

Tallinn University of Technology
School of Information Technologies

Olutosin Ajibola Ademola

184602IVEM

**3D DEEP MULTI-NEURAL NETWORKS USING
SERIALIZATION AND SIGMOID AVERAGING
FOR LUNG NODULE CLASSIFICATION**

Master's thesis

Supervisor: Professor Olev Martens
PhD

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Olutosin Ajibola Ademola

184602IVEM

**3D SÜGAVAD MULTI-NÄRVIVÕRGUD
JÄRJESTIKUSTAMISE JA SIGMOID KESKMISE
KASUTAMISEGA KOPSU MOODULITE
DETEKTEERIMISEKS JA
KLASSIFITSEERIMISEKS**

Magistritöö

Juhendaja: Professor Olev Märtens
Doktorikraad

Tallinn 2020

Author's declaration of originality

I hereby certify that I am the main author of this master's thesis. All the used materials, resources and the works of other people related to this thesis are referenced appropriately. Also, this thesis has not been presented anywhere for examination purpose.

Name of author: Olutosin Ajibola Ademola

06.05.2020

Abstract

A computer aided detection (CADE) and diagnosis (CADx) application software for the analysis of lung nodules have shown great benefits over the past years. It helps radiologists and medical professionals to give precise and accurate diagnosis result with a high confidence level. There are a quite diverse modalities of medical images, however, computed tomography (CT) has been found to be the most economical and easily accessible medical imaging technique. A traditional CADE application combines computer vision/image processing techniques and machine learning algorithms that extract salient features from the region of interest of the image to generate a single features vector mapped by the machine learning algorithms. The setback associated with this method is that it is very important to determine the salient features which would be the best descriptor of the image as it determines the level of false positive rate (FPR). This becomes more difficult as the image becomes complex. FPR has been the major reason why CADE software has not been fully adopted for lung cancer diagnosis, and thus, a need to develop a robust approach that can minimize the rate of false positive in the model is inevitable. In this master's thesis, a "*Deep Multi Neural 3D Convolutional Neural Networks Using Serialization and Sigmoid Averaging*" was developed using data randomization and variational hyperparameters. The false positive rate of each generated by each model was compared with averaged false positive rate of the three models stacked. The ensemble networks reduced these error rates (21%, 20%, and 16%) generated by the models ($3D CNN_{i-k}$) respectively to an averaged error rate of 19%.

This thesis is written in English and it is 90 pages long, including 5 chapters, 29 figures, and 3 tables.

Keywords: 3D convolutional neural networks, ensemble models, multi neural networks, lung nodule detection and classification.

Annotatsioon

Kopsu noodulite analüüsimiseks kasutatav arvuti abil tuvastav (CADE) ja diagnoosi(CADx) tarkvara on viimastel aastatel näidanud suurt kasu. See aitab radioloogidel ja meditsiinitöötajatel anda täpsed diagnoosimise tulemusi kõrge usaldusnivooga. Kompuutertomograafia (CT) on osutunud kõige ökonoomsemaks ja hõlpsamini juurdepääsetavaks meditsiiniliseks pildistamise tehnikaks. Traditsiooniline CADe rakendus ühendab arvutinägemise / pilditöötuse tehnikaid ja masinõppe algoritme, mis eraldavad pildi huvipakkuvast piirkonnast olulised tunnused, et genereerida masinõppe algoritmide abil üks kaardistatud omaduste vektor. Väga oluline on kindlaks teha olulised omadused, mis annaks parima kujutise kirjelduse, kuna see määrab valepositiivse määra (FPR) taseme. FPR on olnud peamine põhjus, miks CADe tarkvara ei ole kopsuvähi diagnoosimiseks täielikult kasutusele võetud. Seetõttu on vajadus töötada välja lähenemine, mis minimeeriks mudelis valepositiivsete esinemissageduse. Selles magistritöös töötati välja “3D sügavad multi-närvivõrgud järjestikustamise ja sigmoid keskmise kasutamisega kopsu moodulite klassifitseerimiseks”, kasutades andmete randomiseerimist ja variatsioonilisi hüperparameetreid. Iga mudeli genereeritud valepositiivse määra võrreldi kolme vinnastatud mudeli keskmise valepositiivsete tulemuste määraga. Ansamblivõrgud vähendasid nende mudelite (3D CNNi – k) tekitatud veamäärasid (21%, 20% ja 16%) vastavalt, keskmiseks veamääraks jäi 19%.

See lõputöö on kirjutatud inglise keeles ja see on 90 lehekülge pikk, sisaldades 5 peatükki, 29 joonist ja 3 tabelit.

Märksõnad: 3D-konvolutsioonilised närvivõrgud, ansamblimudel, mitmiknärvivõrgud, kopsunoodulite tuvastamine ja klassifitseerimine.

List of abbreviations and terms

1D	One Dimensional
2D	Two Dimensional
3D CNN	Three Dimensional Convolutional Neural Network
AI	Artificial Intelligence
ANNs	Artificial Neural Networks
CADe	Computer Aided Detection
CADeC	Computer Aided Detection and Classification
CADx	Computer Aided Diagnosis
CapsNet	Capsule Network
CT	Computed Tomography
DBN	Deep Belief Network
DICOM	Digital Imaging and Communications in Medicine
DRL	Deep Reinforcement Learning
FPR	False Positive Rate
GPUs	Graphical Processing Units
IDE	Integrated Development Environment
LIDC-IDRI	Lung Image Database Consortium Image Database Resource Initiative
MHD/RAW	MetaImage/Raw
PC	Personal Computer
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SA	Stacked Autoencoder
Tanh	Hyperbolic Tangent
WHO	World Health Organization

Table of contents

Author’s declaration of originality	3
Abstract.....	4
Annotatsioon.....	5
List of abbreviations and terms	6
List of figures	9
List of tables	11
Acknowledgement.....	12
1 Introduction.....	13
1.1 Background.....	14
1.2 Problem Statement.....	15
1.3 Project Objective	16
1.4 Project Requirements and Specifications	17
1.5 Motivation and Research Contribution.....	17
1.6 Chapter Review	18
2 Overview of Deep Neural Networks.....	20
2.1 DNN Introduction	20
2.1.1 Building Blocks of DNN	21
2.1.2 Network Propagation	22
2.2 Building Blocks of Convolutional Neural Networks	24
2.2.1 2D Convolutional Neural Networks	25
2.2.2 3D Convolutional Neural Networks.....	26
2.3 Deep 3D Multi-Neural Networks	27
3 State of the Art and Related Works	30
3.1 Lung Nodule Detection and Classification Phases.....	30

3.1.1	Lung Image Acquisition	30
3.1.2	Lung Image Preprocessing.....	31
3.1.3	Lung Feature Extraction	31
3.1.4	Post Processing and Classification	32
4	Methodology and Project Development	34
4.1	Dataset Description.....	34
4.2	Development Environment.....	35
4.3	Implementation Pipelines	36
4.3.1	3D CNN Multi-Neural Networks Pipeline	36
4.3.2	Image Pre-processing Phase	37
4.3.3	Ensemble 3D CNN Models Development.....	39
4.3.4	CADeC Software Inference Engine Development	47
4.4	Ensemble 3D CNN Models and Inference Engine Evaluation.....	49
4.5	Guidelines on Running Inference Application for Nodule Classification	52
5	Conclusion	55
5.1	Conclusion	55
5.2	Challenges	56
5.3	Future Research.....	56
Appendix 1		57
Directory Sorting Code		57
3D CT Scans Preprocessing Code.....		60
Appendix 2		65
3D CNN _i Model Code		65
3D CNN _j Model Code		67
3D CNN _k Model Code		70
Appendix 3		73
CADeC Software Inference Engine Code.....		73
References		87

List of figures

Figure 1. Structure of Human Neuron and Artificial Neuron [16].	20
Figure 2. A multilayer perceptron fully connected neural networks [19].	21
Figure 3. Forward and Backward propagation in neural network [19].	22
Figure 4. Global and local minima visualization during backward propagation [23].	23
Figure 5. Different modalities in medical imaging [25].	24
Figure 6. (512 x 512) computed tomography scans of the lung [26].	25
Figure 7. 2D CNN on (64x64) CT scan using (5x5) kernel filter [26].	26
Figure 8. 3D CNN on (64x64) CT scan using (5x5x5) kernel filter [26].	26
Figure 9. A simple Autoencoder network.	28
Figure 10. A 3D Convolutional Autoencoder [29].	29
Figure 11. A 3D DBN (Deep Belief Networks) [30].	29
Figure 12. Sensitivity and FPR of some of the different models proposed in different literature studies [3].....	33
Figure 13. Software packages and frameworks used for development.	35
Figure 14. Structure of the implementation pipeline of the Ensemble 3D CNN models.	36
Figure 15. A single 512 x 512 pixels single channel CT- scan of the lung marked as nodule.	37
Figure 16. Flowchart of image pre-processing phase to extract the 3D array and dumped in NumPy file.....	38
Figure 17. The architecture of $3D CNN_i$ model.	40
Figure 18. The architecture of the $3D CNN_j$ model.	41
Figure 19. Plot showing the training and validation accuracy of $3D CNN_i$ over 25 epochs.....	43
Figure 20. Plot showing the training and validation loss of model $3D CNN_i$ over 25 epochs.....	43

Figure 21. Plot showing the training and validation loss of $3D CNN_j$ over 50 epochs.	43
Figure 22. Plot showing the training and validation accuracy of $3D CNN_j$ over 50 epochs.....	44
Figure 23. The architecture of the $3D CNN_k$ model.....	45
Figure 24. Plot showing the training and validation accuracy of $3D CNN_k$ over 200 epochs.....	46
Figure 25. Plot showing the training and validation accuracy of $3D CNN_k$ over 200 epochs.....	46
Figure 26. Blocks showing the implementation pipeline of the CADeC software inference engine.....	48
Figure 27. Project directory showing ensemble model weights and the inference software written in python.	52
Figure 28. Dicom files of a patient classified as volume nodule using the application in a light mode.	53
Figure 29. MHD/RAW CT files of a patient classified as volume nodule using the application in a light mode.	54

List of tables

Table. 1. Confusion matrix showing the four base parameters for a classification model evaluation.....	16
Table 2. Different models proposed for detecting and classifying pulmonary nodules in the lung using CT scans [3].	32
Table 3. The evaluation results of each model and the ensemble networks using built in python statistical metrics in sklearn (python library for scientific computing).	51

Acknowledgement

My sincere gratitude goes to almighty God for giving me the grace to be healthy and alive to complete this master's thesis. The journey started two years ago and being able to push through to the very end cannot be outside the grace I received from the Lord.

My deepened appreciation also goes to my supervisor in person of Professor Olev Martens for his unconditional continuous guidance, support and advice throughout the phase of this thesis. In which without his supervision and support, this thesis wouldn't have been possible. He is someone I have always seen worth to be emulated in career and personality.

To my big brother and his wife, Mr. and Mrs. Olutosin, I am grateful to you both for your prayers and encouraging words. Those words fueled me with strength and the courage to move on whenever I stumble upon any difficult.

To the duo, Olutosin Tobiloba and Olutosin Michael, I would like to use this medium to say a big thank you to both of you for being a source of encouragement and what has been keeping me going.

Finally, I cannot end this acknowledgment without my mother, Mrs. Olutosin Victoria Adenike, words cannot express how grateful, blessed and fortunate I am to have you as a mother, father and confidant. You are truly a symbol of a true mother with a heart of a warrior. I am using this medium to acknowledge your impact and most especially in the completion of this phase of my life, and I sincerely and whole-heartedly dedicate this thesis to you.

1 Introduction

WHO (World Health Organization) reported that about 9.6 million people died as a result of cancer in 2018 and 18% of these total death cases (i.e. 1.76 million deaths) were accounted to be caused by lung cancer [1]. The lung is the organ in the human thoracic region that is responsible for human respiration. It plays a vital role in human existence and having unusual growth around the lung could pose a great threat to life thus leading to death.

The chance of survival of cancers is usually less than 20% generally within five years after diagnosis, because over 60% of patients were diagnosed with cancer at the latter stage where no possible treatment could be possibly given [2]. According to studies, about 50% of people diagnosed with early stage cancer can live for about five years [3]. This led to the research and development of Artificial Intelligence (AI) based Computer Aided Detection and Classification (CADeC) software as a decision support tool for radiologists and medical professionals to give precise and accurate CT scan results with a high confidence.

The major drawback of this CADeC software is the false positive rate (FPR) resulted from the error learnt by the model during training or features that the model could not learn thereby reducing the overall accuracy of the software application. The FPR is the likelihood of the model to give a wrong result which is one of the metrics that characterizes the precision and accuracy of the application.

This is a big bottleneck in the use CADeC software for medical imaging analysis which creates the reason why it has not been fully and widely adopted for lung cancer diagnosis [4], and thus a need to develop a robust approach that can lower the rate of false positive in the application is inevitable.

1.1 Background

Deep neural networks for computer vision/image processing have shown tremendous breakthrough in several domains especially in medical image analysis. Deep neural networks are biological-neurons inspired mathematical and statistical algorithms developed to solve non-linear complex problems [5]. These networks are a sub-set of machine learning but proved to perform better than classical machine learning due to its deep layered architecture/topology that allows it to be used for automatic extraction of salient features from unstructured and high dimensional dataset [6]. It has been established that deep neural network is big data driven, computationally intensive and there is no standardized DNNs that has proven to solve all problems [7]. It is shown by several studies that models that are trained with high quality and varieties of data will have higher accuracy and outperforms than a model trained will less amount of data due to its wider scope of learned parameters/features during training [8].

DNN is a topology based technology and there are over 20 billion DNN/CNN architectures [7]. As stated, there is no such thing as a single DNN model that can solve all problems efficiently due to its dependency on structure of the layers and several hyperparameters. A few categories of these networks are CNN (Convolutional Neural Networks), DBN (Deep Belief Network), RNN (Recurrent Neural Networks), DBN (Deep Belied Network), SA (Stacked Autoencoder), DRL (Deep Reinforcement Learning), CapsNet (Capsule Network) etc. however, some of these networks are generic in nature (i.e. RNN, SA, DRL etc.) and can be combined with other networks to solve a problem while some are specific to image processing application such as CNN. In general, the overall architecture of neural networks can be broken into different layers: input, hidden and output layers. Each layer interacts with the neurons/nodes that are connected to it and the significance of each node is dependent on the weights that are activated using mathematical functions such as ReLU, sigmoid or softmax.

As stated, some neural networks are generic while some are specific in terms of choice of problem solving. CNN is a category of ANN developed to solve strictly image-based problem. An image can be in 2D (two-dimensional) (i.e. the height and width pixels) or 3D three-dimensional (i.e. height, width and depth). The choice of neural architecture is dependent on the dimension of the image to processed. Computed tomography scans are multi-layer 2D scans, thus require 3D image processing approach. However, individual slice can be treated and processed separately as 2D

array, but a robust, less error-prone and efficient approach is to process all multi-layer 2D(s) as a 3D array. Image processing application using deep learning involves developing a unique architecture or using an already existing architecture and fine-tuning the hyperparameters to suit the specific problem. However, it has been observed that some image processing problems especially medical image analysis are more complex such that a single multi-layer neural network will not give a suitable level of accuracy due to high false positive rate.

This master's thesis is to develop a robust approach using multi-neural networks and softmax averaging to solve this problem of false positive rate, and thus show how this approach will increase the overall accuracy of the model to a level that it can be utilized in real-time lung cancer diagnostic application system.

1.2 Problem Statement

Analyzing CT scan using an automated CADe system for automatic lung nodule detection and classification is a very complex process due to the highly likelihood of false positive rate impact. This has hindered its full adoption in lung cancer diagnosis application software eco-system. This major challenge needs to be solved to increase its wide adoption and most importantly an early accurate diagnosis can further improve the chance of survival of the patient to about 50% [3].

False positive rate being the likelihood of the CADeC application software to give a wrong prediction. It is dependent on a lot of factors such as the quality of the CT scans, over/under filtering of the region of interest of the lung nodule during image pre/post-processing stage, the type of neural model used in developing the application, improper tuned architecture and hyperparameters. Table. 1 showed the four base metrics that are considered in the evaluation of a classification model.

Table 1. Confusion matrix showing the four base parameters for a classification model evaluation.

		Predicted class	
		<i>P</i>	<i>N</i>
Actual Class	<i>P</i>	True Positives (TP)	False Negatives (FN)
	<i>N</i>	False Positives (FP)	True Negatives (TN)

False Positive Rate is ratio of False Positive to the sum of False Positive and True Negatives. This can be written as:

$$FPR = \frac{FP}{FP + TN} \quad 1.1$$

False Positives are voxel labels predicted to be nodules but were not nodules. From the above equation, the higher FP, the higher the FPR and the goal is to have a model with low FPR as possible. However, the False Negative Rate needs to be carefully considered especially in medical application as defined as:

$$FNR = \frac{FN}{FN + TP} \quad 1.2$$

And achieving an even distribution/balance between FPR and FNR is very challenging in deep neural network applications.

1.3 Project Objective

The goal of this master's thesis is to demonstrate the effectiveness and efficiency of using heterogenous 3D multi-neural networks in the reduction of false positive rate in a CADeC application for an automatic lung nodule diagnosis. Multi-neural networks refer to the deep neural network version of a standard ensemble techniques employed that combines several machine

learning models to solve a single problem set. To achieve this, 3D multi-neural networks and a frontend application (i.e. inference engine) would be developed. The frontend application will be stacked on the top of the extracted weights from the ensemble 3D CNN models.

The predicted results (i.e. the voxel nodule or voxel non-nodule) would be extracted from the sigmoid layers of the heterogenous models and the combined outcomes extracted from the sigmoid activations would be averaged. This averaged result would be compared with the threshold level set and the final predicted value will be returned by the frontend application to the user space. The final predicted outcome generated by the ensemble multi-neural networks would also be compared with the predicted probability generated by each 3D model.

1.4 Project Requirements and Specifications

The requirement of this project was defined as:

- The application must be able to detect and classify lung voxel nodule candidate given a raw 3D CT scan in either dicom or mhd/raw (metainage) image files.

The specifications for the project was defined as:

- The application frontend must be interactive such that user can specify the specific image format to be analyzed (i.e. dicom or mhd/raw image files).
- The software must have a higher reliability and low FPR.

1.5 Motivation and Research Contribution

This thesis is research and application development focused as it involves carrying out research on 3D CNN models that can be used in the analysis of the 3D CT scans for lung nodule detection and classification and how these different models can be combined as a single application using serialization to be used in a real-life medical application system. This will be a source of motivation for field medical professionals in bioinformatics and academic researchers to use the method

proposed to solve different complex problems not just in medical imaging area but other domains where neural networks models are applicable.

This master's thesis focused mainly on FPR reduction using 3D multi neural networks approach. FPR is one of the major metrics used in evaluating deep learning model based on classification. The heterogeneous 3D CNN multi neural networks were implemented using python as the programming language, spyder IDE and keras framework which is a wrapper to speed-up the overall development time. The architecture and hyperparameters of each network differ and the result of individual model will be evaluated and compared as shown in the later chapter.

Different works have carried out on lung nodule detection and classification using different methods such as thresholding and morphological operation for features extraction fed to three-layers feedforward networks by Gupta et al [9], Manikandan et al [10], Shaukat et al [11] and Naqi et al [12] obtained a good result by combining manual features extraction technique and SVM based classifier. Qi et al proposed a 3D CNN based classifier for lung nodule detection using 3D voxel [13]. Li et al proposed a new framework to reduce false positive using 3 hetero-shape CNN having different input shapes and then combined to form an ensemble network [14].

As presented by Valente et al [15], a concise and comprehensive review on CADe was published in 2018. This review covered the current and modern methods used in CT based CADe system for automatic lung nodule detection, database used, and the experimental results obtained. It was established from the review that no author proposed/implemented a hybrid 3D multi-neural networks ensemble technique for the reduction of false positive rate in a CADe application software.

1.6 Chapter Review

This chapter shows how this thesis is being structured. Each chapter is buttressed as follows:

The first chapter of this thesis covers the introduction, problem statement and research motivation. The chapter 2 further progresses with the overview of DNN and its applications in medical imaging analysis. This part demystifies DNN, building blocks of DNN, how these building blocks fit together, the enabling factors of DNN and other use cases of DNN were also discussed. Medical imaging techniques, its acquisition techniques, different modalities, the advantages & disadvantages of each modality and why the modality used in this thesis was chosen. The later part of this chapter gives an introductory overview of the category of DNN employed in this thesis.

The chapter 3 describes in detail the state of art techniques that have been proposed for CADe and CADx software application systems. Both DNN and non-DNN methods were referenced. The later part of this chapter covers the topologies of DNN, the dimension of the raw CT scan and how this dimension determines the kind of neural network and topology to be employed.

In chapter 4, the development of the heterogenous 3D CNN multi neural networks and frontend application were described in detail. The methods and implementation pipelines developed were explained at length. Also, the ensemble techniques used in developing the different 3D CNN models was discussed. The 3D neural architectures, hyperparameters, input and output shape formats of each model were described in detail. This chapter also covers the description of dataset used, choice of IDE, frameworks and programming language used were discussed in detail.

The chapter 5 of this thesis covers the evaluation and results comparison of the models developed using random test data. The results of all models were analyzed based on these metrics (FPR, precision, recall/sensitivity, accuracy and specificity) before and after sigmoid averaging to show how the technique improves the overall false positive rate and the confidence level. The conclusion was drawn based on the results from the previous chapter (i.e. chapter 4). Also, the challenges faced were buttressed, how these challenges can be mitigated were discussed, the future work and possible optimization were detailed.

2 Overview of Deep Neural Networks

This chapter covers in the overview of DNN, its motivation, different architectures, the building blocks, function of each block/node and how all blocks fit together to form the entire network. A detailed description of CNN was discussed, why it is used in medical image analysis and how the architecture evolves were also described.

2.1 DNN Introduction

Deep neural networks are mathematical and statistical models conceptualized as networks of interconnected nodes and layers as shown in Figure 1 [16]. These algorithms are bio-inspired by how human brain functions. DNN is a machine learning technique that allows computers to learning complex mapping between the input and output layers, then using these mappings learnt on new set data to make new predictions.

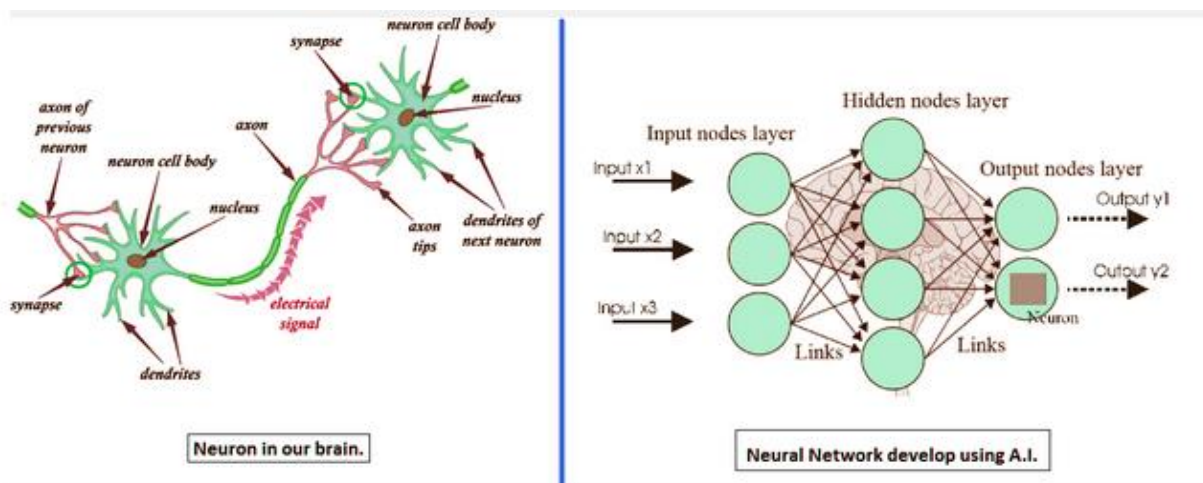


Figure 1. Structure of Human Neuron and Artificial Neuron [16].

DNN has three high level layers categorized as input, hidden and output. Each layer is made up of nodes that are interconnected with other nodes in other layers. This overall concept is what is

referred to as a network (i.e. interlinking of different nodes from the input layer to the output layer and vice versa). As mentioned earlier, there are variant topologies of DNN, however, regardless of these variations, there are components that are common to all architectures.

2.1.1 Building Blocks of DNN

The fundamental building blocks of DNN nodes or neurons. These neurons formed the input layer, hidden layer, output layer of the DNN. Other building blocks are links/weights, activation functions and biases. Some DNNs may not possess hidden layer (i.e. just an input layer and an output layer) while some can have a single hidden layer (i.e. perceptron) [17] or multi-layer (i.e. multilayer perceptron) [18], however, each neural network component has specific function in the network regardless of the number of hidden layers as shown in Figure 2 [19].

The input layer is the receptive layer that consists of nodes that accept the input data. Each input node also called neuron represents each observation point of the input data. The number nodes present at the input layer is a function of the number of observation points in the input data.

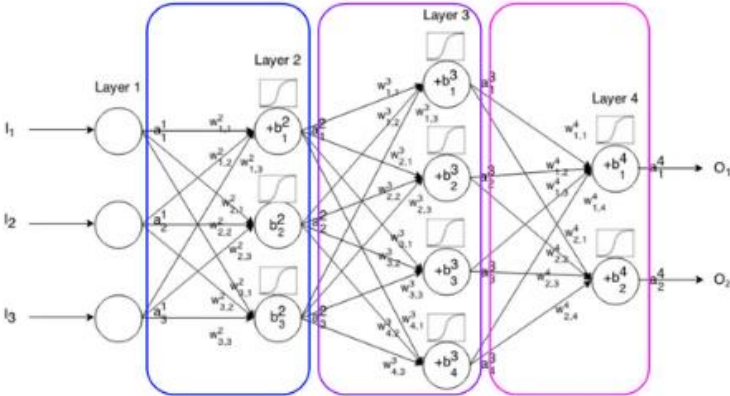


Figure 2. A multilayer perceptron fully connected neural networks [19].

The hidden layer also called the computation layer has its own nodes and each node processes the data it receives from the input layer nodes. Other neurons in the hidden layers in a multilayer perceptron can process data from preceding hidden layers as shown in Figure 2 (this has two hidden layers that are fully connected, layer one being the input layer having three neurons/nodes, the first hidden layer has 3 nodes each having a sigmoid activation function. However, some other

activation functions such as ReLU, Tanh, Leaky ReLU can also be used. The second hidden layer has four neurons and each neuron processes the data from the hidden layer two and the final output layer having two neurons each having its activation function).

2.1.2 Network Propagation

Propagation of neural networks defines how the network maps data between the input and the output layer during training. The most popular algorithm used is called stochastic gradient descent whose aim is to reduce the loss to the barest minimum by finding the global minima (loss) as shown in Fig. 20. DNN has three major phases: training, validation and inference phases. The efficiency of the model is dependent on the first two phases thereby effort needs to be placed in these phases in order to achieve optimal accuracy. There are two propagation methods used by DNN during the training phase to learn and understand the complex mappings between the raw input datapoints and the corresponding output labels, these methods are forward propagation and backward propagation which is responsible for weights update as shown in Figure 3 [19]. However, there are other categories such as unsupervised and semi-supervised networks [20].

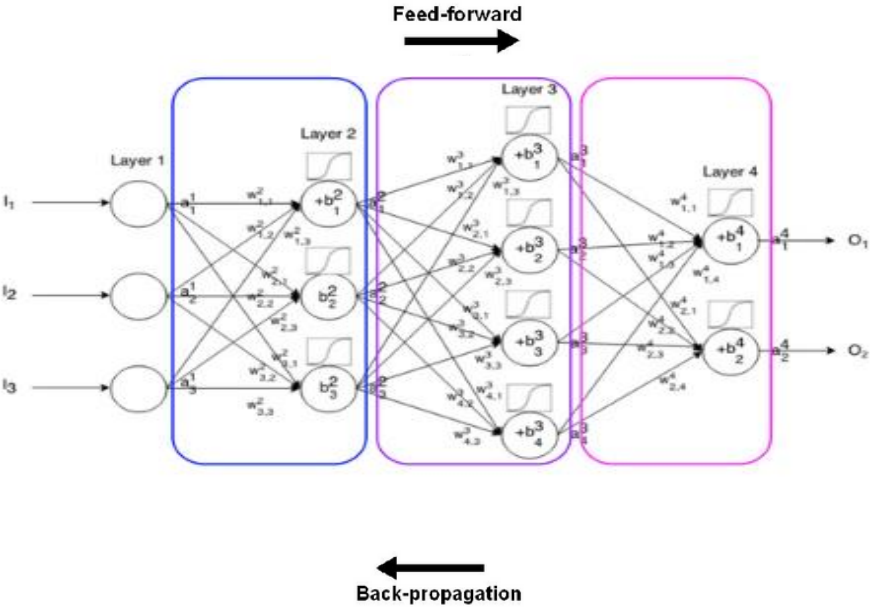


Figure 3. Forward and backward propagation in neural network [19].

Forward propagation in neural network starts with a random initialization of the weights of the input neurons. The weights indicate the strength of the connecting link between the nodes in the network and determines the nature of the output. A node with zero weight means the output will not be affected regardless of the value of that that node and vice versa. These weights and the corresponding input neurons are processed using an activation function at each node of the proceeding layer which generates an output [19]. The nature of the output produced is dependent on the type of activation function used. The final output is computed at the latter nodes of the output layer to generate a predicted output. The difference between this predicted output and actual output is calculated and this is called the loss function.

Backward propagation uses this loss function to update the weights accordingly and the objective to obtain a loss of zero (0) in an ideal network [19]. However, there is no such thing as ideal network because there are different challenges related to obtaining the global minimum (i.e. state of having zero or near zero loss value) such as exploding gradient [21], vanishing gradient [22] and difficulty in identifying the global minimum value due several local minima. To further improve how fast the network converges i.e. obtaining a lower loss value, the hidden layers can be biased. These biases are weights added to the hidden layer in other to shift the computed/learned function such that it fits on the original function as shown in Figure. 4 [23].

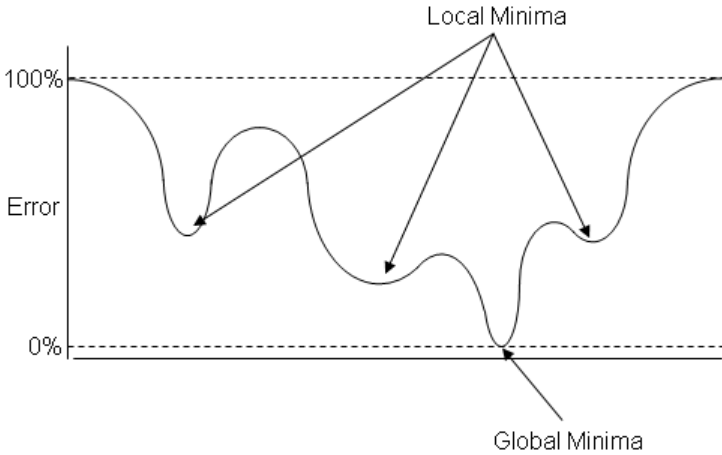


Figure 4. Global and local minima visualization during backward propagation [23].

Obtaining the optimal network model that gives a higher accuracy is an iterative process that requires several parameters tuning in the validation stage. This makes make the whole process

computationally intensive and this is one of the cons of neural networks. Nonetheless, this is achievable due to availability of powerful computers having parallel GPUs. The final output function generated is tested and validated using test/validated data in order to test how well the model has learnt and how efficiently it will perform on new data.

2.2 Building Blocks of Convolutional Neural Networks

As mentioned in the sub-chapter (building blocks of DNN), there are variants of DNN and one of these variants is Convolutional Neural Network (CNN). CNN was developed by Yann Lecun in 1989 to solve image classification problem. It combines convolutional operation via the convolution layer and building blocks of typical DNN or ANN [24]. Ever since a lot of improvements have been made to this simple architecture (i.e. LeNet) to a very complex one (such as UNet) which is used to perform semantic image segmentation problem [24].

Medical images come in different modalities as shown in Figure 5 and due to its complexity, studies and results obtained showed that CNN is a great approach to adopt in CADe application software [25].



Figure 5. Different modalities in medical imaging [25].

CNN has shown tremendous potential in medical image domain. However, there is no de-facto standard about which network architecture suits best, but some results have shown that some architectures perform better than the other in solving a problem. CNN could be 1D (one

dimensional), 2D (two dimensional) or 3D three dimensional and choice of what dimension to use is dependent on the dimension of the input data.

Medical images regardless of the modalities have standardized image dimension format which is (512 by 512) and varying depth called slice as shown in Figure 6.

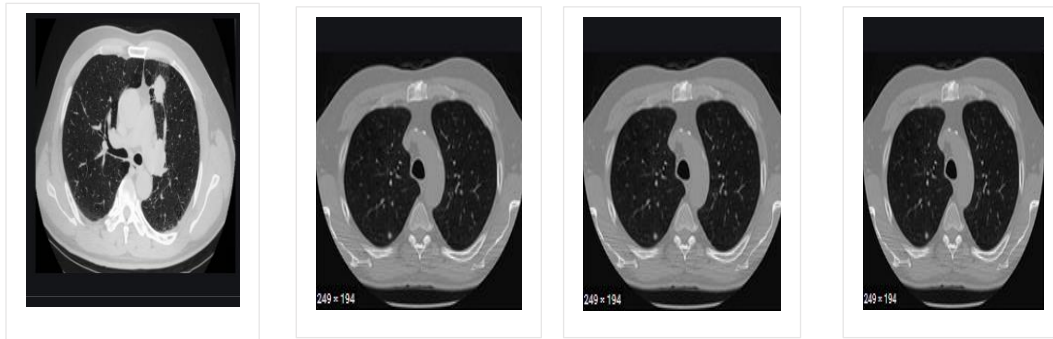


Figure 6. (512 x 512) computed tomography scans of the lung [26].

Medical image scans are standardized, and the dimensions of the images produced are of same width and height, but the total thickness/depth would vary due to the variation of in sizes of patients' body structures. In 2D CNN, each slice will be treated as 2D image and the appropriate filter size will be designed. However, 3D CNN accepts 3D images as input which is the default dimension of computed tomography scan.

2.2.1 2D Convolutional Neural Networks

Convolution in neural networks is the same convolution operation introduced in performing mathematical operation of two signals or functions to create another signal or function. In medical imaging analysis, images obtained are usually large and every pixel in the image matters. 2D CNN employs a (d by d) filter that slides through the entire image of dimension (height, width, channels) from left to right in a top to bottom approach until the last edge of image is reached as shown in Figure 7 [26]. This improves generalization and thus reduces the size of pixels that will be processed by DNN. The choice of filter size or kernel size is not fixed and choosing the right one is dependent on the comparison of previous results obtained.

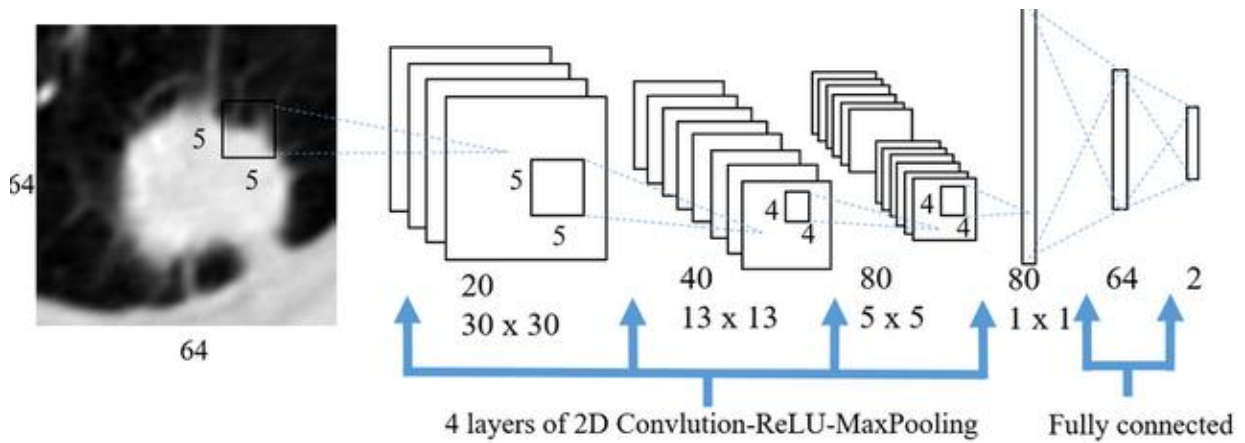


Figure 7. 2D CNN on (64x64) CT scan using (5x5) kernel filter [26].

A typical simple 2D CNN architecture usually have a single or two convolution layers and can be more as shown in Fig. 7. This architecture has one convolutional layer activated by ReLU activation function, 1 maximum pooling layer, a flattened layer, two fully connected layers and an output layer. The convolving filter used is a (5x5) pixels filter and twenty of these filters were used to generate more information from the image that are fed to the proceeding layers of the network.

2.2.2 3D Convolutional Neural Networks

A 3D CNN has an input dimension of (depth, height, width, channels) and the size of filter of (d x d x d) as shown in Figure 8. The default dimension of computed tomography scan is 3D and can be fed directly as input for the convolution layer.

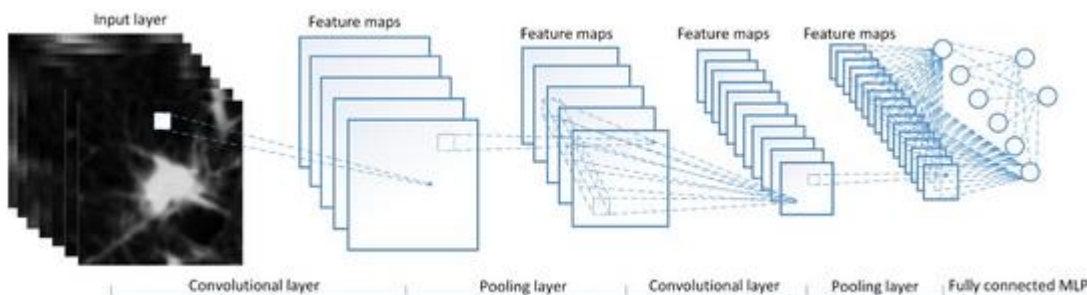


Figure 8. 3D CNN on (64x64) CT scan using (5x5x5) kernel filter [26].

The architecture of the network shown in Figure 8 is a simple architecture having two convolutional layers, two pooling layers, a single fully connected layer and two nodes as the output layer. Also, studies have shown that 3D CNN outperforms 2D CNN with a tradeoff of computing resources [27].

2.3 Deep 3D Multi-Neural Networks

The reduction of low false positive rate in deep neural network models has been a major challenge for lung nodule detection and classification in medical image applications in general according to the review carried out by Zhang et. al. [4]. FPR can be reduced by transfer learning, regularization and ensemble technique. However, care needs to be exercised due to how complicated medical images are and the choice needs to be carefully examined.

Transfer learning involves using an existing model to extract salient features from the input images and using the features extracted to develop a model. This method has shown some improvements but adopting such method in medical image analysis comes with its disadvantages. Regularization is an image processing technique that involves manipulating the image such is flipping, zooming, shearing etc. this increases the generalization of the model which in turn improves the model performance slightly. Ensemble or multi-neural networks is not a common approach applied in medical image applications and exploring this approach is one of the factors that contributes to the novelty of this thesis work. 3D-Multi-neural networks involve developing different 3D neural models for the same problem set and averaging the results of all models.

The mean of these results is computed, and it is returned as the final predicted value. G. Huang et al uses snapshots of model at different iterations from one training process [28]. There are several 3D neural networks models (3D CNN, 3D CAE and 3D DBN) that can be employed to solve 3D image based problems which can be adopted in lung nodule detection and classification. Has described in previous section, 3D CNN combined convolutional operation in a higher dimensional space with DNN.

Autoencoder tries to reproduce the input it is fed with and learn the mapping in the process as shown in Figure 9. It uses two functions an encoder and decoder, the encoder converts the input data into internal representation and the decoder tries to reconstruct the encoded result to produce the input data.

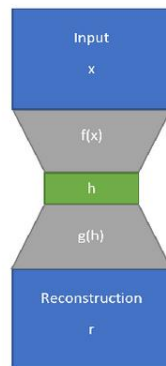


Figure 9. A simple Autoencoder network.

There are variations of autoencoders applied in image processing applications such as undercomplete, denoising and convolutional autoencoders. Undercomplete autoencoder applies a dimensionality restrictive rule such that the dimension of the output produced by the encoder is less than the input. Denoising involves random noise introduction to corrupt certain region of the input image preceding the encoder. Convolutional autoencoder applies kernel filters on the input image as shown in Figure 10 followed by down-sampling, up-sampling and de-convolution to reconstruct the input image [29].

Each green block consists of sub-network with group normalization and the reconstructed image of three channels produced by the decoder.

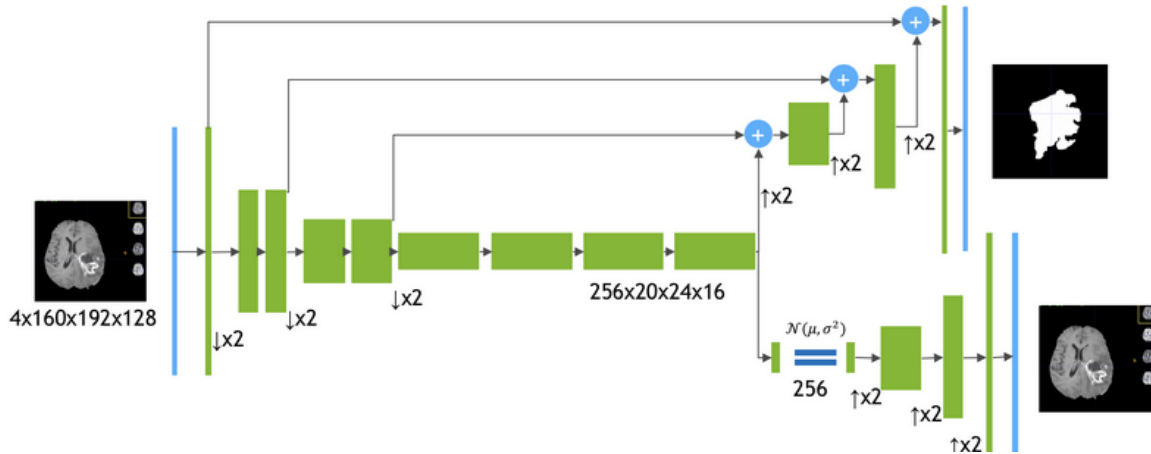


Figure 10. A 3D Convolutional Autoencoder [29].

DBNs are neural networks that are generative in nature that produce set of possible outcomes based on the input fed to it. DBNs are made up of networks of unsupervised RBMs, the hidden layers lack inter-connections making the layers independent such that hidden layer of the first RBM becomes the visible layer of the next stack of proceeding RBM stack as shown in Figure 11 [30].

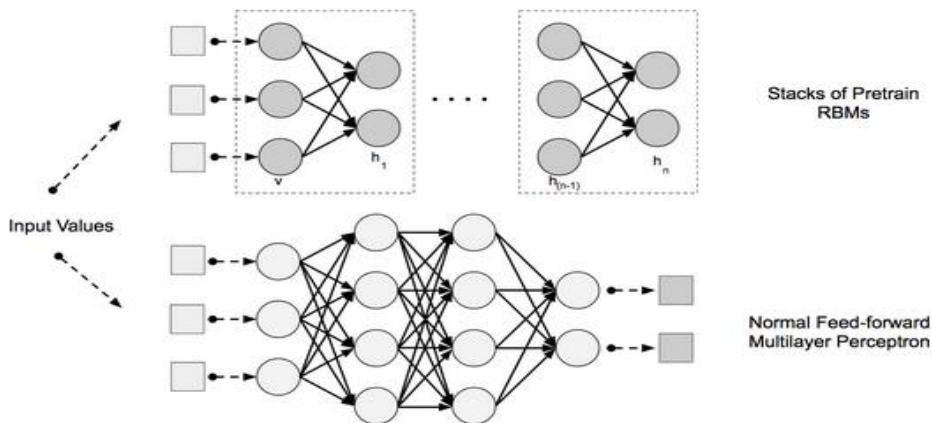


Figure 11. A 3D DBN (Deep Belief Networks) [30].

3 State of the Art and Related Works

DNN or ANN has been around for over two decades , but it was not adopted early because it requires high computing power and large dataset which were not available at the time of birth of ANN. Due to rapid rise in technological innovation by researchers and professionals in semiconductor industries allowing the manufacturing of chips powering highly efficient computing machines gave ANN its space again and has proven beyond measure with massive adoption in different domains such as robotics, automobile and medical applications.

A lot of methods have been introduced to solve nodule detection and classification and the results would have been biased but was mitigated due to the availability of public lung images dataset which can be used as benchmark for testing different algorithms or models. Lung Image Database Consortium LIDC-IDRI [31], JSRT [32], LUNA16 [33], and LNDb [34].

3.1 Lung Nodule Detection and Classification Phases

3.1.1 Lung Image Acquisition

Lung cancer has attracted the attentions of medical professionals in the field of radiology and medicine due to its cause of high mortality rate worldwide. Different bodies in different countries have since developed a means of collecting thorax images by partnering with different clinical institutions for fostering research programs for finding effective and novel methods for combating lung cancer.

LIDC-IDRI has a database containing 1018 chest CT scans annotated into three classes based on nodule size by four different radiologists to reduce biasedness that could be introduced through human error while JSRT has 247 chest images divided into 93 images without nodule and 154 images with nodules. Also, Centro Hospitalar e Universitário de São João (CHUSJ) in

Porto, Portugal has total of 294 scans annotated by 3 certified radiologists with years of experience following LIDC guideline. All are publicly available for model development and test.

3.1.2 Lung Image Preprocessing

Lung image preprocessing involves filtering out parts of the image that are not relevant and reconstructing missing regions. This does not only improve the accuracy of the model which allows the ANN model to learn useful parts of the image but also reduces the size of the network thus increasing how fast the model learns. Q. Wang et al proposed active appearance approach in their paper [35] and Teramoto, H. Fujita et al. proposed enhancement filter for nodule enhancement [36].

H.O. Madero et al used hough transform which is an image processing technique which is used to find imperfect instances of object within class of shapes by voting procedure [36], Lu et al was based on discrete wavelet transform and unsharp energy mask [37] and B. Chen et al applied a non-linear median filter to eliminate the noise from the image [38]. However, there are numerous states of the art techniques that can be used in digital image processing other than the ones proposed such as contrast stretching proposed by M. Javaid et al and T. Messay [39][40].

3.1.3 Lung Feature Extraction

CT scans show the image of the two lungs, heart, chest bones, tissues etc. However, this is quite tricky due to the presence of organs and tissues in human chest. Segmenting the region of interest is very important for accurate and precise detection of lung nodule. Naqi et al applied morphological operation and thresholding [41]. B. Zhao et al used thresholding method for separating the lungs from other organs and tissue [42], B Li. et al used edge detection technique [43] and F. Shaukat, C. Seiffert and I. Ali et al applied filters based on morphological operations [44][45][46].

3.1.4 Post Processing and Classification

The evaluation of model can be done using different techniques and the underlying reason is to reduce the error rate for better accuracy and precision. High accuracy and precision are highly demanded in medical application because this is specific to human lives and while trying to save life, we should

Table 2. Different models proposed for detecting and classifying pulmonary nodules in the lung using CT scans [3].

Name of Author (s)	Algorithms used for nodule detection	Algorithms used for classification
Wang et al. [50]	Central Focused CNN	Chan Vese Model
Teramoto et al. [36]	-	CNN
Madero et al. [36]	Discrete wavelet transforms	Support vector classifier
Lu et al. [37]	Fussy entropy and variation level set method	Regression tree
Javaid et al. [39]	K-means clustering and morphological filter	Support vector classifier, rule-based method
Wang et al. [51]	Thresholding and region growing	Adaptive fuzzy C-means methods and Mahalanobis
Liu et al. [52]	Spatial fuzzy C-means (SFCM)	-
Zhang et al. [53]	Thresholding and morphological filter	Support vector classifier
Gupta et al. [9]	Three sub-algorithm modules	Three layered feed-forward neural network
Akram et al. [48]	Multiple optimal thresholding	Support vector classifier

Table 2 shows a summary of the proposed state of the art models for detecting and classifying nodules in the lung. G. Zhang et al in the review paper the level of FPR of different model proposed for CADe system [4] as shown in Figure 12.

make it better not worsen it. Generally, model evaluation can be done using confusion matrix which is a matrix that shows the distribution of True Positives (TP), False Positives (FP), True Negatives (TN)

and False Negatives (FN). This evaluation method has proven useful in so many cases that involves classification or predicting the likelihood of a result using learned parameters.

Qi et al proposed a 3D Convolutional Neural Network for the reduction of false positive in nodule detection [47], Gupta et al three sub-modules and candidate augmentation was employed to reduce false positive rate [9], S. Akram et al developed an approach based on statistical features for intensity values [48], Dhara et al used SVM for classifying nodules as benign or malignant [49].

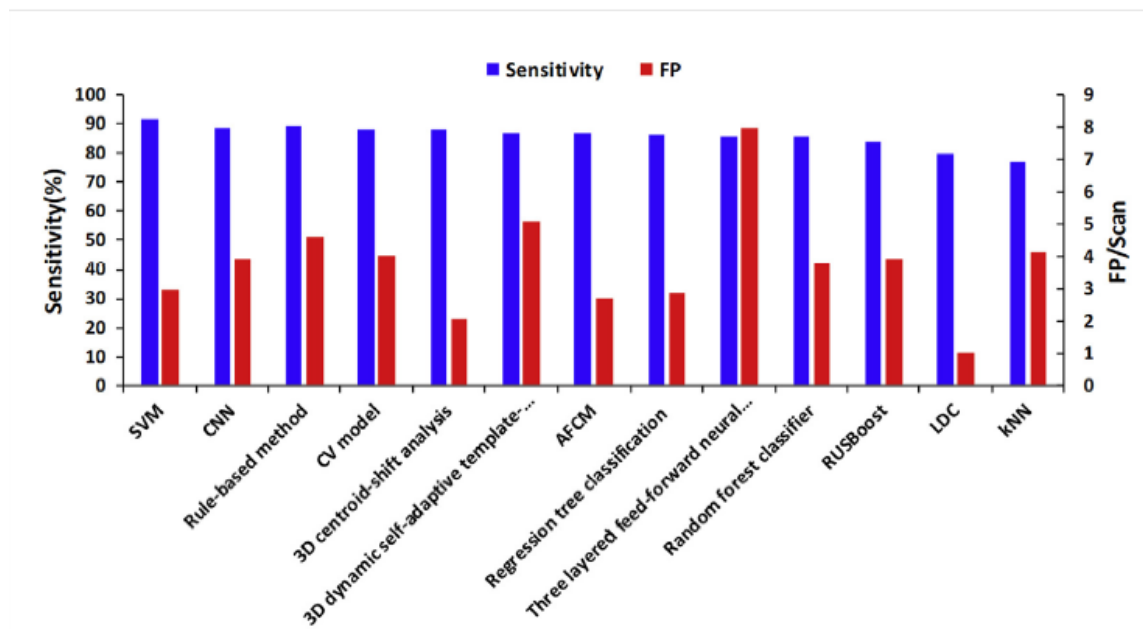


Figure 12. Sensitivity and FPR of some of the different models proposed in different literature studies [3].

4 Methodology and Project Development

Developing an end-to-end inference/analytical application software engine powered by deep neural networks is a very complex task. A dynamic approach called divide and conquer strategy used in solving dynamic problems in software engineering was adopted. Divide and conquer involves breaking a complex problem into sub-problems and solving each sub-problem one at a time and thereby combining all solutions as one. This helps to reduce the complexity of the problem and thus, makes the overall solution very effective.

The inference engine is an application software that runs on the top of deep neural networks. This software engine detects and classifies new 3D CT scan (both DICOM and MHD/RAW image formats). Using software engineering approach, the implementation pipelines were developed for the heterogenous 3D multi neural networks and the inference engine (application frontend). The pipelines define the multi-neural network flow and the inference engine for the lung nodule detection & classification and both pipelines can be structured into 3 main parts (i.e. input, process and output phases).

4.1 Dataset Description

The dataset used in this thesis was obtained from LNDb database which contained 294 3D CT scans collected from Centro Hospitalar e Universitário de São João (CHUSJ) in Porto, Portugal between 2016 and 2018. The total scans were annotated by 5 radiologists having 4 years of annotation experience. This database was used out of the few sources of databases available publicly due to it's simplistic, concise and straightforward way of presenting the metadata information for all the patients' scans included in the database.

Each radiologist marked their findings using the annotation standard set by LIDC-IDRC as shown below:

- I. Non-nodule: this lesion was marked by the radiologist as non-nodule but has nodule-like attributes
- II. Nodule $\geq 3\text{mm}$: this lesion was considered nodule with in-plane dimension greater or equal to 3mm.
- III. Nodule $< 3\text{mm}$: this lesion was considered nodule with in-plane dimension less than or equal to 3mm.

This dataset has a total of 294 of the anonymized patient files in 3D having dimension of $(512 * 512 * N)$, where N is the depth size of each CT scan which varies from patient to patient due to body size differences. This variation in depth is usually typical of every CT scan.

4.2 Development Environment

Important factors to be considered in the development of deep learning based software application are the choice of development environment, programming language and frameworks. I have decided to use python due its multi-paradigm and has become de-facto for machine learning models development. The deep learning framework of choice was Keras due to its high abstraction APIs and human readability. OpenCV 4.0 and Anaconda integrated with Spyder 4.0 was chosen as the development environment as shown in Figure 13. The version of each software dependencies will be specified in the appendix section of this thesis.

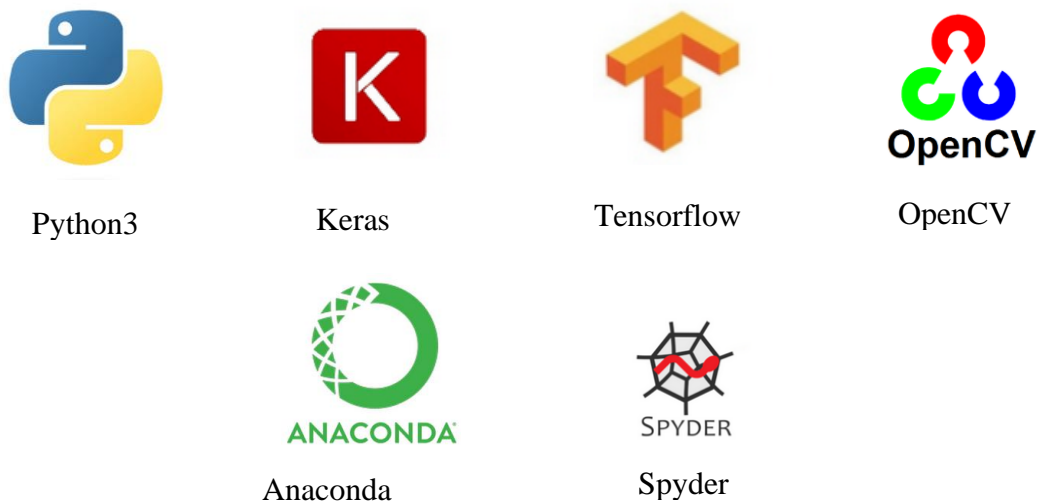


Figure 13. Software packages and frameworks used for development.

4.3 Implementation Pipelines

The pipelines define the implementation flow of the 3D deep neural network models development and the inference engine for the CADeC software for the nodule detection and classification.

4.3.1 3D CNN Multi-Neural Networks Pipeline

The implementation pipeline has 4 stages, the raw CT scan pre-processing stage, models development stage, prediction result extraction stage and averaging as shown in Figure 14.

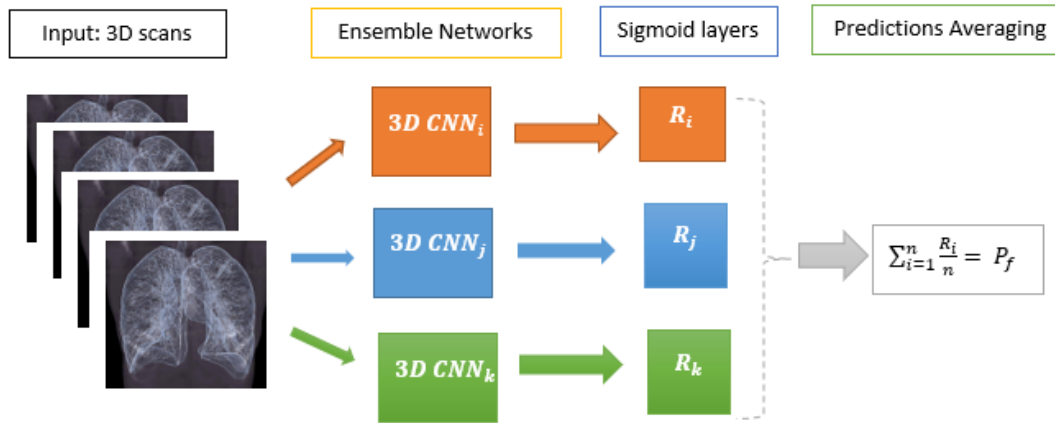


Figure 14. Structure of the implementation pipeline of the Ensemble 3D CNN models.

The files consist of raw images of the lungs encoded with certain information that helps radiologists in the proper analysis during diagnosis session. The pre-processed images were stored in NumPy array that served as a common input for the trained three 3D deep neural network models. The inference engine averages the predicted probability results from serialized 3D models to generate its final output inference.

The pre-processing phase is a very crucial phase in the model development pipeline. The more the quality of the salient features extracted, the better the model in term of accuracy and low error rate.

4.3.2 Image Pre-processing Phase

All deep neural networks have specific data formats that can be processed by the network. This means that the raw data need to be processed into an array of vector before usage. The LNDb dataset are in mhd/raw format which is one the most used file formats for medical imaging (there are nifty and dicom). This file stores the meta-data (i.e. all useful information about the CT-scan) and slices of (512 x 512) single channel gray scale image as shown in Figure 15.

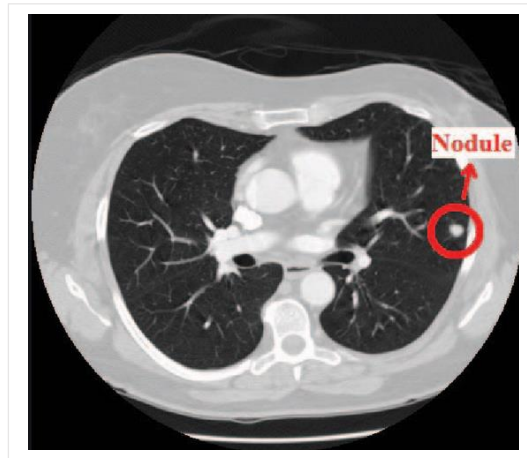


Figure 15. A single 512 x 512 pixels single channel CT- scan of the lung marked as nodule.

The total number of slices varies from one patient to another due to difference in body sizes, this is the major challenge that needs to be solved in the pre-processing phase. Also, 512 by 512 image in best practice is not advisable to be used directly because it will take the model a very long time to train and might not to converge easily due to the size of trainable parameters that would be generated by the network before training. This implies that each slice needs to be resized, and for better understand, a flowchart was developed for the entire processes as shown in Figure 16 and the equivalent python code named **3D-scan-processing.py** included in the appendixes section of this thesis with proper lines of comments for ease of re-production for future works.

The flowchart started with loading of all libraries needed and the creation of the directory and appropriate text file referencing the local paths where all the mhd/raw files are stored. The is a very important part of the pre-processing phase because it ensures the right file for each patient was read appropriately and accurately. Text files and corresponding labels extracted from the dataset were mapped to the path of the raw image files. The images and the corresponding voxel

label (nodule and non-nodule) were converted into array and stored as NumPy array to ease further pre-processing operations carried out.

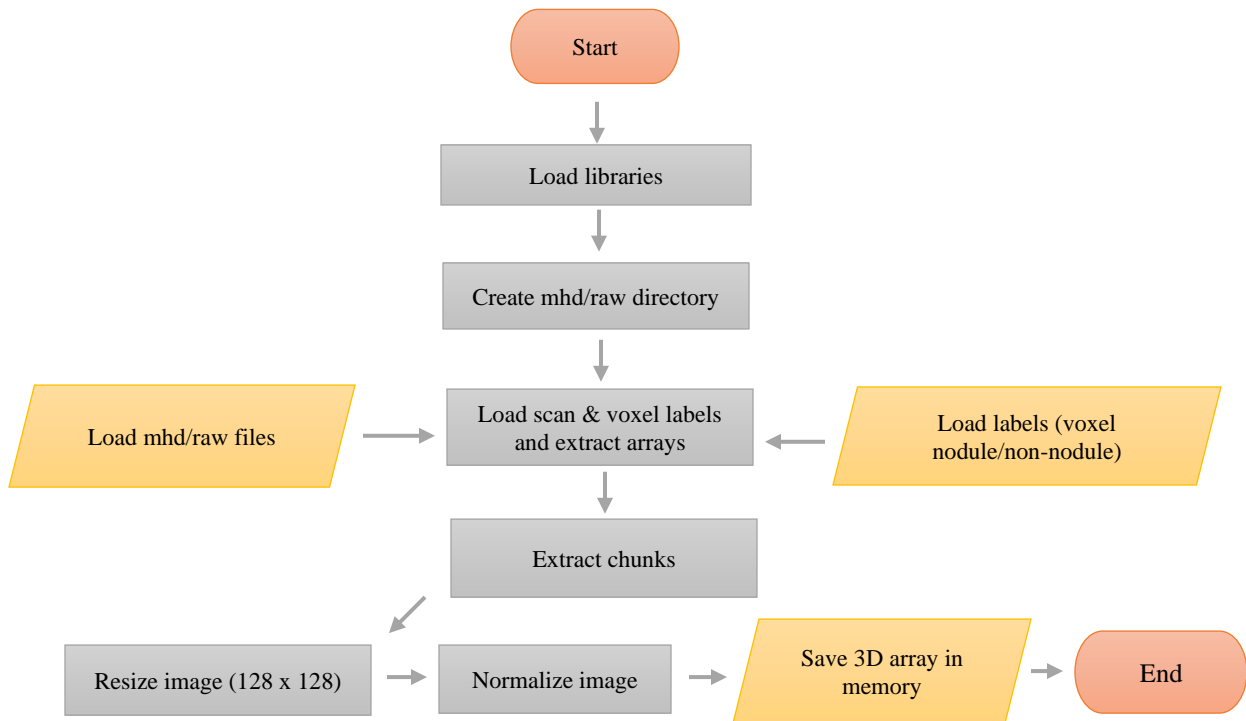


Figure 16. Flowchart of image pre-processing phase to extract the 3D array and dumped in NumPy file.

Chunks of slices (2D arrays) from the total slices present in the image were extracted and each chunk was resized to 128 pixels in width and height. This compression and extraction techniques were carried out to ensure that the array has a fixed depth corresponding to the acceptable neural network file format. Each 2D array which is of size (128 by 128 pixels) corresponds to a slice whose thickness is 2.5mm embedded the image file meta-data.

The 3D array of chunks of the 140 patients and the corresponding nodule/non-nodule voxel labels were converted into NumPy array that was saved on the in memory for further usage by the multi-neural network models developed. This was done to prevent repeating the pre-processing operations all the time. It is also worth stating that running the pre-processing phase alone on my Dell Intel CoreI5 PC took approximately 2 weeks for each round of pre-processing to complete.

4.3.3 Ensemble 3D CNN Models Development

The ensemble strategy used in this project to create the different 3D CNN models was based on data randomization and variational hyperparameters tuning. Data randomization involves generating different splits of the training dataset from the same dataset base. This is usually the case when there is shortage of dataset and this is true of most real-world deep neural networks based application where getting enough dataset to train the model is difficult. Hyperparameters are networks variables that need to be initialized prior to training the model as discussed in previous section.

Ensemble networks of three 3D CNN were developed combining the two ensemble techniques. The first and second models of the 3D CNN ($3D\ CNN_i$ and $3D\ CNN_j$) were based on the same architecture, randomized training data split and varying compilation parameters as shown in Figure 17 and 18. The compilation parameters varied were the rate of decay and number of epochs. This variation allows the model to learn differently and thus, generates complex mappings based on their understanding of the networks. Both generated a total parameter of 12,396,145 parameters, 12,394, 345 were trainable and 1,800 were non-trainable.

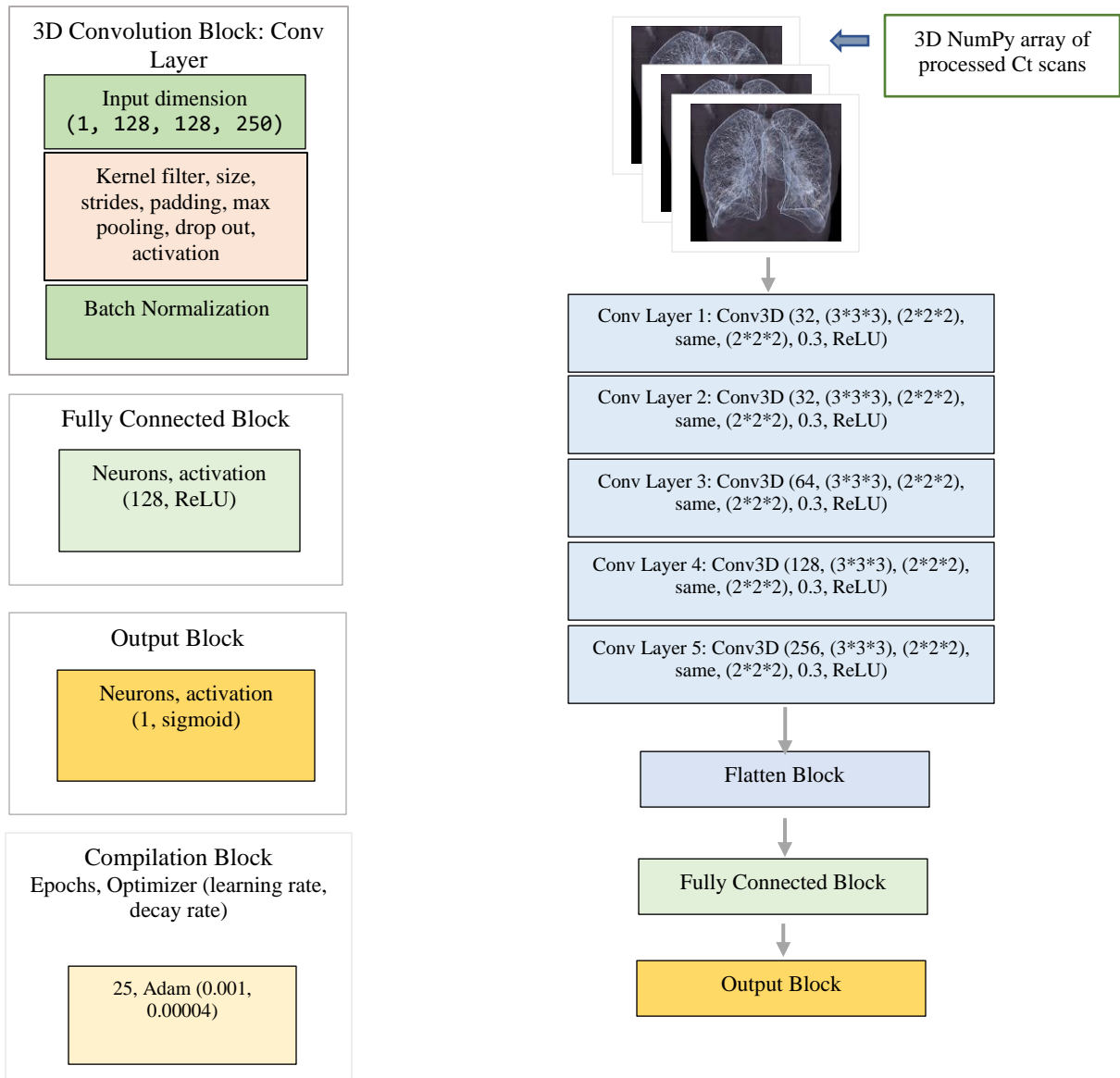


Figure 17. The architecture of 3D CNN_i ensemble model.

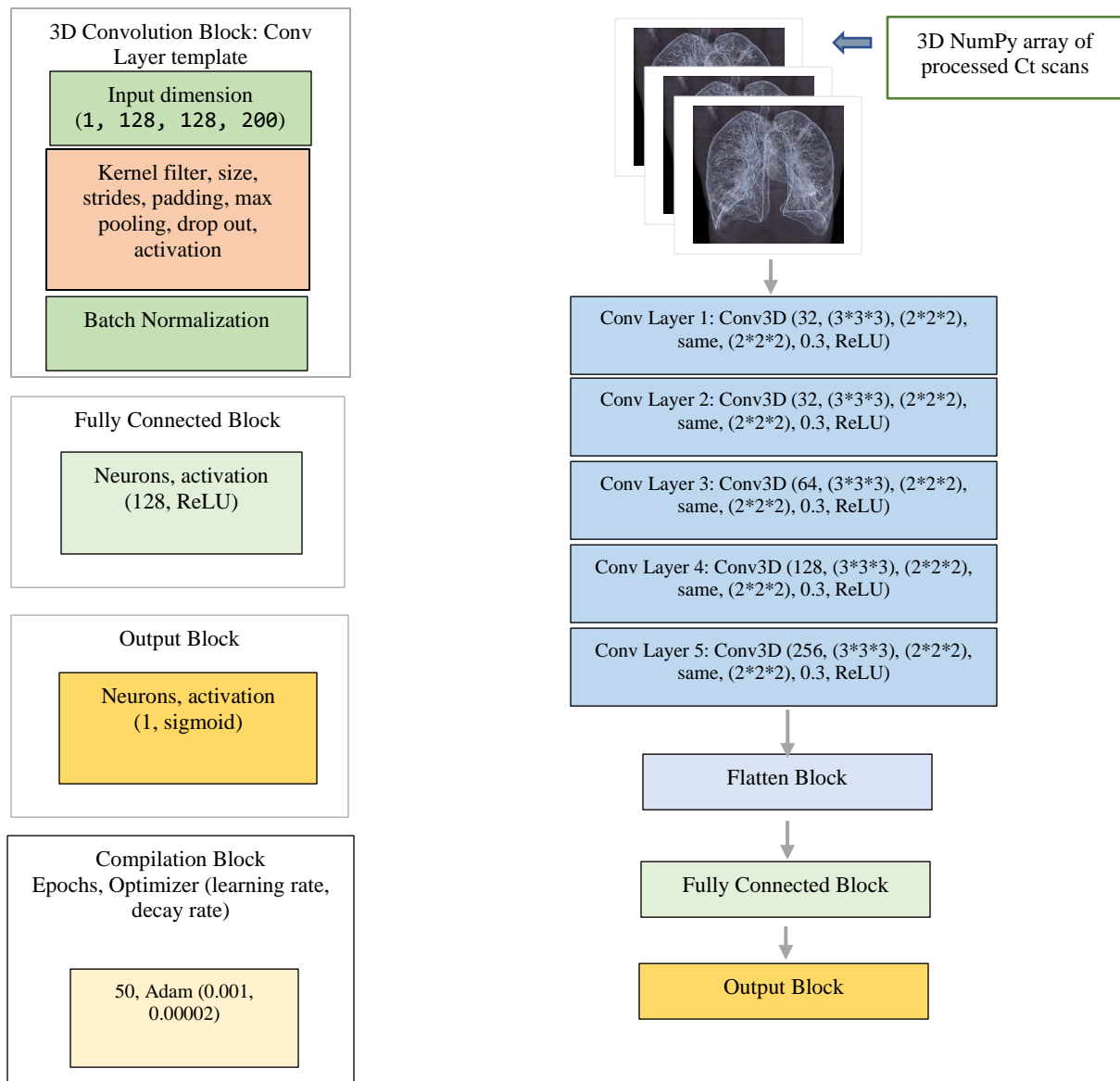


Figure 18. The architecture of the 3D CNN_i ensemble model.

Both networks have 5 convolutional layers, 5 max pooling layers, a flattened layer, a single fully connected layer and an output layer. However, the input dimension of the image uses CHWD format (Channel, Height, Width, Depth) See Figure 17 and 18, the first convolution layer block has 32 kernel filters of size (3,3,3), strides of size (1,1,1), padding as same as the input file, a pool

size of (2, 2, 2) and ReLU activation function. Each filter convolved through the NumPy array fed to it and creates a feature maps in which 3 of the input arrays resulted in a single feature map matrix during a single convolution operation. The first convolutional layer generated a total parameter of 896 as shown in Fig. 17 and 18 for both 3D models. All other convolutional layers followed the same process preceded by the max pooling layer for spatial orientation of the features generated. The strides determine how much the filters would slide through to image horizontally, vertically and depth wise.

The NumPy file was loaded from local memory and fed to the first convolution layer using the accepted input shape format (channels, height, width, depth). Drop-out of 0.3 was used to deactivate the weights of the nodes that are not/less contributing to the overall output of the model which prevents the model from over-fitting. Max pooling layers do not contribute to the number of trainable parameters but played a very important role to improve the generalization of the model. The output of last max pooling layer was transformed into a one dimensional vector which forms the input layer of the DNN.

There are 128 neurons activated by the nonlinear activation function called ReLU that generates an output between 0 and maximum of the input parameters. The output of the fully connected layer was linked to a single node activated by sigmoid function that generates an output between 0 and 1 using a threshold of 0.5. This implies that if the predicted output has a probability less than 0.5, the neuron will activate 0 and if the probability is between 0.5 and 1, the neuron will activate 1.

The dataset was splitted into training and validation data using a test size of 20% of the total dataset size. The training and test dataset were generated at random for each ensemble network. A total of 1,296,145 trainable parameters were generated and the whole network was trained using gradient descent optimizer using adaptive moment and loss function of binary cross-entropy. The model was compiled and validated on validated samples of the images for an epoch (i.e. number of iterations) of 25 & 50 epochs respectively during training (see appendix 2).

The accuracy and loss of the ensemble 3D CNN models generated during the training and validation phase were shown in Figure 19, 20, 21 and 22 respectively.

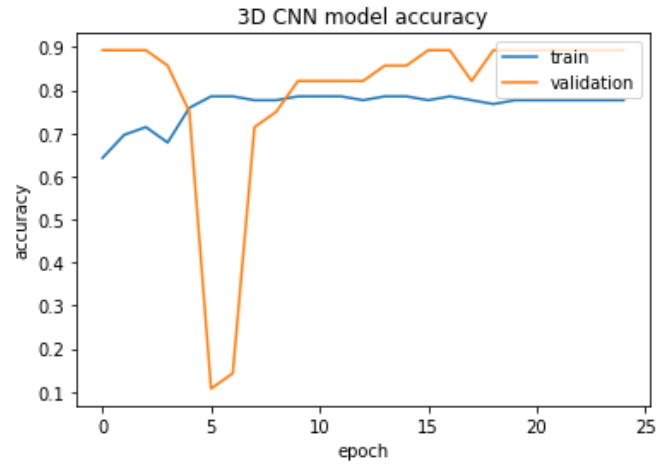


Figure 19. Plot showing the training and validation accuracy of *ensemble3D CNN_i* over 25 epochs.

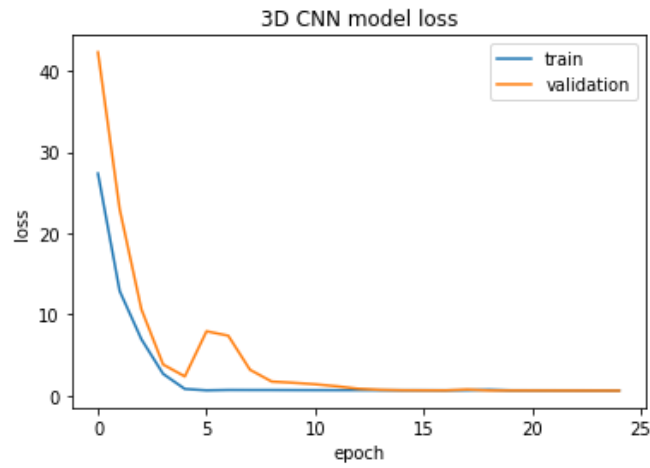


Figure 20. Plot showing the training and validation loss of *ensemble 3D CNN_i* over 25 epochs.

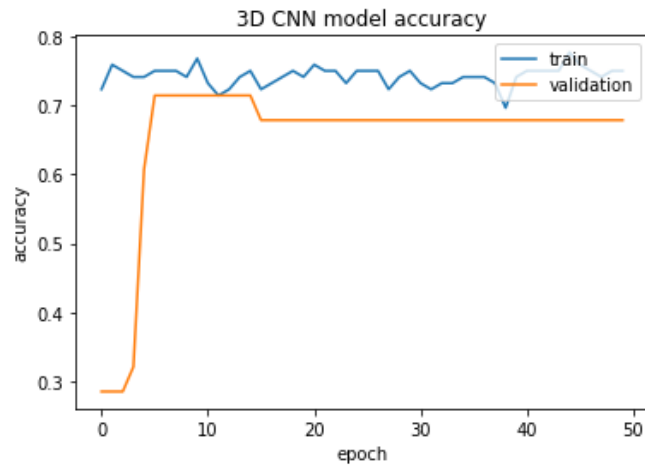


Figure 21. Plot showing the training and validation accuracy of *ensemble 3D CNN_j* over 50 epochs.

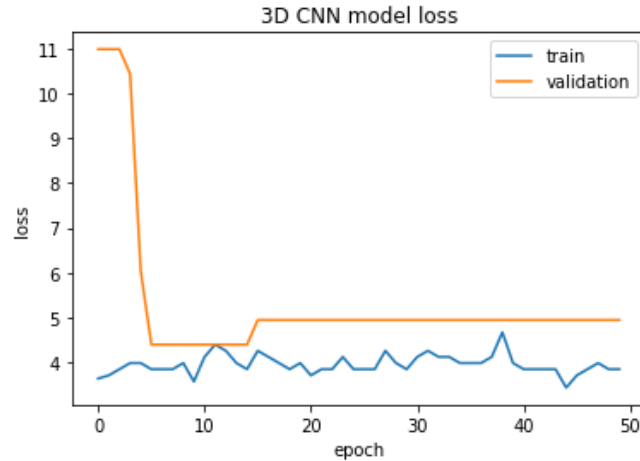


Figure 22. Plot showing the training and validation loss of *ensemble 3D CNN_j* over 50 epochs.

The accuracies of the both networks increased over the epochs while the losses flatten out. Also, the difference between the training and validation accuracies of both models were very close, likewise the losses. This showed that the models did not over-fit as shown in Figure 19, 20, 21 and 22. The validation flatness line seen in Figure 21 and 22 showed that at epochs above 10, the validation score were very close in value and this was as a result of the small amount of the dataset used. However, the accuracy of *model 3D CNN_j* was slightly higher than that of *model 3D CNN_i* which was as a result of the randomization and different hyperparameters initialized. This is the benefits of developing ensemble or multiple neural networks because each model has learnt uniquely and can compliment the weakness of the other models in the overall ensemble networks.

The third variant of the emsemble networks was developed using different network archchitecure and compilation hyperparameters as shown in Figure 23. Using a totally different configuration from other ensemble models would help to improve generalization error (i.e. lower the generalization error) such that the overall model would be able to perform well on unseen data rather than the test data generated from the base dataset.

The ensemble model has three convolutional layers, three max pooling layers, a flatten layer , two fully connected layers and an output layer. This achitecture is quite different from the other two variants of the 3D CNN ensemble models in previous section. The number of filters used are thirty-two, sixty-four and one hundred and twenty eight respectively, see Figure 23. The fully connected layer is deeper in number of neurons/nodes and number of layers used.

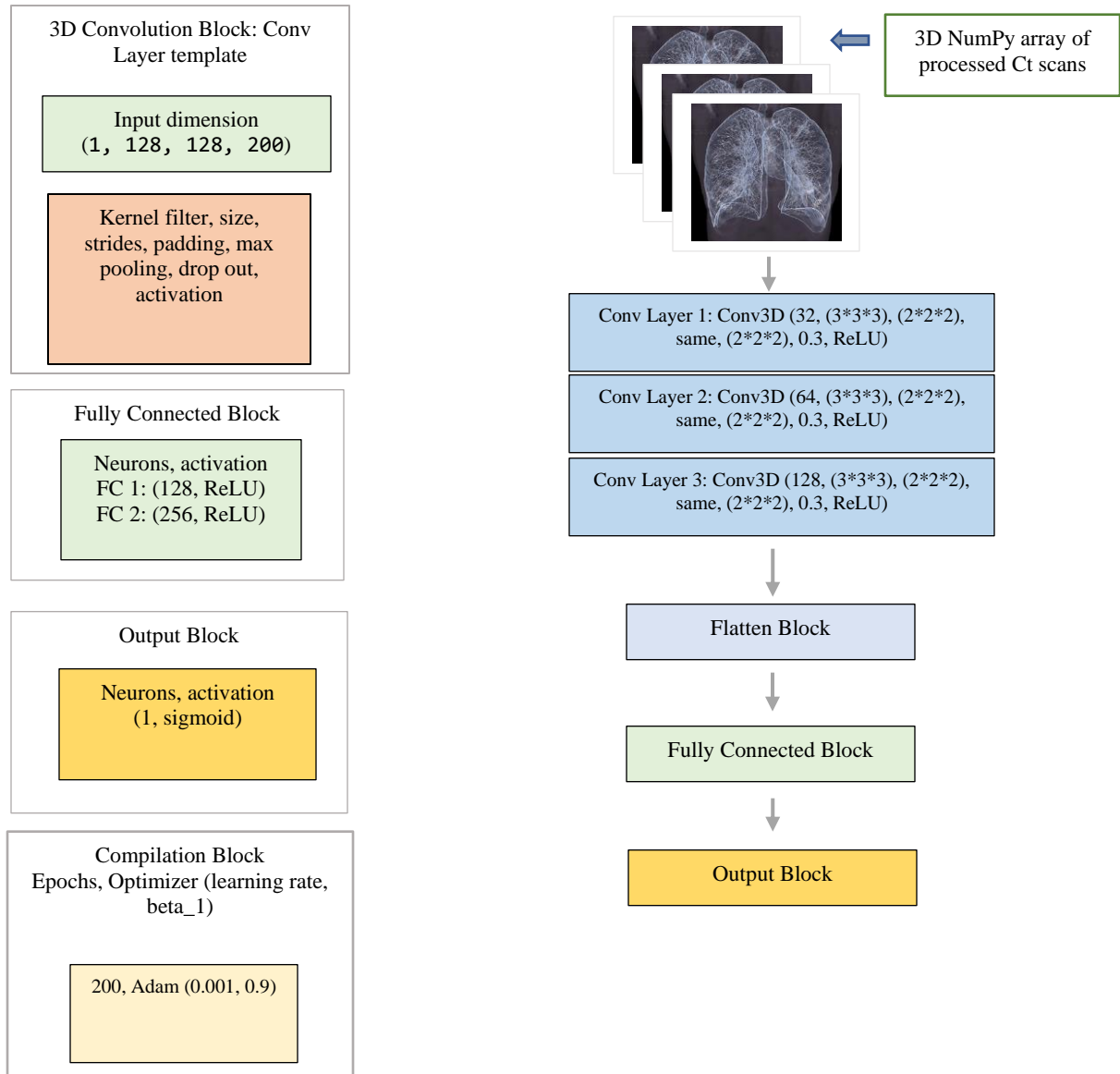


Figure 23. The architecture of 3D CNN_k ensemble model.

The deeper the number of layers, the more features the networks would learn. However, care needs to be taken because of over-fitting where the network memorizes instead of learning which will increase the FPR when tested on unseen data.

The ensemble 3D CNN_k resulted to a total of 6,070,977 trainable parameters due to the different

configuration parameters initialized before training. The network was trained for 200 epochs and the accuracy and loss plots generated during training and validation are shown in Figure 24 and 25.

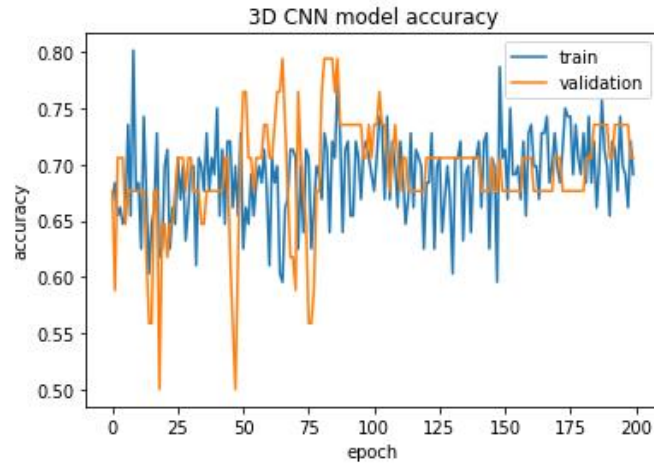


Figure 24. Plot showing the training and validation accuracy of *ensemble 3D CNN_k* over 200 epochs.

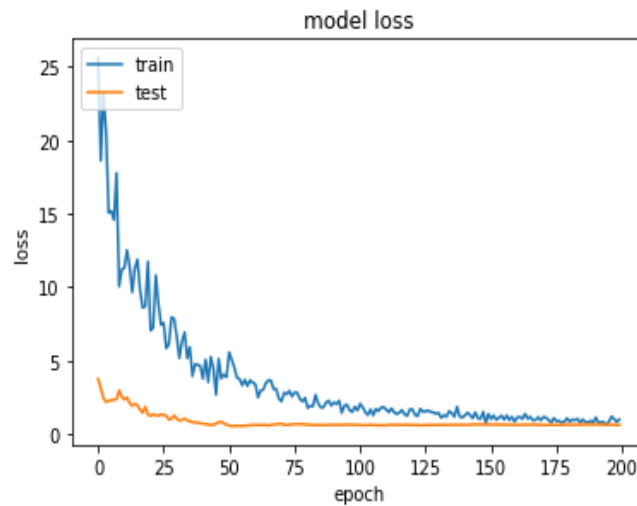


Figure 25. Plot showing the training and validation loss of *ensemble 3D CNN_k* over 200 epochs.

The accuracy plot was quite rough and this was due small amount of dataset used in training and validating the performance of the model. The loss gradually reduces and flattens out from epoch 75 and above which showed that the network did not over-fit.

4.3.4 CADeC Software Inference Engine Development

The final phase of the CADeC application is the inference engine which was written in python (code in the appendix 11). This engine is responsible for predicting and averaging the predicted results of the voxel nodule and non-nodule of a raw CT scan passed to it as input as shown in Figure 26. The engine is dependent on the in memory saved weights of all the ensemble 3D DNN models developed.

The inference engine has an interactive shell that allows users to specify the file format and the application mode. The application supports only dicom and mhd/raw medical imaging formats, users can also specify either of the two application modes supported (light and extended modes). Light mode returns to the console the last slice of the CT scan and the final predicted voxel computed while the extended version gives a more detailed analysis report of each ensemble model such as the classification accuracy, loss, false positive rate, false negative rate, true negative rate and true positive rate. It also returns to the console the predicted probability of each ensemble model, the averaged probability, last slice of the CT scan and the final voxel prediction.

The weights of the ensemble models were loaded from local storage into application memory by the inference engine controller. The file type and mode specified by the user were extracted using argument parser extractor and image pre-processor loads the appropriate image file specified by the user, processes it into the neural network format the ensemble models were all built on. The base controller engine determines what to process based on the mode specified by the user and calls the appropriate sub-routine.

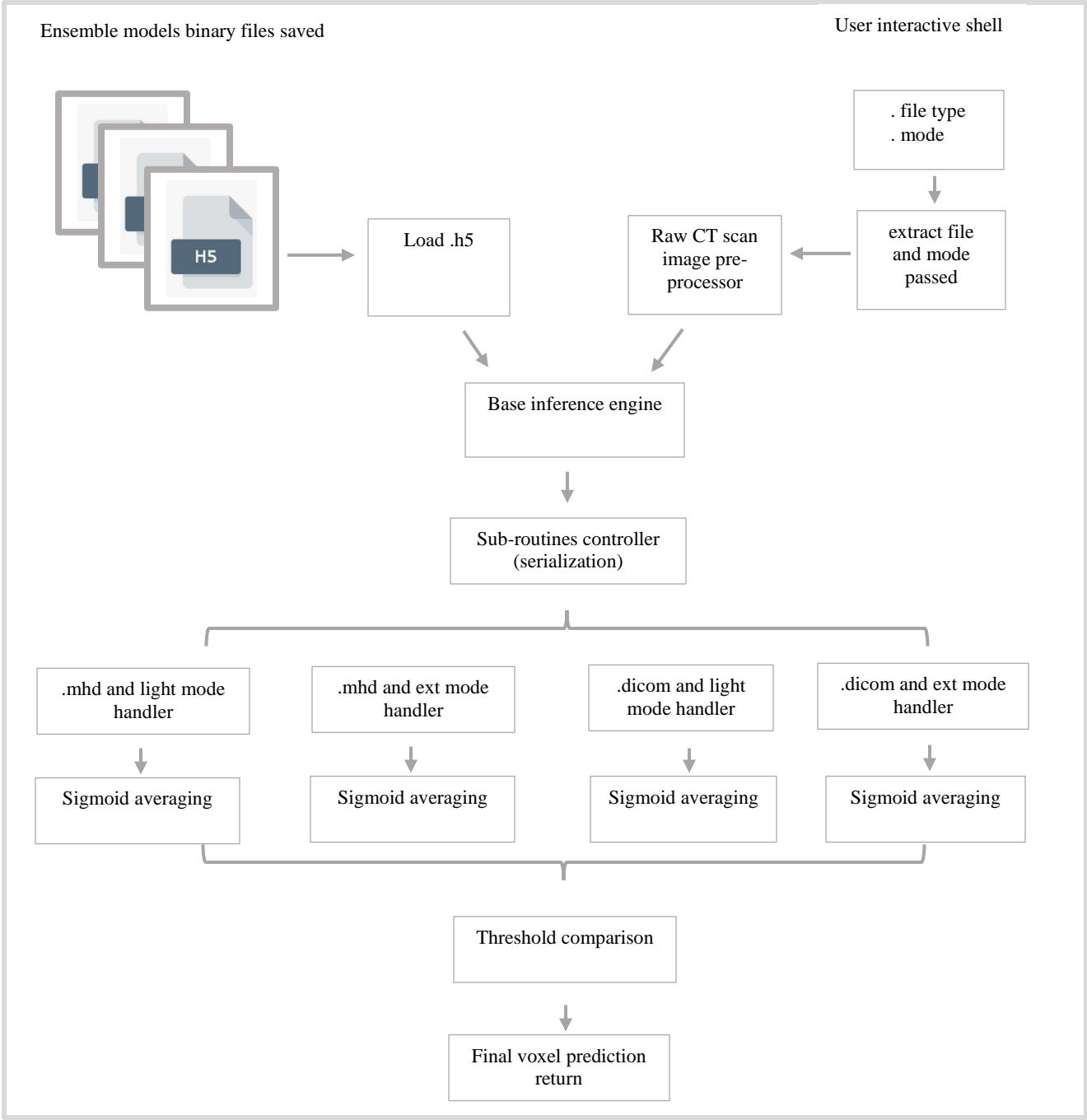


Figure 26. Blocks showing the implementation pipeline of the CADeC software inference engine.

There are 4 sub-routines: mhd-light mode handler, mhd-extended mode handler, dicom-light mode and dicom-extended mode handler. Each handler is a sub-routine that is controlled by the sub-routines' controller and can only be called once during the program life cycle. The handler returns the probabilities of the voxel nodule or non-nodule (i.e. if CT has a voxel nodule or non-nodule) predicted by each ensemble network averaged by sigmoid averaging block. The averaged probability is compared by the comparison block based on the threshold value set and returns the final predicted voxel status (nodule or non-nodule).

This is very useful in pre-screening and post-screen of large/small database of CT scans for nodule detection and identification.

4.4 Ensemble 3D CNN Models and Inference Engine Evaluation

It is of great importance to evaluate any deep neural network model developed and the application running on the top of it. There are several statistical metrics that can be used and in this project, the most important metrics that are usually employed in evaluating classification model are considered. These metrics are sensitivity, specificity, accuracy and F1-score.

The evaluation was carried out on 20% of the total dataset for both networks $3D CNN_i$ and $3D CNN_j$. The first ensemble network was trained for 25 epochs, initial learning rate of 0.001, decay of 0.00004 and a batch size of 8 as detailed in Figure 17. However, $3D CNN_j$ was trained for 50 epochs, a batch size of 8 and compiled using initial learning rate of 0.001 and rate of decay of 0.00002, see Figure 18.

The last model in the ensemble networks differs in the network architecture as specified in Figure 19. It was trained for 200 epochs, batch size of 10 and compiled using the same learning rate as the other two ensemble models. The results of evaluation of the *ensemble networks* ($3D CNN_{i-k}$) were given in Table 3.

- Sensitivity or TPR defines how accurately the model can detect and classify voxel to be a nodule. The higher this sensitivity value the better as defined by:

$$\text{Sensitivity or Recall} = \frac{TP}{TP + FN} \quad 4.1$$

- Specificity is the true negative rate which defines how accurately the model can detect and classify voxel as non-nodule. This can be described by:

$$\text{Specificity} = \frac{TN}{TN + FP} \quad 4.2$$

- Receiver operating characteristics shows the relationship between false positives rate and true positive. FPR is given by:

$$\text{FPR} = \frac{FP}{FP + TN} \quad 4.3$$

- Accuracy is how well the model can detect and classify the true positives and true negatives

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad 4.4$$

- Precision is the total number of predicted class the model was able to accurately predict. This is defined by:

$$\text{Precision} = \frac{TP}{TP + FP} \quad 4.5$$

- F1-score plays a very vital role in the evaluation of model with imbalance class. It is the average of the sensitivity and precision and it is defined as:

$$\text{F1 score or measure} = \frac{2 (\text{Recall} * \text{Precision})}{\text{Recall} + \text{Precision}} \quad 4.6$$

Table 3. The evaluation results of each model and the ensemble networks using built in python statistical metrics in sklearn (python library for scientific computing).

Networks	Sensitivity	Specificity	Precision	F1-score	FPR	Accuracy
$3D CNN_i$	0.85	0.81	0.88	0.86	0.21	0.86
$3D CNN_j$	0.82	0.83	0.83	0.75	0.20	0.80
$3D CNN_k$	0.90	0.79	0.80	0.83	0.16	0.90
Average	0.86	0.81	0.87	0.81	0.19	0.85

The averaged results in Table 3 showed the convergence of the ensemble neural networks and how each model in the network complements one another. The last model ($3D CNN_k$) of the ensemble networks has the highest accuracy (90%), F1 score (83%), sensitivity (90%), FPR (16%) and a lowest specificity score (79%) (see Table 3) using test size of 0.2 of the total dataset. Looking at these results, it is very much possible to have decided that $3D CNN_k$ of ensemble networks is best model because it causes a significant reduction in the FPR of both $3D CNN_i$ and $3D CNN_j$. However, this would have been a false assumption because of the model's lowest specificity score which was complemented by both $3D CNN_i$ and $3D CNN_j$ models to give an average of 81%.

The first model of the ensemble network ($3D CNN_i$) generated the highest FPR and the average of the ensemble networks gave 19% as compared to 21% and 20% generated by $3D CNN_i$ and $3D CNN_j$ models of the ensemble networks.

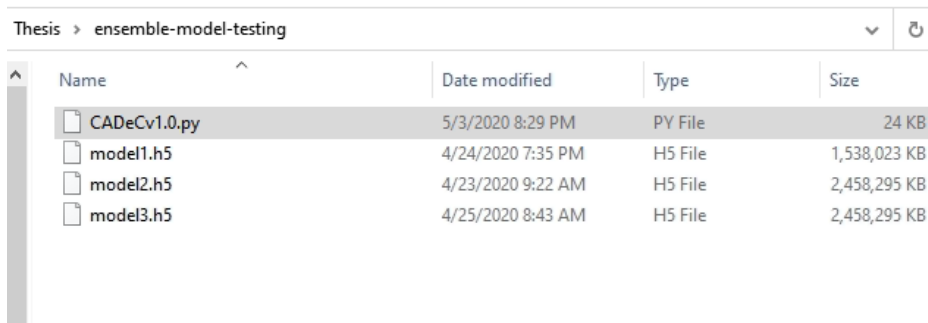
The ensemble networks of the three 3D CNN models resulted to a better generalization and less bias/variance, and this showed the effectiveness and overall efficiency of using heterogenous/ensemble based 3D deep multi neural networks in the CADeC application for lung nodule diagnosis.

4.5 Guidelines on Running Inference Application for Nodule Classification

This chapter shows how to run the application on a new patient CT scan in order to classify if it has volume nodule or non-nodule lesion. The application can process both dicom and mhd/raw CT image files of the lung. It takes two arguments, the full path of the scan files to be processed and preferred application modes (see chapter 4.3.4). To be able to run the application, all the dependencies must be installed appropriately as shown below:

- Python 3, keras 2.3.1 and theano 1.0.4
- Tensorflow 2.0.0
- Matplotlib
- Numpy
- OpenCV 3.4.3.18 or above
- pydot
- Graphviz

The application files can be cloned from the repository I created for purpose of this thesis using the link https://bitbucket.org/olutosin_/masters-thesis/src/master/. However, the weights of the ensemble model cannot be uploaded to the repository due to the file size restriction and as an alternative measure, all the files were stored on a flash-drive. There are three main folders, in ensemble models testing folder, there are four files as shown in Figure 27. Three of the files are the generated weights stored in HDF5 binary file format of the three 3D CNN that made up the ensemble model and the fourth file is the inference engine built on the stack of the ensemble model named CADeCv1.0.py.



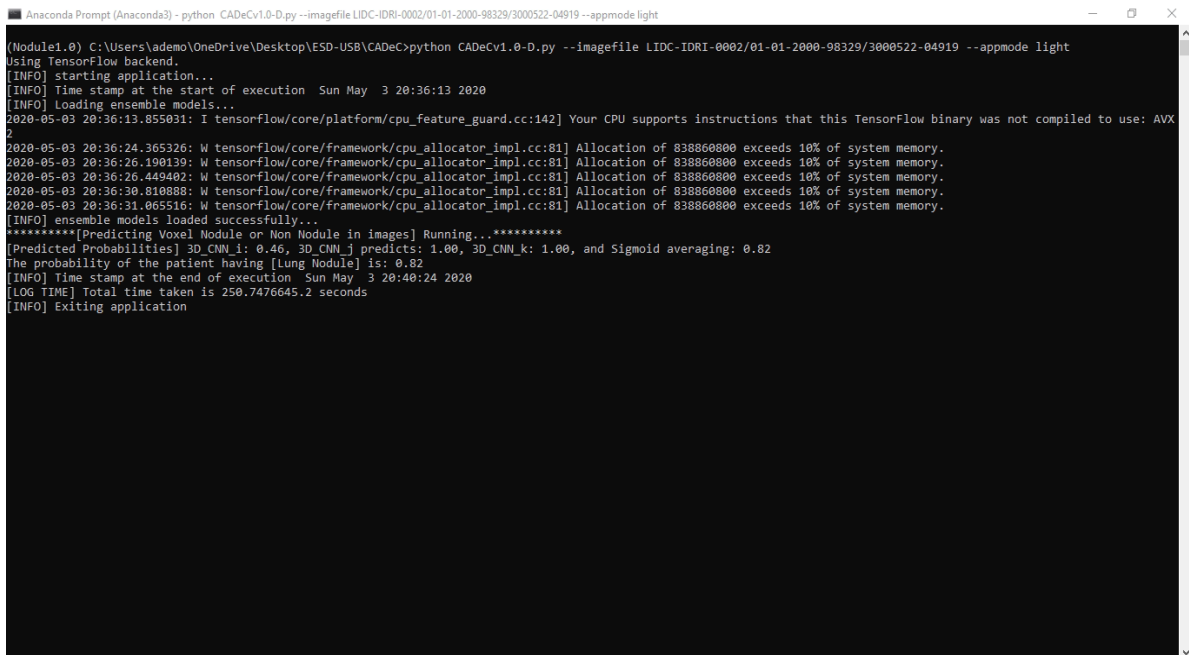
Name	Date modified	Type	Size
CADeCv1.0.py	5/3/2020 8:29 PM	PY File	24 KB
model1.h5	4/24/2020 7:35 PM	H5 File	1,538,023 KB
model2.h5	4/23/2020 9:22 AM	H5 File	2,458,295 KB
model3.h5	4/25/2020 8:43 AM	H5 File	2,458,295 KB

Figure 27. Project directory showing ensemble model weights and the inference software written in python.

To run the application on a new scan, the image files can be copied to the folder where the CADeCv1.0.py is residing. The code to run the application is defined as follow:

```
python CADeCv1.0.py --imagefile "path to file" --appmode "app mode" 4.7
```

The imagefile argument allows the application to know where the CT scan to be classified is stored and the appmode allows the user to specify which version of the application is to be used (see chapter 4.3.4). The outputs generated by the application are shown in Figure 28 and 29 respectively.



```
Anaconda Prompt (Anaconda3) - python CADeCv1.0-D.py --imagefile LIDC-IDRI-0002/01-01-2000-98329/3000522-04919 --appmode light
(Node1.0) C:\Users\ademo\OneDrive\Desktop\ESD-USB\CADeC>python CADeCv1.0-D.py --imagefile LIDC-IDRI-0002/01-01-2000-98329/3000522-04919 --appmode light
Using TensorFlow backend.
[INFO] starting application...
[INFO] Time stamp at the start of execution Sun May 3 20:36:13 2020
[INFO] Loading ensemble models...
2020-05-03 20:36:13.855031: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX
2
2020-05-03 20:36:24.365326: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 838860800 exceeds 10% of system memory.
2020-05-03 20:36:26.190139: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 838860800 exceeds 10% of system memory.
2020-05-03 20:36:26.449402: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 838860800 exceeds 10% of system memory.
2020-05-03 20:36:30.810888: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 838860800 exceeds 10% of system memory.
2020-05-03 20:36:31.065516: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 838860800 exceeds 10% of system memory.
[INFO] ensemble models loaded successfully...
*****[Predicting Voxel Nodule or Non Nodule in Images] Running...*****
[Predicted Probabilities] 3D_CNN_l: 0.46, 3D_CNN_j predicts: 1.00, 3D_CNN_k: 1.00, and Sigmoid averaging: 0.82
The probability of the patient having [Lung Nodule] is: 0.82
[INFO] Time stamp at the end of execution Sun May 3 20:40:24 2020
[LOG TIME] Total time taken is 250.7476645.2 seconds
[INFO] Exiting application
```

Figure 28. Dicom files of a patient classified as volume nodule using the application in a light mode.

```
Anaconda Prompt (Anaconda3)
(Node1e1.0) C:\Users\ademo\OneDrive\Desktop\ESD-USB\CADeC>python CADeCv1.0-D.py --imagefile LNDb-0001.mhd --appmode light
Using TensorFlow backend.
[INFO] starting application...
[INFO] Time stamp at the start of execution Sun May 3 20:29:36 2020
[INFO] Loading ensemble models...
2020-05-03 20:29:36.619319: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX
2
2020-05-03 20:29:37.032729: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 524288000 exceeds 10% of system memory.
2020-05-03 20:29:38.195461: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 524288000 exceeds 10% of system memory.
2020-05-03 20:29:38.349526: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 524288000 exceeds 10% of system memory.
2020-05-03 20:29:41.489631: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 524288000 exceeds 10% of system memory.
2020-05-03 20:29:41.647102: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 524288000 exceeds 10% of system memory.
[INFO] ensemble models loaded successfully...
*****[Predicting Voxel Nodule or Non Nodule in images] Running..*****
3D_CNN_l: 0.43, 3D_CNN_j: 0.82, 3D_CNN_k: 1.00, and Sigmoid averaging: 0.75
The probability of the patient having [Lung Nodule] is: 0.75
[INFO] Time stamp at the end of execution Sun May 3 20:31:26 2020
[LOG TIME] Total time taken is 109.6552775.2 seconds
[INFO] Exiting application

(Node1e1.0) C:\Users\ademo\OneDrive\Desktop\ESD-USB\CADeC>
```

Figure 29. MHD/RAW CT files of a patient classified as volume nodule using the application in a light mode.

Figure 28 and 28 showed the console outputs generated when the application was tested with two different files from different patients. In Figure 28, the overall classification result generated was a probability of 82% (i.e. the patient probability score of the lesion on the lung being nodule volume is 82% and 18% score of being non nodule volume). The output shown in Figure 19 was a probability of 75% of the lesion being a volume nodule and 25% otherwise.

5 Conclusion

5.1 Conclusion

CADe applications have shown great potential in the automatic detection of lung nodules in human body. The applications are powered by different machine learning and deep learning algorithms and the accuracy/precision of these applications vary due to different factors such as the quality of the CT scans, quality of annotation, the algorithm itself and computing resource. Each of this factor contributes to the overall results of the models that would drive the applications.

These variations constitute to the error rate which is otherwise called FPR generated by the models. The FPR is the likelihood of the model to give a false positive (i.e. predicting that the patient lesion is a nodule whereas it is not) and this has been a big setback that has affected CADe from being fully adopted widely by the radiologists and medical professionals for lung nodule diagnosis (see chapter 1).

This thesis presented a novel approach using deep multi-neural networks or ensemble of three 3D CNN serialized and the predicted result from the last sigmoid layer node of each network is being averaged to reduce the FPR of the overall ensemble networks. This novel approach combined dataset and architecture variation techniques. Most of the state of art approaches were based on either 2D CNN or classical machine learning techniques that involve manual feature extraction from CT scans which tends to be very difficult resulting into poor performance in term of accuracy and precision (see chapter 3).

The overall averaged error rate (FPR) generated by the ensemble networks resulted to 19% compared to 21% and 20% generated by $3D\ CNN_i$ and $3D\ CNN_j$ models of the ensemble networks. This little difference is quite significant especially in a lung nodule detection and classification application software (CADeC).

I will like to emphasize that better ensemble networks that can give better results can still be

achieved through hyperparameters optimization and increasing the size of the training dataset, however, this was hindered due to limitation of computing resources that was used in the training, testing and validation of the models developed.

As stated above, the requirements and specifications of this thesis were fulfilled. However, what can be further optimized would be discussed in the section of future work.

5.2 Challenges

The classification of lung nodule using deep neural networks comes with different challenges, and one of the major challenges is having different categories of lung nodules such as shape, texture, position, size etc. Most publicly available dataset do not have the nodules attributes separated as distinct dataset and this introduces bias due to the uneven distribution of scans belonging to each attribute, and thus, reduces the accuracy of the models when validated on a new raw CT scan.

The effect of this on ensemble model is low, however, training the ensemble model of wide varieties dataset would further reduce the error but at a trade-off of high computation power.

5.3 Future Research

The novel technique used in this master's thesis can be further developed and applied in the prediction of the bounding boxes for visual representation of the regions/locations of the lung nodule. One of the use cases can be as a tool for automatic post/pre screening of the CT scans to improve confidence level of the lung nodule analysis.

Predicting the nodules locations and enclosing bounding boxes within the exact lung nodule locations can be quite challenging due to high false positive rate. This usually leads to making wrong predictions of nodule candidate and enclosing bounding boxes within these wrong regions. Hence, the novel approach introduced in this master's thesis can be further explored to reduce this false positive rate of the bounding boxes.

Appendix 1

This section contains the complete python code that was used to pre-process the mhd/raw CT scans images into the format (NumPy array) accepted by the 3D CNN. There are two code files in this section: directory sorting file and the main 3D image pre-processor (see chapter 4.3.2).

Directory Sorting Code

```
1. # Import all libraries required
2. import matplotlib.pyplot as plt
3. import SimpleITK as sitk
4. from pathlib import Path
5. import pandas as pd
6. import numpy as np
7. import warnings
8. import shutil
9. import glob
10. import math
11. import cv2
12. import os
13.
14. print("[INFO] starting application...")
15.
16.
17. # scans directory
18. warnings.filterwarnings("ignore")
19. ScansPath = os.listdir("../Nodule1.0/ct-scans/")
20. files = []
21. for file in ScansPath:
22.     if file.endswith(".mhd"):
23.         files.append(file)
24. print(len(files))
25.
26.
27. # creating folders with corresponding
28. # names
29. full_path = []
30. parent_dir = os.listdir("../Nodule1.0/ct-scans/")
31. for parent in parent_dir:
32.     if parent.endswith(".mhd"):
33.         path = parent
34.         p = str(path[0:9])
35.         print(p)
36.         path_ = os.path.join("../Nodule1.0/scan_dir/", p)
37.         os.makedirs(path_)
```

```

38.         full_path.append(path_)
39. print(full_path)
40.
41. # creating corresponding textfiles
42. # creating text files to hold paths
43. # to .mhd and .raw to file
44. full_path_ = []
45. for file in os.listdir("../Nodule1.0/scan_dir/"):
46.     path = os.path.join("../Nodule1.0/scan_dir/", file)
47.     full_path_.append(path)
48. print(full_path_)
49.
50. start_num = 1
51.
52. # creating folders
53. for folder in full_path_:
54.     path = folder + "/"
55.     print(path)
56.     k = path[22:31]
57.     print(k)
58.     for file in ScansPath:
59.         if file.endswith(".mhd"):
60.             mhd_copy = file
61.             y = mhd_copy.split(".", 1)[0]
62.             if k == y:
63.                 print(k, y)
64.                 filename = "../Nodule1.0/scan_textpath/" + y + ".txt"
65.                 f = open(filename, "a")
66.                 paths = os.path.join("../Nodule1.0/ct-scans/", file)
67.                 f.write(paths + "\n")
68.
69.         if file.endswith(".raw"):
70.             raw_copy = file
71.             x = raw_copy.split(".", 1)[0]
72.             if k == x:
73.                 print(k, x)
74.                 print(file, path)
75.                 filename = "../Nodule1.0/scan_textpath/" + x + ".txt"
76.                 f = open(filename, "a")
77.                 path_ = os.path.join("../Nodule1.0/ct-scans/", file)
78.                 f.write(path_ + "\n")
79.                 start_num += 1
80. print(len(os.listdir("../Nodule1.0/scan_textpath")))
81.
82. # loading text_files
83.
84. text_file = []
85. path_to_textfile = os.listdir("../Nodule1.0/scan_textpath/")
86. for text in path_to_textfile:
87.     complete_path = os.path.join("../Nodule1.0/scan_textpath/", text)
88.     text_file.append(complete_path)
89. print(text_file)
90. print(" ")
91.
92. # load the labels
93. labels_df = pd.read_csv("../Nodule1.0/lnDb-labels/LNDb_label.csv", index_col=0)
94. print(labels_df.tail())
95.
96. position = []
97. for i, txt in enumerate(text_file):
98.     f = open(txt, "r")

```

```

99.     mhd = f.readline()
100.    raw = f.readline()
101.    path = mhd
102.    path = path[27:31]
103.    position.append(path)
104.    print(len(position))
105.
106.    image3D = []
107.    for j, i in enumerate(position[:140]):
108.        label = labels_df.get_value(j, 'Nodule')
109.        path_to_scan = "../Nodule1.0/ct-scans/LNDb-{}".format(i)
110.        image = sitk.GetArrayFromImage(sitk.ReadImage(os.path.abspath(path_to_scan)))
111.        image3D.append(image)
112.        print(image.shape, label, j)
113.
114.    # make a copy of image3D
115.    scan3D = image3D.copy()
116.    print(len(scan3D))
117.
118.    # visualizing individual slice
119.    for i, slice in enumerate(image3D[30]):
120.        if (i < 180):
121.            print(slice.shape)
122.            plt.imshow(slice, cmap = "gray")
123.            plt.show()
124.        else:
125.            break
126.
127.
128.    # helping functions
129.    # @lst is a list
130.    # @n is chunks size
131.    def extract(mylist, a):
132.        for i in range(0, len(mylist), a):
133.            yield mylist[i:i + a]
134.
135.    def average(l):
136.        return sum(l) / len(l)
137.
138.    # 3D
139.    # @ test is the main list == patients
140.    # for i, patient in enumerate(test)
141.    # print(patient.shape)
142.    # 3D list of list
143.    w = 64
144.    h = 64
145.    #img_stack = 180
146.
147.    new_slice3D = []
148.    length_slices_3D = []
149.    label_main = []
150.    roll_axis3D = []
151.    new_slices3D = []
152.
153.    # creates a list of number
154.    stack = range(1, 181)
155.    img_stack = []
156.    for i in stack:
157.        img_stack.append(i)
158.
159.

```

```

160. for i, patients in enumerate(scan3D):
161.     print(patients.shape, i)
162.
163.     label = labels_df.get_value(i, 'Nodule')
164.     label_main.append(label)
165.
166.     new_slices = []
167.     for i, slice in enumerate(patients):
168.         if (i < 179) or (i == 179):
169.             img_sm = cv2.resize(np.array(slice), (w, h), interpolation=cv2.INTER_CUBIC)
170.
171.             new_slices.append(img_sm)
172.             print("shape of image stack = {a} and i = {b}".format(a = img_sm.shape, b =
i))
173.         else:
174.             break
175.
176.     slices_resized_3d = np.asarray(new_slices)
177.     # Length of Main 3D
178.     length_slices_3D.append(len(slices_resized_3d))
179.
180.     # Main 3D array
181.     new_slices3D.append(slices_resized_3d)
182.
183.     # append length of slices after each loop
184.     input = slices_resized_3d
185.     print("shape before rolling:", np.array(new_slices3D).shape)
186.
187.     # Rolling
188.     ipt = np.rollaxis(np.rollaxis(input, 2, 0), 2, 0)
189.     roll_axis3D.append(ipt)
190.     print(ipt.shape)
191.     print("New slice 3D:", np.array(new_slices3D).shape)
192.
193.
194. # saving data
195. data_array = np.array(new_slices3D)
196. np.save("50-250-128-128.npy", data_array)
197. label_array = np.array(label_main)
198. np.save("label-50-250-128-128.npy", label_array)
199.
200. print(os.listdir())
201.
202. print("[INFO] Program execution status: success")

```

3D CT Scans Preprocessing Code

```

1. # File for 3D CT scan pre-processing
2. # Import all libraries required
3. import matplotlib.pyplot as plt
4. import SimpleITK as sitk
5. from pathlib import Path
6. import pandas as pd
7. import numpy as np
8. import warnings
9. import shutil
10. import glob

```

```

11. import math
12. import cv2
13. import os
14.
15. print("[INFO] starting application...")
16.
17. # scans directory
18. warnings.filterwarnings("ignore")
19. ScansPath = os.listdir("../Nodule1.0/ct-scans/")
20. files = []
21. for file in ScansPath:
22.     if file.endswith(".mhd"):
23.         files.append(file)
24. print(len(files))
25.
26.
27. # creating folders with corresponding
28. # names
29. full_path = []
30. parent_dir = os.listdir("../Nodule1.0/ct-scans/")
31. for parent in parent_dir:
32.     if parent.endswith(".mhd"):
33.         path = parent
34.         p = str(path[0:9])
35.         print(p)
36.         path_ = os.path.join("../Nodule1.0/scan_dir/", p)
37.         os.makedirs(path_)
38.         full_path.append(path_)
39. print(full_path)
40.
41. # creating corresponding textfiles
42. # creating text files to hold paths
43. # to .mhd and .raw to file
44. full_path_ = []
45. for file in os.listdir("../Nodule1.0/scan_dir/"):
46.     path = os.path.join("../Nodule1.0/scan_dir/", file)
47.     full_path_.append(path)
48. print(full_path_)
49.
50. start_num = 1
51.
52. # creating folders
53. for folder in full_path_:
54.     path = folder + "/"
55.     print(path)
56.     k = path[22:31]
57.     print(k)
58.     for file in ScansPath:
59.         if file.endswith(".mhd"):
60.             mhd_copy = file
61.             y = mhd_copy.split(".", 1)[0]
62.             if k == y:
63.                 print(k, y)
64.                 filename = "../Nodule1.0/scan_textpath/" + y + ".txt"
65.                 f = open(filename, "a")
66.                 paths = os.path.join("../Nodule1.0/ct-scans/", file)
67.                 f.write(paths + "\n")
68.
69.         if file.endswith(".raw"):
70.             raw_copy = file
71.             x = raw_copy.split(".", 1)[0]

```

```

72.         if k == x:
73.             print(k, x)
74.             print(file, path)
75.             filename = "../Nodule1.0/scan_textpath/" + x + ".txt"
76.             f = open(filename, "a")
77.             path_ = os.path.join("../Nodule1.0/ct-scans/", file)
78.             f.write(path_ + "\n")
79.             start_num += 1
80. print(len(os.listdir("../Nodule1.0/scan_textpath")))
81.
82. # loading text_files
83.
84. text_file = []
85. path_to_textfile = os.listdir("../Nodule1.0/scan_textpath/")
86. for text in path_to_textfile:
87.     complete_path = os.path.join("../Nodule1.0/scan_textpath/", text)
88.     text_file.append(complete_path)
89. print(text_file)
90. print(" ")
91.
92. # load the labels
93. labels_df = pd.read_csv("../Nodule1.0/lndb-labels/LNDb_label.csv", index_col=0)
94. print(labels_df.tail())
95.
96. position = []
97. for i, txt in enumerate(text_file):
98.     f = open(txt, "r")
99.     mhd = f.readline()
100.    raw = f.readline()
101.    path = mhd
102.    path = path[27:31]
103.    position.append(path)
104. print(len(position))
105.
106. image3D = []
107. for j, i in enumerate(position[:140]):
108.    label = labels_df.get_value(j, 'Nodule')
109.    path_to_scan = "../Nodule1.0/ct-scans/LNDb-{}.mhd".format(i)
110.    image = sitk.GetArrayFromImage(sitk.ReadImage(os.path.abspath(path_to_scan)))
111.    image3D.append(image)
112.    print(image.shape, label, j)
113.
114. # make a copy of image3D
115. scan3D = image3D.copy()
116. print(len(scan3D))
117.
118. # visualizing individual slice
119. for i, slice in enumerate(image3D[30]):
120.    if (i < 180):
121.        print(slice.shape)
122.        plt.imshow(slice, cmap = "gray")
123.        plt.show()
124.    else:
125.        break
126.
127.
128. # helping functions
129. # @lst is a list
130. # @n is chunks size
131. def extract(mylist, a):
132.    for i in range(0, len(lst), a):

```

```

133.         yield mylst[i:i + a]
134.
135. def average(l):
136.     return sum(l) / len(l)
137.
138. # 3D
139. # @ test is the main list == patients
140. # for i, patient in enumerate(test)
141. # print(patient.shape)
142. # 3D list of list
143. w = 64
144. h = 64
145. #img_stack = 180
146.
147. new_slice3D = []
148. length_slices_3D = []
149. label_main = []
150. roll_axis3D = []
151. new_slices3D = []
152.
153. # creates a list of number
154. stack = range(1, 181)
155. img_stack = []
156. for i in stack:
157.     img_stack.append(i)
158.
159.
160. for i, patients in enumerate(scan3D):
161.     print(patients.shape, i)
162.
163.     label = labels_df.get_value(i, 'Nodule')
164.     label_main.append(label)
165.
166.     new_slices = []
167.     for i, slice in enumerate(patients):
168.         if (i < 179) or (i == 179):
169.             img_sm = cv2.resize(np.array(slice), (w, h), interpolation=cv2.INTER_CUBIC)
170.
171.             new_slices.append(img_sm)
172.             print("shape of image stack = {a} and i = {b}".format(a = img_sm.shape, b =
i))
173.         else:
174.             break
175.
176.     slices_resized_3d = np.asarray(new_slices)
177.     # Length of Main 3D
178.     length_slices_3D.append(len(slices_resized_3d))
179.
180.     # Main 3D array
181.     new_slices3D.append(slices_resized_3d)
182.
183.     # append length of slices after each loop
184.     input = slices_resized_3d
185.     print("shape before rolling:", np.array(new_slices3D).shape)
186.
187.     # Rolling
188.     ipt = np.rollaxis(np.rollaxis(input, 2, 0), 2, 0)
189.     roll_axis3D.append(ipt)
190.     print(ipt.shape)
191.     print("New slice 3D:", np.array(new_slices3D).shape)

```

```
192.  
193.  
194.# saving data  
195.data_array = np.array(new_slices3D)  
196.np.save("50-250-128-128.npy", data_array)  
197.label_array = np.array(label_main)  
198.np.save("label-50-250-128-128.npy", label_array)  
199.  
200.print(os.listdir())  
201.  
202.print("[INFO] Program execution status: success")
```


Appendix 2

This section holds the python code that was used to develop the three models that made up the ensemble networks $3D CNN_i$, $3D CNN_j$ and $3D CNN_k$ (see chapter 4.4.3).

3D CNN_i Model Code

```
1. # Import all libraries required
2. import keras
3. from keras.layers import Dense, Dropout, Activation, Flatten
4. from keras.layers.normalization import BatchNormalization
5. from keras.preprocessing.image import ImageDataGenerator
6. from keras.layers import Convolution3D, MaxPooling3D
7. from sklearn.model_selection import train_test_split
8. from keras.utils import np_utils, generic_utils
9. from keras.optimizers import Adam, SGD
10. from keras.utils import to_categorical
11. from keras.models import Sequential
12. from sklearn import model_selection
13. from sklearn import preprocessing
14. import matplotlib.pyplot as plt
15. from datetime import datetime
16. import numpy as np
17. import time
18. import cv2
19. import os
20.
21. print("[INFO] starting application...")
22.
23.
24. # Load the dataset
25. dataset = np.load("50-250-128-128.npy")
26. label = np.load("label-50-250-128-128.npy")
27.
28. dataset_ = np.rollaxis(np.rollaxis(dataset, 3, 1), 3, 2)
29. data_ = dataset_
30.
31. # Transforms the label
32. labels = np.ones((50, ), dtype = int)
33. labels = label
34.
35. # Transforming dataset
36. Train_data = [data_, labels]
37. X_train, Y_train =(Train_data[0], Train_data[1])
38.
39. print("Shape of X_train is :", X_train.shape)
40.
41. # Transforming 4D into 5D for CNN Model
42. Train_set = np.ones((50, 1, 128, 128, 250))
```

```

43. for h in range(50):
44.     Train_set[h][0][:][:][:] = X_train[h,:,:,:]
45.
46.
47. # Developing the Model
48. model = Sequential()
49.
50.
51. # Conv Block 1
52. model.add(Convolution3D(32, kernel_size = (3,3,3), strides = (1,1,1), padding = "same",
    data_format = "channels_first", activation = "relu"))
53. model.add(MaxPooling3D(pool_size = (2,2,2)))
54. model.add(BatchNormalization())
55. model.add(Dropout(0.3))
56.
57. # Conv Block 2
58. model.add(Convolution3D(32, kernel_size = (3,3,3), strides = (1,1,1), padding = "same",
    data_format = "channels_first", activation = "relu"))
59. model.add(MaxPooling3D(pool_size = (2,2,2)))
60. model.add(BatchNormalization())
61. model.add(Dropout(0.3))
62.
63.
64. # Conv Block3
65. model.add(Convolution3D(64, kernel_size = (3,3,3), strides = (1,1,1), padding = "same",
    data_format = "channels_first", activation = "relu"))
66. model.add(MaxPooling3D(pool_size = (2,2,2)))
67. model.add(BatchNormalization())
68. model.add(Dropout(0.3))
69.
70. # Conv Block4
71. model.add(Convolution3D(128, kernel_size = (3,3,3), strides = (1,1,1), padding = "same"
    , data_format = "channels_first", activation = "relu"))
72. model.add(MaxPooling3D(pool_size = (2,2,2)))
73. model.add(BatchNormalization())
74. model.add(Dropout(0.3))
75.
76. # Conv Block5
77. model.add(Convolution3D(256, kernel_size = (3,3,3), strides = (1,1,1), padding = "same"
    , data_format = "channels_first", activation = "relu"))
78. model.add(MaxPooling3D(pool_size = (2,2,2)))
79. model.add(BatchNormalization())
80. model.add(Dropout(0.3))
81.
82.
83. model.add(Flatten())
84.
85. # FCC Blocks
86. model.add(Dense(128, activation="relu"))
87.
88.
89. # Output Block
90. model.add(Dense(1, activation = "sigmoid"))
91.
92. # Model Summary
93. print(model.summary())
94.
95. # Compilation and Training of Model
96. from keras.optimizers import Adam
97. model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])

```

```

98.
99. # Splitting dataset
100.X_train_new, X_val_new, y_train_new,y_val_new = train_test_split(Train_set, Y_train, t
    est_size=0.2, random_state=None)
101.
102.# Training Model
103.hist = model.fit(X_train_new, y_train_new, validation_data=(X_val_new,y_val_new),batch_
    size=10, nb_epoch = 25, shuffle=True)
104.
105.start_of_training = datetime.now()
106.model.save("../model1.h5")
107.
108.end_of_training = datetime.now()
109.
110.time_taken = end_of_training - start_of_training
111.print(time_taken)
112.
113.# Model Evaluation
114.import pydot
115.import graphviz
116.score = model.evaluate(X_val_new, y_val_new)
117.print('Test score loss:', score[0])
118.print('Test score accuracy:', score[1])
119.
120.
121.# summarize history for loss and accuracy
122.plt.figure()
123.plt.plot(hist.history['loss'])
124.plt.plot(hist.history['val_loss'])
125.plt.title('3D CNN model')
126.plt.ylabel('loss')
127.plt.xlabel('epoch')
128.plt.legend(['train', 'validation'], loc='upper right')
129.plt.show()
130.
131.
132.plt.plot(hist.history['accuracy'])
133.plt.plot(hist.history['val_accuracy'])
134.plt.title('3D CNN model accuracy')
135.plt.ylabel('accuracy')
136.plt.xlabel('epoch')
137.plt.legend(['train', 'validation'], loc='upper right')
138.plt.show()
139.
140.from keras.utils.vis_utils import plot_model
141.plot_model(model, to_file = "../thesis_ct_scan/140-1-180-64-64-Model25.png")
142.
143.
144.print("[INFO] Application sucessfully executed")

```

3D CNN; Model Code

```

1. # Import all libraries required
2. import keras
3. from keras.layers import Dense, Dropout, Activation, Flatten
4. from keras.layers.normalization import BatchNormalization
5. from keras.preprocessing.image import ImageDataGenerator

```

```

6. from keras.layers import Convolution3D, MaxPooling3D
7. from sklearn.model_selection import train_test_split
8. from keras.utils import np_utils, generic_utils
9. from keras.optimizers import Adam, SGD
10. from keras.utils import to_categorical
11. from keras.models import Sequential
12. from sklearn import model_selection
13. from sklearn import preprocessing
14. import matplotlib.pyplot as plt
15. from datetime import datetime
16. import numpy as np
17. import time
18. import cv2
19. import os
20.
21. print("[INFO] starting application...")
22.
23. # Load the dataset
24. dataset = np.load("50-200-128-128.npy")
25. label = np.load("label-50-200-128-128.npy")
26.
27. dataset_ = np.rollaxis(np.rollaxis(dataset, 3, 1), 3, 2)
28. data_ = dataset_
29.
30. # Transform the labels
31. labels = np.ones((50, ), dtype = int)
32. labels = label
33.
34.
35. Train_data = [data_, labels]
36. X_train, Y_train =(Train_data[0], Train_data[1])
37. print("Shape of X_train is :", X_train.shape)
38.
39. # Transforming 4D into 5D for CNN Model
40. Train_set = np.ones((50, 1, 128, 128, 200))
41. for h in range(50):
42.     Train_set[h][0][:][:][:] = X_train[h,:,:,:]
43.
44.
45. # Developing the Model
46. model = Sequential()
47.
48. # Conv Block1
49. model.add(Convolution3D(32, kernel_size = (3,3,3), strides = (1,1,1), padding = "same",
    data_format = "channels_first", activation = "relu"))
50. model.add(MaxPooling3D(pool_size = (2,2,2)))
51. model.add(BatchNormalization())
52. model.add(Dropout(0.3))
53.
54. # Conv Block2
55. model.add(Convolution3D(32, kernel_size = (3,3,3), strides = (1,1,1), padding = "same",
    data_format = "channels_first", activation = "relu"))
56. model.add(MaxPooling3D(pool_size = (2,2,2)))
57. model.add(BatchNormalization())
58. model.add(Dropout(0.3))
59.
60. # Conv Block3
61. model.add(Convolution3D(64, kernel_size = (3,3,3), strides = (1,1,1), padding = "same",
    data_format = "channels_first", activation = "relu"))
62. model.add(MaxPooling3D(pool_size = (2,2,2)))
63. model.add(BatchNormalization())

```

```

64. model.add(Dropout(0.3))
65.
66. # Conv Block4
67. model.add(Convolution3D(128, kernel_size = (3,3,3), strides = (1,1,1), padding = "same"
, data_format = "channels_first", activation = "relu"))
68. model.add(MaxPooling3D(pool_size = (2,2,2)))
69. model.add(BatchNormalization())
70. model.add(Dropout(0.3))
71.
72. # Conv Block5
73. model.add(Convolution3D(256, kernel_size = (3,3,3), strides = (1,1,1), padding = "same"
, data_format = "channels_first", activation = "relu"))
74. model.add(MaxPooling3D(pool_size = (2,2,2)))
75. model.add(BatchNormalization())
76. model.add(Dropout(0.3))
77.
78.
79.
80. model.add(Flatten())
81.
82. # FCC Blocks:
83. model.add(Dense(128, activation="relu"))
84.
85.
86. # Output Block
87. model.add(Dense(1, activation = "sigmoid"))
88.
89. # Model Summary
90. print(model.summary())
91.
92. # Compilation and training of the model
93. model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])
94.
95.
96. # Splitting dataset
97. X_train_new, X_val_new, y_train_new,y_val_new = train_test_split(Train_set, Y_train, t
est_size=0.2, random_state=None)
98.
99.
100.# Training Model
101.print(format("[INFO] starting training...", "^100"))
102.hist = model.fit(X_train_new, y_train_new, validation_data=(X_val_new,y_val_new),batch_
size=5, nb_epoch = 50, shuffle=True)
103.
104.
105.start_of_training = datetime.now()
106.model.save("model2.h5")
107.#hist = model.fit(train_set, Y_train, batch_size=batch_size,
108.#         nb_epoch=nb_epoch,validation_split=0.2, show_accuracy=True,
109.#         shuffle=True)
110.end_of_training = datetime.now()
111.
112.time_taken = end_of_training - start_of_training
113.print("Totatl training time is: ", time_taken)
114.
115.
116.# Model Evaluation
117.import pydot
118.import graphviz
119.score = model.evaluate(X_val_new, y_val_new)

```

```

120.print('Test score loss:', score[0])
121.print('Test score accuracy:', score[1])
122.
123.
124.# Plot history for loss and accuracy of the model
125.plt.figure()
126.plt.plot(hist.history['loss'])
127.plt.plot(hist.history['val_loss'])
128.plt.title('3D CNN model')
129.plt.ylabel('loss')
130.plt.xlabel('epoch')
131.plt.legend(['train', 'validation'], loc='upper right')
132.plt.show()
133.
134.
135.plt.plot(hist.history['accuracy'])
136.plt.plot(hist.history['val_accuracy'])
137.plt.title('3D CNN model accuracy')
138.plt.ylabel('accuracy')
139.plt.xlabel('epoch')
140.plt.legend(['train', 'validation'], loc='upper right')
141.plt.show()
142.
143.from keras.utils.vis_utils import plot_model
144.plot_model(model, to_file = "Model2.png")
145.
146.
147.print("[INFO] Application successfully executed")

```

3D CNN_k Model Code

```

1. # Import all libraries required
2. import keras
3. from keras.layers import Dense, Dropout, Activation, Flatten
4. from keras.layers.normalization import BatchNormalization
5. from keras.preprocessing.image import ImageDataGenerator
6. from keras.layers import Convolution3D, MaxPooling3D
7. from sklearn.model_selection import train_test_split
8. from keras.utils import np_utils, generic_utils
9. from keras.optimizers import Adam, SGD
10. from keras.utils import to_categorical
11. from keras.models import Sequential
12. from sklearn import model_selection
13. from sklearn import preprocessing
14. import matplotlib.pyplot as plt
15. from datetime import datetime
16. import numpy as np
17. import time
18. import cv2
19. import os
20.
21. # Load the dataset
22.
23. dataset = np.load("50-200-128-128.npy")
24. label = np.load("label-50-200-128-128.npy")
25.
26. dataset_ = np.rollaxis(np.rollaxis(dataset, 3, 1), 3, 2)
27. data_ = dataset_

```

```

28.
29. # Transforms the label
30. labels = np.ones((50, ), dtype = int)
31. labels = label
32.
33. # Transforming dataset
34. Train_data = [data_, labels]
35. X_train, Y_train =(Train_data[0], Train_data[1])
36.
37. print("Shape of X_train is :", X_train.shape)
38.
39. # Transforming 4D into 5D for CNN Model
40. Train_set = np.ones((50, 1, 128, 128, 200))
41. for h in range(50):
42.     Train_set[h][0][:][:][:] = X_train[h,:,:,:]
43.
44.
45.
46. # Developing the Model
47. model = Sequential()
48.
49.
50. # Conv Block1
51. model.add(Convolution3D(32, kernel_size = (3,3,3), strides = (1,1,1), padding = "same",
    data_format = "channels_first", activation = "relu"))
52. model.add(MaxPooling3D(pool_size = (2,2,2)))
53. model.add(BatchNormalization())
54. model.add(Dropout(0.3))
55.
56. # Conv Block2
57. model.add(Convolution3D(64, kernel_size = (3,3,3), strides = (1,1,1), padding = "same",
    data_format = "channels_first", activation = "relu"))
58. model.add(MaxPooling3D(pool_size = (2,2,2)))
59. model.add(BatchNormalization())
60. model.add(Dropout(0.3))
61.
62. # Conv Block3
63. model.add(Convolution3D(128, kernel_size = (3,3,3), strides = (1,1,1), padding = "same"
    , data_format = "channels_first", activation = "relu"))
64. model.add(MaxPooling3D(pool_size = (2,2,2)))
65. model.add(BatchNormalization())
66. model.add(Dropout(0.3))
67.
68. model.add(Flatten())
69.
70. # FCC Blocks:
71. model.add(Dense(128, activation="relu"))
72.
73. model.add(Dense(256, activation="relu"))
74.
75. # Output Block
76. model.add(Dense(1, activation = "sigmoid"))
77.
78. # Model Summary
79. print(model.summary())
80.
81.
82. # Compilation and Training of Model
83. from keras.optimizers import Adam
84. model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])

```

```

85.
86.
87. # Splitting dataset
88. X_train_new, X_val_new, y_train_new, y_val_new = train_test_split(Train_set, Y_train, t
    est_size=0.2, random_state=None)
89.
90.
91. # Training Model
92. hist = model.fit(X_train_new, y_train_new, validation_data=(X_val_new, y_val_new), batch_
    size=5, nb_epoch = 200, shuffle=True)
93.
94. start_of_training = datetime.now()
95. model.save("model3.h5")
96.
97.
98. end_of_training = datetime.now()
99.
100. time_taken = end_of_training - start_of_training
101. print(time_taken)
102.
103.
104. # Model Evaluation
105. import pydot
106. import graphviz
107. score = model.evaluate(X_val_new, y_val_new)
108. print('Test score loss:', score[0])
109. print('Test score accuracy:', score[1])
110.
111.
112. # summarize history for loss and accuracy
113. plt.figure()
114. plt.plot(hist.history['loss'])
115. plt.plot(hist.history['val_loss'])
116. plt.title('3D CNN model')
117. plt.ylabel('loss')
118. plt.xlabel('epoch')
119. plt.legend(['train', 'validation'], loc='upper right')
120. plt.show()
121.
122.
123. plt.plot(hist.history['accuracy'])
124. plt.plot(hist.history['val_accuracy'])
125. plt.title('3D CNN model accuracy')
126. plt.ylabel('accuracy')
127. plt.xlabel('epoch')
128. plt.legend(['train', 'validation'], loc='upper right')
129. plt.show()
130.
131. from keras.utils.vis_utils import plot_model
132. plot_model(model, to_file = "Model2.png")

```


Appendix 3

This section holds the python code for the CADeC software inference engine developed for evaluating the ensemble networks (see chapter 4.3.4).

CADeC Software Inference Engine Code

```
1. # Import all libraries required
2. import keras
3. from keras.models import load_model
4. import matplotlib.pyplot as plt
5. import pydicom as pydicom
6. import SimpleITK as sitk
7. from pathlib import Path
8. from glob import glob
9. import pandas as pd
10. import numpy as np
11. import argparse
12. import warnings
13. import imutils
14. import shutil
15. import glob
16. import math
17. import time
18. import cv2
19. import os
20.
21.
22. # Application message
23. print("[INFO] starting application...")
24.
25. # Log time
26. print("[INFO] Time stamp at the start of execution ", time.ctime())
27.
28. start_time = time.time()
29.
30.
31. # Clearing console warnings
32. warnings.filterwarnings("ignore")
33.
34. # Initialize aeguments parser
35. ap = argparse.ArgumentParser()
36. ap.add_argument("-i", "--imagefile", required = "True",
37.                 help = "full path to the mhd/raw or dicom 3D CT scans folder")
38. ap.add_argument("-a", "--
39.                 appmode", required = "True", type = str, default = "extended",
39.                 help = "application mode specification: light or extended")
```

```

40. args = vars(ap.parse_args())
41.
42.
43. # Process passed arguments
44. imagefile_ = args["imagefile"]
45. imageformat = imagefile_.split(".")[1]
46. appmode_ = args["appmode"]
47. appmode = appmode_.lower()
48.
49. #print(imagefile_)
50. #print(imageformat)
51. #print(appmode)
52.
53. # Load Ensemble Models (in HFD5 format)
54. print("[INFO] Loading ensemble models...")
55. CNN_3D_i = load_model("model1.h5") # 50-250-128-128
56. CNN_3D_j = load_model("model2.h5") # 50-200-128-128
57. CNN_3D_k = load_model("model3.h5") # 50-200-128-128
58. print("[INFO] ensemble models loaded successfully...")
59.
60.
61. # DICOM handling functions
62. # Load the slices
63. def load_scan(path):
64.
65.     slices = [pydicom.read_file(path + "/" + s) for s in os.listdir(path) if s.endswith
("dcm")]
66.     slices.sort(key = lambda x: int(x.InstanceNumber))
67.
68.     try:
69.         slice_thickness = np.abs(slices[0].ImagePosition[2] - slices[1].ImagePosition[2
])
70.     except:
71.         slice_thickness = np.abs(slices[0].SliceLocation - slices[1].SliceLocation)
72.     for s in slices:
73.         s.SliceThickness = slice_thickness
74.
75.     # Raw slices
76.     return slices
77.
78.
79. # Extract array from dicom slices
80. def get_pixels_hu(scans):
81.     image = np.stack([s.pixel_array for s in scans])
82.     # Converting to int16
83.     image = image.astype(np.int16)
84.
85.     #print(image)
86.     #print(image == -2000)
87.     # Setting outside of the pxel to 1
88.     # Intercept is usually -1024 and air is 0
89.     image[image == -2000] = 0
90.
91.     #print(image)
92.
93.     # Converting to HU
94.     intercept = scans[0].RescaleIntercept
95.     slope = scans[0].RescaleSlope
96.
97.     if slope != 1:
98.         image = slope * image.astype(np.float64)

```

```

99.         image = image.astype(np.int16)
100.
101.     image += np.int16(intercept)
102.     image_ = np.array(image, dtype = np.int16)
103.     image_ = list(image_)
104.     return image_
105.
106.
107. # dicom handler
108. #data_path = "../LIDC-IDRI-0002/01-01-2000-98329/3000522-04919"
109. def dicom_handler():
110.
111.     data_path = args["imagefile"]
112.     path = os.listdir(data_path)
113.     #print(len(path))
114.
115.     patient = load_scan(data_path)
116.     #print(len(patient))
117.     slices3D = get_pixels_hu(patient)
118.
119.     # list holding the 3D array
120.     return slices3D
121.
122.
123. # Image pre-processing into deep neural net format
124. # CNN_3D_i: (1 * 1 * 128 * 128 * 250)
125. # CNN_3D_j: (1 * 1 * 128 * 128 * 200)
126. # CNN_3D_k: (1 * 1 * 128 * 128 * 200)
127.
128. def mhd_3D_i_preprocessor():
129.
130.     #Load raw 3D scan
131.     length_slices_3D = []
132.     roll_axis3D = []
133.     new_slices3D = []
134.     slices_3D = []
135.
136.     image3D = []
137.     for i in range(1):
138.
139.         image = sitk.GetArrayFromImage(sitk.ReadImage(imagefile_))
140.         #print(image.shape)
141.         image3D.append(image)
142.
143.         # Display CT scan: call slice viewer
144.         # Pre-process image to the format (1 * 1 * 128 * 128 * 250)
145.         w = 128
146.         h = 128
147.
148.         # Loop through the 3D array of the CT scan
149.         for i, patients in enumerate(image3D):
150.             #print(patients.shape, i)
151.
152.             new_slices = []
153.
154.             for i, slice in enumerate(patients):
155.                 if (i < 249) or (i == 249):
156.                     img_sm = cv2.resize(np.array(slice), (w, h), interpolation=cv2.INTER_CU
BIC)
157.                     #print("Data type: ", type(img_sm))
158.                     new_slices.append(img_sm)

```

```

159.             #print("shape of image stack = {a} and i = {b}".format(a = img_sm.shape
, b = i))
160.
161.         else:
162.             break
163.
164.         slices_resized_3d = np.asarray(new_slices).astype('int16')
165.         #print("Data type of slices_resized_3d: ", type(slices_resized_3d))
166.         # Length of Main 3D
167.         length_slices_3D.append(len(slices_resized_3d))
168.
169.         # Main 3D array
170.         new_slices3D.append(slices_resized_3d)
171.
172.         # append length of slices after each loop
173.         input = slices_resized_3d
174.         #print("shape before rolling:", np.array(new_slices3D).shape)
175.
176.         # Rolling
177.         ipt = np.rollaxis(np.rollaxis(input, 2, 0), 2, 0)
178.         roll_axis3D.append(ipt)
179.         #print(ipt.shape)
180.         #print("New slice 3D:", np.array(new_slices3D).shape)
181.         #print("New slice 3D rolled:", np.array(roll_axis3D).shape)
182.
183.     return roll_axis3D
184.
185.
186. def mhd_3D_j_preprocessor():
187.
188.     length_slices_3D = []
189.     roll_axis3D = []
190.     new_slices3D = []
191.     slices_3D = []
192.
193.     #Load raw 3D scan
194.     image3D = []
195.     for i in range(1):
196.
197.         image = sitk.GetArrayFromImage(sitk.ReadImage(imagefile_))
198.         #print(image.shape)
199.         image3D.append(image)
200.
201.     # Display CT scan: call slice viewer
202.
203.     # Pre-process image to the format (1 * 1 * 128 * 128 * 250)
204.     w = 128
205.     h = 128
206.
207.
208.     # Loop through the 3D array of the CT scan
209.     for i, patients in enumerate(image3D):
210.         #print(patients.shape, i)
211.
212.         new_slices = []
213.
214.         for i, slice in enumerate(patients):
215.             if (i < 199) or (i == 199):
216.                 img_sm = cv2.resize(np.array(slice), (w, h), interpolation=cv2.INTER_CU
BIC)
217.                 #print("Data type: ", type(img_sm))

```

```

218.             #print("Maximum pixel value: ",np.max(img_sm))
219.             new_slices.append(img_sm)
220.             #print("shape of image stack = {a} and i = {b}".format(a = img_sm.shape
, b = i))
221.
222.             else:
223.                 break
224.
225.             slices_resized_3d = np.asarray(new_slices).astype('int16')
226.             #print("Data type of slices_resized_3d: ", type(slices_resized_3d))
227.             # Length of Main 3D
228.             length_slices_3D.append(len(slices_resized_3d))
229.
230.             # Main 3D array
231.             new_slices3D.append(slices_resized_3d)
232.
233.             # append length of slices after each loop
234.             input = slices_resized_3d
235.             #print("shape before rolling:", np.array(new_slices3D).shape)
236.
237.             # Rolling
238.             ipt = np.rollaxis(np.rollaxis(input, 2, 0), 2, 0)
239.             roll_axis3D.append(ipt)
240.             #print(ipt.shape)
241.             #print("New slice 3D:", np.array(new_slices3D).shape)
242.             #print("New slice 3D rolled:", np.array(roll_axis3D).shape)
243.
244.         return roll_axis3D
245.
246. def mhd_3D_k_preprocessor():
247.
248.     length_slices_3D = []
249.     roll_axis3D = []
250.     new_slices3D = []
251.     slices_3D = []
252.
253.     #Load raw 3D scan
254.     image3D = []
255.     for i in range(1):
256.
257.         image = sitk.GetArrayFromImage(sitk.ReadImage(imagefile_))
258.         #print(image.shape)
259.         image3D.append(image)
260.
261.     # Display CT scan: call slice viewer
262.
263.     # Pre-process image to the format (1 * 1 * 128 * 128 * 250)
264.     w = 128
265.     h = 128
266.
267.
268.     # Loop through the 3D array of the CT scan
269.     for i, patients in enumerate(image3D):
270.         #print(patients.shape, i)
271.
272.         new_slices = []
273.
274.         for i, slice in enumerate(patients):
275.             if (i < 199) or (i == 199):
276.                 img_sm = cv2.resize(np.array(slice), (w, h), interpolation=cv2.INTER_CU
BIC)

```

```

277.             #print("Data type: ", type(img_sm))
278.             #print("Maximum pixel value: ",np.max(img_sm))
279.             new_slices.append(img_sm)
280.             #print("shape of image stack = {a} and i = {b}".format(a = img_sm.shape
, b = i))
281.
282.         else:
283.             break
284.
285.         slices_resized_3d = np.asarray(new_slices).astype('int16')
286.         #print("Data type of slices_resized_3d: ", type(slices_resized_3d))
287.         # Length of Main 3D
288.         length_slices_3D.append(len(slices_resized_3d))
289.
290.         # Main 3D array
291.         new_slices3D.append(slices_resized_3d)
292.
293.         # append length of slices after each loop
294.         input = slices_resized_3d
295.         #print("shape before rolling:", np.array(new_slices3D).shape)
296.
297.         # Rolling
298.         ipt = np.rollaxis(np.rollaxis(input, 2, 0), 2, 0)
299.         roll_axis3D.append(ipt)
300.         #print(ipt.shape)
301.         #print("New slice 3D:", np.array(new_slices3D).shape)
302.         #print("New slice 3D rolled:", np.array(roll_axis3D).shape)
303.
304.
305.     return roll_axis3D
306.
307. def dicom_3D_i_preprocessor():
308.
309.     #Load raw 3D scan
310.     image3D = []
311.     length_slices_3D = []
312.     roll_axis3D = []
313.     new_slices3D = []
314.
315.     image3D = dicom_handler()
316.
317.     # Display CT scan: call slice viewer
318.
319.     # Pre-process image to the format (1 * 1 * 128 * 128 * 250)
320.     w = 128
321.     h = 128
322.
323.     # Loop through the 3D array of the CT scan
324.     for i, patients in enumerate(image3D):
325.         #print(patients.shape, i)
326.
327.         new_slices = []
328.
329.         for i, slice in enumerate(patients):
330.             if (i < 249) or (i == 249):
331.                 img_sm = cv2.resize(np.array(slice), (w, h), interpolation=cv2.INTER_CU
BIC)
332.                 #print("Data type: ", type(img_sm))
333.                 #print("Maximum pixel value: ",np.max(img_sm))
334.                 new_slices.append(img_sm)

```

```

335.             #print("shape of image stack = {a} and i = {b}".format(a = img_sm.shape
, b = i))
336.
337.         else:
338.             break
339.
340.         slices_resized_3d = np.asarray(new_slices).astype('int16')
341.         #print("Data type of slices_resized_3d: ", type(slices_resized_3d))
342.         # Length of Main 3D
343.         length_slices_3D.append(len(slices_resized_3d))
344.
345.         # Main 3D array
346.         new_slices3D.append(slices_resized_3d)
347.
348.         # append length of slices after each loop
349.         input = slices_resized_3d
350.         #print("shape before rolling:", np.array(new_slices3D).shape)
351.
352.         # Rolling
353.         ipt = np.rollaxis(np.rollaxis(input, 2, 0), 2, 0)
354.         roll_axis3D.append(ipt)
355.         #print(ipt.shape)
356.         #print("New slice 3D:", np.array(new_slices3D).shape)
357.         #print("New slice 3D rolled:", np.array(roll_axis3D).shape)
358.
359.     return roll_axis3D
360.
361.
362. def dicom_3D_j_preprocessor():
363.
364.     length_slices_3D = []
365.     roll_axis3D = []
366.     new_slices3D = []
367.     slices_3D = []
368.
369.     #Load raw 3D scan
370.     image3D = dicom_handler()
371.
372.     # Display CT scan: call slice viewer
373.
374.     # Pre-process image to the format (1 * 1 * 128 * 128 * 250)
375.     w = 128
376.     h = 128
377.
378.
379.     # Loop through the 3D array of the CT scan
380.     for i, patients in enumerate(image3D):
381.         #print(patients.shape, i)
382.
383.         new_slices = []
384.
385.         for i, slice in enumerate(patients):
386.             if (i < 199) or (i == 199):
387.                 img_sm = cv2.resize(np.array(slice), (w, h), interpolation=cv2.INTER_CU
BIC)
388.                 #print("Data type: ", type(img_sm))
389.                 #print("Maximum pixel value: ", np.max(img_sm))
390.                 new_slices.append(img_sm)
391.                 #print("shape of image stack = {a} and i = {b}".format(a = img_sm.shape
, b = i))
392.

```

```

393.         else:
394.             break
395.
396.         slices_resized_3d = np.asarray(new_slices).astype('int16')
397.         #print("Data type of slices_resized_3d: ", type(slices_resized_3d))
398.         # Length of Main 3D
399.         length_slices_3D.append(len(slices_resized_3d))
400.
401.         # Main 3D array
402.         new_slices3D.append(slices_resized_3d)
403.
404.         # append length of slices after each loop
405.         input = slices_resized_3d
406.         #print("shape before rolling:", np.array(new_slices3D).shape)
407.
408.         # Rolling
409.         ipt = np.rollaxis(np.rollaxis(input, 2, 0), 2, 0)
410.         roll_axis3D.append(ipt)
411.         #print(ipt.shape)
412.         #print("New slice 3D:", np.array(new_slices3D).shape)
413.         #print("New slice 3D rolled:", np.array(roll_axis3D).shape)
414.
415.     return roll_axis3D
416.
417.
418. def dicom_3D_k_preprocessor():
419.
420.     length_slices_3D = []
421.     roll_axis3D = []
422.     new_slices3D = []
423.     slices_3D = []
424.
425.     #Load raw 3D scan
426.     image3D = dicom_handler()
427.
428.     # Display CT scan: call slice viewer
429.
430.     # Pre-process image to the format (1 * 1 * 128 * 128 * 250)
431.     w = 128
432.     h = 128
433.
434.
435.     # Loop through the 3D array of the CT scan
436.     for i, patients in enumerate(image3D):
437.         #print(patients.shape, i)
438.
439.         new_slices = []
440.
441.         for i, slice in enumerate(patients):
442.             if (i < 199) or (i == 199):
443.                 img_sm = cv2.resize(np.array(slice), (w, h), interpolation=cv2.INTER_CU
BIC)
444.                 #print("Data type: ", type(img_sm))
445.                 #print("Maximum pixel value: ", np.max(img_sm))
446.                 new_slices.append(img_sm)
447.                 #print("shape of image stack = {a} and i = {b}".format(a = img_sm.shape
, b = i))
448.
449.         else:
450.             break
451.

```



```

452.     slices_resized_3d = np.asarray(new_slices).astype('int16')
453.     #print("Data type of slices_resized_3d: ", type(slices_resized_3d))
454.     # Length of Main 3D
455.     length_slices_3D.append(len(slices_resized_3d))
456.
457.     # Main 3D array
458.     new_slices3D.append(slices_resized_3d)
459.
460.     # append length of slices after each loop
461.     input = slices_resized_3d
462.     #print("shape before rolling:", np.array(new_slices3D).shape)
463.
464.     # Rolling
465.     ipt = np.rollaxis(np.rollaxis(input, 2, 0), 2, 0)
466.     roll_axis3D.append(ipt)
467.     #print(ipt.shape)
468.     #print("New slice 3D:", np.array(new_slices3D).shape)
469.     #print("New slice 3D rolled:", np.array(roll_axis3D).shape)
470.
471.
472.     return roll_axis3D
473.
474.
475.# Sub-routines definition
476.# dicom light handler
477.def dicom_light_handler():
478.
479.     # Save arrays processed in separate list : dicom
480.     dicom_3D_i_array = np.array(dicom_3D_i_preprocessor())
481.     dicom_3D_j_array = np.array(dicom_3D_j_preprocessor())
482.     dicom_3D_k_array = np.array(dicom_3D_k_preprocessor())
483.
484.     # Transforming 4D into 5D for CNN Model
485.     input_array_i = np.ones((1, 1, 128, 128, 250))
486.     for h in range(1):
487.         input_array_i[h][0][:][:][:] = dicom_3D_i_array[h,:,:,:]
488.
489.
490.     input_array_j = np.ones((1, 1, 128, 128, 200))
491.     for h in range(1):
492.         input_array_j[h][0][:][:][:] = dicom_3D_j_array[h,:,:,:]
493.
494.
495.     input_array_k = np.ones((1, 1, 128, 128, 200))
496.     for h in range(1):
497.         input_array_k[h][0][:][:][:] = dicom_3D_k_array[h,:,:,:]
498.
499.     # Predict voxel lesion type
500.     CNN_3D_i_predict = CNN_3D_i.predict(input_array_i)
501.     CNN_3D_j_predict = CNN_3D_j.predict(input_array_j)
502.     CNN_3D_k_predict = CNN_3D_k.predict(input_array_k)
503.
504.     # Compute averaging
505.     sigmoid_average = (CNN_3D_i_predict + CNN_3D_j_predict + CNN_3D_k_predict) / 3.0
506.
507.     # Return the three predicted results
508.     return [CNN_3D_i_predict, CNN_3D_j_predict, CNN_3D_k_predict, sigmoid_average]
509.
510.
511.# Sub-routines definition
512.# dicom extended handler

```

```

513. def dicom_extended_handler():
514.
515.     # Save arrays processed in separate list : dicom
516.     dicom_3D_i_array = np.array(dicom_3D_i_preprocessor())
517.     dicom_3D_j_array = np.array(dicom_3D_j_preprocessor())
518.     dicom_3D_k_array = np.array(dicom_3D_k_preprocessor())
519.
520.     # Print the summary of all blocks in the three networks
521.     print(format("The summary of CNN_3D_i architecture", "^80"))
522.     CNN_3D_i.summary()
523.     print(" ")
524.     print(format("The summary of CNN_3D_j architecture", "^80"))
525.     CNN_3D_j.summary()
526.     print(" ")
527.     print(format("The summary of CNN_3D_k architecture", "^80"))
528.     CNN_3D_k.summary()
529.     print(" ")
530.
531.     # Transforming 4D into 5D for CNN Model
532.     input_array_i = np.ones((1, 1, 128, 128, 250))
533.     for h in range(1):
534.         input_array_i[h][0][:][:][:] = dicom_3D_i_array[h,:,:,:]
535.
536.
537.     input_array_j = np.ones((1, 1, 128, 128, 200))
538.     for h in range(1):
539.         input_array_j[h][0][:][:][:] = dicom_3D_j_array[h,:,:,:]
540.
541.
542.     input_array_k = np.ones((1, 1, 128, 128, 200))
543.     for h in range(1):
544.         input_array_k[h][0][:][:][:] = dicom_3D_k_array[h,:,:,:]
545.
546.     # Predict voxel lesion type
547.     CNN_3D_i_predict = CNN_3D_i.predict(input_array_i)
548.     CNN_3D_j_predict = CNN_3D_j.predict(input_array_j)
549.     CNN_3D_k_predict = CNN_3D_k.predict(input_array_k)
550.
551.     # Compute averaging
552.     sigmoid_average = (CNN_3D_i_predict + CNN_3D_j_predict + CNN_3D_k_predict) / 3.0
553.
554.     # Return the three predicted results
555.     return [CNN_3D_i_predict, CNN_3D_j_predict, CNN_3D_k_predict, sigmoid_average]
556.
557.
558. # mhd light handler
559. def mhd_light_handler():
560.
561.     # Save arrays processed in separate list : mhd
562.     mhd_3D_i_array = np.array(mhd_3D_i_preprocessor())
563.     mhd_3D_j_array = np.array(mhd_3D_j_preprocessor())
564.     mhd_3D_k_array = np.array(mhd_3D_k_preprocessor())
565.
566.     # Transforming 4D into 5D for CNN Model
567.     input_array_i = np.ones((1, 1, 128, 128, 250))
568.     for h in range(1):
569.         input_array_i[h][0][:][:][:] = mhd_3D_i_array[h,:,:,:]
570.
571.
572.     input_array_j = np.ones((1, 1, 128, 128, 200))
573.     for h in range(1):

```

```

574.         input_array_j[h][0][:][:] = mhd_3D_j_array[h,:,::]
575.
576.
577.     input_array_k = np.ones((1, 1, 128, 128, 200))
578.     for h in range(1):
579.         input_array_k[h][0][:][:] = mhd_3D_k_array[h,:,::]
580.
581.     # Predict voxel lesion type
582.     CNN_3D_i_predict = CNN_3D_i.predict(input_array_i)
583.     CNN_3D_j_predict = CNN_3D_j.predict(input_array_j)
584.     CNN_3D_k_predict = CNN_3D_k.predict(input_array_k)
585.
586.     # Compute averaging
587.     sigmoid_average = (CNN_3D_i_predict + CNN_3D_j_predict + CNN_3D_k_predict) / 3.0
588.
589.     # Return the three predicted results
590.     return [CNN_3D_i_predict, CNN_3D_j_predict, CNN_3D_k_predict, sigmoid_average]
591.
592.
593. # Sub-routines definition
594. # mhd extended handler
595. def mhd_extended_handler():
596.
597.     # Save arrays processed in separate list : mhd
598.     mhd_3D_i_array = np.array(mhd_3D_i_preprocessor())
599.     mhd_3D_j_array = np.array(mhd_3D_j_preprocessor())
600.     mhd_3D_k_array = np.array(mhd_3D_k_preprocessor())
601.
602.     # Print the summary of all blocks in the three networks
603.     print(format("The summary of CNN_3D_i architecture", "*^80"))
604.     CNN_3D_i.summary()
605.     print(" ")
606.     print(format("The summary of CNN_3D_j architecture", "*^80"))
607.     CNN_3D_j.summary()
608.     print(" ")
609.     print(format("The summary of CNN_3D_k architecture", "*^80"))
610.     CNN_3D_k.summary()
611.
612.
613.     # Transforming 4D into 5D for CNN Model
614.     input_array_i = np.ones((1, 1, 128, 128, 250))
615.     for h in range(1):
616.         input_array_i[h][0][:][:] = mhd_3D_i_array[h,:,::]
617.
618.
619.     input_array_j = np.ones((1, 1, 128, 128, 200))
620.     for h in range(1):
621.         input_array_j[h][0][:][:] = mhd_3D_j_array[h,:,::]
622.
623.
624.     input_array_k = np.ones((1, 1, 128, 128, 200))
625.     for h in range(1):
626.         input_array_k[h][0][:][:] = mhd_3D_k_array[h,:,::]
627.
628.
629.     # Predict voxel lesion type
630.     CNN_3D_i_predict = CNN_3D_i.predict(input_array_i)
631.     CNN_3D_j_predict = CNN_3D_j.predict(input_array_j)
632.     CNN_3D_k_predict = CNN_3D_k.predict(input_array_k)
633.
634.     # Compute averaging

```

```

635.     sigmoid_average = (CNN_3D_i_predict + CNN_3D_j_predict + CNN_3D_k_predict) / 3.0
636.
637.     # Return the three predicted results
638.     return [CNN_3D_i_predict, CNN_3D_j_predict, CNN_3D_k_predict, sigmoid_average]
639.
640.
641. # Base inference engine
642. def base_inference_engine():
643.
644.     print(format("[Predicting Voxel Nodule or Non Nodule in images] Running...", "*^80"
645. ))
646.     # Sub routine controller
647.     if imageformat == "mhd":
648.         if appmode == "light":
649.
650.             mhd_ = mhd_light_handler()
651.             print("3D_CNN_i: %2.2f, 3D_CNN_j: %2.2f, 3D_CNN_k: %2.2f, and Sigmoid avera
652. ging: %2.2f" %(mhd_[0], mhd_[1], mhd_[2], mhd_[3][0][0]))
653.
654.             if(mhd_[3] >= 0.5):
655.                 print("The probability of the patient having [Lung Nodule] is: %2.2f" %
656. (mhd_[3][0][0]))
657.
658.             else:
659.
660.                 print("The probability of the patient having [Non Lung Nodule] is %2.2f
661. : " %(1 - mhd_[3][0][0]))
662.
663.             elif appmode == "extended":
664.
665.                 mhd_ = mhd_extended_handler()
666.                 print("3D_CNN_i predicts: %2.2f, 3D_CNN_j predicts: %2.2f, 3D_CNN_k predict
667. s: %2.2f, and Sigmoid averaging: %5.2f" %(mhd_[0], mhd_[1], mhd_[2], mhd_[3]))
668.
669.                 if(mhd_[3] >= 0.5):
670.
671.                     print("The probability of the patient having [Lung Nodule] is: %2.2f" %
672. (mhd_[3][0][0]))
673.
674.                     else:
675.
676.                         print("The probability of the patient having [Non Lung Nodule] is: %2.2
677. f" %(1 - mhd_[3][0][0]))
678.
679.                 elif imageformat != "mhd":
680.                     if appmode == "light":
681.                         dicom_ = dicom_light_handler()
682.                         print("[Predicted Probabilities] 3D_CNN_i: %2.2f, 3D_CNN_j predicts: %2.2f,
683. 3D_CNN_k: %2.2f, and Sigmoid averaging: %1.2f" %(dicom_[0], dicom_[1], dicom_[2], dico
684. m_[3][0][0]))
685.
686.                         if(dicom_[3] >= 0.5):
687.                             print("The probability of the patient having [Lung Nodule] is: %2.2f" %
688. (dicom_[3][0][0]))
689.
690.                         else:

```

```

686.         print("The probability of the patient having [Non Lung Nodule] is: %2.2
        f" %(1 - dicom_[3][0][0]))
687.
688.
689.
690.         elif appmode == "extended":
691.             dicom_ = dicom_extended_handler()
692.             print("3D_CNN_i: %2.2f, 3D_CNN_j: %2.2f, 3D_CNN_k: %2.2f, and Sigmoid avera
        ging: %2.2f" %(dicom_[0], dicom_[1], dicom_[2], dicom_[3][0][0]))
693.
694.             if(dicom_[3] >= 0.5):
695.                 print("The probability of the patient having [Lung Nodule] is: %2.2f" %
        (dicom_[3][0][0]))
696.
697.             else:
698.
699.                 print("The probability of the patient having [Non Lung Nodule] is: %2.2
        f" %(1 - dicom_[3][0][0]))
700.
701.
702. # Slice viewer: CNN_3D_i
703. def slice_viewer_200_slices(slices_):
704.
705.     i = 0
706.     # Display CT scan
707.     for slice_ in slices_[0:1]:
708.         print(slice_.shape)
709.         for mask in slice_:
710.             plt.imshow(mask, cmap= "gray")
711.             plt.show()
712.             print(i)
713.             i = i + 1
714.             if i == 200:
715.                 break
716.
717. # Slice viewer: CNN_3D_j and CNN_3D_k
718. def slice_viewer_250_slices(slices_):
719.     i = 0
720.     # Display CT scan
721.     for slice_ in slices_[0:1]:
722.         print(slice_.shape)
723.         for mask in slice_:
724.             plt.imshow(mask, cmap= "gray")
725.             plt.show()
726.             print(i)
727.             i = i + 1
728.
729.             if i == 250:
730.                 break
731.
732. # Invoke the base inference handler
733. base_inference_engine()
734.
735. # Log time
736. end_time = time.time()
737.
738. Print("[INFO] Timestamp at the end of execution: ", time.ctime())
739.
740.
741. log_diff = end_time - start_time
742. Print("[INFO] The total time taken to perform the inference is: %5.2f" %(log_diff))

```

```
743.  
744.print("[INFO] Exiting application")
```

References

- [1] M. Plummer, C. de Martel, J. Vignat, J. Ferlay, F. Bray, and S. Franceschi, 'Global burden of cancers attributable to infections in 2012: a synthetic analysis', *Lancet Glob. Heal.*, vol. 4, no. 9, pp. e609–e616, Sep. 2016.
- [2] C. I. Henschke *et al.*, 'Early Lung Cancer Action Project: A Summary of the Findings on Baseline Screening', *Oncologist*, vol. 6, no. 2, pp. 147–152, Apr. 2001.
- [3] 'Cancer survival rate: A tool to understand your prognosis - Mayo Clinic'. [Online]. Available: <https://www.mayoclinic.org/diseases-conditions/cancer/in-depth/cancer/art-20044517>. [Accessed: 07-Apr-2020].
- [4] G. Zhang *et al.*, 'Automatic nodule detection for lung cancer in CT images: A review', *Computers in Biology and Medicine*, vol. 103. Elsevier Ltd, pp. 287–300, 01-Dec-2018.
- [5] E. Guresen and G. Kayakutlu, 'Definition of Artificial Neural Networks with comparison to other networks', in *Procedia Computer Science*, 2011, vol. 3, pp. 426–433.
- [6] N. O'Mahony *et al.*, 'Deep Learning vs. Traditional Computer Vision', in *Advances in Intelligent Systems and Computing*, 2020, vol. 943, pp. 128–144.
- [7] F. Iandola, 'Exploring the Design Space of Deep Convolutional Neural Networks at Large Scale', no. April, 2016.
- [8] 'Impact of Dataset Size on Deep Learning Model Skill And Performance Estimates'. [Online]. Available: <https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/>. [Accessed: 07-Apr-2020].
- [9] A. Gupta, T. Saar, O. Martens, and Y. Le Moullec, 'Automatic detection of multisize pulmonary nodules in CT images: Large-scale validation of the false-positive reduction step: Large-scale', *Med. Phys.*, vol. 45, no. 3, pp. 1135–1149, Mar. 2018.
- [10] T. Manikandan and N. Bharathi, 'Lung Cancer Detection Using Fuzzy Auto-Seed Cluster Means Morphological Segmentation and SVM Classifier', *J. Med. Syst.*, vol. 40, no. 7, Jul. 2016.
- [11] F. Shaukat, G. Raja, A. Gooya, and A. F. Frangi, 'Fully automatic detection of lung nodules in CT images using a hybrid feature set', *Med. Phys.*, vol. 44, no. 7, pp. 3615–3629, Jul. 2017.
- [12] S. M. Naqi, M. Sharif, and M. Yasmin, 'Multistage segmentation model and SVM-ensemble for precise lung nodule detection', *Int. J. Comput. Assist. Radiol. Surg.*, vol. 13, no. 7, pp. 1083–1095, Jul. 2018.
- [13] Q. Dou, H. Chen, L. Yu, J. Qin, and P. A. Heng, 'Multilevel Contextual 3-D CNNs for False

- Positive Reduction in Pulmonary Nodule Detection’, *IEEE Trans. Biomed. Eng.*, vol. 64, no. 7, pp. 1558–1567, Jul. 2017.
- [14] G. Cui *et al.*, ‘Automatic lung nodule detection using multi-scale dot nodule-enhancement filter and weighted support vector machines in chest computed tomography’, *PLoS One*, vol. 14, no. 1, Jan. 2019.
- [15] I. R. S. Valente, P. C. Cortez, E. C. Neto, J. M. Soares, V. H. C. de Albuquerque, and J. M. R. S. Tavares, ‘Automatic 3D pulmonary nodule detection in CT images: A survey’, *Computer Methods and Programs in Biomedicine*, vol. 124. Elsevier Ireland Ltd, pp. 91–107, 01-Feb-2016.
- [16] ‘Artificial Intelligence vs Machine Learning vs Deep Learning vs Data Science – mc.ai’. [Online]. Available: <https://mc.ai/artificial-intelligence-vs-machine-learning-vs-deep-learning-vs-data-science/>. [Accessed: 07-Apr-2020].
- [17] ‘Single-layer Neural Networks (Perceptrons)’. [Online]. Available: <https://www.computing.dcu.ie/~humphrys/Notes/Neural/single.neural.html>. [Accessed: 07-Apr-2020].
- [18] ‘Multilayer Perceptron — DeepLearning 0.1 documentation’. [Online]. Available: <http://deeplearning.net/tutorial/mlp.html>. [Accessed: 07-Apr-2020].
- [19] ‘Back Propagation, the Easy Way (Part 2) - Towards Data Science’. [Online]. Available: <https://towardsdatascience.com/back-propagation-the-easy-way-part-2-bea37046c897>. [Accessed: 07-Apr-2020].
- [20] ‘What are the types of machine learning? - Towards Data Science’. [Online]. Available: <https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f>. [Accessed: 07-Apr-2020].
- [21] G. Philipp, D. Song, and J. G. Carbonell, ‘The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions’, Dec. 2017.
- [22] S. Hochreiter, ‘The vanishing gradient problem during learning recurrent neural nets and problem solutions’, *Int. J. Uncertainty, Fuzziness Knowledge-Based Syst.*, vol. 6, no. 2, pp. 107–116, 1998.
- [23] D. Danciu, ‘Dynamics of neural networks as nonlinear systems with several equilibria’, in *Advancing Artificial Intelligence Through Biological Process Applications*, IGI Global, 2008, pp. 331–357.
- [24] M. Z. Alom *et al.*, ‘A state-of-the-art survey on deep learning theory and architectures’, *Electron.*, vol. 8, no. 3, pp. 1–67, 2019.
- [25] ‘Medical Imaging Modalities. | Download Scientific Diagram’. [Online]. Available: https://www.researchgate.net/figure/Medical-Imaging-Modalities_fig1_337399287. [Accessed: 07-Apr-2020].
- [26] ‘The network architecture of slice-level 2D CNN. | Download Scientific Diagram’. [Online]. Available: https://www.researchgate.net/figure/The-network-architecture-of-slice-level-2D-CNN_fig1_315133236. [Accessed: 07-Apr-2020].
- [27] G. Kang, K. Liu, B. Hou, and N. Zhang, ‘3D multi-view convolutional neural networks for lung nodule classification’, *PLoS One*, vol. 12, no. 11, p. e0188290, Nov. 2017.

- [28] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, ‘Snapshot ensembles: Train 1, get M for free’, in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2019.
- [29] A. Myronenko, ‘3D MRI brain tumor segmentation using autoencoder regularization’.
- [30] ‘Deep Belief Networks — An Introduction - Analytics Army - Medium’. [Online]. Available: <https://medium.com/analytics-army/deep-belief-networks-an-introduction-1d52bb867a25>. [Accessed: 07-Apr-2020].
- [31] S. G. Armato *et al.*, ‘The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): A completed reference database of lung nodules on CT scans’, *Med. Phys.*, vol. 38, no. 2, pp. 915–931, 2011.
- [32] ‘JSRT Database | Japanese Society of Radiological Technology’. [Online]. Available: <http://db.jsrt.or.jp/eng.php>. [Accessed: 07-Apr-2020].
- [33] A. A. A. Setio *et al.*, ‘Validation, comparison, and combination of algorithms for automatic detection of pulmonary nodules in computed tomography images: The LUNA16 challenge’, *Med. Image Anal.*, vol. 42, pp. 1–13, Dec. 2017.
- [34] J. Pedrosa *et al.*, ‘LNDb: A Lung Nodule Database on Computed Tomography’, Nov. 2019.
- [35] B. Wang *et al.*, ‘Pulmonary nodule detection in CT images based on shape constraint CV model’, *Med. Phys.*, vol. 42, no. 3, pp. 1241–1254, Feb. 2015.
- [36] A. Teramoto, H. Fujita, O. Yamamuro, and T. Tamaki, ‘Automated detection of pulmonary nodules in PET/CT images: Ensemble false-positive reduction using a convolutional neural network technique’, *Med. Phys.*, vol. 43, no. 6Part1, pp. 2821–2827, May 2016.
- [37] L. Lu, Y. Tan, L. H. Schwartz, and B. Zhao, ‘Hybrid detection of lung nodules on CT scan images’, *Med. Phys.*, vol. 42, no. 9, pp. 5042–5054, Aug. 2015.
- [38] B. Chen *et al.*, ‘Automatic segmentation of pulmonary blood vessels and nodules based on local intensity structure analysis and surface propagation in 3D chest CT images’, *Int. J. Comput. Assist. Radiol. Surg.*, vol. 7, no. 3, pp. 465–482, Jul. 2012.
- [39] M. Javaid, M. Javid, M. Z. U. Rehman, and S. I. A. Shah, ‘A novel approach to CAD system for the detection of lung nodules in CT images’, *Comput. Methods Programs Biomed.*, vol. 135, pp. 125–139, Oct. 2016.
- [40] T. Messay, R. C. Hardie, and T. R. Tuinstra, ‘Segmentation of pulmonary nodules in computed tomography using a regression neural network approach and its application to the Lung Image Database Consortium and Image Database Resource Initiative dataset’, *Med. Image Anal.*, vol. 22, no. 1, pp. 48–62, May 2015.
- [41] S. M. Naqi, M. Sharif, and M. Yasmin, ‘Multistage segmentation model and SVM-ensemble for precise lung nodule detection’, *Int. J. Comput. Assist. Radiol. Surg.*, vol. 13, no. 7, pp. 1083–1095, Jul. 2018.
- [42] B. Zhao, G. Gamsu, M. S. Ginsberg, L. Jiang, and L. H. Schwartz, ‘Automatic detection of small lung nodules on CT utilizing a local density maximum algorithm.’, *J. Appl. Clin. Med. Phys.*, vol. 4, no. 3, pp. 248–260, 2003.

- [43] B. Li *et al.*, ‘Segmentation of ground glass opacity pulmonary nodules using an integrated active contour model with wavelet energy-based adaptive local energy and posterior probability-based speed function’, *Mater. Express*, vol. 6, no. 4, pp. 317–327, Aug. 2016.
- [44] F. Shaukat, G. Raja, A. Gooya, and A. F. Frangi, ‘Fully automatic detection of lung nodules in CT images using a hybrid feature set’, *Med. Phys.*, vol. 44, no. 7, pp. 3615–3629, Jul. 2017.
- [45] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, ‘RUSBoost: A Hybrid Approach to Alleviating Class Imbalance’, *Syst. HUMANS*, vol. 40, no. 1, 2010.
- [46] I. Ali *et al.*, ‘Lung Nodule Detection via Deep Reinforcement Learning’, *Front. Oncol.*, vol. 8, no. APR, p. 108, Apr. 2018.
- [47] Q. Dou, H. Chen, L. Yu, J. Qin, and P. A. Heng, ‘Multilevel Contextual 3-D CNNs for False Positive Reduction in Pulmonary Nodule Detection’, *IEEE Trans. Biomed. Eng.*, vol. 64, no. 7, pp. 1558–1567, Jul. 2017.
- [48] S. Akram, M. Y. Javed, A. Hussain, F. Riaz, and M. U. Akram, ‘Intensity-based statistical features for classification of lungs CT scan nodules using artificial intelligence techniques’, *J. Exp. Theor. Artif. Intell.*, vol. 27, no. 6, pp. 737–751, Nov. 2015.
- [49] A. K. Dhara, S. Mukhopadhyay, A. Dutta, M. Garg, and N. Khandelwal, ‘A Combination of Shape and Texture Features for Classification of Pulmonary Nodules in Lung CT Images’, *J. Digit. Imaging*, vol. 29, no. 4, pp. 466–475, Aug. 2016.
- [50] S. Wang *et al.*, ‘Central focused convolutional neural networks: Developing a data-driven model for lung nodule segmentation’, *Med. Image Anal.*, vol. 40, pp. 172–183, Aug. 2017.
- [51] G. Zhang *et al.*, ‘An Appraisal of Nodule Diagnosis for Lung Cancer in CT Images’, *Journal of Medical Systems*, vol. 43, no. 7. Springer New York LLC, p. 181, 01-Jul-2019.
- [52] J. kui Liu *et al.*, ‘An Assisted Diagnosis System for Detection of Early Pulmonary Nodule in Computed Tomography Images’, *J. Med. Syst.*, vol. 41, no. 2, p. 30, Feb. 2017.
- [53] W. Zhang, X. Wang, X. Li, and J. Chen, ‘3D skeletonization feature based computer-aided detection system for pulmonary nodules in CT datasets’, *Comput. Biol. Med.*, vol. 92, pp. 64–72, Jan. 2018.