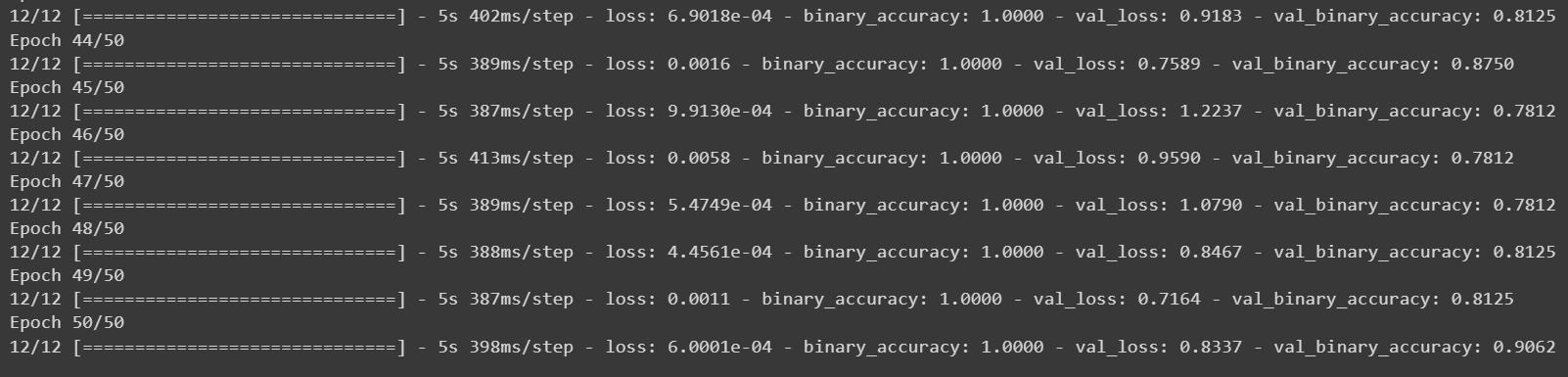
1. For VGG19:

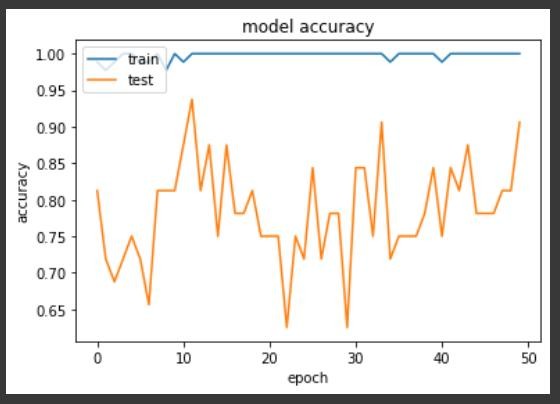


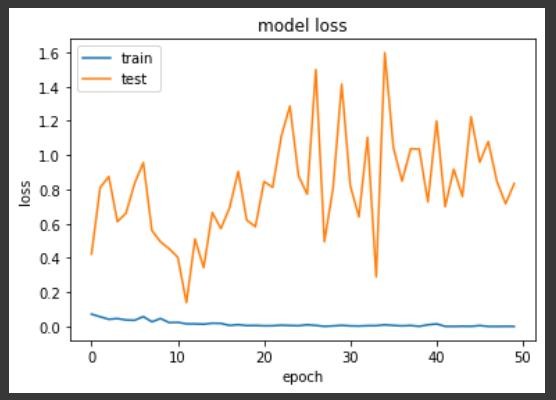




#### For ResNet50:







**Accuracy difference table:**

|  |  |  |
| --- | --- | --- |
| S. No | Model | Accuracy |
| 1. | VGG16 | 0.8750 |
| 2. | VGG19 | 0.9062 |
| 3. | ResNet50 | 0.9062 |

## **Code/Output:**





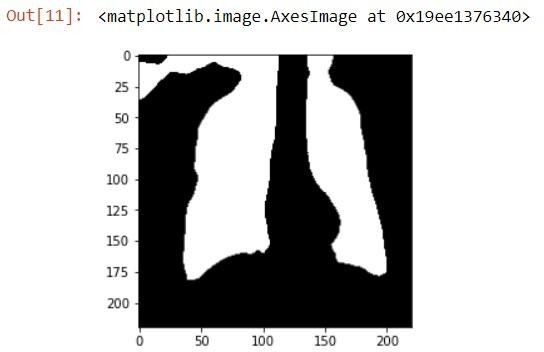
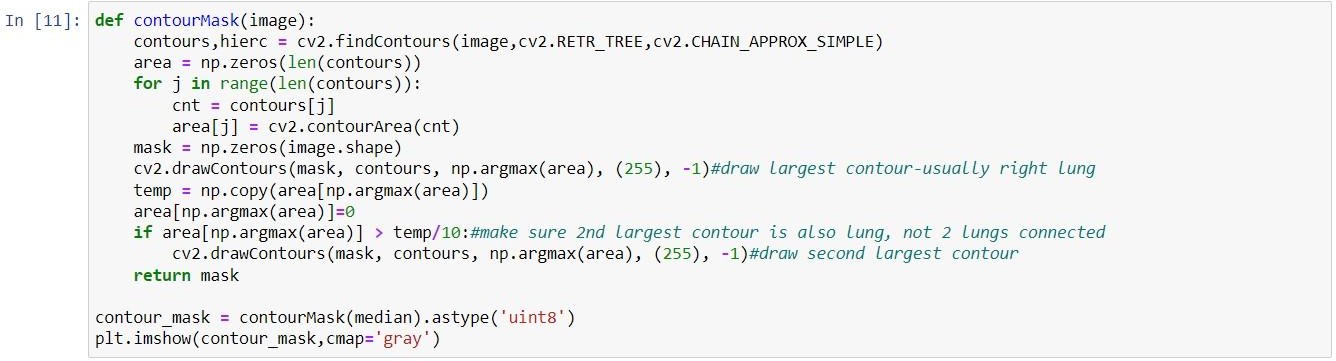






















**5.3 Results and discussions:**

We use a learning rate of 3 × 10−4 for lung segmentation. Momentum is set to 0.9. Size of mini-batch is set to 8. Resolution of input images is resized to 224 × 224 by bilinear interpolation. At the same time, we also use our own data set to implement some other convolutional neural network methods, including U-net, Deeplab-v3 and FCN , and the results of various segmentation algorithms were compared with the comprehensive lung parenchyma area manually segmented by experienced doctor. In the course of the experiment, the manually segmented images in the medical records were the ultimate gold standard.

Four representative images are selected for experiment, and our algorithm and other algorithms are used to segment the images. Because U-net is the best in all comparison methods, the images were classified in four groups, include: original lung CT images, ground trues, segmentation results from our network and segmentation results from U-net.

**CHAPTER 6**

**CONCLUSION**

# CNN:

The method in this paper is improved on the basis of the VGG-16 network, replacing part of the convolutional layer in the original network with a dilated convolution, and at the same time cancelling the pooling layer, so that the convolution kernel parameters can be unchanged.

The receptive field of the convolution kernel is enlarged, and the calculation amount is reduced and the accuracy is improved. Compared with the existing methods, the method proposed in this paper solves some shortcomings of other methods.

Traditional image processing methods cannot accurately segment the lung nodules and blood vessels attached to the edge of the lung, nor can they separate the left and right lungs that are close in distance.

The use of CNNs can solve some of the shortcomings of traditional image processing methods, and is superior to traditional image processing methods in various performance indicators, which can prove that CNNs can be used in the field of CT image segmentation.

# Open CV

Medical imaging is the hot topic nowadays, detecting diseases in a human body is the critical task for all researchers. Computer vision provides different path for recognizing objects in an image, and machine learning algorithms helps to identify efficiently. Image segmentation approaches, recognized image in to different segments such as color, texture, shapes, size, edge, objects, and regions. OpenCV, NumPy and Matplotlib tries to summaries code of thresholding, canny edge detection, clustering, and regional growth in Image Segmentation Approach Using Python… 139 python 3.6.5, given image as an input and display image as an output but get hidden patterns or meaningful information. The result section clearly shows, block-based image segmentation is one of the best solutions for medical imaging, it separates accurately background and foreground of an image (chest x-ray). Our main objective, to ensure better understanding of IS approaches in medical field which help to diagnose diseases, take a suitable action in terms of treatment and reduce the rate of patients recodes in health sector department.

**REFERENCES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No** | **Project Name** | **Publishing Year** | **Journal Name** | **Author Name** |
| **1.** | **Pneumonia Detection on Chest X-Ray Using Machine Learning Paradigm** | **2019** | **Springer** | **Tej Bahadur Chandra, Kesari Verma** |
| **2.** | **ResNet-50 vs VGG-19 vs**  **training from scratch: A comparative analysis of the segmentation and classification of Pneumonia from chest X-ray images** | **2021** | **KeAi** | [**A.Victor**](https://www.sciencedirect.com/science/article/pii/S2666285X21000558) **Ikechukwu,S.Murali, R.Deepu, R.C.Shivamurthy** |
| **3.** | **An effective approach for CT lung segmentation using mask region-based convolutional neural networks** | **2020** | **ELSEVIER** | [**Qinhua Hu, Luís**](https://www.sciencedirect.com/science/article/pii/S0933365719305871?casa_token=vNS44FXXVvYAAAAA%3AicAigMZSCVojqqHXOxrkwPKiahy3wGhVZLHmlZwXqCMjHhyhnizXnwzoQksi-zUr2d2pvL7f) **Fabrício de F. Souza, Gabriel Bandeira Holanda, Shara S.A.Alves, Francisco Hércules dos S. Silva, Tao Han, Pedro P. Rebouças Filhob.** |
| **4.** | **A Deep Learning Method for Lung** | **2019** | **IEEE** | **Hieu Trung Huynh, Vo Nguyen Nhat Anh** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Segmentation on Large Size Chest X-Ray Image** |  |  |  |
| **5.** | **Enhanced lung image segmentation using deep learning** | **2022** | **Springer** | [**Shilpa Gite**](https://link.springer.com/article/10.1007/s00521-021-06719-8)**,** [**Abhinav Mishra**](https://link.springer.com/article/10.1007/s00521-021-06719-8) **&** [**Ketan Kotecha**](https://link.springer.com/article/10.1007/s00521-021-06719-8) |
| **6.** | **Survey on image segmentation techniques.** | **2017** | **Procedia Computer Science** | **Zaitoun, N. M., and M. J. Aqel.** |
| **7.** | **Colour image segmentation** | **2010** | **AMS** | **Osman, M. K.** |
| **8.** | **”Automated segmentation procedure** | **2016** | **CITSM** | **Riza, Bob Subhan** |
| **9.** | **Contour Detection and Completion for Inpainting and Segmentation Based on Topological Gradient and Fast Marching Algorithms** | **2011** | ***IEEE***  ***Transactions on Pattern Analysis and Machine Intelligence*** | **Didier Auroux** |