The Potential of Artificial Intelligence to Identify Cancer and Aid Research

Author(s) Wilson B. Peguero Rosario

Master of Science in Data Science Capstone Design Specification

Grand Canyon University

Instructor: Professor TBA

Revision: 2

Date: November 23, 2022

**TABLE OF CONTENTS**

System Entity 3

Functional Requirements 3

source code listing 4

Code Review 9

Implementation Plan 9

Application Functionality and Execution 11

User guide 12

**System Entity**

The application will use the Dash library in conjunction with pydicom, pandas, and the TensorFlow libraries. The pydicom library will allow one to open and extract information from the uploaded DICOM images. The data collected using pydicom is transferred over to a pandas DataFrame object that in turn is used to run predictions through a TensorFlow Model object that was trained on previous data and designed based on the VGG-16 CNN model architecture. The Dash library is used to create an application class which builds the website. Ideally the website is hosted locally due to the size of the model (approximately 4Gb), but due to limitations made by the ISP provider, one is not able to provide public access to the website. A copy of the finalized code is provided as part of the system entity.

**Functional Requirements**

|  |  |  |
| --- | --- | --- |
| Functional Requirement Number | Functional Requirement Description | Functional Requirement Function(s)/Class(es) |
| FR1 | The system must be able to load DICOM files. | capapp.parse\_contents()   * Function used to load the uploaded DICOM files for data extraction and transformation |
| FR2 | The system must be able to make predictions based on loaded DICOM files. | pipeline.extract\_data()   * Function which uses the pydicom library to read and extract useful data from the DICOM files. * Activated upon the user interaction with the predict button on the main page GUI.   pipeline.transform\_data()   * Function which uses basic python operations to prepare the data for model input.   models.tumor\_classifier()   * Function that uses TensorFlow’s Functional API to * Uses a combination of convolutional neural networks, pooling layers, and linear layers for final classification. * Outputs the prediction for use. * Activated once data is fully extracted from user selected files.   Pipeline.predict()   * Loads the model for making predictions |
| FR3 | The system must be able to display data extracted. | Capapp.update\_output()   * Displays the data within a dashboard format. * Contains a list of image files and some basic data related to the image file. * The list will also have a button to the right to allow the user to another frame that displays the targeted image. * Allows the download of CSV file. |

**Source Code Listing**

The third Iteration of the Capstone Project Application.

After failing to create the offline GUI through the use of Tkinter library and matplotlib, an online version of the same GUI has been replicated using the Dash library. This library uses the plotly library on the backend to develop plots and display them within html that the Dash library develops.

capapp.**parse\_contents**(*contents*, *filename*, *date*)

Load the content.

Function set to load the uploaded content and transform the data into a readable format for the machine learning model to make predictions.

**Parameters:**

* **contents** (*Unkown*) – Contains the uploaded file encoded into its 64-bit version. This is then decoded and read using the pydicom library for data processing.
* **filename** (*string*) – The name of the uploaded file. this does not impact the data processing aspect or the predictions, but it is necessary to point out which file the user is looking at.
* **date** (*datetime*) – Used to let the user know the date in which the file was uploaded. Will be used within report to download.

capapp.**update\_output**(*list\_of\_contents*, *list\_of\_names*, *list\_of\_dates*)

Load the main dashboard.

Models Module

This file will contain all of the actualized models created from the abstract model class(es) made within the base.py file.

models.**base\_image\_classifier**(*img\_height: float*, *img\_width: float*)

Basic Image Classifier for model comparison improvement.

…

A class containing a simple classifier for any sort of image. The models stemming from this class will function to only classify the image in one manner alone (malignant or non-malignant). This model will not contain any rescaling or data augmentation to show how significant the accuracy between a model with rescaling and data augmentation is against a model without any of these.

**Parameters:**

* **img\_height** (*float*) – The height, in pixels, of the input images. This can be the maximum height of all images within the dataset to fit a varied amount that is equal or less than the declared height.
* **img\_width** (*float*) – The width, in pixels, of the input images. This can also be the maximum width of all images within the dataset to fit a varied amount that is equal or smaller in width to the declared dimension.
* **batch\_size** (*int*) – One of the factors of the total sample size. This is done to better train the model without allowing the model to memorize the data.

**Returns:**

* **inputs** (*{img\_input, cat\_input}*) – Input layers set to receive both image and categorical data. The image input contains images in the form of a 2D numpy array. The categorical input is a 1D array containing patient information. This is mainly comprised of categorical data, but some nominal data.
* **x** (*Dense Layer*) – The last layer of the model developed. As the model is fed through as the input of the next layer, the last layer is required to create the model using TensorFlow’s Model class.

models.**base\_tumor\_classifier**(*img\_height: float*, *img\_width: float*)

Base Tumor Classification Algorithm.

…

A class containing a simple classifier for side-view image. The models stemming from this class will include rescaling for the sake and purpose of normalizing the data.

**Parameters:**

* **img\_height** (*float*) – The height, in pixels, of the input images. This can be the maximum height of all images within the dataset to fit a varied amount that is equal or less than the declared height.
* **img\_width** (*float*) – The width, in pixels, of the input images. This can also be the maximum width of all images within the dataset to fit a varied amount that is equal or smaller in width to the declared dimension.

**Returns:**

* **inputs** (*{img\_input, cat\_input}*) – Input layers set to receive both image and categorical data. The image input contains images in the form of a 2D numpy array. The categorical input is a 1D array containing patient information. This is mainly comprised of categorical data, but some nominal data.
* **output** (*Dense Layer*) – The last layer of the model developed. As the model is fed through as the input of the next layer, the last layer is required to create the model using TensorFlow’s Model class.

models.**tumor\_classifier**(*img\_height: float*, *img\_width: float*)

Complete Tumor Classification Algorithm.

…

A class containing a simple classifier for any sort of image. The models stemming from this class will include rescaling and data augmentation for the sake and purpose of normalizing the data.

**Parameters:**

* **img\_height** (*float*) – The height, in pixels, of the input images. This can be the maximum height of all images within the dataset to fit a varied amount that is equal or less than the declared height.
* **img\_width** (*float*) – The width, in pixels, of the input images. This can also be the maximum width of all images within the dataset to fit a varied amount that is equal or smaller in width to the declared dimension.
* **batch\_size** (*int \**) – One of the factors of the total sample size. This is done to better train the model without allowing the model to memorize the data.

**Returns:**

* **inputs** (*{img\_input, cat\_input}*) – Input layers set to receive both image and categorical data. The image input contains images in the form of a 2D numpy array. The categorical input is a 1D array containing patient information. This is mainly comprised of categorical data, but some nominal data.
* **output** (*Dense Layer*) – The last layer of the model developed. As the model is fed through as the input of the next layer, the last layer is required to create the model using TensorFlow’s Model class.
* *—*
* *\*Deprecated*

Pipeline Module

Algorithms used to process data before modeling.

…

A set of algorithms used to feed in and process data before used within the model. This will contain the data extraction from its rawest form and output the final form of the data set. The main source of data will be image related from the Cancer Imaging Archive.

pipeline.**calculate\_confusion\_matrix**(*fin\_predictions: DataFrame*)

Calculate the confusion matrix using pandas.

Calculates the confusion matrix using a csv file that contains both the predictions and actual labels. This function then creates a crosstab of the data to develop the confusion matrix.

Parameter(s)

fin\_predictions*Pandas DataFrame*

DataFrame containing the prediction and actual labels.

**returns:**

* **ct** (*Pandas DataFrame*) – Cross tab containing the confusion matrix of the predictions compared to the actual labels.
* **metrics** (*Dictionary*) – Contains the basic metrics obtained from the confusion matrix. The metrics are the following: - Accuracy - Precision - Recall - F1 Score

pipeline.**extract\_data**(*file*) → dict

Extract the data from the .dcm files.

…

Reads each independent file using the pydicom library and extracts key information, such as the age, sex, ethnicity, weight of the patient, and the imaging modality used.

**Parameters:**

**file** (*Unknown*) – Either the path to the file or the file itself. In the case that the .dcm file is already loaded, the algorithm will proceed to extract the data. Otherwise, the algorithm will load the .dcm file and extract the necessary data.

**Returns:**

**datapoint** – Dictionary comprised of the image data (numpy array), and the metadata associated with the DICOM file as its own separate *key:value* pair. This only pertains to the patient data and NOT the metadata describing how the image was taken.

**Return type:**

dictionary

**Raises:**

* **InvalidDicomError** – The file selected for reading is not a DICOM or does not end in .dcm. Set in place to stop the algorithm in the case that any other filetype is introduced. Causes an error to be printed and the program to exit.
* **AttributeError** – Occurs in the case that the DICOM file does not contain some of the metadata used for classifying the patient. In the case that the metadata does not exist, then the model continues on with the classification and some plots may be missing from the second page.

pipeline.**load\_data**(*filename: str*, *batch\_size: int*)

Load the data using tensorflow data set library.

…

Uses the os library and the TensorFlow Data api to load, batch, and process the data for training.

Parameter

filename*str*

Leads to a file containing the paths to all of the DICOM files as well as metadata.

batch\_size*int*

Factor of the length of the data set.

**returns:**

* **X** (*TensorFlow Dataset*) – Zipped dataset containing both image data and categorical data together.
* **y** (*TensorFlow Dataset*) – Data set containing the classifications of the data.

pipeline.**load\_testing\_data**(*filename: str*) → DataFrame

Load the data used for testing.

Loads a dataset to be fed into the model for making predictions. The output of the testing data will be comprised of a dictionary that can be fed directly into the model.

Parameter(s)

filename*str*

path to file containing the file paths to test data.

pipeline.**load\_training\_data**(*filename: str*, *first\_training: bool = True*, *validate: bool = False*)

Load the DICOM data as a dictionary. …

Creates a dictionary containing three different numpy arrays. The first array is comprised of multiple DICOM images, the second contains the categorical data as a vector, and the third contains the classification in numerical form.

**Parameters:**

* **filename** (*str*) – path to a file which contains the metadata, classification, and path to the DICOM file. Will also contain some sort of ID to better identify the samples.
* **batch\_size** (*int*) – Factor of the dataset size. Currently set to one as the standard for testing purposes.

**Returns:**

**data** – Dictionary containing the encoded values for the metadata and the transformed image for input to the model.

**Return type:**

dictionary

pipeline.**predict**(*data: DataFrame*, *model\_name: str*) → DataFrame

Make predictions based on dataset.

Extracts the image data and required categories for loading into the model.

pipeline.**rescale\_image**(*img: ndarray*) → ndarray

Rescale the image to a more manageable size.

Changes the size of the image based on the length and width of the image itself. This is to reduce the amount of computations required to make predictions based on the image.

Parameter(s)

img*Numpy Array*

array containing the raw values of images.

pipeline.**transform\_data**(*datapoint: dict*) → dict

Transform the data into an format that can be used for displaying and modeling.

…

Grabs the extracted data and begins transforming the data into a format that can be used for display in a dashboard as well as for modeling purposes.

**Parameters:**

**datapoint** (*dictionary*) – Contains the image and related metadata in *key:value* pair format.

**Returns:**

**datapoint** – same dictionary with the categorical data transformed into numerical (from text).

**Return type:**

dictionary

**Raises:**

* **AttributeError** – Indicator of the *key* does not exists.
* **KeyError** – Indicator of the *key* does not exists.

**Code Review**

Originally, the capapp was an application that focused mainly on a desktop application (that functioned as an executable). After review by the last professor and some advice from the fellow students of the last class, a decision was made to transition over to a web-based application. Another factor that influenced one towards the we-based application was also the failure of pyinstaller library to create the executable.

**Implementation Plan**

Overview

The purpose of this application is to save radiologists time while providing the best possible treatments to patients. Medical imaging is a technique through which radiologists are able to view the interior of the body and diagnose the patient based on what they have observed (MedlinePlus, 2019). The average radiologists observed (at least in 2017) approximately 20 to 100 scans a day, with each scan containing multiple images for review (Silverman, 2017). The use of DICOM files to make predictions will place the software application in a position where it must also ensure it meets the HIPAA compliance requirements regarding Personal Health Information (PHI) (Crotti, 2020). Once the software meets the HIPAA requirements for PHI, then hospitals will be able to use the software safely.

Installing the software will be done by downloading an executable file that can be found at a public Github page. As the application can be downloaded on any site, it provides administrators with the ability to distribute the application to all computers simultaneously. For future iterations, as the use of EHR systems increase, the software application will be an application developed within an EHR system such as EPIC (Evans, 2016).

Assumptions, Dependencies, Constraints

A lack of HIPAA compliance may be the biggest factor in delays to deployment. Before a product can be HIPAA compliant, multiple investigations must be conducted (Crotti, 2020). Running the risk of not securing the data may lead to not only investigations, but also possible loss of business and lawsuits (Crotti, 2020). Second to this, is the accuracy of the machine learning model. The daily rate of misdiagnosis caused by either systems or humans in radiology is between 3%-5% daily (Brady, 2016). To mitigate this, the model will have an accuracy upon its test of at least 95% with the goal (in terms of accuracy) being 98%. The third factor that may affect deployment is the popularity of EHR systems. As demand for standardization rises in hospitals, so does the need of EHR systems that not only neatly displays data, but also allows for ease download and transfer to other hospitals (Evans, 2016). Besides these three factors, the application should be ready for deployment.

Operational Readiness

Testing model predictions with regressive data in conjunction with comparison between models will be the main form of assessing deployment readiness. As spoken before, the model must be fully trained and saved before making the Graphical User Interface (GUI) into an executable file. This is because without the predictions, the entire software application itself becomes useless. One can also assess readiness based on the current market for offline medical software applications. Many locations within hospitals do not have wireless internet capabilities due to the noise and disturbance it may cause when in use with other medical devices, therefore offline GUIs are popular to use even today (Concordia University, 2015).

Data Conversion

The conversion of data will be based on the GUI itself. The user will first load the paths to the DICOM files that he or she wishes to diagnose, then the GUI application will begin extracting the DICOM data that it requires to make prediction and plot some charts. Once the main dashboard of the GUI is loaded, the user will be able to download a csv containing an array of the image used together with the patient metadata at the click of a button. Within the csv, will be a set of probabilities regarding whether the image should be classified as benign or malignant.

Phased Rollout

The first phase of the rollout will consist of a free version of the software without the ability to download the full report. Only the patient ID and predicted classification columns will be downloaded for the radiologist to update the diagnosis on his images. The second phase will include a purchasable and updated version that will allow the radiologist to download the complete report displayed within the application. Depending on the popularity of EHR systems, a port containing similar functionality to the offline application will be introduced as an extension for the EHR system. All of this will be run through either an on-premises server or alternatively through a cloud service.

Support

Support will be provided through online documentation within the desired website. The documentation will provide a guide on how to use the application while a dedicated email address will be provided for the sake of allowing interaction between hospitals and the company. In a future phase, a forum will be built that would allow users to comment on the issues that they may be encountering.

Release Planning

The first release will be considered as version 1.0. Any version before this release will be considered beta or experimental and only recommended for experimentation and feedback. After version 1.0, the product will be provided with both a free version and a paid version. In the case that the paid version does not reach targets, then a price range will be set in place while extending the ability for the user to extract more data from the DICOM file.

**Application Functionality and Execution**

The application can be accessed through the page [Web App - Unavailable (capstonetumorgui.azurewebsites.net)](https://capstonetumorgui.azurewebsites.net/)

Alternatively, the entire application is available within the following Github page:

[wpeguero/capstone: Capstone Project information. This contains the algorithm used for the Master's in Data Science Thesis. (github.com)](https://github.com/wpeguero/capstone)

As a third alternative, a zip folder has been sent as part of the project.

**User Guide**

This is the guide to using the application known as “The Capstone Project”. Once the application has been fully setup, the user may be able to access the website. The website will appear similar to the image below:

Graphical user interface, application

Description automatically generated

Click on the drag and drop or Select Files area to load multiple images (alternatively, the images can be dragged to the same area.

A computer screen capture

Description automatically generated with medium confidence

Select the image(s) and click open. Some time may pass before the images are displayed, but the following is part of the dashboard after the images load:

Chart, bubble chart

Description automatically generated

Below is the second half of the dashboard:

Shape

Description automatically generated

The export button can then be used to download a copy of the table which will display as Data.csv.

# Bibliography

Brady, A. P. (2016). Error and discrepancy in radiology: inevitable or avoidable? *Insights into Imaging*, *8*(1), 171–182. https://doi.org/10.1007/s13244-016-0534-1

Concordia University. (2015, June 16). *Can personal devices interfere with hospital care? Study examines whether wireless transmitters represent a danger to patients*. ScienceDaily. https://www.sciencedaily.com/releases/2015/06/150616123921.htm#:~:text=New%20research%20helps%20define%20a

Crotti, N. (2020, November 9). *How does HIPAA affect medical devices?* Medical Design and Outsourcing. https://www.medicaldesignandoutsourcing.com/how-does-hipaa-affect-medical-devices/

Evans, R. S. (2016). Electronic Health Records: Then, Now, and in the Future. *Yearbook of Medical Informatics*, *25*(S 01), S48–S61. https://doi.org/10.15265/iys-2016-s006

MedlinePlus. (2019). *Diagnostic Imaging*. Medlineplus.gov; National Library of Medicine. https://medlineplus.gov/diagnosticimaging.html

Silverman, L. (2017, September 4). *Scanning The Future, Radiologists See Their Jobs At Risk*. NPR.org. https://www.npr.org/sections/alltechconsidered/2017/09/04/547882005/scanning-the-future-radiologists-see-their-jobs-at-risk