University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Clinical Decision Support System for Lung Cancer Diagnosis Using Convolutional Neural Networks

A special problem in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Computer Science

Submitted by:

Fatima Naiza O. Asaad

June 2018

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Clinical Decision Support System for Lung Cancer Diagnosis Using Convolutional Neural Networks" prepared and submitted by Fatima Naiza O. Asaad in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

____
Marvin John C. Ignacio, M.Sc. (*cand.*)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*cand.*)	____________	____________
2. Avegail D. Carpio, M.Sc.	____________	____________
3. Perlita E. Gasmen, M.Sc. (*cand.*)	____________	____________
4. Richard Bryann L. Chua, Ph.D. (*cand.*)	____________	____________
5. Ma. Sheila A. Magboo, M.Sc.	____________	____________
6. Vincent Peter C. Magboo, M.D., M.Sc.	____________	____________
7. Geoffrey A. Solano, Ph.D. (*cand.*)	____________	____________

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

____
Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

____
Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

____
Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

According to WHO, lung cancer is the leading cause of cancer-related deaths worldwide. A study has found that early detection of lung cancer using a patient's CT scans has been effective in reducing the deaths caused by lung cancer. Through the use of convolutional neural networks with CT scans as input, a clinical decision support system for lung cancer diagnosis is developed to aid doctors that are non-radiologists in classifying if a patient is positive or negative for lung cancer. The current model used by the system needs to be further enhanced before being deployed for use by doctors. If the model is improved, it could be helpful in providing second opinion on detecting lung cancer in patients.

Keywords: lung cancer diagnosis, convolutional neural networks, deep learning, low-dose computed tomography (LDCT) scan, clinical decision support system

Contents

List of Figures

I. Introduction

A. Background of the Study

Lung cancer is one of the leading cancers worldwide. According to the World Health Organization (WHO), it is the leading cause of cancer-related deaths among both men and women. Of the 8.8 million cancer-related deaths reported in 2015, lung cancer accounted for approximately 1.69 million cases. This was more than the number of cancer-related deaths from liver and colorectal cancer combined (approximately 1.59 million cases) [1]. In the Philippines, according to the Philippine Council for Health Research and Development (PCHRD), it is the top cause of cancer-related deaths among men and the third most common among women [2].

Early detection of cancer before it can cause symptoms is called *cancer screening*. This is usually done to people at high risk of cancer, but do not show symptoms of cancer [3]. For lung cancer, two common screening tests involve the use of chest X-ray (CXR) and the use of low-dose computed tomography (LDCT) scan of the chest.

A United States-based clinical trial on lung cancer screening, called the National Lung Screening Trial (NLST), was sponsored by the National Cancer Institute and was conducted by the American College of Radiology Imaging Network and Lung Screening Study Group. The trial enrolled 53,454 participants and they were recruited between 2002 to 2004. The objecive of this trial was to compare the use of LDCT scan of the chest against the standard CXR as lung cancer screening tests. The study ended in 2010 and the findings were published in 2011. There were 247 deaths from lung cancer per 100,000 person-years in the LDCT group and 309 deaths per 100,000 person-years in the CXR group, representing a relative reduction in lung cancer mortality with LDCT screening of 20%. The NLST showed a reduction of 20% in lung cancer mortality in high-risk subjects scanned with LCDT, compared to the control group that received CXR [4]. In December 2013, the United States Preventive Services Task Force (USPSTF), "an independent panel of experts in primary care and prevention that systematically reviews the evidence of effectiveness and develops recommendations for clinical preventive services", changed its long-standing recommendation regarding lung cancer screening. Before the trial, they believed that there was insufficient evidence to recommend for or against screening for lung cancer. Because of the findings of the NLST, the USPSTF now recommends annual screening for lung cancer using LDCT scan for patients who meet all of the following criteria: 55 to 80 years old, have at least 30 pack-year smoking history, and are either still smoking or have quit smoking within the past 15 years. The USPSTF also recommends that the screening be discontinued once a person has not smoked for 15 years, or develops a health problem that substantially limits life expectancy or the ability or willingness to have curative lung surgery [5].

Clinical decision support systems (CDSSs) are software that aid doctors in clinical decision-making tasks by providing case-specific advice to each patient. CDSSs constitute a major topic in the application of the field of artificial intelligence (AI) in medicine. AI is a thriving field that attempts to understand and build intelligent machines. These days, intelligent machines can now automate routine labor, understand speech or images, and make diagnoses in medicine. An example of its application is a system that can interpret medical images, such as a CXR or an LDCT scan. To interpret a medical image means to detect a possible disease from this image. When this system is given a medical image as input, it can mark conspicuous structures and sections, and/or evaluate the conspicuous structures.

There are two main types of CDSSs: knowledge-based and nonknowledged-based [6]. Knowledged-based CDSSs are based on the rule-based expert systems developed back in the 1970s. These systems are computer programs that are capable of performing at the level of a human expert in a narrow problem domain. Knowledge of these systems are written as IF-THEN statements, called rules. These rules provide some description of how to solve a problem. The IF-THEN structure of rules relates given information or facts in the IF part to some action in the THEN part [7, p. 25].

On the other hand, nonknowledge-based CDSS use an approach to AI called *machine learning* (ML) [6]. The approach is based on learning from past experiences and/or recognizing patterns in an input data. Instead of gathering a large number of information from human experts and writing rules to solve a problem based on this information, the system is given a model with which it can use to evaluate examples, and set of instructions to modify this model when it makes a m istake in its task. Over time, this model would be able perform its task accurately [8, p. 4].

A disadvantage of conventional ML algorithms is manual feature extraction. In ML terminology, a feature is defined as an individual measurable characteristic of a phenomenon being observed. For example, for a system that is given a speech as input and is tasked to recognize whether the speaker is a man, woman, or child, one possible feature is an estimate of the size of the speaker's vocal tract [9, p. 3]. Traditionally, ML researchers spent a large amount of time in manually deciding which characteristics of a dataset can be used as indicators to label that data reliably. This process of manual creation of exhaustive feature sets is called feature extraction. These features are then provided to an ML algorithm. For many tasks, it is difficult to know what features should be extracted. Suppose a system has a task of detecting cars from images. Since cars have wheels, the presence of wheels might be considered as a feature. However, it is difficult to describe what exactly a wheel looks like in terms of pixel values. It has a simple geometric shape but an input image may have shadows on the wheel, or they may be an object in the foreground that is obscuring part of the wheel. These are just a few examples of complications that would make feature extraction difficult for this task [9, p. 3]. Recent research in ML has attempted to build models that can automate the process of feature extraction as well as resemble the structures found in a human brain to simulate how humans learn. This body of research is commonly referred to as *deep learning*.

Deep learning models are based on the biological neural networks, called artificial neural networks (ANNs), that constitute the human brain. There are two main properties of ANNs that follow the general idea of how the brain works. First, the most basic information-processing unit of the network is the artificial neuron. These are based on the biological neurons of the brain and are organized in layers. Like the biological neurons, the artificial neurons pass on some, but not all, information they receive though their connections to other artificial neurons. Second, biological neural networks exhibit plasticity, that is, in response to a stimulation pattern, connections between neurons leading to the "right answer" are strengthened while those leading to the "wrong answer" are weakened. Neurons of an ANN can be trained to pass only useful signals through layers just as the biological neurons of the human brain can be trained to pass forward only the signals that use useful in achieving the larger goals of the brain [10, p. 5].

The process of learning in a neural network is also called *training*. The connections among the neurons in the networks have weights. These weights signify the correlation between a signal and the network's outcome. A bigger weight signifies a tighter correlation. The biases are scalar values added to the input to ensure that at least a few nodes per layer are activated regardless of signal strength. Inputs through these connections affect the network's interpretation of data more than the inputs from connections with smaller weights. Training the model is the process of continuously readjusting the weights and biases of the network until a well-trained model is achieved [10, p. 56]. The most common learning algorithm in neural networks is the *backpropagation algorithm*.

The dataset used for training and evaluating the model, called *training data*, is typically split into three sets: the *training set*, the *validation set*, and the *test set*. The model is trained on the training set. The fitted model is then evaluated on the validation set. Training the model on the training and validation set is continuously done until a well-trained model is, hopefully, achieved. The *final* model is then evaluated on the test set.

To quantify how close a neural network is to the ideal and well-trained model, a metric based on the error observed in the network's predictions is calculated. These errors are aggregated over the entire dataset and averaged. The result is a number that represents how close the neural network being trained is to its ideal network [10, p. 71]. This value is called the *training error* because it is the error returned when the model is executed on the training set. The errors returned when the model is executed on the validation set and the test set are called the *validation error* and the *generalization error*, respectively.

To make the networks train better and faster, certain values of the network can be tuned. These are called *hyperparameters* [10, p. 78]. An example of a hyperparameter is the *learning rate*. In layman's terms, the learning rate is how quickly a network abandons old beliefs for new ones. Another hyperparameter is the *epoch*. An epoch is a single iteration over the entire training dataset. The number set as the value of the epoch is the number of iterations that will be done over the dataset. The *batch size* is another hyperparameter. This value is the number of training examples utilized in one iteration.

An example of a class of neural network is the Convolutional Neural Network (CNN). This network is often used in image recognition systems, including medical images. A deep learning library called *Keras* can be used to develop a CNN model.

Lung cancer detection tools have been built before. It has been found that usage of CNN is best for this classification problem. Song et al. [11] developed a lung cancer detection system using different models to compare their performance: a deep neural network-based (DNN; made up of numerous fully-connected layers) and a CNN-based one. The CNN-based model attained the best performance: an accuracy of 84.15%, a sensitivity of 83.96%, and a specificity of 84.32%. In contrast, the DNN performed poorly in all three metrics compared to the CNN: 82.37%, 83.96%, and 81.35%.

For lung cancer detection using deep learning-based tools, the model is going to look at the lung nodules (if present) in patient's CT scans. If present, the size, thickness, and shape of the nodule determines the probability of a patient having lung cancer.

B. Statement of the Problem

Current deep learning-based CDSS for lung cancer screening lacks the feature that allows an AI expert to train a new model. This allows researchers to experiment with different sets of hyperparameters and network architectures to obtain a better classification performance.

While lung cancer screening by LDCT scan has been found to reduce mortality by 20%, the NLST also showed that LDCT scan has resulted in high false-positive rates of around 25%. A false-positive result leads to unnecessary stress and anxiety for patients and their families as well as additional follow-up imaging (which leads to more exposure to radiation), and interventional treatment (which may be invasive) [4].

C. Objectives of the Study

C..1 General Objective

To develop a deep learning-based clinical decision support system that can aid doctors (non-radiologists) in determining whether or not a patient has lung cancer based on the patient's thoracic LDCT scans.

C..2 Specific Objectives

1. Allow a doctor (non-radiologist) to:
    (a) Perform classification of CT scans (positive or negative for lung cancer)
        i. Upload LDCT scans (DICOM format) of a patient.
        ii. View the classification result (positive or negative for lung cancer) generated by the system.
2. Allow an AI expert to:
    (a) Perform classification of CT scans (positive or negative for lung cancer)
        i. Upload LDCT scans (DICOM format) of a patient.
        ii. View the classification result (positive or negative for lung cancer) generated by the system.
    (b) Create a new model.
        i. Upload the training data to the system.
            A. Upload LDCT scans (DICOM format).
            B. Upload a CSV file where each row contains the filename of the CT scan and a label of whether the scan is positive (1) or negative (0) for lung cancer.
        ii. Select pretrained network to use from a list of options (supported by the Keras library).
        iii. Set a value for the following hyperparameters:
            A. Number of epochs
            B. Batch size
            C. Optimizer
            D. Learning rate

iv. Train the neural network.

A. View the training accuracy and loss per epoch after the model is created.

B. View the validation accuracy and loss per epoch after the model is created.

C. View the test accuracy and loss after the model is created.

(c) View a list of the models created.

(d) View the specific details of a created model.

i. View the model's training accuracy, training loss, validation accuracy, validation loss, test accuracy, and test loss.

ii. View a graph that contains the accuracy (training and validation) of every epoch while the model was being trained.

iii. View a graph that contains the loss (training and validation) of every epoch while the model was being trained.

iv. View a visualization of the layers of the model.

D. Significance of the Project

The creation of a deep learning-based CDSS for lung cancer diagnosis can be used to aid doctors that are non-radiologists in classification of CT scans of a patient as positive or negative for lung cancer.

The CDSS is implemented as a web application where users can upload CT scans of patients of DICOM file format. Because the DICOM file format includes metadata about the patient, there is a privacy risk when uploading CT scans. However a web application was chosen instead of a native application because the server is sure to have access to a GPU. While a CPU can be used to train a model and perform classification, a study has found that training with a GPU is more efficient than with a CPU [12].

Allowing an AI expert to train new models will give him/her the capability to experiment with different architectures and different sets of hyperparameters. Also, because research in the deep learning field is currently thriving, when a better network architecture is developed in the future, the system can make use of this architecture (after it has been supported by the Keras library).

E. Scope and Limitations

The following are the scope and limitations of the system:

1. The system will only perform lung cancer detection: determining if the set of images uploaded are positive or negative for lung cancer.

2. The training data used is limited to the thoracic LDCT scans found in the Kaggle Data Science Bowl 2017 dataset.

3. The training data used was undersampled to avoid a class imbalance. The final training data consists of 136,000 images: 95,200 for the training set, 20,400 for the validation set, and 20,400 for the test set.

4. The training data is limited to thoracic LDCT scans of DICOM file format.

5. The input when performing prediction is assumed to be thoracic LDCT scans of DICOM file format.

6. The dimension of the training and input data is 512 x 512.

7. Because the thoracic LDCT scans used as training data are from adult American patients, the system is not guaranteed to be accurate when evaluated using thoracic LDCT scans of a non-adult and/or non-American patients.

8. The input can only contain the possibility of lung cancer and no other abnormalities.

9. The system will handle the partitioning the training data into 3 sets (training set, validation set, and test set) when creating a new model.

10. When creating a new model, the AI expert will provide the CT scans as well as a CSV file that contains the label of each scan: 0=has lung cancer, 1=no lung cancer.

F. Assumptions

The following are assumed for the effective run of the system:

1. The system is only used as an aid for lung cancer detection. The final diagnosis will still be given by the radiologist.

2. The system is designated for use of doctors who are not specialized in diagnosing diseases using medical imaging techniques (non-radiologists).

II. Review of Related Literature

Detection of lung cancer from results of imaging tests is a very active research topic among machine learning researchers. A study by Mahale et al.[13] proposed a model using SVM algorithm for feature selection and classification, and lung CT scans as input. Modified Fuzzy Possibilistic C Means (MFPCM) was used for segmentation and Gabor filter for De-noising the medical images. Different SVM kernels were applied to the CT images (Linear, radial basis function (RBF), Polynomial). The accuracy for the usage of RBF kernel is better as compared to other SVM kernels.

Another study by Trivedi et al.[14] used SVM classifiers for diagnosis of lung cancer using CT scan images as input. The main objective of the authors is to improve the accuracy rate for lung cancer diagnosis by designing a hybrid SVM.

Convolutional Neural Networks (CNNs) are widely used in the field of medical image analysis for image representation and classification of medical images. This is because of CNN's ability to automatically extract the features needed by the algorithm to perform its task. A study done by Abdolmanafi et al.[15] demonstrates this capability by developing a fully automated tissue classification method that uses a CNN as the feature extractor. In this study, the CNN is used to classify the coronary artery layers from an input of optical coherence tomography (OCT) images of pediatric patients.

Rao et al.[16] designed a CNN (CanNet) to classify tumors (positive or negative for lung cancer) seen in lung cancer screening computed tomography (CT) scans. The data used came from the Lung Image Database Consortium and Image Database Resource Initiative (LIDC-IDRI). The results showed that the classification accuracy of CanNet is better compared to a traditional artificial neural network (ANN), with 50 neurons in the hidden layer, and also an existing CNN, LeNet, built for image classification. With CanNet, the authors could get up to 45% improvement in accuracy compared to LeNet and about 14% compared to a tradiational ANN.

In a study by Song et al.[11], three types of deep neural networks (CNN, DNN, and SAE) were used compare their performance on the classification of the benign and malignant nodules. The dataset came from the LIDC-IDRI. The CNN achieved the best performance: 84.15% accuracy, 83.96% sensitivity, and 84.32% specificity.

Li et al.[17] designed a deep convolutional neural network (DCNN) for the automated feature learning and nodule classification of thoracic CT scan images. The dataset used was again from the LIDC-IDRI. In total, 62,492 region-of-interest (ROI) samples (40,772 nodules and 21,720 nonnodules) were used.

CNNs can also aid in the reduction of the false-positive rates of lung cancer screening. A study by Dou et al.[18] proposed novel method of employing 3D CNNs for false-positive reduction in automated pulmonary nodule detection from volumetric CT scans. The proposed method was validated in the LUNA16 challenge held in conjuction with ISBI 2016, where the authors achieved the highest competition performance metric score in the false-positive reduction track. Experimental results demonstrated the importance and effectiveness of integrating multi-level contextual information into 3D CNN framework for automated pulmonary nodule detection in volumetric CT data.

Rossetto et al.[19] presented an ensemble of Convolution Neural Networks using multiple preprocessing methods to increase the accuracy of the automated labeling of the scans. This was done by implementing ensembles of CNNs along with a voting system to get the consensus of the two networks. The initial results of the best method show both a high accuracy (97.5%) and a low percentage of false-positives (¡10%). A similar study that used DCNN for lung cancer detection, but with microscopic images as input, was done by Teramoto et al.[20] The DCNN used for the classification consists of three convolutional layers, three pooling layers, and two fully connected layers. It was trained using 76 cases of cancer cells by exfoliative or interventional cytology under bronchoscopy or CT-guided fine needle aspiration cytology.

Approximately 71% of the images were classified correctly.

Alakwaa et al.[21] demonstrated lung cancer detection and classification of CT scans with unmarked nodules using a 3D CNN. The data came from the Kaggle Data Science Bowl 2017 (KDSC17). Thresholding was used as the initial segmentation approach to segment out the lung tissues from the rest of the CT scan. Then, a modified U-Net trained on the LUng Nodule Analysis 2016 (LUNA16) dataset (selected data from the LIDC-IDRI) was used to first detect nodule candidates in the KDSC17 dataset. These candidates were fed into the 3D CNN for the actual classification of the CT scans as positive or negative for lung cancer. A test set accuracy of 86.6% was produced by the 3D CNN.

III. Theoretical Framework

A. Lung Cancer

Lung cancer, also called lung carcinoma, is defined as the uncontrolled growth of abnormal cells in one or both lungs of the body [22, p. 4]. The two main types of lung cancer are small-cell lung carcinoma and non-small-cell lung carcinoma. The most common symptoms are coughing, coughing up blood, weight loss, shortness of breath, and chest pains.

The only recommended screening test for lung cancer is low-dose computed (LDCT) scan. Research has not found any benefits in using two other available screening tests: sputum cytology and chest X-ray (CXR) [23].

The U.S. Preventive Services Task Force (USPSTF) recommends annual lung cancer screening with LDCT for people who [5]:

- Have a smoking history of 30 pack years or more (heavy smoking), and
- Smoke now or have quit within the past 15 years, and
- Are between 55 and 80 years old.

Screening for lung cancer has at least three risks [23]:

1. **A false-positive result** - the screening test may suggest that person has lung cancer even though cancer is not present. This can lead to additional follow-up imagin tests and invasive treatment, such as surgery, that are not needed and may have more risks.
2. **Overdiagnosis** - the screening test may find cases of cancer that may never have caused a problem for the patient. Like the false-positive result, this can lead to additional unnecessary treatment.
3. **Can cause cancer** - repeated LDCT tests can cause cancer in healthy people.

These risks are the reason why the USPSTF recommends lung cancer screening to be done only for adults who do not show symptoms of lung cancer, but are at high risk for developing it because of their smoking history and age.

The USPSTF also recommends that annual lung screening should stop when the person being screened [5]:

- Has not smoked in 15 years, or
- Develops a health problem that substantially limits life expectancy or the ability or willingness to have curative lung surgery

B. Computed Tomography (CT) Scan

A CT scan uses X-rays and a computer to produce detailed images of a structures inside the body. A CT scan can be used for diagnosing conditions, guiding doctors in deciding whether a patient is in need of further tests or treatments, and monitoring conditions [24].

CT scans are more sensitive than the standard chest X-rays for identifying lung cancer, including small (early stage, operable) lung cancers. The false-positive rates with CT scans range from 5% to 40% [22, p. 42].

For lung cancer screening, the only recommended test is low-dose CT (LDCT) scan. The difference between conventional CT scan and LDCT scan is the amount of radiation a patient is exposed to. LDCT scan uses approximately 5 times less radiation than the conventional CT scan. Depending on the size of the patient, an LDCT scan typically delivers 1-4 millisieverts of radiation exposure, while a conventional CT scan typically delivers between 5-20 millisieverts [25]

Figure 1 shows three examples of thoracic LDCT scans that contains nodules and non-nodules.

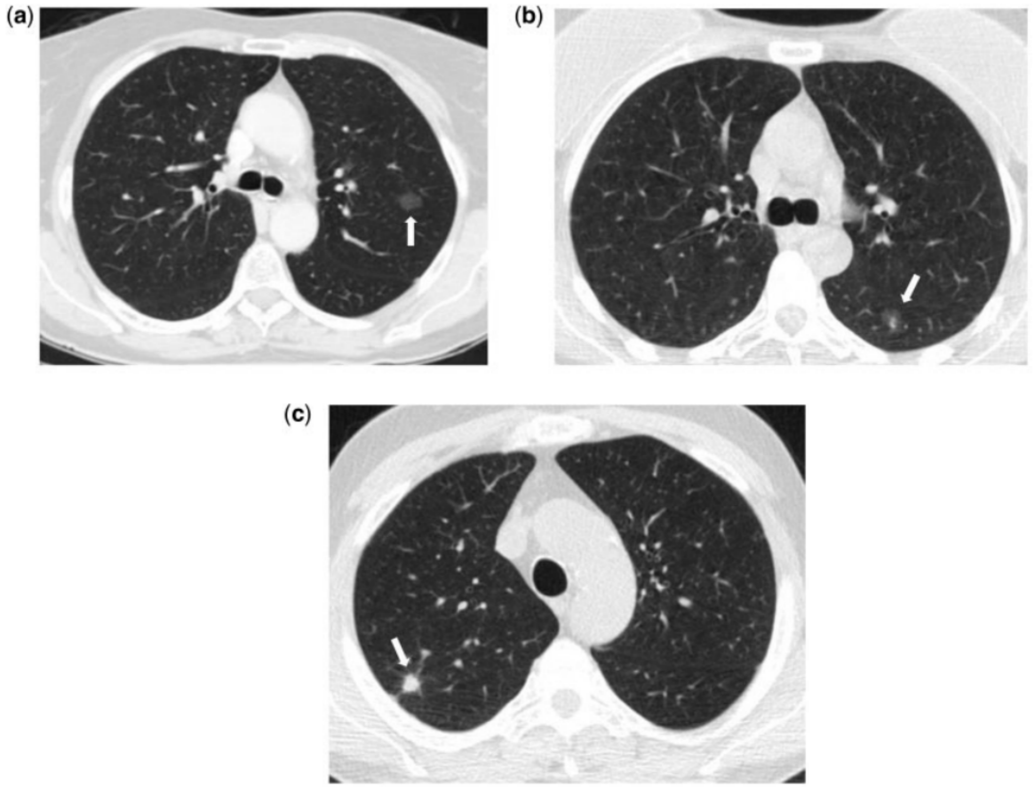


Figure 1: (a) Transverse unenhanced LDCT scan of a 63-year old man shows a non-solid nodule in the upper lobe. (b) Transverse unenhanced LDCT scan of a 66-year old woman shows a part-solid nodule in the lower left lobe. (c) Transverse unenhanced LDCT scan of a 58-year old man shows a solid nodule in the upper right lobe.

C. Clinical Decision Support System (CDSS)

A CDSS is a health information technology system designed for providing physicians and other health professionals with clinical decision support (CDS) [26]. CDS includes computerized alerts and reminders, clinical guidelines, patient-data reports and summaries, diagnostic support, among other tools [27].

D. Digital Imaging and Communications in Medicine (DICOM)

DICOM is a standard for handling, storing, printing, and transmitting information in medical imaging. It includes a definition of a file format of patient data (such as the output image of a CT scanner) as well as a network communications protocol for communication between systems that support the file format [28].

E. Convolutional Neural Network (CNN)

CNNs are a specialized kind of neural network for processing data that has a known, grid-like topology. This includes images because these are represented as a 2D grid of pixels. CNNs employ an operation called *convolution*. The goal of a CNN is to learn higher-order features in the data via convolutions.

CNNs have three major groups of layers, as shown in Figure 2:

1. Input layer

2. Feature-extraction (learning) layers

3. Classification layer

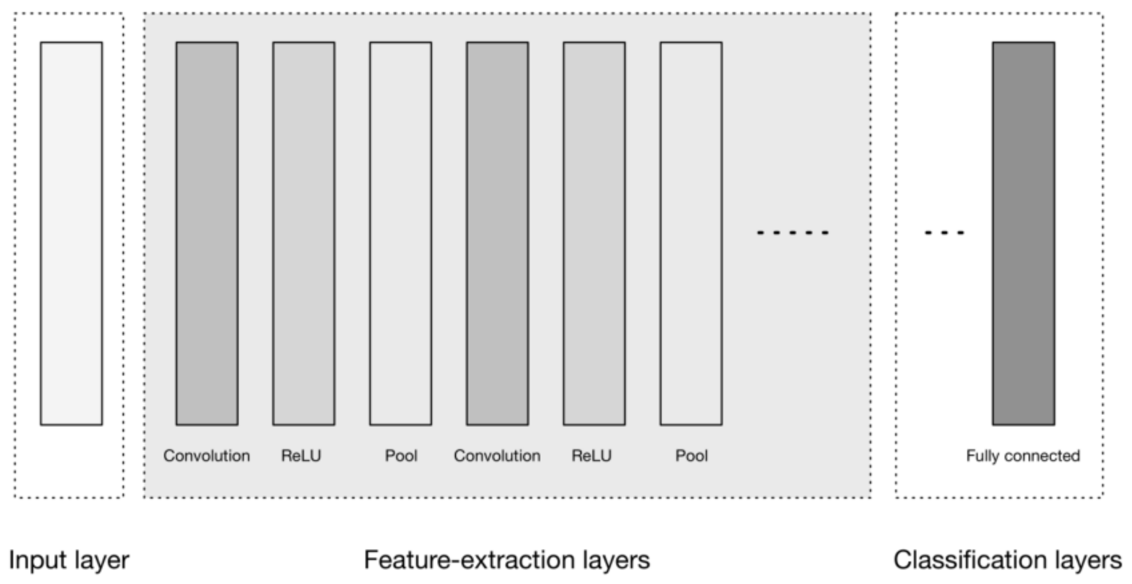


Figure 2: High-level general CNN architecture

The input layer is where we load and store the raw input data for processing in the network. The input data specifies the width, height, and number of color channels.

The feature extraction layers, or the layers that enable the neural network to learn, are composed of two repeating layers:

1. **Convolution layers** - the building blocks of CNN architectures. These layers transform the input data by using a patch of locally connecting neurons from the previous layer. They will then compute a dot product between the region of the neurons in the input layer and the weights to which they are connected in the output layer. The output generally has the same spatial dimensions, or smaller. Sometimes though, the number of elements in the third dimension of the input (depth) increases.

2. **Pooling layers** - commonly inserted between successive convolutional layers. These layers reduce the data progressively over the network (known as *downsampling*). The most common downsampling operation is *max pooling*.

After several convolutional and pooling layers, the final reasoning in the network is done via the classification layers. These layers are usually **fully connected layers**. Just like in conventional neural networks, the neurons in these layers have connections to all neurons in the previous layer. The class scores to be used as output of the network are computed in these layers.

F. Keras

Keras a neural networks API capable of running on top of TensorFlow, CNTK, and Theano. It allows for easy and fast prototyping and supports both convolutional networks and recurrent networks, as well as combinations of the two. [29]

IV. Design and Implementation

A. Kaggle Data Science Bowl 2017

The dataset used for this study comes from the Kaggle Data Science Bowl 2017. The dataset is composed of anonymized high-resolution lung scans from hundreds of patients. This dataset was provided by the National Cancer Institute (NCI) of the United States of America to create algorithms that could determine if a patient has lung cancer based on his/her CT scans.

B. System Overview

The CDSS Lung Cancer Diagnosis detects and classifies patient's thoracic LDCT scans as either positive or negative for lung cancer based on the CNN model loaded in the system.

The CDSS is split into two parts: the client and the server. Figure 3 shows a high-level overview of the system.

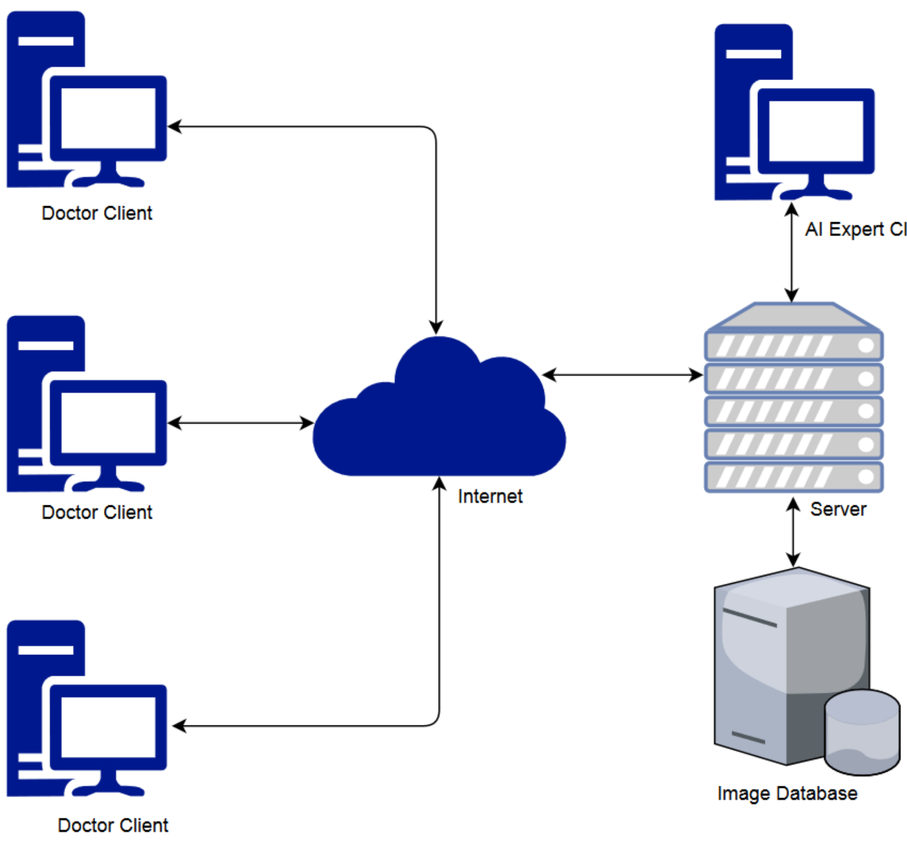


Figure 3: System Overview

The server contains the classification and image storing processes while the client is further divided into two types and the purpose depends on this type.

The two types of clients are:

1. Doctor Client (DC)

2. AI Expert Client (AEC)

The DC client is accessed though the Internet via a web browser. This is what doctors use to interact with the system to classify slices of a patient's thoracic LDCT scans. The AEC is installed in a desktop computer. It is through this that the AI expert can interact with the system to train and evaluate models.

C. Context Diagram

The system's input and output requirements are defined by the context diagram shown in Figure 4.

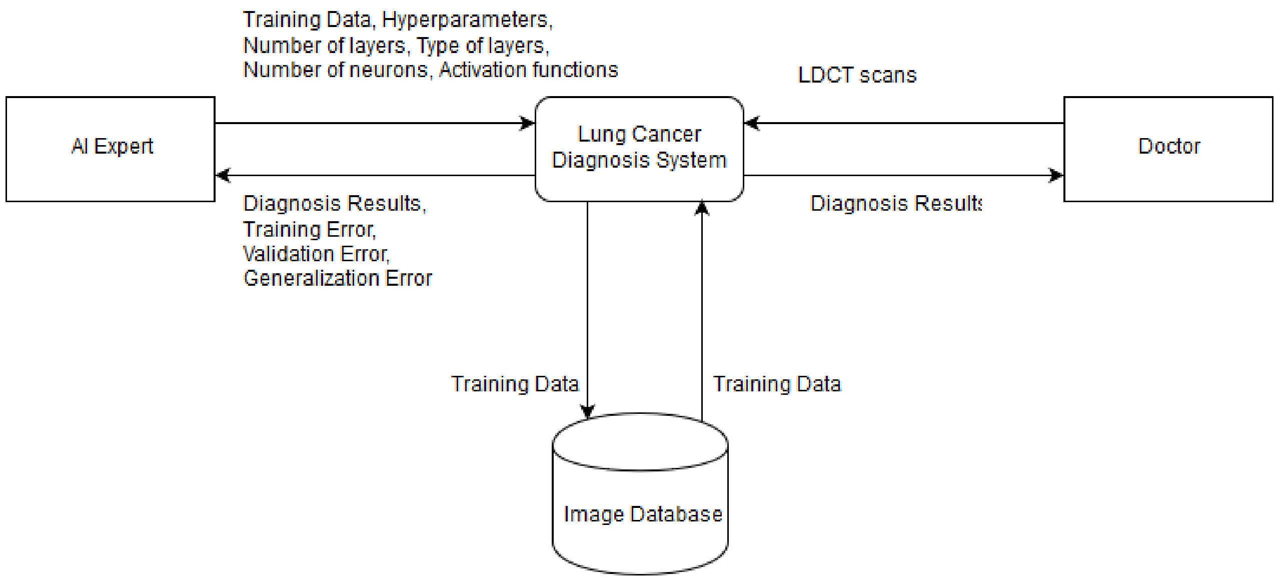


Figure 4: Context Diagram

The context diagram shows the general interaction of the users and the system. The AI expert needs to input the training data, define the model architecture (set training parameters like activiation functions, the layers, the number of neurons in a layer, etc), and set the values of the hyperparameters. The output given to the AI expert includes the diagnosis results of each case, training error, validation error, and generalization error.

The doctor inputs thoracic LDCT scans of a single patient and the system returns the diagnosis results.

D. Entity-Relationship Diagram

The figure below shows the entity-relationship diagram of the system's database.

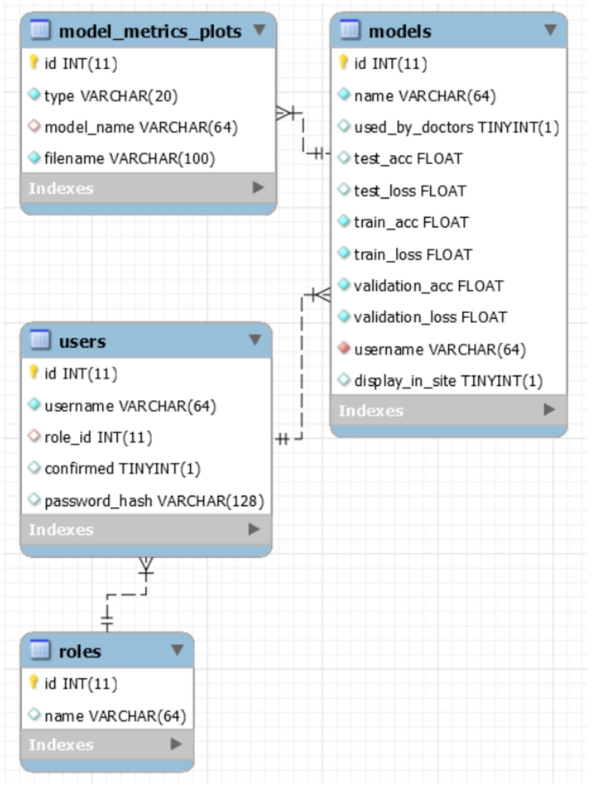


Figure 5: Entity-Relationship Diagram

The users table specify attributes about the user including his/her username and role_id. The roles table specify the allowed roles of a client.

The models table stores the filename in the server of the model as well as the performance of the model: the training accuracy, training loss, validation accuracy, validation loss, test accuracy, and test loss of a model. Included in this table as well is a boolean that indicates whether the model is the system model.

The model_metrics_plots table stores the filename in the server of the graph of the accuracy and loss of the model vs the epoch while it was being trained.

E. Use Case Diagram

The specific functionalities of the users are shown in the use case diagram in Figure 6.

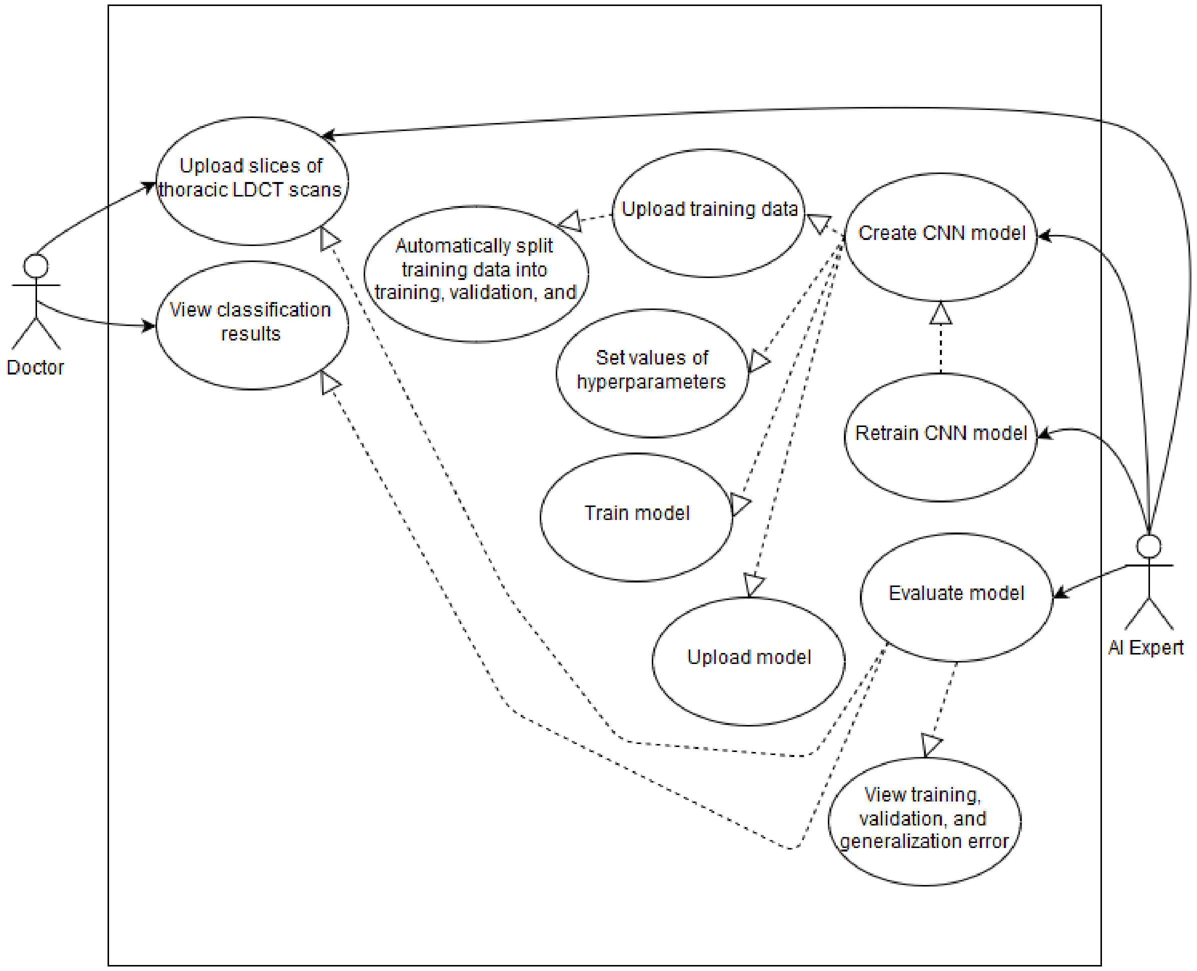

Upload slices of thoracic LDCT scans
View classification results
Doctor
Automatically split training data into training, validation, and
Upload training data
Create CNN model
Set values of hyperparameters
Train model
Retrain CNN model
Evaluate model
Upload model
AI Expert
View training, validation, and generalization error

Figure 6: Use Case Diagram

The doctor can upload slices of thoracic LDCT scans of a patient and view the diagnosis results generated by the system.

The AI expert can choose to either perform prediction of a patient's LDCT scans, train a new CNN model, or view his/her created models. When creating a new model, the training data needs to be uploaded. The system automatically partitions the data into three sets (training, validation, and test). The AI expert then defines the model architecture (convolutional base), sets the values of the hyperparameters, then trains the model. After creating a new model, the system determines if the new model has a better performance compared to the new model. If so, the system replaces the system model with the new model. The next time a doctor uses the system to classify slices of CT scans, the new model will be used.

F. Activity Diagram

F..1 Doctor

A doctor using a web browser supported by the system can request for a lung cancer diagnosis of a patient using thoracic LDCT scans as input.

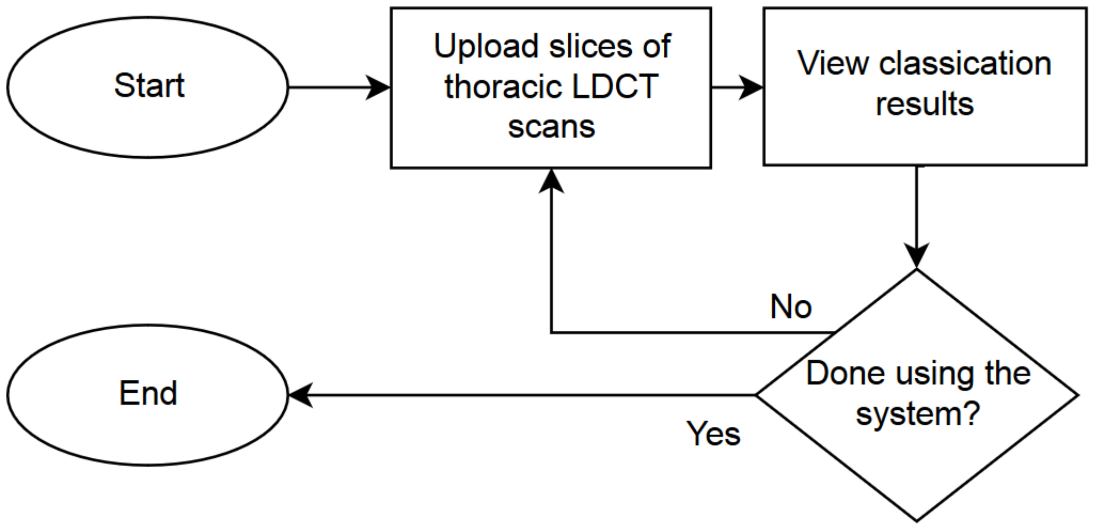

Start
Upload slices of thoracic LDCT scans
View classication results
No
Done using the system?
Yes
End

Figure 7: Activity Diagram of Doctor

F..2 AI Expert

The training dataset will be used by the AI Expert to produce a CNN model that performs the kung cancer classification. If there is a better architecture, the AI Expert can create a new model that uses a different architecture which can produce better diagnosis results.

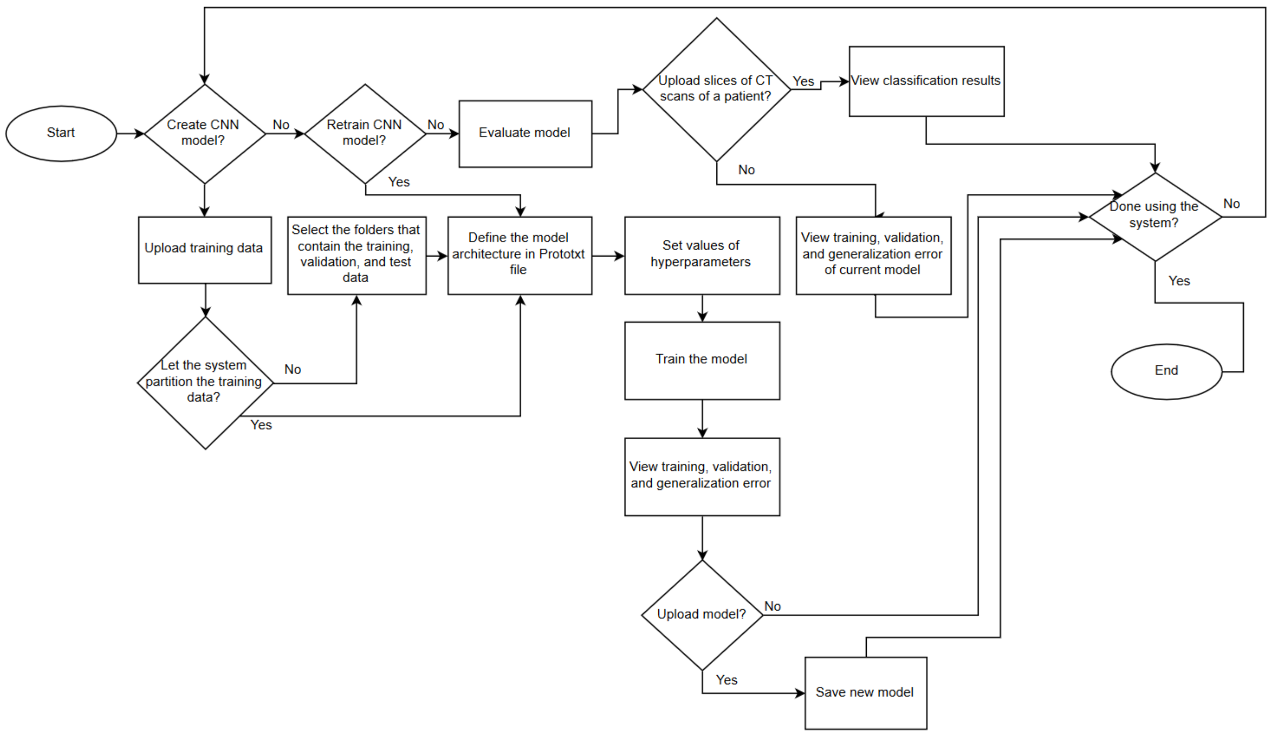

Start
Create CNN model?
No
Retrain CNN model?
No
Evaluate model
Upload slices of CT scans of a patient?
Yes
View classification results
No
Yes
Upload training data
Select the folders that contain the training, validation, and test data
Define the model architecture in Prototxt file
Set values of hyperparameters
View training, validation, and generalization error of current model
Done using the system?
No
Yes
End
Let the system partition the training data?
No
Yes
Train the model
View training, validation, and generalization error
Upload model?
No
Yes
Save new model

Figure 8: Activity Diagram of AI Expert

G. Technical Architecture

The minimum recommended requirements for the server machine include:

1. Intel®Core-TM i5 CPU or higher
2. 8 GB of RAM or higher
3. An NVIDIA GPU card with Compute Capability of 3.0 or higher
4. Windows 7/8/8.1/10

The recommended web browsers that doctors can use to interact witht the system are:

1. Google Chrome
2. Mozilla Firefox

3. Safari

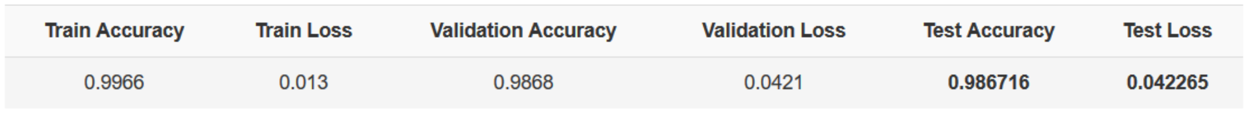
4. Microsoft Edge

V. Results

The dataset was trained via transfer learning. A pretrained model trained on the ImageNet database was utilized. Only the convolutional neural network of the pre-trained model (called convolutional base) was used in this system. Because there are 1000 classes when classifying the images in the ImageNet database, the classifier of the pretrained models was not used. A new set of fully-connected layers for classifiy-ing the dataset into 2 results (positive or negative for lung cancer) was trained from scratch.

The model was trained with 136,000 images. The training set contains 95,200 images while the validation and test set contains 20,400 images each. The VGG16 neural network architecture was used as the convolutional base of the model.

The model attained a test accuracy of 98.6716% and a test loss of 0.042265.



Train Accuracy	Train Loss	Validation Accuracy	Validation Loss	Test Accuracy	Test Loss
0.9966	0.013	0.9868	0.0421	**0.986716**	**0.042265**

Figure 9: Performance of the Model

The web application starts with the registration page. A user can register either as an AI expert or as a doctor.

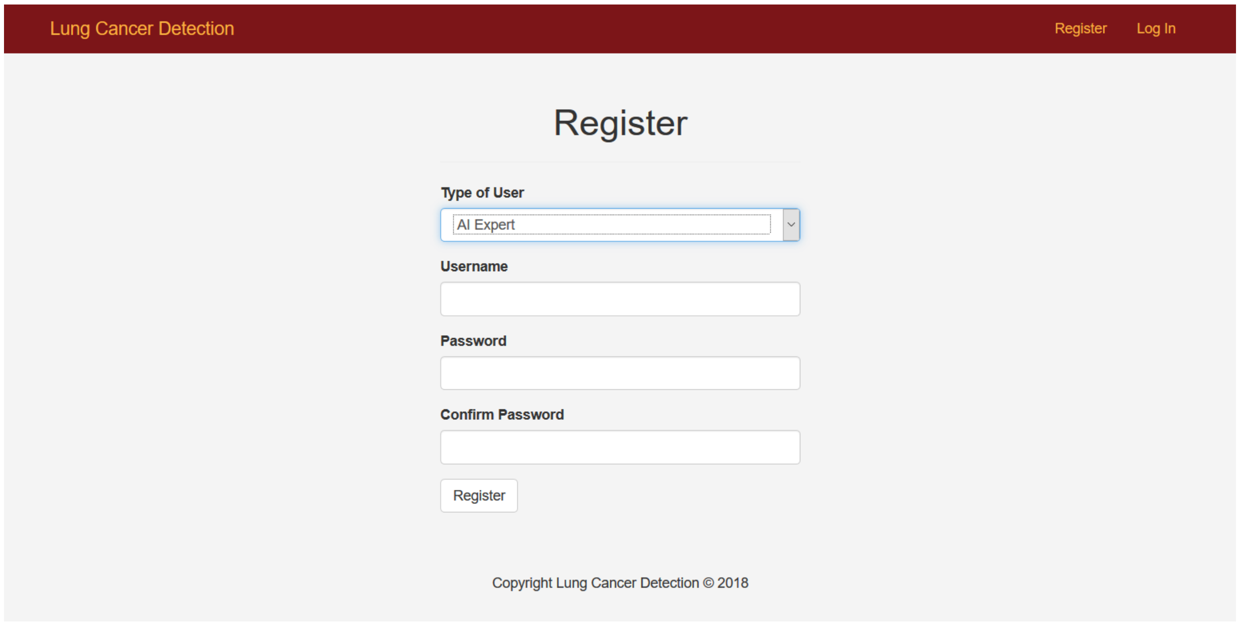


Figure 10: Registration Page

Registered users can login to the web application via the login page shown below.

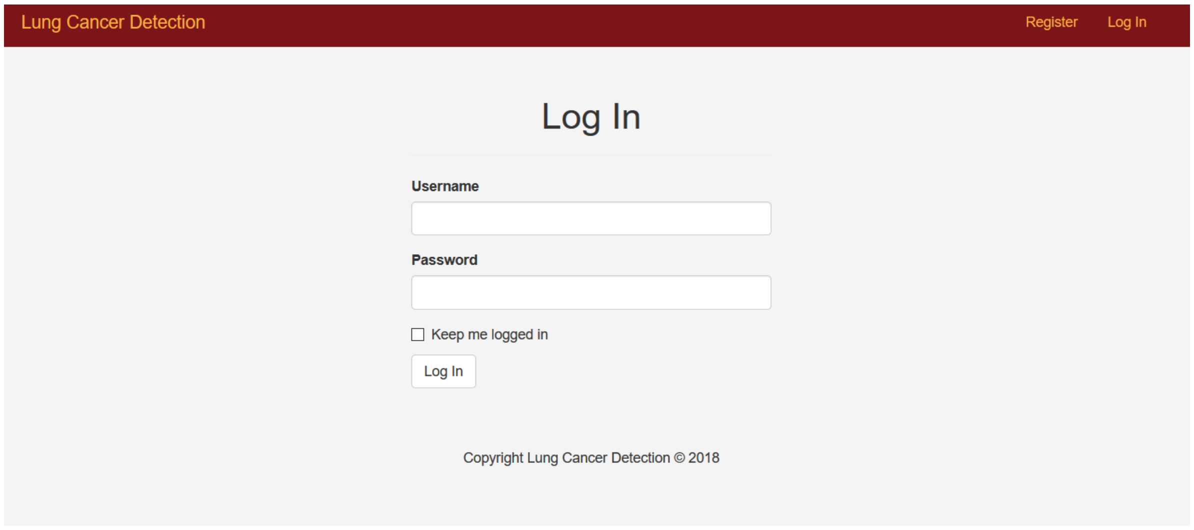


Figure 11: Login Page

If a doctor logs in, the doctor is redirected to the page that allows him/her to perform prediction of lung cancer based on a patient's CT scans.

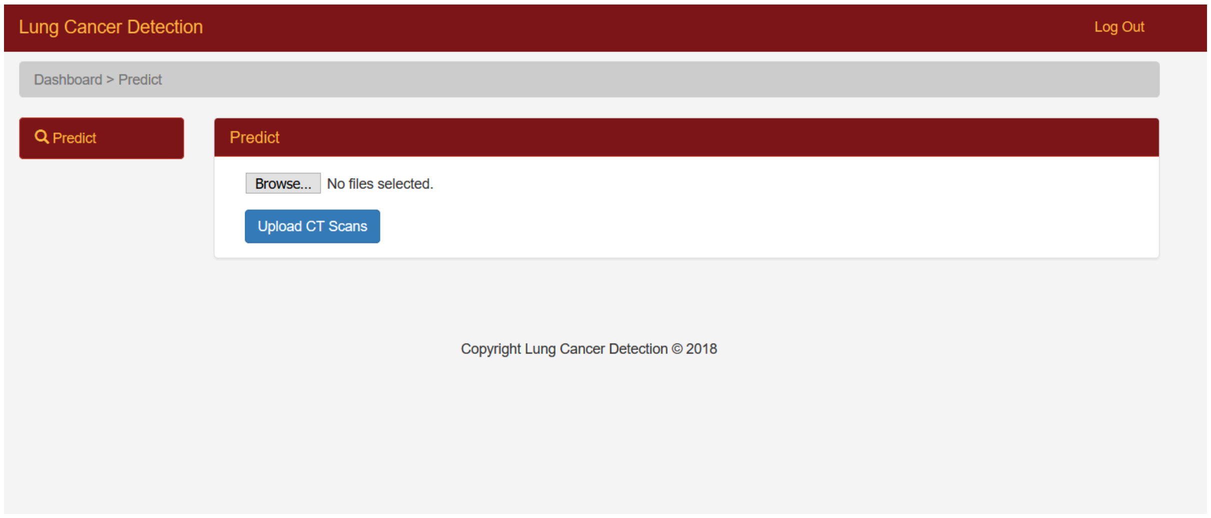
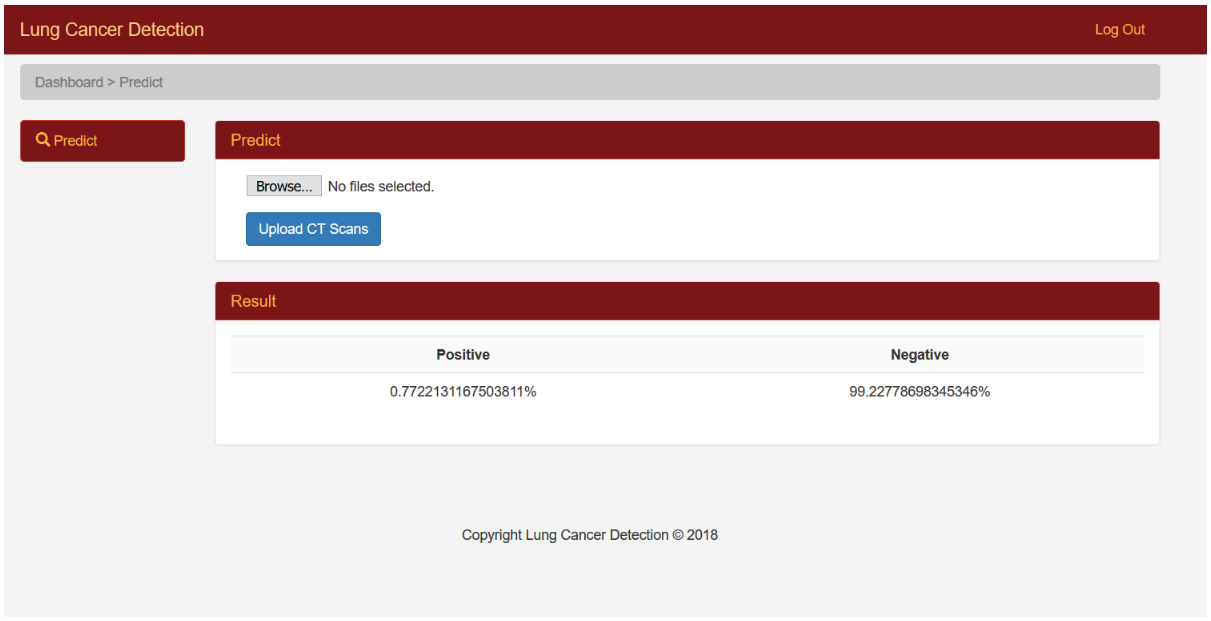


Figure 12: Prediction Page Seen by a Logged In Doctor

After uploading the CT scans, the page is updated to show the results of the classification (positive or negative for lung cancer).



Positive	Negative
0.7722131167503811%	99.22778698345346%

Figure 13: Prediction Page Updated with the Results of the Classification

When an AI expert is logged in, the web application is redirected to the page that allows him/her to perform prediction of lung cancer based on a patient's CT scans.

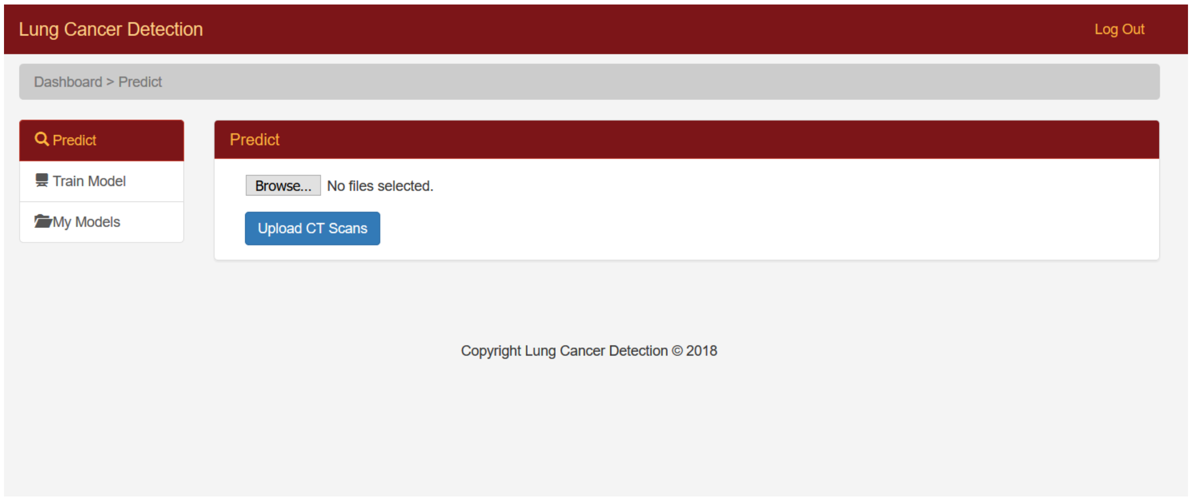
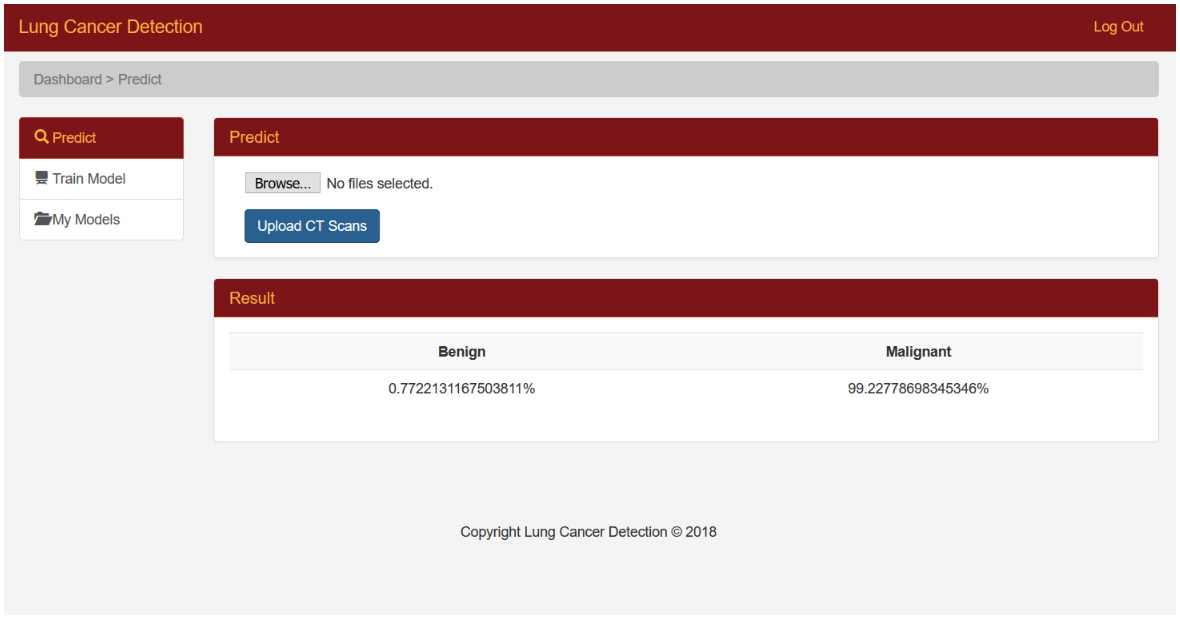


Figure 14: Prediction Page Seen by a Logged In AI Expert

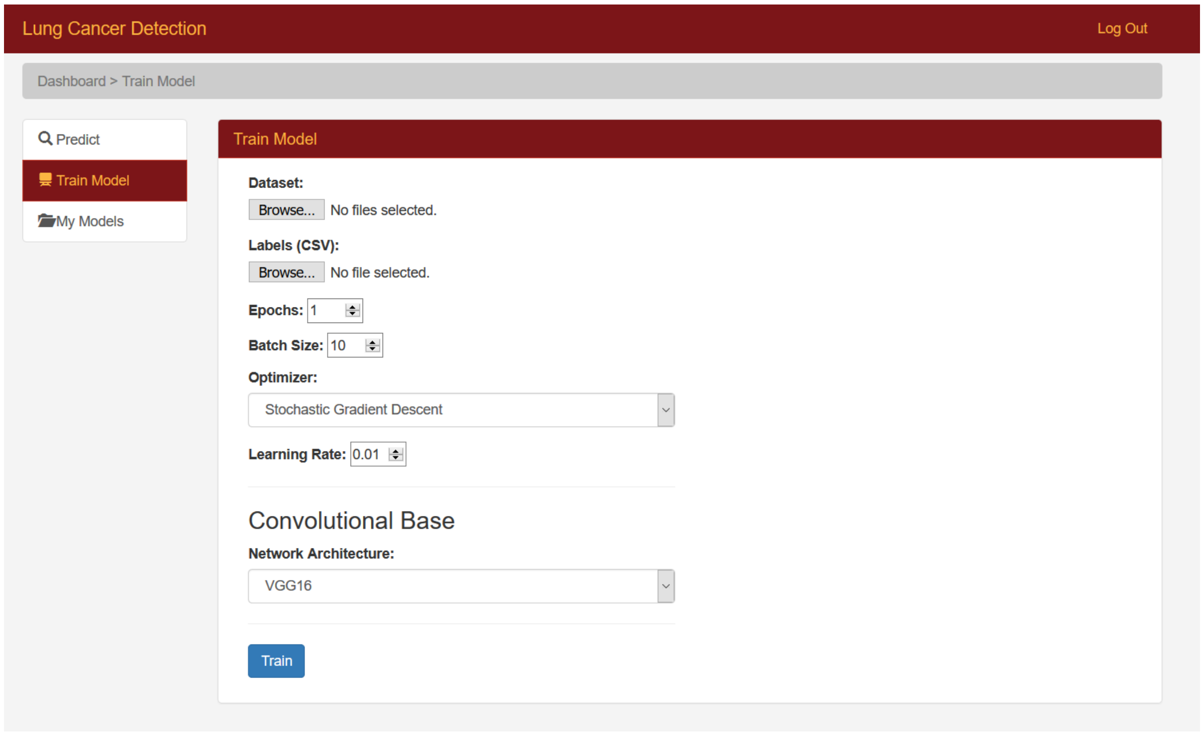
After uploading the CT scans, the page is updated to show the results of the classification (positive or negative for lung cancer).



Benign	Malignant
0.7722131167503811%	99.22778698345346%

Figure 15: Prediction Page Updated with the Results of the Classification

The AI expert can train a new model via the train model page. This page allows the AI expert to upload the training data as well as the labels of the training data in CSV file format. The hyperparameters of the model can also be tweaked in this page. The AI expert may specify the number of epochs, batch size, optimizer, and learning rate of the optimizer. The AI expert can also select a convolutional neural network to use from a list of pretrained convolutional neural networks.



Dataset: No files selected.

Labels (CSV): No file selected.

Epochs: 1

Batch Size: 10

Optimizer: Stochastic Gradient Descent

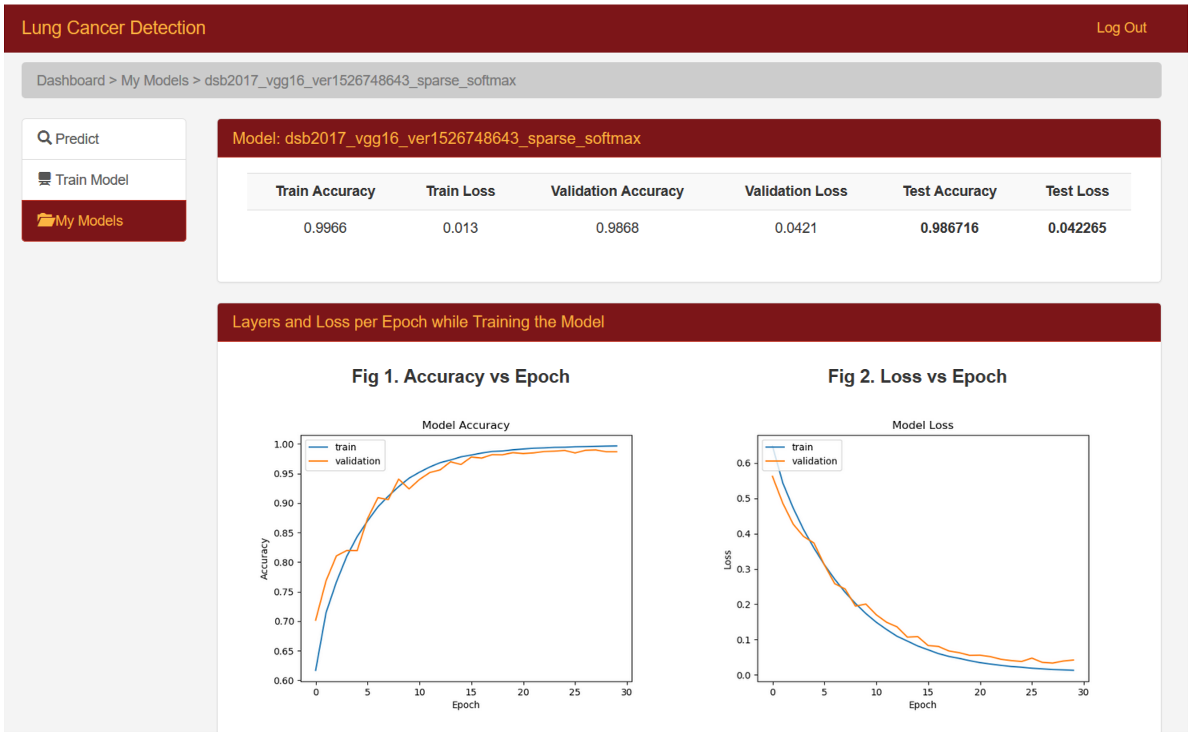
Learning Rate: 0.01

Network Architecture: VGG16

Figure 16: Train Model Page

The training data is automatically partitioned by the system into 3 sets: the training set, validation set, and the test set.

After the model is trained, the AI expert can view the details of the new model. These include the performance of the model: the training loss and accuracy, validation loss and accuracy, and test loss and accuracy. A graph of the accuracy and loss of the model attained per epoch when the model was being trained can also be viewed. An image that shows the layers of the model is also displayed.


Lung Cancer Detection

Log Out

Dashboard > My Models > dsb2017_vgg16_ver1526748643_sparse_softmax

Predict

Train Model

My Models

Model: dsb2017_vgg16_ver1526748643_sparse_softmax

Train Accuracy	Train Loss	Validation Accuracy	Validation Loss	Test Accuracy	Test Loss
0.9966	0.013	0.9868	0.0421	**0.986716**	**0.042265**

Layers and Loss per Epoch while Training the Model

Fig 1. Accuracy vs Epoch

Fig 2. Loss vs Epoch

Figure 17: Model Details Page

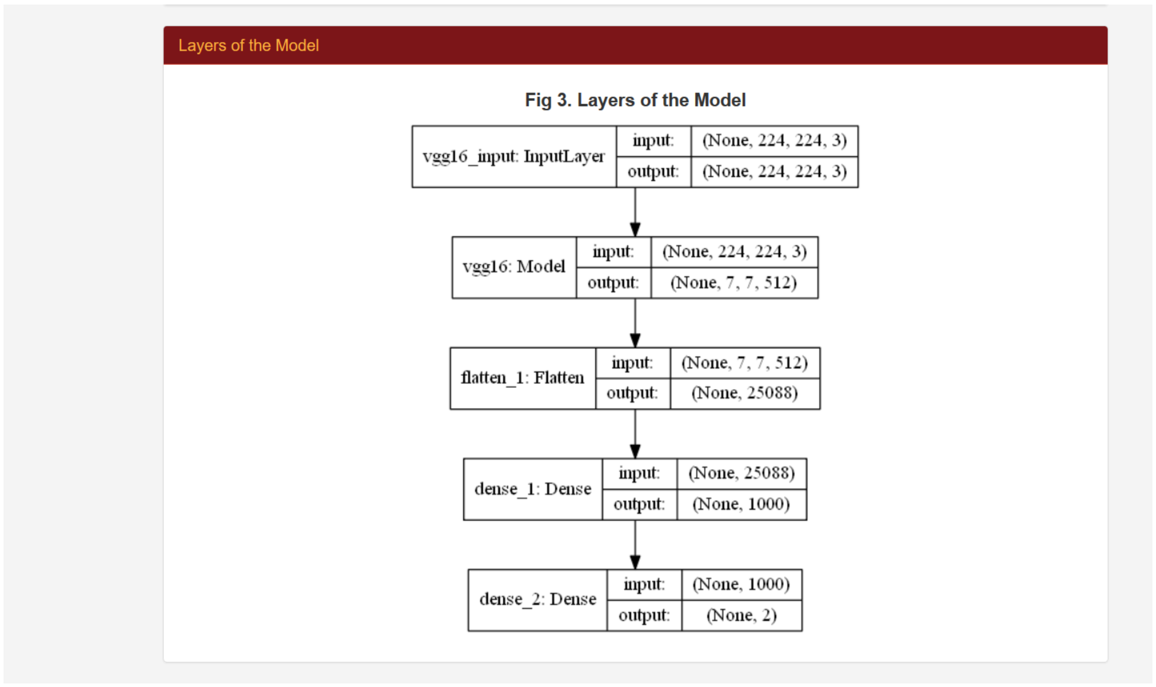
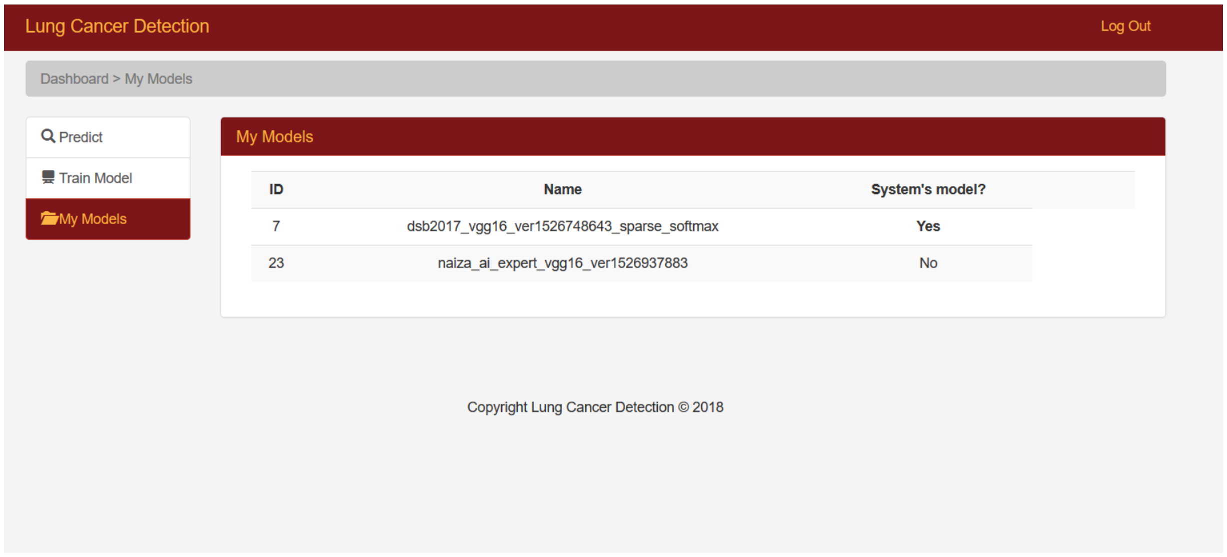

Layers of the Model

Fig 3. Layers of the Model

Layer		Shape
vgg16_input: InputLayer	input:	(None, 224, 224, 3)
	output:	(None, 224, 224, 3)
vgg16: Model	input:	(None, 224, 224, 3)
	output:	(None, 7, 7, 512)
flatten_1: Flatten	input:	(None, 7, 7, 512)
	output:	(None, 25088)
dense_1: Dense	input:	(None, 25088)
	output:	(None, 1000)
dense_2: Dense	input:	(None, 1000)
	output:	(None, 2)

Figure 18: Model Details Page (continued)

The AI expert can also view the list of models he/she has created. The page also displays whether the model is the model used by the system for prediction.



ID	Name	System's model?
7	dsb2017_vgg16_ver1526748643_sparse_softmax	Yes
23	naiza_ai_expert_vgg16_ver1526937883	No

Copyright Lung Cancer Detection © 2018

Figure 19: List of Models Created by the AI Expert

VI. Discussions

The system developed is a clinical decision support system for lung cancer diagnosis. It is a web application that allows users (doctors and AI experts) to upload a patient's CT scan that would then be classified as either positive or negative for lung cancer. The system utilizes concepts in deep learning to develop a model that can learn and classify the CT scans of a patient. Specifically, the model is composed of convolutional neural networks (CNN) and fully-connected layers for learning and classification. CNNs have been found to be best tools for image classification problems.

The network architecture used was the VGG16 network architecture [30]. This was the network architecture used by the team that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) on 2014. The ILSVRC is an annual competition that evaluates algorithms for object detection and image classification at large scale. In this competition, the algorithms are tasked at detecting and classifying images (1000 classes).

The system model used a pretrained model with a VGG16 network architecture as the convolutional base. Pretrained means the model contains weights learned from a different dataset. In this case, the dataset is the ImageNet database. This concept of using a pretrained model on a different dataset is called *transfer learning.* A study has shown that using a pretrained model (e.g. trained on the ImageNet dataset) on a medical image dataset has led to a good performance [31]. Because of this, a similar idea was employed for the system developed for this study. The medical image dataset used is the Kaggle Data Science Bowl 2017 dataset. It contains low-dose thoracic CT scans of adult, American patients.

The model obtained a performance of 98.6716% test accuracy and a 0.042265 test loss. This is significanly higher compared to the current state of the art models. One possible explanation for this is on how the training data was split. Because the slices are fed to the network individually, there is a chance that a slice from the CT scan of a patient, will be put in the training set, another will be put in the validation set, and another will be put in the test set. However, a single person's CT scan slices will have the same features. Because of this, there is a chance that the model has overfit on the training data. While it has not seen the test set during training, it has seen a slice of the same patient. This could be the reason why the model has obtained a high performance, but does sometimes does poorly when performing prediction.

VII. Conclusions

The developed Clinical Decision Support System for Lung Cancer Diagnosis is a web application that predicts whether or not a patient has lung cancer based on the patient's low-dose thoracic computer tomography (LDCT) scans. The system does this by using a trained model made up of convolutional and fully-connected layers. The Kaggle Data Science Bowl 2017 (KDSB 2017) dataset was used to train the model.

The current system model was trained using the Keras library for approximately 8.5 hours. The dataset consists of 136,000 images (95,200 for the training set, 20,400 each for the test and validation set).

The system is a web application that was built using the Python programming language, the Flask web framework, and the MySQL database.

Once logged in, a user is presented with a page that allows him/her to upload a single patient's CT scans and obtain a prediction of whether or not the patient has lung cancer.

Besides classification, the system also allows an AI expert to train a new model. He/she can specify values of certain hyperparamters such as the number of epochs, batch size, the optimizer, and the learning rate of the optimizer. This gives an AI expert the flexibility to experiment with different sets of data and hyperparameters to obtain a model with better performance.

The current performance of the system model is much higher than the performance of the current state of the art models found in recent research papers. The system model's prediction performance however does not seem to match the high test accuracy reported. One of the possible reasons is that overfitting occured. As such, further training needs to be done before the system can be deployed for use by doctors.

VIII. Recommendations

There are implementations and processes that could be improved but the current ones used were favored because of knowledge, time, and resource constraints.

It is highly suggested that a more complex preprocessing of the dataset is performed on the training data. Because the presence of lung nodules is the indicator for the possibility of lung cancer, it is best to first perform segmentation and feed only the ROI of the CT scan to the model. It would be best to first work on a segmenter using a different training data, such as the LUng Nodule Analysis 2016 dataset, because this provides the necessary data for segmentation. The trained segmentation model can then be used with the Kaggle Data Science Bowl 2017 dataset.

The system's model has a convolutional base made up of convolutional neural networks (CNNs) that accept 2D images. Hence, the three-dimensionality of a patient's CT scans is disregarded because each slice is fed to the network individually. State of the art models for lung cancer detection use convolutional neural networks that accept 3D input (4D tensors). These 3D CNNS have been found to have better performance.

Currently the system saves every model trained. It would be an improvement if the AI expert can also retrain a created model. This would significantly lessen the disk space utilized by the system in the server.

A progress bar can be implemented to provide some visual feedback to the user regarding how close the system is to finishing its task of training the model and/or of classifying a patient's CT scans.

The prediction page can be enhanced by showing a gallery of images and each image's classification as to whether or not it is positive or negative for lung cancer.

The retrain feature can be improved by allowing the AI expert to specify the number of fully-connected layers of the classifier as well as the number of units per fully-connected layer.

IX. Bibliography

[1] World Health Organization, "Cancer: Fact Sheet (February 2017)." http://www.who.int/mediacentre/factsheets/fs297/en/. Accessed: 2017-11-24.

[2] Philippine Council for Health Research and Development, "Lung Cancer." http://www.pchrd.dost.gov.ph/index.php/news/library-health-news/3540-lung-cancer. Accessed: 2017-10-10.

[3] Centers for Disease Control and Prevention, "Cancer Screening Tests." https://www.cdc.gov/cancer/dcpc/prevention/screening.htm. Accessed: 2017-11-30.

[4] National Lung Screening Trial Research Team, D. R. Aberle, A. M. Adams, C. D. Berg, W. C. Black, J. D. Clapp, R. M. Fagerstorm, I. F. Gareen, C. Gatsonis, P. M. Marcus, and J. D. Sicks, "Reduced lung-cancer mortality with low-dose computed tomographic screening.," *New England Journal of Medicine*, vol. 365, no. 5, pp. 395–409, 2011.

[5] U.S. Preventive Services Task Force, "Lung Cancer: Screening." https://www.uspreventiveservicestaskforce.org/Page/Document/UpdateSummaryFinal/lung-cancer-screening. Accessed: 2017-11-30.

[6] E. S. Berner and T. J. L. Lande, "Overview of Clinical Decision Support Systems," in *Clinical Decision Support Systems: Theory and Practice, 3rd Edition* (E. S. Berner, ed.), ch. 1, pp. 1–17, Switzerland: Springer International Publishing Switzerland, 2016.

[7] M. Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems, 2nd Edition*. Essex, England: Pearson Education Limited, 2002.

[8] N. Buduma and N. Locascio, *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms.* California, United States of America: O'Reilly Media, Inc., 2017.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[10] J. Patterson and A. Gibson, *Deep Learning: A Practitioner's Approach.* CA, United States of America: OReilly Media, Inc., 2017.

[11] Q. Song, L. Zhao, X. Luo, and X. Dou, "Using Deep Learning for Classification of Lung Nodules on Computed Tomography Images," *Journal of Healthcare Engineering*, vol. 2017, 2017. [Article ID 8314740, 7 pages].

[12] J. Lawrence, J. Malmsten, A. Rybka, D. A. Sabol, and K. Triplin, "Comparing TensorFlow Deep Learning Performance Using CPUs, GPUs, Local PCs and Cloud," *Proceedings of Student-Faculty Research Day, CSIS, Pace University*, 2017.

[13] A. Mahale, C. Rawool, D. Tolani, D. Bathija, and K. Jewani, "SVM classifier based CAD system for Lung Cancer Detection," *International Journal Of Engineering And Computer Science*, vol. 6, no. 5, pp. 21336–21342, 2017.

[14] A. Trivedi and P. Shukla, "Lung Cancer Diagnosis by Hybrid Support Vector Machine," *International Conference on Smart Trends for Information Technology and Computer Communications*, pp. 177–187, 2016.

[15] A. Abdolmanafi, L. Duong, N. Dahdah, and F. Cheriet, "Deep feature learning for automatic tissue classification of coronary artery using optical coherence tomography," *Biomedical Optics Express*, vol. 8, no. 2, pp. 1203–1220, 2017.

[16] P. Rao, N. A. Pereira, and R. Srinivasan, “Convolutional neural networks for lung cancer screening in computed tomography (CT) scans,” *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pp. 489–493, 2016.

[17] W. Li, P. Cao, D. Zao, and J. Wang, “Pulmonary Nodule Classification with Deep Convolutional Neural Networks on Computed Tomography Images,” *Computational and Mathematical Methods in Medicine*, vol. 2016, 2016. [Article ID 6215085, 7 pages].

[18] Q. Dou, H. Chen, L. Yu, J. Qin, and P.-A. Heng, “Multilevel Contextual 3D CNNs for False Positive Reduction in Pulmonary Nodule Detection,” *EEE Transactions on Biomedical Engineering*, vol. 64, no. 7, pp. 1558–1567, 2017.

[19] A. M. Rossetto and W. Zhou, “Deep Learning for Categorization of Lung Cancer CT Images,” *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pp. 272–273, 2017.

[20] A. Teramoto, T. Tsukamoto, Y. Kiriyama, and H. Fujita, “Automated Classification of Lung Cancer Types from Cytological Images Using Deep Convolutional Neural Networks,” *BioMed Research International*, vol. 2017, 2017. [Article ID 4067832, 6 pages].

[21] W. Alakwaa, M. Nassef, and A. Badr, “Lung Cancer Detection and Classification with 3D Convolutional Neural Network (3D-CNN),” *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 8, no. 8, 2017.

[22] S. Falk and C. Williams, *Lung Cancer: The Facts, 3rd Edition.* New York, United States of America: Oxford University Press, 2010.

[23] Centers for Disease Control and Prevention, "What Screening Tests Are There?." https://www.cdc.gov/cancer/lung/basic_info/screening.htm. Accessed: 2017-12-1.

[24] NHS Choices, "CT Scan." https://www.nhs.uk/conditions/CT-scan/Pages/Introduction.aspx. Accessed: 2017-10-25.

[25] Swedish Medical Center, "Low-Dose CT Scan for Lung Cancer Screening ." https://www.swedish.org/services/thoracic-surgery/our-services/lung-cancer-screening-program/low-dose-ct-scan-for-lung-cancer-screening. Accessed: 2017-12-1.

[26] OpenClinical, "Clinical Decision Support Systems." http://www.openclinical.org/dss.html. Accessed: 2017-10-25.

[27] The Office of the National Coordinator for Health Information Technology, "Clinical Decision Support (CDS)." https://www.healthit.gov/policy-researchers-implementers/clinical-decision-support-cds. Accessed: 2017-10-25.

[28] DICOM Library, "About DICOM." https://www.dicomlibrary.com/dicom/. Accessed: 2017-10-28.

[29] F. Chollet, "Keras." https://github.com/fchollet/keras. Accessed: 2017-11-26.

[30] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *International Conference on Learning Representations 2015*, 2015.

[31] R. Paul, S. H. Hawkins, Y. Balagurunathan, M. B. Schabath, R. J. Gillies, L. O. Hall, , and D. B. Goldgof, "Deep Feature Transfer Learning in Combination with

Traditional Features Predicts Survival Among Patients with Lung Adenocarcinoma," *Tomography*, vol. 2, no. 4, pp. 388–395, 2016.

X. Appendix

A. Source Code

launch.py

```
from app import create_app, db
from app.models import User, Role, TrainedModel,
        ModelMetricsPlot
from flask_script  import Manager, Server, Shell
from flask_migrate import MigrateCommand, Migrate

app = create_app('development')
manager = Manager(app)
migrate = Migrate(app, db)


def _make_context():
    return dict(app=app, db=db, User=User, Role=Role,
            TrainedModel=TrainedModel, ModelMetricsPlot=
            ModelMetricsPlot)


if __name__ == '__main__':
    manager.add_command('db', MigrateCommand)
    manager.add_command('runserver', Server(host='localhost',
            port=5000, use_debugger=True))
    manager.add_command('shell', Shell(make_context=
            _make_context))

    manager.run()
```

config.py

```
import os

BASE_DIR = os.path.abspath(os.path.dirname(__file__))
APP_DIR = os.path.join(BASE_DIR, 'app')
STATIC_DIR = os.path.join(APP_DIR, 'static')
MODEL_DIR = os.path.join(BASE_DIR, 'models')
INFERENCE_DIR = os.path.join(STATIC_DIR, 'inference')
DEEP_LEARNING_DIR = os.path.join(BASE_DIR, '
        deep_learning')
MODEL_METRICS_GRAPH_DIR = os.path.join(BASE_DIR, '
        model_metrics_graphs')


class Config:
    TEMPLATES_AUTO_RELOAD = True

    SQLALCHEMY_DATABASE_URI = 'mysql://sp_admin:
            sp_admin@localhost/sp'
    SECRET_KEY = os.environ.get('SECRET_KEY') or '
            pz9w1sec9tbhyjL08NwNDNIlsG9Cz7Pf'
    SQLALCHEMY_COMMIT_ON_TEARDOWN = True
    SQLALCHEMY_TRACK_MODIFICATIONS = False

    STATIC_DIR = STATIC_DIR
    MODEL_DIR = MODEL_DIR
    INFERENCE_DIR = INFERENCE_DIR


class DevelopmentConfig(Config):
    DEBUG = True


class TestingConfig(Config):
    pass


class ProductionConfig(Config):
    pass


config = {
    'development': DevelopmentConfig,
    'testing': TestingConfig,
    'production': ProductionConfig,


    'default': DevelopmentConfig
}
```

app/models.py

```
from app import db

from itsdangerous import TimedJSONWebSignatureSerializer as
        Serializer

from flask_login import AnonymousUserMixin, UserMixin

from werkzeug.security import check_password_hash,
        generate_password_hash

import app


class Role(db.Model):
    __tablename__ = 'roles'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)

    users = db.relationship('User', backref='role', lazy='
            dynamic')

    def __repr__(self):
        return '<Role %r>' % self.name


class User(db.Model, UserMixin):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True, index=
            True, nullable=False)
    password_hash = db.Column(db.String(128))
    role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))
    confirmed = db.Column(db.Boolean, default=False)

    @property
    def password(self):
        raise AttributeError('password is not a readable
                attribute')

    @password.setter
    def password(self, password):
        self.password_hash = generate_password_hash(password)

    def verify_password(self, password):
        return check_password_hash(self.password_hash,
                password)

    def generate_confirmation_token(self, expiration=3600):
        s = Serializer(app.config['SECRET_KEY'], expiration)
        return s.dumps({'confirm': self.id})

    def confirm(self, token):
        s = Serializer(app.config['SECRET_KEY'])
        try:
            data = s.loads(token)
        except:
            return False

        if data.get('confirm') != self.id:
            return False
        self.confirmed = True
        db.session.add(self)
        return True

    def __repr__(self):
        return '<User %r>' % self.username


class TrainedModel(db.Model):
    __tablename__ = 'models'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True, index=True
            , nullable=False)
    train_acc = db.Column(db.Float, nullable=False)
    train_loss = db.Column(db.Float, nullable=False)
    validation_acc = db.Column(db.Float, nullable=False)
    validation_loss = db.Column(db.Float, nullable=False)
    test_acc = db.Column(db.Float, nullable=True)
    test_loss = db.Column(db.Float, nullable=True)
    # Every model is associated with an AI expert
    username = db.Column(db.String(64), db.ForeignKey('users.
            username'), nullable=False)
    used_by_doctors = db.Column(db.Boolean(), default=False)
    display_in_site = db.Column(db.Boolean(), default=False)


class ModelMetricsPlot(db.Model):
    __tablename__ = 'model_metrics_plots'
    id = db.Column(db.Integer, primary_key=True)
    type = db.Column(db.String(20), nullable=False) # 'loss' or
            'acc'
    filename = db.Column(db.String(100), nullable=False)
    model_name = db.Column(db.String(64), db.ForeignKey('
           models.name'))

    def __repr__(self):
        return '<graph of {} of model {}>'.format(self.type,
               self.model_name)
```

app/main/__init__.py

```
from flask import Blueprint

main = Blueprint('main', __name__, static_folder='static')

from . import views, errors
```

app/main/errors.py

```
from flask import render_template
from . import main


# When writing error handlers inside a blueprint, if the
       errorhandler decorator is used,
# the handler will only be invoked for errors that originate in
       the blueprint.
# To install application-wide error handlers, the
       app_errorhandler must be used instead.


@main.app_errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404


@main.app_errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500
```

app/main/forms.py

```
from flask_wtf import FlaskForm
from wtforms import BooleanField, DecimalField, FileField,
       SelectField, StringField, SubmitField, PasswordField, \
    validators, ValidationError
from ..models import User


class LoginForm(FlaskForm):
    username = StringField('Username', [validators.
           DataRequired(), validators.Length(min=1, max=64)])
    password = PasswordField('Password', [validators.
           DataRequired()])
    remember_me = BooleanField('Keep me logged in')
    submit = SubmitField('Log In')


class RegistrationForm(FlaskForm):
    # TODO: Make email the unique identifier of a user rather
           than a username.
    # TODO: Add email confirmation

    # email = StringField('Email', [validators.DataRequired('
           This field is required.'),
    #                              validators.Email("Please
           enter a valid email address."),
    #                              validators.Length(min=1,
           max=254)]) # max length was googled

    role = SelectField('Type of User', [validators.
           DataRequired('This field is required.')],
                       choices=[('AI Expert', 'AI Expert'), ('
                              Doctor', 'Doctor')])

    username = StringField('Username', [validators.
           DataRequired('This field is required.'),
                                        validators.Length(min
                                               =1, max=64),
                                        validators.Regexp('^[A
                                               -Za-z][A-Za-z0
                                               -9_.]*$', 0, '
                                               Usernames must
                                               have only '
                                               '
                                               letters
                                               ,
                                               numbers
                                               ,
                                               dots
                                               ,
    password = PasswordField('Password', [validators.
           DataRequired('This field is required.'),
                                          validators.EqualTo('
                                                 password2',
                                                 message='
                                                 Passwords must
                                                  match.')])
    password2 = PasswordField('Confirm Password', [validators.
           DataRequired('This field is required.')])
    submit = SubmitField('Register')

    @staticmethod
    def validate_username(self, field):
        if User.query.filter_by(username=field.data).first():
            raise ValidationError('Username already in use.')


class TrainingDataForm(FlaskForm):
    training_data = FileField('Training Data', [validators.
           DataRequired()], render_kw={'multiple': True})
    training_data_csv = FileField('Training Data Labels (CSV
           File):')
    validation_data = FileField('Validation Data', [validators.
           DataRequired()], render_kw={'multiple': True})
    validation_data_csv = FileField('Validation Data Labels (
           CSV file):')
    learning_rate = DecimalField('Learning Rate')
    epoch = DecimalField('Epoch')
    # model_name = StringField('Model Name', [validators.
           DataRequired()])
    role = SelectField('Optimizer', [validators.DataRequired('
           This field is required.')],
                       choices=[('Stochastic Gradient Descent',
                              'Stochastic Gradient Descent'), ('
                              RMSprop', 'RMSprop'),
                                 ('Adam', 'Adam'), ('AdaGrad',
                                        'AdaGrad')])
    submit = SubmitField('Retrain')
```

app/main/views.py

```
from deep_learning.util import convert_dicoms_to_pngs, predict,
       retrain
from deep_learning.model import get_pretrained_model

from flask import render_template, redirect, url_for, request,
       flash, make_response
from flask_login import current_user, login_required,
       logout_user, login_user

from . import main
from .forms import LoginForm, RegistrationForm,
       TrainingDataForm

from .. import db, login_manager
from ..models import User, Role, TrainedModel,
       ModelMetricsPlot
from ..util import allowed_ct_scan_format, allowed_label_format,
        \
    AI_EXPERT_LINKS, DOCTOR_LINKS,
           SITE_ADMIN_LINKS, LinkIndex, set_link_active,
           get_link_active, UserRoleID, \
    get_network_architecture_string, get_optimizer_string

from config import MODEL_METRICS_GRAPH_DIR

import os


@login_manager.user_loader
def load_user(user_id):
    return User.query.get((int(user_id)))


@main.route('/', methods=['GET', 'POST'])
def index():
    if current_user.username == 'Guest':
        # return render_template('index.html', title='Home')
        return redirect(url_for('main.login'))

    file_urls = []
    has_uploaded_data = False
    benign_ave = 0
    malignant_ave = 0

    if request.method == 'POST':
        # check if the POST request has the file part
        if ' files ' not in request. files :
            flash ('No file part')
            return redirect (request. url)
        uploaded_files = request. files . getlist (' files ')

        # NOTE: If user does not select file , browser also
              submits an empty part without filename
        if len( uploaded_files ) == 0:
            flash ('No selected CT scans')
            return redirect (request. url)
        if uploaded_files :
            # NOTE: Check if files uploaded are allowed
            all_files_okay = True
            for file in uploaded_files :
                if not allowed_ct_scan_format( file .filename):
                    all_files_okay = False
                    break

            if not all_files_okay :
                flash ('Uploaded non−dicom files')
                return redirect (request. url)

            dicoms = list( uploaded_files )
            # NOTE: mritopng only works with saved dicom
                  images so write the images to a folder (in the
                  app's dir) first
            converted_pngs_dir, time_as_version =
                  convert_dicoms_to_pngs(dicoms)

            # NOTE: PREDICTION HERE!!
            benign_ave, malignant_ave = predict(
                  converted_pngs_dir)
            benign_ave ∗= 100
            malignant_ave ∗= 100

            # NOTE: These lines are for rendering the CT scans
                  .
            has_uploaded_data = True
            old_uploaded_data = os. listdir (converted_pngs_dir)
            uploaded_data = [os.path.join(converted_pngs_dir,
                  png_data) for png_data in old_uploaded_data]

            for index, uploaded_img in enumerate(
                  uploaded_data):
                file_urls .append('/static/inference/' +
                      time_as_version + '_PNG/' +
                      old_uploaded_data[index])
            print( file_urls )

    path_to_go_to = ''
    links = []
    user_role_id = UserRoleID(current_user.role_id)
    if user_role_id == UserRoleID.AI_EXPERT:
        path_to_go_to = 'ai_expert_index.html'
        links = list (AI_EXPERT_LINKS)
    elif user_role_id == UserRoleID.DOCTOR:
        path_to_go_to = 'doctor_index.html'
        links = list (DOCTOR_LINKS)
    elif user_role_id == UserRoleID.SITE_ADMIN:
        path_to_go_to = 'site_admin_index.html'
        links = list (SITE_ADMIN_LINKS)
        print (' links :', links)
        set_link_active (links , LinkIndex.DASHBOARD)

        return render_template(path_to_go_to, links=links, title
              ='Predict', user_count=32, model_count=7)

    set_link_active (links , LinkIndex.PREDICT)

    return render_template(path_to_go_to, title ='Predict', links
          =links,
                           has_uploaded_data=
                                 has_uploaded_data, file_urls=
                                 file_urls ,
                           benign_ave=benign_ave,
                                 malignant_ave=malignant_ave)


@main.route('/register /', methods=['GET', 'POST'])
def register ():
    form = RegistrationForm()
    if form.validate_on_submit():
        new_user = User()
        new_user.username = form.username.data
        new_user.password = form.password.data
        new_user. role_id = Role.query. filter_by (name=form.role
              .data).first () .id
        db.session .add(new_user)
        print(new_user.id)

        flash ('You can now log in.')
        return redirect ( url_for ('main.login'))
    return render_template(' register .html', form=form, title='
          Register')


@main.route('/login/', methods=['GET', 'POST'])
def login ():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query. filter_by (username=form.username.
              data).first()
        if user is not None and user.verify_password(form.
              password.data):
            # print(user. is_authenticated )
            # print(user. is_active )
            # print(user.is_anonymous)
            # print(user. get_id ())
            login_user (user, form.remember_me.data)
            return redirect (request.args.get ('next') or url_for
                  ('main.index'))
        flash (' Invalid username or password')
    return render_template('login .html', form=form, title='Log
          In')


@login_required
@main.route('/logout/')
def logout():
    logout_user ()
    # flash("You have been logged out.")
    return redirect ( url_for ('main.index'))


@main.route('/my_models/', methods=['GET', 'POST'])
@login_required
def my_models():
    links = list (AI_EXPERT_LINKS)
    set_link_active (links , LinkIndex.MY_MODELS)

    username = current_user.username
    user_models = TrainedModel.query.filter_by(username=
          username, display_in_site=1).all()
    #user_models = TrainedModel.query.filter_by(username=
          username).all()

    return render_template('my_models.html', title='My Models
          ', user_models=user_models, links=links)


@main.route('/train/', methods=['GET', 'POST'])
@login_required
def train ():
    done_training = False
    graphs = []
    about_model = {}
    model = None

    if request.method == 'POST':
        # check if the POST request has the file part
        if 'dataset' not in request. files :
            flash ('No dataset was provided')
        if ' dataset_labels ' not in request. files :
            flash ('Please provide the labels of the dataset')
            return redirect (request. url)

        dataset = request. files . getlist ('dataset')

        # NOTE: If user does not select file , browser also
              submits an empty part without filename
        # if len(dataset) == 0: # NOTE: Not sure if request.
              files. getlist () returns a [] if it finds nothing so
               I won't use the pythonic "if not uploaded_files :"
        if not dataset:
            flash ('Dataset was not uploaded')
            return redirect (request. url)
        else :
            # Check if files uploaded are allowed
            all_dataset_scans_okay = True
            for file in dataset:
                if not allowed_ct_scan_format( file .filename):
                    all_dataset_scans_okay = False
                    break

            if not all_dataset_scans_okay :
                flash ('Uploaded a non−DICOM CT Scan')
                return redirect (request. url)
        labels = request.files['dataset_labels']
        if not allowed_label_format(labels.filename):
            flash('The file containing the labels of the dataset
                must be in CSV format.')
            return redirect(request.url)

        epochs = request.form.get('epochs')
        batch_size = request.form.get('batch_size')
        network_architecture = request.form.get('
            network_architecture')
        optimizer = request.form.get('optimizer')
        learning_rate = request.form.get('learning_rate')

        print('dataset:', len(dataset), 'scans', dataset)
        print('labels:', labels)
        print('epochs:', epochs)
        print('batch size:', batch_size)
        print('network architecture:', network_architecture)
        print('optimizer:', optimizer)
        print('learning rate:', learning_rate)

        # Get model from models table
        # model_name = TrainedModel.query.filter_by(
            used_by_doctors=1).first().name
        # model = get_pretrained_model(model_name)
        # print('model:', model)

        ai_expert_username = current_user.username
        model_name, acc_graph_filename, loss_graph_filename,
            layers_image_filename = retrain(
            ai_expert_username, dataset, labels, epochs,
            batch_size, network_architecture, optimizer,
            learning_rate)
        done_training = True

        '''
        acc_graph_filename = 'NEW_MODEL_ACCURACY.png'
        loss_graph_filename = 'NEW_MODEL_LOSS.png'
        layers_image_filename = 'NEW_LAYERS.png'
        '''

        graphs.append({
            'src': os.path.join('/static/model_metrics_graphs',
                layers_image_filename),
            'fig_title': 'Layers of the Model',
            'image_alt': 'visualization of the layers of the
                model',
        })

        graphs.append({
            'src': os.path.join('/static/model_metrics_graphs',
                acc_graph_filename),
            'fig_title': 'Accuracy vs Epoch',
            'image_alt': 'accuracy vs accuracy graph of the
                model',
        })

        graphs.append({
            'src': os.path.join('/static/model_metrics_graphs',
                loss_graph_filename),
            'fig_title': 'Loss vs Epoch',
            'image_alt': 'loss vs epoch graph of the model',
        })

        about_model['epochs'] = epochs
        about_model['batch_size'] = batch_size
        about_model['network_architecture'] =
            get_network_architecture_string(
            network_architecture)
        about_model['optimizer'] = get_optimizer_string(
            optimizer)

        print('done training')

        model = TrainedModel.query.filter_by(name=
            model_name).first()

    links = None
    if UserRoleID(current_user.role_id) == UserRoleID.
        AI_EXPERT:
        links = list(AI_EXPERT_LINKS)
        set_link_active(links, LinkIndex.RETRAIN_MODEL)

    form = TrainingDataForm()
    return render_template('train.html', title='Train Model',
        links=links, form=form, done_training=done_training,
        graphs=graphs, about_model=about_model, model=
        model)


@main.route('/models/')
@login_required
def models():
    links = list(SITE_ADMIN_LINKS)
    set_link_active(links, LinkIndex.MODELS)

    site_models = None
    link_active = LinkIndex(get_link_active(links))
    if link_active == LinkIndex.MODELS:
        site_models = TrainedModel.query.filter_by(
            display_in_site=0).all()
        #site_models = TrainedModel.query.all()
        print('site models:', site_models)

    return render_template('models.html', title='Models', links
        =links, models=site_models)


@main.route('/users/', methods=['GET', 'POST'])
@login_required
def users():
    links = list(SITE_ADMIN_LINKS)
    set_link_active(links, LinkIndex.USERS)

    site_users = None
    link_active = LinkIndex(get_link_active(links))
    if link_active == LinkIndex.USERS:
        site_users = User.query.all()

    return render_template('users.html', title='Users', links=
        links, users=site_users)


@main.route('/model/view/<int:id>/')
@login_required
def model(id):
    graphs = []
    the_model = TrainedModel.query.filter_by(id=id).first()
    layers_image_filename = ModelMetricsPlot.query.filter_by(
        model_name=the_model.name, type='layers').first().
        filename
    acc_graph_filename = ModelMetricsPlot.query.filter_by(
        model_name=the_model.name, type='acc').first().
        filename
    loss_graph_filename = ModelMetricsPlot.query.filter_by(
        model_name=the_model.name, type='loss').first().
        filename

    print(layers_image_filename)
    print(acc_graph_filename)
    print(loss_graph_filename)

    title = the_model.name[:-3]
    links = list(AI_EXPERT_LINKS)
    set_link_active(links, LinkIndex.MY_MODELS)

    graphs.append({
        'src': os.path.join('/static/model_metrics_graphs',
            layers_image_filename),
        'fig_title': 'Layers of the Model',
        'image_alt': 'visualization of the layers of the model',
    })

    graphs.append({
        'src': os.path.join('/static/model_metrics_graphs',
            acc_graph_filename),
        'fig_title': 'Accuracy vs Epoch',
        'image_alt': 'accuracy vs accuracy graph of the model',
    })

    graphs.append({
        'src': os.path.join('/static/model_metrics_graphs',
            loss_graph_filename),
        'fig_title': 'Loss vs Epoch',
        'image_alt': 'loss vs epoch graph of the model',
    })

    return render_template('model.html', title=title, links=
        links, graphs=graphs, model=the_model)

@main.route('/model/fine-tune/<int:id>/')
@login_required
def fine_tune_model(id):
    title = 'Fine-Tune Model'
    links = list(AI_EXPERT_LINKS)
    set_link_active(links, LinkIndex.MY_MODELS)
    return render_template('fine_tune_model.html', title=title,
        links=links)
```

app/static/style.css

```
a.model-view {

}
```

```
a.model−fine−tune {
    color: #ffc107 !important;
}

a.model−evaluate {
    color: green !important;
}

a.model−delete {
    color: red !important;
}

body {
    background: #f4f4f4;
}

@−moz−document url−prefix() {
  fieldset  { display: table−cell; }
}

.number{
    width: 4em;
}

/∗ Navbar ∗/
.navbar {
    margin−bottom: 10px;
    border−radius: 0;
}

.navbar−nav > li > a, .navbar−brand {
    /∗padding−top: 6px !important;∗/
    padding−bottom: 0 !important;
    height: 33px;
}

.navbar−default {
    /∗background−color: #e74c3c;∗/
    background−color: #7C1518;
    border−color: #7C1518;
}

.navbar−default .navbar−brand {
    color: #FBB542;
}

.navbar−default .navbar−brand:hover,
.navbar−default .navbar−brand:focus {
    color: #fcd089;
}

.navbar−default .navbar−text {
    color: #FBB542;
}

.navbar−default .navbar−nav > li > a {
    color: #FBB542;
}

.navbar−default .navbar−nav > li > a:hover,
.navbar−default .navbar−nav > li > a:focus {
    color: #fcd089;
}

.navbar−default .navbar−nav > .active > a,
.navbar−default .navbar−nav > .active > a:hover,
.navbar−default .navbar−nav > .active > a:focus {
    color: #fcd089;
    background−color: #c0392b;
}

.navbar−default .navbar−nav > .open > a,
.navbar−default .navbar−nav > .open > a:hover,
.navbar−default .navbar−nav > .open > a:focus {
    color: #fcd089;
    background−color: #c0392b;
}

.navbar−default .navbar−toggle {
    border−color: #c0392b;
}

.navbar−default .navbar−toggle:hover,
.navbar−default .navbar−toggle:focus {
    background−color: #c0392b;
}

.navbar−default .navbar−toggle .icon−bar {
    background−color: #FBB542;
}

.navbar−default .navbar−collapse,
.navbar−default .navbar−form {
    border−color: #FBB542;
}

.navbar−default .navbar−link {
    color: #FBB542;
}

.navbar−default .navbar−link:hover {
    color: #fcd089;
}

/∗ Custom ∗/
.main−color−bg {
    /∗background−color: #e74c3c !important;∗/
    background−color: #7C1518 !important;
    border−color: #c0392b !important;
    color: #FBB542 !important;
}

.dash−box {
    text−align: center;
}

/∗ Header ∗/
#header {
    background: #333333;
    color: #FBB542;
    padding−bottom: 10px;
    margin−bottom: 15px;
}

#header .create {
    padding−top: 20px;
}

/∗ Breadcrumb ∗/
.breadcrumb {
    background: #cccccc;
    color: #FBB542;
}

.breadcrumb a {
    color: #333333;
}

/∗ Progress Bars ∗/
.progress−bar {
    background: #333333;
    color: # ffffff ;
}

/∗ Footer ∗/
#footer {
    background−color: #f4f4f4;
    color: #333333;
    text−align: center;
    padding: 30px;
    margin−top: 30px;
}

@media (max−width: 767px) {
    .navbar−default .navbar−nav .open .dropdown−menu > li
          > a {
        color: #ecf0f1;
    }

    .navbar−default .navbar−nav .open .dropdown−menu > li
          > a:hover,
    .navbar−default .navbar−nav .open .dropdown−menu > li
          > a:focus {
        color: #ffbbbc;
    }

    .navbar−default .navbar−nav .open .dropdown−menu > .
          active > a,
    .navbar−default .navbar−nav .open .dropdown−menu > .
          active > a:hover,
    .navbar−default .navbar−nav .open .dropdown−menu > .
          active > a:focus {
        color: #ffbbbc;
        background−color: #c0392b;
    }
}


/∗ CT SCANS GALLERY ∗/
div.gallery {
    border: 1px solid #ccc;
}
```

```
div.gallery:hover {
    border: 1px solid #777;
}

div.gallery img {
    width: 100%;
    height: auto;
}

div.desc {
    padding: 15px;
    text-align: center;
}

* {
    box-sizing: border-box;
}

.responsive {
    padding: 0 6px;
    float: left;
    width: 24.99999%;
}

@media only screen and (max-width: 700px) {
    .responsive {
        width: 49.99999%;
        margin: 6px 0;
    }
}

@media only screen and (max-width: 500px) {
    .responsive {
        width: 100%;
    }
}

.clearfix:after {
    content: "";
    display: table;
    clear: both;
}
```

app/templates/ai_expert_index.html

```
{% extends 'dashboard.html' %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block breadcrumb_content %}
    Dashboard > Predict
{% endblock %}

{% block panel_title_content %}
    Predict
{% endblock %}

{% block panel_body_content %}
    <div class="col-md-6">
        <form method="post" enctype="multipart/form-data
                ">
            <div class="form-group">
                <!--Browse<input type="file" name="files"
                        multiple="" style="display: none;">-->
                <input type="file" class="form-control-file"
                        name="files" multiple="">
            </div>
            <button type="submit" class="btn btn-primary">
                    Upload CT Scans</button>

        </form>
    </div>
    <div class="col-md-6">
    </div>
{% endblock %}


{% block more_panels_area %}
    {% if has_uploaded_data %}
        <div class="panel panel-default">
            <div class="panel-heading main-color-bg">
                <h3 class="panel-title">Result</h3>
            </div>
            <div class="panel-body">
                <table class="table table-striped table-hover
                        ">
                    <tr>
                        <th class="text-center">Benign</th>
                        <th class="text-center">Malignant</
                                th>
                    </tr>
                    <tr>
                        <td class="text-center">{{ benign_ave
                                }}%</td>
                        <td class="text-center">{{
                                malignant_ave }}%</td>
                    </tr>
                </table>
            </div>
        </div>
    {% endif %}
{% endblock %}
```

app/templates/base.html

```
{% extends 'bootstrap/base.html' %}

{% block styles %}
    {{ super() }}
    <link rel="stylesheet" href="{{ url_for('main.static',
            filename='style.css') }}">
{% endblock %}

{% block title %} {{ title }} | Lung Cancer Detection {%
        endblock %}

{% block navbar %}
    <nav class="navbar navbar-default">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle
                        collapsed" data-toggle="collapse" data-
                        target="#navbar"
                          aria-expanded="false" aria-controls="
                                navbar">
                    <span class="sr-only">Toggle navigation
                            </span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="{{ url_for('
                        main.index') }}">Lung Cancer Detection
                        </a>
            </div>
            <div id="navbar" class="collapse navbar-collapse
                    ">
                {% if current_user.is_authenticated %}
                    <ul class="nav navbar-nav navbar-right
                            ">
                        <li><a href="{{ url_for('main.logout')
                                }}">Log Out</a></li>
                    </ul>
                {% else %}
                    <ul class="nav navbar-nav navbar-right
                            ">
                        <li><a href="{{ url_for('main.register')
                                }}">Register</a></li>
                        <li><a href="{{ url_for('main.login')
                                }}">Log In</a></li>
                    </ul>
                {% endif %}
            </div>
        </div>
    </nav>
{% endblock %}

{% block content %}
    {% block header_content %}
    {% endblock %}

    <div class="container">
        {% for message in get_flashed_messages() %}
            <div class="alert alert-warning">
                <button type="button" class="close" data-
                        dismiss="alert">x</button>
                {{ message }}
            </div>
        {% endfor %}
    </div>

    {% block page_content %}
    {% endblock %}

    <footer id="footer">
        <p>Copyright Lung Cancer Detection &copy; 2018</p
                >
    </footer>
{% endblock %}

{% block scripts %}
    {{ super() }}
    <!-- Menu Toggle Script -->
    <script>
```

```
        $("#menu−toggle").click(function (e) {
            e.preventDefault();
            $("#wrapper").toggleClass("toggled");
        });
    </script>
{% endblock %}
```

app/templates/dashboard.html

```
{% extends 'base.html' %}

{% block header_content %}

    <section id="breadcrumb">
        <div class="container">
            <ol class="breadcrumb">
                <li class="active">
                    {% block breadcrumb_content %}
                        Dashboard
                    {% endblock %}
                </li>
            </ol>
        </div>
    </section>
{% endblock %}

{% block page_content %}
    <section id="main">
        <div class="container">
            <div class="row">
                <div class="col−md−2"> <!−− sidebar −−>
                    <div class="list−group">
                        {% for l in links %}
                            {% if l. is_active %}
                                <a href="{{ url_for(l.href) }}"
                                    class="list−group−item
                                   active main−color−bg
                                   "><span class="
                                   glyphicon glyphicon−{{ l.
                                   icon }}" aria−hidden="
                                   true"></span> {{ l.title
                                   }}</a>
                            {% else %}
                                <a href="{{ url_for(l.href) }}"
                                    class="list−group−item
                                   "><span class="
                                   glyphicon glyphicon−{{ l.
                                   icon }}" aria−hidden="
                                   true"></span> {{ l.title
                                   }}
                                </a>
                            {% endif %}
                        {% endfor %}
                        {% block other_links %}
                        {% endblock %}
                    </div>
                </div>

                <div class="col−md−10"> <!−− main area
                        −−>
                    <div class="panel panel−default">
                        <div class="panel−heading main−color
                                −bg">
                            <h3 class="panel−title">
                                {% block panel_title_content %}
                                {% endblock %}

                            </h3>
                        </div>
                        <div class="panel−body">
                            {% block panel_body_content %}
                            {% endblock %}
                        </div>
                    </div>

                    {% block more_panels_area %}
                    {% endblock %}
                </div>
            </div>
        </div>
    </section>
{% endblock %}
```

app/templates/doctor_index.html

```
{% extends 'dashboard.html' %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block breadcrumb_content %}
    Dashboard > Predict
{% endblock %}

{% block panel_title_content %}
    Predict
{% endblock %}

{% block panel_body_content %}
    <div class="col−md−6">
        <form method="post" enctype="multipart/form−data
                ">
            <div class="form−group">
                <input type="file" class="form−control−file"
                        name="files" multiple="">
            </div>
            <button type="submit" class="btn btn−primary">
                    Upload CT Scans</button>

        </form>
    </div>
    <div class="col−md−6">
    </div>
{% endblock %}


{% block more_panels_area %}
    {% if has_uploaded_data %}
        <div class="panel panel−default">
            <div class="panel−heading main−color−bg">
                <h3 class="panel−title">Result</h3>
            </div>
            <div class="panel−body">
                <table class="table table−striped table−hover
                        ">
                    <tr>
                        <th class="text−center">Positive</th
                                >
                        <th class="text−center">Negative</th
                                >
                    </tr>
                    <tr>
                        <td class="text−center">{{ benign_ave
                                }}%</td>
                        <td class="text−center">{{
                                malignant_ave }}%</td>
                    </tr>
                </table>
            </div>
        </div>
    {% endif %}
{% endblock %}
```

app/templates/index.html

```
{% extends 'base.html' %}

{% block styles %}
    {{ super() }}
    <style>
        .center {
            display: block;
            margin−left: auto;
            margin−right: auto;
            width: 50%;
    }
    </style>

{% endblock %}

{% block page_content %}
    <div class="container">

        <div class="row">
            <div class="col−md−6" class="text−center">
                <img src="{{ url_for('main.static', filename='
                        upseal.png') }}" alt="official UP seal"
                        class="center">
            </div>
            <div class="col−md−6" style="padding−top: 100
                    px;">
                Hello
            </div>
        </div>
    </div>
{% endblock %}
```

app/templates/login.html

```
{% extends 'base.html' %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block page_content %}
    <div class="container">
```

```
        <div class="row">

            <div class="col-md-4">
            </div>

            <div class="col-md-4">
                <div class="page-header">
                    <h1 align="center">Log In</h1>
                </div>
                {{ wtf.quick_form(form) }}
            </div>

            <div class="col-md-4">
            </div>
        </div>
    </div>
{% endblock %}
```

app/templates/logout.html

```
{% extends 'base.html' %}
```

app/templates/model.html

```
{% extends 'dashboard.html' %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block breadcrumb_content %}
    Dashboard > My Models > {{ title }}
{% endblock %}

{% block panel_title_content %}
    Model: {{ title }}
{% endblock %}

{% block panel_body_content %}
    <div class="col-md-12 table-responsive">
        <table class="table table-striped table-hover text-
              center">
            <tr>
                <th class="text-center">Train Accuracy</th>
                <th class="text-center">Train Loss</th>
                <th class="text-center">Validation Accuracy
                      </th>
                <th class="text-center">Validation Loss</th>
                {% if model.test_acc %}<th class="text-center
                      ">Test Accuracy</th>{% endif %}
                {% if model.test_loss %}<th class="text-
                      center">Test Loss</th>{% endif %}
            </tr>
            <tr><td>{{ model.train_acc }}</td>
            <td>{{ model.train_loss }}</td>
            <td>{{ model.validation_acc }}</td>
            {% if model.test_acc %}
            <td>{{ model.validation_loss }}</td>
            <td><strong>{{ model.test_acc }}</strong></td
                >
            <td><strong>{{ model.test_loss }}</strong></td
                ></tr>
            {% else %}
            <td>{{ model.validation_loss }}</td></tr>
            {% endif %}
        </table>
    </div>
{% endblock %}

{% block more_panels_area %}
    <div class="panel panel-default">
        <div class="panel-heading main-color-bg">
            <h3 class="panel-title">Layers and Loss per
                  Epoch while Training the Model</h3>
        </div>
        <div class="panel-body">
            <div class="col-md-6">
                <h4 class="text-center"><strong>Fig 1. {{
                      graphs[1].fig_title }}</strong></h4>
                <p><img src='{{ graphs[1].src }}' alt='{{
                      graphs[1].image_alt }}' class='img-
                      responsive center-block'></p>
            </div>

            <div class="col-md-6">
                <h4 class="text-center"><strong>Fig 2. {{
                      graphs[2].fig_title }}</strong></h4>
                <p><img src='{{ graphs[2].src }}' alt='{{
                      graphs[2].image_alt }}' class='img-
                      responsive center-block'></p>
            </div>
        </div>
    </div>
    <div class="panel panel-default">
        <div class="panel-heading main-color-bg">
            <h3 class="panel-title">Layers of the Model</h3
                  >
        </div>
        <div class="panel-body">
            <h4 class="text-center"><strong>Fig 3. {{ graphs
                  [0].fig_title }}</strong></h4>
            <p><img src='{{ graphs[0].src }}' alt='{{ graphs
                  [0].image_alt }}' class='img-responsive center
                  -block'></p>
        </div>
    </div>
{% endblock %}
```

app/templates/models.html

```
{% extends 'dashboard.html' %}

{% block breadcrumb_content %}
    Dashboard > Models
{% endblock %}

{% block panel_title_content %}
    List of Models
{% endblock %}

{% block panel_body_content %}
    <div class="col-md-12 table-responsive">
        {% if models %}
            <table class="table table-striped table-hover">
                <tr>
                    <th>ID</th>
                    <th>Username of AI Expert</th>
                    <th>Name</th>
                    <th>Train Accuracy</th>
                    <th>Train Loss</th>
                    <th>Validation Accuracy</th>
                    <th>Validation Loss</th>
                    <th>Test Accuracy</th>
                    <th>Test Loss</th>
                    <th>Used as system's model?</th>
                </tr>

                {% for model in models %}
                    <tr>
                        <td>{{ model.id }}</td>
                        <td>{{ model.username }}</td>
                        <td>{{ model.name }}</td>
                        <td>{{ model.train_acc }}</td>
                        <td>{{ model.train_loss }}</td>
                        <td>{{ model.validation_acc }}</td>
                        <td>{{ model.validation_loss }}</td>
                        <td>{{ model.test_acc }}</td>
                        <td>{{ model.test_loss }}</td>
                        <td><strong>{% if model.
                              used_by_doctors %} Yes {% else
                              %} No {% endif %}</strong></
                              td>
                    </tr>
                {% endfor %}
            </table>
        {% endif %}
    </div>
{% endblock %}
```

app/templates/my_models.html

```
{% extends 'dashboard.html' %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block breadcrumb_content %}
    Dashboard > My Models
{% endblock %}

{% block panel_title_content %}
    My Models
{% endblock %}

{% block panel_body_content %}
    <div class="col-md-12 table-responsive">
    {% if user_models %}
        <table class="table table-striped table-hover text-
              center">
            <tr>
                <th class="text-center">ID</th>
                <th class="text-center">Name</th>
                <th class="text-center">System's model?</th
                      >
                <th class="text-center"></th>
                <th class="text-center"></th>
                <th class="text-center"></th>
                <th class="text−center"></th>
            </tr>

            {% for model in user_models %}
            <tr class="clickable−row" data−href="{{ url_for('
                    main.model', title='About Model ' + model.id
                    |string + ': ' + model.name, id=model.id)
                    }}">
                <td>{{ model.id }}</td>
                <td>{{ model.name[:−3] }}</td>
                <td>{% if model.used_by_doctors %} <strong>
                        Yes</strong> {% else %} No {% endif
                        %}</td>
            </tr>
            {% endfor %}
        </table>
    {% endif %}
    </div>
  </div>
</div>


{% endblock %}


{% block styles %}
    {{ super() }}

    <style>
    tr . clickable −row {
        cursor: pointer;
    }
    tr . clickable −row:hover td {
        background−color: #007bff !important;
        color : white;
    }
    </style>
{% endblock %}


{% block scripts %}
    {{ super() }}

    <script>
    jQuery(document).ready(function($) {
        $ ('. clickable −row').click(function() {
            window.location = $(this).data('href') ;
        });
    });
    </script>
{% endblock %}

{% block more_panels_area %}
{% endblock %}
```

app/templates/register.html

```
{% extends 'base.html' %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block page_content %}
    <div class="container">
        <div class="row">
            <div class="col−md−4"></div>

            <div class="col−md−4">
                <div class="page−header">
                    <h1 align="center">Register</h1>
                </div>

                {{ wtf.quick_form(form) }}
            </div>

            <div class="col−md−4"></div>
        </div>
    </div>
{% endblock %}
```

app/templates/train.html

```
{% extends 'dashboard.html' %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block breadcrumb_content %}
    Dashboard > Train Model
{% endblock %}

{% block panel_title_content %}
    {% if done_training %}
        Metrics of the New Model
    {% else %}
        Train Model
    {% endif %}
{% endblock %}

{% block panel_body_content %}
    {% if done_training %}
        <div class="col−md−12 table−responsive">
            <table class="table table−striped table−hover text
                    −center">
                <tr>
                    <th class="text−center">Train Accuracy
                            </th>
                    <th class="text−center">Train Loss</th>
                    <th class="text−center">Validation
                            Accuracy</th>
                    <th class="text−center">Validation Loss
                            </th>
                    {% if model.test_acc %}
                        <th class="text−center">Test
                                Accuracy</th>{% endif %}
                    {% if model. test_loss %}
                        <th class="text−center">Test Loss</
                                th>{% endif %}
                </tr>
                <tr>
                    <td>{{ model.train_acc }}</td>
                    <td>{{ model.train_loss }}</td>
                    <td>{{ model.validation_acc }}</td>
                    {% if model.test_acc %}
                        <td>{{ model.validation_loss }}</td>
                        <td><strong>{{ model.test_acc }}</
                                strong></td>
                        <td><strong>{{ model.test_loss }}</
                                strong></td></tr>
                    {% else %}
                        <td>{{ model.validation_loss }}</td
                                ></tr>
                    {% endif %}
            </table>
        </div>
    {% else %}
        <div class="col−md−6">
            <form method='post' enctype='multipart/form−
                    data'>
                <div class="form−group">
                    <label for="dataset">Dataset:</label>
                    <input type="file" name="dataset"
                            multiple="" class="form−control−file
                            ">
                </div>

                <div class="form−group">
                    <label for="dataset_labels">Labels (CSV):
                            </label>
                    <input type="file" name="dataset_labels"
                            class="form−control−file">
                </div>

                <p><strong>Epochs: </strong><input type
                        ="number" value="1" name="epochs"
                        step="1" min="1" max="1000" class="
                        number"></p>
                <p><strong>Batch Size: </strong><input
                        type="number" value="10" name="
                        batch_size" min="10" max="100" class="
                        number"></p>

                <div class="form−group">
                    <label for="exampleFormControlSelect1">
                            Optimizer:</label>
                    <select class="form−control" id="
                            exampleFormControlSelect1" name="
                            optimizer">
                        <option value="sgd" selected>
                                Stochastic Gradient Descent</
                                option>
                        <option value="rmsprop">RMSprop</
                                option>
                        <option value="adam">Adam</option
                                >
                        <option value="adagrad">AdaGrad</
                                option>
                    </select>
                </div>

                <p><strong>Learning Rate: </strong><input
                        type="number" value="0.01" name="
                        learning_rate" step="0.01" class="number
                        "></p>
                <hr>

                <h3>Convolutional Base</h3>
            <div class="form-group">
                <label for="network_architecture">Network
                    Architecture:</label>
                <select class="form-control" id="
                    exampleFormControlSelect1" name="
                    network_architecture">
                    <option value="vgg16" selected>
                        VGG16</option>
                    <option value="vgg19">VGG19</
                        option>
                    <option value="xception">Xception</
                        option>
                    <option value="resnet50">ResNet50</
                        option>
                    <option value="inceptionv3">
                        InceptionV3</option>
                    <option value="inceptionresnetv2">
                        InceptionResNetV2</option>
                    <option value="densenet121">
                        DenseNet121</option>
                    <option value="densenet169">
                        DenseNet169</option>
                    <option value="densenet201">
                        DenseNet201</option>
                </select>
            </div>

            <hr>

            <p>
                <button type="submit" class="btn btn-
                    primary">Train</button>
            </p>
        </form>
    </div>
    <div class="col-md-6">
    </div>
  {% endif %}
{% endblock %}

{% block more_panels_area %}
  {% if done_training %}
    <div class="panel panel-default">
        <div class="panel-heading main-color-bg">
            <h3 class="panel-title">Layers and Loss per
                Epoch while Training the Model</h3>
        </div>
        <div class="panel-body">
            <div class="col-md-6">
                <h4 class="text-center"><strong>Fig 1.
                    {{ graphs[1].fig_title }}</strong></
                    h4>
                <p><img src='{{ graphs[1].src }}' alt='{{
                    graphs[1].image_alt }}' class='img-
                    responsive center-block'>
                </p>
            </div>

            <div class="col-md-6">
                <h4 class="text-center"><strong>Fig 2.
                    {{ graphs[2].fig_title }}</strong></
                    h4>
                <p><img src='{{ graphs[2].src }}' alt='{{
                    graphs[2].image_alt }}' class='img-
                    responsive center-block'>
                </p>
            </div>
        </div>
    </div>
    <div class="panel panel-default">
        <div class="panel-heading main-color-bg">
            <h3 class="panel-title">Layers of the Model
                </h3>
        </div>
        <div class="panel-body">
            <h4 class="text-center"><strong>Fig 3. {{
                graphs[0].fig_title }}</strong></h4>
            <p><img src='{{ graphs[0].src }}' alt='{{
                graphs[0].image_alt }}' class='img-
                responsive center-block'>
            </p>
        </div>
    </div>
  {% endif %}
{% endblock %}
```

app/util/__init__.py

```
from enum import Enum

class LinkIndex(Enum):
    PREDICT = 0
    RETRAIN_MODEL = 1
    MY_MODELS = 2
    SYSTEM_MODEL = 3
    # FINE_TUNE_MODEL = 4

    DASHBOARD = 0
    MODELS = 1
    USERS = 2


class UserRoleID(Enum):
    AI_EXPERT = 1
    DOCTOR = 2
    SITE_ADMIN = 3


AI_EXPERT_LINKS = [
    {'is_active': False, 'href': 'main.index', 'icon': 'search',
        'title': 'Predict'},
    {'is_active': False, 'href': 'main.train', 'icon': '
        blackboard', 'title': 'Train Model'},
    {'is_active': False, 'href': 'main.my_models', 'icon': '
        folder-open', 'title': 'My Models'},
    #{'is_active': False, 'href': 'main.model', 'icon': '
        pushpin', 'title': 'System Model'},
    # {'is_active': False, 'href': 'main.my_models', 'icon': '
        wrench', 'title': 'Fine-tune Model'}
]

DOCTOR_LINKS = [
    {'is_active': False, 'href': 'main.index', 'icon': 'search',
        'title': 'Predict'},
]

SITE_ADMIN_LINKS = [
    {'is_active': True, 'href': 'main.index', 'icon': '
        dashboard', 'title': 'Dashboard'},
    {'is_active': False, 'href': 'main.models', 'icon': 'stats',
        'title': 'Models'},
    {'is_active': False, 'href': 'main.users', 'icon': 'user', '
        title': 'Users'}
]


def set_link_active(links, index_to_true):
    for index, link in enumerate(links):
        if LinkIndex(index) == LinkIndex(index_to_true):
            links[index]['is_active'] = True
        else:
            links[index]['is_active'] = False


def get_link_active(links):
    for index, link in enumerate(links):
        if links[index]['is_active']:
            return index


ALLOWED_CT_SCAN_EXTENSIONS = set(['dcm'])
ALLOWED_LABEL_EXTENSIONS = set(['csv'])


def allowed_ct_scan_format(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower
        () in ALLOWED_CT_SCAN_EXTENSIONS


def allowed_label_format(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower
        () in ALLOWED_LABEL_EXTENSIONS


def get_network_architecture_string(network_architecture):
    full_name = ''
    if network_architecture == 'vgg16':
        full_name = 'VGG16'
    elif network_architecture == 'vgg19':
        full_name = 'VGG19'
    elif network_architecture == 'xception':
        full_name = 'Xception'
    elif network_architecture == 'resnet50':
        full_name = 'ResNet50'
    elif network_architecture == 'inceptionv3':
        full_name = 'InceptionV3'
    elif network_architecture == 'inceptionresnetv2':
        full_name = 'InceptionResNetV2'
    elif network_architecture == 'densenet121':
        full_name = 'DenseNet121'
    elif network_architecture == 'densenet169':
        full_name = 'DenseNet169'
    elif  network_architecture == 'densenet201':
        full_name = 'DensetNet201'

    return full_name


def get_optimizer_string(optimizer):
    full_name = ''
    if optimizer == 'sgd':
        full_name = 'Stochastic Gradient Descent'
    elif optimizer == 'adam':
        full_name = 'Adam'
    elif optimizer == 'adagrad':
        full_name = 'Adagrad'
    elif optimizer == 'rmsprop':
        full_name = 'RMSprop'

    return full_name
```

XI. Acknowledgement

I have been waiting for this moment since the first semester of my fourth year in ComSci: writing the acknowledgment page of my SP. My SP days are by far, the most stressful days of my life. Thus, I'd like to use this little corner of my document to thank the people who were around to guide me and/or lift me up during these trying times.

To the NTTC library, the place I frequently go to whenever I need to study during the weekday, thank you for existing.

To my adviser, Sir Marvin John Ignacio, thank you for the guidance, for the crash course in Deep Learning, and for convincing me to continue with my proposal after I texted you that I didn't want to go through with it as I felt I wasn't ready yet. Because of my poor time management skills, I texted you the same thing a few hours before my defense. And again, you convinced me that I was ready to defend my SP. I definitely wouldn't be here without your guidance. Thank you sir!

To Ma'am Eden, thank you for always guiding us ComSci students. Thank you for helping me with my subjects na nagkakaproblema pagdating sa grades or sa enrollment.

To my guidance counselor Ms. Elgie, thank you for listening to my problems during the lowest point of my college life. I probably wouldn't be in UP Manila anymore had you not lent your ears when I needed someone impartial to listen.

To Paul and Ara, thank you for being there during those awful two semesters of my college life. Thank you for being there after. Kahit na lagi niyo akong inaasar noon kung may kaibigan na ba ako sa ComSci after ko mag-shift.

To my brother Amir, thank you for doing my share of the chores because I was rarely at home. Thank you for always including me whenever you're brewing coffee.

To Ate Carol and Kuya Bryant, thank you for the nights you accommodated us for our all-nighters. Thank you for the snacks and drinks, for always asking us if

we're comfortable, and for assuring us that we're always welcome in your home. This SP definitely would not be finished on time if not for your hospitality. Thank you po nang sobra!

To JB, thank you for updating me sa mga nangyayari sa Physics at HI hahaha. Thank you for always telling me na makakahabol din ako sa 2nd round ng defense. I'm super grateful na nakilala kita nang husto sa huling taon ko sa ComSci.

To MM, thank you for listening to my rants. I'm sorry di ko laging nasasagot ComSci-related questions mo dahil busy sa SP. Thank you for being encouraging all the time. Nakakahawa ang enthusiasm mo sa life hahaha.

To the most important people in my ComSci life: Abe, Eddie, and Faye. Honestly, where would I even be without you guys? Firstly, thank you for being my constants during our SP days. Thank you for pushing me to finish my SP, for encouraging me noong akala ko di na ako makakadefend tsaka noong 4th Physics exam ko haha. Hinhintay ko ang Tagaytay, Bohol, Thailand, at Japan travels. I don't think I'll be able to say everything I want to say to you guys here. Still, allow me to say my thanks to you three individually.

To Abe, thank you for being so bright and cheerful all the time. Nakakahawa nang sobra. Thank you for listening to my problems. Sana makilala ka naman ng mga crush mo? Salamat sa paghatid sa akin pauwi kahit na dapat di ka na kumakaliwa sa Pedro Gil kasi QC pa uwi mo haha. Next time sa south naman tayo tumambay. Nakakasawa ang QC cyst

To Faye, thank you for pushing me to do all that I can whenever I'm struggling with schoolwork. Thank you for listening whenever I share a too-personal problem. Thank you for being patient with my mood swings. Sorry kung madalas tayo mag-away? Salamat din sa paghatid sa amin everytime na kung saan-saan tayo napupunta. Sana nasimot na lahat ng issue bago tayo grumaduate friend

To Eddie, thank you for being my MP groupmate sa karamihan ng MPs natin sa

ComSci. Thank you for listening whenever I feel like ranting. Thank you sa corny jokes? Thank you sa Divisoria dates, sa random lakwatsa na napagdesisyunan habang nasa gitna tayo ng paggawa ng SP kasi bigla tayong tinamad. Thank you sa pagyaya sa akin mag-take ng Weight Training. Biro lang di ko natutuwa roon. Sana friends pa tayo sa December

To my parents, I will also be thanking you both individually.

To Inah, thank you for being understanding because I was often late in going home. Thank you for reminding to sleep early, eat healthy foods, and drink less unhealthy drinks. Thank you for whipping up coffee when Amir and I needed it during our sleepless nights. Thank you for the Lipovitan na magugulat na lang ako ay inaabot niyo po sa akin every morning na may exam ako. Thank you for comforting me when I cried early in the morning of my proposal.

To Mmah, first of all, thank you for buying me a new laptop. I selfishly asked for one because I needed a more powerful machine for my SP. I could've requested access to Pagaspas in the server room or even worked on a different SP topic altogether, but I didn't. I told you about how I needed a better laptop and you told me you'd buy me one. It's not even just the laptop. When I asked for something, you'd say yes immediately. I missed you while you were still in Cotabato. Welcome home!

To the both of you, Inah and Mmah, thank you for all the sacrifices you've done for me. I'm sorry it took me 6 years to graduate. I hope I still make you proud.