Designing an AI-driven System at Scale for Detection of Abusive Head Trauma using Domain Modeling

by

Aditya Vikram

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2020 by the
Graduate Supervisory Committee:

Javier Gonzalez-Sanchez, Co-Chair
Ashraf Gaffar, Co-Chair
Michael Findler

ARIZONA STATE UNIVERSITY

May 2020

ABSTRACT

Traumatic injuries are the leading cause of death in children under 18, with head trauma being the leading cause of death in children below 5. A large but unknown number of traumatic injuries are non-accidental, i.e. inflicted. The lack of sensitivity and specificity required to diagnose Abusive Head Trauma (AHT) from radiological studies results in putting the children at risk of re-injury and death. Modern Deep Learning techniques can be utilized to detect Abusive Head Trauma using Computer Tomography (CT) scans. Training models using these techniques are only a part of building AI-driven Computer-Aided Diagnostic systems. There are challenges in deploying the models to make them highly available and scalable.

The thesis models the domain of Abusive Head Trauma using Deep Learning techniques and builds an AI-driven System at scale using best Software Engineering Practices. It has been done in collaboration with Phoenix Children Hospital (PCH). The thesis breaks down AHT into sub-domains of Medical Knowledge, Data Collection, Data Pre-processing, Image Generation, Image Classification, Building APIs, Containers and Kubernetes. Data Collection and Pre-processing were done at PCH with the help of trauma researchers and radiologists. Experiments are run using Deep Learning models such as DCGAN (for Image Generation), Pretrained 2D and custom 3D CNN classifiers for the classification tasks. The trained models are exposed as APIs using the Flask web framework, contained using Docker and deployed on a Kubernetes cluster.

The results are analyzed based on the accuracy of the models, the feasibility of their implementation as APIs and load testing the Kubernetes cluster. They suggest the need for

Data Annotation at the Slice level for CT scans and an increase in the Data Collection process. Load Testing reveals the auto-scalability feature of the cluster to serve a high number of requests.

DEDICATION

To Mom for her relentless belief in me.

To Dad for his love for Pediatrics.

To my brother for his companionship and support.

And finally, to this institution for the opportunity.

ACKNOWLEDGMENTS

I would first like to express my sincere gratitude to my co-chairs, Dr. Javier Gonzalez-Sanchez and Dr. Ashraf Gaffar, and Dr. Lois Sayrs from Phoenix Children Hospital, for their continuous guidance and support which made this work possible. Throughout the course of the thesis, they have helped me navigate academic and logistics problems and constantly shared their constructive feedback with me.

I would like to thank Dr. Michael Findler for agreeing to be part of my committee, reviewing my work and providing feedback. I would like to also thank many collaborators who have helped me throughout the project, in particular, Bethany Sussman, for helping me with the data acquisition, Jon Plasencia, for his guidance on the project and, Jonathan Turner, for all his help with the logistics of the data. I would also like to thank Jodie and Anas for their help in the study.

Last but not the least, I would like to thank my family who have always been a constant support and my friends in Tempe who have helped me remain sane amongst all the chaos happening around.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Abusive head trauma (AHT) can be defined as an inflicted injury to brain or skull due to intentional impact or violent shaking (Paul et al. 195). In 2009, the American Academy of Pediatrics endorsed the nomenclature “abusive head trauma” (AHT) as the most comprehensive term describing the constellation of injuries to the brain and spine that result from inflicted head injuries to infants and children (Christian et al. 1409). AHT affects 14-40 cases per 100,000 live births (Fanconi et al. 1023; Sieswerda-Hoogendoorn et al. 617; Hobbs et al. 952; Keenan et al. 621). An estimated 18-25% of children diagnosed with AHT die, and up to 80% of survivors will live with significant lifelong physical, developmental, and emotional sequelae (Currie et al. 111; Damashek et al. 735; Glaser 97; Springer et al. 517). AHT is associated with the highest risk of death in infants (Springer et al. 517) and toddlers under 4 years of age (Palusci et al. 25). The actual incidence of AHT is unknown since it is clinically difficult to differentiate from accidental head trauma except in the most serious, near fatal or fatal patients. Patients may present with symptoms that are non-specific to head trauma such as lethargy, vomiting and fussiness. As much as 31% of children with AHT may be misdiagnosed on initial visit (Letson et al. 36). Establishing a correct diagnosis requires taking into account all the clinical aspects of the case, radiological studies and caregiver’s story of how the injury happened. A wrong diagnosis means putting the child’s life in risk of re-injury or death. In a study done by (Hettler et al. 602), 80% of deaths could have been prevented in the study group if AHT could have been diagnosed earlier.

Computer Tomography (CT) and Magnetic Resonance Imaging (MRI) are common diagnostic tools to investigate the likelihood of AHT in a hospital. If pediatric patients present in anther medical setting, providers typically refer patients to a hospital where these diagnostic tools can be used. Radiologic indications are well-described for AHT and include seizures, multiple subdural hematoma (SDH) over convexity, subarachnoid and subaxial hemorrhages, posterior fossa SDH, hypoxic ischemic injury (HII) and edema (Kemp et al. 1103; Feldman et al. 636). However, Kemp (Kemp et al. 1103) suggest relying on SDH as a radiologic finding associated with AHT only when presenting with unexplained traumatic head injury, where no explanation has been provided or the explanation provided did not match the injury pattern. There are few significant differences in imaging and clinical characteristics between non-fatal injured and fatal injured patients (Sarnaik et al. 302). Fatal injured AHT patients demonstrate more bleeding on CT scans (Rolfes et al. 30).

Few radiologic findings are specific to AHT. Moreover, radiologists require specialized training in child abuse and neglect to be able to correctly identify radiologic features of AHT. To date, few hospitals in the US have the experience and expertise to provide definitive diagnoses of AHT. Radiologists with specialized training to detect AHT consider a number of key features such as the amount of blood, tissue death, age of the blood, and the estimated amount of force required to incur the observed injury. With the advent of Machine Learning, specifically deep learning algorithms, there is a promise of replicating radiologic findings of the most knowledgeable and experienced pediatric neurological radiologists but perhaps also provide a diagnosis earlier, with fewer features to prevent subsequent injury and death.

Deep Learning algorithm, specifically, Convolutional Neural Networks (CNN) have performed really well on Computer Vision Tasks such as Image Classification, Image Segmentation and Object Detection. Its use in the medical field for Computer Aided Diagnosis has shown really good results in applications (Mcbee et al. 1472) such as lesion detection in chest X-rays (Detection of tuberculosis, pleural effusion, cardiomegaly, mediastinal enlargement) (Bar et al. 294; Lakhani et al. 574), lung nodules classification in chest X-rays (Nibali et al. 1799; Ciompi et al.), classification of tumors in mammograms (Huynh et al.), brain MRI segmentation (Moeskops et al. 1252; Akkus et al. 449), hemorrhage detection using CT scans (Kuo et al. 22737; Grewal et al.), etc. Deep Learning algorithms are able to detect features from data automatically which has helped in their widespread use. Increasing the number of layers leads to extraction of more subtle features which gives better results.

1.1 Challenges

Despite the advantages of Deep Learning algorithms, medical data is currently underutilized in healthcare settings for predictive analytics. There are multiple reasons for this, starting with the proprietary nature of the data which under HIPAA is not easily accessed outside of medical research. Medicine has now incorporated more formal informatics triaging for advanced clinical professionals but there remains a lag. Healthcare data is collected for clinical purposes and not necessarily research purposes; often data from clinical systems needs to be extracted and reformatted to make it usable for research purposes, a time-consuming and potentially expensive process. A major obstacle is the amount of data required my Deep Learning algorithms and machine learning in general.

Often a medical condition, even if fairly frequent, may still take years to collect sufficient number of cases or require a collaboration of hospitals to jointly contribute data, a daunting prospect with numerous legal obstacles regarding data sharing and security of propriety personal health information. Even if sufficient imaging data were collected, the computer hardware and software requirements are not typically available in medical settings unless the research is funded by a major health funding source such as NIH. Another obstacle is expertise required to execute DL algorithms. Even national Hospitals and Healthcare networks may not have the technological expertise, and would likely require collaboration among hospitals, researchers and engineers. All of these obstacles have hindered advancement of research in area of Computer Aided Diagnosis.

1.2 Problem to be Addressed

This thesis serves as a first attempt at solving all the stages of a novel medical task, i.e., Detection of AHT in pediatric patients, using Deep Learning. The work includes building a production ready software capable of holding the Deep Learning algorithm which can be used as a screening tool for all head trauma cases of children less than two years by healthcare professionals.

With the goal of building a software capable of classifying Accidental Head Trauma from Abusive Head Trauma, the thesis model the classification task domain. The modeling hypothesis is inspired from modeling the problem of detecting Skin Cancer (“IBM Watson Cognitive Service for Visual Recognition”). The work considers the modelling task divided into levels arranged in a stack (one on top of the other) as shown in Figure 1.

- The base level (Level 1) refers to the classification task done by humans. It means that the features in an image are differentiable by human eye. An example would be a dermatologist being able to detect lesions in skin images easily. The images in Level 1 have clear demarcations in features which make them easy to detect.
- Level 2 refers to classification done by Machine Learning (or Deep Learning algorithms). The feature detection in Level 2 is hard for human eye but easy for Deep Learning algorithms. An example would be work done by (Esteva et al. 115) using Deep Neural Networks. CNNs (algorithm used for classification of images) consist of Convolution and Pooling layers which are responsible for feature detection. The learnt features are later passed through a Fully Connected Network for final result.
- Level 3 adds extra visual cues to the data and feed the data to Deep Learning algorithms. Visual Cues refer to feature engineering and image annotations on complex images. The feature identification part is usually done by domain experts (e.g. radiologists for medical data). Recent studies have shown CNNs and GANs (Generative Adversarial Network) being used for feature annotation in images.
- Level (Level 4) is the top level and includes other cues (such as natural language reports about the image, information about the image) which would help in the modelling task. Electronic Health Records, Patient information, etc., can be used with the images for building ensemble networks capable of classification task with high accuracy.

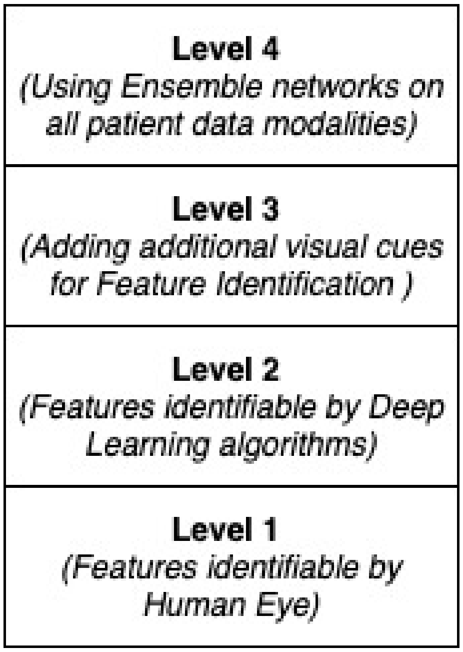

Level 4
(Using Ensemble networks on all patient data modalities)
Level 3
(Adding additional visual cues for Feature Identification)
Level 2
(Features identifiable by Deep Learning algorithms)
Level 1
(Features identifiable by Human Eye)

Figure 1: The Domain Modeling Hypothesis

1.3 Proposed Solutions and Contribution

The thesis models the domain of AHT, and subsequently, Traumatic Brain Injury (TBI) and test the modeling hypothesis for Level 2. The modelling helps in understanding the problem domain and its components. This leads to building of robust Deep Learning models for classification and generation of CT scans. The thesis also proposes a production ready web tool which holds the models. The work can be used by the healthcare professionals at PCH for screening all the TBI patients less than two years of age. Finally, the work will also lay down a foundation for further research and collaboration between researchers at ASU (Arizona State University) and PCH.

The thesis starts with Data Gathering for classification task which itself was a challenge. The data was procured from PACS (Picture Archiving and Communication System) at PCH. A major contribution to thesis was Data Anonymization, Cleaning and

Organization (done at PCH by the author due to HIPPA compliances). The success of Deep Learning lies behind massive amount of data and computing power. Due to availability of limited data from PCH, the work proposes use of Generative Adversarial Networks (GANs) for data augmentation and synthesis. The classification models are built using state-of-the-art pre-trained architectures and the results are compared.

Additionally, a big part of the thesis is bridging the gap between model development and its deployment as a production level software. The work investigates the use of container orchestration framework, specifically Kubernetes, for high availability and scalability of the model. Finally, the models are deployed on a web tool which can be accessed using a web browser.

The work gives an overview of AHT, its incidence, clinical and radiological interpretations. It discusses the challenges faced in a hospital environment for carrying out research when patient data is involved. A literature survey of existing work in the area of Computer Aided Diagnosis and Radiology is also discussed.

1.4 Evaluation Plan

The evaluation of the proposed solution includes:

- Evaluation of the Deep Learning Models: The classification models are tested based on the binary accuracy generated when the models are fitted on the validation data set. The hyperparameters of the models are updated based on the test result.
- Evaluation of the Software tool: Unit testing, Integration testing and Alpha testing are done based on best practices in Software Engineering. The Kubernetes cluster is load tested using Siege (a tool to test simulated traffic on Kubernetes cluster).

1.5 Document Roadmap

The rest of the thesis is structured into the following sections:

- *Chapter 2* establishes the context of AHT by summarizing the Medical Background in detail.
- *Chapter 3* describes the literature review of the key concepts used in the thesis.
- *Chapter 4* provides an in-depth review of the implementation details of the Deep Learning models and the software architecture.
- *Chapter 5* evaluates and analyzes the results using the evaluation plan.
- *Chapter 6* provides detail about the challenges faced, the conclusion and the scope for future work.

CHAPTER 2

BACKGROUND MEDICAL KNOWLEDGE

Abusive Head Trauma (AHT) refers to child abuse resulting in head injuries. It is the leading cause of death in children below the age of two from traumatic injuries. It is estimated that 20 out of every 10000 infant hospital admits are due to abusive head trauma. The number is not an accurate reflection of the incidence since AHT is often under-recognized and under-reported (Frasier et al.; Runyan and Desmond 112; Keenan et al. 621). A common problem with diagnosing Abusive Head Trauma is lack of gold-standard medical test which increases the burden of physicians. They have to consider multiple factors such as history of the patient, history provided by the caregiver (which might be skewed or misleading) and radiological studies. Since AHT patients are at a greater risk of re-injury or death, a confirmed diagnosis can save lives. But an incorrect diagnosis can lead to distress in family, emotional trauma and legal issues. Generally, multiple departments (Radiologists, ER specialists, Surgeons, Child Services Workers, etc.) work in tandem to come with a confirmed diagnosis of AHT.

2.1 Literature

AHT is also called Inflicted Head Trauma, Inflicted Pediatric Neurotrauma, and Non-accidental Head Trauma (Frasier et al.; Christiamn et al. 1409). The distinction between AHT and Accidental Head Trauma is minimal. The patients are usually brought into the Emergency Department with clinical symptoms of injuries or bruises from household falls (seldom poorly explained by the caregiver), respiratory distress or seizure disorders, or more non-specific symptoms such as crying, vomiting or irritability. Patients with non-specific symptoms and lack of external trauma are more likely to go un-diagnosed (Jenny et al. 621). A study presented by (Talvik et al. 782) stated that there is a high correlation between crying in babies and AHT. Infants below the age of 1 year are at a higher risk of AHT with median age of patients being 2 months – 6 months (Hettler et al. 602; Reece et al. 11; Keenan et al. 621; Herman et al. 65). This coincides perfectly with the peak crying age in children.

There are multiple challenges faced by physicians when tasked with diagnose AHT in patients. The first and most important challenge is the presentation of the history of trauma by caregivers. Since most of the diagnosed AHT patients are infants, the history of trauma is provided by the parents and which is often misleading or skewed. The physicians have to rely on patient evaluations and studies and then corroborate with the history provided by the caregiver. Another challenge faced by diagnostic clinicians is that they do not use validated screening tools for diagnosing AHT. Most hospitals are not ‘pediatric-focused”; they see considerably more adult than pediatric patients. A study conducted by (Jenny et al. 621) found that 54 out of 173 (31.2%) cases were missed by Physicians as AHT and were later diagnosed by a different physician with mean time to correct diagnosis being 7 days. It also stated that 4 out 5 deaths in the study group could have been prevented with a correct diagnosis from the beginning.

Radiology plays a vital role in establishing a correct diagnosis of suspected AHT. One of the first tests to perform if AHT is suspected, is, a Computer Tomography (CT) scan. It is often chosen because it does not require sedation and the vast majority of patients who report an injury typically describe some type of “fall”. CT is the best diagnostic tool to identify any possible fractures and underlying any intracranial pathology (Sieswerda-Hoogendoorn et al. 617). Even though CT scans involve a high dose of radiation, pediatric radiologists follow APA and radiologic guidelines for suggested doses which are as low as possible for pediatric patients. The radiologist can identify conditions such as hemorrhage, hematoma, midline shifts, etc. Magnetic Resonance (MR) scans are less common especially if a fracture is found via CT and no underlying symptoms or conditions (such as loss of consciousness, lethargy, vomiting, and seizures) indicates head swelling or tissue impairment. The interpretation of the scans requires a radiologist trained in child abuse and pediatric radiology. However, the radiologist does not typically use additional clinical information on the patient to complete the report. The report acts as an important evidence for the diagnosis of AHT (Sieswerda-Hoogendoorn et al. 617).

2.2 Comparing Abusive and Accidental Head Trauma

Distinguishing Abusive Head Trauma from Accidental Head Trauma is a challenging task even for trained specialists. Even though incidence of features such as Subdural Hematoma, Retinal Hemorrhages, Cerebral Edema/ Ischemia, Apnea, etc., are very high in AHT cases when compared to Accidental Head Trauma, a definitive diagnosis is still very difficult due to lack of a gold standard testing process (Piteau et al. 315; Kemp et al. 1103; Hymel et al. 1537; Cowley et al. 290). Hospitals have been working on establishing concrete markers for the diagnosis of AHT and distinguishing them from Accidental Trauma. (Amagasa et al. 265) conducted a study to identify the definitive markers of AHT in Japanese population and how the markers differentiate AHT from Accidental Head Trauma. The study had 166 clinical cases in total where 57 cases were Definite AHT, 24 were suspected AHT and 85 were accidental cases. Radiological studies, clinical notes and outcomes were compared between the groups. The study found that clinical symptoms such as retinal hemorrhages and bruising were higher for the AHT group when compared to the accidental group while the scalp findings had an opposite incidence. The CT reports revealed the prevalence of Subdural Hematoma and cerebral edema/ ischemia to be higher in the AHT group while Skull fractures and Epidural Hematoma were more prevalent in the accidental group. Even though the study establishes concrete markers for AHT, the results could not be generalized since the age distribution of the AHT study participants was significantly higher when compared to countries such as USA. Another interesting find of the study was the absence of clinical findings (present in other studies) such as bruising or rib fractures in the AHT group. The study also concluded that presence of Subdural Hematoma without injuries to skull (from an impact) is highly suggestive of AHT.

(Bechtel et al. 165) conducted a study at Yale New Haven Children's Hospital to determine the clinical features in pediatric cases of age below 2 years which differentiate AHT from Accidental Head Trauma. The study group consisted of 87 cases divided into 15 AHT cases and 72 Accidental Head Trauma cases. The primary measure was dilated ophthalmic retinal scans to look for retinal hemorrhages while the secondary measure was using CT scans to look for intracranial injury, skull fractures or bleeds. The patient's mental status and clinical presentations (such as occurrence of seizures or convulsions) was also taken into account. The study found out that the AHT group was more likely to have Subdural Hematomas while presence of Epidural or Subarachnoid Hemorrhages, and, Skull fractures were even in both the groups. 60% of AHT group had retinal hemorrhages while only 10% of the Accidental Head Trauma group had them. Seizures and Abnormal Mental Status was more common in the AHT group. Presence of Scalp hematomas were more common in Accidental Head Trauma group when compared to AHT group.

Retinal Hemorrhages are the hallmarks of AHT in pediatric patients. However, these cannot be diagnosed without an ophthalmological consult which occur for only a handful of patients highly suspected of AHT. The consult happens only after a child is admitted; therefore, children discharged from the Emergency Department would never have the opportunity to receive the consult. A literature survey done by (Kemp et al. 1103) checked the evidence behind the neurological features associated with AHT and how they are different for Non-AHT cases. In total, the data from 2353 patients from 21 studies were selected which was divided into 893 cases of AHT and remaining 1460 cases of Non-Abusive Head Trauma. A clear association between Subdural Hemorrhage and AHT was defined from the studies. These hemorrhages were frequently multiple (in closed head injuries) and present in the posterior fossa and the interhemispheric fissure. Extradural hemorrhages were associated with Non-AHT. Hypoxic Ischemic Injuries and cerebral edema were significantly associated with AHT. Most of the findings were derived from CT studies but MRIs would be preferred in future because of its increased sensitivity. Clinical features such as retinal hemorrhages, apnea, rib bruises and fractures and absence of skull fractures (but presence of intracranial injury) was confirmed in an earlier literature study by the same author (Reading 150). It is worthwhile noting that milder forms and early detection of AHT is not aided by clinical features such as Retinal Hemorrhages, etc.

The markers derived from the studies listed above can be identified as clinical presentations or derived from radiological studies (such as CT scans). Understanding these features and their presentation helps in understanding the domain of AHT and, subsequently, Traumatic Brain Injury (TBI).

2.3 Subdural Hematoma

Subdural Hematoma (SDH) (also called Subdural Hemorrhage) is the collection of blood in the subdural space. The subdural space lies between the meningeal layer of dura and sub-arachnoid layer which is attached using tight joints. Bridging veins from dura to sub-arachnoid layer may cause bleeding and strip the two layers, filling with blood. This collection of blood is referred to Subdural Hematoma (Salazar et al.). Figure 2 explains the anatomy of brain and highlights the Dura area and Subdural Hemorrhage.

The occurrence of SDH vary with the age of the patient. For infants, SDH is an indicator of non-Accidental Trauma while for neonates or newborns, these are associated with delivery labour. The incidence for these cases is very less. For other population, SDH are result of high blunt force trauma such as Motor Vehicle collisions. In patients with AHT, SDH is presented with other symptoms such as altered neurological status, coagulopathy, cerebral atrophy, cysts, or bruises, etc.

A confirmed diagnosis of Subdural Hematoma requires a CT scan (and in some cases, an MRI scan). Figure 3 demonstrates the presence of SDH in a CT slice. It can be seen on the left part of the brain in the CT slice.

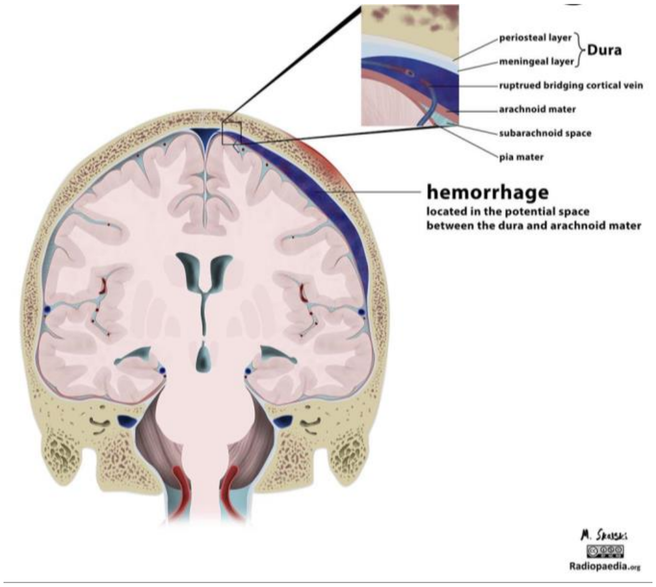


Figure 2: The Brain Anatomy Demonstrating Subdural Hemorrhage; *Source: Case courtesy of Dr Matt Skalski, Radiopaedia.org, rID: 21542*

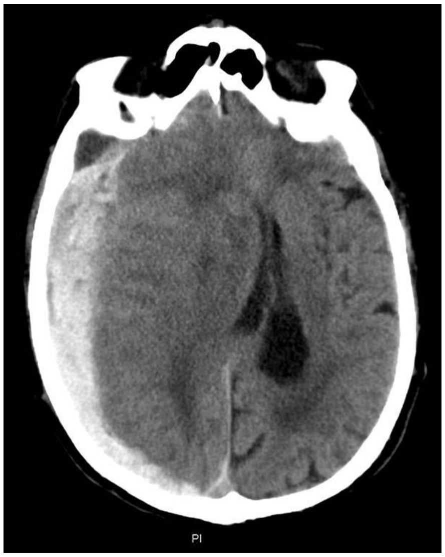


Fig 3: SDH in CT Slice; *Source: Case courtesy of Assoc Prof Frank Gaillard, Radiopaedia.org, rID: 35891*

2.4 Retinal Hemorrhages

The eyeball is made up of 3 layers, namely, sclera, choroid, and, retina (as observed from transversion section). Sclera is the white outermost layer of the eyeball used to provide attachment for the muscles that control the eye movement. Choroid is the middle layer used for regulating the eyeball temperature. It also provides blood flow to the retinal layers. Retina is the innermost layer used for vision (Hansen et al. 671). The retina itself is multilayered. Blood in the initial retinal layer (preretinal), within the layers (intraretinal) or in the last retinal layer (sub-retinal) are called as retinal hemorrhages. Figure 4 clearly demonstrates the anatomy of the eye and shows the three layers of the eyeball.

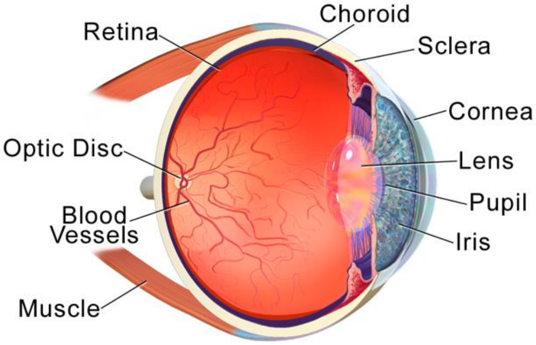


Figure 4: Anatomy of the Eye; *Source: Blausen.com staff (2014). "Medical gallery of Blausen Medical 2014". WikiJournal of Medicine 1 (2). DOI:10.15347/wjm/2014.010.*

Retinal Hemorrhage are mainly caused by head trauma (such as blow to the head from accidents), hypertension, and, blockage in retinal vein. They are diagnosed by an ophthalmologist by performing a dilated fundoscopy with indirect ophthalmoscopy. The test is often indicated for children with intracranial hemorrhages to diagnose AHT (Levin et al. 376). Figure 5 shows a normal fundus after fundoscopy while Figure 6 shows presence of Retinal Hemorrhage in the fundus after the fundoscopy.

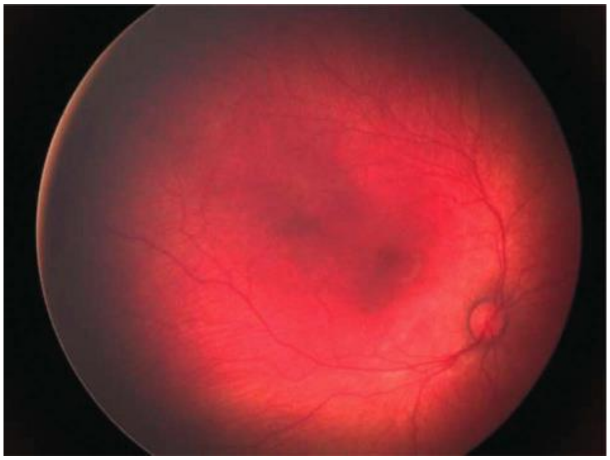


Figure 5: Normal Fundus; *Source: Image Credit Laura S. Plummer, MD.*

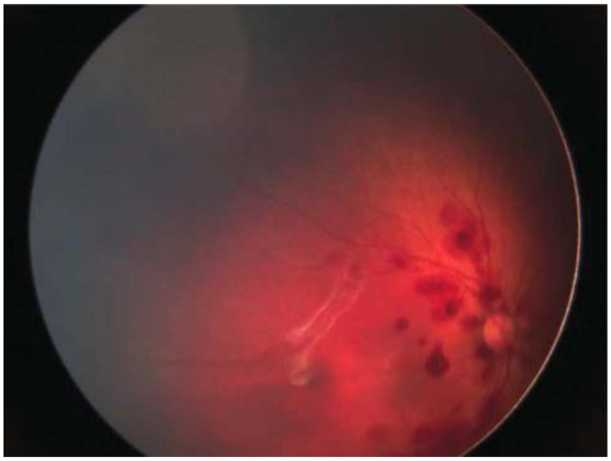


Figure 6: Non-specific Retinal Hemorrhage; *Source: Image Credit Laura S. Plummer, MD.*

Although retinal scans are informative, they are not widely used as a diagnostic tool since they require an ophthalmological consult and available only for highly suspicious cases. This is the reason why they have not been used in the research.

2.5 Other Features Presented in AHT

Other features which are observed in children diagnosed with Abusive Head Trauma are Cerebral edema, skull fracture with Intracranial Injury and Apnea. Cerebral Edema is the swelling of brain due to lack of blood supply which constricts the amount of oxygen in Brain Tissues. They are often presented with SDH in AHT cases.

Skull fractures are fractures in the cranial bone surrounding the brain (the skull). They are caused by direct blow to the head (from collisions or accidents). Skull fracture in children are often indicative of Abuse if there is no history of accidents. They can be diagnosed using an X-ray or CT scan.

Apnea is a condition where the breathing mechanism ceases and the movement of respiration muscles halts (“Apnea”). Apnea is a classical marker to distinguish AHT from accidental head trauma indicating high association of apnea with AHT cases.

CHAPTER 3

KEY CONCEPTS

This chapter illustrates the key concepts and methods required to understand the work completed in the thesis. The chapter starts with introducing the concept of Domain Modeling, and then, talks about the field of Computer Vision and Image Processing. A brief introduction to Deep Learning in Computer Vision is provided, and then the key architectures, namely, Artificial Neural Network, Convolutional Neural Network and Generative Adversarial Network are explained. The chapter also talks about use of Transfer Learning and then moves onto Deploying Machine Learning Models using Docker and Kubernetes.

3.1 Domain Modeling

The thesis proposes a model for the problem domain of Abusive Head Trauma in Pediatric Patients using Deep Learning Techniques. According to (“What is the Domain Model”), a domain model is an organized and structured knowledge of the problem, representing important concepts and challenges of the problem domain, and identifying the relationship among the entities within the specified domain. A simplification would be scoping the problem domain, understanding the challenges and providing hypothesis for the challenges (backed by experiments). AHT domain is modeled based on existing literature on Deep Learning use cases in Healthcare, and, provides hypothesis to the problems in the domain using Software Design.

With the explosion of big data and abundant computing power, there has been a paradigm shift in the way problems are solved. Traditionally, building applications involved deducing rules from requirements which governed the application and subsequently encoding them as source code. But with the advent of Artificial Intelligence, there is a move towards automation and solving problems using the historical data and its underlying properties (Khomh et al. 81). Some interchangeable words often used for these techniques are Artificial Intelligence (AI) and Machine Learning (ML). Artificial Intelligence is an umbrella term used to define building systems with intelligent capabilities, while Machine Learning is a set of learning techniques where machines learn from data without being programmed explicitly. Healthcare is a promising industry for AI with use cases in Computer Aided Diagnosis, Electronic Health Records, etc. Computer Aided Diagnosis refers to assistive systems which help doctors in diagnosis of a disease using medical images ("Computer-aided Diagnosis"). Deep Learning (DL), a subset of Machine Learning (ML), is thoroughly being used for building the diagnosis assistive systems. DL algorithms are capable of extracting features from medical images which are not visible to the human eye and using them for diagnosis.

Solving a novel problem requires a thorough understanding of the domain, its components and the challenges. This is where domain modeling proves to be really helpful. It is an essential step in design of any system (using traditional techniques or AI). The domain is the description of the problem being explored. It provides sufficient information on the background of the problem, existing literature and the impact of the problem. It breaks down the problem into entities/ components with each entity having a specific use and meaning. These entities can be represented as abstractions of smaller problems in the domain. It is important that the domain is scoped, and the problem is defined. The solution to the problem described by the domain, is the model ("What is the Domain Model"). It is an explanation to the different entities of the domain. The model provides solutions using hypotheses and experiments to these components (which may be abstractions of smaller problems) or specific and structured knowledge about these entities. The knowledge can be later used for designing systems which represent the domain model. Such design technique is called as Domain Driven Design (DDD).

Formally, DDD is defined as a model driven approach to designing software application representing the target application domain, its entities and relationships in form of Domain Model, which is used to design the application architecture (Rademacher et al. 230). Figure 7 perfectly encapsulates the Domain Driven Design process where, the Domain model is embedded in the Software Development Process. Once a clean Domain Model is ready, it can be applied to the surrounding use cases and then the infrastructure can be built for those use cases.

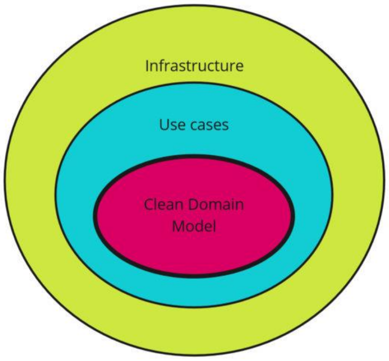


Figure 7: Domain Model in Software Development; *Source: (Grzybek "Attributes of Clean Domain Model.")*

3.2 Computer Vision and Image Processing

Computer Vision (CV) is a field of Computer Science and Artificial Intelligence related to helping computers “see” and “analyze” real world objects through multimedia inputs such as images and videos (Brownlee “A Gentle Introduction to Computer Vision”). Understanding the image means understanding the objects in the image or understating the image, as a single entity. Image Processing consist of set of techniques used to create new images from original input images by modifying them. Modification may include cropping images, changing resolution, changing color scheme, altering image angles, removing noise, etc.

Computer Vision, sometimes also referred to as visual recognition problems, is an evolving field which utilize Image Processing techniques (sometimes as a pre-cursor) to understand complex images and extract features. The domain of Medical Image Analysis is a Computer Vision Task since it requires computers to understand medical imaging modalities such as Computer Tomography (CT) scans, Magnetic Resonance Imaging (MRI), Positron Emission Tomography (PET) scan, etc. CV is still a challenging field (due to inherent complexity of the input images) with new research being published every day. Computer Vision and Pattern Recognition (CVPR), a top conference for Machine Learning and Computer Vision, reported a 56% increase in paper submissions from 2018 to 2019 (“General Welcome and the Organizing Committee”).

Computer Vision applications for Medical Image Analysis task include Image Classification, Image Segmentation, Image Detection, Image Generation and Image Translation. Some popular use cases where CV is used extensively are Self-driving and

Autonomous Vehicles, Object Character Recognition (OCR), Face Detection and Recognition, Video Surveillance, Motion Capture, etc.

3.3 Deep Learning in Computer Vision

Deep Learning is a subset of Machine Learning algorithms related to Artificial Neural Network (ANN). ANN is a Machine Learning and Data Mining technique, loosely inspired from human brain and its workflow.

The below figure (Figure 8 (b)) gives an overview of an ANN and computation in the nodes for each layer. Neural Networks are loosely based on human neurons. The first layer is called the input layer (i) responsible for feeding in the data. The last layer is the output layer (l) which gives the result (or prediction) of the algorithm. All the layers in between input and output layers are called as hidden layers (j, k). Most ANNs have a forward and backward pass. They are trained by feeding in historical labeled data (known outputs to each input). This type of training is referred to as Supervised Learning.

Each layer consists of computation units called nodes (represented by circles in the picture) in Figure 8 (a). Nodes are interconnected in nature, i.e., the output of one node is fed into input of all the nodes in the next layer. The inputs to each node (equivalent of synapses in a human neuron) have a *weight* associated with them. The weights can be thought of as the importance of the input in the final result computation. Figure 8 (a) gives an overview of the computation inside a node (including the summation operation and the activation of the result).

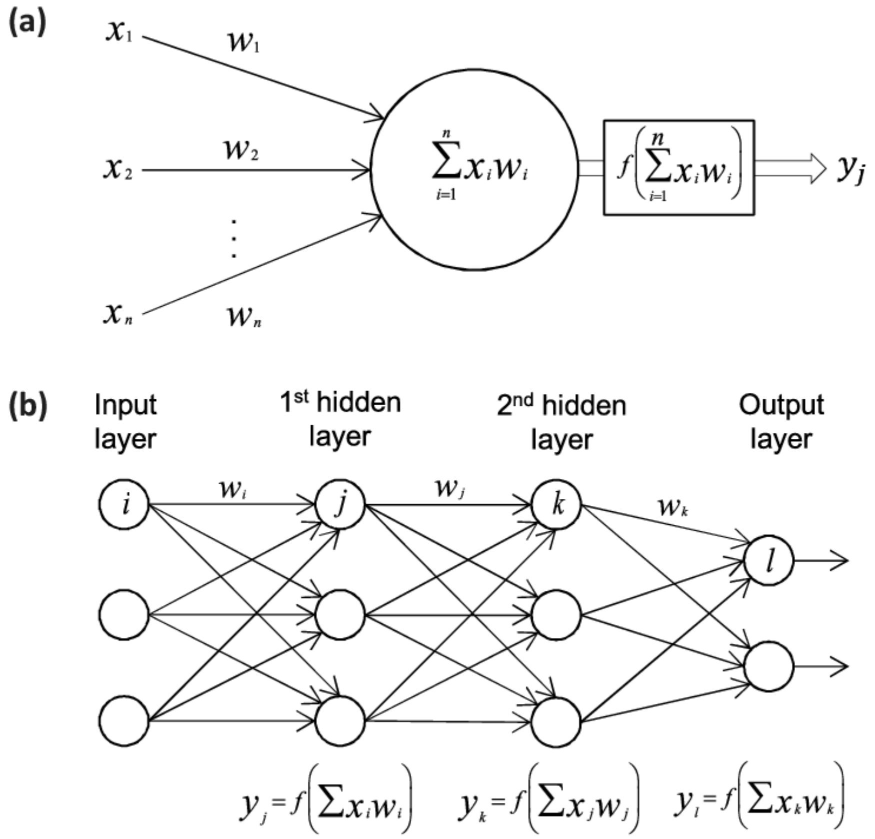


Figure 8: a) Computation Inside a Node b) A 4-layer ANN; *Source: (Vieira et al. 58)*

In the forward pass, the weights are multiplied with the input values and summarized. After the summation operation, the value is fed through an *activation function* (to introduce non-linearity) which decides whether an input should be passed along or not. The computation is run along till the output node. The output nodes calculate the predicted result. The predicted output is compared with the original expected output using a *cost function* and error minimization algorithms such as *gradient descent*. An error is generated using cost function and the goal here is to minimize the error. In the backward pass, the error is fed backwards in the network using a process called *backpropagation*. The idea here is to update the weights using the propagated error using *optimization algorithms*. Once the error has been backpropagated, the ANN does another forward pass to calculate the output. The process of forward and backward pass keeps repeating until the loss is minimized and weights are adjusted to the optimal level. The trained model can then be used to make predictions on new data.

Neural Networks (NN) have been around for a long time with the first model based on Human neurons, McCulloch-Pitts model, being published in 1943. Machine Learning has come a long way since then. Traditional ML algorithms required Feature Extraction as a pre-cursor to training the model. Feature Extraction was an expensive technique which often required expert domain knowledge. With the availability of cheap computing power and rich datasets, Deep Learning was able to extract features from data automatically using "deep-layers" in the NN (as shown in figure 9). This led to tremendous growth in the field of Machine Learning with research pivoting towards Deep Learning instead of Traditional Machine Learning techniques. It is worthwhile noting that the power behind Deep Learning lies in the availability of a large amount of input data to learn the features from and computing power to train the models.

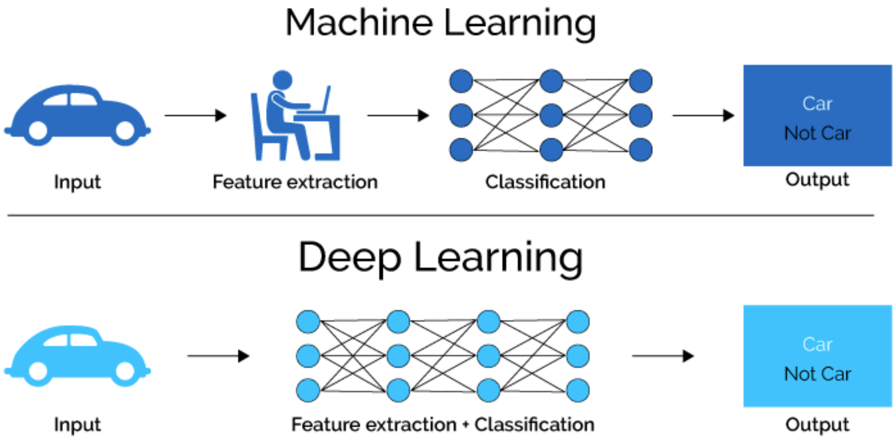


Figure 9: Machine Learning Versus Deep Learning; *Source: ("A.I. Technical")*

3.4 Convolutional Neural Network

Computer Vision tasks require Deep Neural Networks to use digital inputs such as images or videos; one of the most frequently used architectures is Convolutional Neural Network (CNN). One of earlier works on CNN was published by Kunihiko Fukushima in 1980, with a landmark paper being published in 1998 by the pioneers in the field of AI (Lecun et al. 2278), which introduced the LeNet-5 architecture to classify digits. But the credit for helping CNNs to utilize their full potential goes to (Krizhevsky et al. 84) who won the 2012 ImageNet challenge. ImageNet is an image classification challenge to classify over 1.5 million images in over 1000 classes. Since 2012, all the winners of the ImageNet competition have used a CNN like architecture.

The input to a CNN is an image (or an array of images for 3D models) in pixel format distributed over three input channels, Red, Green and Blue (RGB) or Greyscale. Figure 10 illustrates how images are encoded to provide input to CNN.

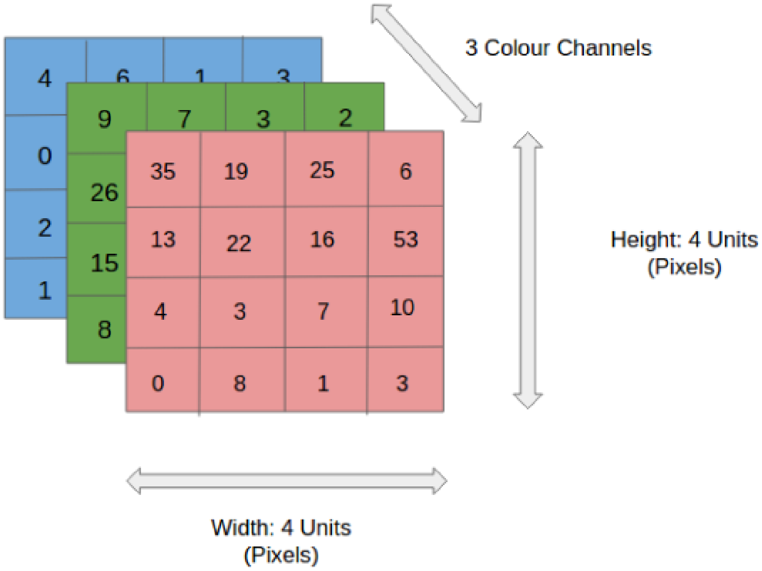


Figure 10: Input Image Encoded into Three Channels; *Source: (Saha)*

The main operations in a CNN are Convolution and Max Pooling. The idea is to learn high level and low-level representations/features from the images. Each image is passed through a series of convolution layers with kernels and pooling layers. The multiple convolution and pooling layers are used as feature extractors. The network is then connected to a Fully Connected Neural Network (FCN) which performs the target task of classification or segmentation. Figure 11 gives an overview of the CNN architecture.

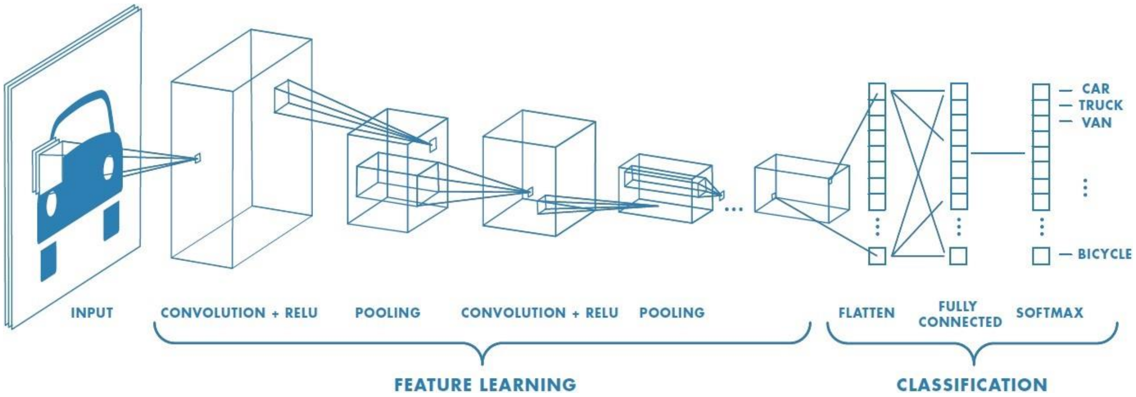


Figure 11: Architectural Overview of Convolutional Neural Network; *Source: (Prabhu)*

The convolution layer is used to extract features from the input image using kernels or filters. Kernels can be thought of as feature extractors (represented as a matrix). The convolution operation requires matrix multiplication of the input image and the filter. The depth of the filter is the depth of the image (number of channels). Different filters can be used to extract different features from the input image in the same layer (such as image blurring, sharpening, edge detection, etc.). A feature map is generated by sliding the filter over the entire image. The feature maps are then passed through activation function (e.g. ReLU in Figure 11) to introduce non-linearity. Figure 12 represents the feature maps.

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**

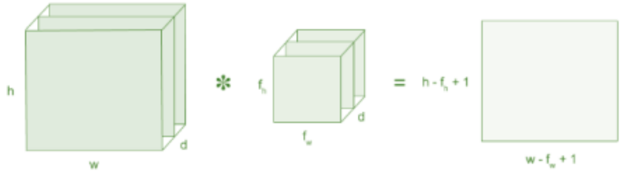
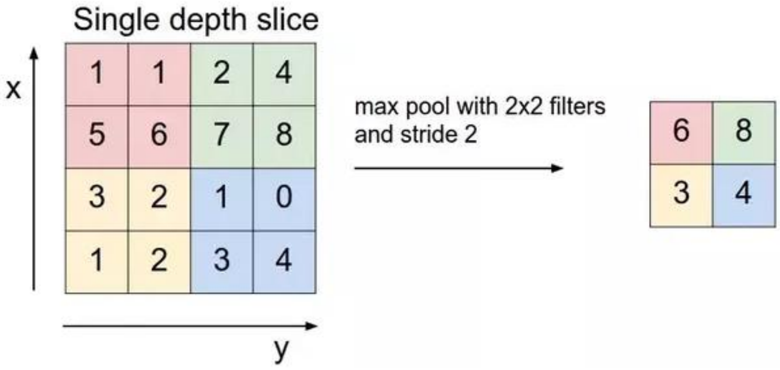


Figure 12: Overview of Convolution Operation; *Source: (Prabhu)*

The pooling layer reduces the size of feature maps while keeping the important information intact using max pooling or average pooling. This helps in decreasing the computational power required to process all the feature maps. Only the dominant features are kept while the non-dominant features are discarded. Figure 13 represents the max pooling operation.



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

Figure 13: Max Pooling Operation; *Source: (Prabhu)*

The output is then flattened and fed to a fully connected layer with a Softmax activation function (if the target task is classification). Backpropagation is used for training the network and building the model. Some state-of-the-art CNN architectures used in the thesis are ResNet50, DenseNet121 and InceptionV3.

3.5 Generative Adversarial Network

Another interesting use case of Computer Vision is Image Generation. Generative Adversarial Networks (GAN) were introduced in 2014 by Ian Goodfellow (Goodfellow et al.). Since then, GANs have been an area of extensive research with recent papers by NVIDIA (Karras et al.), showing state-of-the-art results in image generation.

The idea behind GANs is using two competing neural networks, a generator and a discriminator, which are adversaries. The job of the generator is to generate original-like images while the job of the discriminator is to identify whether the generated images are real or fake. During the training of GAN, the goal of the generator is to become better and better at fooling the discriminator while the goal of the discriminator is to get better at identifying whether the generated image is real or fake. Figure 14 illustrates the architecture of a Deep Convolutional GAN (DCGAN) (Radford et al.).

The input to the generator is random noise and it uses De-convolutional operations (using Transpose operation) for upsampling (increasing the dimension of) the noise. The generator never gets to see the original input images. The discriminator gets to see both the original images as well as images generated by the generator. The generated image as well as the ground truth (original) image is fed into the discriminator and it returns a probability whether an image is real or not. The discriminator uses simple convolution operations to check the authenticity of the images. The feedback is sent to the generator which tries to improve its images. Both the networks are adversaries which are in a constant battle to improve themselves and minimize their losses. The thesis uses DCGAN for generating Brain CT slices as a method for data generation.

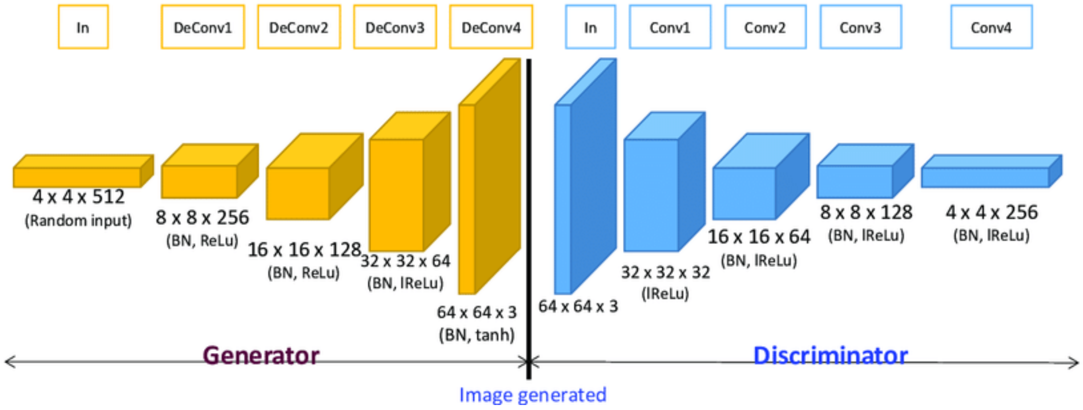


Figure 14: Architectural Overview of DCGAN; *Source: (Suh et al.)*

3.6 Transfer Learning

Transfer Learning is a ML knowledge transfer process wherein knowledge and features from models trained on a large dataset (such as ImageNet) can be used to solve a new problem. Deep Learning models such as CNN use the initial model layers as feature extractors. The initial layers learn top level and abstract features such as edges, colors, shapes, etc. The subsequent deeper layers learn the lower level and minute details of the dataset. Figure 15 encapsulates the process of Transfer Learning.

ImageNet (Deng et al. 248), is a large hierarchical database containing over 14 million images belonging to over 20,000 classes. State-of-the-art CNNs have been trained on ImageNet whose knowledge can be used to solve other Computer Vision problems. The trained CNNs are available as pre-trained models for public use. Some pre-trained model used as feature extractors in this thesis are ResNet50 (He et al. 770), DenseNet121 (Huang et al. 2261), and InceptionV3 (Szegedy et al. 2818). The architectures of these models are discussed in the next chapter.

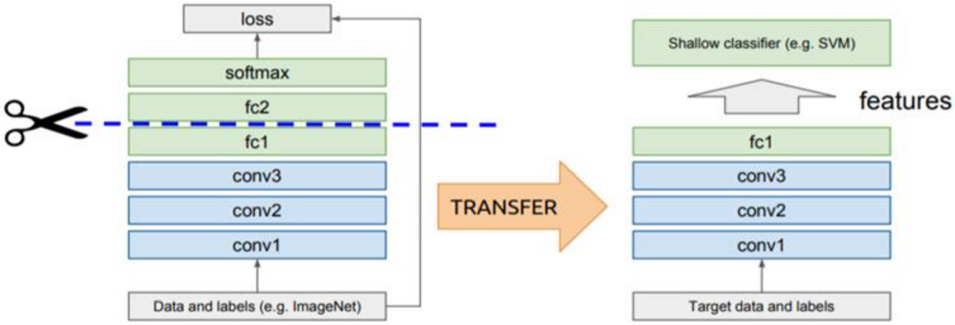


Figure 15: Transfer Learning; *Source: (Sarkar)*

Transfer Learning solves the problem of unavailability of a large-scale annotated dataset usually required to train models from scratch. It is extensively used in domains such as Natural Language Processing for creating word embeddings and representations, in Computer vision for tasks such as Object Detection, Localization and Segmentation, and Speech and Audio processing for segmenting samples from a song, etc. (Devlin et al.) and (Cer et al.) are some examples employing use of transfer learning to train state-of-the-art word and sentence encoding models. (Tan et al. 270) categorized deep transfer learning into four categories, namely, Instance Based learning, Mapping Based learning, Network based learning and Adversarial based learning.

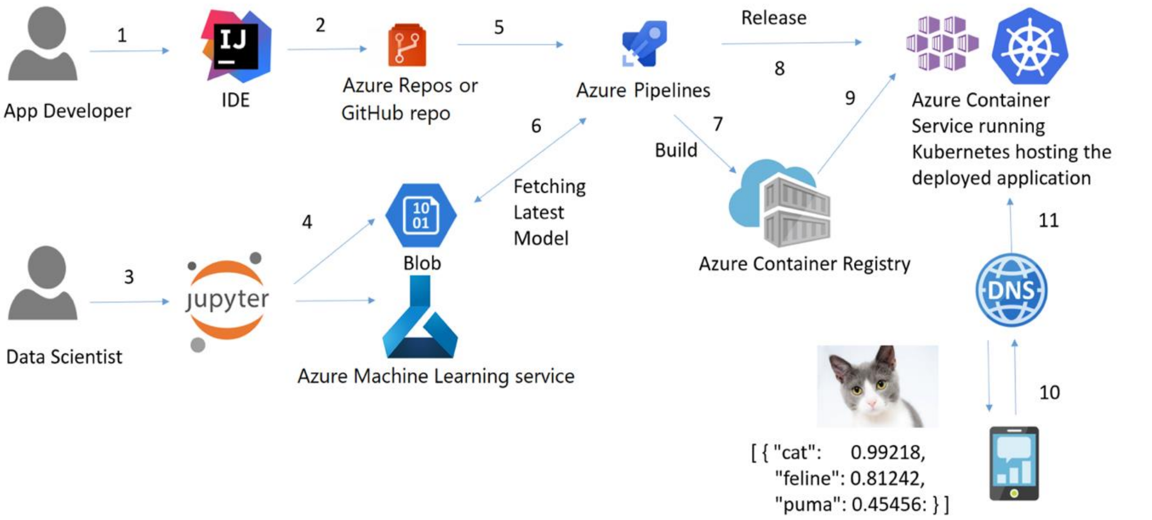
Even though Transfer Learning is extensively being experimented with to solve different problems, the results in the field of Medical Image Analysis have not been as good as other fields. This is mainly because of lack of an ImageNet like database for pre-training models on Medical Images. The pre-trained models on ImageNet do not work very well on medical images since the Medical images (CT images, MR images, PET images, X-rays) have a very different structure from natural images. (Chen et al.) proposed MedicalNet, a set of pre-trained 3D models trained on Medical Images from multiple sources. (Zhou et al. 384) proposed Model Genesis, a set of autodidactic models for medical image analysis, trained in a self-supervised fashion.

3.7 Machine Learning Deployment

Once a Machine Learning model is trained, the next step in the ML pipeline is to deploy the model in a scalable fashion. Businesses employ Machine Learning engineers and Data Scientists to train accurate models and then ship the model to Software Engineers for hosting them as production level software. Since the models are often retrained using batch training (training model with available data and then deploying it) or real-time training (continuous retraining of models with real time data), a well-established, quick prototyping and fast turnaround Continuous Integration and Continuous Delivery pipeline (CI/CD) has to be in place. Moreover, since the number of use cases for ML are increasing rapidly, and hence the model users, there is a need for these models to be highly available and scalable so that they can support the high number of requests. Figure 16 illustrates a CI/CD pipeline built using Microsoft Azure Services.

Python is the most widely used language for prototyping Machine Learning models. The availability of multi-purpose libraries along with a wide-open source community makes it really easy to build and test ML models (Matthews "6 Reasons Why Python Is Suddenly Super Popular"). The ML models are commonly stored as Pickle or Json files which can be loaded using the common ML libraries such as Keras (“Keras: The Python

Deep Learning Library.”), Scikit-learn (“Scikit-learn.”), etc. One of the most common way of serving a model as part of a production level software is exposing it as a micro-service, implemented as a RESTful API (Application Programming Interface). The microservice architecture is a great alternative to monolith systems, breaking down the structure into groups of services which are loosely coupled, highly scalable, independent and owned by a single team, and have a specific utility. This type of architectural pattern gives organizations the flexibility to add new functionalities as services in a rapid fashion (“What Are Microservices?”). REST (Representational Style Transfer) is an architectural style to design and build these services. These services can be built using different programming languages and backend servers such as Python (Flask or Django backend), Java, etc.


App Developer
1
IDE
2
Azure Repos or GitHub repo
5
Azure Pipelines
Release
8
Azure Container Service running Kubernetes hosting the deployed application
9
7
Build
6
Fetching Latest Model
Azure Container Registry
11
DNS
10
Data Scientist
3
jupyter
4
Blob
Azure Machine Learning service

```
[ { "cat":     0.99218,
    "feline": 0.81242,
    "puma": 0.45456: } ]
```

Figure 16: A CI/CD Pipeline for Deploying Machine Learning Models Using Microsoft Azure Components; *Source: (Marktab)*

In order for the models (services embedded in an application) to be used by the external world, the services are deployed on a web server exposed by a domain name. The deployment process includes encapsulating the application with its dependencies as a container. These containers are essentially a snapshot of the application along with its dependencies. The containers are deployed to a web server such as Azure Container Registry (by Microsoft) or Elastic Container Registry (by Amazon) which can host these containers. Kubernetes, a container orchestration mechanism, is used to scale these containers based on the application traffic. Each cloud provider has their own version of Kubernetes for deploying the containers at scale. As evident from the process, serving the ML models efficiently, involves building them using best Software Engineering and Design practices.

3.8 Containerization Using Docker

The motivation behind containerization comes from being able to run an application on any Linux based server without worrying about the dependencies of the application. Docker bundles the application along with required libraries, configurations and other dependencies into a container which can be run on on-premise servers or public cloud. It essentially employs virtualization for running multiple independent containers by sharing the OS resource between each container (Connor “What Is Docker Container”). Figure 17 explains how Virtual Machines (VMs) are different from Docker containers.

Docker consists of a Docker Engine, responsible for building docker images, and Docker Hub, for sharing the images on different Container registries. The first step to using containerization with Docker is to write a Dockerfile listing the dependencies which need to be bundled together and how to run the application once the dependencies are copied. Once the Dockerfile is written, Docker images can be built using the command line interface of Docker Engine. A Docker Image is a snapshot of the container which can be spun up on a Linux server or Windows server (using virtualization).

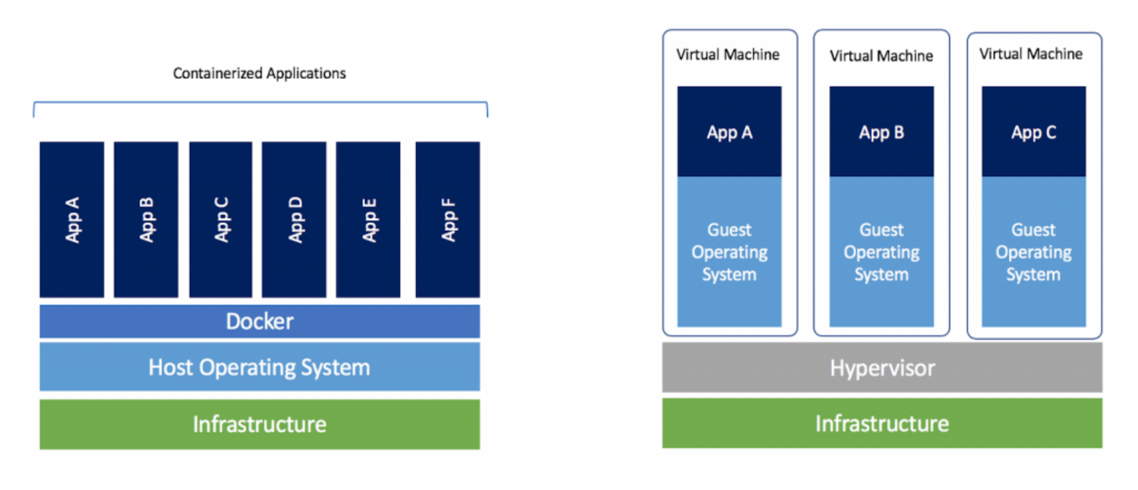


Figure 17: Docker Containers Versus Virtual Machines; *Source: (Mouat et al.)*

Containers make any application highly portable. Different teams can work on different parts of the same project and each part can live in a container. This significantly decreases the deployment time and helps in iterating over changes more quickly.

3.9 Container Orchestration Using Kubernetes

Kubernetes (K8s) is an open-source container orchestration manager used for managing and deploying containers at scale (“Production-Grade Container Orchestration.”). It was initially announced by Google in 2014 as an open source project to decouple application containers from the systems they run on (Mcluckie “Containers, VMs, Kubernetes and VMware”). Since then, Kubernetes has been used by all the companies which are looking for deploying and managing containers in scale (Bernstein 81). The major cloud providers, such as Amazon, Microsoft and Google, have rolled out a "managed" version of their Kubernetes service which essentially manages the cluster load automatically. The thesis uses Amazon's Elastic Container Registry for deploying container to the Kubernetes cluster. Figure 18 provides a high-level picture of the different components of Kubernetes architecture.

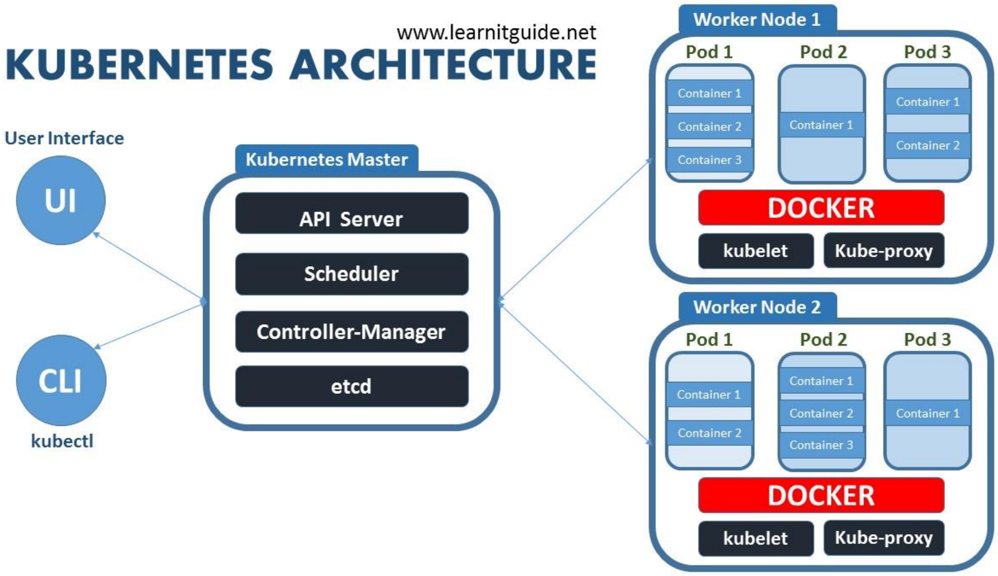


Figure 18: An Architectural Overview of Kubernetes (K8s); *Source: ("What Is Kubernetes - Learn Kubernetes from Basics.")*

A *Node* is a worker machine in Kubernetes using hardware (VMs or physical system) for computation. The physical systems can be a server in server farms and a virtual machine can be systems on a cloud platform. A *Pod* is a wrapper around one or more containers which are deployed to a single node. The containers in a pod share the same CPU resources among them and can communicate with each other easily. The pods can be replicated to support scalability and availability. Multiple pods can exist in a node. Nodes pool their resource together to form *Clusters.* Nodes can handle load between them within a cluster (to ensure high availability even if a node is killed).

The *worker nodes (*where pods are running) are controlled by the *master node* using kubelet. *kubelet* ensures that the containers are running inside pods in a node and sends their health back to the master node. *kube-proxy* is a network proxy running on each node used to perform network routing. It exposes the worker nodes to the internet. A user can interact with the master node via a Web UI or *kubectl* which is the Kubernetes command line interface.

Pods can be created inside a Kubernetes cluster using *Deployment* which is a template for deploying containers and their replicas. The containers inside the pods are made available to the internet using *Service* which specifies the visibility of the pod. *Ingress Controllers* or *Load Balancers* can be used to communicate with a Service inside a pod (Sanche “Kubernetes 101”).

Some other features of Kubernetes include Automated Pod scheduling, Horizontal scaling (increasing the number of pods) and Load Balancing, and Auto scalable infrastructure. The availability of automatic scaling is the main reason why Kubernetes has been used in the thesis for deploying the trained models as APIs.

CHAPTER 4

IMPLEMENTATION

This chapter explains the implementation details of the research conducted as part of the thesis. The chapter starts with describing the Domain Model designed for the problem statement. Each subsequent section explains the implementation methodology of a sub-domain. The Data Collection strategy at Phoenix Children Hospital is described and the data preprocessing techniques are explained. The experiment setup for training Generative Adversarial Networks for Image Generation and 2D and 3D Convolutional Neural Networks for Image Classification are later illustrated. Finally, the chapter explains the process used to deploy the trained models as APIs using Flask, Docker and Kubernetes.

4.1 The Domain Model

One of the contributions of the thesis is modeling the domain of Abusive Head Trauma using AI techniques. Solving any novel problem requires background knowledge of the problem and its different components. Once this knowledge is acquired, the problem can be broken down into smaller sub-problems which are easier to solve. One can compare this to the famous "Divide and Conquer" strategy used to solve multiple puzzles in the field of computing. This research explains the domain of Abusive Head Trauma (and subsequently Traumatic Brain Injury), gives background knowledge on differentiating AHT from Non-AHT cases, and breaks the problem into sub-domains of Data Collection, Data Pre-processing, Image Synthesis, Image Classification, Building APIs, Containerization using Docker, and its orchestration using Kubernetes. Figure 19 demonstrates the intersection of domains of Medical Knowledge, Deep Learning and Software Engineering used in modeling the domain.

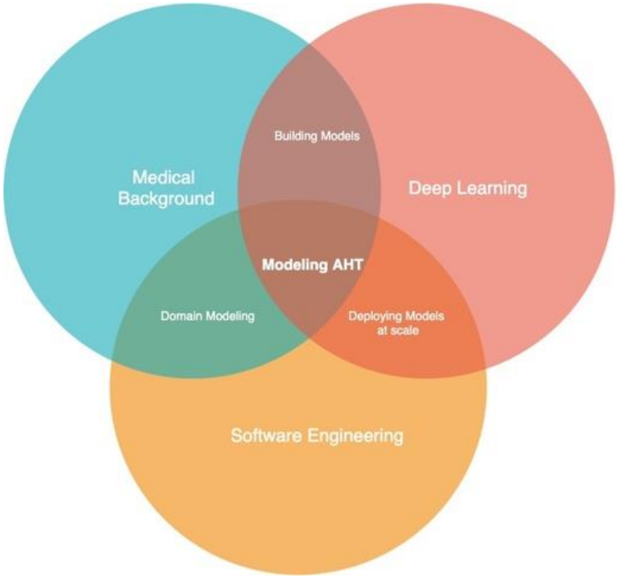


Figure 19: Domain Intersection for Modeling AHT

A domain map is generated using the knowledge derived from the Domain Intersection (as shown in Figure 19) and the Domain Modeling hypothesis for Classification task (as shown in Figure 1). Figure 20 illustrates the domain map of Abusive Head Trauma. AHT domain has been divided into six sub-domains encompassing Medical Knowledge, Deep Learning and Software Engineering. Each subdomain is divided into sub-areas representing significant entities, problems and experiments to solve the problem. The first domain, Medical Knowledge, has been described in detail in Chapter 2. Each subsequent section describes the sub-domains and their implementations.

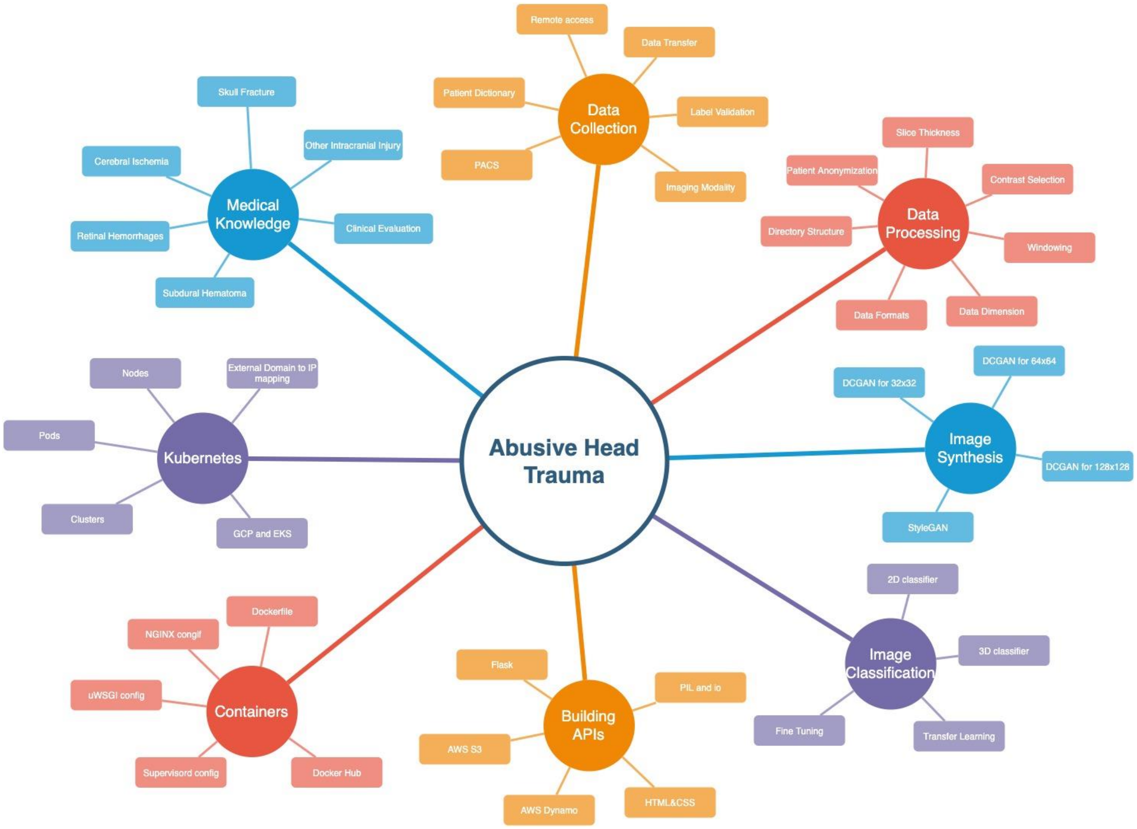


Figure 20: Domain Map of AHT

4.2 Data Collection

This research was completed in accordance with all federal, state and hospital policy under the Health Insurance Portability and Accountability Act (HIPAA). Data used in this research was collected under Phoenix Children's hospital Institutional Review Board (IRB) 09-055 and completed under Phoenix Children's Hospital Institutional Review Board (IRB) 17-015. The data was extracted from the Forensic Registry and includes all pediatric patients <17 years of age who presented in the PCH Emergency Department or transferred to PCH from another medical setting between 2010-2016 with any head injury suspicious of physical abuse. In total, 804 patient injuries met the study criteria for inclusion. The primary goal of the study is to build a tool which use AI techniques on Computer

Tomography (CT) images to classify whether the scans belong to patient with Abusive Head Trauma or Accidental Head Trauma. The tool would be used for screening all pediatric patients in Emergency Department who present with any head trauma or suspicion of head trauma based on symptoms (seizures, lethargy, loss of consciousness, etc.).

AT PCH, the child protection team (CPT), is made up of clinicians, social workers and forensic specialists; and members of the state child protection agency and members of law enforcement meet weekly to perform a comprehensive forensic evaluation of patients referred to the CPT based on suspicion of physical abuse. The team after reviewing all of the available medical information coupled with other agency information, makes a determination on whether the injury is presumed or determined "Probable Abuse". If the medical information is not sufficient to rule out an accidental trauma, the patient's forensic determination is "Undetermined". If the medical information is sufficient to rule out inflicted injury, the injury as determined "Probable Not Abuse". Since this study only included pediatric patients with head trauma and Potential AHT (regardless of any other injuries the patient might have been inflicted, the study population was labelled as follows: presumptive or determined AHT, Non-AHT and Undetermined. The patient data was stored in the hospital PACS (Picture Archiving and Communication system) and was not deidentified. In order to use the PACS images, the PACS data was de-identified and labelled using a subject ID key match. Data extracted from the forensic registry and maintained for purposed of analysis under PCH 17-015 was used to label PACS image data as AHT, Non-AHT and Undetermined. The data for each patient was downloaded manually (in batches of 15 – 30) from PACS into a PCH computer. Due to security issues, the data for each patient had to be downloaded manually using the subject ID after deidentification. For validation of labelling, each subject ID was matched to an MRN maintained in a screening log and then to a data table containing the AHT labels to confirm the label. Since there was no way to download only the CT studies of a patient, the downloaded information contained all the radiological (including CT, X-ray, radiological reports, etc.). The folder structure was complex in nature. Figure 21 illustrates the folder structure.

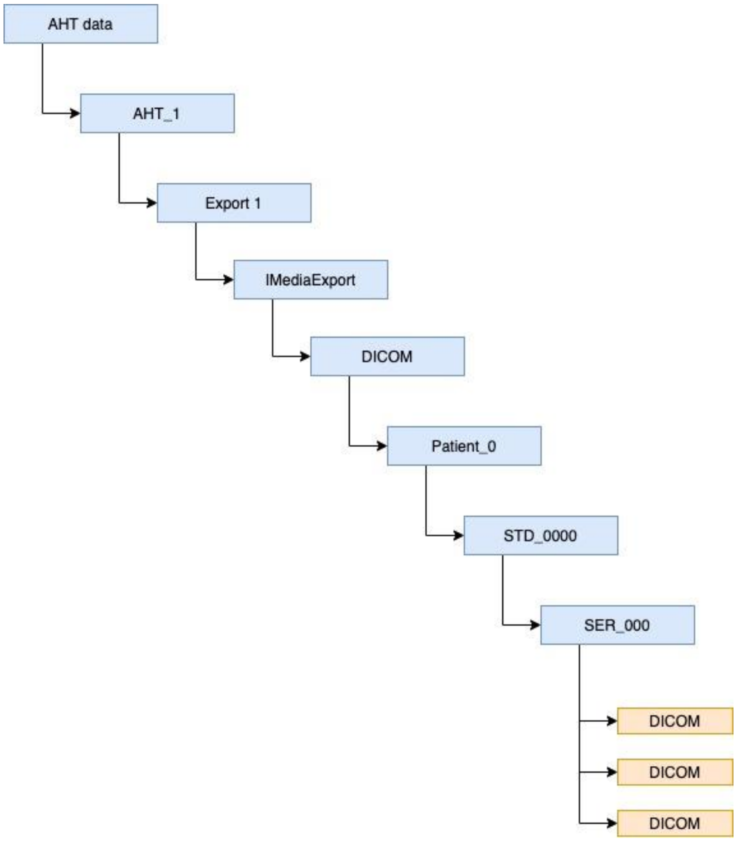


Figure 21: Folder Structure of Deidentified Data Downloaded from PACS

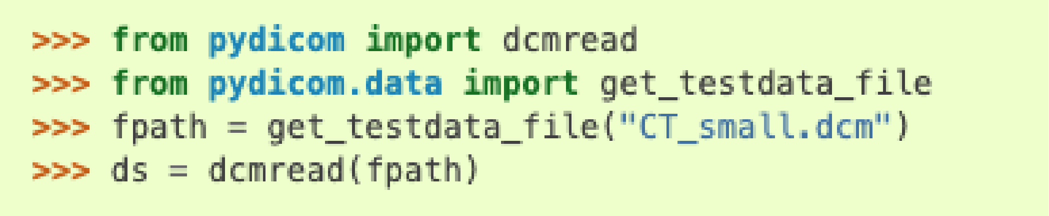
The base folder had five sub-folders called AHT_*folder_num* which represented patient MRNs from five different data dictionaries. Each AHT folder had multiple batches of data in sub-folders called Export_*number*. Each export had two additional folder before containing the patient studies called IMediaExport and DICOM. The DICOM folder contained the patient studies (15 – 30) in each DICOM folder. For each patient, the folder STD_*number* represented the study number (one or more for each patient). For each study, the subfolders SER_*number* represented series of the study. The series indicated the type of imaging for the patient (such as skeletal x-ray, CT with Slice thickness 3mm, CT with bone window, etc.). Each series contained the imaging in DICOM format. DICOM (Digital Imaging and Communications in Medicine) is an internationally accepted medical imaging standard which contain imaging results, study information and patient information in form of metadata. All the DICOM images in the series were deidentified before downloading from PACS. So, the next step was CT image selection.

4.3 Data Pre-processing

Data Anonymization was a challenging part due to the volume of the data and presence of all imaging modalities in the Patient folders. The total size of the data was ~252 Gigabytes. So, data-preprocessing had to be done in chunks. A virtual Linux server was built with high memory (16 Gigabytes) for faster reading of data.

4.3.1 Data Anonymization

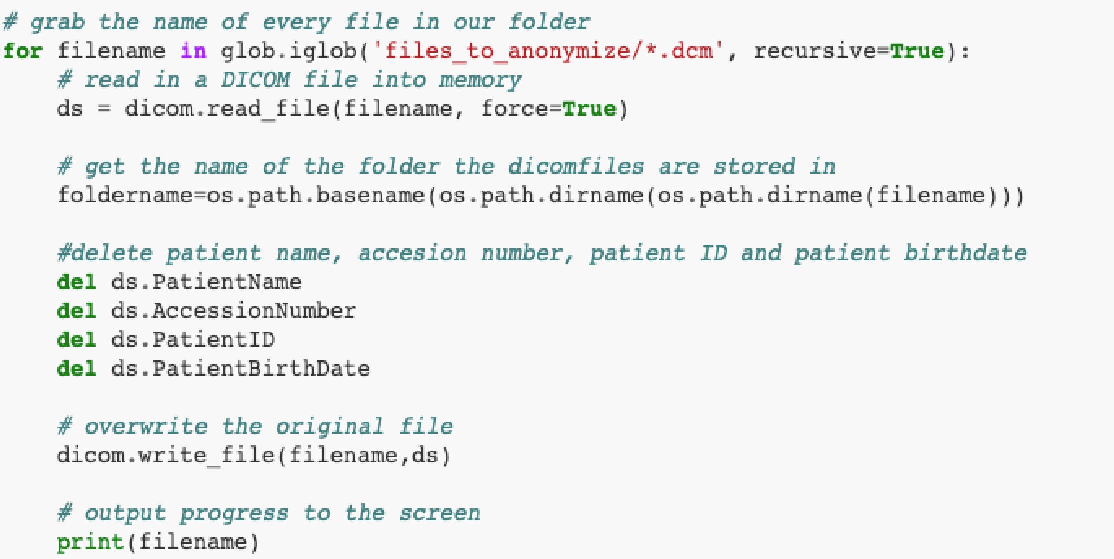
An anonymization script was written in Python which read all the DICOM one by one and extracted the DICOM fields. A DICOM file can be read in Python using the PyDicom library (“Pydicom Documentation.”). Figure 22 demonstrates the code snapshot for reading DICOM files.



```
>>> from pydicom import dcmread
>>> from pydicom.data import get_testdata_file
>>> fpath = get_testdata_file("CT_small.dcm")
>>> ds = dcmread(fpath)
```

Figure 22: Reading DICOM File Using PyDicom; *Source: (“Pydicom Documentation.”)*

Once all the DICOM files in a series were read for a patient, the patient name, patient date of birth, patient id, study date and time, and accession number were replaced by dummy values. An excel map was updated with the original patient information and the newly created dummy patient information for future reference. Figure 23 demonstrates the code snapshot for reading DICOM and deleting patient information.



```
# grab the name of every file in our folder
for filename in glob.iglob('files_to_anonymize/*.dcm', recursive=True):
    # read in a DICOM file into memory
    ds = dicom.read_file(filename, force=True)

    # get the name of the folder the dicomfiles are stored in
    foldername=os.path.basename(os.path.dirname(os.path.dirname(filename)))

    #delete patient name, accesion number, patient ID and patient birthdate
    del ds.PatientName
    del ds.AccessionNumber
    del ds.PatientID
    del ds.PatientBirthDate

    # overwrite the original file
    dicom.write_file(filename,ds)

    # output progress to the screen
    print(filename)
```

Figure 23: Deleting Patient Information from DICOM Files; *Source: (“A Multi-Platform DICOM Toolbox for Academic Radiologists.”)*

4.3.2 Brain CT Anatomical Plane and Slice Thickness Selection

The next step was to select only the image modality which belonged to a Brain CT series from the multiple series of scans present for each patient. For Brain CT modality, the series of scans with slice thickness of 3 mm were selected. Only the axial planes were persevered for the series. This was done to ensure a standardized dataset. Series with Bone windows was also dropped from the dataset. A data-preprocessing script was written to achieve all of the above tasks. The DICOM files per series per patient were read one by one. The Slice thickness metadata tag was extracted and if the value of Slice thickness was 3.00, the series were preserved. The slices also had a comment metadata field which described the type of windowing (bone or brain) and imaging modality (X-ray or CT) for the series. If the comment contained words, “Brain” and “CT”, the slices were preserved. These slices were then stored to a different location in the anonymized patient id folder.

After this process, the data contained 171 patients with AHT label and 207 patients with Non-AHT label. The total number of slices for each patient ranged from 26 to 45. The slice thickness for each slice was 3 mm and each slice contained only the brain window. The size of the pixel array for each size was 512x512.The data was then transferred to a laptop with a GPU (NVIDIA RTX 2060) for building and training the deep learning models.

4.3.3 Viewing the Slices in the Right Window

Windowing is a technique used by radiologists to highlight certain voxels of a brain CT for correct diagnosis. It is sometimes also referred to as Contrast enhancement. It is achieved by changing the Window Width (WW) and Window Level/Center (WL). WW is the measure of range of CT number an image can store. WL is the midpoint of the range of CT numbers stored (Murphy “Windowing (CT)”). Windowing is useful to detect certain abnormalities such as subdural hematoma (which is more visible in subdural window). The thesis uses three window settings, namely brain window (WW:80 WL:40), subdural window (WW:130-300 WL:50-100) and bone window (WW:2800 WL:600) (Murphy). Each slice is converted into each of these windows and then the windows are fused together as three channels of the image (Brain-Subdural-Bone channel) (Reppic “Gradient & Sigmoid Windowing.”). Figure 24 shows a slice in all the three different windows.

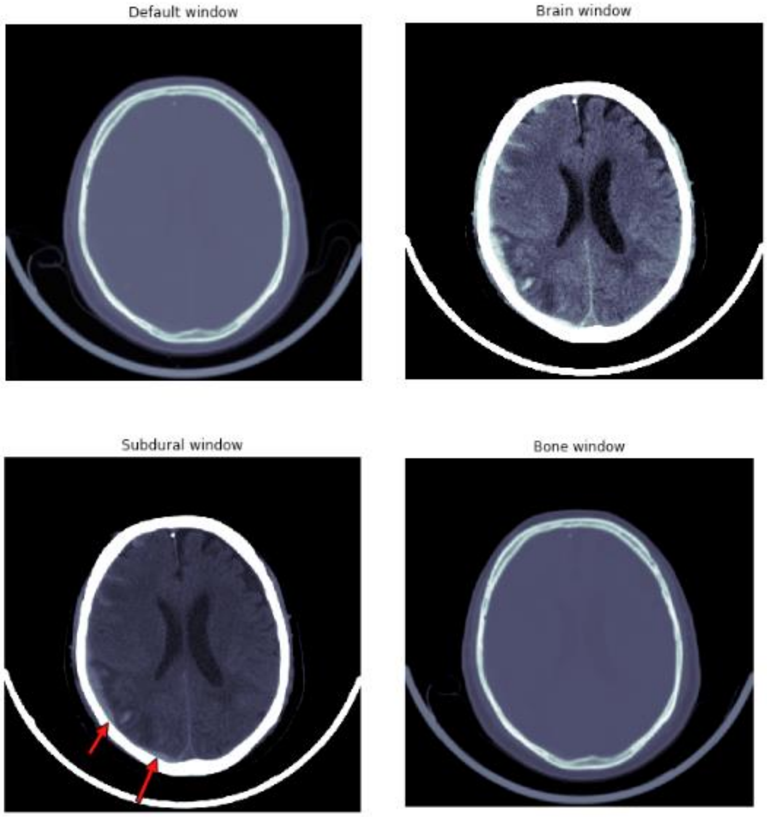
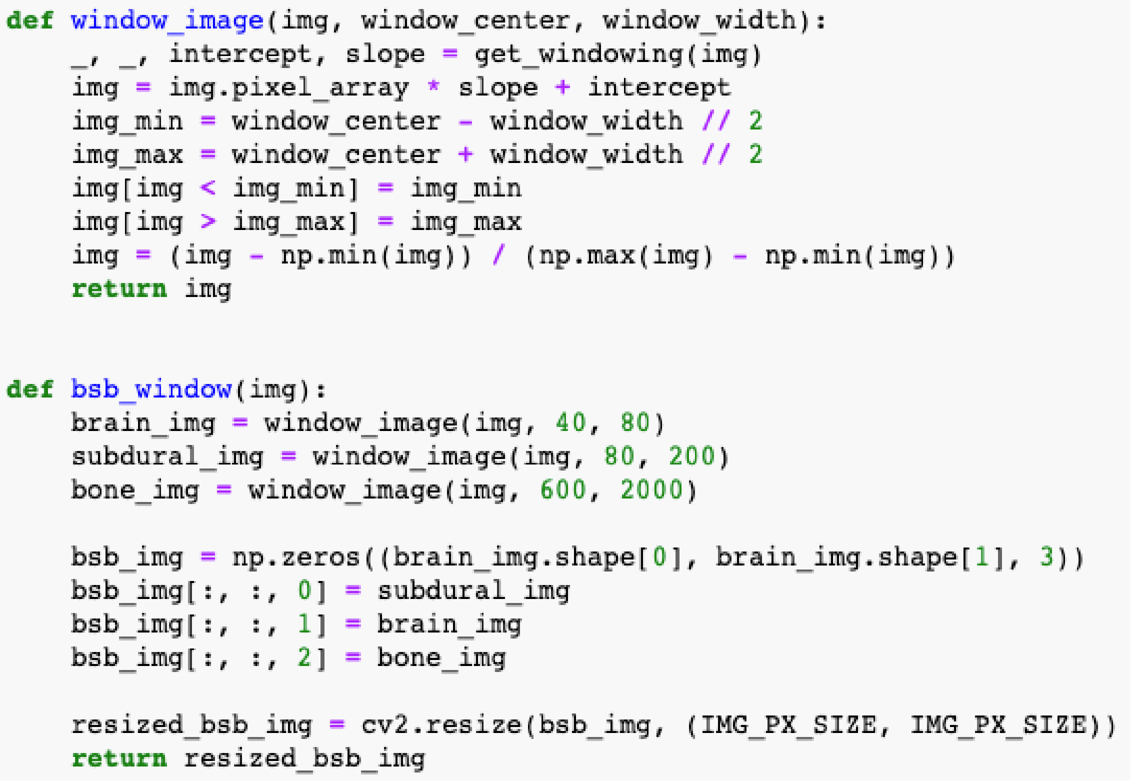


Figure 24: A DICOM Slice in Brain, Subdural and Bone Window; *Source: (Reppic “Gradient & Sigmoid Windowing.”)*

Some pathologies are more prominent in some window settings. Figure 24 shows Subdural Hematoma (indicated by red arrows in the bottom-left slice) being clearly visible in the Subdural Window setting. Since the most prominent CT markers of AHT are Subdural Hemorrhage, Cerebral Edema and Skull fracture, each slice is converted into the brain, subdural and bone window and fused together as three channels (just like RGB channel for a color image). Figure 25 demonstrates the code for the same.



```python
def window_image(img, window_center, window_width):
    _, _, intercept, slope = get_windowing(img)
    img = img.pixel_array * slope + intercept
    img_min = window_center - window_width // 2
    img_max = window_center + window_width // 2
    img[img < img_min] = img_min
    img[img > img_max] = img_max
    img = (img - np.min(img)) / (np.max(img) - np.min(img))
    return img


def bsb_window(img):
    brain_img = window_image(img, 40, 80)
    subdural_img = window_image(img, 80, 200)
    bone_img = window_image(img, 600, 2000)

    bsb_img = np.zeros((brain_img.shape[0], brain_img.shape[1], 3))
    bsb_img[:, :, 0] = subdural_img
    bsb_img[:, :, 1] = brain_img
    bsb_img[:, :, 2] = bone_img

    resized_bsb_img = cv2.resize(bsb_img, (IMG_PX_SIZE, IMG_PX_SIZE))
    return resized_bsb_img
```

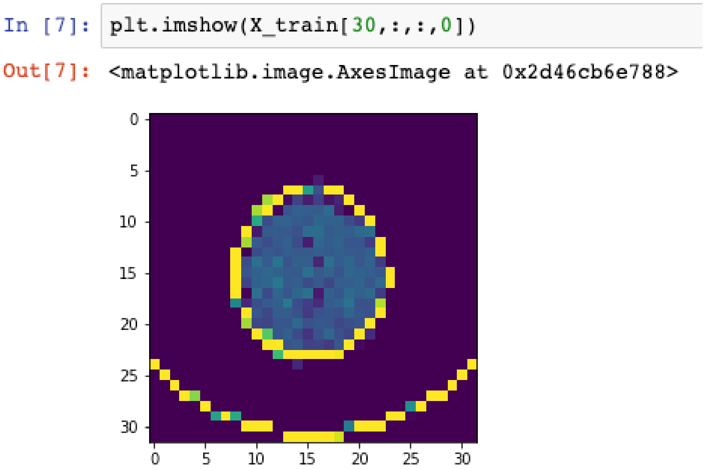
Figure 25: Code Snippet to Convert the Slices into the Three Channels

4.4 Image Generation Using GANs

A major problem in building Deep Learning models for Medical images is unavailability of large amount of labeled data. GANs have been effective in generating new images which are from the same distribution as the training images. DCGAN is used to generate new image slices from the three channel (brain-subdural-bone) input images. Two different image dimensions are generated using DCGAN, 32x32 and 128x128. Higher dimension images could not be generated due to GPU memory limitations.

4.4.1 DCGAN for 32x32 images

The DCGAN to generate 32x32 image takes a 32x32 image as training input. The code is implemented in Keras. Figure 26 shows an example input file of shape 32x32, plotted using matplotlib.



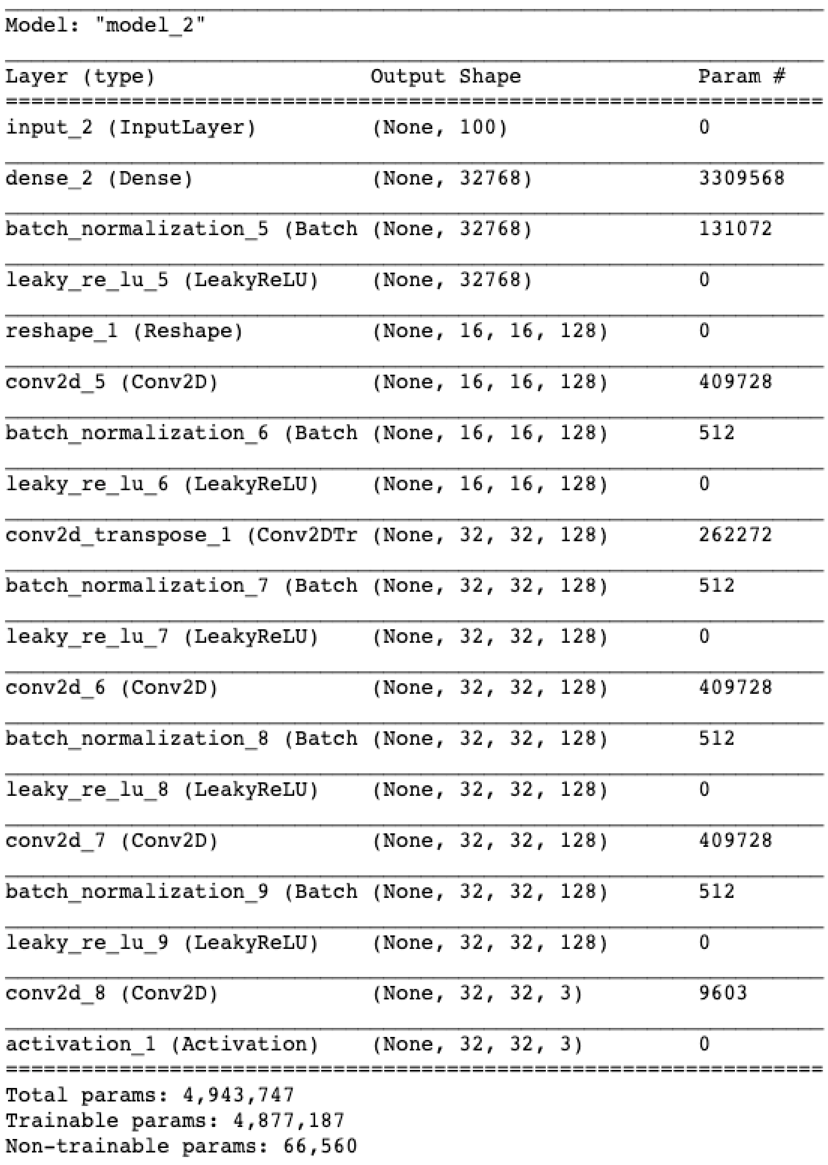
```
In [7]: plt.imshow(X_train[30,:,:,0])

Out[7]: <matplotlib.image.AxesImage at 0x2d46cb6e788>
```

Figure 26: 32x32 Input Image for DCGAN

The GAN was compiled on Adam Optimizer with a learning rate of 0.001 and beta1 of 0.5 for both the generator and discriminator. The loss function was binary crossentropy and it was trained for 80 epochs on batch size of 16. The results are discussed in the next chapter.

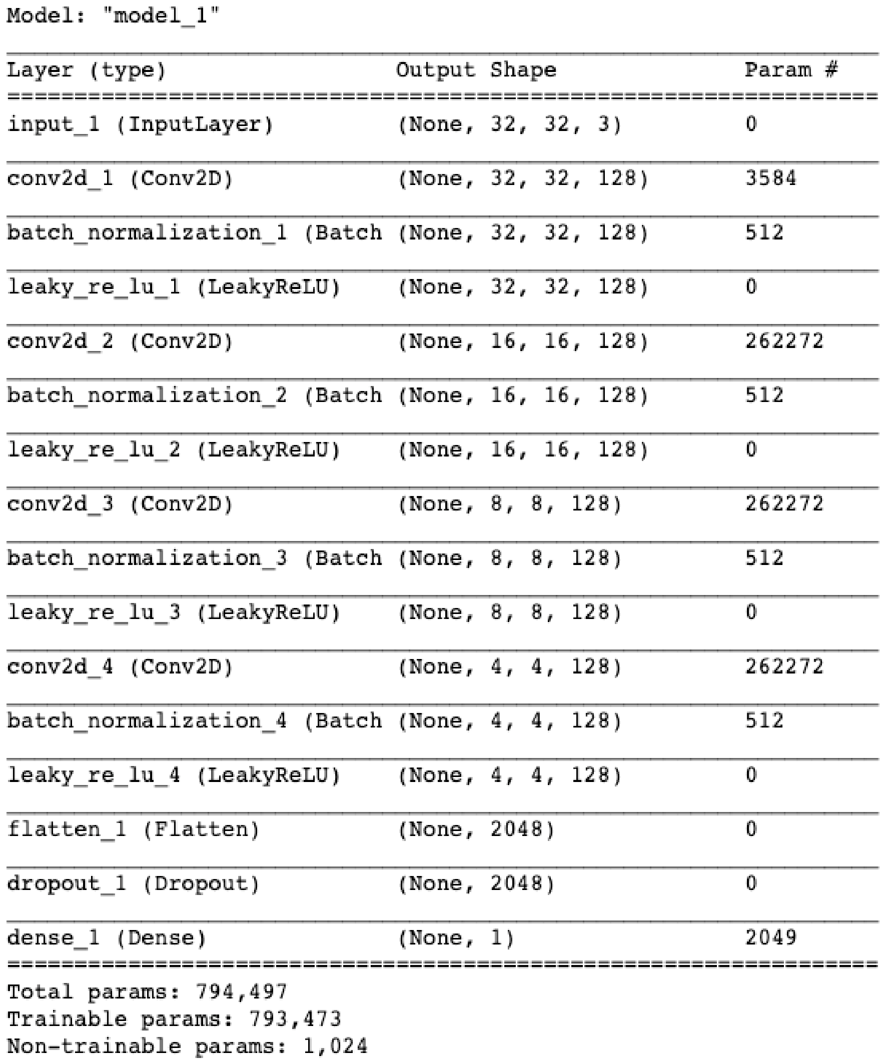
The generator for DCGAN consists of dense layer, followed by Convolution and Deconvolution layers with LeakyReLU activations. Dropout and Batch Normalization are added to increase robustness of the model. Figure 27 explains the configuration of the generator where the columns represents the layer, the output shape and the parameters.



```
_________________________________________________________________
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         (None, 100)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 32768)             3309568   
_________________________________________________________________
batch_normalization_5 (Batch (None, 32768)             131072    
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 32768)             0         
_________________________________________________________________
reshape_1 (Reshape)          (None, 16, 16, 128)       0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       409728    
_________________________________________________________________
batch_normalization_6 (Batch (None, 16, 16, 128)       512       
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 16, 16, 128)       0         
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 128)       262272    
_________________________________________________________________
batch_normalization_7 (Batch (None, 32, 32, 128)       512       
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 32, 32, 128)       0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 128)       409728    
_________________________________________________________________
batch_normalization_8 (Batch (None, 32, 32, 128)       512       
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 32, 32, 128)       0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 128)       409728    
_________________________________________________________________
batch_normalization_9 (Batch (None, 32, 32, 128)       512       
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 32, 32, 128)       0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 3)         9603      
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 3)         0         
=================================================================
Total params: 4,943,747
Trainable params: 4,877,187
Non-trainable params: 66,560
```

Figure 27: Generator Configuration for 32x32 DCGAN

The discriminator consists of multiple convolution layers (with dropout, batch normalization and LeakyReLU activation) and a dense layer (with sigmoid activation) as output layer to generate a probability. The numpy array is the 32x32 image with 3 channels. Figure 28 explains the configuration of the discriminator where the columns represents the layer, the output shape and the number of parameters.



```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 128)       3584      
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 128)       512       
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 128)       0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       262272    
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 128)       512       
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 128)       0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 128)         262272    
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 128)         512       
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 128)         0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 128)         262272    
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 128)         512       
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2049      
=================================================================
Total params: 794,497
Trainable params: 793,473
Non-trainable params: 1,024
```

Figure 28: Discriminator Configuration for 32x32 DCGAN

4.4.2 DCGAN for 128x128 Images

The DCGAN to generate 128x128 images was implemented in PyTorch. The GAN was trained on 128x128 images with three channels (brain-subdural-bone). Figure 29 shows a subset of training images of size 128x128.

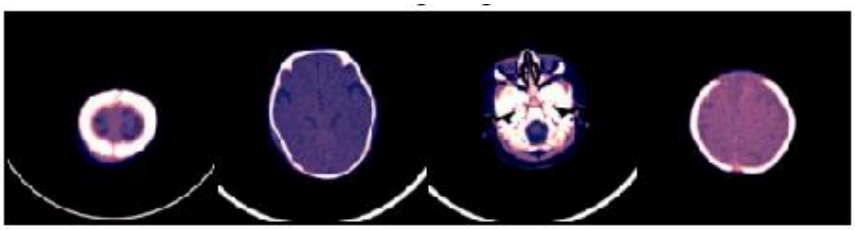
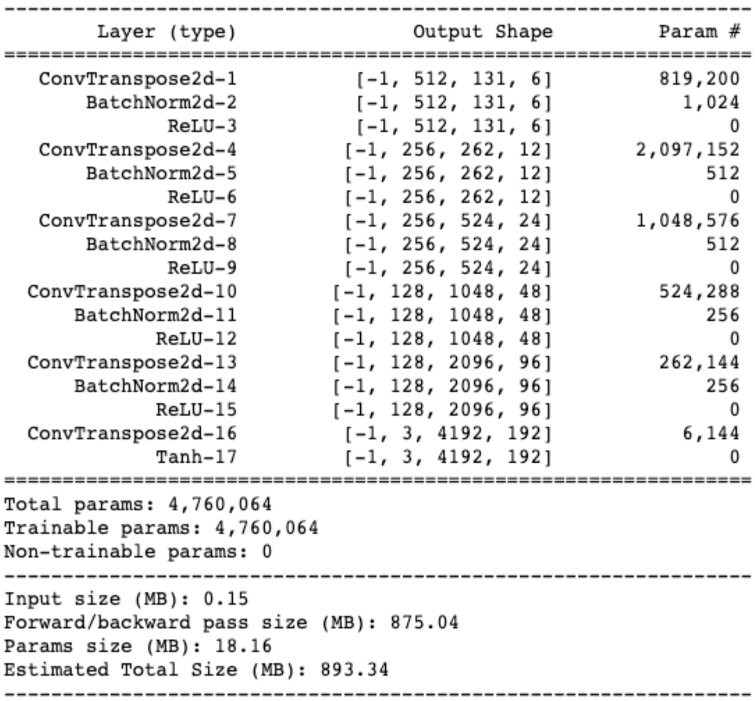


Figure 29: Training Images for 128x128 DCGAN

The GAN was trained for 200 epochs using Adam optimizer with learning rate of 0.0001 and beta1 of 0.9. The loss function was binary crossentropy and the batch size was 8. The results are discussed in the next chapter.

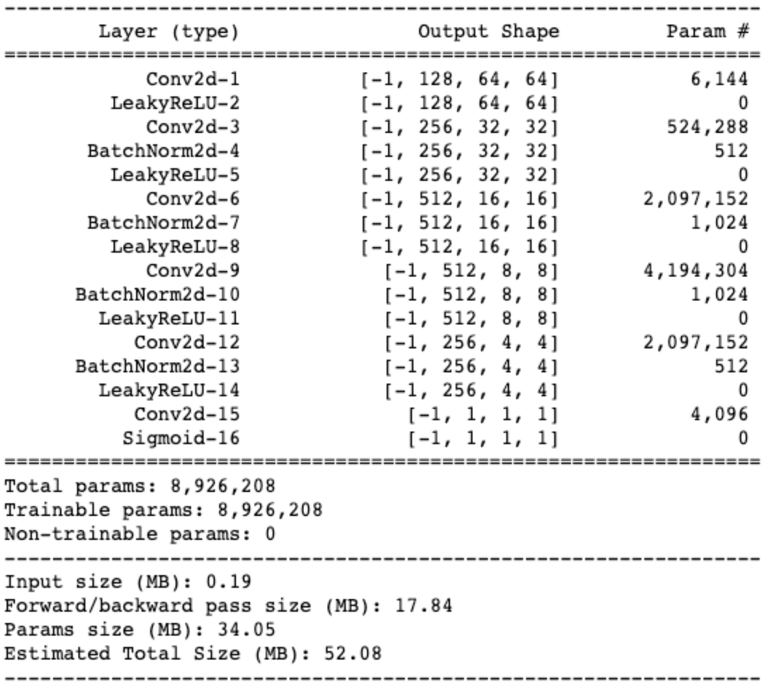
The generator consists of Deconvolution layers with Batch Normalization and ReLU activation. The last layer consists of a Tanh activation. It takes a torch tensor of size (3, 128, 128) as input (128x128 images with three channels: brain-subdural-bone). The Generator layer wise configuration is demonstrated by Figure 30.

The discriminator consists of convolution layers with batch normalization and LeakyReLU activation. The last layer has a sigmoid activation function to generate the probability of image being real. It takes a tensor of shape (100, 128, 128). The Discriminator layer wise configuration is demonstrated by Figure 31.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1         [-1, 512, 131, 6]         819,200
       BatchNorm2d-2         [-1, 512, 131, 6]           1,024
              ReLU-3         [-1, 512, 131, 6]               0
   ConvTranspose2d-4        [-1, 256, 262, 12]       2,097,152
       BatchNorm2d-5        [-1, 256, 262, 12]             512
              ReLU-6        [-1, 256, 262, 12]               0
   ConvTranspose2d-7        [-1, 256, 524, 24]       1,048,576
       BatchNorm2d-8        [-1, 256, 524, 24]             512
              ReLU-9        [-1, 256, 524, 24]               0
  ConvTranspose2d-10       [-1, 128, 1048, 48]         524,288
      BatchNorm2d-11       [-1, 128, 1048, 48]             256
             ReLU-12       [-1, 128, 1048, 48]               0
  ConvTranspose2d-13       [-1, 128, 2096, 96]         262,144
      BatchNorm2d-14       [-1, 128, 2096, 96]             256
             ReLU-15       [-1, 128, 2096, 96]               0
  ConvTranspose2d-16        [-1, 3, 4192, 192]           6,144
             Tanh-17        [-1, 3, 4192, 192]               0
================================================================
Total params: 4,760,064
Trainable params: 4,760,064
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.15
Forward/backward pass size (MB): 875.04
Params size (MB): 18.16
Estimated Total Size (MB): 893.34
----------------------------------------------------------------
```

Figure 30: Generator Configuration for 128x128 GAN



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 128, 64, 64]           6,144
         LeakyReLU-2          [-1, 128, 64, 64]               0
            Conv2d-3          [-1, 256, 32, 32]         524,288
       BatchNorm2d-4          [-1, 256, 32, 32]             512
         LeakyReLU-5          [-1, 256, 32, 32]               0
            Conv2d-6          [-1, 512, 16, 16]       2,097,152
       BatchNorm2d-7          [-1, 512, 16, 16]           1,024
         LeakyReLU-8          [-1, 512, 16, 16]               0
            Conv2d-9            [-1, 512, 8, 8]       4,194,304
      BatchNorm2d-10            [-1, 512, 8, 8]           1,024
        LeakyReLU-11            [-1, 512, 8, 8]               0
           Conv2d-12            [-1, 256, 4, 4]       2,097,152
      BatchNorm2d-13            [-1, 256, 4, 4]             512
        LeakyReLU-14            [-1, 256, 4, 4]               0
           Conv2d-15              [-1, 1, 1, 1]           4,096
          Sigmoid-16              [-1, 1, 1, 1]               0
================================================================
Total params: 8,926,208
Trainable params: 8,926,208
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 17.84
Params size (MB): 34.05
Estimated Total Size (MB): 52.08
----------------------------------------------------------------
```

Figure 31: Discriminator Configuration for 128x128 GAN

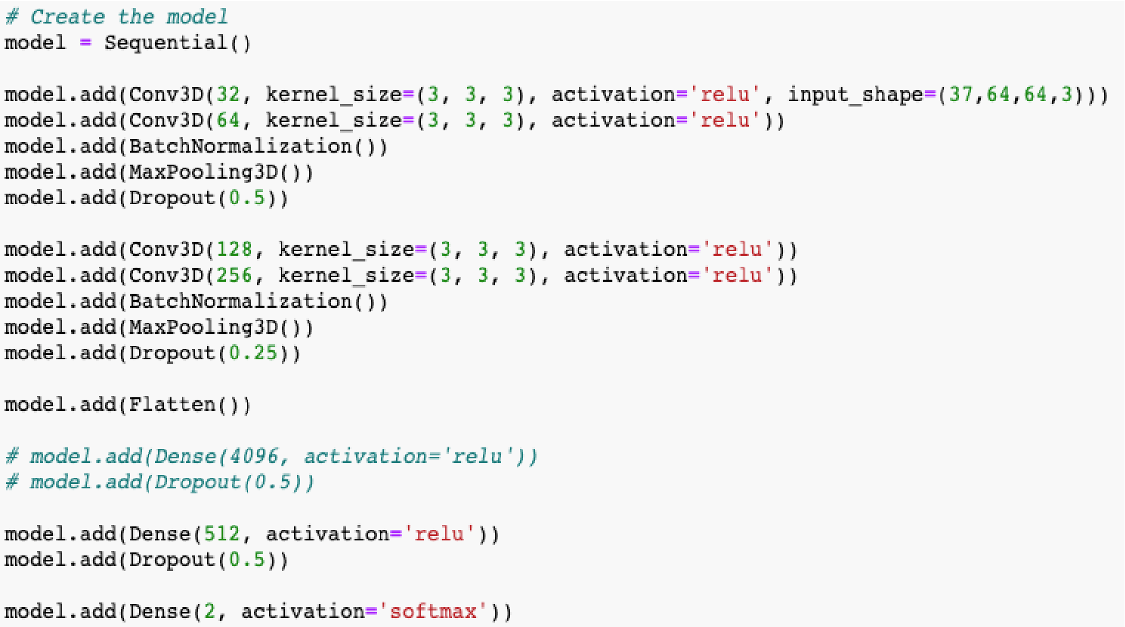
4.5 Image Classification

This thesis includes experiments to classify CT scans into AHT and Non-AHT classes without any image segmentation techniques and slice level labels. 2D and 3D classifiers are built using CNN based architecture. The results are evaluated using binary accuracy, validation accuracy and validation loss.

4.5.1 3D Classifier

A Brain CT scan consists of multiple slices with each slice representing a cross section area of the brain. Radiologists use all the slices of the CT scan in order to make a diagnosis since it is possible that an abnormality is identified in a set of specific slices. 3D classifiers use all the slices as a patient input (shape: number of slices, row, col, channels). This helps them leverage the knowledge from the entire set of images as whole and use 3D kernels to extract features. 2D classifiers are unable to leverage the context from adjacent inputs. 3D classifiers are more resource intensive because of the large input so a lot of network tuning is required to train the model in the GPU memory.

A custom 3D classifier was written in Keras. The classifier consisted of 3D convolution layers with kernel size of (3, 3, 3) and relu activation. 3D pooling was used to reduce the output size of convolution layers. Batch Normalization and Dropout was used to increase robustness of the model and reduce any overfitting. Two dense layers were added in the end with the last layer having a softmax activation function to predict the probabilities of both the classes. The number of slices as input is 37. The model was trained on 132 Probable Abuse patients and 167 Probable Non Abuse patients and validated on 35 patients of each class. The model was compiled using a RMSProp optimizer with learning rate of 0.0001. The loss function was binary crossentropy and the model was evaluated using binary accuracy, validation accuracy and validation loss. The model was trained for 100 epochs with a batch size of 8. Figure 32 demonstrates the code snapshot for the classifier's configuration. The results of the classifier are discussed in the next chapter.



```
# Create the model
model = Sequential()

model.add(Conv3D(32, kernel_size=(3, 3, 3), activation='relu', input_shape=(37,64,64,3)))
model.add(Conv3D(64, kernel_size=(3, 3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling3D())
model.add(Dropout(0.5))

model.add(Conv3D(128, kernel_size=(3, 3, 3), activation='relu'))
model.add(Conv3D(256, kernel_size=(3, 3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling3D())
model.add(Dropout(0.25))

model.add(Flatten())

# model.add(Dense(4096, activation='relu'))
# model.add(Dropout(0.5))

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))
```

Figure 32: Code Snapshot of 3D Classifier

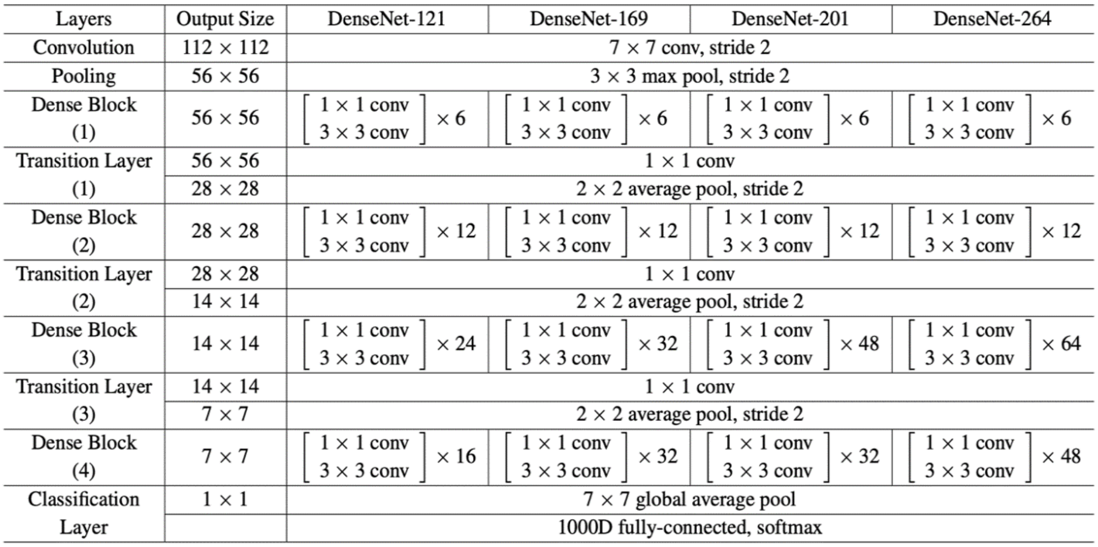
4.5.2 2D Classifiers

2D classifiers are computationally less expensive than 3D classifiers. They are generally used when slice-level label is available since it is possible that all the slices are not representative of the class. Since slice-level label was unavailable and data annotation was too costly at this instance, all the slices belonging to AHT patients were merged and saved as png files after windowing. The same was done to all the slices belonging to Non-AHT patients. In total, the dataset consisted of 6179 slices for AHT patients and 7400 slices for

Non-AHT patients. After data augmentation, the 2D models were trained on 10864 images and tested on 2715 images. Three pretrained models were used, namely, DenseNet121, ResNet50 and InceptionV3. In the experiments, the pretrained models (on ImageNet) were used as feature extractors.

4.5.2.1 Using DenseNet121

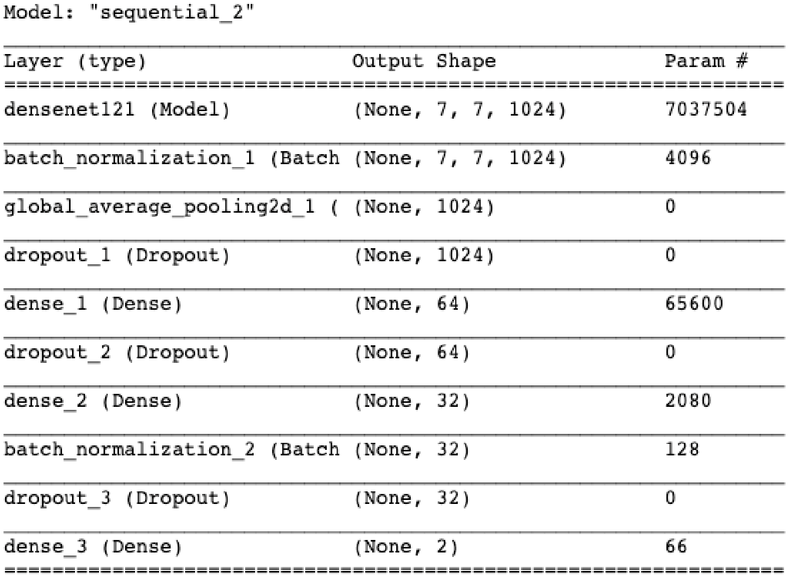
DenseNet (Densely Connected Convolutional Neural Network) was introduced at CVPR 2017 and achieved state of the performance on image classification datasets. It required less training parameters relative to other models at the time. Figure 33 explains the configuration of DenseNet family.



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Figure 33: Densenet121 Configuration; *Source: ("PyTorch.")*

The model is connected to a series of Dense layers with Batch Normalization and Dropout with relu activation. The last dense layer has a softmax activation to predict the probabilities of each class. The model is compiled with Adam optimizer (learning rate of 0.0001), binary crossentropy loss and evaluated using binary accuracy, validation accuracy and validation loss metrics. The model is trained for 50 epochs with a batch size of 16. The input image size is (224, 224, 3). Figure 34 shows the final model configuration with columns representing Layer type, Output shape and number of parameters.

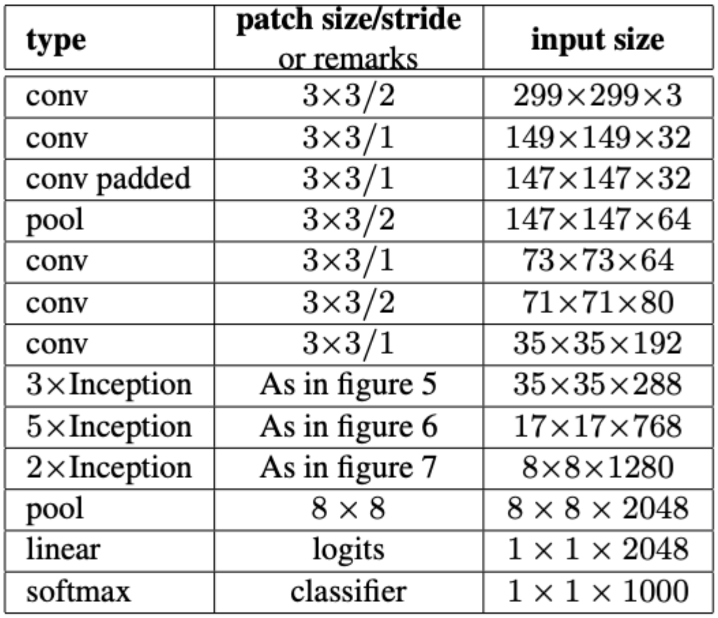


```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Model)          (None, 7, 7, 1024)        7037504
_________________________________________________________________
batch_normalization_1 (Batch (None, 7, 7, 1024)        4096
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                65600
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
batch_normalization_2 (Batch (None, 32)                128
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 66
=================================================================
```

Figure 34: DenseNet121 Used as the Base Model and Feature Extractor

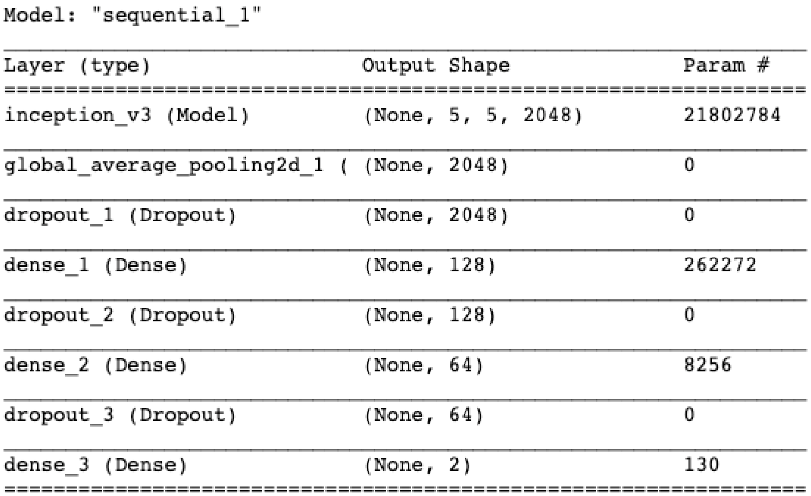
4.5.2.2 Using InceptionV3

The pretrained model InceptionV3 was released by Google at CVPR 2016. The model consists of 42 layers and was runner up at the ILSVRC (ImageNet Large Scale Visual Recognition Competition). The configuration details of InceptionV3 are explained in the Figure 35 with each column representing the layer type, the stride value and the input size of the layer.



type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 35: InceptionV3 Configuration; *Source: ("PyTorch.")*



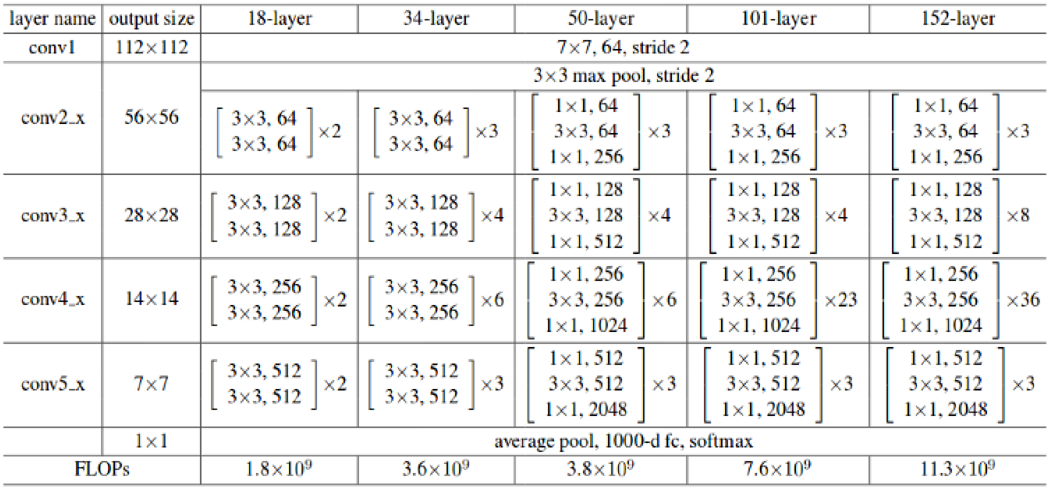
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
inception_v3 (Model)         (None, 5, 5, 2048)        21802784  
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               262272    
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256      
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 130       
=================================================================
```

Figure 36: InceptionV3 Used as Base Model and Feature Extractor

The base model (as shown in Figure 36) is followed by a series of Dense layers (with relu activation) and dropouts. The last dense layer has a softmax activation to predict the probabilities of each class. The input image size is (224,224,3) and the training is done in batches of 16. Data Generators are used to augment images and feed them to the model. The model is compiled with Adam optimizer (with a learning rate of 0.0002) and the loss is calculated using the binary crossentropy loss function. The model is trained for 50 epochs and evaluated on the binary accuracy, validation accuracy and validation loss metrics.

4.5.2.3 Using ResNet50

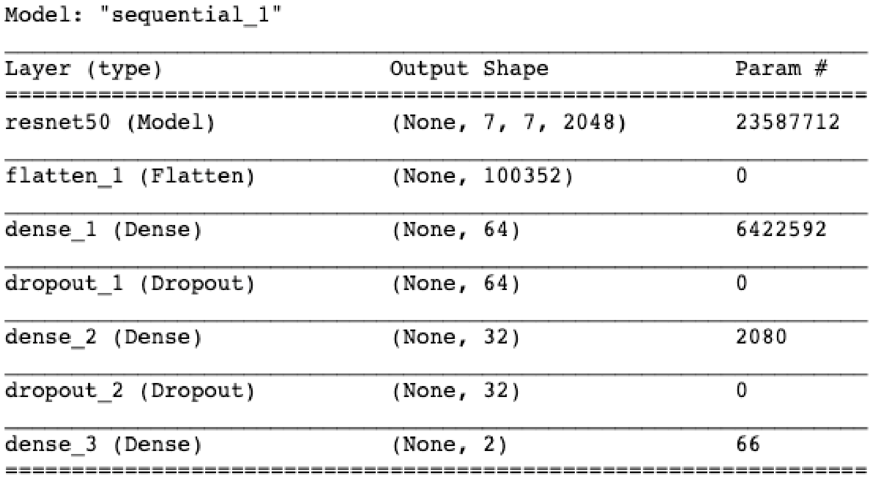
As the name suggests, Resnet50 has a depth of 50 layers. It was originally trained on over 1 million images and 1000 classes. It was the winner of 2015 ILSVRC challenge. Figure 37 illustrates the configuration details of the Resnet family.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 37: Resnet Family Configuration Details; *Source: (*“PyTorch.”*)*

The pretrained model is followed by two Dense layers (with relu activation) and dropouts. The output layer is a Dense layer with softmax activation. Images are fed using Keras’s ImageDataGenerator function which is capable of splitting data into train and validation sets. The input image has dimension (224, 224, 3). The model is compiled using RMSprop optimizer with learning rate of 0.0002. The loss function chosen is binary crossentropy. The model is trained for 50 epochs and evaluated on the binary accuracy, validation accuracy and validation loss metric. The configuration details are explained in Figure 38 with each column representing the layer type, the output shape and the number of parameters.

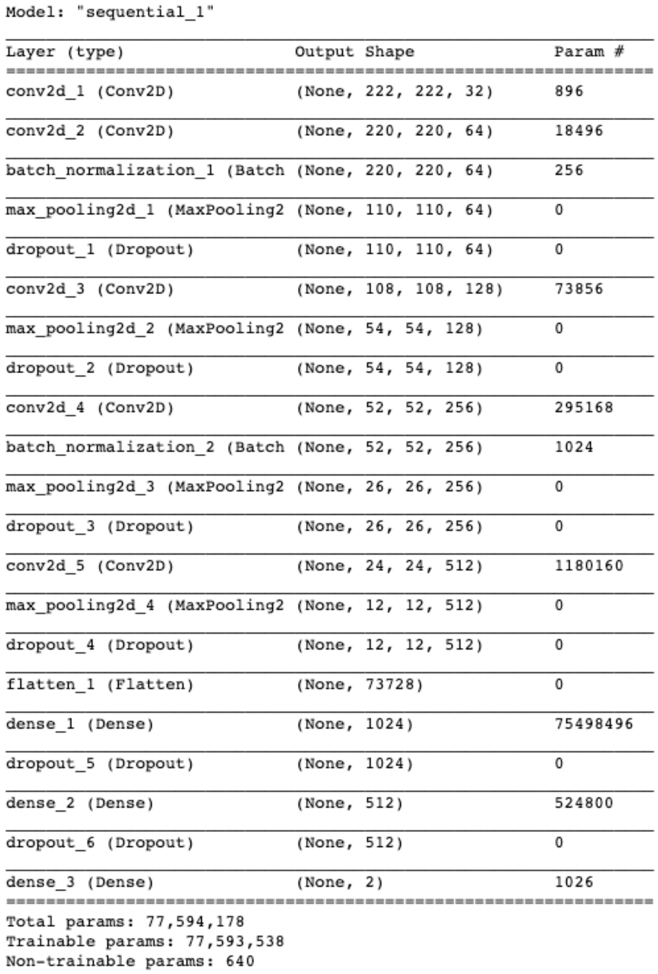


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 7, 7, 2048)        23587712
_________________________________________________________________
flatten_1 (Flatten)          (None, 100352)            0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                6422592
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 66
=================================================================
```

Figure 38: ResNet50 Used as Base Model and Feature Extractor

4.5.2.4 Using 2D custom classifier

A custom 2D classifier is written in Keras to compare the performance with the pretrained models used. Keras’s ImageDataGenerator is used to augment the training using rotation, shear, zoom and flipping. The data generator creates batches of images to train and validate the model. The batch size used is 16. The model consists of a four Convolution layers with Pooling, dropout and Batch Normalization in between. Three Dense layers are added with dropouts. All the layers except the last dense layer have relu activation functions. The last dense layer is the output layer with softmax activation to predict the probabilities of the classes. The model is compiled using RMSprop optimizer with a learning rate of 0.0002. The loss function used is binary crossentropy. The model is trained for 80 epochs and is evaluated using binary accuracy, validation accuracy and validation loss metric. The configuration details are explained in Figure 39 with each column representing the layer type, the output shape and the number of parameters.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 220, 220, 64)      18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 220, 220, 64)      256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 110, 110, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 110, 110, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 108, 108, 128)     73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 54, 54, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 54, 54, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 52, 52, 256)       295168
_________________________________________________________________
batch_normalization_2 (Batch (None, 52, 52, 256)       1024
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 26, 26, 256)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 26, 26, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 512)       1180160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 512)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 12, 12, 512)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 73728)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              75498496
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 1026
=================================================================
Total params: 77,594,178
Trainable params: 77,593,538
Non-trainable params: 640
```

Figure 39: Model Configuration for Custom 2D Classifier

4.6 Building APIs Using Flask, Docker and Kubernetes

Flask is a python-based web framework used widely for rapid prototyping of applications (“Flask.”). The thesis uses the micro-service architecture to build APIs using Flask. Three functional APIs are built based on three sub-domains of the thesis: Data Pre-processing, Image Synthesis and Image Classification. A fourth API is built to upload the slices provided by the user as input. The development was done in PyCharm (“PyCharm.”) IDLE using Python (version 3.6). Conda Package manager (“Conda.”) was used to create a virtual environment and isolate all the dependencies of the project.

4.6.1 System Architecture

The overall architecture of the web tool is demonstrated in the Figure 40. AWS Dynamo is used as the database for the application. It is a fully managed NoSQL database which is well integrated with other components of the AWS Ecosystem. The files (CT slices) uploaded by the user are stored in a S3 bucket. S3 is a highly scalable and available object storage service offered by AWS. The Flask application communicates with S3 and Dynamo using AWS python client called Boto3. Dynamo contains the mapping to the unique user identifier and the S3 path where their uploaded files are stored.

Each time a user accesses the web tool, a unique identifier is generated for the user. The user can then upload the CT slices to either view them in the correct window setting or predict the occurrence of AHT. The uploaded slices are then stored in S3 and the dynamo table is updated with the right mapping. Once the files are uploaded, the user can select the use case, which is either view the slices in the right window or get the prediction results. For both the use cases, the S3 bucket path is retrieved from the dynamo table using the user identifier as the key. Once the S3 path is available, the CT slices are downloaded and processed using Python scripts. Since Flask is a lightweight framework, it is not advisable to store CT slices inside the application memory. Moreover, the AWS cloud components are highly scalable in nature and have low latency.

Flask itself comes with a small server which can be used for local testing, but it is not advisable to use it in production setting. A real web server is needed to handle a large amount of requests. This is where NGINX and uWSGI comes into play. NGINX is a web proxy server which handles the client requests. The requests are transferred to uWSGI using UNIX sockets. uWSGI is a Web Server Gateway Interface capable of handling multiple requests at a time. It is capable of invoking a callable object within Flask application to sever the requests.

Docker is the de-facto choice of container used throughout industry. A Docker Image is created for the web application using a Dockerfile. The external package requirement is managed using a requirements.txt file which can be explicitly stated in the Dockerfile. Once the docker image is created, it is pushed to a container registry. The container registry of choice is Amazon ECR (Elastic Container Registry). AWS-Command Line Interface is used to upload the Docker image to ECR.

Kubernetes is becoming an industry standard for managing a large number of complex containers on the cloud. It makes deployment and scaling of container simple. The thesis uses Amazon EKS (Elastic Kubernetes Service) because of its smooth support and low latency with other AWS components in the ecosystem. In order to deploy the container at scale, a Kubernetes cluster is created. Kubectl, the native Kubernetes command line interface is used for communicating with the cluster. Once the cluster is created, the container is launched into the cluster pods using a Kubernetes deployment. The pods are not exposed to the outside world and in order to connect with them, a Kubernetes service, called load balancer is used. The load balancer exposes the pods with an external IP address. The External IP is used to communicate with the web tool (running inside the pod) from Internet.

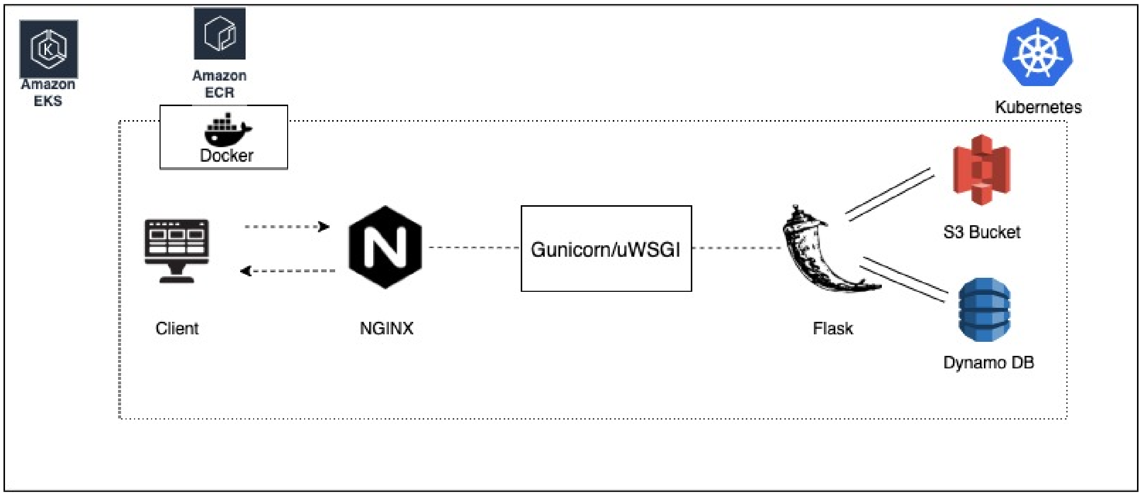


Figure 40: Architecture Diagram of the Web Application

4.6.2 User Interface

The web tool has a user interface built using HTML and CSS. Once on the landing page (as shown in Figure 41), the user has two options to select, either upload the CT Dicoms and get redirected to the next page, or, using the Image Generation option to generate Brain CT images using DCGAN. Multiple DICOM images can be chosen at once and uploaded. Upon clicking the upload button, a post request is submitted to the *upload_files* API. The request contains the CT slices as data. The user is then redirected to the display_options page (as shown in Figure 42).

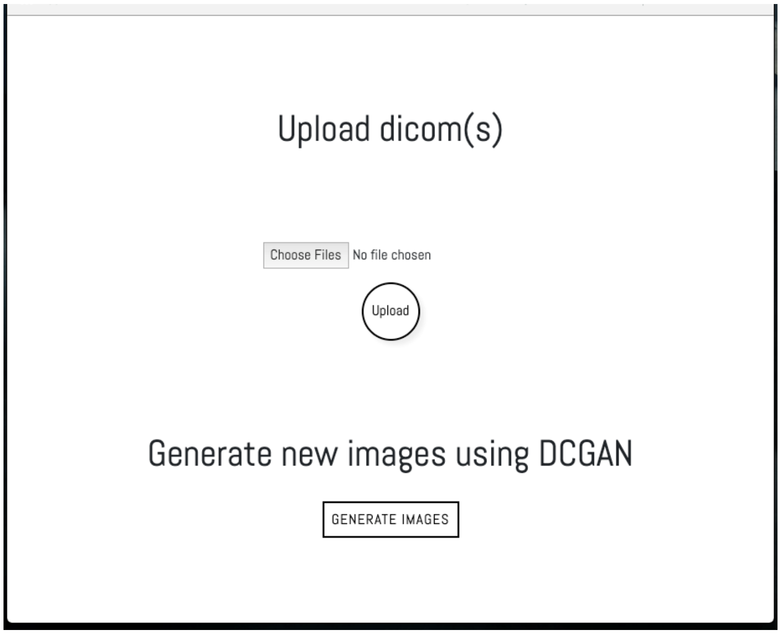
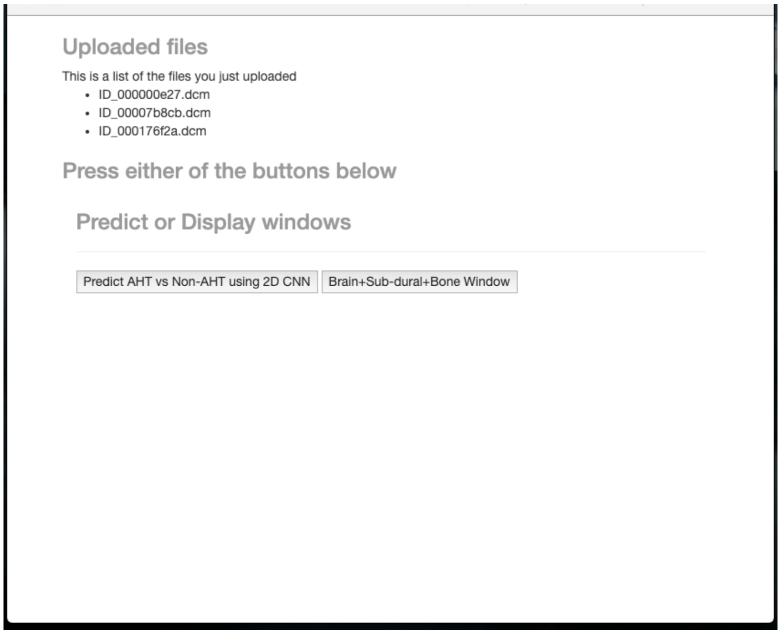


Figure 41: Snapshot of the Landing Page with Upload DICOM and Generate Image


Uploaded files

This is a list of the files you just uploaded

- ID_000000e27.dcm
- ID_00007b8cb.dcm
- ID_000176f2a.dcm

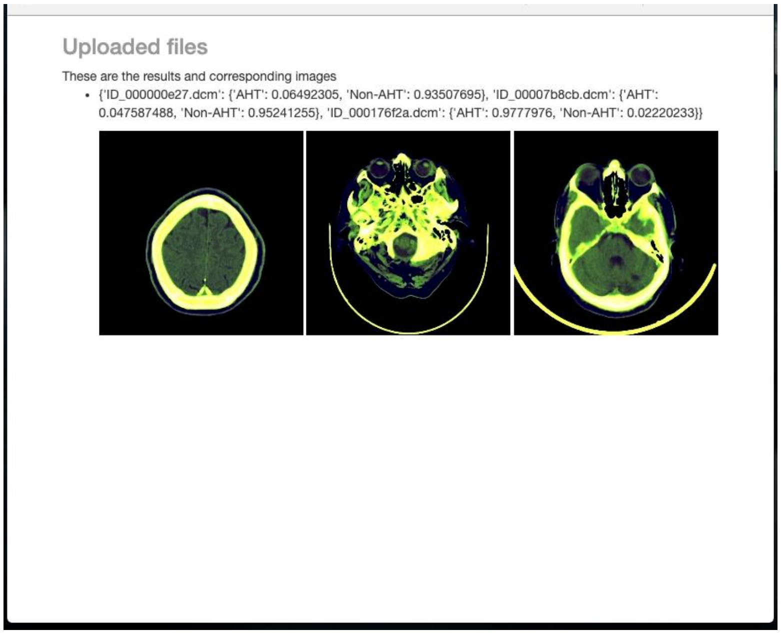
Press either of the buttons below

Predict or Display windows

Predict AHT vs Non-AHT using 2D CNN
Brain+Sub-dural+Bone Window

Figure 42: Snapshot of the display_options Page with Predict and View Options

Upon clicking the predict button, a POST request is submitted to the *predict_2dcnn* API with the user identifier generated in the landing page as the request parameter. A database call is made to retrieve the S3 bucket path containing the stored CT slices, the slices are fetched and then fed into the trained model for the prediction. The API returns the probability for each image of falling into the AHT and Non-AHT class as demonstrated in Figure 43.



Uploaded files

These are the results and corresponding images

- {'ID_000000e27.dcm': {'AHT': 0.06492305, 'Non-AHT': 0.93507695}, 'ID_00007b8cb.dcm': {'AHT': 0.047587488, 'Non-AHT': 0.95241255}, 'ID_000176f2a.dcm': {'AHT': 0.9777976, 'Non-AHT': 0.02220233}}

Figure 43: Snapshot of result_2dcnn Page Displaying Prediction Results

Upon clicking the Brain-Subdural-Window button on display_options page, a GET call is made to the *show_windows* API with the user identifier as the request parameter. Again, a dynamo call is made to retrieve the S3 bucket path containing the stored CT slices, the CT slices are fetched and then fed into multiple data preprocessing functions to generate the final images in the right window setting. The response object of API contains the images and displayed on window_view page (as shown in Figure 44).

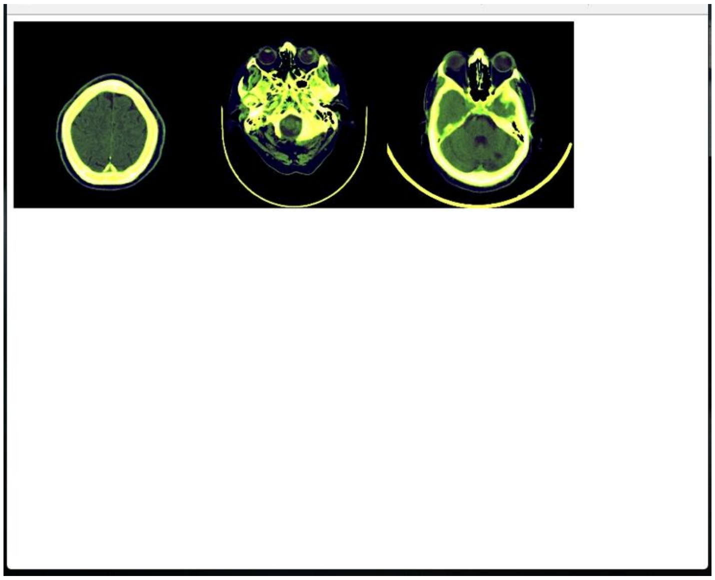


Figure 44: Snapshot of window_view Page Displaying Slices After Windowing

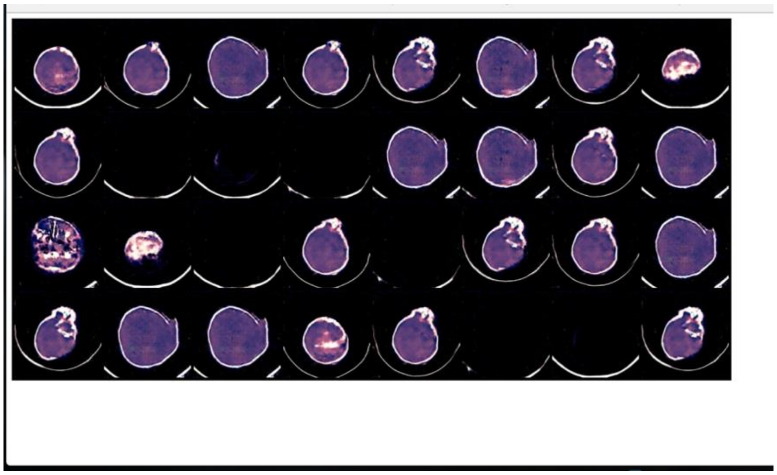


Figure 45: Snapshot of gan_view Page Displaying Generated Images

If the user selects the generate images option in the landing page, a GET call is made to the *generate_new_images* API to generate new images using the DCGAN. A grid of images is generated and returned as the response object. The grid is displayed in the gan_view webpage as shown in Figure 45.

4.6.3 Packages and Libraries

A modular approach to programming has been followed while building the web tool. The programs are broken into multiple functions with each function responsible for a single task. The APIs make use of the functions for processing data. Functions with similar usage have been abstracted into segregated into separate python files to give more meaning to the files. Three packages have been created, namely, models, trained_models (which is a sub-package of models) and templates. Figure 46 shows the packages.

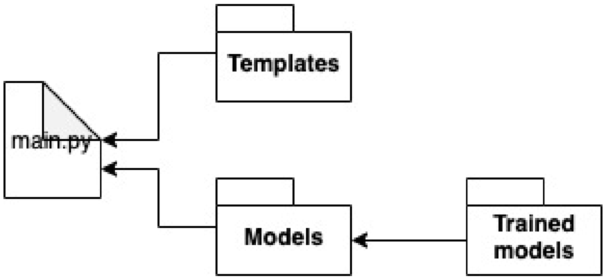


Figure 46: Package diagram of the Web Tool

The trained model package contains the 2D trained model and trained generator of DCGAN. The models package contains data preprocessing helper functions. The template package contains the html files and static css files. Other helper functions are present in independent python scripts in the base directory along with the main.py file which is the starting point of the web application.

Writing code from scratch was simply not possible for all the functionalities due to complexity involved in handling Medical Data and training models. Furthermore, using libraries minimizes time spent in writing code from scratch. It also helped in making the application more production ready. The Data-Preprocessing steps used libraries such as PyDicom (to read Dicom scans), Numpy (for handling numerical arrays), Matplotlib (for making graphs and plotting images) and PIL (pillow for image analysis). The Deep Learning models are trained using Keras, Tensorflow and PyTorch. Boto3 was used to interact with AWS components and Json was used for data conversion. Flask library was used to create the web application werkzeug to extract files from POST requests. All the libraries helped in building the web tool more robust.

CHAPTER 5

RESULTS AND ANALYSIS

This chapter provides a review and analysis of the results obtained from running the experiments described in the previous chapter. The chapter is divided into three parts, the results of trained classifiers, the results of trained GANs and the results of load testing the Kubernetes cluster where the web tool resides.

5.1 Results of Trained Classifiers

2D classifiers were trained on AHT and Non-AHT labels. Since the slice level label was not available, the 2D classifiers were trained on all the slices using the patient level label. The 2D models were trained on 10864 images and tested on 2715 images. Each image was first converted into the Brain-Subdural-Bone window and then resized to 224x224. All the slices for each class were stacked and read using Keras's ImageDataGenerator. The Pre-trained models were used as feature extractors and then connected to classifier. Hyperparameter tuning was done for each model and best hyperparameters were selected for comparison with other models.

3D classifiers were trained on AHT and Non-AHT Patient level label. For each patient, the first 37 slices were selected for training and testing, and the dataset consisted of 167 Probable Abuse and 202 Probable Not Abuse patients. It was then divided into testing and validation set (80-20) split and then converted to numpy arrays which served as input to 3D classifier. Table 1 summarizes the performance of the 2D and 3D classification models.

Table 1

Classification Results of Different Models

Model Type	Binary Accuracy	Validation Accuracy	Loss	Optimizer
2D custom model	0.5429	0.5435	2.87571	RMSprop
2D Pretrained Model with Resnet50	0.6566	0.5435	0.53523	RMSprop
2D Pretrained Model with Densenet121	0.9068	0.6236	0.31526	Adam
2D Pretrained Model with InceptionV3	0.5674	0.5443	0.61241	Adam
3D Custom Model	0.5574	0.5625	0.6175	RMSprop

Table 1 demonstrates that the 2D and 3D classifier do not perform well on the available dataset with few models overfitting the training data.

5.1.1 2D Classifier Results

For each 2D classifier, the corresponding accuracy and loss graph is presented. They illustrate the performance of the classifier during the training process. Each 2D classifier is trained for 80 epochs with the hyperparameters selected after tuning the models. The input to each classifier was an image (slice) of size 224x224.

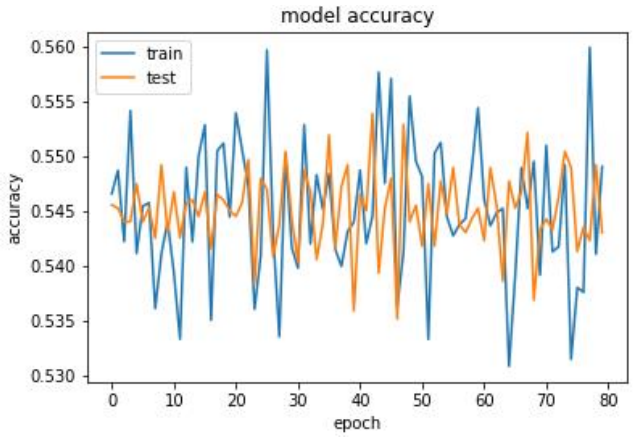


Figure 47: Model Accuracy for 2D Custom Classifier

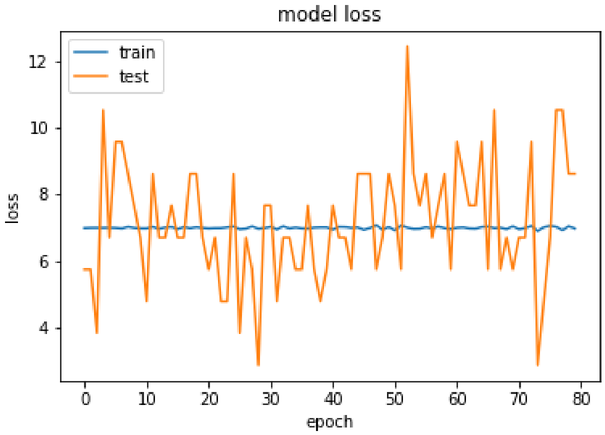


Figure 48: Model Loss for 2D Custom classifier

As visible by the graphs in Figure 47 and Figure 48, the 2D custom classifier does not overfit the data but performs poorly. It is unable to learn new features as evident from the binary as well as the validation accuracy. The train loss is almost constant since no new features are being learned and the output is always random (as validated from the testing loss).

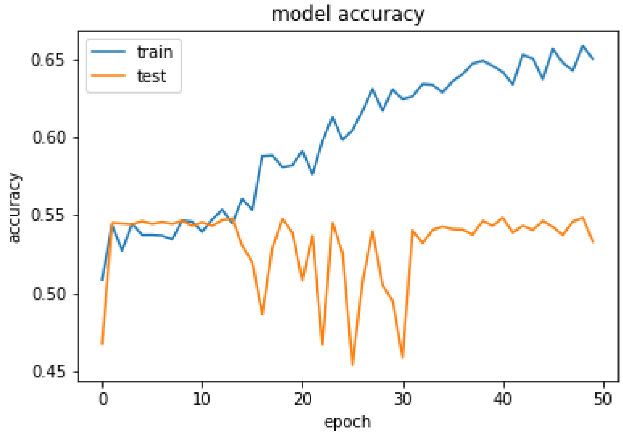


Figure 49: Model Accuracy for 2D Classifier with Resnet50 Base

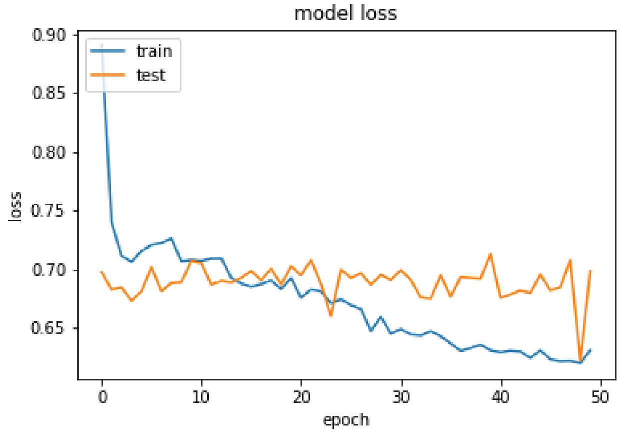


Figure 50: Model Loss for 2D Classifier with Resnet50 Base

For ResNet50 pretrained model, the model accuracy graph (Figure 49) is indicative of the model learning new features in the starting epochs but starting to overfit the data in the later epochs. The training loss is decreasing throughout the graph due to overfitting of the model while the validation loss is almost constant (as shown in Figure 50) as no new generalized feature is learned by the model.

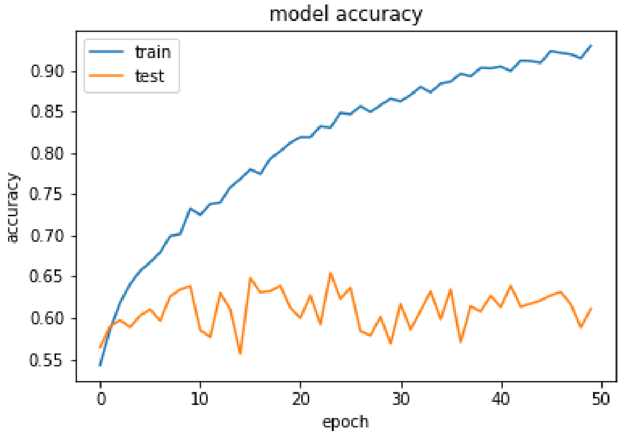


Figure 51: Model Accuracy for 2D Classifier with Densenet121 Base

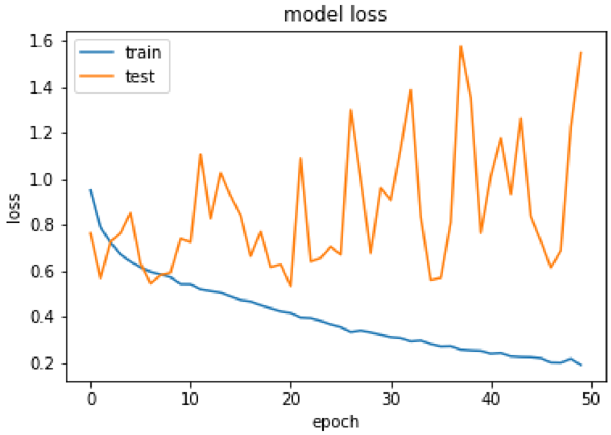


Figure 52: Model Loss for 2D Classifier with Densenet121 Base

The Densenet121 base model overfits the most when compared to the other trained models. Training accuracy of over 90% is achieved but the validation accuracy remains low (but higher than other models). The training loss can be seen constantly decreasing with the increase in number of iterations while the validation accuracy varying quite a bit as evident from Figure 51 and Figure 52.

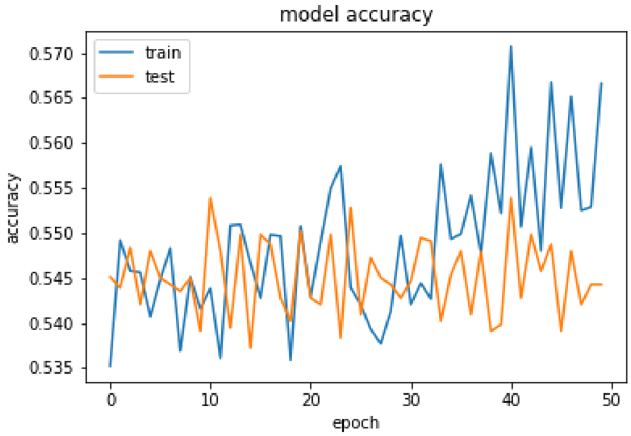


Figure 53: Model Accuracy for 2D Classifier with InceptionV3 Base

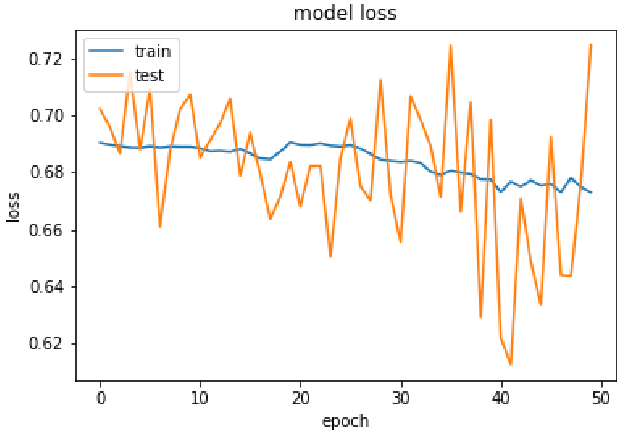


Figure 54: Model Loss for 2D Classifier with InceptionV3 Base

The IncetionV3 pretrained models performs similarly as the other models but does not overfit the data. The training and validation accuracy do not change a lot and the training loss is almost constant as evident from Figure 53 and Figure 54.

5.1.2 3D Classifier Results

The 3D classifier model achieved a training accuracy of 0.5743 and validation accuracy of 0.5000 at the end of 100 epochs. Figure 55 and Figure 56 demonstrate the accuracy and loss during training.

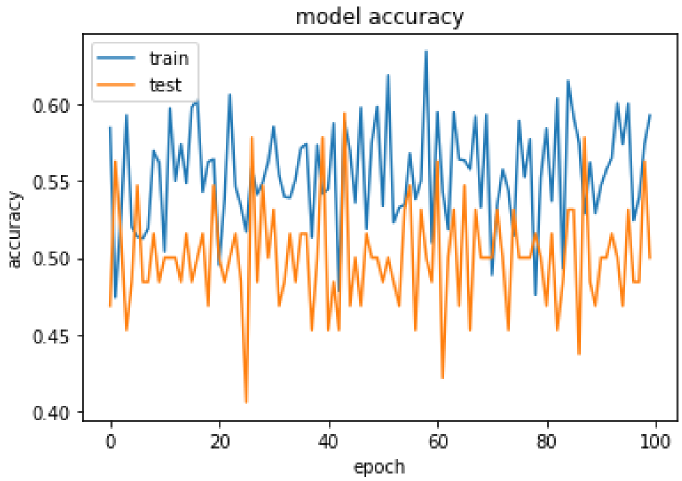


Figure 55: Model Accuracy for 3D Custom Classifier

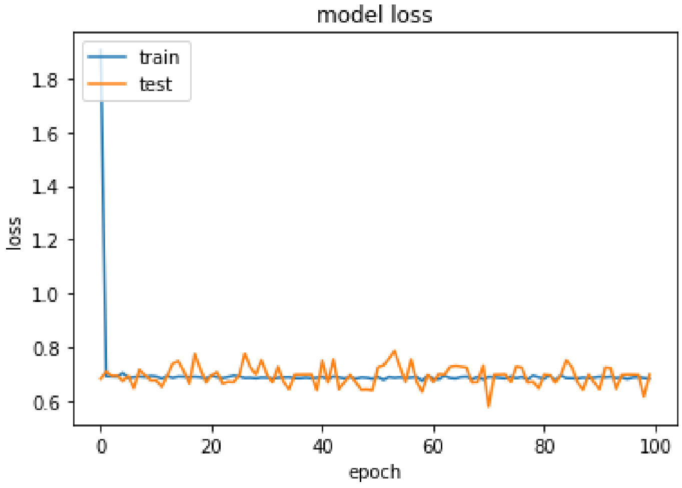


Figure 56: Model Loss for 3D Custom Classifier

5.1.3 Remarks

The models were trained on a RTX 2060 GPU with up to 6 GB of available memory. The poor training accuracy for 2D and 3D models is due to multiple reasons with the first being unavailability of slice level labels. The range of slice for each patient is between 15 – 45 with average number of slices being 37. Each slice describes a different cross-section of the brain and all the slices for a patient are not reflection of the class label. This leads to discrepancy in the training data. Since Image Annotations are very costly and medical image annotation require hours of a radiologist, the thesis conducted experiments using the patient level labels.

Another reason for the poor results is the limited amount of data. Deep Learning techniques require a large amount of data to learn generalized features. Since the data set was limited, data augmentation techniques were used to increase the data, but they were not true reflection of the original data. One of the reasons to perform GAN experiments was to check the viability of using their results for data synthesis.

The last reason for low performance could be the use of small batch sizes for training. Since the GPU has limited memory, training was conducted using small batch sizes and batch gradient descent was used.

5.2 Results of Trained DCGAN

Two different images sizes were generated using DCGANs: 32x32 and 128x128. Below are the results of the experiments.

5.2.1 32x32 DCGAN

The DCGAN was trained on a batch size of 16. Figures 57, 58, 59 and 60 demonstrate results during training and Figure 61 shows a newly generated image after training.

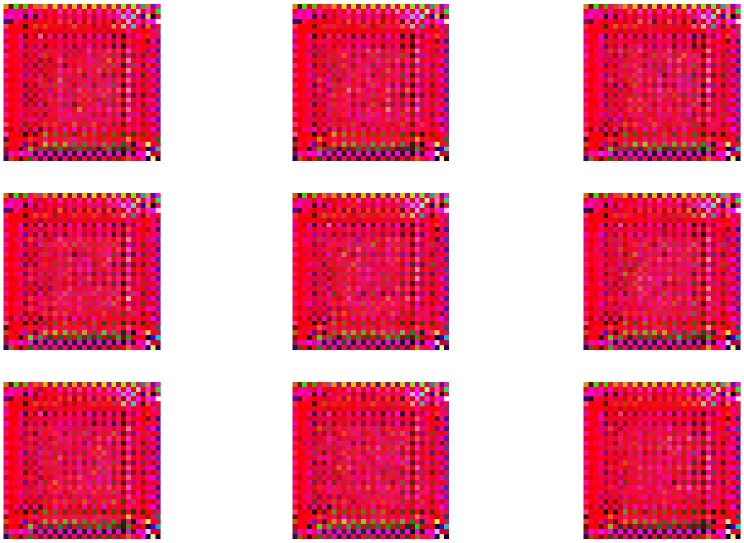


Figure 57: Images Generated after Epoch 1

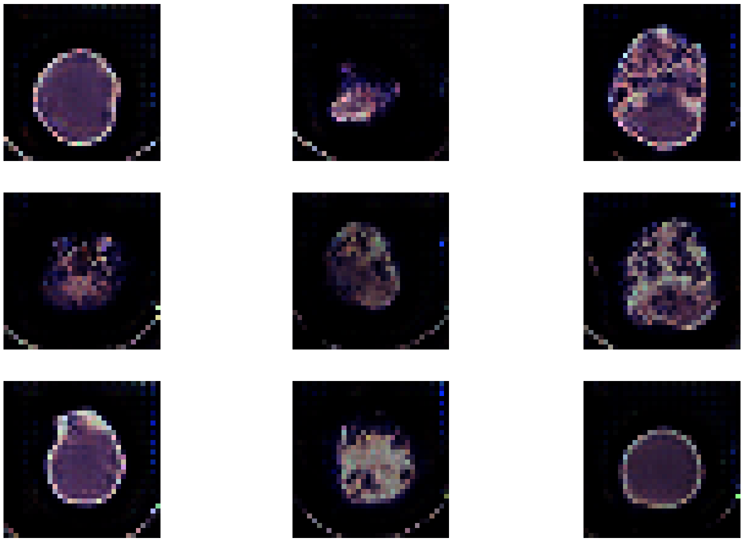


Figure 58: Images Generated after Epoch 30

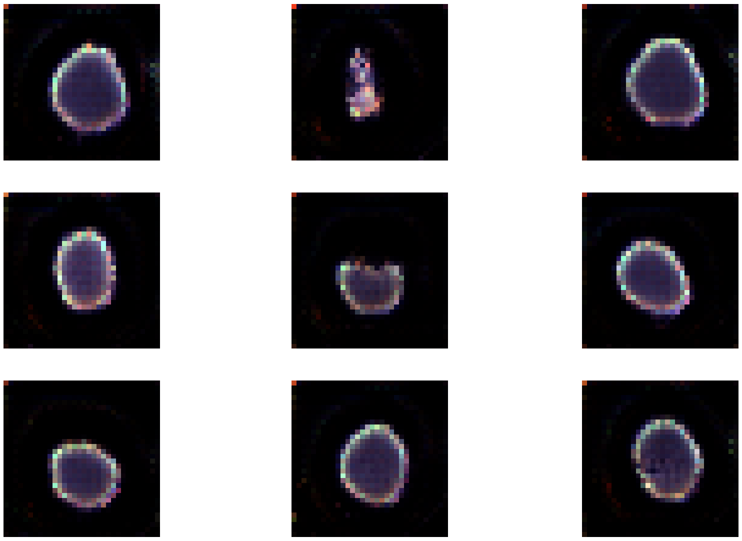


Figure 59: Images Generated after Epoch 50

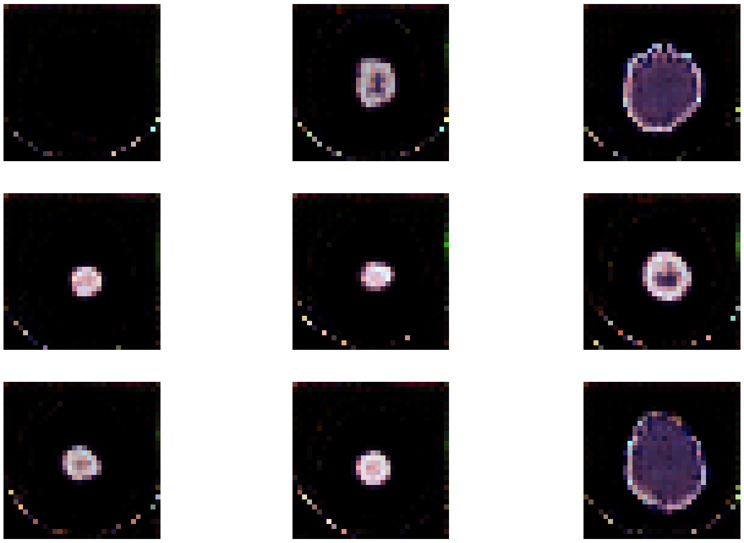


Figure 60: Images Generated after Epoch 80

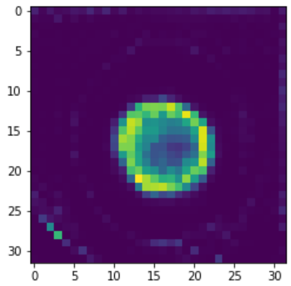


Figure 61: A Randomly Generated Image after Epoch 80

5.2.2 128x128 DCGAN

The DCGAN was trained for 200 epochs with a batch size of 8. The Generator and Discriminator loss was visualized throughout the training process. Since generator and discriminator are adversaries in nature, both the losses remain constant in order to become better at what they do. Figure 62 shows the loss and Figure 63 shows a grid of images generated after training.

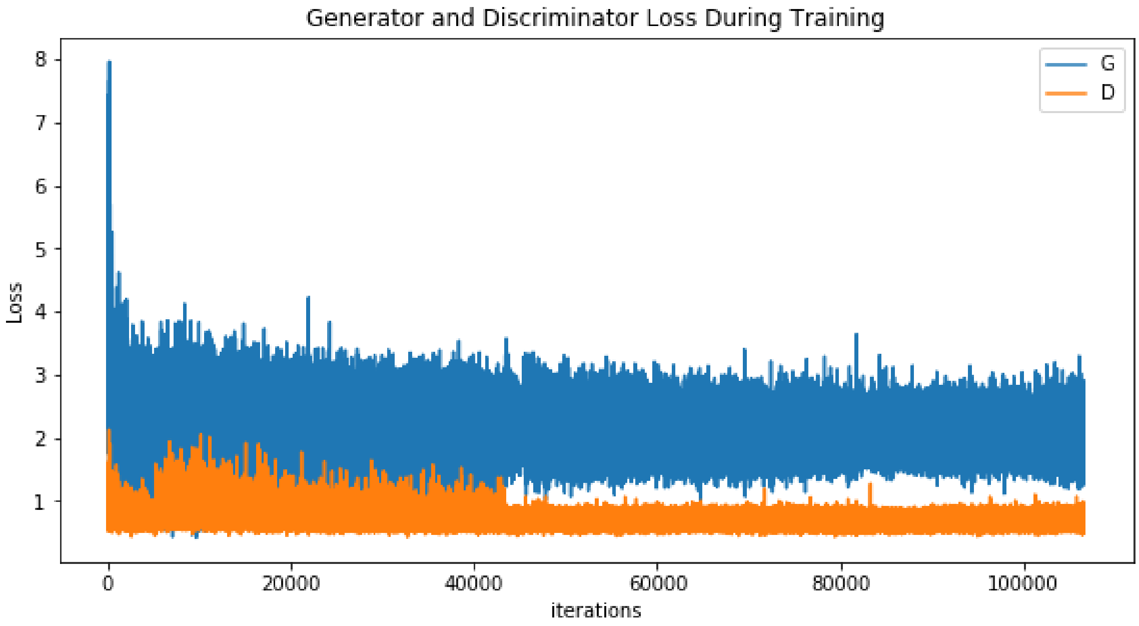


Figure 62: Generator and Discriminator Loss During Training for 128x128 DCGAN

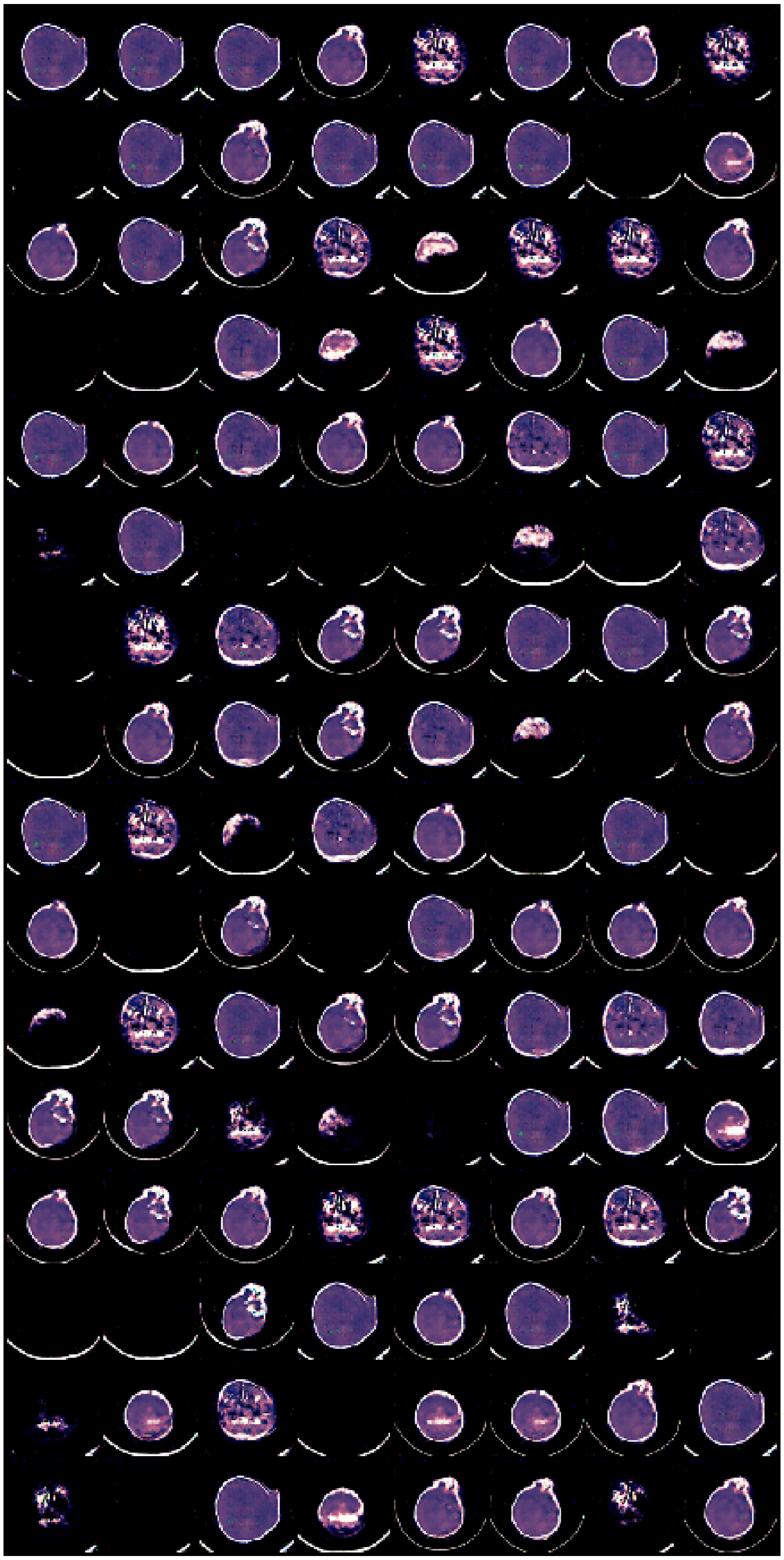


Figure 63: Grid of 170 Images Generated after 200 Epochs

5.2.3 Remarks

GANs are notoriously hard to train and tuning the hyperparameters is a daunting task. Even a slight change of a single hyperparameter may result in the loss becoming zero rapidly. Images with better contrast can be generated by refining the training data and increasing the batch size. The training data used had all the slices of the patients. Some of the initial slices do not display a relevant information and contain only the skeletal part of the head. Those slices can be removed to increase the quality of the images. Also, using higher dimension images will lead to learning better features from the images. Since the GPU memory was limited, images of higher dimensions could not be trained.

5.3 Load Testing the Kubernetes Cluster

The Kubernetes cluster is tested using Siege. It is an open source command line tool used for load testing. It tests the cluster by simulating traffic with a configurable number of users.

The cluster consists of a Node group created using EC2 instances. There default node size is 1 with up to 5 nodes available during high load. Each EC2 instance is of type m5.large and a disk size of 20 Giga Bytes for each node.

Once the web tool (contained using Docker and stored in ECR) is deployed, it has to be exposed using Load Balancers so that it is accessible by internet. Load Balancers are created based on the Virtual Private Cloud and spin up the default number of instances.

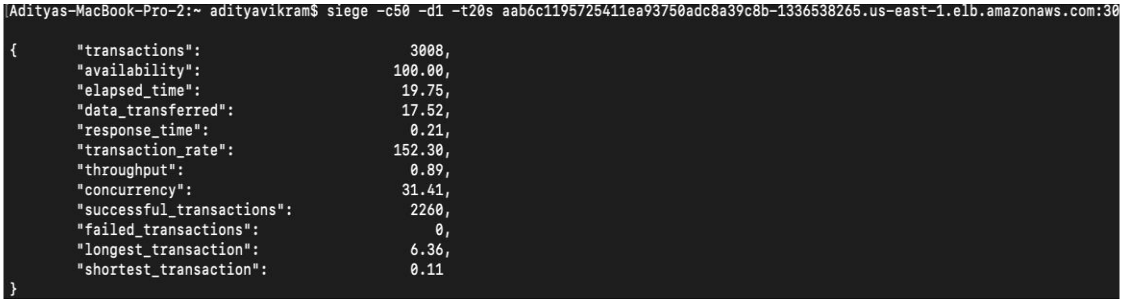
Three load tests are run in siege, namely, test the landing page, test the generate_new_images page and test the predict_2dcnn page. The results are displayed in Table 2 and explained later.

Table 2

Load Testing Using Siege

Page	Transactions	Availability	Throughput	Failed Transactions
Landing Page	3008	100%	89%	0
Generate New Images	25	100%	57%	0
Predict Model	0	0	0	15

The landing page is load tested by adding 50 concurrent users with a delay of 1 second. The results demonstrate that the site is available 100% time with a throughput of 89%. Since the landing page is a static in nature, all the transactions are successful, and the cluster does not have to auto-scale (Horizontal scaling is sufficient). The results of the test are visible in Figure 64.

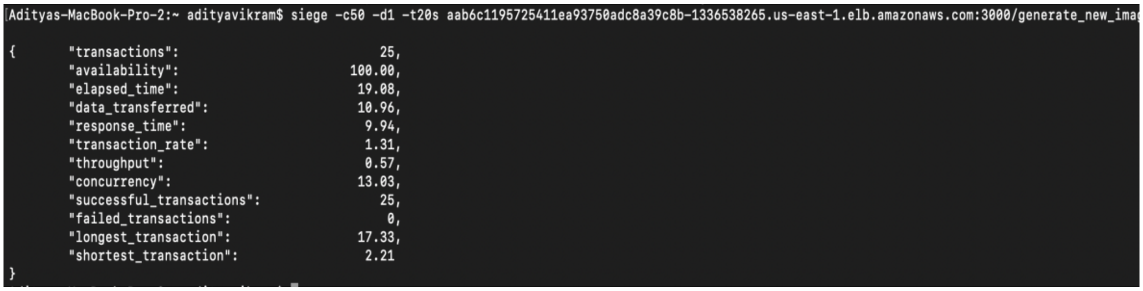


```
Adityas-MacBook-Pro-2:~ adityavikram$ siege -c50 -d1 -t20s aab6c1195725411ea93750adc8a39c8b-1336538265.us-east-1.elb.amazonaws.com:30

{       "transactions":                      3008,
        "availability":                    100.00,
        "elapsed_time":                     19.75,
        "data_transferred":                 17.52,
        "response_time":                     0.21,
        "transaction_rate":                152.30,
        "throughput":                        0.89,
        "concurrency":                      31.41,
        "successful_transactions":           2260,
        "failed_transactions":                  0,
        "longest_transaction":               6.36,
        "shortest_transaction":              0.11
}
```

Figure 64: Load Testing the Landing Page

The generate_new_images page is load tested by adding 50 concurrent users with a delay of 1 second. The results demonstrate that the page is available 100% time with a throughput of 57% (as evident from Figure 65). The number of transactions is very few compared to the landing page because of the time to complete the API request. For each API request, the Generator model is loaded which itself is time consuming. Nevertheless, the pods are able to auto-scale (horizontal scaling) to satisfy the requests.

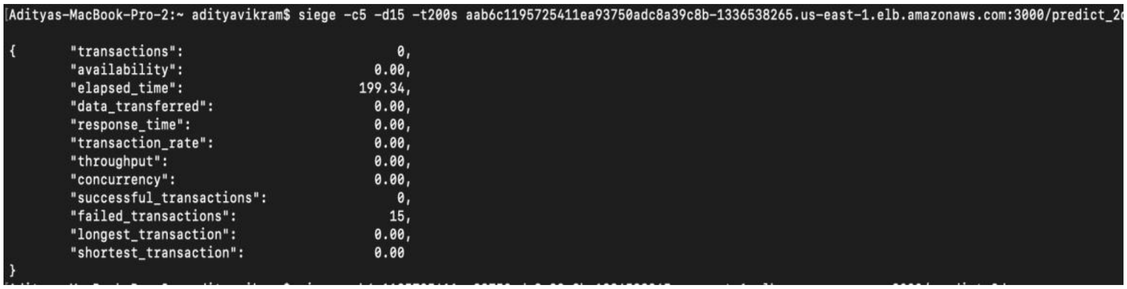


```
[Adityas-MacBook-Pro-2:~ adityavikram$ siege -c50 -d1 -t20s aab6c1195725411ea93750adc8a39c8b-1336538265.us-east-1.elb.amazonaws.com:3000/generate_new_imag

{       "transactions":                         25,
        "availability":                     100.00,
        "elapsed_time":                      19.08,
        "data_transferred":                  10.96,
        "response_time":                      9.94,
        "transaction_rate":                   1.31,
        "throughput":                         0.57,
        "concurrency":                       13.03,
        "successful_transactions":              25,
        "failed_transactions":                   0,
        "longest_transaction":               17.33,
        "shortest_transaction":               2.21
}
```

Figure 65: Load Testing the generate_new_images Page

The predict_2dcnn page is load tested by adding 5 concurrent users with a time delay of 15 seconds. The results show (as evident from Figure 66) that the site becomes unavailable after being unable to serve the number of requests. This is due to the calls happening in the backend. In order to use the predict function, multiple packages are loaded and then the model is read and compiled. This itself takes ~120 seconds. Also, the size of the model and packages are big, and all of these are loaded for every request which puts a load on the pod resources. Once the model is compiled, it is able to serve the requests. But the load testing tool times out the request if the responses are not received within 200 seconds.



```
[Adityas-MacBook-Pro-2:~ adityavikram$ siege -c5 -d15 -t200s aab6c1195725411ea93750adc8a39c8b-1336538265.us-east-1.elb.amazonaws.com:3000/predict_2d

{       "transactions":                         0,
        "availability":                      0.00,
        "elapsed_time":                    199.34,
        "data_transferred":                  0.00,
        "response_time":                     0.00,
        "transaction_rate":                  0.00,
        "throughput":                        0.00,
        "concurrency":                       0.00,
        "successful_transactions":              0,
        "failed_transactions":                 15,
        "longest_transaction":               0.00,
        "shortest_transaction":              0.00
}
```

Figure 66: Load Testing the predict_2dcnn Page

Few steps which can be taken to scale predict_2dcnn function are increasing the resources for each pod, adding node groups to support cluster scaling, and hosting the model at an online repository (such as S3 bucket) which makes it easier to be loaded. It is worthwhile noting that the services are very expensive in terms of cost per use and they quickly stack up if the services are not controlled.

CHAPTER 6

CHALLENGES, CONCLUSION AND FUTURE SCOPE

This chapter illustrates the challenges faced while working on the research project with Phoenix Children Hospital as a volunteer researcher. A conclusion to the research work as part of the thesis is provided along with the next steps to complete the research study as part of the PCH IRB.

6.1 Challenges

The thesis was carried out with PCH and involved deidentified patient image. A great deal of work had to be completed by hospital clinical staff and researchers prior to the start of this study. All the work had to be conducted meeting strict regulatory standards and the requirements of the PCH IRB. There were multiple challenges regarding the logistics of accessing data, strict hospital security issues and attaching available labels. Starting with meeting the logistics of access to the data, I had to become credentialed for research at PCH as a student learner and as a volunteer researcher. This process took almost a month since it involved an extensive background check. Since the data was stored in the PACS system, the data had to be extracted from PACS while meeting the strict information technology security requirements of the hospital information system and the limitations of the PCH available hardware. My site liaison, Dr. Lois Sayrs, facilitated several significant meetings to increase our ability to access the data and store it for analysis. PCH provided the study with a separate HIPAA-compliant server to maintain initial data for de-identification for labelling. PCH also permitted deidentified data to be migrated to a secure

HIPAA-compliant partitioned space on a ASU supercomputer to facilitate data analysis. Once the credentialing was done, the data download was started. PACS system does not allow a large amount of data (over 10 Gigabytes) to be downloaded at once. So, the data had to be downloaded in batches. This process itself took over two weeks.

Another challenge was transferring data to a computer with GPU capabilities so that the processing could be faster. The security team at the hospital had to be involved to make sure the data did not contain any patient information. Also, since the size of the data itself was very large (~250 GiB), it was a challenge from transfer perspective.

The last major challenge is unavailability of slice level labels. Currently, the dataset contains labels (AHT versus Non-AHT) at the patient level where each patient has a slice range of 26-45. Dr. Sayrs assisted with labelling by confirming the data table before the table was matched to the subject ID for image data. All of the slices for each patient are not reflective of the label (some slices may contain irrelevant or confusing information). This is the main reason why the classification models showcase poor results.

6.2 Conclusion

The aim of the thesis was to model the domain of Abusive Head Trauma and divide it into different sub-domains. It served as a first attempt to solve all the stages of a novel medical problem using AI techniques and Best Software Engineering practices. Within the scope of the thesis, the Data Collection was completed and different strategies for Data Pre-processing were explored. A set of experiments were conducted to train Classification and Generative models. The results of these experiments concluded the need for slice level labels, image annotations and an increase in the available dataset.

The thesis also explored strategies for deploying trained models at scale by exposing them as APIs. These APIs were written in Python using Flask Backend and then contained using Docker. The container was then deployed using Kubernetes and auto scaled on Amazon’s EKS platform. The testing results from Siege concluded the need for horizontal and vertical auto-scaling of the cluster keeping in mind the costs associated with it.

6.3 Future Scope

Since this thesis is only a first pass at the problem of Detection of AHT, and this would be an ongoing project at PCH, a lot of work lies ahead. The starting point for future work should be Data Annotation. The thesis has established the need for slice level labels. Abusive Head Trauma itself is not a pathology which can be seen in CT scans, but it is inferred from a set of pathologies (such as SDH and Skull Fractures) which are seen on CT scans. Therefore, it is necessary to use segmentation techniques for data annotation at the slice level. If SDH can be segmented (and bounding box can be created), it can greatly enhance the results of the classifiers. Moreover, the annotations will help classifiers distinguish between the right pathologies. Models such as U-Net can be used for segmentation task. In order to generate slice level labels, techniques such as Self-supervised learning and Weakly supervised learning can be used. This would greatly reduce the amount of work required if manual slice labeling is done.

To build a more scalable infrastructure for machine learning models (specifically AHT model), cloud Machine Learning frameworks such as SageMaker (by AWS) can be utilized. This would decrease the model load time and would provide much faster results.

Another methods worth looking into are Kubeflow for Machine Learning and TensorFlow Servings.

As part of the ongoing PCH IRB, the above-mentioned strategies would be used to derive the next steps while going forward with the study.

REFERENCES

"A Multi-Platform DICOM Toolbox for Academic Radiologists." Web.

"A.I. Technical – Machine Learning vs. Deep Learning." Edited by Info@lawtomated.com, Lawtomated, 2 Feb. 2020. Web.

"Apnea." ScienceDaily, ScienceDaily, www.sciencedaily.com/terms/apnea.htm. Web.

"Computer-aided Diagnosis." Wikipedia. Wikimedia Foundation, 04 Feb. 2020. Web. 31 Mar. 2020. Web.

"Conda." Conda. Anaconda. Web. 31 Mar. 2020.

"Flask." Pallets. Web. 31 Mar. 2020.

"General Welcome and the Organizing Committee." IEEE Computer Society Opening Remarks and Awards of the RecordBreaking 2019 IEEECVF Conference on Computer Vision and Pattern Recognition CVPR Comments. Web. 31 Mar. 2020.

"Keras: The Python Deep Learning Library." Home - Keras Documentation. Web. 31 Mar. 2020.

"Production-Grade Container Orchestration." Kubernetes. Web. 31 Mar. 2020.

"PyCharm: The Python IDE for Professional Developers by JetBrains." JetBrains. JetBrains. Web. 31 Mar. 2020.

"Pydicom Documentation." Pydicom Documentation - Pydicom 1.4.2 Documentation. Web. 31 Mar. 2020.

"PyTorch." PyTorch, pytorch.org/hub/pytorch_vision_densenet/. Web.

"Retinal Hemorrhages: Abusive Head Trauma or Not?" Pediatric Emergency Care 34.9 (2018): 671-72. Web.

"Scikit-learn." Scikit. Web. 31 Mar. 2020.

"What Are Microservices?" Microservices.io. Web. 31 Mar. 2020.

"What Is Kubernetes - Learn Kubernetes from Basics." LearnITGuide.net, Web. 11 Aug. 2018.

Akkus, Zeynettin, Alfiia Galimzianova, Assaf Hoogi, Daniel Rubin, and Bradley Erickson. "Deep Learning for Brain MRI Segmentation: State of the Art and Future Directions." Journal of Digital Imaging 30.4 (2017): 449-59. Web.

Amagasa, Shunsuke, Hikoro Matsui, Satoshi Tsuji, Satoko Uematsu, Takashi Moriya, and Kosaku Kinoshita. "Characteristics Distinguishing Abusive Head Trauma from Accidental Head Trauma in Infants with Traumatic Intracranial Hemorrhage in Japan." Acute Medicine & Surgery 5.3 (2018): 265-71. Web.

Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. "Dermatologist-level Classification of Skin Cancer with Deep Neural Networks." Nature 542.7639 (2017): 115-118. Web.

Bar, Yaniv, Idit Diamant, Lior Wolf, Sivan Lieberman, Eli Konen, and Hayit Greenspan. "Chest Pathology Detection Using Deep Learning with Non-medical Training." 2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI) 2015 (2015): 294-97. Web.

Bechtel, Kirsten, Kathleen Stoessel, John M Leventhal, Eileen Ogle, Barbara Teague, Sylvia Lavietes, Bruna Banyas, Karin Allen, James Dziura, and Charles Duncan. "Characteristics That Distinguish Accidental from Abusive Injury in Hospitalized Young Children with Head Trauma." Pediatrics 114.1 (2004): 165-68. Web.

Bernstein, David. "Containers and Cloud: From LXC to Docker to Kubernetes." IEEE Cloud Computing 1.3 (2014): 81-84. Web.

Brown, Philip. "What Is the Domain Model in Domain Driven Design?" Culttt. Culttt, 20 Aug. 2018. Web.

Brownlee, Jason. "A Gentle Introduction to Computer Vision." Machine Learning Mastery., 05 July 2019. Web. 31 Mar. 2020.

Cer, Daniel, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. "Universal Sentence Encoder." (2018). Web.

Chen, Sihong, Kai Ma, and Yefeng Zheng. "Med3D: Transfer Learning for 3D Medical Image Analysis." (2019). Web.

Christian, Cindy W, Robert Block, Carole Jenny, Crawford, James, Flaherty, Emalee, Hibbard, Roberta A, Kaplan, Rich, Corwin, David L, Saul, Janet, and Hurley, Tammy Piazza. "Abusive Head Trauma in Infants and Children." Pediatrics 123.5 (2009): 1409-411. Web.

Cowley, Laura Elizabeth, Charlotte Bethan Morris, Sabine Ann Maguire, Daniel Mark Farewell, and Alison Mary Kemp. “Validation of a Prediction Tool for Abusive Head Trauma.” Pediatrics 136.2 (2015): 290-98. Web.

Craven, Connor. “What Is Docker Container.” Sdxcentral. Aug. 2019. Web. Mar. 2020.

Currie, Janet, and Cathy Spatz Widom. “Long-Term Consequences of Child Abuse and Neglect on Adult Economic Well-Being.” Child Maltreatment 15.2 (2010): 111-20. Web.

Damashek, Amy, Melanie Mcdiarmid Nelson, and Barbara L Bonner. “Fatal Child Maltreatment: Characteristics of Deaths from Physical Abuse versus Neglect.” Child Abuse & Neglect 37.10 (2013): 735-44. Web.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” (2018). Web.

Fanconi, Manuela, and Ulrich Lips. “Shaken Baby Syndrome in Switzerland: Results of a Prospective Follow-up Study, 2002–2007.” European Journal of Pediatrics 169.8 (2010): 1023-028. Web.

Feldman, K W, R. Bethel, R P Shugerman, D C Grossman, M S Grady, and R G Ellenbogen. “The Cause of Infant and Toddler Subdural Hemorrhage: A Prospective Study.” Pediatrics 108.3 (2001): 636-46. Web.

Francesco Ciompi, Kaman Chung, Sarah J. Van Riel, Arnaud Arindra Adiyoso Setio, Paul K. Gerke, Colin Jacobs, Ernst Th. Scholten, Cornelia Schaefer-Prokop, Mathilde M. W. Wille, Alfonso Marchianò, Ugo Pastorino, Mathias Prokop, and Bram Van Ginneken. “Towards Automatic Pulmonary Nodule Management in Lung Cancer Screening with Deep Learning.” Scientific Reports 7.1 (2017): 46479. Web.

Frasier, Lori, Tanya S. Hinds, and Francois M. Luyet. Pediatric Abusive Head Trauma Pocket Atlas. Volume Two, Medical Mimics. 2016. Web

Glaser, Danya. "Child Abuse and Neglect and the BrainA Review." Journal of Child Psychology and Psychiatry 41.1 (2000): 97-116. Web.

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative Adversarial Networks.” (2014). Web.

Grewal, Monika, Muktabh Mayank Srivastava, Pulkit Kumar, and Srikrishna Varadarajan. “RADNET: Radiologist Level Accuracy Using Deep Learning for HEMORRHAGE Detection in CT Scans.” (2017). Web.

Grzybek, Kamil. “Attributes of Clean Domain Model.” Kamil Grzybek, Web. 28 Oct. 2019.

Herman, Bruce E., Kathi L. Makoroff, and Howard M. Corneli. “Abusive Head Trauma.” Pediatric Emergency Care 27.1 (2011): 65-69. Web.

Hettler, Joeli, and David S Greenes. “Can the Initial History Predict Whether a Child with a Head Injury Has Been Abused?” Pediatrics 111.3 (2003): 602-07. Web.

Hobbs, Childs, Wynne, Livingston, Seal, and Hobbs, C. “Subdural Haematoma and Effusion in Infancy: An Epidemiological Study.” Archives of Disease in Childhood 90.9 (2005): 952-55. Web.

Huang, Gao Q., Kilian Q. Weinberger, Zhuang Liu, and Laurens Van Der Maaten. “Densely Connected Convolutional Networks.” Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 2017 (2017): 2261-269. Web.

Huynh, Benjamin Q, Hui Li, and Maryellen L Giger. “Digital Mammographic Tumor Classification Using Transfer Learning from Deep Convolutional Neural Networks.” Journal of Medical Imaging (Bellingham, Wash.) 3.3 (2016): 034501. Web.

Hymel, Kent P, Veronica Armijo-Garcia, Robin Foster, Terra N Frazier, Michael Stoiko, LeeAnn M Christie, Nancy S Harper, Kerri Weeks, Christopher L Carroll, Phil Hyden, Andrew Sirotnak, Edward Truemper, Amy E Ornstein, Ming Wang, Herman, Bruce E, Narang, Sandeep K, Graf, Jeanine M, Dias, Mark, Pullen, Deborah A, and Boos, Stephen C. “Validation of a Clinical Prediction Rule for Pediatric Abusive Head Trauma.” Pediatrics 134.6 (2014): E1537-1544. Web.

“IBM Watson Cognitive Service for Visual Recognition.” All Answers Ltd. allanswers.co.uk, November 2018. Web. 16 April 2020.

Jenny, C., Kp Hymel, A. Ritzen, Se Reinert, and TC Hay. “Analysis of Missed Cases of Abusive Head Trauma.” Jama-Journal Of The American Medical Association 281.7 (1999): 621-26. Web.

Jia Deng, R., Wei Dong, Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “ImageNet: A Large-scale Hierarchical Image Database.” 2009 IEEE Conference on Computer Vision and Pattern Recognition (2009): 248-55. Web.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2016 (2016): 770-78. Web.

Karras, Tero, Samuli Laine, and Timo Aila. “A Style-Based Generator Architecture for Generative Adversarial Networks.” (2018). Web.

Keenan, Runyan, Marshall, Nocera, Merten, Sinal, and Keenan, H. “A Population-Based Study of Inflicted Traumatic Brain Injury in Young Children.” Journal of the American Medical Association 290.5 (2003): 621-26. Web.

Kemp, A M, T. Jaspan, J. Griffiths, N. Stoodley, M K Mann, V. Tempest, and S A Maguire. “Neuroimaging: What Neuroradiological Features Distinguish Abusive from Non-abusive Head Trauma? A Systematic Review.” Archives of Disease in Childhood 96.12 (2011): 1103-11012. Web.

Khomh, Foutse, Bram Adams, Jinghui Cheng, Marios Fokaefs, and Giuliano Antoniol. “Software Engineering for Machine-Learning Applications: The Road Ahead.” IEEE Software 35.5 (2018): 81-84. Web.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” Communications of the ACM 60.6 (2017): 84-90. Web.

Kuo, Weicheng, Christian Häne, Pratik Mukherjee, Jitendra Malik, and Esther L Yuh. “Expert-level Detection of Acute Intracranial Hemorrhage on Head Computed Tomography Using Deep Learning.” Proceedings of the National Academy of Sciences of the United States of America 116.45 (2019): 22737-2745. Web.

Lakhani, Paras, and Baskaran Sundaram. “Deep Learning at Chest Radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks.” Radiology 284.2 (2017): 574-82. Web.

Lecun, Y., Bottou, Bengio, and Haffner. “Gradient-based Learning Applied to Document Recognition.” Proceedings of the IEEE 86.11 (1998): 2278-324. Web.

Letson, Megan M, Jennifer N Cooper, Katherine J Deans, Philip V Scribano, Kathi L Makoroff, Kenneth W Feldman, and Rachel P Berger. “Prior Opportunities to Identify Abuse in Children with Abusive Head Trauma.” Child Abuse & Neglect 60 (2016): 36-45. Web.

Leventhal, John, M. Asnes, Andrea Pavlovic, and G. Moles. “Diagnosing Abusive Head Trauma: The Challenges Faced by Clinicians.” Pediatric Radiology 44.Supplement 4 (2014): 537-42. Web.

Levin, Alex V., Cindy W. Christian, Carole Jenny, James E. Crawford-Jakublak, Emaiee Fiaherty, Roberta A. Hibbard, and Rich Kaplan. “Clinical Report-The Eye Examination in the Evaluation of Child Abuse.” Pediatrics 126.2 (2010): 376-80. Web.

Marktab. “Create a CI/CD Pipeline with Azure Pipelines - Team Data Science Process.” Create a CI/CD Pipeline with Azure Pipelines - Team Data Science Process Microsoft Docs, Microsoft. Web.

Matthews, Kayla. “6 Reasons Why Python Is Suddenly Super Popular.” KDnuggets. July 2017. Web. 31 Mar. 2020.

Mcbee, Morgan P, Omer A Awan, Andrew T Colucci, Comeron W Ghobadi, Nadja Kadom, Akash P Kansagra, Srini Tridandapani, and William F Auffermann. “Deep Learning in Radiology.” Academic Radiology 25.11 (2018): 1472-480. Web.

Mcluckie, C. “Containers, VMs, Kubernetes and VMware.” Google Cloud Platform Blog. 25 Aug. 2014. Web. 31 Mar. 2020.

Moeskops, Pim, Max A Viergever, Adrienne M Mendrik, Linda S De Vries, Manon J. N. L Benders, and Ivana Isgum. “Automatic Segmentation of MR Brain Images With a Convolutional Neural Network.” IEEE Transactions on Medical Imaging 35.5 (2016): 1252-261. Web.

Mouat, Adrian, et al. “Are Containers Replacing Virtual Machines?” Docker Blog, Web. 1 Sept. 2018.

Murphy, Andrew. “Windowing (CT): Radiology Reference Article.” Radiopaedia Blog RSS. Web. 31 Mar. 2020.

Nibali, Aiden, Zhen He, and Dennis Wollersheim. “Pulmonary Nodule Classification with Deep Residual Networks.” International Journal of Computer Assisted Radiology and Surgery 12.10 (2017): 1799-808. Web.

Palusci, Vincent J, and Theresa M Covington. “Child Maltreatment Deaths in the U.S. National Child Death Review Case Reporting System.” Child Abuse & Neglect 38.1 (2014): 25-36. Web.

Paul, Alexandra R, and Matthew A Adamo. “Non-accidental Trauma in Pediatric Patients: A Review of Epidemiology, Pathophysiology, Diagnosis and Treatment.” Translational Pediatrics 3.3 (2014): 195-207. Web.

Piteau, Shalea J., Michelle G. K. Ward, Nick J. Barrowman, and Amy C. Plint. “Clinical and Radiographic Characteristics Associated With Abusive and Nonabusive Head Trauma: A Systematic Review.” Pediatrics 130.2 (2012): 315-23. Web.

Prabhu, Raghav. “Understanding of Convolutional Neural Network (CNN) - Deep Learning.” Medium, Medium, Web. 21 Nov. 2019.

Rademacher, Florian, Sabine Sachweh, and Albert Zündorf. “Towards a UML Profile for Domain-driven Design of Microservice Architectures.” Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10729 (2018): 230-45. Web.

Radford, Alec, Luke Metz, and Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.” (2015). Web.

Reading, Richard. “Which Clinical Features Distinguish Inflicted from Non-Inflicted Brain Injury? A Systematic Review.” Child: Care, Health and Development 36.1 (2010): 150-51. Web.

Reece, Robert M, and Robert Sege. “Childhood Head Injuries: Accidental or Inflicted?” Archives of Pediatrics & Adolescent Medicine 154.1 (2000): 11-15. Web.

Reppic. “Gradient & Sigmoid Windowing.” Kaggle. Kaggle, 23 Oct. 2019. Web. 31 Mar. 2020.

Rolfes, Mary, Julie Guerin, Peter Kalina, and Justin Brucker. "Neuroimaging of Pediatric Abusive Head Trauma." Applied Radiology 48.3 (2019): 30-38. Web.

Runyan, Desmond K. “The Challenges of Assessing the Incidence of Inflicted Traumatic Brain Injury: A World Perspective.” American Journal of Preventive Medicine 34.4 (2008): S112-115. Web.

Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks-the ELI5 Way.” Medium, Towards Data Science, Web. 17 Dec. 2018.

Salazar, Misael F. Garza. Hematomas: Types, Treatments and Health Risks. 2012. Recent Advances in Hematology Research. Web.

Sanche, Daniel. “Kubernetes 101: Pods, Nodes, Containers, and Clusters.” Medium. Google Cloud - Community, 14 May 2018. Web. 31 Mar. 2020.

Sarkar. “A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning.” Medium, Towards Data Science, Web. 17 Nov. 2018.

Sarnaik, Ajit, Nikki Ferguson, Miller O’Meara, Am Agrawal, Iqbal Deep, Shruti Buttram, Akash Bell, Sandra Wisniewski, Michael Luther, J. Hartman, and Stephen Vavilala. “Age and Mortality in Pediatric Severe Traumatic Brain Injury: Results from an International Study.” Neurocritical Care 28.3 (2018): 302-13. Web.

Sieswerda-Hoogendoorn, Tessa, Boos, Stephen, Spivack, Betty, Bilo, Rob A. C., Van Rijn, Rick R., General Paediatrics, Other Research, and Radiology Nuclear Medicine. “Educational Paper Abusive Head Trauma Part II: Radiological Aspects.” European Journal of Pediatrics 171.4 (2012): 617-23. Web.

Springer, Kristen W, Jennifer Sheridan, Daphne Kuo, and Molly Carnes. “Long-term Physical and Mental Health Consequences of Childhood Physical Abuse: Results from a

Large Population-based Sample of Men and Women.” Child Abuse & Neglect 31.5 (2007): 517-30. Web.

Suh, Sungho, Haebom Lee, Jo Jun, and Paul Lukowicz. “Generative Oversampling Method for Imbalanced Data on Bearing Fault Detection and Diagnosis.” Applied Sciences 9.4 (2019): Applied Sciences, 2019, Vol.9(4). Web.

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the Inception Architecture for Computer Vision.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2016 (2016): 2818-826. Web.

Talvik, Inga, Alexander, Randell C, and Talvik, Tiina. “Shaken Baby Syndrome and a Baby's Cry.” Acta Paediatrica (Oslo, Norway : 1992) 97.6 (2008): 782-85. Web.

Tan, Chuanqi, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. “A Survey on Deep Transfer Learning.” Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11141 (2018): 270-79. Web.

Vieira, Sandra, Walter H.L Pinaya, and Andrea Mechelli. “Using Deep Learning to Investigate the Neuroimaging Correlates of Psychiatric and Neurological Disorders: Methods and Applications.” Neuroscience and Biobehavioral Reviews 74.Pt A (2017): 58-75. Web.

Zhou, Zongwei M., Vatsal B. Sodha, Md Mahfuzur Rahman Siddiquee, Ruibin Feng, Nima Tajbakhsh, Jianming Liang, and Michael B. Gotway. “Models Genesis: Generic Autodidactic Models for 3d Medical Image Analysis.” Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11767 (2019): 384-93. Web.