# Machine Learning Analysis of Radiology Scans For Automated Decision Support in Lung Cancer Detection

Anirudh Myakala, Vamshidar Reddy, Pragnakar Pedapenki, Chaitanya Krishna Sai Kosaraju

***Abstract:*** **In the United States, lung cancer strikes 225,000 people every year, and accounts for $12 billion in health care costs. Early detection is critical to give patients the best chance at recovery and survival. The image assessments in use today are identifying lung lesions[1] as potentially cancerous that later turn out to not to be cancer leading to unnecessary patient anxiety, additional follow-up imaging and interventional treatments. Using a data set of thousands of lung scans provided by the National Cancer Institute, we developed algorithms using Convolute Neural Networks(CNN)[2] that accurately determine when lesions in the lungs are cancerous. This will dramatically reduce the false positive rate that plagues the current detection technology, get patients earlier access to life-saving interventions, and give radiologists[3] more time to spend with their patients. The lung scans consist of over a thousand low-dose CT images[4] from high-risk patients in DICOM format[5] with certain critical metadata included in it. Mango (Multi-image Analysis GUI) was used initially to understand the lung system and to identify area of interest (cancer region) of each patient. Later code was developed in python to preprocess the whole data and to the generate the 3D images of area of interest for all the patients. ARGO cluster at GMU was used for 3D image generation since it required huge computation power. Later these images were pushed into GPU enabled machine learning(ML) algorithms developed in tensor flow and theano (on ARGO Cluster) for building the model. At the end both models were compared in assessing the probability of a new lung CT scan being cancerous.**

*Index Terms* – Machine Learning, Epochs, Super Computing, Parallel Processing, Deep Learning.

## I. INTRODUCTION

This paper is based on Kaggle' s competition - Data Science Bowl 2017.First it describes various aspects of convolute Neural networks(CNN) and their difference from normal neural networks. Next, it mentions different methodologies to the study lung CT scans and basic preprocessing that must be done before it can be fed into a machine learning model. Finally, it compares results (accuracy) from different machine learning (ML) models and different preprocessing methods.

## II. APPROACH

Since data consists of lung scans in DICOM format, we are required to have knowledge on various biological aspects of lung along with techniques to deal with such format. So, we took some references from Kaggle participants who are experts in medical field for proceeding with our work. Initially we built a 3D CNN model using TensorFlow based on the work references from sentdex [7]. He performed his analysis only on a specific dimension of 50X50X20 and only for 10 epochs. Also, his code was also dimension specific i.e. runs only for the 50x50x20 pixel images. Later we modified the code at several other stages to achieve dimension independency (i.e. can run for any resolution) and optimized to achieve parallel processing on ARGO for full resolution image feed. And we also increased the no. of epochs from 10 to 500 to achieve better accuracy. Next, we extended the work of Guido Zuidhof [8] by building a 2D CNN using theano who just stopped with only preprocessing of data without building any analytical model. In this model, we used image projections from different preprocessing methods to feed CNN. Here the size of the data was reduced from 140GB to

[1] A region in an organ that has suffered damage
[2] Used Tensor flow and Theano libraries to build CNN
[3] Who specialize in diagnosing and treating diseases and injuries using medical imaging techniques
[4] Computer-processed combinations of many X-ray images taken from different angles
[5] Digital Imaging and Communications in Medicine (**DICOM**) is a standard for storing, and handling information in medical imaging.

300MB because of taking 2d projections instead of 3D image and thus making a cost-effective model. It was also run for 500 epochs.

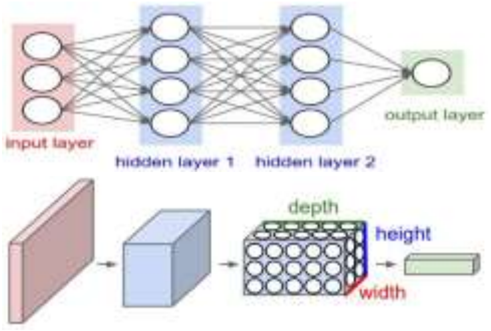
## III. BACKGROUND

### Convolutional Neural Networks (CNNs/ConvNets):

Convolutional Neural networks are similar to ordinary neural networks which contains neurons that can adapt and change weights and biases. Each neuron receives input, performs a dot product and optionally follows it with a non-linearity. The whole network will give a single score that can be differentiable. The network will have a loss function (softmax) on fully connected layer. Main difference between CNNs and ordinary neural network is that ConvNet explicitly assumes inputs as images, which allows us to manipulate certain properties of architecture. This will make feed forward function more efficient and reduces the parameters in network.

### Architecture Overview:

ConvNets take advantage of the fact that input consists of images unlike the ordinary neural networks where each neuron is fully connected to all the neurons in previous layer and completely independent among same layer. The layers of Convnet have neurons arranged in 3 dimensions which transforms the 3D input volume to 3D output volume of neuron activations [1].



Top: Original Neural Network
Bottom: Convolutional Neural Network

**Figure 1: Architecture. Source [1]**

### Layers used to build ConvNets:

A simple convnet will have a Convolutional Layer, Pooling Layer, and Fully-Connected Layer. Each layer transforms one volume of activations to another through a differentiable non-linear function. A simple ConvNet will have the following architecture:

[INPUT --- CONV --- RELU --- POO L --- FC]

- INPUT will have raw pixels of the images
- CONV layer will compute output of the dot product of input pixel matrix and feature or filter matrix
- RELU layer applies elementwise activation function, such as max (0, x) thresholding at zero. This layer basically replaces all negative values of matrix to zero. This doesn't affect the size of volume.
- POOL layer will downsamples the size of volume i.e. it takes max value of feature matrix and reduces dimensions of matrix.
- FC (Fully-Connected) layer will calculate the class scores. In our model, it gives two outputs (1x1x2), one for cancer another for non-cancer.

By passing all image matrices through these layers, ConvNet will transform original pixel values to the class scores. Some layers will have tuning parameters like weights and biases while others don't have them. CONV & FC layers will have weights and biases parameters while RELU/POOL layers will have fixed functions. In summary, Convolutional Neural Network is a list of layers that transform image volume into an output volume.

### The Convolution Operation:

Convolutional layer is the most important in ConvNet and performs most of the computational heavy lifting.

A filter slides over the input image and creates feature map. Different filters produce different feature maps of same original input image. CNN learns values of these filters on its own during training process. As we use more filters, more feature images will be generated and our network becomes better in recognizing new images.

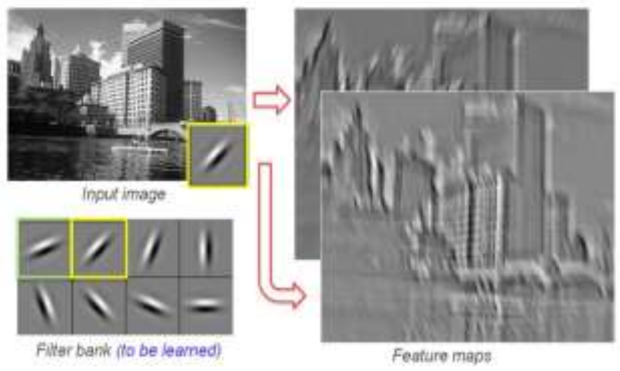


**Figure 2: Generating Feature Maps. Source [4]**

The size of feature map depends on three parameters [2]:

- **Depth:** Depth corresponds to number of filters used for convolution operation. If we use three distinct filters we get three different feature maps.
- **Stride:** Stride is the number of pixels by which we slide our filter matrix over input matrix. When stride is 1, we skip 1 pixel at a time. When stride is 2, we skip 2 pixels at a time. As we increase number we smaller feature maps, which also results in loss of information from input image.
- **Padding:** Padding is done by adding zero pixels on the boundaries of input image matrix. This is done so that filter can be applied to the border elements of input image and to retain feature map size same as input image. This has been clearly explained in [3].

**Rectified Linear Unit (ReLU):**

ReLU is a non-linear operation. Output is given by:

**Output = Max (zero, Input)**

Since most of the real-world data or problems will be non-linear, we use ReLU in our ConvNet. ReLU replaces all negative pixel values in feature map by zero. ReLU operation when applied on the feature map generated in Figure 2 above is shown below in Figure 3.

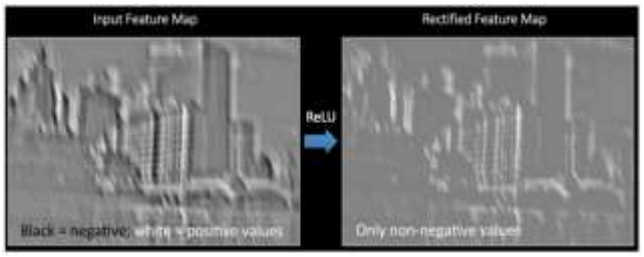


**Figure 3: ReLU Operation. Source [5]**

**The Pooling Step:**

Pooling reduces dimensionality of feature map which retains most of the important information. There are three types of pooling: Max, Average, Sum etc. We have used only Max pooling in our project which takes element from the window matrix which we specified.

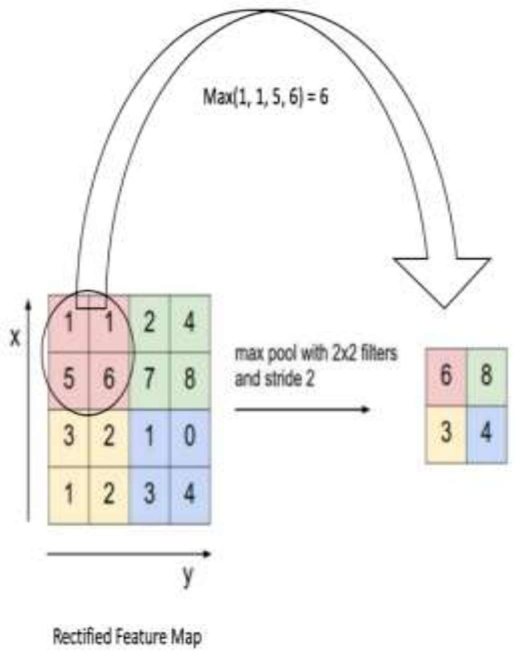


**Figure 4: Max Pooling. Source [1]**

**Fully Connected Layer:**

Fully connected layer is a multi-layer perception that uses a SoftMax activation function in output layer. In fully connected layer each neuron in a layer is fully connected with all neurons in the previous layer. This layer uses features from previous layers and classify input images into different classes. The sum of output probabilities from fully connected layer is 1. This is done using SoftMax function.

The predicted output from this layer is compared with the actual output and calculated error or log loss. Back propagation is used to calculate the gradients of error with respect to all weights and use gradient descent to update weights to minimize the error [1]

## IV. DATA

The dataset consisting of over a thousand low-dose CT images from high-risk patients in DICOM format was taken from Kaggle. Each image contains a series with multiple axial slices of the chest cavity. Each image has a variable number of 2D slices, which can vary based on the machine taking the scan and patient. The DICOM files have a header that contains the necessary information about the patient id, as well as scan parameters such as the slice thickness.

Here is a brief overview of data

- Sample data (Approx. 1.5 GB)
  - 20 patient records.
- Master copy (Approx. 140 GB)
  - 1596 patients record
  - 1398 training data set
  - 198 trial data set

## Metadata:

The CT scan of each patient comprised of around 200 images(slices) with meta data included in each slice. From the metadata included in the DICOM file of single slice the following features were important

- ✓ Patient ID
- ✓ Patient’s Birth date
- ✓ Pixel Data
- ✓ Image Position, Pixel Spacing
- ✓ Rescale Intercept, Rescale slope

The extracted features were primarily used during the preprocessing of the data. But the Patients birth data has been masked to protect the identity of the patient.

**Exploratory Analysis:**

After features were extracted were performed some exploratory analysis.

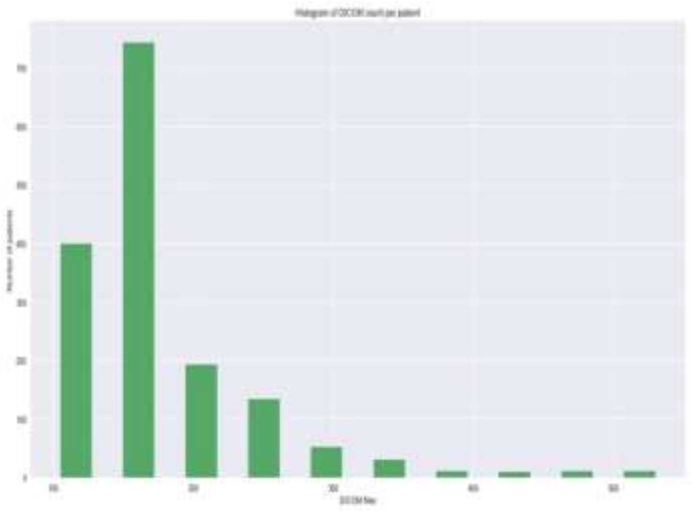
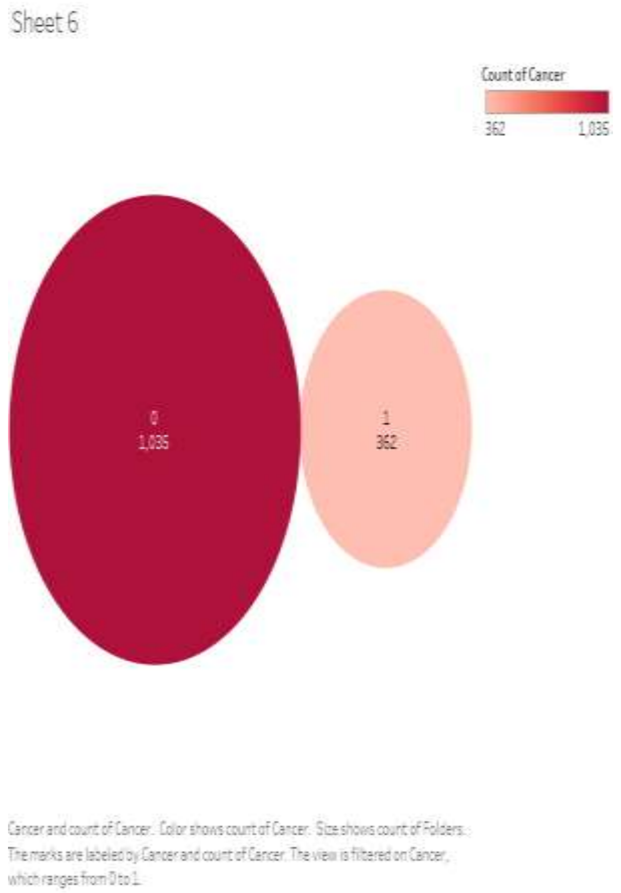


**Figure 5: No. of Patients VS No. of 2d slices in their 3D CT scan**

Histogram shows us the different CT scans (3D image) have different no. of 2D slices. Although majority of patients (around 700) CT scans have no. of slices in between 100-200 some patients had no. of slices up to 500. [6]



The figure illustrates the no. of patients in the dataset who have cancer and who don't. There are 1035 patients without cancer (0) and only 362 patients with cancer (1).

Since lot of meta data was masked to protect the patient identity we could not produce lot of visualizations. But from the above plots we can conclude that the data is not uniform and requires a lot of preprocessing. And to handle the data of this size(~140GB) we need huge computation power and for which ARGO cluster was used.

## V. DATA PREPROCESSING

We know there are lot of noises and the data is not uniform. Because each was patient scanned on different machine there were different no. of slices of CT scan for different patients. Though the image resolution was same for all slices and all patients at (512X512) the pixel distance was different. A scan is having a pixel spacing of [2.5, 0.5, 0.5], which means that the distance between slices is 2.5 millimeters and for another scan it is [1.5, 0.725, 0.725].

Also, since we are building two different models we require two different preprocessing steps to be performed before feeding the data into the model.

1. Type A Preprocessing
2. Type B Preprocessing

### 1. Type A Preprocessing:

This method is adapted from the work of sentdex (participant of Kaggle) where he implemented for sample dataset but we did it for whole dataset.

Type A preprocessing involves two steps

(a) Making the no. of slices of all patient to a constant number (20)

Using python, the total slices were first integrated to a single quantity and then broken down to 20 slices. [7]

| Initial image Pixel size & No. of slices | Images after resizing & Normalizing |
|---|---|
| ▶ (512, 512) 134 | ▶ (X, X) 20 |
| ▶ (512, 512) 128 | ▶ (X, X) 20 |
| ▶ (512, 512) 133 | ▶ (X, X) 20 |
| ▶ (512, 512) 550 | ▶ (X, X) 20 |

Where X=50,150,512 depending on the model

(b) Change the colored figure to greyscale to skip the complexities with colors during model building

The flow of both the process is shown below where 512x512xR is converted into 50x50x20 and then changed to greyscale.

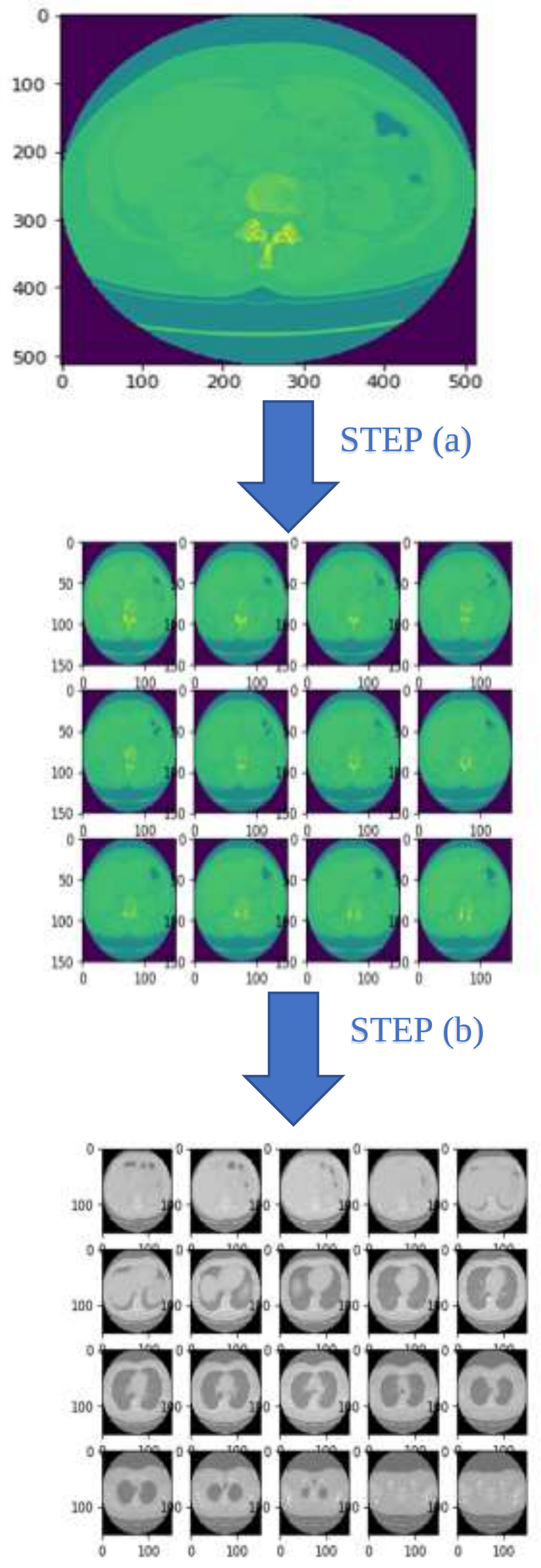


**Figure 7: Type A Preprocessing**
Process illustrates for one patient with X=150. Similarly, 1395 images of all patients were generated on ARGO cluster for different values of X

## 2. Type B Preprocessing

This method is adapted from the work of Guido Zuidhof (participant of Kaggle) where he just implemented it for a sample dataset without even building a model. Whereas we implemented it for whole dataset and built a 2D CNN model.

Type B preprocessing can also be called as masking. It is the process to extract the finest potential nodular region that can be feeded to the model for better accuracy. To perform masking the following procedure is followed

**Lung Identification:**

Hounsfield Unit (HU) is a measure of radio density. Different parts in body have different HU values and based on its value any part can be identified. So, the best way to identify the lung would be by converting the pixel values in scans to HU unit using the below formula.

HU=Pixel Value * Slope +Intercept

Where Rescale slope and Rescale Intercept are attributes from the from meta data.

**Lung Segmentation:**

After identifying the lung, the next thing would be to segment the lung to find the region of fine granular nodules inside the lung where the probability of finding the lesions which are cancerous is very high. This is achieved in 4 steps

IMAGE A-->IMAGE B-->IMAGE C--> IMAGE D

Using the HU value of lung (-500) and soft tissues between 100 to 300 we extract nodular region. [8]

IMAGE A: HU=400. Extracts the CT scan which will yield the bones and all the tougher parts

IMAGE B:( CT SCAN – IMAGE A). Here we can see that all the non-bony part has been extracted from the CT scan.

IMAGE C: HU = -320.Extracts the soft tissues.

IMAGE D: IMAGE B – IMAGE Removes soft tissues from non-bony part which results in the fine nodular region.

**PATIENT 1**

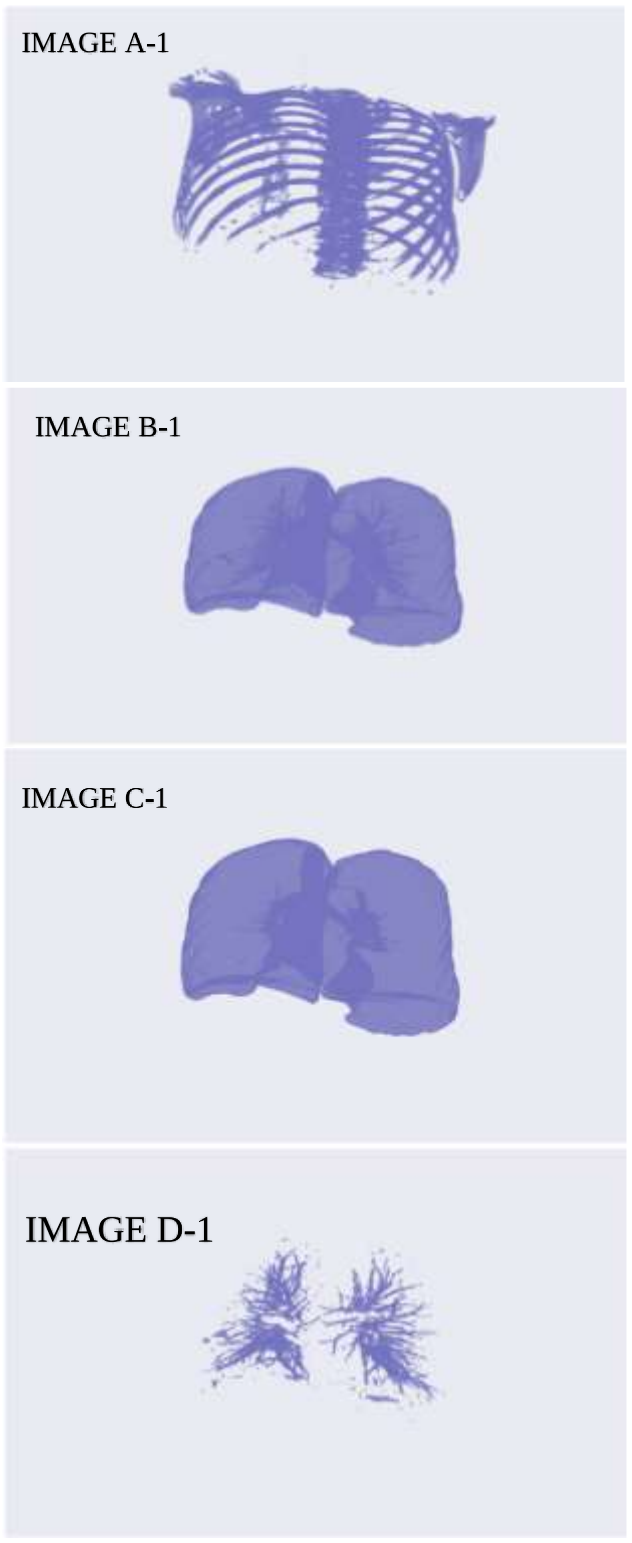


**Figure 8: Type B Preprocessing**
Process illustrates for a patient.

**PATIENT 2**

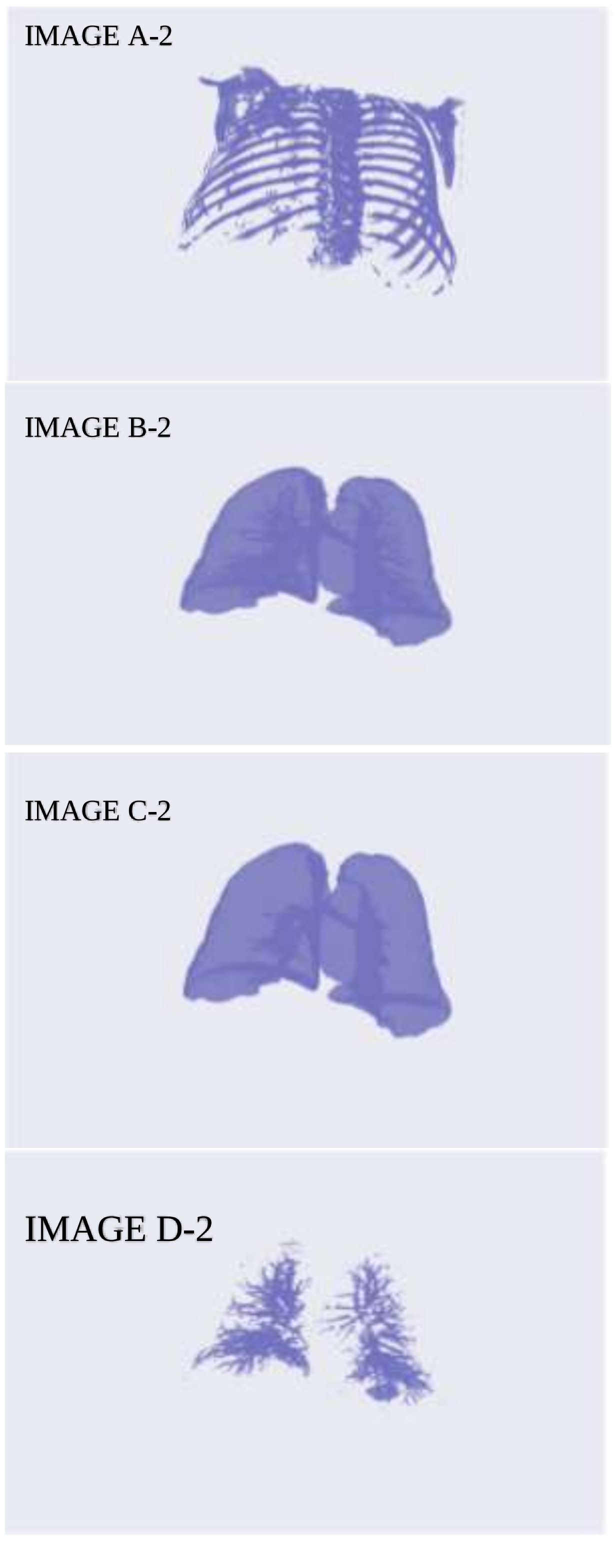


**Figure 9: Type B Preprocessing**
Process illustrates for a different patient.
Similarly, nodular regions of 1395 patients were generated on ARGO cluster

## VI. MODELS

In our analysis, we used Convolutional neural networks (CNNs) in 3D and 2D on two modules of python namely

1. TensorFlow
2. Theano

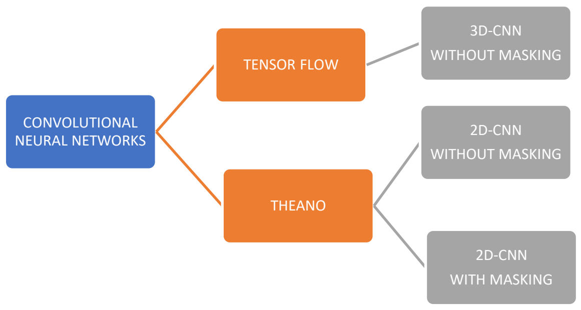


**Figure 10: CNN Models**

-Without Masking refers to building model from output of Type A Preprocessing

- Masking refers to building model from output of Type B Preprocessing

### 1. TENSOR FLOW:

Libraries like Tensor flow and Theano are used for deep learning. They are much like Numpy[6] i.e. number crunching libraries. As we already knew that we will be dealing with large arrays and complex matrix computations, we have decided to use these libraries which utilizes full capacity of ARGO cluster by distributing processing across CPU cores and GPU cores. We used 24 cores available on one node on ARGO cluster by creating threads by using intra_op_parallelism_threads[7] function.

In implementing this code, we took references from sentdex [7] of Kaggle. However, our code was most optimal and we performed analysis on different pixels to check the accuracies where as he performed only on one i.e. 50x50x20.

Images obtained from Type A Preprocessing were converted into a 3D numpy array and were pushed into 3D-CNN with three different resolutions i.e. 50x50x20, 150x150x20, 512x512x20.

Convolution Networks works by moving a filter (which can be made specific or can be random) across the input image that we feed into network. For a 2D image, the filter will be a two-dimensional filter (length x breadth) and a three-dimensional filter for a 3D array or image. The filter pixels can be manually edited if we have some domain knowledge like shape of cancerous lesions[1] which can improve performance of network. We have given a random filter for the model to learn itself from training data.

The following flow chart shows how data flows in a Convolutional Neural Network that we implemented for 50x50x20 resolution.

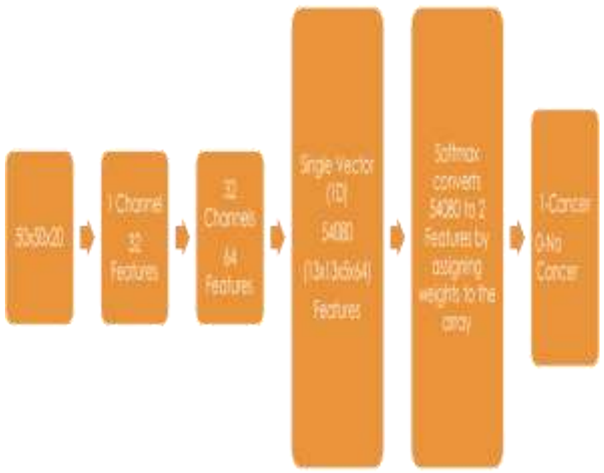

50x50x20
1 Channel 32 Features
32 Channels 64 Features
Single Vector (1D) 54080 (13x13x5x64) Features
Softmax converts 54080 to 2 Features by assigning weights to the array
1-Cancer 0-No Cancer

**Figure 11: Flow in CNN**

Input image is processed into first convolutional layer using filter weights (3x3x3 pixels). This

[6] Fundamental package for scientific computing with Python

[7] This function creates threads which can be used for multi thread processing of data.

gives us an output of 32 images one for each filter in conv1. These 32 images will have a half resolution as input images as we have used strides of 2x2x2 which slides filter by skipping 2 pixels. As we have passed 50x50x20 images for first run, now these output images from conv1 will have resolution of 25x25x10. [9]

These 32 images are then processed into second convolutional layer (conv2). There are 64 output channels so there are total of 32x64 = 2048 filters in conv2. As there are 2048 filters, there will be 2048 output images from conv2 which will have a resolution of 13x13x5.

These pixel arrays are then flattened to a single vector of length 13x13x5x64 = 54080, which is used as input for fully connected layer with 1024 neurons.

Convolutional filters are randomly chosen at first. The error between predicted and true class of input image is measured as cross-entropy. Optimizer automatically propagates this error back through network and updates filter weights which improves classification error.

Similar process is followed for the images with the 150X150 and 512X512 with 10 Epochs[8]

**Results (Whole dataset):**

| EPOCHS | 50x50 | 150x150 | 512x512 |
|---|---|---|---|
| EPOCH 1 | 0.65 | 0.59 | 0.58 |
| EPOCH 2 | 0.55 | 0.60 | 0.52 |
| EPOCH 3 | 0.58 | 0.63 | 0.63 |
| EPOCH 4 | 0.69 | 0.55 | 0.59 |
| EPOCH 5 | 0.50 | 0.63 | 0.66 |
| EPOCH 6 | 0.60 | 0.62 | 0.65 |
| EPOCH 7 | 0.63 | 0.64 | 0.64 |
| EPOCH 8 | 0.68 | 0.54 | 0.61 |
| EPOCH 9 | 0.50 | 0.61 | 0.57 |
| EPOCH 10 | 0.55 | 0.62 | 0.65 |
|  | 0.59 | 0.60 | 0.62 |

As we can observe in the graph below that as we increase number of epochs the model is decreasing the classification error and the accuracy is improving. Also, increasing the pixel resolution there is increase in the Accuracy as we are not losing data.

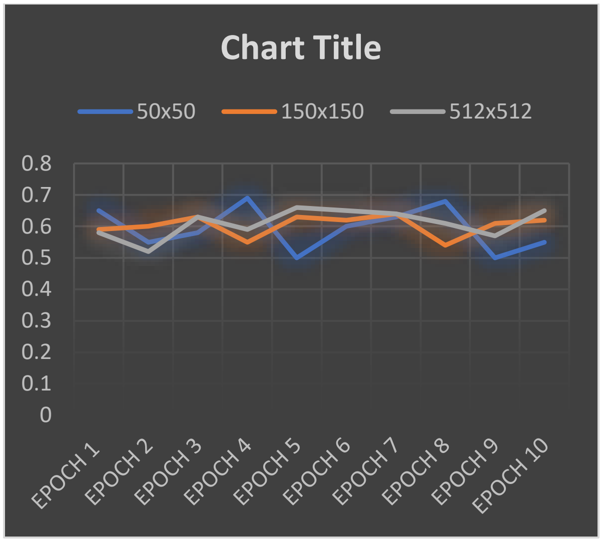


**Figure 12: Chart of Results from TensorFlow**

So, we can improve this accuracy by doing two things, one is running more number of epochs and the second is increasing size of dataset.

However, we achieved higher accuracy than sentdex [7] of Kaggle by running the model with the highest pixel resolution of 512X512 which he could not because of lack of computation.

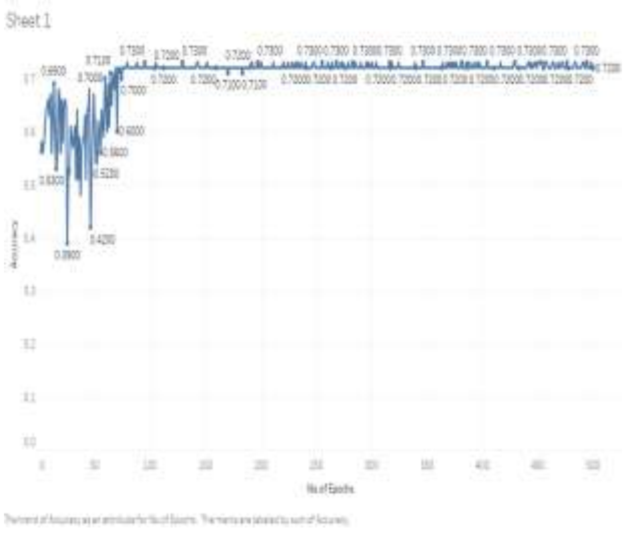


**Figure 13: 500 Epochs for 50x50x20 Accuracy: 0.73**

[8] One Epoch is a single pass through the entire training set, followed by testing of the verification set

**2.THEANO:**

On Theano we implemented two-dimensional convolution neural network, by learning a course on deep learning by JEREMY HOWARD (president of kaggle) and using techniques taught by him on image classification model to classify dogs and cat images. [10]

The following figure is the software technology stack of technology we used for this section of project

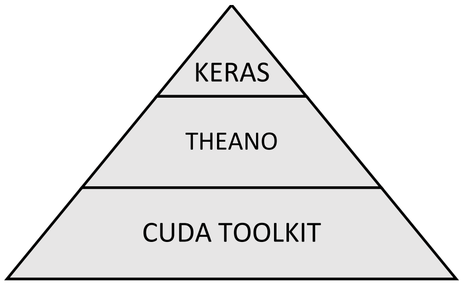


**Figure 14: Technology Stack**

Cuda toolkit is an interface to NVidia GPU's, Theano is deep learning framework built on python, Keras sits on top of theano which gives simplified interface to theano.

We learned that convolutional neural network does extremely well when images are visually distinctive from the human point of view. these images are however not, so it is better we do not keep lofty expectations of impressive results.

So basically, here we are trying to build a prediction model that can differentiate a lung scan whether it belongs to a patient who has cancer or not. To accomplish this, first we need to segregate the whole dataset into cancer and non-cancer set and then process all the DICOM images of all the patients. The processing can be done in 2 ways. One way is to generate a two-dimensional projection of a three-dimensional projection and other is to take images on slices.

**Method 1**: All projections are taken from one specific constant angle. [Masked: Images from Type B Preprocessing]

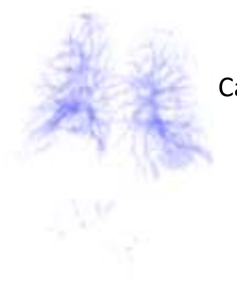


**Figure 15:2D Projections of 3D Nodular region of a cancer patient**

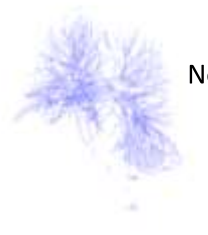


**Figure 16: 2D Projections of 3D Nodular region of a non-cancer patient**

**Method 2**: [Un-Masked: Images from Type A preprocessing with X=512]

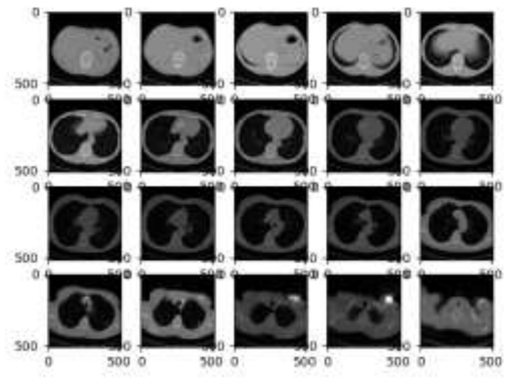


**Figure 17:2D Projections of 3D Nodular region of a cancer patient**

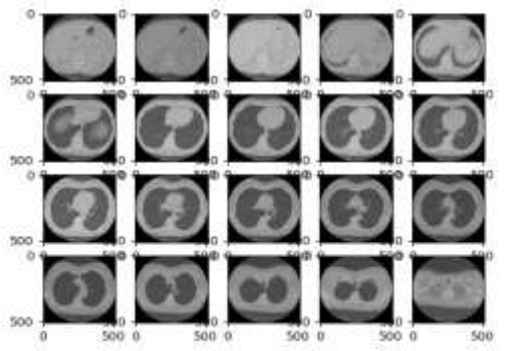
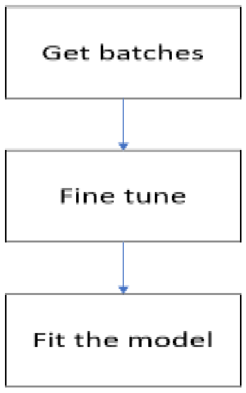


**Figure 18:2D Projections of 3D Nodular region of a non-cancer patient**

Now we have the images ready to be fed into the model. The problem here is reduced to a simple classification problem. Which has been addressed by 2D Convolutional Neural Network. Deep learning technique was not an efficient way to classify images all changed when computers became powerful and became inexpensive. Since we are having ARGO computing resources at our disposal, we can implement this technique for our project.

Now we have good news and bad news. The good news is since we are taking images we can reduce the size of the dataset we are working down to around 100 MB. The bad news is since we are taking projections we will miss details like density.

We repurposed the code of imagenet winning team VGG, they built an image classification model to classify dogs and cat images [10]. This is the flow of the program.



In the get batches section, the program takes 64 patient records at a time from the path we have mentioned. Next in the Fine tune section we modify our prediction model to work to classify patient lung images. In the final section, we fit the model, this is where the heavy computation happens. After this section is done the model automatically validates the validation data set provided.

This method in the right direction with more no. of patient's lung images can get every efficient and will give impressive results.However, for now, we are only limited to patient records of around 1400.

**Results (Whole dataset):**

After the model was run we the output results are tabulated below and from them we can say that the model did not perform better than TensorFlow because the highest accuracy achieved 0.54 against that of 0.62 in TensorFlow.

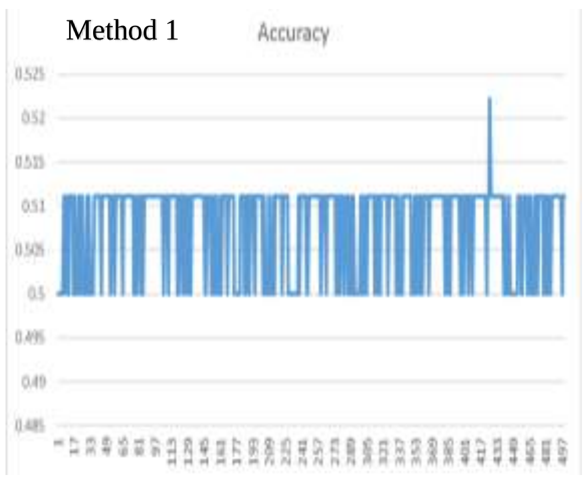
**Method 1:**

| EPOCHS | Run1 512x512 | Run 2 512x512 | Run 3 512x512 |
|---|---|---|---|
| EPOCH 1 | 0.48 | 0.51 | 0.48 |
| EPOCH 2 | | 0.50 | 0.48 |
| EPOCH 3 | | 0.51 | 0.50 |
| EPOCH 4 | | 0.51 | 0.50 |
| EPOCH 5 | | 0.51 | 0.50 |
| EPOCH 6 | | | 0.50 |
| EPOCH 7 | | | 0.51 |
| EPOCH 8 | | | 0.51 |
| EPOCH 9 | | | 0.53 |
| EPOCH 10 | | | 0.51 |
| | 0.48 | 0.51 | 0.50 |

**Method 2:**

| EPOCHS | Run1 512x512 | Run 2 512x512 | Run 3 512x512 |
|---|---|---|---|
| EPOCH 1 | 0.54 | 0.50 | 0.50 |
| EPOCH 2 | | 0.52 | 0.51 |
| EPOCH 3 | | 0.48 | 0.50 |
| EPOCH 4 | | 0.51 | 0.54 |
| EPOCH 5 | | 0.50 | 0.49 |
| EPOCH 6 | | | 0.50 |
| EPOCH 7 | | | 0.50 |
| EPOCH 8 | | | 0.50 |
| EPOCH 9 | | | 0.49 |
| EPOCH 10 | | | 0.50 |
| | 0.54 | 0.50 | 0.50 |

Method 1



Method 2

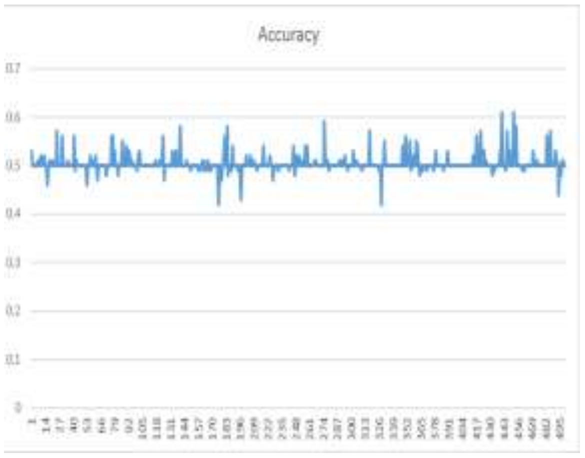


**Figure 19: 500 Epochs for 512x512**
**Method 1-Accuracy 0.50**
**Method 2-Accuracy 0.51**

## VII. COMPUTATION

**Argo Cluster:**

The ARGO Cluster [11] is a batch computing resource available to all faculty at George Mason University. It was assembled by Dell technicians during the week of March 11th, 2013 and consists of the following hardware:

- Dell PowerEdge R720, dual Head Nodes each with Intel Xeon E5-2670 with 8 core CPU, 1TB hard drive, and 64GB RAM.
- 35 Dell C8220 Compute Nodes, each with dual Intel Xeon E5-2670 (2.60GHz) 8 core CPUs, with 64 GB RAM. (Total Cores – 528 and 1056 total threads, RAM > 2TB)
- 2 DELL PowerEdge R815 Fat Nodes, each with quad AMD Opteron 6276 with 16 core CPUs, with 512 GB RAM. (Total Cores – 128, RAM – 1TB)
- Dell NSS PowerEdge R620 with 150TB storage space.
- Interconnect networks are InfiniBand (for message passing), Gigabit Ethernet (for I/O).

The ARGO cluster consists of 50 compute nodes totaling 1060 cores and over 3 TB of memory. The cluster is managed by 2 head nodes, argo-1 and argo-2, which is assigned to the user in a round-robin fashion upon login. 50 compute nodes have dual Intel processors with 64 to 128 GB RAM. Nodes 34 and 35, referred to as “fat nodes”, have quad AMD Opteron processors with 512 GB RAM each. There are 2 GPU compute nodes powered by K80 GPUs (node 40 -4 K80 GPUs, node 50 - 2 K80 GPUs) [12]. Secure shell (SSH) is only way for logging into the ARGO Cluster. Logging machine name is argo.orc.gmu.edu. WinSCP which is a GUI-based SSH transfer client for windows is used for accessing cluster and for file transfers.

Since, Argo cluster is a shared computing cluster, so we cannot run our jobs on the cluster whenever we want, we must write a bash script and submit our job to Slurm, a workload manager. In which we mention how long our job will take to finish and what all computing resources we need to run our job. After that Slurm allocates time slot and dedicated resources for our jobs.

To do preprocessing we have used 100 CPU's and 8 Gb Ram per CPU. We are being able to take advantage of the computing resources available to use by parallelizing our tasks. Here is a figure to help us understand how it was done.

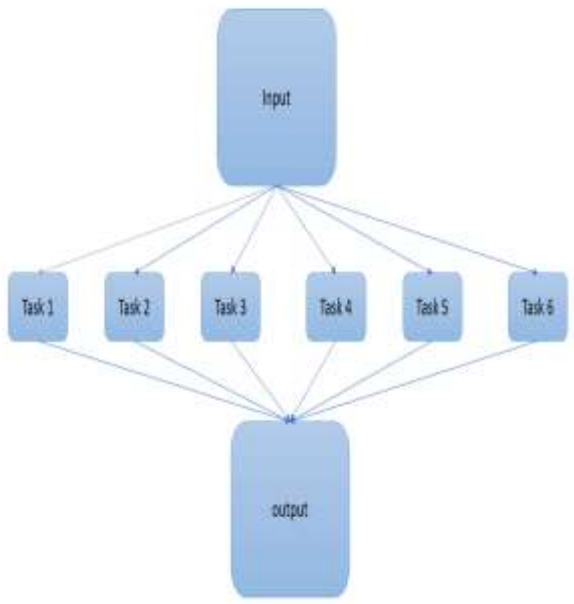


**Figure 20: Working on ARGO cluster**

While building the model on keras we used the node 040 which contained NVidia Tesla GPU and while building Model on TensorFlow we used 24 CPU's.

In the whole, the cluster completed the following three primary tasks

- ✓ Generate 3D Images
- ✓ Convert 3D images into a numpy array
- ✓ Run the convolute neural network model

To generate an image on local system, it will take approximately 20 minutes. To generate all the images, it will take around 1,092,000 minutes [at a stretch 38 days]. This challenge was overcome by using the ARGO computing resources and parallelize it, by doing this we could accomplish this task in around an hour for each kind of image.

## VIII. TOOLS

The following software tools were used in this project

- ✓ Mango (Multi-image Analysis GUI) –For initial Analysis
- ✓ Python –For preprocessing and model building
  - Pandas
  - Numpy
  - Open CV
  - TensorFlow
  - Theano
- ✓ WinSCP –For file transfer on ARGO Cluster
- ✓ Putty – For interface with ARGO cluster and computational runs

## IX. CONCLUSIONS

In totality, the following are the conclusions drawn from the project.

1. TensorFlow outperformed Theano with an accuracy of 73%
2. As the data set is imbalanced with many negative cases i.e. 74% of whole data overall, we got low accuracies of 73% on Tensor flow model and 50% on Theano.
3. If a new CT scan is feed into model, the tensor flow model gives output i.e. contains cancerous lesion or not with a 73% confidence.
4. We are dealing with small number of samples but extremely high

dimensionality.

5. This full-scale image implementation wouldn’t have been possible without using super computation power (ARGO Cluster).

6. To increase accuracy of model, we can add more data to model, run more epochs or modify filters based on domain knowledge instead of random features.

7. CNN models were built with full-fledged image resolution on entire dataset which were not done by any participant on Kaggle.

8. Uniquely built a model (2D CNN on Theano) by taking projections which reduced the data size from 140GB to around 300MB. This can act as a cost-effective solution.

## VIII. FUTURE WORK

Here are few exercises we would like to try in our future work as part of continuation of this project:

- Take help of the radiologist to feed selected features into conv1 of 3D CNN in TensorFlow
- Changing learning rate for optimizer from 1e-3 to 1e-5 as see if it affects accuracy
- Changing number of convolution layers and number of fully connected layers and number of neurons in fully connected layers.
- Will try using 2x2 stride in convolution layer instead of max-pooling layer.

## IN TEXT REFERENCES

[1] CS231n Convolutional Neural Networks for Visual Recognition, Stanford

[2] An Intuitive Explanation of Convolutional Neural Networks

[3] Understanding Convolutional Neural Networks for NLP

[4] Machine Learning – What is the number of filter in CNN?

[5] Neural Networks by Rob Fergus, Machine Learning Summer School 2015

[6] https://www.kaggle.com/anokas/exploratory-data-analysis-4

[7] https://www.kaggle.com/sentdex/first-pass-through-data-w-3d-convnet

[8] https://www.kaggle.com/gzuidhof/full-preprocessing-tutorial

[9] https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb

[10] http://wiki.fast.ai/index.php/Lesson_1_Notes

[11] About ARGO Cluster

[12] ARGO Quick Start User Guide

## REFERENCES

[1] https://www.kaggle.com/anokas/exploratory-data-analysis-4

[2] http://wiki.orc.gmu.edu/index.php/About_ARGO

[3] https://books.google.com/books?id=uvumBu8Srn4C&pg=PA437&lpg=PA437&dq=%E2%80%9CComputer-aided+diagnosis:+a+shape+classification+of+pulmonary+nodules+imaged+by+high-resolution+CT,%E2%80%9D&source=bl&ots=8DHekZ1MpH&sig=

WISf7pOcKaLT7GOaoNzswdA9ntI&hl=en&sa=X&ved=0ahUKEwiL5cf9q8LSAhVs3IMKHTWMCwAQ6AEIKzAE#v=onepage&q&f=false

[4] https://www.kaggle.com/arnavkj95/candidate-generation-and-luna16-preprocessing

[5] http://www.via.cornell.edu/research/

[6] https://www.youtube.com/watch?v=HMcx-zY8JSg#t=4.573879

[7] http://course.fast.ai/start.html

[8] https://www.kaggle.com/sentdex/first-pass-through-data-w-3d-convnet

[9] https://www.hindawi.com/journals/ijbi/2013/942353/tab9/

[10] https://www.kaggle.com/gzuidhof/full-preprocessing-tutorial

[11]http://twiecki.github.io/blog/2014/02/24/ipython-nb-cluster/

[12]https://en.wikipedia.org/wiki/Convolutional_neural_network

[13]https://en.wikipedia.org/wiki/Hounsfield_scale

## GITHUB CODE REFERENCE

https://github.com/pragnakar/DAEN690-Capstone