LUNG SEGMENTATION USING MACHINE LEARNING

Submitted by

AVIRAL SIROTIYA (RA2011047010151)

BHARGAV SINGH JASROTIA(RA2011047010144)

SHIVYA GARG (RA2011047010140)

AYUSH SHAW (RA2011047010122)

Under the guidance of

Dr. S. Selva Kumara Samy

in partial fulfillment for the award of the degree
Of

BACHELOR OF TECHNOLOGY
in

COMPUTATIONAL INTELLIGENCE
of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

S.R.M. Nagar, Kattankulathur, Chengalpattu

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled **“LUNG SEGMENTATION”** isthe Bonafide work of **“**AVIRAL SIROTIYA (RA2011047010151) BHARGAV SINGH JASROTIA(RA2011047010144) SHIVYA GARG (RA2011047010140) AYUSH SHAW (RA2011047010122)”, who carried out the project work under my supervision. Certifiedfurther, that to the best of my knowledge the work reported herein doesnot form any other project report or dissertation on the basis of whicha degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. S. Selva Kumar Samy
Dept. of Computational Intelligence

SIGNATURE

Dr. ANNIE UTHRA
HEAD OF THE DEPARTMENT
Dept. of Computational Intelligence

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide, Dr. S. Selva Kumara Samy , his valuable guidance, consistent encouragement, personal caring, timely help andproviding 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 tome for completing this project work.

AVIRAL SIROTIYA (RA2011047010151)

BHARGAV SINGH JASROTIA(RA2011047010144)

SHIVYA GARG (RA2011047010140)

AYUSH SHAW (RA2011047010122)

ABSTRACT

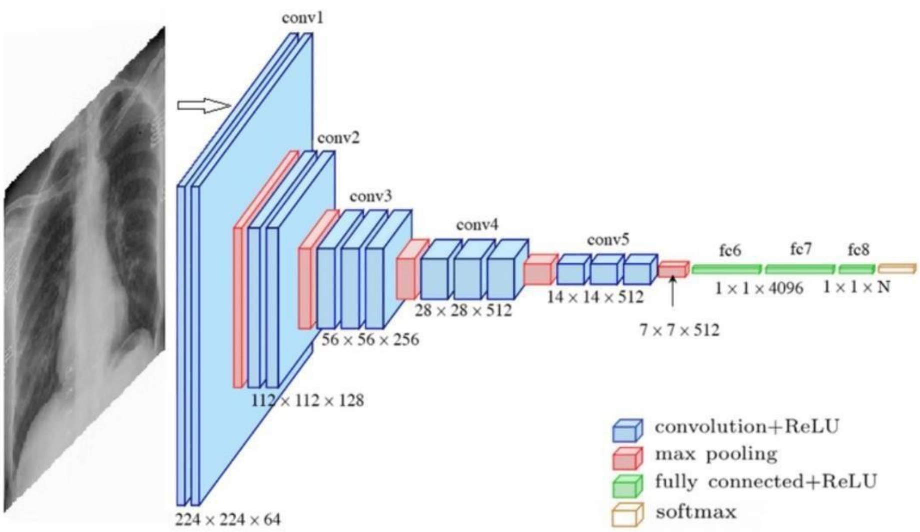
Lung CT image segmentation is a necessary initial step for lung image analysis, it is a prerequisite step to provide an accurate lung CT image analysis such as lung cancer detection. In this work, we propose a lung CT image segmentation using the U-net architecture, one of the most used architectures in deep learning for image segmentation. The architecture consists of a contracting path to extract highlevel 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.

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 (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.

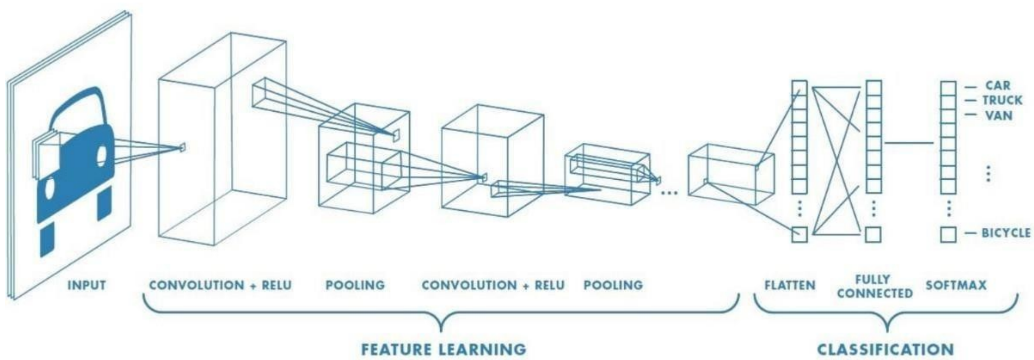
The strength of a convolutional neural network comes from a particular kind of layer called the convolutional layer.

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:



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]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.

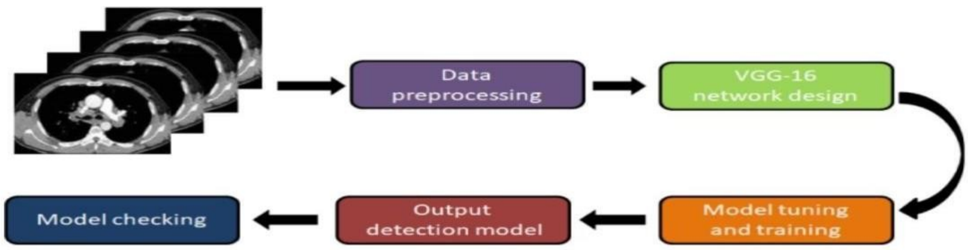


Figure 1: Design process of lung nodule detection system.

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.

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, 9, 13, 23, 24]. 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.

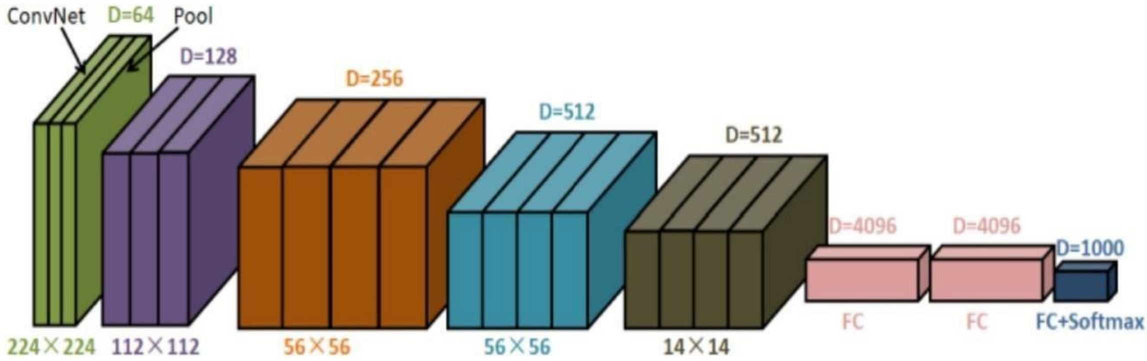


Figure 2: Network structure of VGGNet.

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.

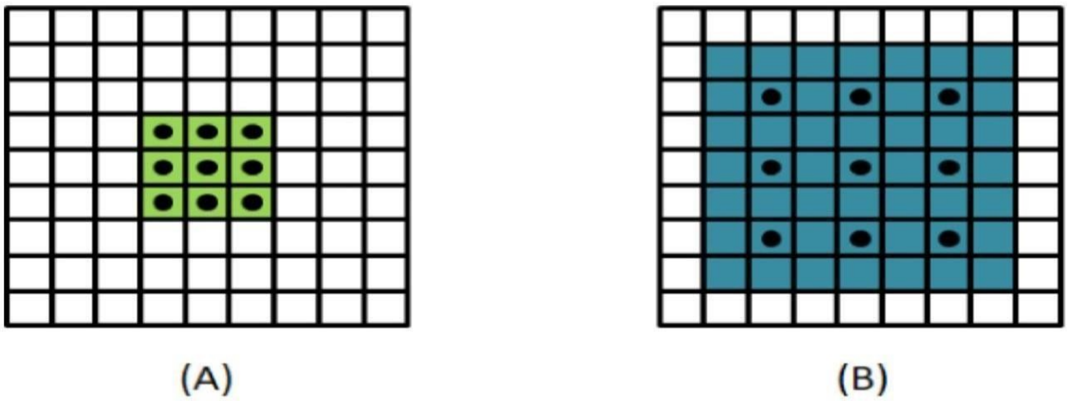


Figure 3: Principle explanation of dilated convolution.

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, k_{sizes} represents the size of the convolution kernel, and r_{rate} represents the size of the dilated coefficient.

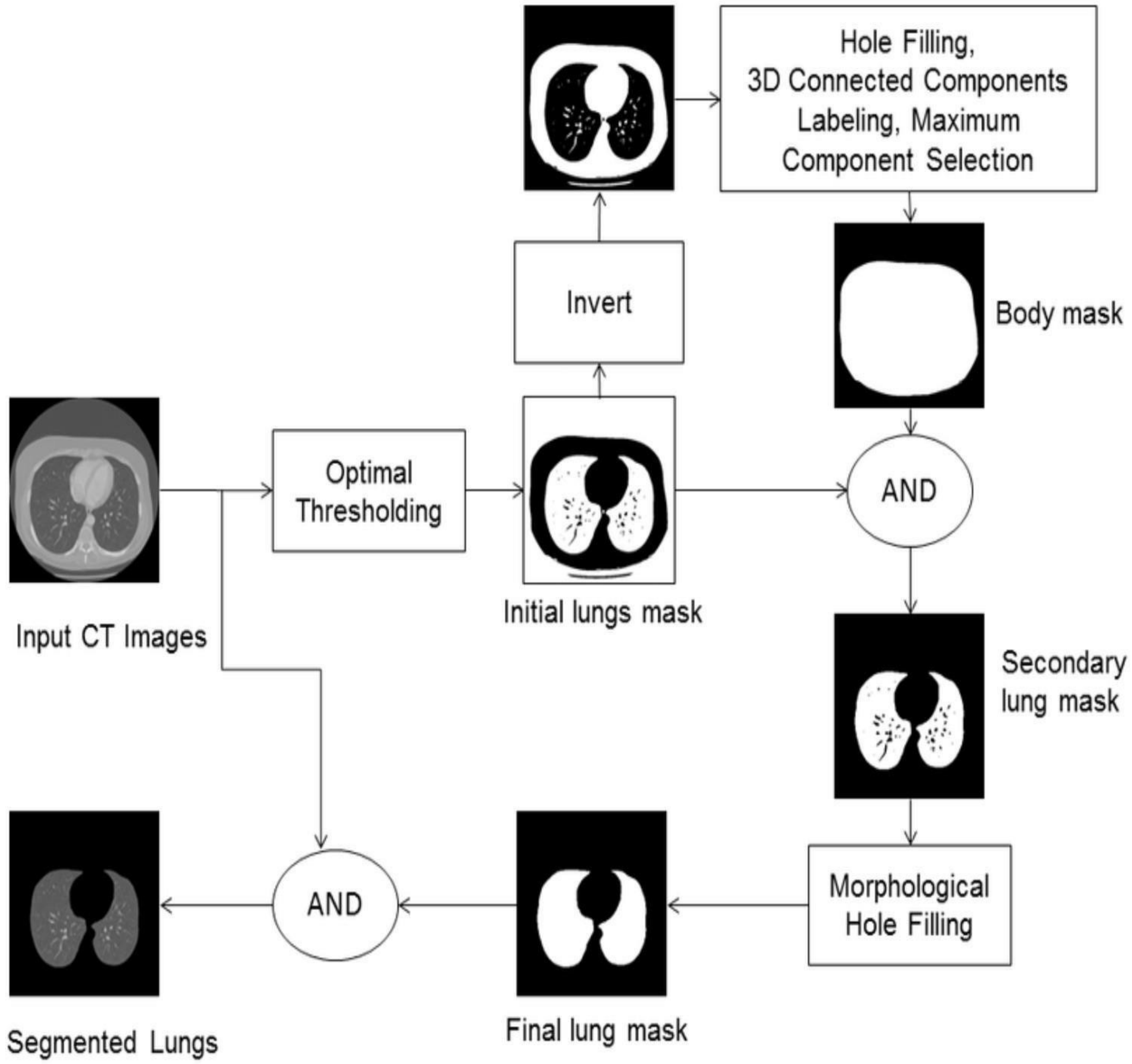
LITERATURE REVIEW

S. No	Project Name	Publishing Year	Journal Name	Author Name
1.	Pneumonia Detection on Chest X-Ray Using Machine Learning Paradigm[1]	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[2]	2021	KeAi	A.Victor Ikechukwu,S.Murali, R.Deepu, R.C.Shivamurthy
3.	An effective approach for CT lung segmentation using mask region-based convolutional neural networks[3]	2020	ELSEVIER	Qinhua Hu, Luís 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[4]			
5.	**Enhanced lung image segmentation using deep learning[5]**	**2022**	**Springer**	**Shilpa Gite, Abhinav Mishra & Ketan Kotecha**
6.	**Survey on image segmentation techniques[6]**	**2017**	**Procedia Computer Science**	**Zaitoun, N. M., and M. J. Aqel.**
7.	**Colour image segmentation [7]**	**2010**	**AMS**	**Osman, M. K.**
8.	**Automated segmentation procedure[8]**	**2016**	**CITSM**	**Riza, Bob Subhan**
9.	**Contour Detection and Completion for Inpainting and Segmentation Based on Topological Gradient and Fast Marching Algorithms[9]**	**2011**	***IEEE Transactions on Pattern Analysis and Machine Intelligence***	**Didier Auroux**

PROPOSED METHODOLOGY

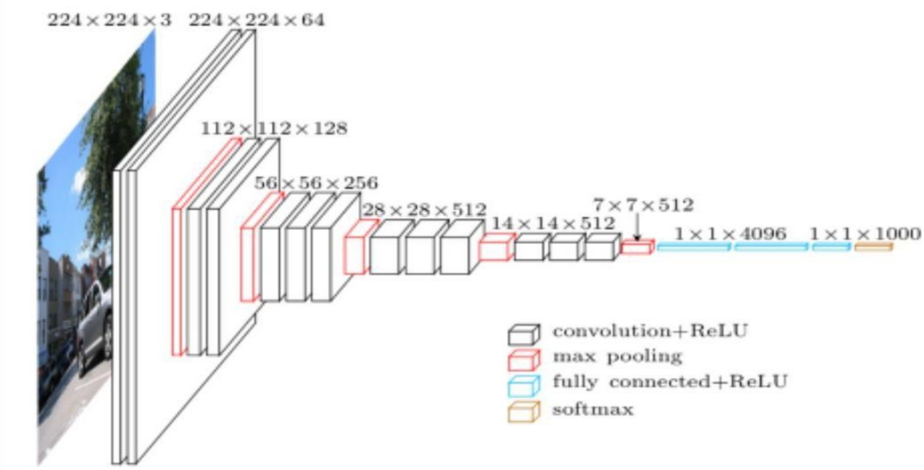
Architecture diagram:



3.1 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.

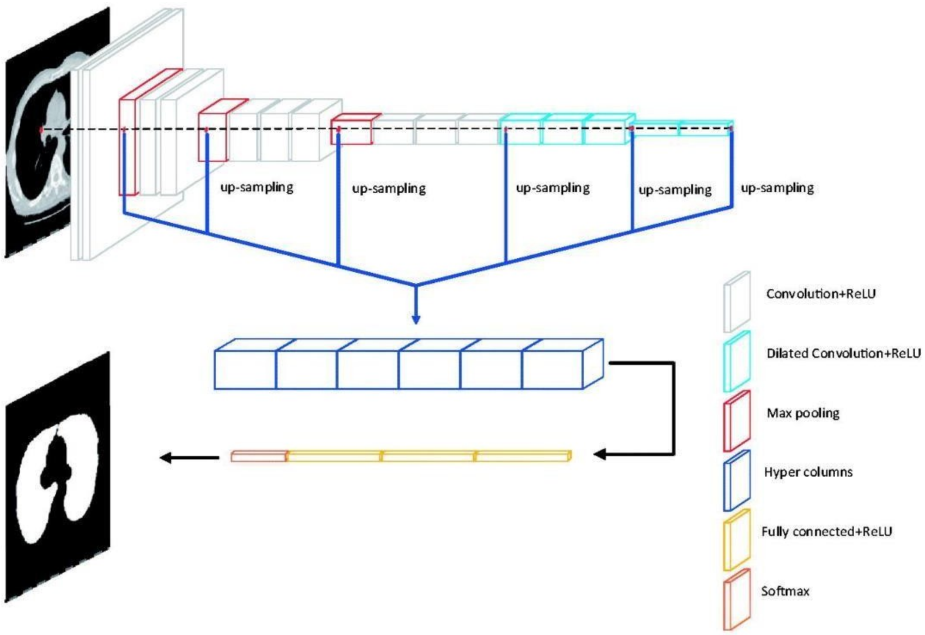
However, VGGNet consists of 138 million parameters, which can be a bit challenging to handle. VGG can be achieved through transfer Learning. In which the model is pretrained on a dataset and the parameters are updated for better accuracy and you can use the parameters values.



Architecture of VGG16

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.



TOOLS AND SOFTWARE USED

Dataset description (example)

Table 1: Distribution ratio of data set.

Experimental data set	Training set	Validation set	Test set
2045	1500	545	205

Table 2: Comparison of segmentation effect and accuracy of different segmentation methods.

Methods	XOR	Hausdorff	Jaccard	Acc	Sen	Spe	FNR	FPR
Threshold	0.634	0.847	0.678	0.679	0.667	0.569	0.333	0.569
Contour	0.486	0.576	0.744	0.724	0.755	0.621	0.245	0.379
Area	0.741	0.479	0.768	0.733	0.698	0.624	0.302	0.376
Statistics	0.614	0.581	0.799	0.624	0.774	0.685	0.226	0.315
Inception v2	0.102	0.401	0.896	0.916	0.867	0.824	0.133	0.176
VGG-16	0.094	0.396	0.957	0.971	0.926	0.899	0.074	0.101

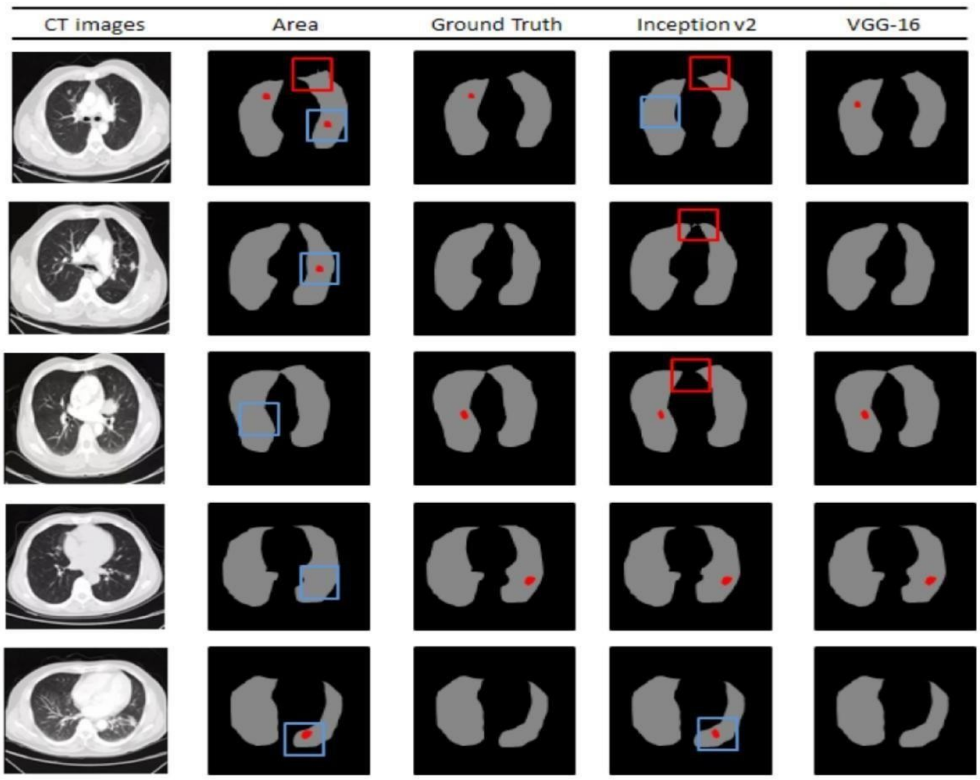


Figure 5: Segmentation results of lung nodules (gray is lung parenchyma, red is lung nodule, red box is segmentation error, blue box is the wrong identification of lung nodules).

RESULT AND DISCUSSION

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.
```

```
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()
```

2) 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.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()
```

3) 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)
```

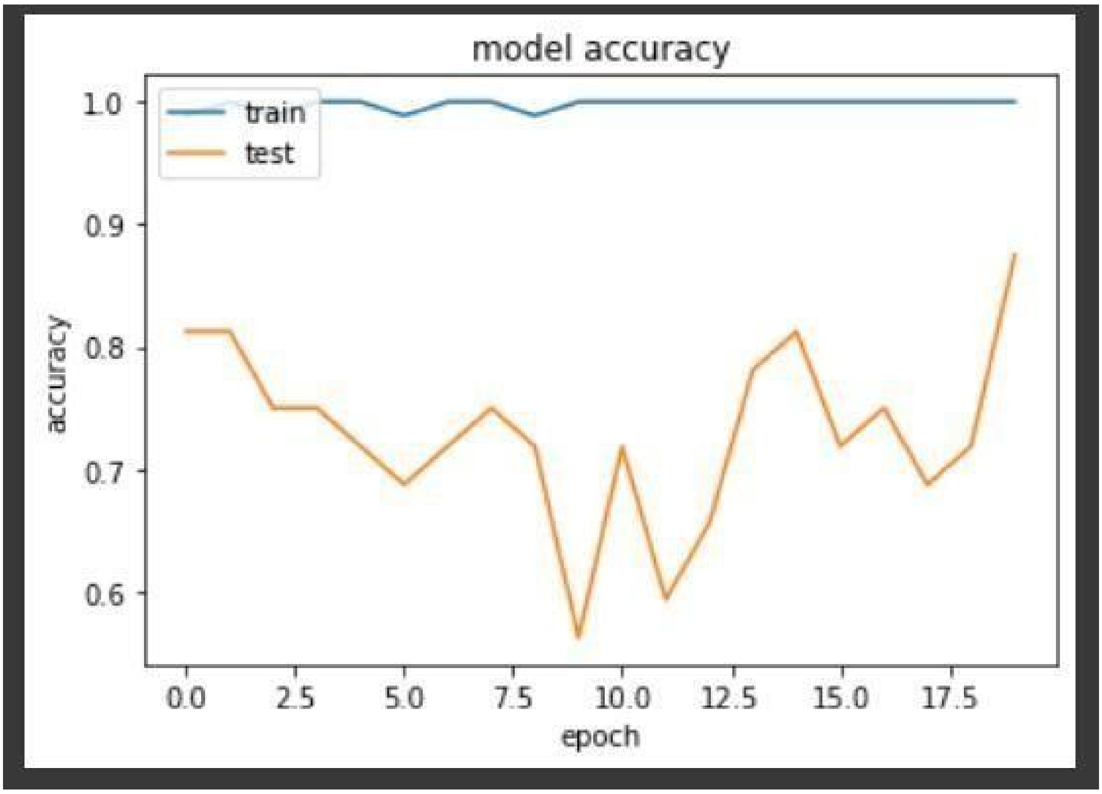
```
# Unfreeze the base model base_model.trainable

= 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()
```

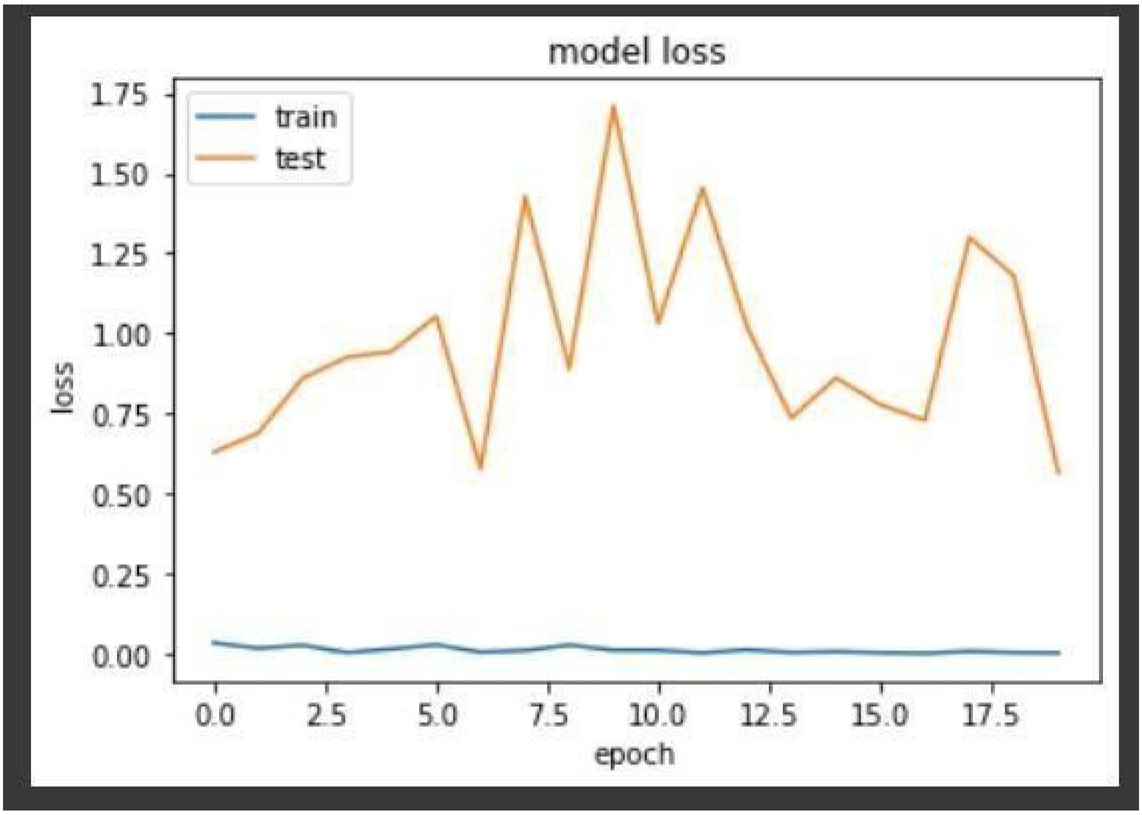
Performance evaluation:

OUTPUT:

4) For VGG16:

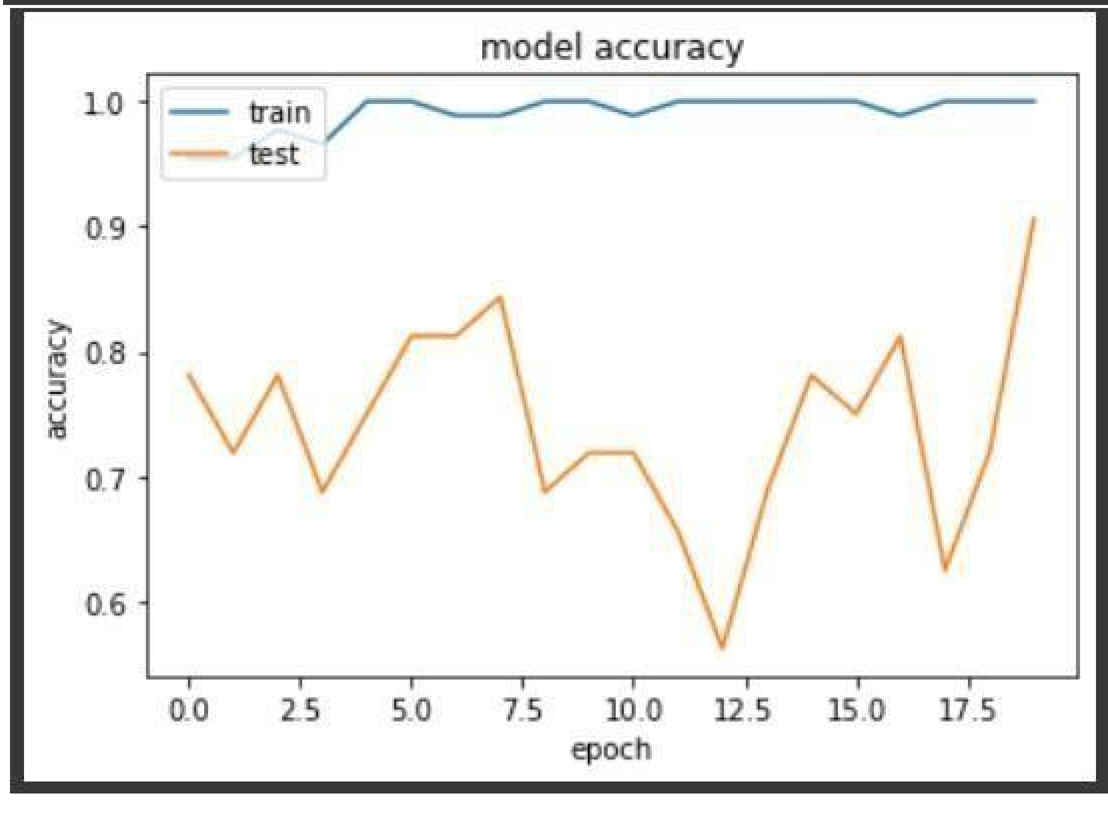
```
12/12 [==============================] - 6s 426ms/step - loss: 0.0127 - binary_accuracy: 1.0000 - val_loss: 1.0312 - val_binary_accurac
Epoch 12/20
12/12 [==============================] - 6s 423ms/step - loss: 0.0045 - binary_accuracy: 1.0000 - val_loss: 1.4545 - val_binary_accuracy: 0.5938
Epoch 13/20
12/12 [==============================] - 6s 429ms/step - loss: 0.0141 - binary_accuracy: 1.0000 - val_loss: 1.0201 - val_binary_accuracy: 0.6562
Epoch 14/20
12/12 [==============================] - 6s 464ms/step - loss: 0.0056 - binary_accuracy: 1.0000 - val_loss: 0.7360 - val_binary_accuracy: 0.7812
Epoch 15/20
12/12 [==============================] - 6s 420ms/step - loss: 0.0085 - binary_accuracy: 1.0000 - val_loss: 0.8622 - val_binary_accuracy: 0.8125
Epoch 16/20
12/12 [==============================] - 6s 442ms/step - loss: 0.0048 - binary_accuracy: 1.0000 - val_loss: 0.7775 - val_binary_accuracy: 0.7188
Epoch 17/20
12/12 [==============================] - 6s 427ms/step - loss: 0.0029 - binary_accuracy: 1.0000 - val_loss: 0.7295 - val_binary_accuracy: 0.7500
Epoch 18/20
12/12 [==============================] - 6s 422ms/step - loss: 0.0095 - binary_accuracy: 1.0000 - val_loss: 1.2987 - val_binary_accuracy: 0.6875
Epoch 19/20
12/12 [==============================] - 6s 434ms/step - loss: 0.0054 - binary_accuracy: 1.0000 - val_loss: 1.1798 - val_binary_accuracy: 0.7188
Epoch 20/20
12/12 [==============================] - 6s 470ms/step - loss: 0.0042 - binary_accuracy: 1.0000 - val_loss: 0.5659 - val_binary_accuracy: 0.8750
```

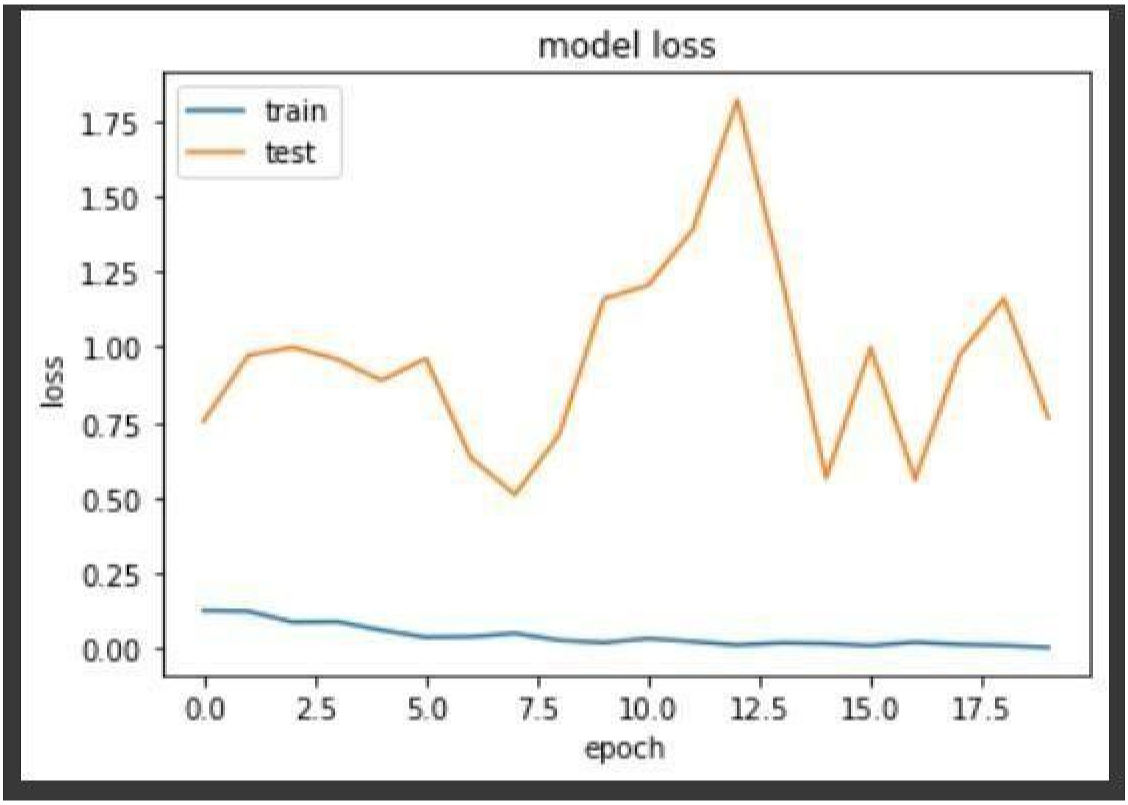




5) For VGG19:

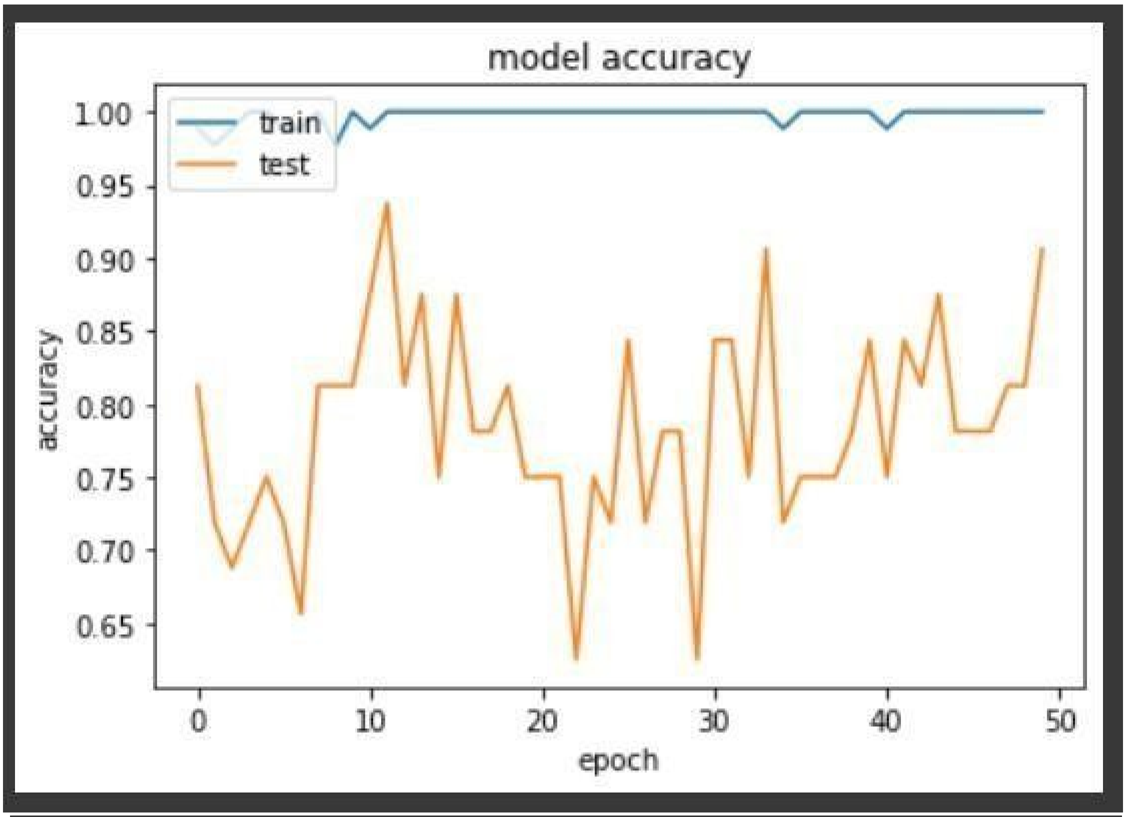
```
12/12 [==============================] - 233s 20s/step - loss: 0.0324 - binary_accuracy: 0.9886 - val_loss: 1.2052 - val_binary_accuracy: 0.7188
Epoch 12/20
12/12 [==============================] - 215s 18s/step - loss: 0.0224 - binary_accuracy: 1.0000 - val_loss: 1.3895 - val_binary_accuracy: 0.6562
Epoch 13/20
12/12 [==============================] - 214s 18s/step - loss: 0.0105 - binary_accuracy: 1.0000 - val_loss: 1.8206 - val_binary_accuracy: 0.5625
Epoch 14/20
12/12 [==============================] - 212s 18s/step - loss: 0.0179 - binary_accuracy: 1.0000 - val_loss: 1.2299 - val_binary_accuracy: 0.6875
Epoch 15/20
12/12 [==============================] - 214s 18s/step - loss: 0.0156 - binary_accuracy: 1.0000 - val_loss: 0.5652 - val_binary_accuracy: 0.7812
Epoch 16/20
12/12 [==============================] - 216s 18s/step - loss: 0.0083 - binary_accuracy: 1.0000 - val_loss: 0.9976 - val_binary_accuracy: 0.7500
Epoch 17/20
12/12 [==============================] - 213s 18s/step - loss: 0.0201 - binary_accuracy: 0.9886 - val_loss: 0.5604 - val_binary_accuracy: 0.8125
Epoch 18/20
12/12 [==============================] - 215s 18s/step - loss: 0.0126 - binary_accuracy: 1.0000 - val_loss: 0.9724 - val_binary_accuracy: 0.6250
Epoch 19/20
12/12 [==============================] - 215s 18s/step - loss: 0.0092 - binary_accuracy: 1.0000 - val_loss: 1.1608 - val_binary_accuracy: 0.7188
Epoch 20/20
12/12 [==============================] - 215s 19s/step - loss: 0.0029 - binary_accuracy: 1.0000 - val_loss: 0.7649 - val_binary_accuracy: 0.9062
```

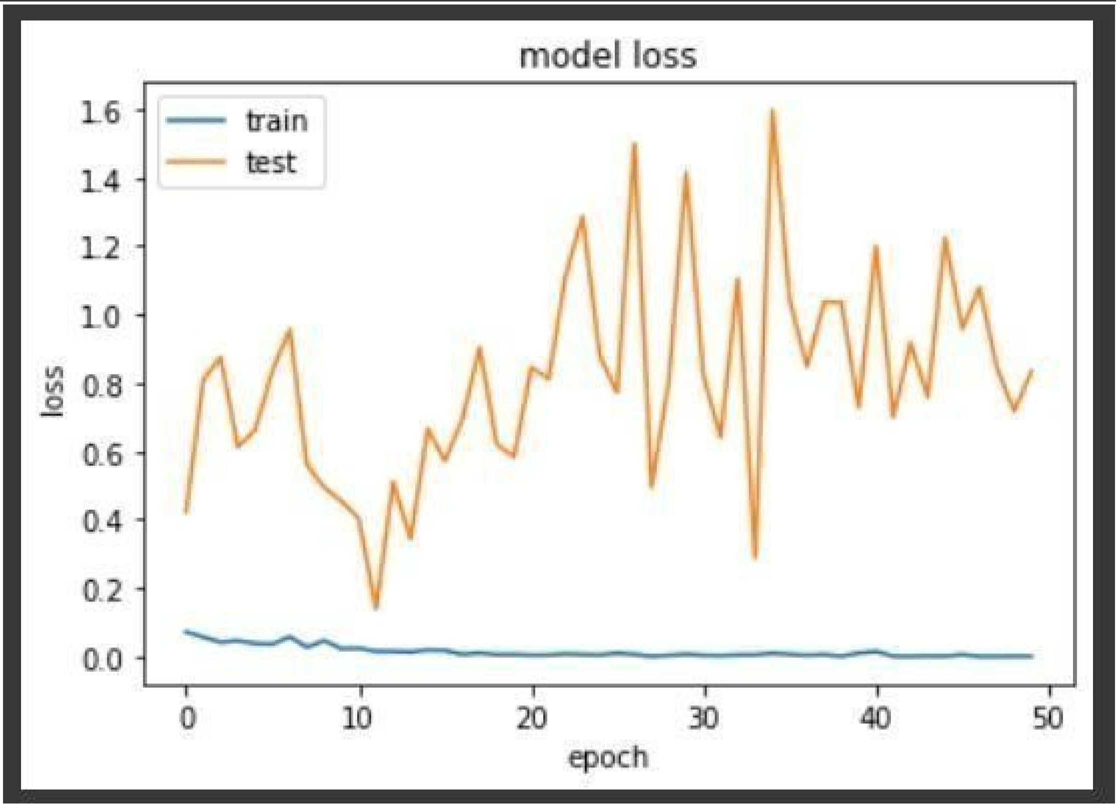




6) For ResNet50:

```
12/12 [==============================] - 5s 402ms/step - loss: 6.9018e-04 - binary_accuracy: 1.0000 - val_loss: 0.9183 - val_binary_accuracy: 0.8125
Epoch 44/50
12/12 [==============================] - 5s 389ms/step - loss: 0.0016 - binary_accuracy: 1.0000 - val_loss: 0.7589 - val_binary_accuracy: 0.8750
Epoch 45/50
12/12 [==============================] - 5s 387ms/step - loss: 9.9130e-04 - binary_accuracy: 1.0000 - val_loss: 1.2237 - val_binary_accuracy: 0.7812
Epoch 46/50
12/12 [==============================] - 5s 413ms/step - loss: 0.0058 - binary_accuracy: 1.0000 - val_loss: 0.9590 - val_binary_accuracy: 0.7812
Epoch 47/50
12/12 [==============================] - 5s 389ms/step - loss: 5.4749e-04 - binary_accuracy: 1.0000 - val_loss: 1.0790 - val_binary_accuracy: 0.7812
Epoch 48/50
12/12 [==============================] - 5s 388ms/step - loss: 4.4561e-04 - binary_accuracy: 1.0000 - val_loss: 0.8467 - val_binary_accuracy: 0.8125
Epoch 49/50
12/12 [==============================] - 5s 387ms/step - loss: 0.0011 - binary_accuracy: 1.0000 - val_loss: 0.7164 - val_binary_accuracy: 0.8125
Epoch 50/50
12/12 [==============================] - 5s 398ms/step - loss: 6.0001e-04 - binary_accuracy: 1.0000 - val_loss: 0.8337 - val_binary_accuracy: 0.9062
```


model accuracy
train
test
accuracy
epoch
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
10
20
30
40
50


model loss
train
test
loss
epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

Accuracy difference table:

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

Results and discussions:

We use a learning rate of $3 \times 10{-}4$ 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.

CONCLUSION

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.

REFERENCES

[1] Chandra, T.B., Verma, K. (2020). Pneumonia Detection on Chest X-Ray Using Machine Learning Paradigm. In:
Chaudhuri, B., Nakagawa, M., Khanna, P., Kumar, S. (eds) Proceedings of 3rd International Conference on Computer Vision and Image Processing. Advances in Intelligent Systems and Computing, vol 1022. Springer, Singapore. https://doi.org/10.1007/978-981-32-9088-4_3

[2] A. Victor Ikechukwu, S. Murali, R. Deepu, R.C. Shivamurthy,
ResNet-50 vs VGG-19 vs training from scratch: A comparative analysis of the segmentation and classification of
Pneumonia from chest X-ray images, Global Transitions Proceedings, Volume 2, Issue 2, 2021, Pages 375-381, ISSN 2666 285X, https://doi.org/10.1016/j.gltp.2021.08.027.

[3] Qinhua Hu, Luís Fabrício de F. Souza, Gabriel Bandeira Holanda, Shara S.A. Alves, Francisco Hércules dos S.
Silva, Tao Han, Pedro P. Rebouças
Filho,
An effective approach for CT lung segmentation using mask region-based convolutional neural networks,
Artificial Intelligence in Medicine, Volume 103, 2020, 101792, ISSN 0933-
3657,
https://doi.org/10.1016/j.artmed.2020.101792
.

[4] H. Trung Huynh and V. Nguyen Nhat Anh, "A Deep Learning Method for Lung Segmentation on Large Size Chest X-Ray Image," 2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF), 2019, pp. 1-5, doi: 10.1109/RIVF.2019.8713648.

[5] Gite, S., Mishra, A. & Kotecha, K. Enhanced lung image segmentation using deep learning. *Neural Comput &*
Applic (2022). https://doi.org/10.1007/s00521-021-
06719-8