London South Bank University

Deep learning research and application of Brain MRI based Convolutional Neural Network for detecting Psychosis

Sheikh Khaled Ahmed

Student ID: 3605631

BSc(Hons) Computer Science

Division of Computer Science and Informatics

Submission date: 11/05/2022

This dissertation is my own original work and has not been submitted elsewhere in fulfilment of the requirements of this or any other award. Any passages taken from my own previous work or other people's work have been quoted and acknowledged by clear referencing to author, source and page(s). Any non-original illustrations are also referenced. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree as a whole.

-Sheikh Khaled Ahmed

Abstract

The recent decade has seen tremendous growth in deep learning architectures as well as the diagnosis of mental health diseases. As a result of technology breakthroughs, many theoretical AI principles are currently being put into practice. Furthermore, AI has gained traction in the medical field, leading research to shift its focus. The implications of Convolutional Neural Networks (CNN) for diagnosing mental health problems such as psychosis will be the emphasis of this paper. A psychologist takes an average of 3-6 years to train, therefore investing in AI to handle these jobs could be a more cost-effective alternative to a quickly available support system. As a result, it is vital to explore and critically evaluate using CNN for diagnosing Psychosis.

The vast majority of this paper will consist of theoretical research on various technologies and advances in the field of matter. A brain MRI dataset will also be used in the development of the described model. The accuracy of different CNN architectures will be tested through experimentation. The models will be evaluated in terms of training and validation loss and a discussion of how various techniques could be used to improve the model further.

Acknowledgements

First and foremost, all praise to Allah, the almighty, the most gracious and most merciful for giving me the confidence and helping me manage the stress of complete this thesis. May Allah's blessing be sent to the final Prophet Muhammad (Peace be upon him), his family and companions

I would like to express my gratitude to my supervisor Dr. Enrico Grisan for providing this topic as an opportunity. His guidance throughout this project has helped me achieve my objectives and his research on medical imaging has been an inspiration.

I must also thank Dr. Bugra Alkan, for teaching me the fundamentals or Artificial Intelligence and being available for discussions. Furthermore, I would like to extend this thanks to all the faculty of engineering staff who have taught me over the course of my studies at LSBU. I could not have completed this project without the information they provided.

Table of Contents

Table of Figures

Table of Figures

Chapter 1: Introduction

1.1 Introduction

Artificial Intelligence has grown tremendously and is now being applied to many different fields. This study examines the Deep Learning concept and how it might be applied to psychological diagnostic. This introduction will talk about Deep Learning, medical imaging, and psychosis.

Deep Learning is a subfield of Machine Learning that "attempts to mimic the human brain, albeit far from matching its ability, allowing systems to cluster data and make predictions with incredible accuracy." (IBM Cloud Education, 2020). To understand human decision making, it employs a concept known as neural networks or artificial neural networks which further expands on the brain idea by mimicking how neurons work in our brain. There are numerous algorithms available for this purpose, the most popular of which are: Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Long Short-Term Memory Networks (LSTM). These neural networks will be discussed in further details in the technical review.

This dissertation research will focus on the CNN application, which is mainly used for image analysis. It can do different image processing tasks such as classification, segmentation, and object detection. (Wang, et al., 2021). The foundation of CNN is the convolutional layer which gives it the ability to classify our images by taking features from sections of the input image and creating filters to train the algorithm for accurate image detection. The benefit

of using CNN is that it is a form of unsupervised learning which doesn't require human interaction or a ridiculous amount of processing power

1.2 Psychosis

Psychosis is a mental health condition in which a person loses contact with reality (Arciniegas, 2022), making them hear or see things that do not exist. Disturbed speech, hallucination, and delusion are the most common symptoms of psychosis. Although there is no direct cure for psychosis, there are treatments that can help manage the symptoms. These are especially useful for detecting first-episode psychosis (FEP). Generally, the earlier a diagnosis is made, the better the prognosis of the disease.

Psychosis can be diagnosed by observing behaviour, physical exams, and blood tests. It can also be analysed using medical imaging devices such as CT scans or brain MRI; this generates images for which lesions associated with Psychosis can be identified. There are no specific guidelines for identifying Psychosis through MRI for several reasons: the patient may present the illness differently from others. Another reason could be that some overlapping symptoms make it difficult to diagnose. Occasionally, an MRI of the brain may not be able to detect early-stage Psychosis.

1.3 Research motivation

The advancement of technology has resulted in accumulating a large amount of data. Significant progress has been made in the Deep Learning infrastructure over the last ten years, making training data into essential information more feasible. Deep Learning techniques applied to the medical field can help doctors improve their clinical workflow and decision-making process. Current MRI data on patients with mental health conditions can be used to train a CNN to detect any issues. The average time to prepare a psychologist is around 3-6 years. Investing in AI to perform these tasks can be a more cost-effective alternative to an easily accessible support system. Pursuing research on CNN's implication on Psychosis can help build a foundation for future research and development.

1.4 Aims and objectives

This project aims to develop/adapt deep CNN algorithms to a database of brain MRI data from first-episode patients. The MRI data highlights the structural changes in the brain of psychiatric patients. The first objective would be to gain a current perception of how deep learning could be adapted to the medical industry. An understanding must be made by identifying relative achievements, the current technologies, and the existing constraints. Exploration of the MRI dataset will take place to gain a data understanding and subsequently apply any data pre-processing. Then develop the CNN model and identify the optimal hyperparameter for the convolution layer.

1.5 Research approach

This dissertation will research the CNN on mental diagnostics and develop a CNN model through a supervised learning method. There is no current research on the exact specification of this project. Therefore, to develop an understanding, literature will be reviewed by submissions from other universities in the related field. Python will be used in conjunction with libraries such as Keras or TensorFlow to develop the CNN, which will produce the computational graph and model. The methodology approach will follow agile and scrum as some requirements are still unknown.

1.6 Report outline

This report comprises of several chapters that aid in the development of this project. To summarise:

- **Literature review**: Explores the literature on the different topic whilst also analysing the different architecture implemented on convolutional neural network
- **Technical review**: Identifies and examines the essential technologies that are necessary for this project.
- **Methodology**: Discusses the approach that will be taken to achieve the goals of the project. Breaks down the project into several different phases
- **Design**: How the development of the project will be built. Explain each convolutional layer needed for a successful project.

- **Implementation**: The development of the code, including screenshots to explain each section
- **Testing**: Building the actual model and testing.
- **Evaluation**: Evaluating the models that have been built in the testing phase and comparing the accuracy.
- **Conclusion**: Summarising the findings of the research and discussing the objectives of this project.
- **Reflection**: Discussion of my personal journey throughout this project

Chapter 2: Literature review

This chapter presents a literature review of the Deep Convolutional Neural Network (CNN) and its current deployment in medical imaging. The research will be explored within the last ten years due to our epistemic limitations in the technological and medical sector, i.e., Machine learning and medical imaging. Analysis of the literature will be based on the following sections. The first section reviews structural changes in the brain for mental health conditions. The next part explore deep learning methods such the traditional Expert systems and Support Vector Machines, which is then compared to Convolution Neural Network (CNN). The final part analyses the application of CNN for computer vision and its different architectures.

2.0.1 Literature related to the scope of research

Table 1 Literature summary

Article name	***Year***	***Summary***	***Authors***
Deep learning based automatic diagnosis of first-episode psychosis, bipolar disorder, and healthy controls	2021	CNN model that can automatically classify gray matter density images of FEP, BD, and HC.	Zhuangzhuang Li, Wenmei Li, Yan Wei, Guan Gui, Rongrong Zhang, Haiyan Liu, Yuchen Chen, Yiqiu Jiang
Common and distinct structural features of schizophrenia and bipolar disorder: The European Network on Psychosis, Affective disorders, and Cognitive Trajectory (ENPACT) study	2017	Schizophrenia and Bipolar disorder share elements of pathology. The study gains data from those two sets of the group and health controls to analyse Gray matter volume using voxel-based morphometry.	Eleonora Maggioni, Benedicto Crespo-Facorro, Igor Nenadic, Francesco Benedetti, Christian Gaser, Heinrich Sauer
How frequent are radiological abnormalities in patients with psychosis? A review of 1379 MRI scans	2012	Investigates abnormalities in patients with psychosis. This concluded that abnormalities could not be identified using MRI scans.	Iris E. Sommer, Gérard A. P. de Kort, Anne Lotte Meijering, Paola Dazzan,

			Hilleke E. Hulshoff Pol, René S. Kahn, Neeltje E. M. van Haren
Classification of first-episode psychosis in a large cohort of patients using a support vector machine and multiple kernel learning techniques	2015	Uses support vector machine for classification of patients with FEP. This source identified that analysis of temporal Gray matter and frontal lobe could help identify early symptoms of psychosis.	Letizia Squarcina, Umberto Castellani, Marcella Bellani, Cinzia Perlini, Antonio Lasalvia
Machine Learning Approaches: From Theory to Application in Schizophrenia	2013	Explores MRI data for innovative bioinformatics methods to detect human brain features. The source revealed that schizophrenia causes structural changes to MRI data.	Elisa Veronese, Umberto Castellani, Denis Peruzzo, Marcella Bellani, Paolo Brambilla
Psychiatric Disorders Classification with 3D Convolutional Neural Networks	2019	Compare techniques of 2D and 3D CNN models for neuroimaging tasks such as Schizophrenia and Bipolar disorder classification.	Stefano Campese, Ivano Lauriola, Cristina Scarpazza, Giuseppe Sartori Fabio Aiolli
A Comparison of 2D and 3D Convolutional Neural Networks for Hand Gesture Recognition from RGB-D Data	2019	Comparison study of 2D and 3D CNN for detecting hand gestures. The conclusion stated that 2D CNN outperforms 3D CNN model in terms of recall, accuracy, and time in task.	Meghdad Kurmanji Foad Ghaderi
What are artificial neural networks?	2008	Introduction to neural networks, describing its concept, how it works, and the classification tasks it can solve.	Anders Krogh
Application of convolutional neural networks in object detection, re-identification, and recognition	2020	The deployment of CNN in areas for security and surveillance and how well it can perform in object detection	Aburasain, Rua
Application of deep learning in detecting neurological disorders from magnetic resonance images: a survey on the detection of Alzheimer's disease, Parkinson's disease and schizophrenia	2020	Compares DL techniques for detecting neurological disorders from MRI data. Does overview of different DL applications and provides accuracy. Lists CNN architecture that may be used in this research	Manan Binth Taj Noor, Nusrat Zerin Zenia, M Shamim Kaiser, Shamim Al Mamun Mufti Mahmud

2.1 Structural features of psychiatric conditions

Symptoms of Psychosis are commonly associated with various mental health conditions such as schizophrenia (SCZ) or bipolar disorder (BD). “Although both brain disorders share pathological elements, their neural underpinnings are still being researched” (Maggioni, et al., 2017)There are still many unknowns when it comes to identifying associated conditions in the brain.

There are over 300 mental illnesses according to DSM-5 (Diagnostics and Statistical Manual of Mental Disorders). Many neurological studies are being conducted to better understand the structural differences between these disorders. (Maggioni, et al., 2017) investigates the structural similarities and differences between SCZ and BD. The researchers used both region of interest approach and voxel-based morphometry to examine MRI data structural significance and grey matter volume. The data was gathered from a large sample of BD and SCZ patients and healthy controls and will be the same dataset used in this research to build a CNN model.

The study sought to understand the neurobiological relationship between the two mental disorders, concluding that they both have shared and distinct neuroanatomical features. Nine years ago, none of the neuropathological findings observed in the patients was interpreted as a possible substrate for organic psychosis (Sommer, et al., 2012). Presently, It is theoretically possible associate the difference between SCZ and BD. The significant advances in research and technology have increased the validity of radiological findings in

Psychosis and other mental diseases. With more structured guidelines for defining psychosis, it becomes more feasible to integrate the data into a Deep Learning algorithm such as the convolutional neural network.

2.2 AI versus psychiatrist

(Shen, et al., 2019) studies a range of papers to compare AI and physician diagnostic performance Because AI is still a relatively new invention to medical professionals and policymakers, its judgement might be concerning. To address their worries, a critical review of the data might assist them in better understanding the importance of AI. The source looks at nine separate publications that compare artificial intelligence and physicians in various medical sectors. According to the findings, AI performed on par with physicians and somewhat better than early-career clinicians. The site does not focus on a single medical profession, but rather gives generic data in a variety of fields. Furthermore, the source utilised sources from 2000 to 2019, indicating a substantial gap in AI progress and potentially affecting the veracity of his conclusions.

2.3 Expert Systems

(Saibene, et al., 2021)discusses the aim of expert systems, as well as their advantages and disadvantages in the medical area An expert system is a knowledge-based system that replicates human decision-making. The capacity of medical expert systems to aid clinicians, as well as their availability, contribute to a greater quality of life. The paper evaluates the benefits of ES by highlighting greater dependability, decreased error and

costs, a well-organized knowledge base, and simpler access to expert information. The paper expressed worry about the difficulty of constructing a database owing to the variety of the data source, as well as the unclear explanations of the decision-making process provided by experts.

2.4 Support Vector Machines (SVM)

The classification of psychiatric disorders using Machine Learning techniques has rapidly evolved over the last decade. Because of technological advancements, many theoretical AI concepts are now being put into practice. The analysis of neuroimaging data has been increasingly accurate in the aspect that it can now be integrated into a supervised learning model (Veronese, et al., 2013). There are numerous factors to consider when building models with MRI data and incorporating them into an ML algorithm. The research article, "Classification of first-episode psychosis in a large cohort of patients using support vector machine and multiple kernel learning techniques" (Squarcina, et al., 2015), discusses how heterogeneity affects the results, so they wanted to take age and gender into account when doing the classification. The article analyses brain structural patterns with the "FreeSurfer" software package, which employs machine learning models and support vector machines (SVM) to detect early stages of psychosis.

A Support Vector Machine (SVM) is a supervised learning model used for regression and classification tasks. It finds the hyperplane in an N dimension to dichotomies the data points. SVM application has been applied in several real-life problems such as image classification, facial detection, bioinformatics

and, as discussed in the literature reviews, the classification of psychiatric disorders. Using an SVM is its effectiveness in high dimension spaces and its memory efficiency and versatility when defining the kernel functions.

An SVM was used to examine the cortical thickness by taking into account age and gender. This was successfully able to distinguish FEP from HC. Mental disorders can be discovered using MRI data using SVM and MKL by identifying frontal and temporal grey matter characteristics. Even though SVM models provide excellent accuracy, it still falls slightly short compared to CNN. SVM uses only a subset of the dataset, following in a faster training time. In contrast, CNN processes the entirety of the dataset resulting in a longer processing time but making up for it with its learning capabilities. Furthermore, Image recognition tasks have been shifting onto CNN as a practical improvement to recall, accuracy and time in task.

2.5 2D and 3D CNN

CNN has been proven and widely used to solve neurological pathologies. It has been used to classify a variety of psychiatric illnesses. However, the traditional CNN has limitations due to the algorithm’s inability to manage the entire 3D structure of the brain. To address the aforementioned issues, (Campese, et al., 2019) proposed a 3D CNN and tested it to compare it to other algorithms. Campese concluded that the results for a 3D CNN have been quite favourable. This opens a debate on whether this research should pursue a 3D or 2D CNN.

Clinical examination is still the gold standard for detecting brain problems. Even though Deep Learning has received a great deal of attention for its visual recognition task. A disadvantage of using a standard SVM or 2D CNN model is that it only examines the intensity value of each voxel rather than the position, resulting in a significant loss of information. Compared to the 3D CNN, the literature shows that working on the brain's 3D structure outperforms the other models because it can correctly capture spatial information. However, the training technique becomes significantly more expensive.

With a 3D dataset, it becomes possible to validate the parameters for the homogeneity and spatial positions, which will increase accuracy. Although 3D-CNN has a structural advantage in visual recognition tasks for 3D images or video, 2D-CNN can also outperform 3D in the recall, accuracy, and time in task (Kurmanji & Ghaderi, 2019). A 3D-CNN is computationally expensive, prone to over-fitting and memory intensive. Brain MRIs are 3D images, so for this research, it becomes functional to explore the 2D ConvNet and 3D ConvNet.

2.6 ConvNet architectures

The invention of ConvNet has led to many different architectures of the model. Developers of CNN models are always looking for ways to improve their models. Fig.1 shows the recent boom in the most popular architecture

development. The main architecture that will be discussed in this section are AlexNet, VGG16 and ResNet

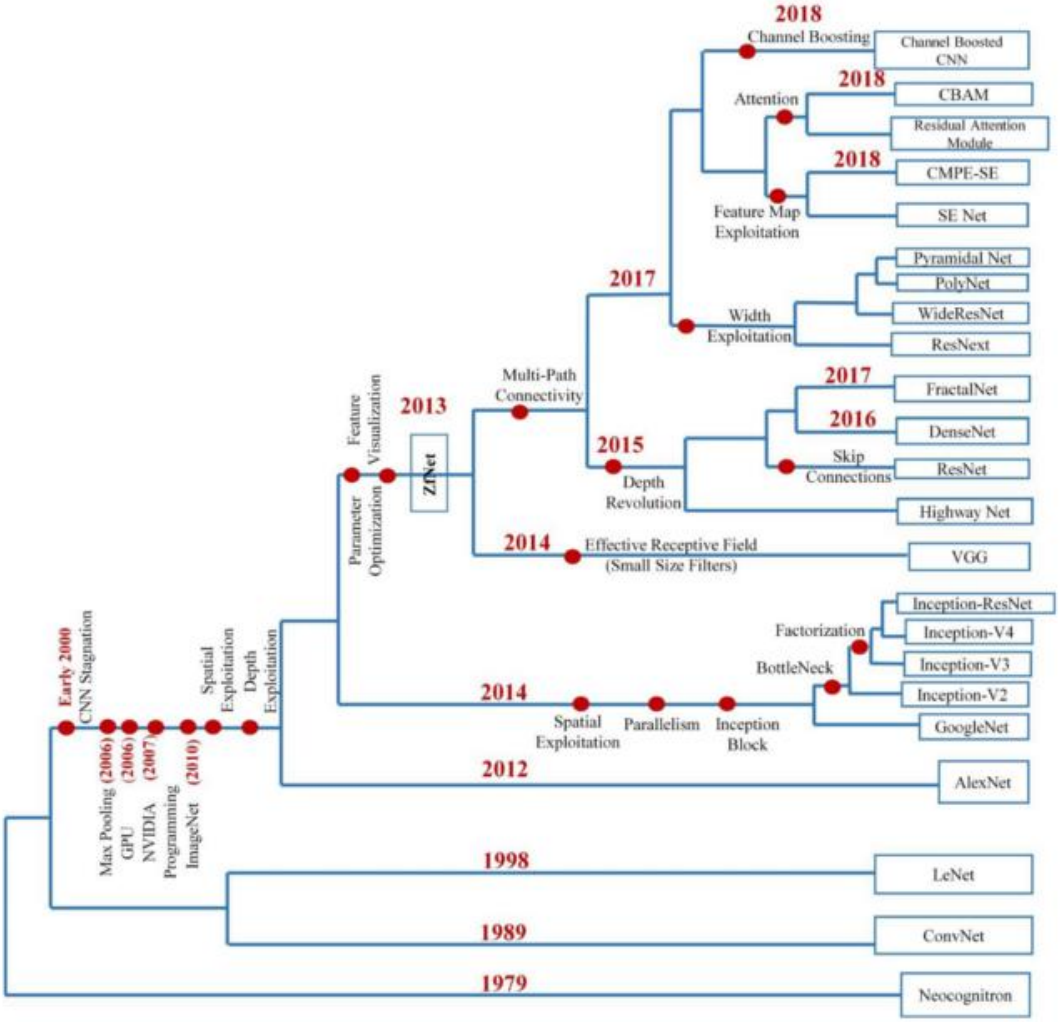

Figure 1 CNN architectures over the years (Dang, 2021)

2.6.1 AlexNet

AlexNet was among the first architecture to gain popularity after the CNN stagnation period. This architecture was developed by Alex Krizhevesky in 2012. The model was proposed to solve the ImageNet challenge or also known as The ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which is a large dataset contain over 15 million images and over 1000 classes. This model proceeded to win with an accuracy of 84.7% and resulted in a

“significant breakthrough in the field of machine learning and computer vision for visual recognition and classification tasks” (Alom, et al., 2018), which then sparked interest in deep learning development. The network consists of five ConvNet layers that also perform Local Response Normalization and max pooling. Then two fully connected layer is used in conjunction with dropout, followed by a SoftMax.

2.6.2 VGG16

VGG16 is another CNN architecture proposed K. Simonyan and A. Zisserman from the University of Oxford in 2014. The model was designed as an improvement to AlexNet by replacing the large Kernel sized filters. As the name suggests it is 16 layers deep and has a total of 138 million parameters. VGG16 proceeded to be trained on the ImageNet dataset, resulting in an accuracy of 92.7%. This enhancement to the AlexNet achieved a better accuracy but would longer to process. AlexNet took 6 simultaneous days to train on the GTX 580 whilst VGG16 would take 2-3 week on the Nvidia Titan. Even so, improvements are always being made to the architecture as the “degradation problem occurring due to compression.” (Qassim, et al., 2018) which utilises residual learning, a method for training neural networks by mapping shortcut connections. This project will aim to implement a simplified version of VGG16 to experiments its capabilities.

2.6.3 ResNet

Residual Network (ResNet) was first described in the paper “Deep Residual Learning for Image Recognition” (He, et al., 2015). This was ground-breaking

work that made it possible to “train hundreds or even thousands of layers and still achieve compelling performance” (Feng, 2017) . Since the release of AlexNet there have been many different architectures presented.

The general consensus is that the network needs to go deeper. However, the network will always be prone to overfitting. In addition, the problem with deep networks is its vanishing gradient problem, where the deeper the network goes the more saturated the performance becomes. There have been solutions to tackling this issue such as adding ReLU activation function or auxiliary loss in the middle layer, but none eliminates this problem. The whole concept of ResNet is to tackle this issue with its *identity shortcut connection* which sets a link between distant layers without involving anything in between. This architecture could be implemented into this project but would be excessive as there is not enough data.

Chapter 3: Technical review

This section of the report will explore the different technology available to accomplish the goals of this project. The main discussion will be on neural networks, python, and its libraries.

3.1 Artificial Neural Networks

Neural networks (NN) are computer systems made up of interconnected nodes. It attempts to mimic the decision-making process caused by the neurons in our brain. To perform computation tasks such as cluster analysis and classification, they can take a raw input of data and “perform sophisticated pattern recognition” (Krogh, 2008). There are several types of NN; this section will review Recurrent Neural Networks, Feedforward Neural networks, and Convolutional Neural networks.

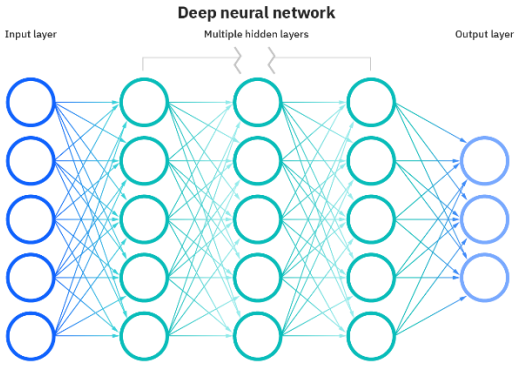

Figure 2 NN model by IBM

3.1.1 Recurrent Neural Networks (RNN)

RNN utilises a recurrency technique. It feeds the input back into the hidden layer to guarantee that the sequential data is collected. This algorithm only takes sequential or time-series data as an input and is commonly used for

ordinal or temporal problems, whereas this research will rely on the classification of images. The main advantages to RNN are its recurrent feed of information and the fact that each neuron retains its memory. The primary application to RNN is usually based on speech recognition, time series or sentiment analysis. A common issue with RNN is that it can suffer from the vanishing and exploding gradient.

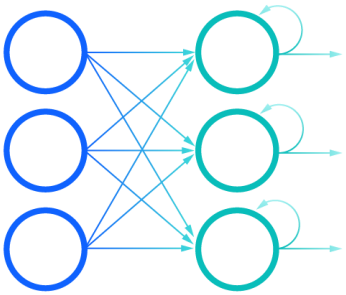

Figure 3 RNN model by IBM

3.1.2 Feedforward Neural Network (FNN)

FNN is the simplest type of NN; it is essentially like the RNN but without recurrency. Every neuron is linked to every perceptron in the following layer, and the data is just fed forwards into the algorithm. The advantage of FNN is its ability to learn any nonlinear functions. It is among the easiest NN and has excellent responsiveness to noisy data. The problem with this is that it can only take input from 1-dimensional vectors, making image classification (a 2D or 3D vector) much more difficult and completely rejecting the options to use it for this research.

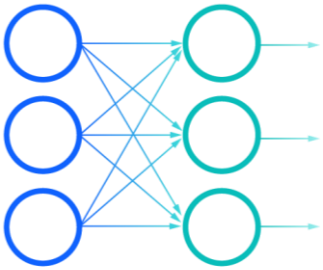

Figure 4 FNN model by IB

3.1.3 Convolutional Neural Network (CNN)

CNN has gained immense popularity in recent years because of its image classification and object detection capabilities. It has three types of layers: convolution, pooling and fully connected. The Convolution layer oversees the extraction of image features to use in the form of multiple filters. The pooling layer is used for dimension reduction after the process of the convolution layer (Aburasain, 2020). The fully connected layer is the layer that comes before the network's final classification output; It's a vector that combines all the pixels from the previous layer into one.

CNN's key advantage is that it excels in visualisation tasks; it can learn the features from input data, allowing for fast and accurate categorisation. The drawback to using a CNN is that it becomes significantly slower when using the maximum pooling operation. The maximum pooling calculates the maximum value for each vector of the image. The CNN needs a large dataset and uses multiple layers to train and process the data. This will take a long time if the computer does not have good hardware capabilities.

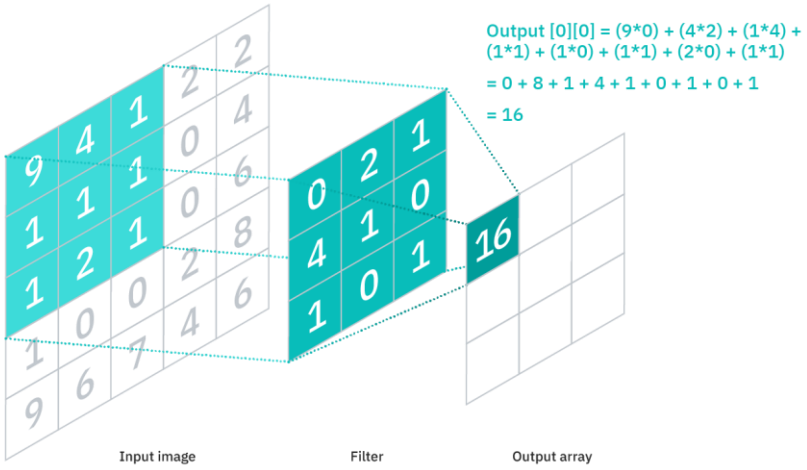

Figure 5 CNN model by IBM

3.2 Programming languages

There are many different programming languages and quite a few support machine learning capabilities. This subsection will review some of the best programming languages for Machine Learning: Python, Scala, and R.

3.2.1 Python

Python's popularity over the last decade has grown exponentially. Its core attraction was based on its simplicity and readability. Since then, it has gained enormous support and a plethora of libraries. Some of the most popular ML libraries are built on Python, increasing its significance for this research. Some ML libraries include: TensorFlow, Keras, and PyTorch. The advantage of using Python is its support and likeliness to the English language, making it easier to understand and code. The disadvantage to using Python is its speed compared to high-performance languages such as C or C++.

3.2.2 Scala

Scala is a programming language based on the criticism of Java. It eliminates all the flaws of Java and allows you to build object-orientated and functional programming in one concise high-level language. Scala has grown in popularity over the recent years for its scalable machine learning and extensive data capabilities. The advantage of using Scala is that it is compatible with Java's massive collection of libraries. In comparison to Python, Scala is almost ten times faster. (Sharma, 2021) but lacks the necessary support.

3.2.3 R

R is a programming language adopted by researchers and statisticians for data analysis and visualisation. It is based on Fortran and C, making it favoured by researchers and giving it a highly active community. The advantage of using R is connecting with another language, analysis capabilities, and data exploration in statistical models. R may be the best choice for creating a rapid machine learning prototype. In contrast, Python does seem to perform better in data manipulation and machine learning tasks with its robust libraries.

3.3 Libraries

3.3.1 NumPy

The NumPy library is used to perform mathematical operations. It offers powerful data structure and can also utilises high-level functions to operate on

matrices and arrays. This will be needed for this research to handle multi-dimensional arrays of an MRI.

3.3.2 Pandas

The Pandas library is used for data manipulation and analysis. This is essential for importing data into the code and will be a fundamental library within this project. With panda you can develop the data structure and perform basic analysis.

3.3.3 Matplotlib

Matplotlib is a plotting library that allows interactive visualisations. The library will be useful for this research to analyse the data and create models for evaluations. This also works in conjunction with NumPy, to enhance mathematical operations.

3.3.4 Scikit-learn

The Scikit library is popular for its machine learning capabilities. It allows various classification, clustering, and regression algorithm. This library will be used to help prepare the data into training and validation partitions.

3.4 Machine Learning libraries

ML libraries are tools built into the programming language; this includes standard learning algorithms to allow classification and regression tasks. This section will review the Python libraries: TensorFlow, Keras and PyTorch. A quick comparison of the three libraries is shown in table 2.

Table 2 Comparison of ML libraries (Terra, 2022)

	Keras	**PyTorch**	**TensorFlow**
API Level	High	Low	High and Low
Architecture	Simple, concise, readable	Complex, less readable	Not easy to use
Datasets	Smaller datasets	Large datasets, high performance	Large datasets, high performance
Debugging	Simple network, so debugging is not often needed	Good debugging capabilities	Difficult to conduct debugging
Does It Have Trained Models?	Yes	Yes	Yes
Popularity	Most popular	Third most popular	Second most popular
Speed	Slow, low performance	Fast, high-performance	Fast, high-performance
Written In	Python	Lua	C++, CUDA, Python

3.4.1 TensorFlow

TensorFlow is an open-source platform which supports ML capabilities. It produces a visual representation of your data, making it a practical approach to debugging. This can be especially helpful for this project when developing the CNN model from scratch. The issue with TensorFlow is its lack of features on the Windows OS, as contrasted to Linux OS. The advantage of using TensorFlow is its capability to take in a large dataset and produce high performance. When stacked alongside other libraries, TensorFlow triumphs in functionality and object detection

3.4.2 Keras

Keras is a library built upon TensorFlow, using the Python language. It utilises a high-level API architecture, meaning the functions will be simpler and more

generic than TensorFlow's low-level API. The upside to building on Keras is its low-level API, making it easy for beginners to learn and allowing a fast deployment of a neural network. The downside to using Keras and its low-level API is its simple functions may not be edited to your exact requirement; this also can lead to low-level backend errors and become difficult to debug. The great thing about Keras is that it can work in conjunction with TensorFlow allowing certain functions to integrate into the network. This library would be most useful when working with a small dataset and need to deploy a quick prototype.

3.4.3 PyTorch

PyTorch is another library that is simpler to learn and code. A comprehensive set of robust APIs for extending the PyTorch Libraries. It is designed to be more adaptable, optimised and faster. The great thing about PyTorch is its ability to generate computational graphs during runtime, allowing the user to visualise their network and make use of the debugging tools if anything goes wrong. The drawback to using this is that it's one of the newer libraries. PyTorch was released in 2016, and compared to Keras and TensorFlow, it has a smaller developer community.

3.5 Technical discussion

After reviewing the different technologies used for this project, this section will provide supportive reasoning for the choice of technology. The focus of this project will be on the Convolutional Neural Network; this is because CNN is

the only neural network with potential for image classification tasks. It has the highest accuracy amongst all the deep learning algorithm that uses image dataset. Compared to SVM, it is easier to understand and can allow a fast deployment.

The leading choice of language would be Python; this is because of the personal familiarity I have for using this language. In addition, Python has exceptional development in Machine Learning. Some of the most popular ML libraries have been developed for Python.

The library that has been considered is PyTorch because of its powerful low-level API. The choice for this library is still subject to change as I have no prior experience using any of these, and I wish to experiment with a different type to see which best meets my requirement.

Chapter 4: Methodology

This project will follow the Agile Scrum methodology. The approach allows flexible management, which will help cover undefined requirements by breaking down tasks into sprints. The sprints will help achieve an incremental progression throughout this project. Breaking down the project can allow room for change and makes each task much more accomplishable. Below is a breakdown of the general tasks.

A breakdown of the tasks is shown in the form of a Gantt chart (fig. 6). This will help visualise the general breakdown of tasks and keep track of progress. The green indicates the duration of the project. The red Finally, the blue relates to the documentation for my dissertation and will overlap some of the red tasks.

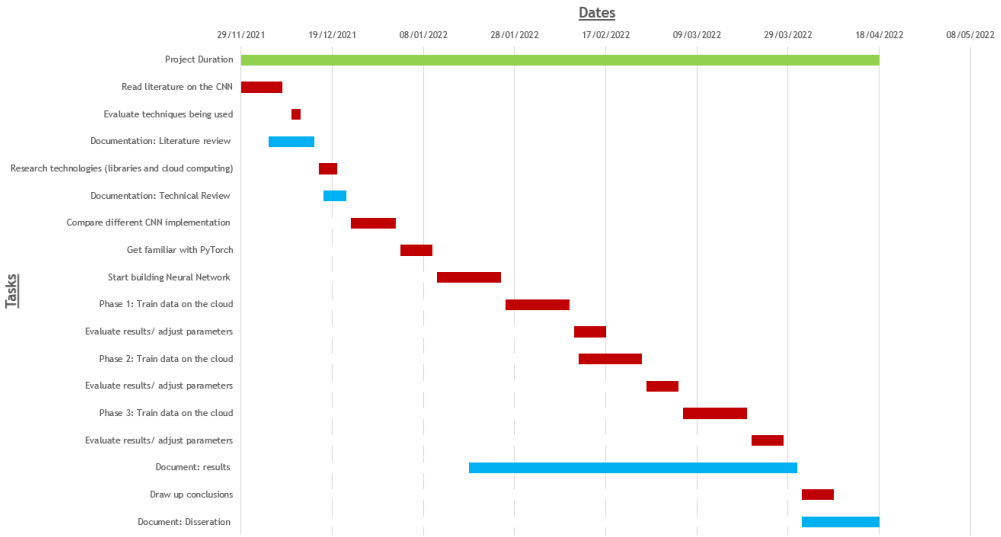

Figure 6 Gantt chart

Jira will be used to manage projects effectively. This enables a visual concept to build a dashboard, populate backlogs, and assign tasks. The tasks specified have been quite vague to appeal to the agile methodology. A bi-weekly sprint cycle will be implemented to meet the project's deadline. The sprints will be

divided into five sections: research, skill development, development, and training.

The research is an essential part of this project. Rather than reinventing the wheel, it is far more advantageous to study the material available on the subject at hand so that it can be used as supporting data to establish a solid foundation for progress. The research phase would focus on broadening the scope of the DL field, including CNN for image classification on psychosis. The sprints have provided enough time for research to gain an understanding of the technologies involved in this project, namely Python.

This project's development phase will be the most time-consuming. For the duration of this sprint, an agile approach will be used to identify an effective method for building a CNN model. Many technical aspects must be considered, such as coding, hyperparameters, and accuracy.

Chapter 5: Design

When developing the model, the CNN architecture must be carefully designed. Fine-tuning the model is critical for achieving greater accuracy and effectiveness. As mentioned in the technical review, A CNN is composed of many different layers. This section will discuss how the model will be implemented and the various layers.

The first step in creating a model is to import the dataset. The dataset we're using is brain MRI images, and the images' details are in a separate excel file. The Pytorch dataset loader will be used to import these files and link them together. After importing the dataset, the next step is to divide it into training and validation sets. Building a reliable Deep Learning model, is a fundamental data science concept. This is done to improve reliability and prevent biased results.

To construct the CNN, the various layers must be built. The layer serves as the input. The input layer will contain image data and the brain MRI dataset. The input will contain the values of each pixel in an image, such as dimensions or colour channel. A brain MRI has a three-dimensional data structure (fig.7). This project may experiment with both 3D and 2D layers to

generate a comparison result. If the 2d layer is used, the three-dimensional matrix must be reshaped into a single value.

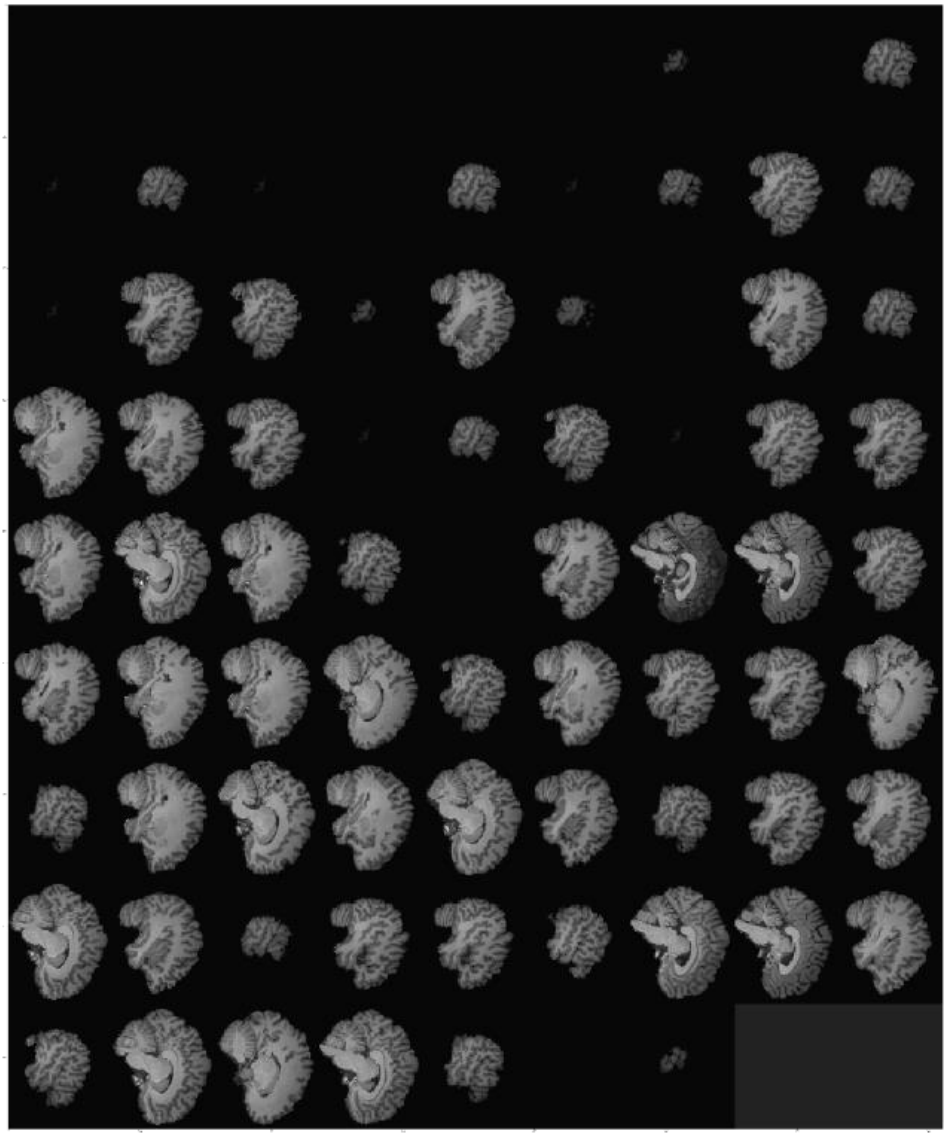

Figure 7 MRI sample

All image transformations are handled by the convolutional layer, also known as the Conv layer. Multiple layers are typically used to extract features from images by computing the dot product of the receptive field. This project aims to use a torch to implement a model based on a 3D convolutional layer. using the following syntax:

*torch. nn.Conv3d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None*)*

This initial model will be built on five layers, with four basic Conv layers and one 3D Conv layer. Depending on computational resources, additional layers can be added to improve performance.

The formula below describes the output results when determining the conv3d layer's input parameters.

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_i, k)$$

The cover layer will also include a ReLU (Rectified Linear Unit) layer, which will be used to convert all negative values to zeros when designing the model to an element-wise activation function. The benefit of this is that it does not activate all neurons simultaneously; if the output is less than 0, the neuron is not activated.

The pooling layer reduces dimensions after each convolution by downsampling the spatial volume. This is necessary to implement because it reduces the already high computational cost. The CNN design will use max poling to reduce the size of each feature map, which will be explicitly applied after the ReLU activation. Max pooling chooses the image's maximum pixel value and is the preferred pooling method over average pooling. This is because average pooling smoothes out the appearance, making it challenging to identify specific features, whereas max pooling selects from the brightest pixel.

This section has gone over the process of creating a CNN model and provided justification for each layer. The initial model proposed will be simple in design to obtain a benchmarking score for further fine-tuning and discovering optimal hyperparameters and layers. A significant amount of time and resources will be invested in developing and testing variants of the models. Using different architectures such as AlexNet, UNett, VGG16, ResNet, etc. Due to time and computational resource constraints, it may not be possible to build the optimal model at this time.

Chapter 6: Implementation

Models are created to interpret data and provide meaningful information to understand it better. CNN has been discussed throughout this report, and it will be the focus of implementation in this section. This section will describe the evolution of the proposed CNN model and identify critical components. The implementation is divided into two parts. Part 1 will use PyTorch to build the CNN model, and Part 2 will move the development to Keras.

The CNN model will be built on Google Colab, a cloud-based service hosted by Jupyter Notebook that allows for in-browser code execution. This is the best option for machine learning because it does not require your personal computational power. Because of the amount of storage and RAM needed to process a 3D Convent, Colab Pro was purchased for this project.

6.1 Part 1: PyTorch

6.1.1 Library:

The first step is to import the required libraries. The libraries exist to help simplify the programming process by inheriting code bundles for specific modules.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import nibabel as nb

from sklearn.model_selection import train_test_split

import torch
from torch.utils.data import Dataset, DataLoader
```

Figure 8 Import libraries

The libraries used in this project will vary and will be imported as needed. The initial libraries, as shown in fig.8, are the most important ones for getting the project started:

- Pandas: A data manipulation and analysis software library. The CSV file associated with the brain MRI dataset will be imported using this.

- Matplotlib: A plotting library that can be used for interactive visualisation and can be used to gain insight into brain MRI data.

- NumPy: A data manipulation library for performing mathematical operations on large datasets with multiple arrays and dimensions.

- Nibabel: Image formatting library that provides full meta data for image data and access to NumPy arrays.

- Sklearn: A library for machine learning. The model selection inheritance allows you to divide the dataset into training and validation segments.

- Torch: Another machine learning library with a diverse set of deep learning algorithms, including neural networks.

6.1.2 Importing datasets

The dataset included images as well as a CSV file containing the associated data. Panda was used to locate the dataset after uploading it to Google Drive.

```
BASE_DIR="/content/drive/MyDrive/brains"
#TRAIN_IMAGES_DIR=os.path.join(BASE_DIR,'train_images')

data_df= pd.read_csv('/content/drive/MyDrive/Classification_clean.csv')

data_df["Class"]=data_df["Diagnosis"]>0
#data_df["Class"].replace(True,2,inplace=True)
#data_df["Class"].replace(False,1,inplace=True)
data_df.Class = data_df.Class.astype('int')


data_df.head()
```

Figure 9 Reading CSV file

By inspecting the file's header (fig.10), you can see that it successfully loads the correct file.

	ID_CF	FirstEpisodes_FEP__HealthyControl_HC_	Diagnosis	Gender	Age	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6	Site 7	Euler	Class
0	CF_001	FEP	3	0	23.857632	0	0	0	0	0	0	0	-21	1
1	CF_002	FEP	3	0	28.952772	0	0	0	0	0	0	0	-33	1
2	CF_003	FEP	1	0	20.199863	0	0	0	0	0	0	0	-55	1
3	CF_004	FEP	1	0	28.517454	0	0	0	0	0	0	0	-26	1
4	CF_005	FEP	2	1	39.101985	0	0	0	0	0	0	0	-24	1

Figure 10 Header

The next step in importing the dataset is to ensure that the images are included. To connect the image path to the CSV file, the Milano MRI Dataset class is created. The code searches for the attribute 'ID CF' and determines whether it matches the image.

```
class Milano_MRI_Dataset(Dataset):
    def __init__(self,df:pd.DataFrame,imfolder:str,train:bool = True, transforms=None):
        self.df=df
        self.imfolder=imfolder
        self.train=train
        self.transforms=transforms

    def __getitem__(self,index):
        filename='w'+self.df.iloc[index]['ID_CF']+'_MRI_sMRI_'+self.df.iloc[index]['ID_CF']+'_brain.nii.gz';

        im_path=os.path.join(self.imfolder,filename)

        nimg = nb.load(im_path)
        x = np.array(nimg.dataobj)
        x=nimg.get_fdata()
        x = transforms.ToTensor()(x)
        x=x.unsqueeze(0).type(torch.FloatTensor);
        m=torch.mean(x)
        s=torch.std(x)
        x=(x-m)/s
        if(self.transforms):
           x=self.transforms(image=x)['image']

        if(self.train):
            y=self.df.iloc[index]['Class']
            return x,y
        else:
            return x

    def __len__(self):
        return len(self.df)
```

Figure 11 Class: Milano_MRI_Dataset

6.1.3 Training and testing

Now it's time to divide the data into training and validation partitions. The Milano MRI Dataset class is loaded and applied to the train loader and valid loader. The test set would use 20% of the total dataset. The training set would use 539 samples, while the validation set would use 135.

6.1.4 CNN model

We can now begin building the CNN model. This is a straightforward implementation that makes use of conv3d. The model will have five layers and a filter size of 3. Filter layers are determined by the input and output channels listed below.

- Layer 1: 1, 32

- Layer 2: 32, 64,
- Layer 3: 64, 128
- Layer 4: 128, 256
- Layer 5: 256, 2048

```
class Simple3DCNN(nn.Module):

    def __init__(self, num_classes):


        super(Simple3DCNN, self).__init__()

        self.conv_layer1 = self._make_conv_layer(1, 32)
        self.conv_layer2 = self._make_conv_layer(32, 64)
        self.conv_layer3 = self._make_conv_layer(64, 128)
        self.conv_layer4 = self._make_conv_layer(128, 256)
        self.conv_layer5=nn.Conv3d(256, 2048, kernel_size=(4, 4, 5), padding=0)

        self.fc5 = nn.Linear(2048, 512)
        self.relu = nn.LeakyReLU()
        self.batch0=nn.BatchNorm1d(512)
        self.drop=nn.Dropout(p=0.25)
        self.fc6 = nn.Linear(512, 256)
        self.relu = nn.LeakyReLU()
        self.batch1=nn.BatchNorm1d(256)

        self.drop=nn.Dropout(p=0.25)
        self.fc7 = nn.Linear(256, num_classes)

    def _make_conv_layer(self, in_c, out_c):
        conv_layer = nn.Sequential(
        nn.Conv3d(in_c, out_c, kernel_size=3, padding=1),
        nn.LeakyReLU(),
        #nn.Conv3d(out_c, out_c, kernel_size=3, padding=1),
        #nn.LeakyReLU(),
        nn.BatchNorm3d(out_c),
        nn.MaxPool3d(kernel_size=2, stride=2),
        )
        return conv_layer
```

Figure 12 CNN model

Other layers are included to optimise the model. Multiple ReLU activation functions are available. 3D Maxpool with stride two and filter size 2. The node has three fully connected layers: (2048, 512), (512, 256), and (256, num classes)—finally, the dropout layer, which guards against overfitting.

6.1.5 Training

The training function is defined after the model is created. This section of code structures the implementation process and aids in model evaluation by printing details of the training and validation partitions for accuracy and loss. The model will run on multiple epochs to help manage computational power and avoid underfitting.

```
def train_model(datasets, dataloaders, model, criterion, optimizer, scheduler, num_epochs, device):

    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs-1))
        print('-' * 10)

        for phase in ['train', 'valid']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0.0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels=labels.to(device)

                # Zero out the grads
                optimizer.zero_grad()

                # Forward
                # Track history in train mode
                with torch.set_grad_enabled(phase == 'train'):
                    model=model.to(device)
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    #loss = criterion(outputs, labels.type(torch.LongTensor).unsqueeze(1).to(device))
                    loss = criterion(outputs, labels.type(torch.LongTensor).to(device))

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
                # Statistics
                running_loss += loss.item()*inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss/len(datasets[phase])
            epoch_acc = running_corrects.double()/len(datasets[phase])

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            if phase == 'valid' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time()-since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    model.load_state_dict(best_model_wts)
    return model
```

Figure 13 Training function

Once the training function is defined, the weights can be for the model, as shown below.

```
class_sample_count = np.array([len(np.where(train_targets == t)[0]) for t in np.unique(train_targets)])
weight = 1. / class_sample_count
class_weight=torch.from_numpy(weight)
class_weight=class_weight.max()/class_weight
class_weight=class_weight/class_weight.max()
class_weight=class_weight.type(torch.FloatTensor)
print(class_weight)
```

Figure 14 Weights

Finally, we can train the model to our specifications. The Adam optimiser will be used for training, which is the recommended algorithm for deep learning models. The criterion is also set to Cross-Entropy Loss. In this case, the number of epochs is set to 10.

6.2 Part 2: Keras

Using the Keras library, this section of the implementation will rebuild the CNN model. This is because the Keras library's syntax is simpler to create. Many CNN architectures produce better results, and the majority of them have been published using Keras. This new implementation will aim to develop and improve the model. The code will be available in the appendix for further examination.

The Keras development process is similar to Torch, with the exception of the use of the torch dataset loader. The first section of code would be to use Panda to import the dataset and join the paths of the images and CSV files. The images are then processed with Nibabel. The dataset is also divided into

train and test segments, with the shuffle set to true to avoid repeatedly using the same partitions.

```
x_train,x_test,y_train,y_test = train_test_split(X, y,
test_size=0.2, shuffle=True, random_state=69)
```

The dataset is now prepared for modelling. The next step in the implementation would be to structure the ConvNet layers and determine the best modelling technique. In the following chapter, the CNN will be developed and tested. The model will also be improved by incorporating techniques from other CNN architectures.

Chapter 7: Testing

7.1 PyTorch 3D CNN

The PyTorch library was used to create the first CNN model implementation. The model included ReLU activation, max pooling, dropout, and FC layer in addition to a 3D convnet with five layers. This version received the highest accuracy score of 64%. When compared to other works of literature, this is not the most accurate. The general architecture of this CNN model could be improved further. Some suggestions for enhancements include optimising hyperparameters and increasing the number of layers and epochs.

```
Epoch 0/9
----------
train Loss: 0.7579 Acc: 0.5399
valid Loss: 0.6982 Acc: 0.5407

Epoch 1/9
----------
train Loss: 0.6951 Acc: 0.5993
valid Loss: 0.6733 Acc: 0.5926

Epoch 2/9
----------
train Loss: 0.6595 Acc: 0.6308
valid Loss: 0.6913 Acc: 0.5852

Epoch 3/9
----------
train Loss: 0.6157 Acc: 0.6790
valid Loss: 0.7274 Acc: 0.5630

Epoch 4/9
----------
train Loss: 0.6050 Acc: 0.6772
valid Loss: 0.6986 Acc: 0.5778

Epoch 5/9
----------
train Loss: 0.5790 Acc: 0.6827
valid Loss: 0.6968 Acc: 0.6074

Epoch 6/9
----------
train Loss: 0.5458 Acc: 0.7310
valid Loss: 0.7149 Acc: 0.6074

Epoch 7/9
----------
```

```
train Loss: 0.5341 Acc: 0.7291
valid Loss: 0.6797 Acc: 0.6296

Epoch 8/9
----------
train Loss: 0.5121 Acc: 0.7570
valid Loss: 0.6999 Acc: 0.6148

Epoch 9/9
----------
train Loss: 0.4969 Acc: 0.7774
valid Loss: 0.6915 Acc: 0.6444

Training complete in 82m 15s
Best val Acc: 0.644444
```

After working with the torch library, It was clear that Keras is the most popular machine learning library with the amount of content available online. Although PyTorch is designed for faster processing, the syntax for Keras is much simpler, and many of the sources have based their architecture on Keras.

7.2 Keras basic model

The first model is based on Keras and has a 2D ConvNet layer. This is a simple design for generating initial accuracy. This model's precise summary is labelled below.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)            (None, 77, 93, 32)        22784

 activation_4 (Activation)    (None, 77, 93, 32)        0

 max_pooling2d_3 (MaxPooling  (None, 39, 47, 32)        0
 2D)

 conv2d_4 (Conv2D)            (None, 37, 45, 64)        18496

 activation_5 (Activation)    (None, 37, 45, 64)        0

 max_pooling2d_4 (MaxPooling  (None, 18, 22, 64)        0
 2D)

 conv2d_5 (Conv2D)            (None, 16, 20, 128)       73856
```

```
 activation_6 (Activation)   (None, 16, 20, 128)       0

 max_pooling2d_5 (MaxPooling  (None, 8, 10, 128)       0
 2D)

 flatten_1 (Flatten)         (None, 10240)             0

 dense_2 (Dense)             (None, 64)                655424

 activation_7 (Activation)   (None, 64)                0

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 770,625
Trainable params: 770,625
Non-trainable params: 0
_________________________________________________________________
```

After ten epochs of training, the results show an accuracy of 69.5%. Although this is 5% better than our 3D CNN model, the validation accuracy results give a final score of 56.3%.

```
Epoch 1/10
17/17 [==============================] - 16s 862ms/step - loss:
0.6930 - accuracy: 0.6827 - val_loss: 0.7590 - val_accuracy: 0.5630
Epoch 2/10
17/17 [==============================] - 14s 847ms/step - loss:
0.6410 - accuracy: 0.7013 - val_loss: 0.7118 - val_accuracy: 0.5630
Epoch 3/10
17/17 [==============================] - 14s 838ms/step - loss:
0.6367 - accuracy: 0.6809 - val_loss: 0.7272 - val_accuracy: 0.5630
Epoch 4/10
17/17 [==============================] - 15s 853ms/step - loss:
0.6415 - accuracy: 0.6957 - val_loss: 0.7119 - val_accuracy: 0.5630
Epoch 5/10
17/17 [==============================] - 14s 840ms/step - loss:
0.6181 - accuracy: 0.6957 - val_loss: 0.7186 - val_accuracy: 0.5630
Epoch 6/10
17/17 [==============================] - 14s 837ms/step - loss:
0.6187 - accuracy: 0.6957 - val_loss: 0.7126 - val_accuracy: 0.5630
Epoch 7/10
17/17 [==============================] - 14s 833ms/step - loss:
0.6137 - accuracy: 0.6957 - val_loss: 0.7028 - val_accuracy: 0.5630
Epoch 8/10
17/17 [==============================] - 14s 837ms/step - loss:
0.6065 - accuracy: 0.6957 - val_loss: 0.7067 - val_accuracy: 0.5630
Epoch 9/10
```

```
17/17 [==============================] - 14s 832ms/step - loss:
0.6111 - accuracy: 0.6957 - val_loss: 0.7185 - val_accuracy: 0.5630
Epoch 10/10
17/17 [==============================] - 14s 827ms/step - loss:
0.6064 - accuracy: 0.6957 - val_loss: 0.7157 - val_accuracy: 0.5630
```

The model was further evaluated by plotting graphs of model loss and train accuracy vs validation accuracy. The graphs show that the validation loss increases while the accuracy remains constant. The model is overfitting in this case.

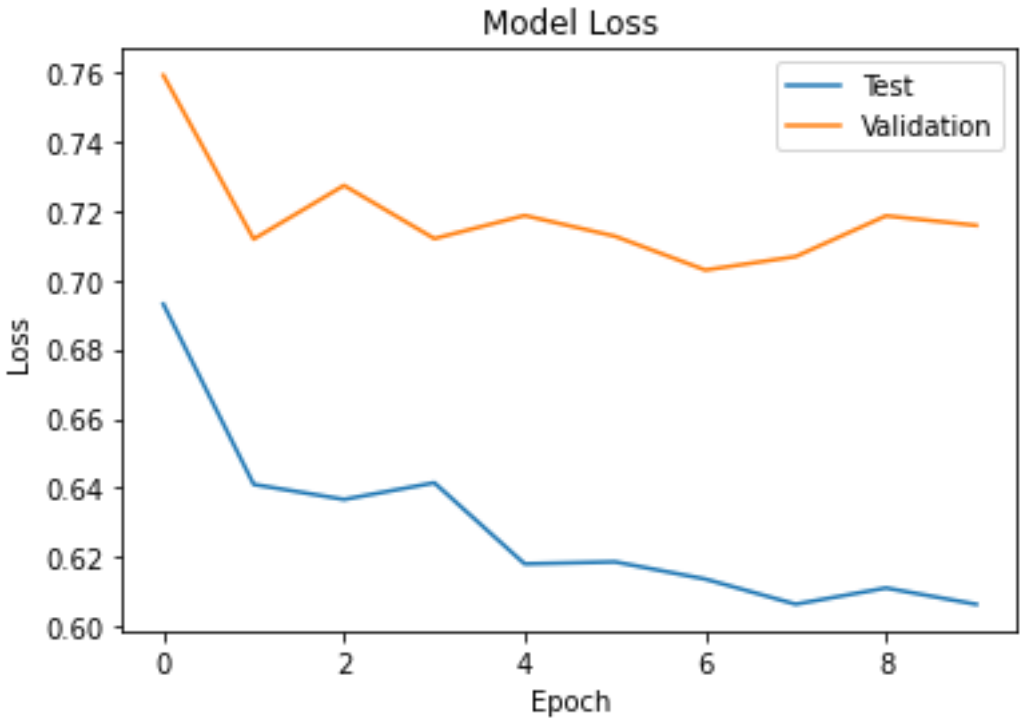

Figure 15 Model loss

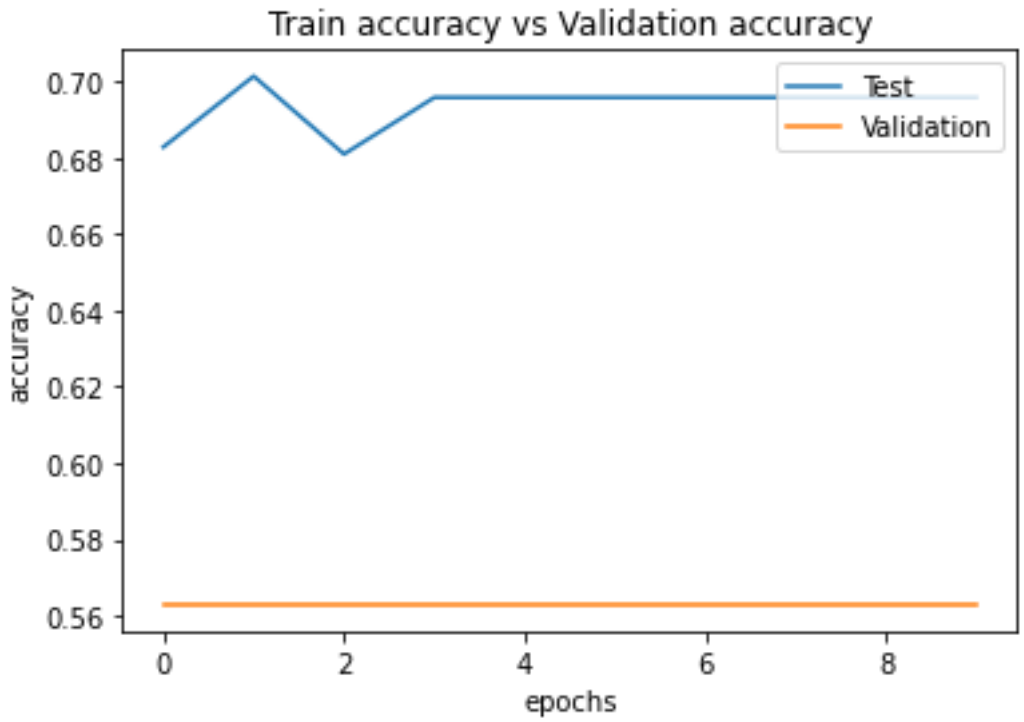

Figure 16 Train accuracy vs validation accuracy

7.3 VGG16

The VGG16 architecture will be used in this section of the testing. This is a significant improvement to the model, which has a total parameter count of 40 million as opposed to the previous model's 770 thousand. The summary of the model is shown below.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 79, 95, 64)        45568

 conv2d_1 (Conv2D)           (None, 79, 95, 64)        36928

 max_pooling2d (MaxPooling2D  (None, 39, 47, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 39, 47, 128)       73856
```

```
conv2d_3 (Conv2D)            (None, 39, 47, 128)       147584

max_pooling2d_1 (MaxPooling  (None, 19, 23, 128)       0
2D)

conv2d_4 (Conv2D)            (None, 19, 23, 256)       295168

conv2d_5 (Conv2D)            (None, 19, 23, 256)       590080

conv2d_6 (Conv2D)            (None, 19, 23, 256)       590080

max_pooling2d_2 (MaxPooling  (None, 9, 11, 256)        0
2D)

conv2d_7 (Conv2D)            (None, 9, 11, 512)        1180160

conv2d_8 (Conv2D)            (None, 9, 11, 512)        2359808

conv2d_9 (Conv2D)            (None, 9, 11, 512)        2359808

max_pooling2d_3 (MaxPooling  (None, 4, 5, 512)         0
2D)

conv2d_10 (Conv2D)           (None, 4, 5, 512)         2359808

conv2d_11 (Conv2D)           (None, 4, 5, 512)         2359808

conv2d_12 (Conv2D)           (None, 4, 5, 512)         2359808

max_pooling2d_4 (MaxPooling  (None, 2, 2, 512)         0
2D)

flatten (Flatten)            (None, 2048)              0

dense (Dense)                (None, 4096)              8392704

dense_1 (Dense)              (None, 4096)              16781312

dense_2 (Dense)              (None, 2)                 8194

=================================================================
Total params: 39,940,674
Trainable params: 39,940,674
Non-trainable params: 0
```

This architecture produced better validation accuracy results, at 68.9%, but this percentage remained constant throughout the epoch, showing problems with the model.

```
Epoch 1/10

17/17 [==============================] - 95s 5s/step - loss:
15.4445 - accuracy: 0.6605 - val_loss: 0.6315 - val_accuracy:
0.6889
```

```
Epoch 2/10
17/17 [==============================] - 92s 5s/step - loss: 0.6499
- accuracy: 0.6642 - val_loss: 0.6302 - val_accuracy: 0.6889
Epoch 3/10
17/17 [==============================] - 92s 5s/step - loss: 0.6403
- accuracy: 0.6642 - val_loss: 0.6206 - val_accuracy: 0.6889
Epoch 4/10
17/17 [==============================] - 92s 5s/step - loss: 0.6396
- accuracy: 0.6642 - val_loss: 0.6211 - val_accuracy: 0.6889
Epoch 5/10
17/17 [==============================] - 93s 5s/step - loss: 0.6397
- accuracy: 0.6642 - val_loss: 0.6204 - val_accuracy: 0.6889
Epoch 6/10
17/17 [==============================] - 94s 6s/step - loss: 0.6392
- accuracy: 0.6642 - val_loss: 0.6231 - val_accuracy: 0.6889
Epoch 7/10
17/17 [==============================] - 93s 5s/step - loss: 0.6411
- accuracy: 0.6642 - val_loss: 0.6203 - val_accuracy: 0.6889
Epoch 8/10
17/17 [==============================] - 93s 5s/step - loss: 0.6392
- accuracy: 0.6642 - val_loss: 0.6217 - val_accuracy: 0.6889
Epoch 9/10
17/17 [==============================] - 93s 5s/step - loss: 0.6388
- accuracy: 0.6642 - val_loss: 0.6210 - val_accuracy: 0.6889
Epoch 10/10
17/17 [==============================] - 93s 6s/step - loss: 0.6411
- accuracy: 0.6642 - val_loss: 0.6204 - val_accuracy: 0.6889
```

After evaluating the model, the Keras model performed better in validation accuracy, but the graphs do not show a healthy model. This could be interpreted as a clear indication that the model is overfitting.

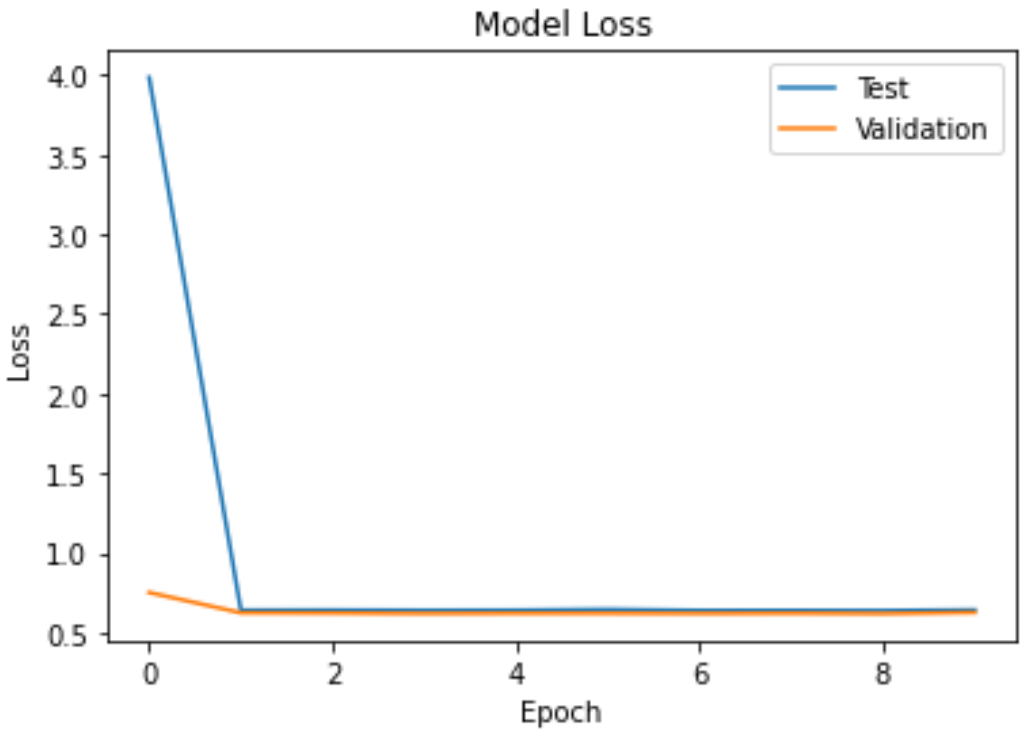

Figure 17 Model loss 2

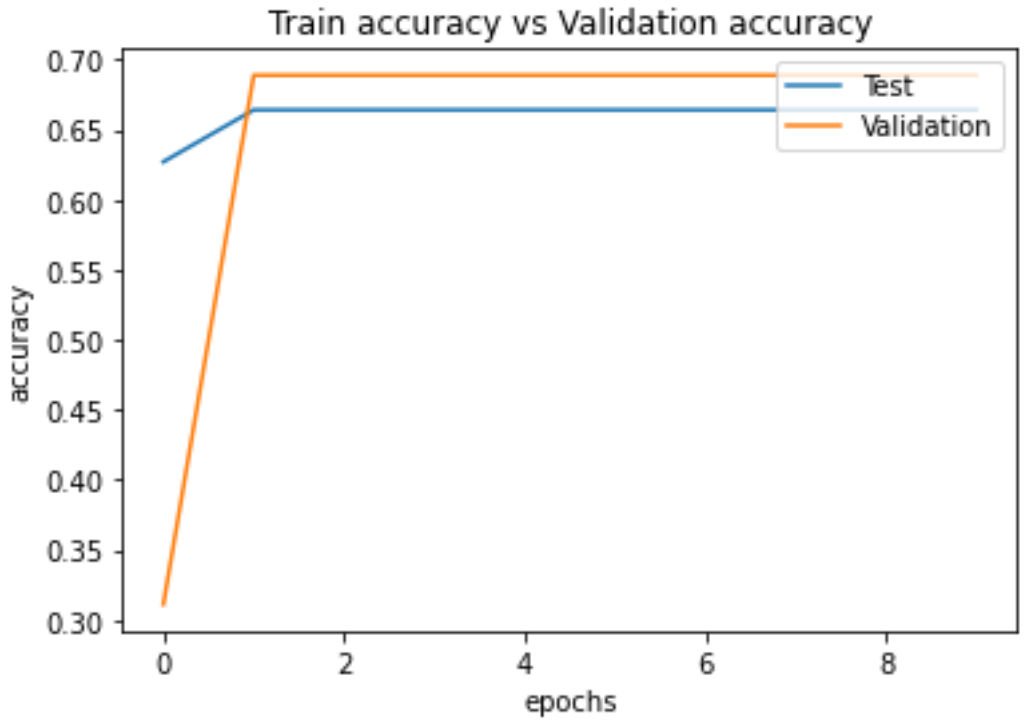

Figure 18 Train accuracy vs validation accuracy 2

Chapter 8: Further works

The time for this project flew quickly since there are still so many different aspects to cover. Some of the important areas that may be researched and developed further include:

1. Fixing basic structure of the code to handle different types of ConvNet layers.
2. Eradicating the overfitting problem by integrating mitigation techniques such as ReLU activation function, Dropout layer or even developing a ResNet based model.
3. Create UNet model to examine a single slice of the brain.
4. Increasing learning efficiency by identifying optimal tuning for hyperparameters.
5. Upgrade computational processing package to train models with significantly more parameters.
6. Utilising matplotlib for further model evaluation

Chapter 9: Evaluation

The first model was created using PyTorch to generate a ConvNet with 3D layers. The initial results for this 3D-CNN achieved the best accuracy of 64.4%. This is an impressive score for a basic implementation of this model. This followed the system's initial design to achieve the classification of Psychosis, but further improvements could have been made by implementing a different architecture. There was no architectural framework supporting 3D layers, so it was impossible to proceed. Furthermore, the computational cost of training this model is high, and training would take a long time. Due to the limited time, it was the best option to switch to Keras, which has a simpler syntax and implement a 2D layer instead, generating a faster result.

The second model was based on the Keras library, utilising a 2D convnet. There were two versions of this model. The first version was built with a basic structure, and the second one implements the VGG16 architecture. This gave an accuracy score of 56.3%, which improved to 68.8% through VGG16. This implementation shared the issues of overfitting, thus harming the validity of the results. This could not be resolved in time, but suggestions to fix this could lie in the dataset's dropout layer, weights, or partition. The primary structuring for this model is laid out and given more time, sufficient testing and configuration could have been carried out to generate a healthy model.

Chapter 10: Conclusion

The Deep Learning application for diagnosing psychosis has a promising future and may be adopted by the general public in the future. The research aided in developing a comprehensive understanding of the field of Deep Learning Diagnostics for Mental Health Diagnostics. There was a wealth of literature demonstrating ground-breaking discoveries and technological trends. The findings of these studies are documented throughout this report, which addresses one of the dissertation's objectives: learning more about the current state of affairs.

The project followed an agile scrum methodology which helped tackle progress of the project in a timely manner. Although more time dedicated to the implementation phase would have led to better results. The results gathered from the various implementation of these models produced an accuracy greater than 55%, which was then subject to configuration and optimisation.

The developed code could access the dataset's paths, which included the CSV file containing the information and the images stored in zip format. The arrays can then be transformed by the code to fit onto a 2D, or 3D model. A data partition was also used to make a training and validation set. The ConvNet was developed with this code thus achieving the main objective of this project.

The actual ConvNet was modelled in two different ways using the PyTorch and Keras libraries. The Torch library was used to implement the 3D ConvNet, which produced an accuracy of 64%. This was a good model to begin with because it did not exhibit overfitting or underfitting issues. The only problem with using this version of the ConvNet was that it was computationally expensive and took a long time to train. If more time and computational resources were available for this project, it would undoubtedly be worthwhile to improve this model by investigating methods proposed by other researchers.

The 2D version of the ConvNet, on the other hand, displayed numerous technical and overfitting issues; despite having a training score of 69.6 percent, the validation reduced this to 56 percent, making it worse than the 3D version. This is because the model showed signs of overfitting, drastically affecting the performance. This used the VGG16 architecture, which was not the best choice for design at the time. Despite being an improvement over AlexNet, the VGG16 lacked any mitigation technique to protect the data from overfitting, underfitting, or vanishing gradient problems. It could be surmised the reason for the overfitting could be the absent of dropout layers, management of weights or the data did not partition properly.

Working with ConvNet in general would almost certainly result in the model being prone to overfitting, which are challenges that every Deep Learning researcher would need to overcome. The report discussed methods of eradicating this problem in section 2.3.3 by using the concept introduced by

ResNet, the “identity shortcut connection”. This concept would have been beneficial in adapting to this model for improved performance. Another issue is the computational process; training a model takes far too long, leaving little room for improvement. And the code would frequently crash, limiting the number of layers we could use.

Overcoming the obstacles faced by the implementation of CNN will only help achieve the best possible model that can potentially be adopted by the health care system. Making it worthwhile to continue research that could potentially help millions of people.

Chapter 11: Reflection

11.1 Journey

Having no prior knowledge of mental illness diagnosis and deep learning has led me to conduct extensive research and reading of various literatures. The research was extremely beneficial in terms of understanding the subjects and gaining a current perspective on the field. As things stand, the need for automated mental illness diagnosis is greater than ever. According to the Institute of Health Metrics Evaluation (IHME), 971 million people were diagnosed with mental illnesses in 2017, but countries only have less than 10 mental health nurses for every 100,000 people. The information gathered from this research has convinced me of the significance of this work and how it may benefit the outdated health-care system.

Having no experience with neural networks it was difficult to even begin the construction of this project. But my familiarity with Python, alongside the data science and artificial intelligence modules helped overcome this. In addition, researching the key technology needed in the technical review helped define the clear goals of this project. The code couldn’t have been built with the assistant of my supervisor, Enrico who provided the dataset loader and led to me to understand more about developing the model and the different architectures that could be implemented.

11.2 Challenges and weaknesses

The development of the code was the most difficult aspect of this project. Building an actual model from scratch proved far more difficult given the time constraints. Many technical aspects had to be considered before the code was ready for modelling. The PyTorch library was difficult to understand because it was not based on Python syntax. I would have spent more time learning how to use the library if I had more time.

Too much time was wasted in attempting in-depth research into the subject, leaving me with insufficient time to complete the development to a satisfactory standard. More time was required for further improvements and testing of the CNN.

The limited computational process was also a challenge when developing the model, as it frequently crashed, forcing me to make several changes. This was partially mitigated by upgrading to Google Cola Pro, but the Google Colab Pro+ may have been required for even more computational power.

11.3 Learning essentials

There has been a lot of research done on neural networks to understand the fundamentals. In addition, to help with this project, I completed a neural network and a CNN course on LinkedIn Learning. To aid in the development of CNN, I have read a variety of articles highlighting the implementation of their various models.

11.4 Strengths

My ability to research into specific subject and gather information has helped greatly for this dissertation. Having no knowledge of the topic at hand led me to go deep into the subject to understand each technical term. Having a good understanding of Python also helped fast track the development phase which also assisted with integrating and troubleshooting the Python code.

11.5 Lessons learnt

This project has helped secure my knowledge on deep learning architectures on both theoretical and practical aspects. I have good fundamental knowledge on the implications of deep learning for mental health diagnosis. This has given me confidence developing a better model that can incorporate new techniques.

References

Aburasain, R., 2020. Application of convolutional neural networks in object detection, re-identification and recognition. *Loughborough University.*

Alom, M. Z. et al., 2018. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *CoRR.*

Arciniegas, D. B., 2022. Psychosis. *Continuum (Minneapolis, Minn.),* 21(3), pp. 715-36.

Campese, S. et al., 2019. Psychiatric Disorders Classification. *Proceedings of the International Neural Networks Society,* pp. 48-57.

Dang, A. T., 2021. Top 10 CNN Architectures Every Machine Learning Engineer Should Know. *Towards Data Science.*

FDA, 2018. *Radiation-Emitting Products and Procedures.* [Online]
Available at: https://www.fda.gov/radiation-emitting-products/radiation-emitting-products-and-procedures/medical-imaging#:~:text=Medical%20imaging%20refers%20to%20several,monitor%2C%20or%20treat%20medical%20conditions.

Feng, V., 2017. An Overview of ResNet and its Variants. *Towards Data Science.*

He, K., Zhang, X., Ren, S. & Sun, J., 2015. Deep Residual Learning for Image Recognition.

IBM Cloud Education, 2020. *IBM Cloud Learn Hub / What is Deep Learning?.* [Online]
Available at: https://www.ibm.com/cloud/learn/deep-learning#:~:text=Deep%20learning%20is%20a%20subset,from%20large%20amounts%20of%20data.

Kurmanji, M. & Ghaderi, F., 2019. A Comparison of 2D and 3D Convolutional Neural Networks for Hand Gesture Recognition from RGB-D Data. *Iranian Conference on Electrical Engineering.*

Li, Z. et al., 2021. Deep learning based automatic diagnosis of first-episode psychosis, bipolar disorder, and healthy controls. *National Library of Medicine,* p. 89.

Maggioni, E., Benedicto, C.-F. & Igor, N., 2017. Common and distinct structural features of schizophrenia and bipolar disorder: The European Network on Psychosis, Affective disorders and Cognitive Trajectory (ENPACT) study. *PLOS ONE.*

Qassim, H., Verma, A. & Feinzimer, D., 2018. Compressed residual-VGG16 CNN model for big data places image recognition. pp. 169-175.

Saibene, A., Assale, M. & Giltri, M., 2021. Expert systems: Definitions, advantages and issues in medical field applications. *Expert Systems with Applications.*

Sharma, R., 2021. Python vs. Scala: Difference Between Python & Scala [2022]. *upGrade,* 5 January.

Shen, J. et al., 2019. Artificial Intelligence Versus Clinicians in Disease Diagnosis: Systematic Revie. *JMIR Med Inform,* Volume 7.

Sommer, E. I. et al., 2012. How Frequent Are Radiological Abnormalities in Patients With Psychosis? A Review of 1379 MRI Scans. *Schizophrenia Bulletin,* p. 815–819.

Squarcina, L., Castellani, U., Bellani, M. & Perlini, C., 2015. Classification of first-episode psychosis in a large cohort of patients using support vector machine and multiple kernel learning techniques. *National Library of Medicine,* p. 238–245.

Terra, J., 2022. Keras vs Tensorflow vs Pytorch: Key Differences Among the Deep Learning Framework. *Simplilearn.*

Veronese, E., Castellani, U., Peruzzo, D. & Bellani, M., 2013. Machine Learning Approaches: From Theory to Application in Schizophrenia. *Computational and Mathematical Methods in Medicine,* p. 12.

Wang, J. et al., 2021. *CNN Explainer.* [Online]
Available at: https://poloclub.github.io/cnn-explainer/

Appendix:

London South Bank University
EST 1892

School of Engineering

BSc FINAL YEAR PROJECT
ETHICAL CONSIDERATIONS
2021-22

YOUR DETAILS

Name of student:	Sheikh Khaled Ahmed
Supervisor:	Enrico Grisan
Project title:	Deep learning research and application of Brain MRI based Convolutional Neural Network for detecting Psychosis
Main aim of project:	Research deep learning methods and the application of a Convolutional Neural Network for detecting mental health illnesses and develop an effective CNN algorithm to detect Psychosis from a brain MRI dataset

CONTACT WITH OTHERS

Will your project bring you into contact with other people (e.g. via an online survey)?

No

If you answered "No", sign the section below and submit this page only to your supervisor for countersigning, otherwise complete the whole form prior to submission. Also, if you answered "No" then only this page needs to be included as an appendix to your dissertation.

YOUR SIGNATURE

Signature: Sheikh Khaled Ahmed Date: 05/05/2022

ETHICAL APROVAL
(To be completed by your supervisor)

I have checked the above for accuracy and I am satisfied that the information provided is an accurate reflection of the intended study.

There are no ethical issues causing my concern ☐

Signature: _____ Date: _____

Name (please print): _____

Code (Part 1):

```
# -*- coding: utf-8 -*-
"""pyTorch-CNN.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1ZYzijANfSTvXRVJ2LoS2h-o-UaWgeQRv
"""

import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import nibabel as nb

from sklearn.model_selection import train_test_split

import torch
from torch.utils.data import Dataset, DataLoader

import torch.nn as nn
import torchvision.transforms as transforms

"""#Importing dataset"""

# BASE_DIR="/content/drive/MyDrive/brains"
# #TRAIN_IMAGES_DIR=os.path.join(BASE_DIR,'train_images')

data_df= pd.read_csv('/content/drive/MyDrive/Classification_clean.csv')

data_df["Class"]=data_df["Diagnosis"]>0
#data_df["Class"].replace(True,2,inplace=True)
#data_df["Class"].replace(False,1,inplace=True)
data_df.Class = data_df.Class.astype('int')


data_df.head()

class Milano_MRI_Dataset(Dataset):
    def __init__(self,df:pd.DataFrame,imfolder:str,train:bool = True,
transforms=None):
        self.df=df
        self.imfolder=imfolder
        self.train=train
        self.transforms=transforms
```

```
    def __getitem__(self,index):
        filename='w'+self.df.iloc[index]['ID_CF']+'_MRI_sMRI_'+self.df.iloc[in
dex]['ID_CF']+'_brain.nii.gz';

        im_path=os.path.join(self.imfolder,filename)

        nimg = nb.load(im_path)
        x = np.array(nimg.dataobj)
        x=nimg.get_fdata()
        x = transforms.ToTensor()(x)
        x=x.unsqueeze(0).type(torch.FloatTensor);
        m=torch.mean(x)
        s=torch.std(x)
        x=(x-m)/s
        if(self.transforms):
           x=self.transforms(image=x)['image']

        if(self.train):
            y=self.df.iloc[index]['Class']
            return x,y
        else:
            return x

    def __len__(self):
        return len(self.df)

"""#Training and testing"""

train, valid = train_test_split(
    data_df,
    test_size=0.2,
    random_state=42,
    stratify=data_df.Class.values
)

# reset index on both dataframes
train = train.reset_index(drop=True)
valid = valid.reset_index(drop=True)

train_targets = train.Class.values

# targets for validation
valid_targets = valid.Class.values

train_dataset=Milano_MRI_Dataset(
    df=train,
    imfolder=BASE_DIR,
    train=True,
```

```
    transforms=None
)

valid_dataset=Milano_MRI_Dataset(
    df=valid,
    imfolder=BASE_DIR,
    train=True,
    transforms=None
)

train_loader = DataLoader(
    train_dataset,
    batch_size=15,
    #num_workers=4,
    shuffle=True,
)

valid_loader = DataLoader(
    valid_dataset,
    batch_size=15,
    #num_workers=4,
    shuffle=False,
)

valid

"""#3D CNN"""

import torch
import torch.nn as nn
import math
from functools import partial
from torch.autograd import Variable

import torch.optim as optim
from torch.optim.lr_scheduler import ReduceLROnPlateau

import time
import datetime
import copy

class Simple3DCNN(nn.Module):

    def __init__(self, num_classes):


        super(Simple3DCNN, self).__init__()

        self.conv_layer1 = self._make_conv_layer(1, 32)
```

```
        self.conv_layer2 = self._make_conv_layer(32, 64)
        self.conv_layer3 = self._make_conv_layer(64, 128)
        self.conv_layer4 = self._make_conv_layer(128, 256)
        self.conv_layer5=nn.Conv3d(256, 2048, kernel_size=(4, 4, 5),
padding=0)

        self.fc5 = nn.Linear(2048, 512)
        self.relu = nn.LeakyReLU()
        self.batch0=nn.BatchNorm1d(512)
        self.drop=nn.Dropout(p=0.25)
        self.fc6 = nn.Linear(512, 256)
        self.relu = nn.LeakyReLU()
        self.batch1=nn.BatchNorm1d(256)

        self.drop=nn.Dropout(p=0.25)
        self.fc7 = nn.Linear(256, num_classes)

    def _make_conv_layer(self, in_c, out_c):
        conv_layer = nn.Sequential(
        nn.Conv3d(in_c, out_c, kernel_size=3, padding=1),
        nn.LeakyReLU(),
        #nn.Conv3d(out_c, out_c, kernel_size=3, padding=1),
        #nn.LeakyReLU(),
        nn.BatchNorm3d(out_c),
        nn.MaxPool3d(kernel_size=2, stride=2),
        )
        return conv_layer

    def forward(self, x):
        #print(x.size())
        x = self.conv_layer1(x)
        #print(x.size())
        x = self.conv_layer2(x)
        #print(x.size())
        x = self.conv_layer3(x)
        #print(x.size())
        x = self.conv_layer4(x)
        #print(x.size())
        x=self.conv_layer5(x)
        #print(x.size())
        x = x.view(x.size(0), -1)
        #print(x.size())
        x = self.fc5(x)
        x = self.relu(x)
        #print(x.size())
        x = self.batch0(x)
        x = self.drop(x)
        x = self.fc6(x)
        x = self.relu(x)
```

```
        x = self.batch1(x)
        x = self.drop(x)
        x = self.fc7(x)

        return x#,x1

model = Simple3DCNN(2)

tmp=next(iter(train_loader))
out=model(tmp[0])

"""#Defining the training function

"""

def train_model(datasets, dataloaders, model, criterion, optimizer, scheduler,
num_epochs, device):

    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs-1))
        print('-' * 10)

        for phase in ['train', 'valid']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0.0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels=labels.to(device)

                # Zero out the grads
                optimizer.zero_grad()

                # Forward
                # Track history in train mode
                with torch.set_grad_enabled(phase == 'train'):
                    model=model.to(device)
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
```

```
                    #loss = criterion(outputs,
labels.type(torch.LongTensor).unsqueeze(1).to(device))
                    loss = criterion(outputs,
labels.type(torch.LongTensor).to(device))

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
                # Statistics
                running_loss += loss.item()*inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss/len(datasets[phase])
            epoch_acc = running_corrects.double()/len(datasets[phase])

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            if phase == 'valid' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time()-since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    model.load_state_dict(best_model_wts)
    return model

"""# Training

"""

class_sample_count = np.array([len(np.where(train_targets == t)[0]) for t in
np.unique(train_targets)])
weight = 1. / class_sample_count
class_weight=torch.from_numpy(weight)
class_weight=class_weight.max()/class_weight
class_weight=class_weight/class_weight.max()
class_weight=class_weight.type(torch.FloatTensor)
print(class_weight)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
datasets={'train':train_dataset,'valid':valid_dataset}
dataloaders={'train':train_loader,'valid':valid_loader}

optimizer = torch.optim.AdamW(model.parameters(), lr=1e-6, weight_decay=0.001)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=100,
gamma=0.1)

criterion=nn.CrossEntropyLoss()
#criterion=nn.BCEWithLogitsLoss()

num_epochs=10

trained_model=train_model(datasets,dataloaders,model,criterion,optimizer,sched
uler,num_epochs,device)
```

Code (Part 2):

```
# -*- coding: utf-8 -*-
"""VGG16.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1VNFGw3xOp68VhQmSEEVPlXG4daTyvr7D
"""

# Commented out IPython magic to ensure Python compatibility.
# import packages
import skimage, os
from skimage.morphology import ball, disk, dilation, binary_erosion,
remove_small_objects, erosion, closing, reconstruction, binary_closing
from skimage.measure import label,regionprops, perimeter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tqdm import tqdm
from skimage.morphology import binary_dilation, binary_opening
from skimage.filters import roberts, sobel
from skimage import measure, feature
from skimage.segmentation import clear_border
from skimage import data
from scipy import ndimage as ndi
#!pip install keras
import keras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, Conv3D, MaxPooling2D
from keras.preprocessing import image
# %matplotlib inline
import nibabel as nib
import random
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import scipy.misc
from glob import glob
from skimage.io import imread
import skimage.transform as skTrans
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
```

```
from matplotlib import pyplot
# Set

BASE_IMG_PATH=os.path.join('/content/drive/MyDrive')

#insert dataset
brainMRIData=pd.read_csv('/content/drive/MyDrive/Classification_clean.csv')
# find class value
brainMRIData["Class"]=brainMRIData["Diagnosis"]>0
# set datatype
brainMRIData.Class = brainMRIData.Class.astype('int')
brainMRIDataCol=brainMRIData.columns
brainMRIData

# perform path
brainMRIImages=glob(os.path.join(BASE_IMG_PATH,'brains','wCF_*'))
print(os.path.join(BASE_IMG_PATH,'brains','wCF_*'))
print(len(brainMRIImages),' matching files found:',brainMRIImages[0])

brainMRIData.columns

# load images
imagesTest=nib.load(brainMRIImages[0])
imagesTest=imagesTest.get_fdata()
random.shuffle(imagesTest)

# for loop for load images
for myimages in imagesTest:
    fileName='w'+ brainMRIData['ID_CF']+'_MRI_sMRI_'+
brainMRIData['ID_CF']+'.nii.gz'

    break

# print filenames
print(fileName)

# EDA
from skimage.util import montage as montage2d
fig, showImg = plt.subplots(1, 1, figsize = (100, 100))
showImg.imshow(montage2d(imagesTest), cmap ='gray')

# Commented out IPython magic to ensure Python compatibility.
from tqdm import tqdm

showImgArr = []

for i in tqdm(range(brainMRIData.shape[0])):
    try:
        import nibabel as nib
```

```
    except:
        raise ImportError('Install NIBABEL')

    t_image=nib.load(brainMRIImages[0]).get_fdata()


    t_image= t_image/255
    showImgArr.append(t_image)
test = np.array(showImgArr)

# %matplotlib inline

test_image=nib.load('/content/drive/MyDrive/brains/wCF_023_MRI_sMRI_CF_023_bra
in.nii.gz')
t_image=test_image.get_fdata()
(fig, ax1) = plt.subplots( figsize = (6, 3))
ax1.imshow(t_image[t_image.shape[2]//2])
ax1.set_title('Image')

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline

test_image=nib.load('/content/drive/MyDrive/brains/wCF_023_MRI_sMRI_CF_023_bra
in.nii.gz')
t_image=test_image.get_fdata()
(fig, ax1) = plt.subplots( figsize = (6, 3))
ax1.imshow(t_image[t_image.shape[2]//2])
ax1.set_title('Image')

import skimage.transform as skTrans
# array
showImgArr = []
# import image
for i in tqdm(range(brainMRIData.shape[0])):
    try:
        import nibabel as nib
    except:
        raise ImportError('Install NIBABEL')

    imagesTest=nib.load(brainMRIImages[10])
    imagesTest=imagesTest.get_fdata()
    # suffle images
    random.shuffle(imagesTest)

    FigT= imagesTest/255
    # append image
    showImgArr.append(FigT)

# set array value
```

```
X = np.array(showImgArr)

X.shape

# find y value
#y = np.array(brainMRIData.drop(['ID_CF', 'First Episodes (FEP) / Healthy
Control (HC)'],axis=1))
y = np.array(brainMRIData['Class'])
y.shape

# split data

# xTrainValue, xTestValue, yTrainValue, yTestValue = train_test_split(X, y,
random_state=32, test_size=0.2)
# yTrainValue

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=16)
# y_train

"""#cnn"""

import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from keras.preprocessing.image import ImageDataGenerator
import numpy as np

#CNN
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(input_shape=(79, 95,
79),filters=64,kernel_size=(3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3),padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
```

```
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=2, activation="softmax"))

model.summary()

"""testing"""

# xTrainValue, xTestValue, yTrainValue, yTestValue = train_test_split(X, y,
random_state=32, test_size=0.2)
# yTrainValue

x_train,x_test,y_train,y_test = train_test_split(X, y, test_size=0.2,
shuffle=True, random_state=69)

# model.compile(loss=keras.losses.BinaryCrossentropy(from_logits=True),
#                   optimizer=tf.keras.optimizers.Adam(lr=1e-5),
metrics=['accuracy'])

model.compile(optimizer='adam',loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history=model.fit(x_train, y_train, epochs=10, validation_data=(x_test,
y_test))


# hist = model.fit_generator(steps_per_epoch=10,generator=x_train,
validation_data=(x_test, y_test), validation_steps=10,epochs=100)
```

```
# history = model.fit(x_train, y_train, epochs=10, batch_size=40,
verbose=1,validation_data=(x_test, y_test))

"""original"""

# plot the accuracy and loss
def plot_accuracy_loss(history):

    fig = plt.figure(figsize=(10,5))

    # Plot accuracy
    plt.subplot(221)
    # find accuracy
    plt.plot(history.history['accuracy'],'bo--', label = "accuracy")
    # find validation accuracy
    plt.plot(history.history['val_accuracy'], 'ro--', label = "val_accuracy")
    # set title
    plt.title("train_acc vs val_accuracy")
    # set labels
    plt.ylabel("accuracy")
    plt.xlabel("epochs")
    # set legend
    plt.legend()

    # Plot loss
    plt.subplot(222)
    # find loss value
    plt.plot(history.history['loss'],'bo--', label = "loss")
    # find validation loss value
    plt.plot(history.history['val_loss'], 'ro--', label = "val_loss")
    plt.title("train_loss vs val_loss")
    # set labels and legend
    plt.ylabel("loss")
    plt.xlabel("epochs")

    plt.legend()
    plt.show()
# plot accuracy and loss
plot_accuracy_loss(history)

# CNNmodel evaluation
lossTest = model.evaluate(x_test, y_test)

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Test', 'Validation'], loc='upper right')
```

```
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Train accuracy vs Validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['Test', 'Validation'], loc='upper right')
plt.show()

"""https://github.com/1297rohit/VGG16-In-
Keras/blob/master/VGG16%20in%20Keras.ipynb

https://www.kaggle.com/code/function9/brain-tumor-mri-cnn-classification-with-
keras/notebook

https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-
beginners-a833c686ae6c

"""
```