CHESTLENS AI

System Verification and Validation Plan

Group 18

Supervisor	
Name:	Dr. Mehdi Moradi
Email:	moradm4@mcmaster.ca
Organization:	McMaster University, CAS
Team Member 1	
Name:	Abhishek Sharma
Email:	shara109@mcmaster.ca
Student Number:	400322503
Team Member 2	
Name:	Anthony Vu
Email:	vua11@mcmaster.ca
Student Number:	400306059
Team Member 3	
Name:	Hussein Saad
Email:	saadh@mcmaster.ca
Student Number:	400307995
Team Member 4	
Name:	Nathan Starr
Email:	starrn@mcmaster.ca
Student Number:	400323095
Team Member 5	
Name:	Yuvraj Jain
Email:	jainy3@mcmaster.ca
Student Number:	400259484

Revision History

Version Number	Authors	Description	Date
0	● Abhishek Sharma ● Anthony Vu ● Hussein Saad ● Nathan Starr ● Yuvraj Jain	Started Initial Draft	Feb 9th, 2024
1	● Abhishek Sharma ● Anthony Vu ● Hussein Saad ● Nathan Starr ● Yuvraj Jain	Updated VnV for final submission	March 27th, 2024

1.0 Table of Contents

2.0 Project Description

Currently, radiographers facilitate the imaging process for X-rays and radiologists manually analyze the produced medical images to find any abnormalities, injuries, or diseases. This can be very time-consuming and like all things has the possibility of errors. To mitigate this issue we are implementing an AI model to identify six common findings. These findings include Atelectasis, Cardiomegaly, Consolidation, Edema, No Finding, and Pleural Effusion. The main objective of using an AI model on chest X-rays is to identify negative results or identify which of these diseases are present and the location of these diseases with the use of visual mapping. We believe that creating an AI for chest X-rays is beneficial since it can be used by radiologists to help confirm or identify any abnormalities, injuries, and diseases present in X-ray images. Also, having the assistance of an AI model will save radiologists time and save hospitals money and resources since the analysis process will be sped up and a diagnosis can be given to patients faster.

SRS Document: Link; Design Document: Link

3.0 Component Test Plan

3.1 UI

3.1.1 Unit Tests

Since we are building a frontend for a Flask API, we are planning to use unittest (https://docs.python.org/3/library/unittest.html) for unit testing all of the components of our UI.

3.1.1.1 Login Page

If the user inputs valid values into the username and password fields, then clicks the login button, we expect that the user is forwarded to the corresponding doctor's page. Otherwise, the user should remain on the login page, and an error saying "incorrect username or password" should be displayed.

3.1.1.2 Doctor's Page

When the user clicks the view button for a patient, we expect that the user is forwarded to the corresponding patient's page.

3.1.1.2 Patient's Page

When the user clicks the upload button, we expect that their device's file explorer opens, allowing them to browse for a DICOM image. Here, we will test if an error correctly shows up when the user inputs a file of the wrong type. If a DICOM image was correctly uploaded, when the user clicks the submit button, we expect that the DICOM image is sent to the backend, and the results are displayed on the frontend.

Also, when the user selects an image from the dropdown we expect that the displayed image changes accordingly.

Furthermore, when the user inputs characters into the comment field, then clicks the save button, we expect that the comment is saved and displayed on the frontend.

3.1.2 Performance Tests and Metrics

The main metric we want to test is latency, specifically how long it takes for a user to get results after uploading a DICOM image to the UI. Currently, results come out within a couple seconds, so we want to test how it performs under different conditions (i.e., internet connection, hardware, limited space on a hosted database). Our threshold will be 10-15 seconds.

3.2 Image Server

3.2.1 Unit Tests

Our goal is to ensure smooth communication between the image server and the user. As such, we devised a variety of tests to cover a wide range of cases. Most importantly, we would like to verify successful uploading of images. This will be achieved by simulating the upload of a test file and checking its expected location to confirm.

Since we are dealing with sensitive data, our system must exhibit strict access control. Manual and automated attempts to access restricted files will be made to confirm that authorized users are being admitted while unauthorized ones are being denied.

All functionality can be achieved with Python scripting and help from the "requests" library (https://pypi.org/project/requests).

3.2.2 Performance Tests and Metrics

The most crucial metric we consider is the upload/download time of DICOM images. At average internet speeds, which is a safe assumption for medical facilities, images must not take any longer than 10 seconds to download, and 15 seconds to upload. We believe this to be a fair requirement factoring in delays from both the server and the user's internet connection.

3.3 Image Processor

3.3.1 Unit Tests

As mentioned, we are using Flask to facilitate the API calls for which we would be doing unit testing to ensure the API endpoints return valid responses. We plan to use the unittest (https://docs.python.org/3/library/unittest.html) module from Python's standard library.

3.3.1.1 Login Page

The unit testing is focused on verifying the behavior of the authentication mechanism implemented. The test cases will include scenarios like:

- Verifying that valid credentials allow access to the Doctor's page.
- Verifying that invalid credentials result in appropriate error messages or redirects.
- Testing the behavior when the login form is submitted with missing or incomplete data.

- Ensuring that authentication tokens or session cookies are set correctly upon successful login.

3.3.1.2 Doctor's Page

The unit testing is focused on testing the functionality related to displaying the list of patients assigned to a specific doctor. The test cases will include scenarios like:

- Verifying that the endpoint returns the expected list of patients assigned to the logged-in doctor.
- Verifying the page redirects to the patient's page when the view button is clicked for the corresponding patient.
- Testing the behavior when there are no patients assigned to the doctor.
- Ensuring that unauthorized access attempts are properly handled, such as redirecting to the login page.

3.3.1.2 Patient's Page

The unit testing for the Patient's page is focused on verifying the different functionalities such as uploading a DICOM image, fetching the converted image, etc. The test cases will include scenarios like:

- Verifying that invalid file type results in appropriate error messages.
- Verifying that the properties of the DICOM image stay the same after conversion to JPEG format.
- Verifying that the save button results in the insertion of the doctor's comment in the corresponding Doctor's SQL table.
- Verifying the radio button fetches the respective images.
- Verifying that the endpoint for the upload button returns the expected JSON with patient details and model output.

Note: All the scenarios above test each API endpoint with multiple input cases, including edge cases. The test cases pass when the response of the API endpoint has a status code of 200 for valid result and status code of 40000 for error messages. To ensure the properties of the image do not change with conversion, we will do manual and automated verification, mainly by inputting DICOM images and checking for outputs in JPEG format.

3.3.2 Performance Tests and Metrics

The main metrics we want to test are latency and load balancing. Although our testing is limited by the storage we have on our hosted database, we will try to create test scenarios that mimic real-world usage patterns like login, file upload, etc. Additionally, the API calls should return the responses within a few seconds, with the threshold being 10-15 seconds (based on the size of the query).

3.4 AI Model

3.4.1 Performance Tests and Metrics

The Python Time library (https://docs.python.org/3/library/time.html) will be used to calculate the runtime of our model when given an image processed by the Image Processor. This will be done 20 times and an average time to execute will be taken. The speed of our model reaches our goals if the average is under 10 seconds.

3.5 Training/Testing Module

Since our model is handling multiple disease labels, and the distribution of these diseases within the MIMIC-CXR dataset is uneven, using accuracy as a test metric becomes problematic. For instance, consider a scenario where only 10% of the images in the dataset are positive for pneumonia, while the remaining 90% are negative. If the model consistently predicts negative for pneumonia, it would achieve an accuracy of 90%, misleadingly suggesting high accuracy when, in reality, it's not effective. To address this issue, we use ROC curves and F1 scores for each disease.

3.5.1 Overfitting Prevention/Testing

To ensure the effectiveness of our model and prevent overfitting to the training data, we've implemented a 70/20/10 split. This means that 70% of the data is dedicated strictly for training, 10% is used for validation throughout the training process, and the remaining 20% is used purely for testing once all training is complete. The ROC Curve and F1 scores are calculated for the 10% validation set and then these results are used to do hyperparameter tuning of the model and early stopping, model checkpointing etc. After training is complete the model's accuracy is evaluated on the testing set by using the ROC and F1 metrics. It's important to note that the accuracy calculated from the testing set does not influence the model's training process. Instead, it serves as an independent measure to assess the overall accuracy of the model and it can be used to check for overfitting as the test set had no influence on the training of the model. Lastly the test set will be used to determine the final accuracy of our model.

3.5.2 ROC Curve

A ROC Curve (Receiver Operating Characteristic Curve) serves as a graphical tool for evaluating the performance of classification models dealing with binary and multiclass problems. It plots the relationship between the True Positive Rate (TPR) and False Positive Rate (FPR) as the classification threshold varies. This visualization is useful since it displays the model's capacity to differentiate between positive and negative instances across different threshold levels.

The TPR, also known as Sensitivity or Recall, measures the proportion of actual positive instances correctly classified by the model, calculated as TPR = TP / (TP + FN). Conversely, the FPR gauges the proportion of actual negative instances incorrectly classified as positive, computed as FPR = FP / (FP + TN).

By plotting ROC curves individually for each disease and determining the area under each curve, a more precise assessment of the model's performance for each specific disease is attained. This approach is useful since it supports imbalanced distribution of diseases, offering a comprehensive evaluation of the model's effectiveness.

The Area Under the ROC Curve (AUC) provides a quantification of the model's overall performance, ranging from 0 to 1. The AUC of the ROC curve reflects how well the model distinguishes between positive and negative instances across different threshold levels. It summarizes the trade-off between true positive rate (TPR) and false positive rate (FPR) across all possible threshold values. A higher AUC indicates better discrimination ability, meaning the model achieves higher TPRs while keeping FPRs low across thresholds.

To compute the ROC curve for the five diseases covered in our model, the sklearn.metrics library will be utilized. For each disease, one label will be designated as "positive," with the rest labeled as "negative." This process will be repeated for all five diseases, generating five binary classification scenarios. The roc_curve function will be used to generate ROC curves, with subsequent calculation of AUC values using the auc function.

Analysis of the ROC curves and AUC values will be done to evaluate the model's disease-specific performance. We would like to have AUC values close to 1 as this indicates robust performance and higher a TPR and a lower FPR, while values close to 0.5 signify weaker performance. Through this analysis, we would like to make decisions regarding the model's accuracy in classifying each specific disease.

3.5.3 Precision

3.5.3.1 Overview

Precision measures the accuracy of the model in correctly identifying the positive case of the disease. In our case, it will be evaluated individually for each disease. It is calculated by dividing the number of correctly identified positive diseases by all positive disease identification (true positives and false positives).

3.5.3.2 Calculation

It will be calculated using sklearn.metrics.precision_score for each disease.

3.5.4 Recall

3.5.4.1 Overview

Recall measures completeness of the model. It does this by measuring how well the model minimizes false negatives. We will be measuring recall for each disease. It is calculated by dividing the number of positives the model gets right by the total number of positives in the data (both those the model got right and wrong).

3.5.4.2 Calculation

It will be calculated using sklearn.metrics.recall_score for each disease.

3.5.5 F1-Score

3.5.5.1 Overview

F1-Score combines how complete the predictions are with the quality of those predictions. Completeness is measured by recall while quality of the model's predictions is captured by precision. F1-Score combines both recall and precision, making it the most important of the 3 metrics. F1 is the Harmonic mean of both scores. When evaluating the models performance the F1 score will have a greater importance than precision and recall.

3.5.5.2 Calculation

It will be calculated using sklearn.metrics.f1_score for each disease.

3.5.6 Grad-CAM

3.5.6.1 Overview

Grad-CAM stands for Gradient-weighted Class Activation Mapping. It returns a heat map of how much each pixel in the original image contributes to the feature map of any identified disease. This highlighted area in the image should correspond with the area of the disease, we are interested in testing it to ensure that this is the case.

3.5.6.2 Calculation

Our dataset is MIMIC-CXR and it contains doctors notes of patients, some of which contain locations of diseases in the chest X-ray. For each disease we will find a set of 5 patients where the doctor's notes on their x-rays contain a clear description of the disease's location. We will then run those patients' images though the model and through visual inspection check if the location highlighted matches the doctor's notes. Then we will calculate the ratio of images where the Grad-CAM highlighted region matches the doctor's notes for each disease. We will include the highlighted images and doctors notes for each of these tests in our testing documentation. Full quantitative validation of the accuracy of attention maps is currently not P0/P1 priority.

3.5.7 Model Performance Validation

To check performance of our model we will be working with two models. One of the models will be pre trained and the second model will be our own model trained from scratch. We will run the test set on each of the three models and compute the ROC Curves and the F1 scores for each disease. We will then compare these metrics for each of the three models and determine the accuracy for each disease. We would like to use this comparison to see which of the three models produces the best overall accuracy and how the accuracy for each disease differs per model.