

ASSERTAINING BRAIN DISEASES USING DEEP LEARNING TECHNIQUES

Submitted in partial fulfilment of the requirements for the award of
Master of Engineering Degree in Computer Science and Engineering

By

PERCY PAULIN J
(Reg. No. 42310007)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

CATEGORY - 1 UNIVERSITY BY UGC

Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE

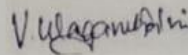
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600119.

APRIL 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **PERCY PAULIN J (Reg no:42310007)** who carried out the project entitled "**ASSERTAINING BRAIN DISEASES USING DEEP LEARNING TECHNIQUES**" under my supervision from December 2023 to April 2024.



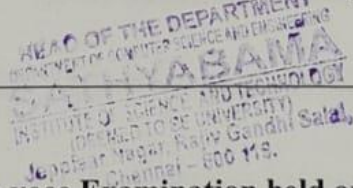
Internal Guide

Dr. V. ULAGAMUTHALVI, M.E., Ph.D.,

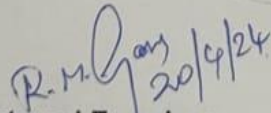


Head of the Department

Dr. L. Lakshmanan M.E., Ph.D.,



Submitted for Viva voce Examination held on 20/04/2024

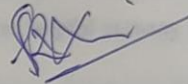

Internal Examiner


External Examiner

DECLARATION

I, PERCY PAULIN J (42310007) hereby declare that the Project Phase-2 Report titled "ASSERTAINING BRAIN DISEASES USING DEEP LEARNING TECHNIQUES" done by me under the Guidance of Dr. V. ULAGAMUTHALVI, M.E., Ph.D., at Sathyabama Institute of Science and Technology is submitted in partial fulfilment of the requirements for the award of Master of Engineering degree in Computer Science and Engineering.

DATE: 20/04/2024.



PLACE: Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of Sathyabama Institute of Science And Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. V. Ulagamuthalvi, M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project phase-2 work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

We propose a novel approach for the classification of brain tumour and Alzheimer's disease stages utilizing Convolutional Neural Network (CNN) architectures. Leveraging the power of deep learning, our model aims to accurately distinguish between different stages of these neurological conditions by analysing medical imaging data, such as MRI scans. The CNN architectures are designed to automatically learn and extract intricate features from the imaging data, enabling robust classification. The proposed methodology not only contributes to the early and accurate diagnosis of brain tumours and Alzheimer's disease but also holds promise for enhancing our understanding of the disease progression. The integration of advanced neural network techniques in medical image analysis showcases the potential for improved diagnostic tools in the field of neurology, ultimately leading to more effective treatment strategies and better patient outcomes.

TABLE OF CONTENT

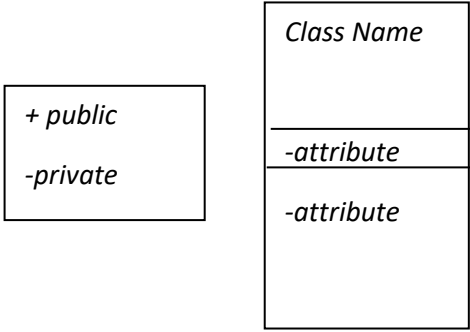
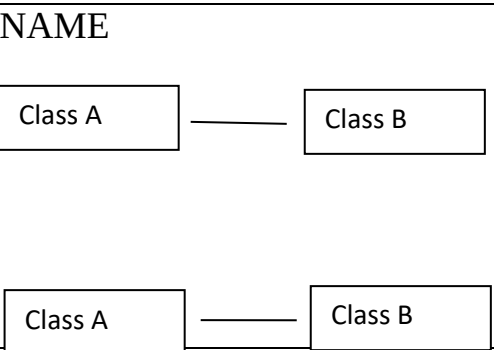
CHAPTER.NO	TITLE	PAGE.NO
	ABSTRACT	1
	LIST OF FIGURES	2
	LIST OF SYMBOLS	3
01	INTRODUCTION	7
	EXISTING SYSTEM	8
	1.1 DRAWBACKS	
	1.2 DATA SCIENCE	9
	1.3 ARTIFICIAL INTELLIGENCE	
	MACHINE LEARNING	12
	PREPARING DATASET	21
	PROPOSED SYSTEM	23
	1.4 ADVANTAGES	
02	LITERATURE SURVEY	15
03	SYSTEM STUDY	24
	3.1 OBJECTIVES	
	3.2 PROJECT GOAL	
	3.3 SCOPE OF THE PROJECT	
04	FEASIBILITY STUDY	25
	LIST OF MODULES	28
05	PROJECT REQUIREMENTS	27
	5.1 FUNCTIONAL REQUIREMENTS	
	5.2 NON-FUNCTIONAL REQUIREMENTS	
	ENVIRONMENT REQUIREMENT	27
	SOFTWARE DESCRIPTION	31
	5.3 ANACONDA NAVIGATOR	
	5.4 JUPYTER NOTEBOOK	
	PYTHON	32
	SYSTEM ARCHITECTURE	33

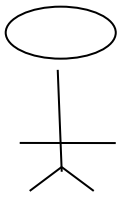
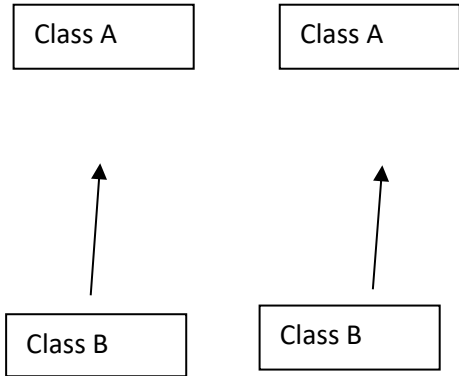
	DEPLOYMENT	46
	HTML	49
	CSS	53
	CODING	58
06	OUTPUT	78
07	CONCLUSION	80

LIST OF FIGURES

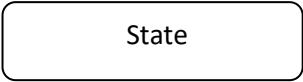
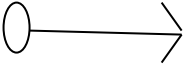
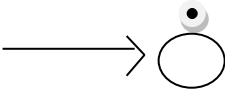
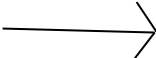
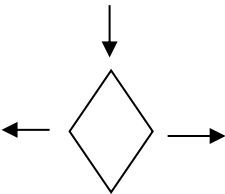
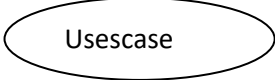
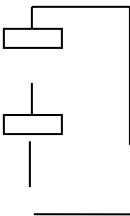
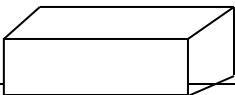
SL.NO	TITLE	PAGE.NO
01	SYSTEM ARCHITECTURE	33
02	MODULE DIAGRAM	29

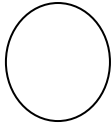
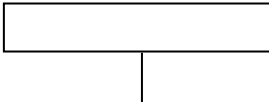
LIST OF SYMBOLS

S.NO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association		Associations represents static relationships between classes. Roles represents the way the

			two classes see each other.
3.	Actor		It aggregates several classes into a single classes.
4.	Aggregation		Interaction between the system and external environment

5.	<i>Relation</i> (uses)	<i>uses</i>	Used for additional process communication.
6.	Relation (extends)	EXTENDS →	Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication	_____	Communication between various use cases.

8.	State		State of the processs.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint
13.	Usecase		Interact ion between the system and external environment.
14.	Component		Represents physical modules which is a collection of components.
			

15.	Node		Represents physical modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.

CHAPTER 1

INTRODUCTION

The classification of brain tumour and Alzheimer's disease stages through the use of Convolutional Neural Network (CNN) architectures represents a crucial intersection of medical imaging and advanced deep learning techniques. This innovative approach leverages the capabilities of CNNs to automatically extract hierarchical features from complex medical images, facilitating the accurate identification and categorization of pathological conditions. In the case of brain tumour classification, CNNs analyse structural abnormalities in neuroimaging data, aiding in the early detection and characterization of tumour. Similarly, for Alzheimer's disease stages, these architectures interpret patterns in brain scans to categorize the severity of the disease, providing valuable insights for timely intervention and personalized treatment strategies. This research not only showcases the potential for CNNs in enhancing medical diagnostics but also underscores their role in revolutionizing healthcare through efficient and automated disease classification methodologies.

Data Science

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains.

The term "data science" has been traced back to 1974, when Peter Naur proposed it as an alternative name for computer science. In 1996, the International Federation of Classification Societies became the first conference to specifically feature data science as a topic. However, the definition was still in flux.

The term “data science” was first coined in 2008 by D.J. Patil, and Jeff Hammerbacher, the pioneer leads of data and analytics efforts at LinkedIn and Facebook. In less than a decade, it has become one of the hottest and most trending professions in the market.

Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data.

Data science can be defined as a blend of mathematics, business acumen, tools, algorithms and machine learning techniques, all of which help us in finding out the hidden insights or patterns from raw data which can be of major use in the formation of big business decisions.

EXISTING SYSTEM:

Haemorrhagic stroke poses a significant threat to human health. A cutting-edge technique called microwave-induced Thermo-acoustic tomography (MITAT) shows promise for brain imaging. However, MITAT faces challenges in transcranial brain imaging due to the complex variations in sound speed and attenuation caused by the human skull. In response, this study introduces a novel approach called deep-learning-based MITAT (DL-MITAT) to detect brain haemorrhages. The study uses a specially designed neural network called Residual Attention U-Net (ResAttU-Net), which outperforms traditional networks. Simulated training sets and images from conventional imaging methods are used. The approach is validated through experiments on bovine skull and porcine brain tissues, showing that DL-MITAT can accurately detect small haemorrhage spots while reducing false positives. The study also examines various factors of the DL-MITAT technique to reveal its strengths and limitations. Overall, the proposed ResAttU-Net-based DL-MITAT method offers promise in overcoming acoustic challenges for transcranial brain imaging, particularly in haemorrhage detection and other brain imaging applications.

Disadvantages:

- They did not implement the deployment process
- They implemented simple method. They did not Implement Convolution neural network and artificial neural network.
- They did not do image preprocessing process.
- They did not do data visualization process.
- They did not do data analysis process by using Plot and Graphs.
- They did not compared more than an architecture to getting better accuracy level.

- They did not figure out performance metrics.
- The accuracy level and performance level are low.
- They did not train the training data to computer properly

Data Scientist:

Data scientists examine which questions need answering and where to find the related data. They have business acumen and analytical skills as well as the ability to mine, clean, and present data. Businesses use data scientists to source, manage, and analyse large amounts of unstructured data.

Required Skills for a Data Scientist:

- **Programming:** Python, SQL, Scala, Java, R, MATLAB.
- **Machine Learning:** Natural Language Processing, Classification, Clustering.
- **Data Visualization:** Tableau, SAS, D3.js, Python, Java, R libraries.
- **Big data platforms:** MongoDB, Oracle, Microsoft Azure, Cloudera.

ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

Artificial intelligence (AI) is intelligence demonstrated by machines, as opposed to the natural intelligence displayed by humans or animals. Leading AI textbooks define the field as the study of "intelligent agents" any system that perceives its environment and takes actions that maximize its chance of achieving its goals. Some popular accounts use the term "artificial intelligence" to describe machines that mimic "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving", however this definition is rejected by major AI researchers.

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, and speech recognition and machine vision.

AI applications include advanced web search engines, recommendation systems (used by Youtube, Amazon and Netflix), Understanding human speech (such as Siri or Alexa), self-driving cars (e.g. Tesla), and competing at the highest level in strategic game systems (such as chess and Go). As machines become increasingly capable, tasks considered to require "intelligence" are often removed from the definition of AI, a phenomenon known as the AI effect. For instance, optical character recognition is frequently excluded from things considered to be AI, having become a routine technology.

Artificial intelligence was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism, followed by disappointment and the loss of funding (known as an "AI winter"), followed by new approaches, success and renewed funding. AI research has tried and discarded many different approaches during its lifetime, including simulating the brain, modelling human problem solving, formal logic, large databases of knowledge and imitating animal behaviour. In the first decades of the 21st century, highly mathematical statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia.

The various sub-fields of AI research are centred on particular goals and the use of particular tools. The traditional goals of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects. General intelligence (the ability to solve an arbitrary problem) is among the field's long-term goals. To solve these problems, AI researchers use versions of search and mathematical optimization, formal logic, artificial neural networks, and methods based on statistics, probability and economics. AI also draws upon computer science, psychology, linguistics, philosophy, and many other fields.

The field was founded on the assumption that human intelligence "can be so precisely described that a machine can be made to simulate it". This raises philosophical arguments about the mind and the ethics of creating artificial beings endowed with human-like intelligence. These issues have been explored by myth, fiction and philosophy since antiquity. Science fiction and futurology have also

suggested that, with its enormous potential and power, AI may become an existential risk to humanity.

As the hype around AI has accelerated, vendors have been scrambling to promote how their products and services use AI. Often what they refer to as AI is simply one component of AI, such as machine learning. AI requires a foundation of specialized hardware and software for writing and training machine learning algorithms. No one programming language is synonymous with AI, but a few, including Python, R and Java, are popular.

In general, AI systems work by ingesting large amounts of labeled training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future states. In this way, a chatbot that is fed examples of text chats can learn to produce life like exchanges with people, or an image recognition tool can learn to identify and describe objects in images by reviewing millions of examples.

AI programming focuses on three cognitive skills: learning, reasoning and self-correction.

Learning processes. This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, provide computing devices with step-by-step instructions for how to complete a specific task.

Reasoning processes. This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.

Self-correction processes. This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

AI is important because it can give enterprises insights into their operations that they may not have been aware of previously and because, in some cases, AI can perform tasks better than humans. Particularly when it comes to repetitive, detail-oriented tasks like

analyzing large numbers of legal documents to ensure relevant fields are filled in properly, AI tools often complete jobs quickly and with relatively few errors.

Artificial neural networks and deep learning artificial intelligence technologies are quickly evolving, primarily because AI processes large amounts of data much faster and makes predictions more accurately than humanly possible.

Natural Language Processing (NLP):

Natural language processing (NLP) allows machines to read and understand human language. A sufficiently powerful natural language processing system would enable natural-language user interfaces and the acquisition of knowledge directly from human-written sources, such as newswire texts. Some straightforward applications of natural language processing include information retrieval, text mining, question answering and machine translation. Many current approaches use word co-occurrence frequencies to construct syntactic representations of text. "Keyword spotting" strategies for search are popular and scalable but dumb; a search query for "dog" might only match documents with the literal word "dog" and miss a document with the word "poodle". "Lexical affinity" strategies use the occurrence of words such as "accident" to assess the sentiment of a document. Modern statistical NLP approaches can combine all these strategies as well as others, and often achieve acceptable accuracy at the page or paragraph level. Beyond semantic NLP, the ultimate goal of "narrative" NLP is to embody a full understanding of common sense reasoning. By 2019, transformer-based deep learning architectures could generate coherent text.

MACHINE LEARNING

Machine learning is to predict the future from past data. Machine learning (ML) is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of Computer Programs that can change when exposed to new data and the basics of Machine Learning, implementation of a simple machine learning algorithm using python. Process of training and prediction involves use of specialized algorithms. It feed the training data to an algorithm, and the algorithm uses this training data to give predictions on a new test data. Machine

learning can be roughly separated into three categories. There are supervised learning, unsupervised learning and reinforcement learning. Supervised learning program is both given the input data and the corresponding labelling to learn data has to be labelled by a human being beforehand. Unsupervised learning is no labels. It provided to the learning algorithm. This algorithm has to figure out the clustering of the input data. Finally, Reinforcement learning dynamically interacts with its environment and it receives positive or negative feedback to improve its performance.

Data scientists use many different kinds of machine learning algorithms to discover patterns in python that lead to actionable insights. At a high level, these different algorithms can be classified into two groups based on the way they “learn” about data to make predictions: supervised and unsupervised learning. Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function from input variables(X) to discrete output variables(y). In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, bio metric identification, document classification etc.



Supervised Machine Learning is the majority of practical machine learning uses supervised learning. Supervised learning is where have input variables (X) and an output variable (y) and use an algorithm to learn the mapping function from the input to the output is $y = f(X)$. The goal is to approximate the mapping function so well that when you have new input data (X) that you can predict the output variables (y) for that data. Techniques of Supervised Machine Learning algorithms include **logistic regression, multi-class**

classification, Decision Trees and support vector machines etc. Supervised learning requires that the data used to train the algorithm is already labelled with correct answers. Supervised learning problems can be further grouped into **Classification** problems. This problem has as goal the construction of a succinct model that can predict the value of the dependent attribute from the attribute variables.

CHAPTER 2

LITERATURE SURVEY

General

A literature review is a body of text that aims to review the critical points of current knowledge on and/or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time period.

Its ultimate goal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis.

A summary is a recap of important information about the source, but a synthesis is a re-organization, reshuffling of information. It might give a new interpretation of old material or combine new with old interpretations or it might trace the intellectual progression of the field, including major debates. Depending on the situation, the literature review may evaluate the sources and advise the reader on the most pertinent or relevant of them. Loan default trends have been long studied from a socio-economic stand point.

Most economics surveys believe in empirical modeling of these complex systems in order to be able to predict the loan default rate for a particular individual. The use of machine learning for such tasks is a trend which it is observing now. Some of the survey's to understand the past and present perspective of loan approval or not.

Review of Literature Survey

Title : Brain Tumor Classification Using Deep Learning Technique - A Comparison between Cropped, Uncropped, and Segmented Lesion Images with Different Sizes

Author : Ali Mohammad Alqudah, Hiam Alquraan , Isam Abu Qasmieh , Amin Alqudah , Wafaa Al-Sharu

Year : 2019

Deep Learning is the newest and the current trend of the machine learning field that paid a lot of the researchers' attention in the recent few years. As a proven powerful machine learning tool, deep learning was widely used in several applications for solving various complex problems that require extremely high accuracy and sensitivity, particularly in the medical field. In general, the brain tumor is one of the most common and aggressive malignant tumor diseases which is leading to a very short expected life if it is diagnosed at a higher grade. Based on that, brain tumor grading is a very critical step after detecting the tumor in order to achieve an effective treating plan. In this paper, we used Convolutional Neural Network (CNN) which is one of the most widely used deep learning architectures for classifying a dataset of 3064 T1 weighted contrast-enhanced brain MR images for grading (classifying) the brain tumors into three classes (Glioma, Meningioma, and Pituitary Tumor). The proposed CNN classifier is a powerful tool and its overall performance with an accuracy of 98.93% and sensitivity of 98.18% for the cropped lesions, while the results for the uncropped lesions are 99% accuracy and 98.52% sensitivity and the results for segmented lesion images are 97.62% for accuracy and 97.40% sensitivity.

Title : Analysis Of Brain Tumor Segmentation using Convolutional Neural Network with Magnetic Resonance Imaging

Author : B.Keerthi Samhitha, P.Yaswanth, Yellappa Gari Srinivasulu, Suja Cherukullapurath Mana and Jithina Jose

Year : 2020

Computer vision plays a vital role in human health care area. The various applications of these techniques have been playing a key role in reducing physicians judgment in diagnosis. Especially, in the case of brain related cases, it is suggested and required to take utmost care and a small error in diagnosis leads to disastrous. Although various methods like X ray, ct scan are available, mri is more reliable and danger free. High contrast images of brain tissues can be taken by using mri technique. Brain Tumor segmentation involves separation of tumors from normal brain tissues. The segmentation process helps in fetching required information for the purpose of diagnosis and what treatment has to be taken. It plays a vital role in medical image processing. The process involves risk because of abnormal form and irregularities at the boundaries. Determination of these tumors at the earlier stage helps to

give better treatment to the patients. Usually, Physicians deal with the affected people by performing manual way of segmentation which results in the absence of time management. To overcome this manual segmentation problem, we developed CNN method of segmentation procedure which performs well by taking various factors like accuracy, recall and precision into consideration. MRI is one of the conservative techniques for prior diagnostic tool of brain tumor. The necessary measures for diagnosis and treatment planning have to be taken place quickly if the tumor is malignant in order to enhance successful treatment. By applying the following method, the doctors are capable of making accurate decision and required treatment can be done in an effective manner. The computer aided diagnostic system, MRI helps in reduction of workload by doctors..

Title : Deep-learning-enabled Microwave-induced Thermoacoustic Tomography based on ResAttU-Net for Transcranial Brain Hemorrhage Detection

Author : Chenzhe Li, Zijun Xi, Gaofer Jin, Weichao Jiang, Baosheng Wang, Xiran Cai, and Xiong Wang, Member

Year : 2023

Hemorrhagic stroke is a leading threat to human's health. The fast-developing microwave-induced thermoacoustic tomography (MITAT) technique holds potential to do brain imaging. However, transcranial brain imaging based on MITAT is still challenging due to the involved huge heterogeneity in speed of sound and acoustic attenuation of human skull. This work aims to address the adverse effect of the acoustic heterogeneity using a deep-learning-based MITAT (DL-MITAT) approach for transcranial brain hemorrhage detection. Methods: We establish a new network structure, a residual attention U-Net (ResAttU-Net), for the proposed DL-MITAT technique, which exhibits improved performance as compared to some traditionally used networks. We use simulation method to build training sets and take images obtained by traditional imaging algorithms as the input of the network. Results: We present ex-vivo transcranial brain hemorrhage detection as a proof-of-concept validation. By using an 8.1-mm thick bovine skull and porcine brain tissues to perform ex-vivo experiments, we demonstrate that the trained ResAttU-Net is capable of efficiently eliminating image artifacts and accurately restoring the hemorrhage spot. It is proved that the DL-MITAT method can reliably suppress false positive rate and detect a hemorrhage spot as small as 3 mm. We also

study effects of several factors of the DL-MITAT technique to further reveal its robustness and limitations. Conclusion: The proposed ResAttU-Net-based DL-MITAT method is promising for mitigating the acoustic inhomogeneity issue and performing transcranial brain hemorrhage detection. Significance: This work provides a novel ResAttU-Net-based DL-MITAT paradigm and paves a compelling route for transcranial brain hemorrhage detection as well as other transcranial brain imaging applications.

Title : Multiple Brain Tumor Classification with Dense CNN Architecture Using Brain MRI Images

Author : Osman Özkaraca , Okan 'Ihsan Ba griaçık , Hüseyin Gürüler , Faheem Khan , Jamil Hussain ,Jawad Khan and Umm e Laila

Year : 2023

Brain MR images are the most suitable method for detecting chronic nerve diseases such as brain tumors, strokes, dementia, and multiple sclerosis. They are also used as the most sensitive method in evaluating diseases of the pituitary gland, brain vessels, eye, and inner ear organs. Many medical image analysis methods based on deep learning techniques have been proposed for health monitoring and diagnosis from brain MRI images. CNNs (Convolutional Neural Networks) are a sub-branch of deep learning and are often used to analyze visual information. Common uses include image and video recognition, suggestive systems, image classification, medical image analysis, and natural language processing. In this study, a new modular deep learning model was created to retain the existing advantages of known transfer learning methods (DenseNet, VGG16, and basic CNN architectures) in the classification process of MR images and eliminate their disadvantages. Open-source brain tumor images taken from the Kaggle database were used. For the training of the model, two types of splitting were utilized. First, 80% of the MRI image dataset was used in the training phase and 20% in the testing phase. Secondly, 10-fold cross-validation was used. When the proposed deep learning model and other known transfer learning methods were tested on the same MRI dataset, an improvement in classification performance was obtained, but an increase in processing time was observed

Title : Accurate brain tumor detection using deep convolutional neural network

Author : Md. Saikat Islam Khan , Anichur Rahman a , Tanoy Debnath

Year : 2023

Detection and Classification of a brain tumor is an important step to better understanding its mechanism. Magnetic Reasoning Imaging (MRI) is an experimental medical imaging technique that helps the radiologist find the tumor region. However, it is a time taking process and requires expertise to test the MRI images, manually. Nowadays, the advancement of Computer-assisted Diagnosis (CAD), machine learning, and deep learning in specific allow the radiologist to more reliably identify brain tumors. The traditional machine learning methods used to tackle this problem require a handcrafted feature for classification purposes. Whereas deep learning methods can be designed in a way to not require any handcrafted feature extraction while achieving accurate classification results. This paper proposes two deep learning models to identify both binary (normal and abnormal) and multiclass (meningioma, glioma, and pituitary) brain tumors. We use two publicly available datasets that include 3064 and 152 MRI images, respectively. To build our models, we first apply a 23-layers convolution neural network (CNN) to the first dataset since there is a large number of MRI images for the training purpose. However, when dealing with limited volumes of data, which is the case in the second dataset, our proposed “23-layers CNN” architecture faces overfitting problem. To address this issue, we use transfer learning and combine VGG16 architecture along with the reflection of our proposed “23 layers CNN” architecture. Finally, we compare our proposed models with those reported in the literature. Our experimental results indicate that our models achieve up to 97.8% and 100% classification accuracy for our employed datasets, respectively, exceeding all other state-of-the-art models.

Title : Multi-Classification of Brain Tumor MRI Images Using Deep Convolutional Neural Network with Fully Optimized Framework

Author : Emrah Irmak

Year : 2021

Brain tumor diagnosis and classification still rely on histopathological analysis of biopsy

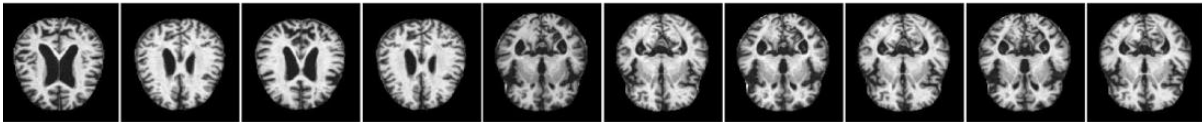
specimens today. The current method is invasive, time-consuming and prone to manual errors. These disadvantages show how essential it is to perform a fully automated method for multi-classification of brain tumors based on deep learning. This paper aims to make multiclassification of brain tumors for the early diagnosis purposes using convolutional neural network (CNN). Three different CNN models are proposed for three different classification tasks. Brain tumor detection is achieved with 99.33% accuracy using the first CNN model. The second CNN model can classify the brain tumor into five brain tumor types as normal, glioma, meningioma, pituitary and metastatic with an accuracy of 92.66%. The third CNN model can classify the brain tumors into three grades as Grade II, Grade III and Grade IV with an accuracy of 98.14%. All the important hyperparameters of CNN models are automatically designated using the grid search optimization algorithm. To the best of author's knowledge, this is the first study for multi-classification of brain tumor MRI images using CNN whose almost all hyper-parameters are tuned by the grid search optimizer. The proposed CNN models are compared with other popular state-of-the-art CNN models such as AlexNet, Inceptionv3, ResNet-50, VGG-16 and GoogleNet. Satisfactory classification results are obtained using large and publicly available clinical datasets. The proposed CNN models can be employed to assist physicians and radiologists in validating their initial screening for brain tumor multi-classification purposes

CHAPTER 3

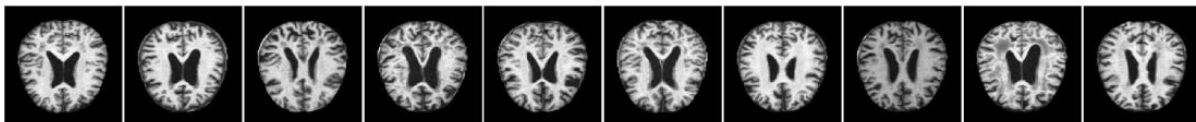
PREPARING DATASET:

This dataset is collected from a website which contains image records of features extracted, which were then split into train and test data.

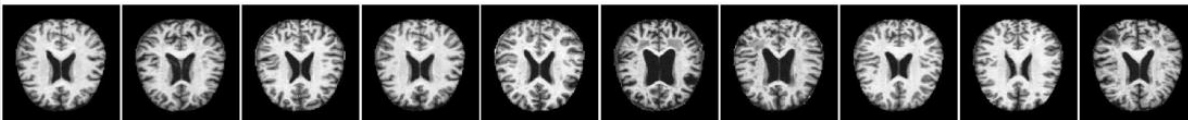
```
TRAINING DATA FOR ALZHEI_MILD_DEMENTED:
===== Images in: DATASET/TRAIN/ALZHEI_MILD_DEMENTED
Images_count : 326
Min_width : 128
Max_width : 128
Min_height : 128
Max_height : 128
```



```
TRAINING DATA FOR ALZHEI_MODERATE_DEMENTED:
===== Images in: DATASET/TRAIN/ALZHEI_MODERATE_DEMENTED
Images_count : 304
Min_width : 128
Max_width : 128
Min_height : 128
Max_height : 128
```

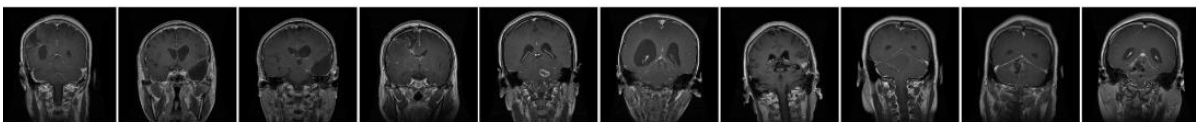


```
TRAINING DATA FOR ALZHEI_NON_DEMENTED:
===== Images in: DATASET/TRAIN/ALZHEI_NON_DEMENTED
Images_count : 319
Min_width : 128
Max_width : 128
Min_height : 128
Max_height : 128
```



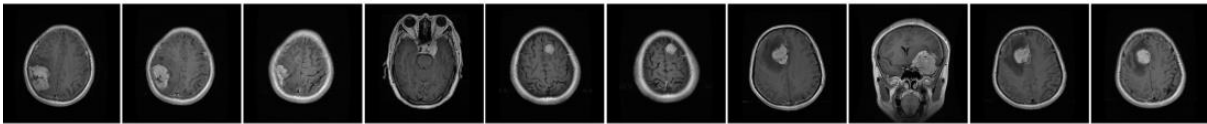
```
TRAINING DATA FOR GLIOMA:
```

```
===== Images in: DATASET/TRAIN/GLIOMA
Images_count : 319
Min_width : 512
Max_width : 512
Min_height : 512
Max_height : 512
```



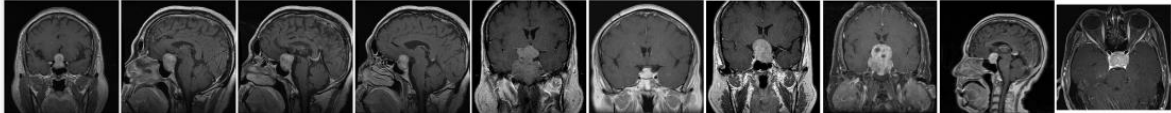
TRAINING DATA FOR MENINGIOMA:

```
===== Images in: DATASET/TRAIN/MENINGIOMA
Images_count : 319
Min_width : 200
Max_width : 1149
Min_height : 207
Max_height : 1019
```



TRAINING DATA FOR PITUITARY:

```
===== Images in: DATASET/TRAIN/PITUITARY
Images_count : 320
Min_width : 201
Max_width : 940
Min_height : 202
Max_height : 900
```



CHAPTER 4

PROPOSED SYSTEM:

The proposed system for classifying stages of brain tumor and Alzheimer's disease utilizes Convolutional Neural Network (CNN) architectures to achieve accurate and efficient diagnoses. Leveraging the power of deep learning, the system processes medical imaging data, such as MRI scans, to identify distinct patterns associated with different stages of these neurological conditions. The CNN models are trained on diverse datasets, incorporating a variety of image features and characteristics specific to each ailment. Through this approach, the system aims to enhance diagnostic precision by automatically recognizing subtle nuances indicative of varying disease stages, ultimately assisting healthcare professionals in making timely and informed treatment decisions for patients with brain tumors or Alzheimer's disease.

Advantages:

- We implemented the deployment process by using Frontend codes like Html, Css, Bootstrap and Python Framework like Django or Flask.
- We implemented image preprocessing process by using Rescale, Horizontal flip, Shear range and Zoom range.
- We implemented data analysis process by using Plot and Graphs.
- We compared more than a two architecture to getting better accuracy level.
- We figure out performance metrics value properly.
- We improved the accuracy level and performance level.
- We implemented Convolution neural network and artificial neural network concept properly.
- We did train the training data to computer properly. We implemented the test data to testing the output values.

SYSTEM STUDY

Aim:

The aim of utilizing Convolutional Neural Network (CNN) architectures for the classification of brain tumor and Alzheimer's disease stages is to develop an effective and accurate computational model for medical image analysis. By leveraging CNN's ability to automatically learn hierarchical features from image data, the objective is to enhance the classification performance in distinguishing different stages of brain tumors and Alzheimer's disease from medical imaging scans. This research aims to contribute to the field of medical diagnostics by providing a robust and automated system that can assist healthcare professionals in early and precise detection of these neurological conditions, ultimately leading to improved patient outcomes and treatment planning. The utilization of CNNs in this context is driven by their proven success in image classification tasks, offering the potential to identify intricate patterns and features in medical images that may be indicative of specific disease stages.

Objectives:

The objective of the research is to develop an efficient and accurate classification system for brain tumor and Alzheimer's disease stages using Convolutional Neural Network (CNN) architectures. This involves the design and implementation of deep learning models capable of analyzing medical imaging data, such as MRI scans, to distinguish between different stages of brain tumors and Alzheimer's disease. The primary goal is to create a robust and reliable classification framework that can assist medical professionals in early and accurate diagnosis, leading to improved patient care and treatment outcomes. The study aims to leverage the power of CNNs in automatically extracting meaningful features from complex medical images, facilitating the development of a computer-aided diagnostic tool for the efficient identification and classification of brain tumor and Alzheimer's disease stages.

Scope:

The classification of brain tumor and Alzheimer's disease stages through the utilization of Convolutional Neural Network (CNN) architectures presents a promising avenue for advancing medical diagnostics and treatment planning. By leveraging deep learning

techniques, these models have the potential to enhance the accuracy and efficiency of disease identification and staging, aiding healthcare professionals in making more informed decisions. The scope of this research extends to developing robust CNN architectures capable of extracting meaningful features from medical imaging data, enabling the automated classification of diverse stages of brain tumors and Alzheimer's disease. Implementation of such models could significantly improve early detection, prognosis, and personalized treatment strategies, ultimately contributing to advancements in neurology and patient care. Additionally, the exploration of CNNs in this context underscores the interdisciplinary collaboration between computer science and medicine, showcasing the potential for cutting-edge technology to positively impact healthcare outcomes.

Feasibility study:

Data Wrangling

In this section of the report will load in the data, check for cleanliness, and then trim and clean given dataset for analysis. Make sure that the document steps carefully and justify for cleaning decisions.

Data collection

The data set collected for predicting given data is split into Training set and Test set. Generally, 7:3 ratios are applied to split the Training set and Test set. The Data Model which was created using Random Forest, logistic, Decision tree algorithms and Support vector classifier (SVC) are applied on the Training set and based on the test result accuracy, Test set prediction is done.

Preprocessing

The data which was collected might contain missing values that may lead to inconsistency. To gain better results data need to be preprocessed so as to improve the efficiency of the algorithm. The outliers have to be removed and also variable conversion need to be done.

CHAPTER 5

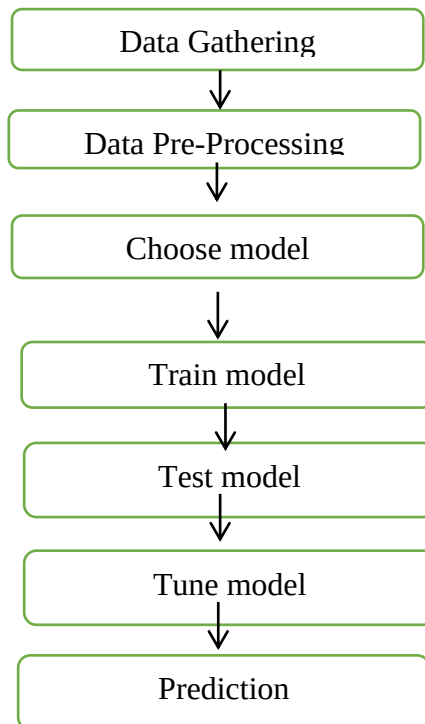
Building the classification model

The prediction of Human Air Pollution, a high accuracy prediction model is effective because of the following reasons: It provides better results in classification problem.

- It is strong in preprocessing outliers, irrelevant variables, and a mix of continuous, categorical and discrete variables.
- It produces out of bag estimate error which has proven to be unbiased in many tests and it is relatively easy to tune with.

Construction of a Predictive Model

Machine learning needs data gathering have lot of past data's. Data gathering have sufficient historical data and raw data. Before data pre-processing, raw data can't be used directly. It's used to pre-process then, what kind of algorithm with model. Training and testing this model working and predicting correctly with minimum errors. Tuned model involved by tuned time to time with improving the accuracy. Process of dataflow diagram



Project Requirements

General:

Requirements are the basic constraints that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements
2. Non-Functional requirements
3. Environment requirements
 - A. Hardware requirements
 - B. software requirements

Functional requirements:

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists requirements of a particular software system. The following details to follow the special libraries like sk-learn, pandas, numpy, matplotlib and seaborn.

Non-Functional Requirements:

Process of functional steps,

1. Problem define
2. Preparing data
3. Evaluating algorithms
4. Improving results
5. Prediction the result

Environmental Requirements:

1. Software Requirements:

Operating System : Windows 10 or later

Tool : Anaconda with Jupyter Notebook

2. Hardware requirements:

Processor	: Intel i3 or later
Hard disk	: minimum 10 GB
RAM	: minimum 4 GB

LIST OF MODULES:

- Manual architecture
- GOOGLE OF INCEPTION
- LENET
- Deployment

GOOGLE NET:

"GoogleNet," also known as "Inception-v1," is a deep learning architecture for image classification. It was developed by a team of researchers at Google, led by Christian Szegedy, and was the winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014.

The main innovation in the GoogleNet architecture is the use of the "Inception module," which incorporates multiple parallel convolutional operations of different filter sizes to capture features at various scales. This allows the network to learn both fine-grained and high-level abstract features efficiently.

Key features of the GoogleNet architecture:

Inception Module: The Inception module consists of a combination of 1x1, 3x3, and 5x5 convolutional filters, along with max-pooling operations. The idea behind this module is to allow the network to learn features at different spatial scales and capture both local and global patterns effectively. Additionally, 1x1 convolutions are used to reduce the dimensionality of the input channels, which helps in reducing computational complexity.

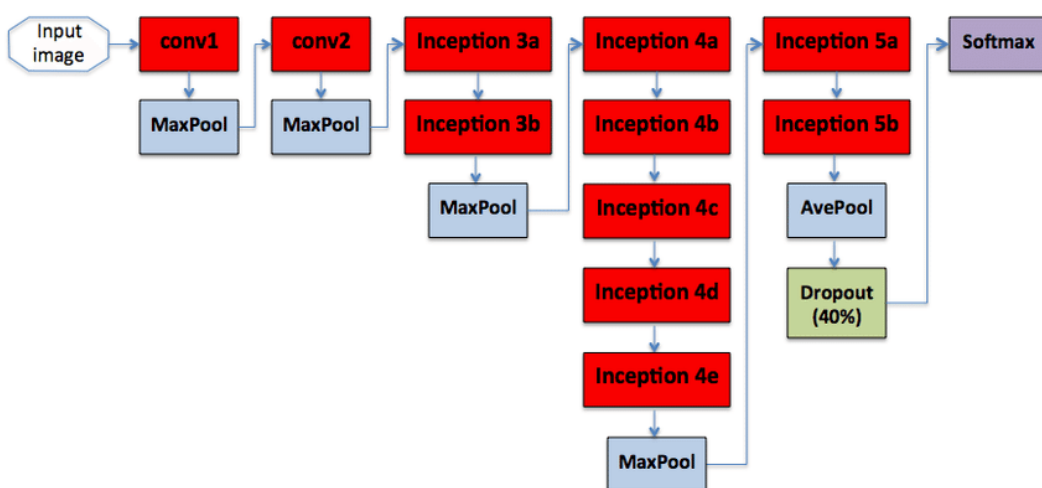
Multiple Stacked Inception Modules: GoogleNet consists of multiple stacked Inception modules to create a deep network. These modules are designed to capture increasingly

complex patterns as the network goes deeper. The inclusion of multiple Inception modules also allows the network to be computationally efficient, as it can utilize different filter sizes simultaneously.

Auxiliary Classifiers: GoogleNet includes auxiliary classifiers at intermediate layers of the network. These auxiliary classifiers serve as "side branches" and are used to combat the vanishing gradient problem during training. They add extra supervision to the network and help in better gradient flow during backpropagation.

Global Average Pooling: Instead of using fully connected layers at the end of the network, GoogleNet employs global average pooling. This operation computes the average value for each feature map in the last convolutional layer, resulting in a single vector that is used as input to the final softmax layer for classification. Global average pooling reduces the number of parameters and helps prevent overfitting.

GoogleNet was one of the early deep learning architectures to demonstrate the effectiveness of deep networks in image classification tasks. It achieved top accuracy in the ILSVRC 2014 competition while being more computationally efficient compared to other models available at the time. Since its introduction, several improved versions of the Inception architecture, such as Inception-v2, Inception-v3, and Inception-v4, have been developed by the research community.



Convolutional layers:

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are

in the network. The most important parameters are the number of kernels and the size of the kernels.

Pooling layers:

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

Dense or Fully connected layers: Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

LENET:

The LeNet architecture is a pioneering convolutional neural network (CNN) architecture developed by Yann LeCun and his colleagues in the early 1990s. It played a crucial role in the advancement of deep learning and was specifically designed for handwritten digit recognition tasks, such as recognizing digits in checks and postal addresses. LeNet laid the foundation for modern CNNs and their applications in image recognition and computer vision tasks.

The LeNet architecture consists of the following layers:

Input Layer: This layer accepts the input image, which is typically a grayscale image of a handwritten digit. The input images are usually of size 32x32 pixels.

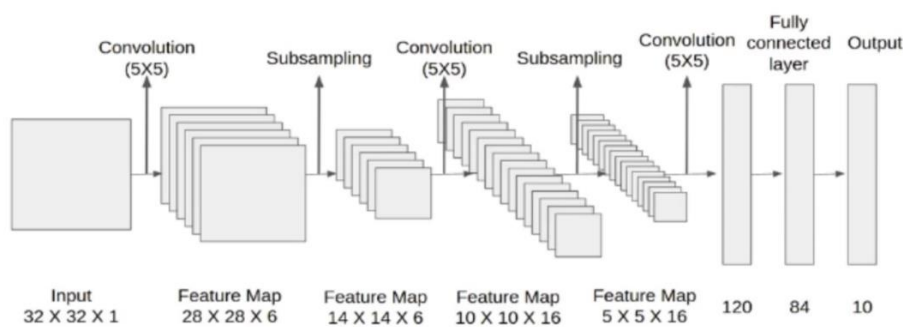
Convolutional Layers: LeNet uses two convolutional layers to extract features from the input images. Each convolutional layer applies convolutional filters (also called kernels) to the input image, capturing different patterns and features. These filters slide over the image to create feature maps.

Subsampling (Pooling) Layers: After each convolutional layer, a subsampling layer (also known as a pooling layer) is applied to reduce the spatial dimensions of the feature maps and help in retaining important information while reducing computation.

Fully Connected Layers: The subsampled feature maps are then flattened and passed through fully connected layers, which are traditional neural network layers. These layers are responsible for making classification decisions based on the extracted features.

Output Layer: The final fully connected layer produces the output of the network, which represents the predicted class probabilities. For handwritten digit recognition, this would typically involve 10 output nodes, each corresponding to a digit class (0 to 9).

LeNet was designed before the deep learning era as we know it today, so it has fewer layers compared to modern architectures like ResNet, VGG, or Inception. However, its fundamental design principles, including the use of convolutional and pooling layers, inspired the development of more complex CNN architectures. LeNet demonstrated the potential of neural networks in handling image recognition tasks and played a pivotal role in the resurgence of interest in neural networks during the early 2000s.



SOFTWARE DESCRIPTION

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system “Conda”. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. So, Anaconda distribution comes with more than 1,400 packages as well as the Conda package and virtual environment manager called Anaconda Navigator and it eliminates the need to learn to install each library independently. The open source packages can be individually installed from the Anaconda repository with the conda install command

or using the pip install command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases they can work together. Custom packages can be made using the `conda build` command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.

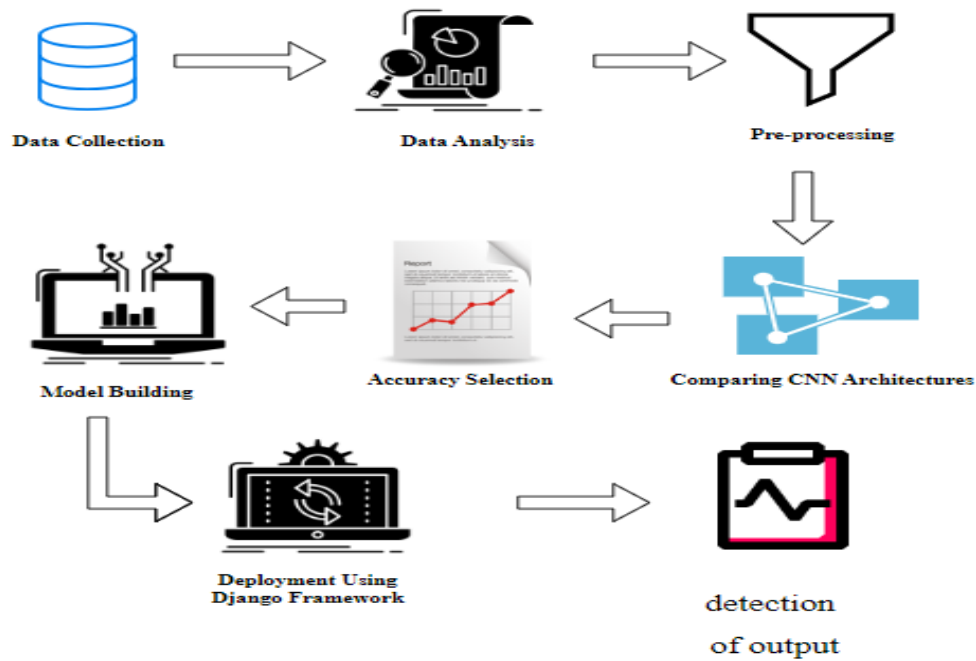
JUPYTER NOTEBOOK

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc...). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

System Architecture:



SOFTWARE DESCRIPTION

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system “Conda”. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. So, Anaconda distribution comes with more than 1,400 packages as well as the Conda package and virtual environment manager called Anaconda Navigator and it eliminates the need to learn to install each library independently. The open source packages can be individually installed from the Anaconda repository with the conda install command or using the pip install command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases they can work together. Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.

ANACONDA NAVIGATOR:

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

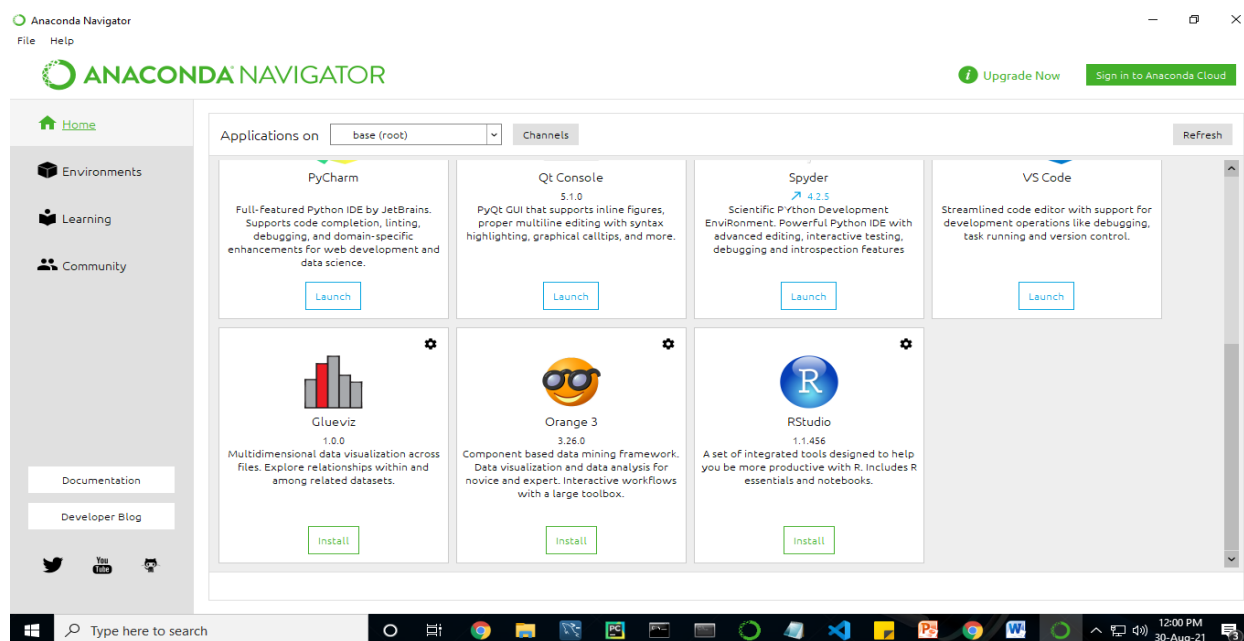
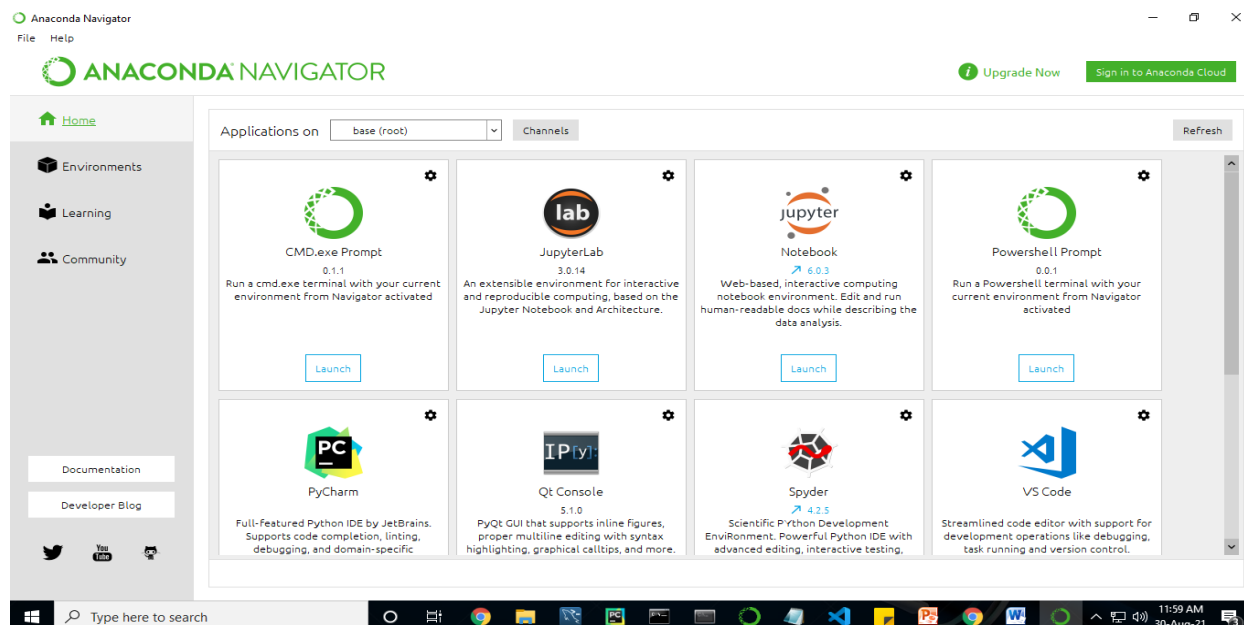
The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- Spyder
- PyCharm
- VSCode
- Glueviz
- Orange 3 App

- RStudio
- Anaconda Prompt (Windows only)
- Anaconda PowerShell (Windows only)



Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Navigator allows you to launch common Python programs and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

Anaconda comes with many built-in packages that you can easily find with `conda list` on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

JUPYTER NOTEBOOK:

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc...). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis

Installation: The easiest way to install the *Jupyter Notebook App* is installing a scientific python distribution which also includes scientific python packages. The most common distribution is called **Anaconda**

Running the Jupyter Notebook

Launching *Jupyter Notebook App*: The Jupyter Notebook App can be launched by clicking on the *Jupyter Notebook* icon installed by Anaconda in the start menu (Windows) or by typing in a terminal (*cmd* on Windows): “jupyter notebook” This will launch a new browser window (or a new tab) showing the Notebook Dashboard, a sort of control panel that allows (among other things) to select which notebook to open. When started, the Jupyter Notebook App can access only files within its start-up folder (including any sub-folder). No configuration is necessary if you place your notebooks in your home folder or subfolders. Otherwise, you need to choose a Jupyter Notebook App start-up folder which will contain all the notebooks.

Save notebooks: Modifications to the notebooks are automatically saved every few minutes. To avoid modifying the original notebook, make a copy of the notebook document (menu file -> make a copy...) and save the modifications on the copy.

Executing a notebook: Download the notebook you want to execute and put it in your notebook folder (or a sub-folder of it).

- ❖ Launch the jupyter notebook app
- ❖ In the Notebook Dashboard navigate to find the notebook: clicking on its name will open it in a new browser tab.
- ❖ Click on the menu *Help* -> *User Interface Tour* for an overview of the Jupyter Notebook App user interface.
- ❖ You can run the notebook document step-by-step (one cell a time) by pressing *shift* + *enter*.
- ❖ You can run the whole notebook in a single step by clicking on the menu *Cell* -> *Run All*.
- ❖ To restart the kernel (i.e. the computational engine), click on the menu *Kernel* -> *Restart*. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc...).

Purpose: To support interactive data science and scientific computing across all programming languages.

File Extension: An IPYNB file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

JUPYTER Notebook App: The *Jupyter Notebook App* is a server-client application that allows editing and running notebook documents via a web browser. The *Jupyter Notebook App* can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

In addition to displaying/editing/running notebook documents, the *Jupyter Notebook App* has a “Dashboard” (Notebook Dashboard), a “control panel” showing local files and allowing to open notebook documents or shutting down their kernels.

kernel: A notebook *kernel* is a “computational engine” that executes the code contained in a Notebook document. The *I python kernel*, referenced in this guide, executes python code. Kernels for many other languages exist (official kernels).

When you open a Notebook document, the associated *kernel* is automatically launched. When the notebook is *executed* (either cell-by-cell or with menu *Cell -> Run All*), the *kernel* performs the computation and produces the results. Depending on the type of computations, the *kernel* may consume significant CPU and RAM. Note that the RAM is not released until the *kernel* is shut-down

Notebook Dashboard: The *Notebook Dashboard* is the component which is shown first when you launch Jupiter Notebook App. The *Notebook Dashboard* is mainly used to open notebook documents, and to manage the running kernels (visualize and shutdown). The *Notebook Dashboard* has other features similar to a file manager, namely navigating folders and renaming/deleting files

Working Process:

- Download and install anaconda and get the most useful package for machine learning in Python.
- Load a dataset and understand its structure using statistical summaries and data visualization.
- Machine learning models, pick the best and build confidence that the accuracy is reliable.

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems. There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

- It will force you to install and start the Python interpreter (at the very least).
- It will give you a bird's eye view of how to step through a small project.
- It will give you confidence, maybe to go on to your own small projects.

When you are applying machine learning to your own datasets, you are working on a project.

A machine learning project may not be linear, but it has a number of well-known steps:

- Define Problem.
- Prepare Data.
- Evaluate Algorithms.
- Improve Results.
- Present Results.

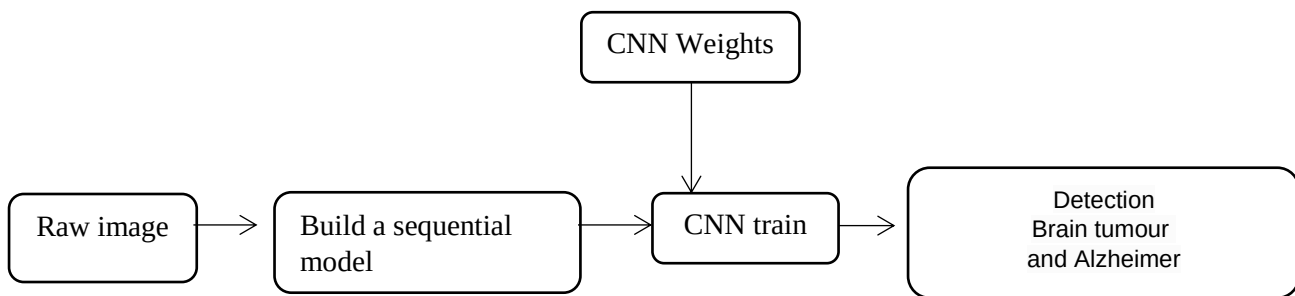
The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

Here is an overview of what we are going to cover:

1. Installing the Python anaconda platform.
2. Loading the dataset.
3. Summarizing the dataset.
4. Visualizing the dataset.
5. Evaluating some algorithms.
6. Making some predictions.

METHODOLOGY

Pre-processing and Training the model (CNN): The dataset is pre-processed such as Image reshaping, resizing and conversion to an array form. Similar processing is also done on the test image. A dataset consisting of about 4 different Brain Tumour out of which any image can be used as a test image for the software.



The train dataset is used to train the model (CNN) so that it can identify the test image and the severity it has CNN has different layers that are Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D. After the model is trained successfully, the software can identify the Paddy Leaves Disease Classification image contained in the dataset. After successful training and pre-processing, comparison of the test image and trained model takes place to predict the knee osteoarthritis

ARCHITECTURE OF CNN

CONVOLUTIONAL NEURAL NETWORK:

A Convolutional neural network (CNN) is one type of Artificial Neural Network. A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data.

Models API:

There are three ways to create Keras models:

- The Sequential model, which is very straightforward (a simple list of layers), but is limited to single-input, single-output stacks of layers (as the name gives away).
- The Functional API, which is an easy-to-use, fully-featured API that supports arbitrary model architectures. For most people and most use cases, this is what you should be using. This is the Keras "industry strength" model.
- Model subclassing, where you implement everything from scratch on your own. Use this if you have complex, out-of-the-box research use cases.

Types of Keras Models

Models in keras are available in two types:

- Keras Sequential Model
- Keras Functional API

1. Sequential Model in Keras

It allows us to create models layer by layer in sequential order. But it does not allow us to create models that have multiple inputs or outputs.

It is best for simple stack of layers which have 1 input tensor and 1 output tensor.

This model is not suited when any of the layer in the stack has multiple inputs or outputs. Even if we want non-linear topology, it is not suited.

2. Functional API in Keras

It provides more flexibility to define a model and add layers in keras. Functional API allows us to create models that have multiple input or output.

It also allows us to share these layers. In other words, we can make graphs of layers using Keras functional API.

As functional API is a data structure, it is easy to save it as a single file that helps in recreating the exact model without having the original code. Also its easy to model the graph here and access its nodes as well.

Model Subclassing in Keras

Sequential model does not allow you much flexibility to create your models. Functional API also only has a little of customization available for you. But you may create your own fully-customizable models in Keras. This is done by subclassing the Model class and implementing a call method.

`Input ()` is used to instantiate a Keras tensor.

A Keras tensor is a symbolic tensor-like object, which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model.

For instance, if `a`, `b` and `c` are Keras tensors, it becomes possible to do: `model = Model(input=[a, b], output=c)`

kernels:

Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values, the same size as the kernel. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map. These are linear transformations, each convolution is a type of affine function.

In computer vision the input is often a 3 channel RGB image. For simplicity, if we take a greyscale image that has one channel (a two dimensional matrix) and a 3x3 convolutional kernel (a two dimensional matrix). The kernel strides over the input matrix of numbers moving horizontally column by column, sliding/scanning over the first rows in the matrix containing the images pixel values. Then the kernel strides down vertically to subsequent rows. Note, the filter may stride over one or several pixels at a time, this is detailed further below.

In other non-vision applications, a one dimensional convolution may slide vertically over an input matrix.

Padding:

To handle the edge pixels there are several approaches:

- Losing the edge pixels
- Padding with zero value pixels
- Reflection padding

Reflection padding is by far the best approach, where the number of pixels needed for the convolutional kernel to process the edge pixels are added onto the outside copying the pixels from the edge of the image. For a 3x3 kernel, one pixel needs to be added around the outside, for a 7x7 kernel then three pixels would be reflected around the outside. The pixels added around each side is the dimension, halved and rounded down.

Traditionally in many research papers, the edge pixels are just ignored, which loses a small proportion of the data and this gets increasing worse if there are many deep convolutional layers. For this reason, I could not find existing diagrams to easily convey some of the points here without being misleading and confusing stride 1 convolutions with stride 2 convolutions. With padding, the output from a input of width w and height h would be width w and height h (the same as the input with a single input channel), assuming the kernel takes a stride of one pixel at a time.

Strides:

It is common to use a stride two convolution rather than a stride one convolution, where the convolutional kernel strides over 2 pixels at a time, for example our 3x3 kernel would start at position (1,1), then stride to (1,3), then to (1, 5) and so on, halving the size of the output channel/feature map, compared to the convolutional kernel taking strides of one.

With padding, the output from an input of width w , height h and depth 3 would be the ceiling of width $w/2$, height $h/2$ and depth 1, as the kernel outputs a single summed output from each stride.

Rectified Linear Unit (ReLU):

A Rectified Linear Unit is used as a non-linear activation function. A ReLU says if the value is less than zero, round it up to zero.

Normalisation:

Normalisation is the process of subtracting the mean and dividing by the standard deviation. It transforms the range of the data to be between -1 and 1 making the data use the

same scale, sometimes called Min-Max scaling. It is common to normalize the input features, standardising the data by removing the mean and scaling to unit variance. It is often important the input features are centred around zero and have variance in the same order. With some data, such as images the data is scaled so that it's range is between 0 and 1, most simply dividing the pixel values by 255.

Batch normalisation:

Batch normalisation has the benefits of helping to make a network output more stable predictions, reduce overfitting through regularisation and speeds up training by an order of magnitude.

Batch normalisation is the process of carrying normalisation within the scope activation layer of the current batch, subtracting the mean of the batch's activations and dividing by the standard deviation of the batches activations.

This is necessary as even after normalizing the input as some activations can be higher, which can cause the subsequent layers to act abnormally and makes the network less stable.

Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

Importantly, batch normalization works differently during training and during inference.

During training (i.e. when using `fit()` or when calling the layer/model with the argument `training=True`), the layer normalizes its output using the mean and standard deviation of the current batch of inputs. That is to say, for each channel being normalized, the layer returns $\gamma * (\text{batch} - \text{mean}(\text{batch})) / \sqrt{\text{var}(\text{batch}) + \text{epsilon}} + \beta$, where:

- epsilon is small constant (configurable as part of the constructor arguments)
- gamma is a learned scaling factor (initialized as 1), which can be disabled by passing `scale=False` to the constructor.

- `beta` is a learned offset factor (initialized as 0), which can be disabled by passing `center=False` to the constructor.

During inference (i.e. when using `evaluate()` or `predict()` or when calling the layer/model with the argument `training=False` (which is the default), the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training. That is to say, it returns $\gamma * (\text{batch} - \text{self.moving_mean}) / \sqrt{\text{self.moving_var} + \text{epsilon}} + \text{beta}$.

`self.moving_mean` and `self.moving_var` are non-trainable variables that are updated each time the layer is called in training mode, as such:

- `moving_mean = moving_mean * momentum + mean(batch) * (1 - momentum)`
- `moving_var = moving_var * momentum + var(batch) * (1 - momentum)`

As such, the layer will only normalize its inputs during inference *after having been trained on data that has similar statistics as the inference data*.

Arguments

- **axis:** Integer, the axis that should be normalized (typically the features axis). For instance, after a Conv2D layer with `data_format="channels_first"`, set `axis=1` in BatchNormalization.
- **momentum:** Momentum for the moving average.
- **epsilon:** Small float added to variance to avoid dividing by zero.
- **center:** If True, add offset of `beta` to normalized tensor. If False, `beta` is ignored.
- **scale:** If True, multiply by `gamma`. If False, `gamma` is not used. When the next layer is linear (also e.g. `nn.relu`), this can be disabled since the scaling will be done by the next layer.
- **beta_initializer:** Initializer for the `beta` weight.
- **gamma_initializer:** Initializer for the `gamma` weight.
- **moving_mean_initializer:** Initializer for the moving mean.
- **moving_variance_initializer:** Initializer for the moving variance.

- **beta_regularizer:** Optional regularizer for the beta weight.
- **gamma_regularizer:** Optional regularizer for the gamma weight.
- **beta_constraint:** Optional constraint for the beta weight.
- **gamma_constraint:** Optional constraint for the gamma weight.

Call arguments

- **inputs:** Input tensor (of any rank).
- **training:** Python boolean indicating whether the layer should behave in training mode or in inference mode.

Deployment:

Deploying the model in Django Framework and predicting output

In this module the trained deep learning model is converted into hierarchical data format file (.h5 file) which is then deployed in our django framework for providing better user interface and predicting the output based on its categories.

Django (Web Framework) :

DJANGO:

Django:

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

Complete:

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and up-to-date documentation.

Versatile:

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). The site you are currently reading is built with Django!

Internally, while it provides choices for almost any functionality you might want (e.g. several popular databases, templating engines, etc.), it can also be extended to use other components if needed

Secure:

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash. A password hash is a fixed-length value created by sending the password through a cryptographic hash function. Django can check if an entered password is correct by running it through the hash function and comparing the output to the stored hash value. However due to the "one-way" nature of the function, even if a stored hash value is compromised it is hard for an attacker to work out the original password.

Django enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and clickjacking (see [Website security](#) for more details of such attacks).

Scalable:

Django uses a component-based “shared-nothing” architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands (e.g. Instagram and Disqus, to name just two).

Maintainable: Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern).

Portable:

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavours of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

HTML Introduction

HTML stands for Hyper Text Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

Basic Construction of an HTML Page

These tags should be placed underneath each other **at the top of every HTML page** that you create.



`<!DOCTYPE html>` — This tag **specifies the language** you will write on the page. In this case, the language is HTML 5.

`<html>` — This tag signals that from here on we are going to write in HTML code.

`<head>` <https://html.com/tags/head/> — This is where all the **metadata for the page** goes — stuff mostly meant for search engines and other computer programs.

`<body>` <https://html.com/tags/body/> — This is where the **content of the page** goes.

Further Tags

Inside the `<head>` tag, there is one tag that is always included: `<title>`, but there are others that are just as important:

`<title>`

This is where we **insert the page name** as it will appear at the top of the browser window or tab.

<meta>

This is where information *about* the document is stored: character encoding, name (page context), description.

Head Tag

<head>

<title>My First Webpage</title>

<meta charset="UTF-8">

<meta name="description" content="This field contains information about your page. It is usually around two sentences long.">.

<meta name="author" content="Conor Sheils">

</head>

Adding Content

Next, we will make<body> tag.

The HTML <body> is where we add the content which is designed for viewing by human eyes.

This includes **text, images, tables, forms** and everything else that we see on the internet each day.

Add HTML Headings To Web Page

In HTML, [headings](#) are written in the following elements:

- <h1>
- <h2>
- <h3>
- <h4>
- <h5>

- `<h6>`

As you might have guessed `<h1>` and `<h2>` should be used for the most important titles, while the remaining tags should be used for sub-headings and less important text.

Search engine bots use this order when deciphering which information is most important on a page.

Creating Your Heading

Let's try it out. On a new line in the HTML editor, type:

```
<h1> Welcome To My Page </h1>
```

And hit save. We will save this file as "index.html" in a new folder called "my webpage."

Add Text In HTML

Adding text to our HTML page is simple using an element opened with the tag `<p>` which **creates a new paragraph**. We place all of our regular text inside the element `<p>`.

When we write text in HTML, we also have a number of other elements we can use **to control the text or make it appear in a certain way**.

Add Links In HTML

As you may have noticed, the internet is made up of lots of [links](#).

Almost everything you click on while surfing the web is a link **takes you to another page** within the website you are visiting or to an external site.

Links are included in an attribute opened by the `<a>` tag. This element is the first that we've met which uses an attribute and so it **looks different to previously mentioned tags**.

```
<a href=http://www.google.com>Google</a>
```

Image Tag

In today's modern digital world, images are everything. The `<img>` tag has everything you need to display images on your site. Much like the `<a>` anchor element, `<img>` also contains an attribute.

The attribute features information for your computer regarding the source, height, width and alt text of the image

```

```

CSS

CSS stands for Cascading Style Sheets. It is the language for describing the presentation of Web pages, including colours, layout, and fonts, thus making our web pages presentable to the users. CSS is designed to make style sheets for the web. It is independent of HTML and can be used with any XML-based markup language. Now let's try to break the acronym:

- Cascading: Falling of Styles
- Style: Adding designs/Styling our HTML tags
- Sheets: Writing our style in different documents

CSS Syntax

Selector {

Property 1 : value;

Property 2 : value;

Property 3 : value;

}

For example:

h1

{

Color: red;

Text-align: center;

}

```
#unique
```

```
{
```

```
color: green;
```

```
}
```

- Selector: selects the element you want to target
- Always remains the same whether we apply internal or external styling
- There are few basic selectors like tags, id's, and classes
- All forms this key-value pair
- Keys: properties(attributes) like color, font-size, background, width, height,etc
- Value: values associated with these properties

CSS Comment

- Comments don't render on the browser
- Helps to understand our code better and makes it readable.
- Helps to debug our code
- Two ways to comment:
 - Single line

CSS How-To

- There are 3 ways to write CSS in our HTML file.
 - Inline CSS
 - Internal CSS
 - External CSS
- Priority order
 - Inline > Internal > External

Inline CSS

- Before CSS this was the only way to apply styles

- Not an efficient way to write as it has a lot of redundancy
- Self-contained
- Uniquely applied on each element
- The idea of separation of concerns was lost
- Example:

```
<h3 style = “color:red”> Have a great day </h3>
```

```
<p style = “color:green”> I did this, I did that </p>
```

Internal CSS

- With the help of style tag, we can apply styles within the HTML file
- Redundancy is removed
- But the idea of separation of concerns still lost
- Uniquely applied on a single document
- Example:

```
<style>
```

```
H1{
```

```
Color:red;
```

```
}
```

```
</style>
```

```
<h3> Have a great day </h3>
```

External CSS

- With the help of <link> tag in the head tag, we can apply styles
- Reference is added
- File saved with .css extension
- Redundancy is removed

- The idea of separation of concerns is maintained
- Uniquely applied to each document
- Example:

```
<head>
```

```
<link rel= “stylesheet” type= “text/css” href= “name of the CSS file”>
```

```
</head>
```

```
h1{
```

```
color:red;  //.css file
```

```
}
```

CSS Selectors

- The selector is used to target elements and apply CSS
- Three simple selectors
 - Element Selector
 - Id Selector
 - Class Selector
- Priority of Selectors

CSS Colors

- There are different colouring schemes in CSS
- **RGB**-This starts with RGB and takes 3 parameter
- **HEX**-Hex code starts with # and comprises of 6 numbers which are further divided into 3 sets
- **RGBA**-This starts with RGB and takes 4 parameter

CSS Background

- There are different ways by which CSS can have an effect on HTML elements
- Few of them are as follows:
 - Color – used to set the color of the background
 - Repeat – used to determine if the image has to repeat or not and if it is repeating then how it should do that
 - Image – used to set an image as the background
 - Position – used to determine the position of the image
 - Attachment – It basically helps in controlling the mechanism of scrolling.

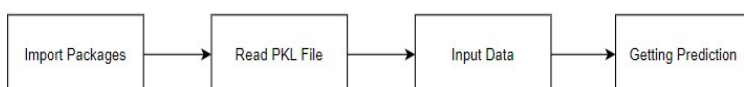
CSS BoxModel

- Every element in CSS can be represented using the BOX model
- It allows us to add a border and define space between the content
- It helps the developer to develop and manipulate the elements
- It consists of 4 edges
 - Content edge – It comprises of the actual content
 - Padding edge – It lies in between content and border edge
 - Border edge – Padding is followed by the border edge
 - Margin edge – It is an outside border and controls the margin of the element

Deploying the model predicting output

In this module the trained machine learning model is converted into pickle data format file (.pkl file) which is then deployed for providing better user interface and predicting the output of Human Stress.

MODULE DIAGRAM



input : data values , output : predicting output

CODING:

MODULE– 1

MANUAL NET ARCHITECTURE

```
import warnings

warnings.filterwarnings('ignore')

import os

import glob

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from PIL import Image

from tensorflow.keras.layers import Convolution2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Activation

from keras.callbacks import ModelCheckpoint

import matplotlib.pyplot as plt

ALZHEI_MILD_DEMENTED = 'DATASET/TRAIN/ALZHEI_MILD_DEMENTED'

ALZHEI_MODERATE_DEMENTED =

'DATASET/TRAIN/ALZHEI_MODERATE_DEMENTED'
```

```
ALZHEI_NON_DEMENTED = 'DATASET/TRAIN/ALZHEI_NON_DEMENTED'
```

```
GLIOMA = 'DATASET/TRAIN/GLIOMA'
```

```
MENINGIOMA = 'DATASET/TRAIN/MENINGIOMA'
```

```
NO_TUMOR = 'DATASET/TRAIN/NO_TUMOR'
```

```
PITUITARY = 'DATASET/TRAIN/PITUITARY'
```

```
def plot_images(item_dir, n=6):
```

```
    all_item_dir = os.listdir(item_dir)
```

```
    item_files = [os.path.join(item_dir, file) for file in all_item_dir][:n]
```

```
    plt.figure(figsize=(80, 40))
```

```
    for idx, img_path in enumerate(item_files):
```

```
        plt.subplot(3, n, idx+1)
```

```
        img = plt.imread(img_path)
```

```
        plt.imshow(img, cmap='gray')
```

```
        plt.axis('off')
```

```
    plt.tight_layout()
```

```
def image_details_print(data,path):
```

```
    print('===== Images in: ', path)
```

```
    for key,values in data.items():
```

```
        print(key,':\t', values)
```

```
def images_details(path):
```

```
    files=[f for f in glob.glob(path + "**/*.*", recursive=True)]
```

```

data={}

data['Images_count']=len(files)

data['Min_width']=10**100

data['Max_width']=0

data['Min_height']=10**100

data['Max_height']=0

for f in files:

    img=Image.open(f)

    width,height=img.size

    data['Min_width']=min(width,data['Min_width'])

    data['Max_width']=max(width, data['Max_width'])

    data['Min_height']=min(height, data['Min_height'])

    data['Max_height']=max(height, data['Max_height'])

    image_details_print(data,path)

print("TRAINING DATA FOR ALZHEI_MILD_DEMENTED:")

images_details(ALZHEI_MILD_DEMENTED)

print("")

plot_images(ALZHEI_MILD_DEMENTED, 10)

print("")

print("TRAINING DATA FOR ALZHEI_MODERATE_DEMENTED:")

print("")

```

```
images_details(ALZHEI_MODERATE_DEMENTED)

print("")

plot_images(ALZHEI_MODERATE_DEMENTED, 10)

print("")

print("TRAINING DATA FOR ALZHEI_NON_DEMENTED:")

print("")

images_details(ALZHEI_NON_DEMENTED)

print("")

plot_images(ALZHEI_NON_DEMENTED, 10)

print("")

print("TRAINING DATA FOR GLIOMA:")

print("")

images_details(GLIOMA)

print("")

plot_images(GLIOMA, 10)

print("")

print("TRAINING DATA FOR MENINGIOMA:")

print("")

images_details(MENINGIOMA)

print("")

plot_images(MENINGIOMA, 10)
```

```

print("")

print("TRAINING DATA FOR NO_TUMOR:")

print("")

images_details(NO_TUMOR)

print("")

plot_images(NO_TUMOR, 10)

print("")

print("TRAINING DATA FOR PITUITARY:")

print("")

images_details(PITUITARY)

print("")

plot_images(PITUITARY, 10)

train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)

training_set=train_datagen.flow_from_directory('dataset/train',target_size=(224,224),batch_size=32,class_mode='categorical')

test_datagen=ImageDataGenerator(rescale=1./255)

test_set=test_datagen.flow_from_directory('dataset/test',target_size=(224,224),batch_size=32,class_mode='categorical')

Classifier=Sequential()

Classifier.add(Convolution2D(32,(3,3),input_shape=(224,224,3),activation='relu'))

Classifier.add(MaxPooling2D(pool_size=(2,2)))

```

```

Classifier.add(Flatten())

Classifier.add(Dense(38, activation='relu'))

Classifier.add(Dense(7, activation='softmax'))

Classifier.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy']
)

model_path = "MANUAL.h5"

callbacks = [

    ModelCheckpoint(model_path, monitor='accuracy', verbose=1, save_best_only=True)

]

epochs = 10

batch_size = 512

#### Fitting the model

history = Classifier.fit(

    training_set, steps_per_epoch=training_set.samples // batch_size,

    epochs=epochs,

    validation_data=test_set,validation_steps=test_set.samples // batch_size,

    callbacks=callbacks)

import matplotlib.pyplot as plt

def graph():

    #Plot training & validation accuracy values

    plt.plot(history.history['accuracy'])

```

```
plt.plot(history.history['val_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()
```

```
graph()
```

```
import matplotlib.pyplot as plt
```

```
def graph():
```

```
    plt.plot(history.history['loss'])

    plt.plot(history.history['val_loss'])

    plt.title('Model loss')

    plt.ylabel('Loss')

    plt.xlabel('Epoch')

    plt.legend(['Train', 'Test'], loc='upper left')

    plt.show()
```

```
graph()
```

Module – 2

```
## GOOGLE OR INCEPTION NET ARCHITECTURE
```

```
## GOOGLE OR INCEPTION NET ARCHITECTURE
```

```
import warnings
```

```
warnings.filterwarnings('ignore')

import tensorflow

import tensorflow as tf

print(tf.__version__)

import keras

import keras.backend as K

from keras.models import Model

from keras.layers import Input, Dense, Conv2D, Conv3D, DepthwiseConv2D,
SeparableConv2D, Conv3DTranspose

from keras.layers import Flatten, MaxPool2D, AvgPool2D, GlobalAvgPool2D,
UpSampling2D, BatchNormalization

from keras.layers import Concatenate, Add, Dropout, ReLU, Lambda, Activation,
LeakyReLU, PReLU

from time import time

import numpy as np

from keras.callbacks import ModelCheckpoint

from tensorflow.keras.callbacks import EarlyStopping

import warnings

warnings.filterwarnings('ignore')

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_fli
p=True,validation_split = 0.2)
```

```

train_data=train.flow_from_directory(directory =
'DATASET/TRAIN',target_size=(224,224),

    batch_size=32,class_mode='categorical')

test=ImageDataGenerator(rescale=1./255)

test_data=test.flow_from_directory(directory = 'DATASET/TEST',target_size=(224,224),

    batch_size=32,class_mode='categorical')

def googlenet(input_shape, n_classes):

    def inception_block(x, f):

        t1 = Conv2D(f[0], 1, activation='relu')(x)

        t2 = Conv2D(f[1], 1, activation='relu')(x)

        t2 = Conv2D(f[2], 3, padding='same', activation='relu')(t2)

        t3 = Conv2D(f[3], 1, activation='relu')(x)

        t3 = Conv2D(f[4], 5, padding='same', activation='relu')(t3)

        t4 = MaxPool2D(3, 1, padding='same')(x)

        t4 = Conv2D(f[5], 1, activation='relu')(t4)

        output = Concatenate()([t1, t2, t3, t4])

        return output

    input = Input(input_shape)

    x = Conv2D(64, 7, strides=2, padding='same', activation='relu')(input)

    x = MaxPool2D(3, strides=2, padding='same')(x)

    x = Conv2D(64, 1, activation='relu')(x)

```

```

x = Conv2D(192, 3, padding='same', activation='relu')(x)

x = MaxPool2D(3, strides=2)(x)

x = inception_block(x, [64, 96, 128, 16, 32, 32])

x = inception_block(x, [128, 128, 192, 32, 96, 64])

x = MaxPool2D(3, strides=2, padding='same')(x)

x = inception_block(x, [192, 96, 208, 16, 48, 64])

x = inception_block(x, [160, 112, 224, 24, 64, 64])

x = inception_block(x, [128, 128, 256, 24, 64, 64])

x = inception_block(x, [112, 144, 288, 32, 64, 64])

x = inception_block(x, [256, 160, 320, 32, 128, 128])

x = MaxPool2D(3, strides=2, padding='same')(x)

x = inception_block(x, [256, 160, 320, 32, 128, 128])

x = inception_block(x, [384, 192, 384, 48, 128, 128])

x = AvgPool2D(7, strides=1)(x)

x = Dropout(0.4)(x)

x = Flatten()(x)

output = Dense(n_classes, activation='softmax')(x)

model = Model(input, output)

model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy',tensorflow.keras.metrics.Precision()])

return model

```

```

input_shape = 224, 224, 3

n_classes = 7

K.clear_session()

model = googlenet(input_shape, n_classes)

model.summary()

model_path = "GOOGLE.h5"

from keras.callbacks import ModelCheckpoint

M = ModelCheckpoint(model_path, monitor='accuracy', verbose=1, save_best_only=True)

epochs = 100

batch_size = 512

#### Fitting the model

history = model.fit(

    train_data, steps_per_epoch=train_data.samples // batch_size,

    epochs=epochs,

    validation_data=test_data, validation_steps=test_data.samples // batch_size,

    callbacks=[M])

history.history.keys()

import matplotlib.pyplot as plt

import numpy as np

plt.figure(figsize=(20, 8))

plt.plot(history.history['accuracy'])

```

```

for i in range(epochs):

    if i%5 == 0:
plt.annotate(np.round(history.history['accuracy'][i]*100,2),xy=(i,history.history['accuracy'][i]))

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.show()

plt.figure(figsize=(20, 8))

plt.plot(history.history['loss'])

for i in range(epochs):

    if i%5 == 0:

plt.annotate(np.round(history.history['loss'][i]*100,2),xy=(i,history.history['loss'][i]))

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.show()

```

Module – 3

LENET ARCHITECTURE

```

import warnings

warnings.filterwarnings('ignore')

import tensorflow

```

```

import tensorflow as tf

print(tf.__version__)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import Convolution2D

from tensorflow.keras.layers import MaxPool2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Activation

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,validation_split = 0.2)

train_data=train.flow_from_directory(directory =
'DATASET/TRAIN',target_size=(224,224),

                                batch_size=32,class_mode='categorical')

test=ImageDataGenerator(rescale=1./255)

test_data=test.flow_from_directory(directory = 'DATASET/TEST',target_size=(224,224),

                                batch_size=32,class_mode='categorical')

MODEL=Sequential()

```

```

MODEL.add(Convolution2D(filters=32, kernel_size=(3,3), strides=(3,3),
input_shape=(224,224,3), padding='valid', activation='relu'))

MODEL.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))

MODEL.add(Convolution2D(filters=128, kernel_size=(3,3), strides=(3,3),
padding='valid', activation='relu'))

MODEL.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))

MODEL.add(Flatten())

MODEL.add(Dense(256, activation='relu'))

MODEL.add(Dense(7, activation='softmax'))

OPT = tensorflow.keras.optimizers.Adam(0.001)

MODEL.compile(optimizer=OPT,loss='categorical_crossentropy',metrics=["accuracy",
tensorflow.keras.metrics.Precision(), tensorflow.keras.metrics.Recall()])

MODEL.summary()

model_path = "LENET.h5"

from tensorflow.keras.callbacks import ModelCheckpoint

M = ModelCheckpoint(model_path, monitor='accuracy', verbose=1, save_best_only=True,
mode='max')

epochs = 200

batch_size = 64

WORKING = MODEL.fit_generator(

    train_data, steps_per_epoch=train_data.samples // batch_size,

    epochs=epochs,

```

```
validation_data=test_data,validation_steps=test_data.samples // batch_size,
```

```
callbacks=[M])
```

```
WORKING.history.keys()
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
plt.figure(figsize=(280, 8))
```

```
plt.plot(WORKING.history['accuracy'])
```

```
for i in range(epochs):
```

```
    if i%5 == 0:
```

```
plt.annotate(np.round(WORKING.history['accuracy'][i]*100,2),xy=(i,WORKING.history['accuracy'][i]))
```

```
plt.title('Model accuracy')
```

```
plt.ylabel('Accuracy')
```

```
plt.xlabel('Epoch')
```

```
plt.show()
```

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(20, 8))
```

```
plt.plot(WORKING.history['loss'])
```

```
for i in range(epochs):
```

```
    if i%5 == 0:
```

```
plt.annotate(np.round(WORKING.history['loss'][i]*100,2),xy=(i,WORKING.history['loss'][i]))
```

```
plt.title('Model Loss')
```

```
plt.ylabel('Loss')
```

```
plt.xlabel('Epoch')
```

```
plt.show()
```

Module – 4

```
from django.shortcuts import render, redirect
```

```
from . models import UserPredictModel, UserPersonalModel
```

```
from . forms import UserPersonalForm, UserPredictForm, UserRegisterForm
```

```
from django.contrib.auth import authenticate, login,logout
```

```
from django.contrib import messages
```

```
import numpy as np
```

```
from tensorflow import keras
```

```
from PIL import Image, ImageOps
```

```
from . import forms
```

```
def Home1(request):
```

```
    return render(request, 'Home1.html')
```

```
def Register(request):
```

```
    form = UserRegisterForm()
```

```
    if request.method == 'POST':
```

```
        form = UserRegisterForm(request.POST)
```

```
        if form.is_valid():
```

```
            form.save()
```

```
            user = form.cleaned_data.get('username')
```

```

        messages.success(request, 'Account was successfully created. ' + user)

        return redirect('Login')

context = {'form':form}

return render(request, 'Register.html', context)

def Login(request):

    if request.method == 'POST':

        username = request.POST.get('username')

        password = request.POST.get('password')

        user = authenticate(username=username, password=password)

        if user is not None:

            login(request, user)

            return redirect('Home2')

        else:

            messages.info(request, 'Username OR Password incorrect')

context = {}

return render(request,'Login.html', context)

def Home2(request):

    return render(request, 'Home2.html')

def Logout(request):

    logout(request)

    return redirect('Home1')

def Info(request):

    if request.method == 'POST':

        fieldss = ['firstname','lastname','age','address','phone','city','state','country']

```

```

form = UserPersonalForm(request.POST)

if form.is_valid():

    print('Saving data in Form')

    form.save()

return render(request, 'Home2.html', {'form':form})

else:

    print('Else working')

    form = UserPersonalForm(request.POST)

    return render(request, 'Info.html', {'form':form})

def Deploy(request):

    print("HI")

    if request.method == "POST":

        form = forms.UserPredictForm(files=request.FILES)

        if form.is_valid():

            print('HIFORM')

            form.save()

            obj = form.instance

            result1 = UserPredictModel.objects.latest('id')

models = keras.models.load_model('C:/Users/SPIRO-11/Desktop/ITPDL13-
FINAL/ITPDL13-FINAL
CODING/DEPLOYMENT/PROJECT/APP/MOBILENET.h5')

data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

image = Image.open("C:/Users/SPIRO-11/Desktop/ITPDL13-
FINAL/ITPDL13-FINAL CODING/DEPLOYMENT/PROJECT/media/" +
str(result1)).convert("RGB")

```

```

size = (224, 224)

image = ImageOps.fit(image, size, Image.ANTIALIAS)

image_array = np.asarray(image)

normalized_image_array = (image_array.astype(np.float32) / 127.0) - 1

data[0] = normalized_image_array

classes = ['glioma','meningioma','notumor','pituitary']

prediction = models.predict(data)

idd = np.argmax(prediction)

a = (classes[idd])

if a == 'glioma':

    a = 'THE GLIOMA TYPE OF BRAIN TUMOR AFFECTED IN THIS
IMAGE. THIS BRAIN TUMOR STAGE 1'

elif a == 'meningioma':

    a = 'THE MENINGIOMA TYPE OF BRAIN TUMOR AFFECTED IN
THIS IMAGE. THIS BRAIN TUMOR STAGE 2'

elif a == 'notumor':

    a = 'THE NONE TYPE OF BRAIN TUMOR AFFECTED. THIS IS
NORMAL OR HEALTHY CONDITIONS.'

elif a == 'pituitary':

    a = 'THE PITUITARY TYPE OF BRAIN TUMOR AFFECTED IN THIS
IMAGE. THIS BRAIN TUMOR STAGE 3'

else:

    a = 'WRONG INPUT'

data = UserPredictModel.objects.latest('id')

data.label = a

```

```

    data.save()

    return render(request, 'Deploy.html',{'form':form,'obj':obj,'predict':a})

else:

    form = forms.UserPredictForm()

    return render(request, 'Deploy.html',{'form':form})

def Problem_Statement(request):

    return render(request,'ProblemStatement.html')

def Team(request):

    return render(request,'Team.html')

def Result(request):

    return render(request,'Result.html')

def Database1(request):

    models = UserPredictModel.objects.all()

    return render(request, 'Database1.html', {'models':models})

def Database2(request):

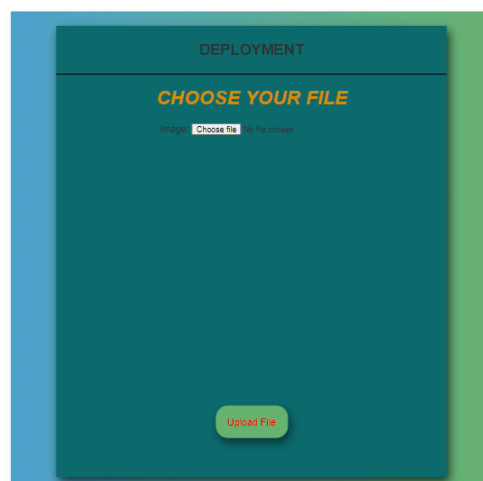
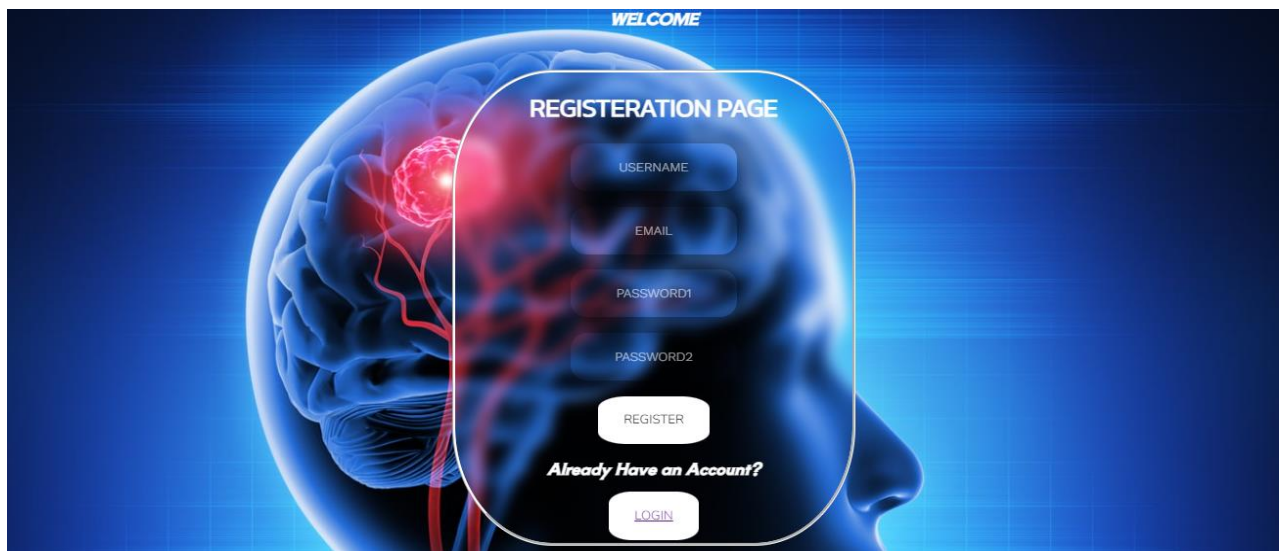
    models = UserPersonalModel.objects.all()

    return render(request, 'Database2.html', {'models':models})

```

CHAPTER 6

OUTPUT:



CHAPTER 7

Conclusion:

In conclusion, the project focused on enhancing brain tumor classification through the implementation of Convolutional Neural Network (CNN) architectures. By leveraging advanced deep learning techniques, the model demonstrated significant strides in accurately identifying and categorizing brain tumors from medical imaging data. The utilization of CNNs proved instrumental in extracting intricate features and patterns, enabling more precise and efficient tumor classification. This research contributes to the ongoing efforts to improve diagnostic accuracy and streamline the decision-making process in the field of medical imaging, ultimately advancing the prospects for early detection and treatment of brain tumors. The success of this project underscores the potential of CNN architectures in revolutionizing medical image analysis and underscores their importance in the realm of healthcare innovation.

Ascertaining Brain Diseases Using Deep Learning Techniques

Percy Paulin J

Department of Computer Science and Engineering

Sathyabama Institute of Science and Technology

Chennai, India

percypaul217@gmail.com

V. Ulagamuthalvi

Department of Computer Science and Engineering

Sathyabama Institute of Science and Technology

Chennai, India

ulagamuthalvi.cse@gmail.com

Abstract — An original methodology for the characterization of cerebrum cancers and Alzheimer's sickness stages using a Convolutional Brain Organization (CNN) design has been proposed. Utilizing the force of profound learning, our model means to precisely recognize various phases of these neurological circumstances by dissecting clinical imaging information, for example, X-ray filters. CNN engineering is intended to gain and concentrate multifaceted highlights from the imaging information, empowering hearty order consequently. The proposed procedure not only adds to the early and exact analysis of mind growths and Alzheimer's infection but additionally for upgrading how we might interpret the illness movement. The reconciliation of cutting-edge brain network procedures in clinical picture examination grandstands the potential for working on analytic devices in nervous system science, at last prompting more successful treatment techniques and better understanding results.

Keywords - Convolutional Neural Networks, CNN, Brain Tumor, Medical Imaging, Magnetic Resonance Imaging (MRI), Tumor Detection, Tumor Classification, Healthcare, Neuro-oncology

I. INTRODUCTION

The characterization of brain cancer and Alzheimer's illness stages using Convolutional Brain Organization (CNN) structures addresses a critical convergence of clinical imaging and high-level profound learning procedures. This imaginative methodology uses the abilities of CNNs to consequently extricate progressive elements from complex clinical pictures, working with the exact distinguishing proof and order of obsessive circumstances. Because of mind cancer order, CNNs dissect primary anomalies in neuroimaging information,

supporting the early identification and portrayal of growth. Likewise, for Alzheimer's sickness arranges, these models decipher designs in cerebrum sweeps to order the seriousness of the illness, giving significant bits of knowledge to opportune mediation and customized therapy methodologies. This exploration not only grandstands the potential for CNNs in improving clinical diagnostics but also highlights their job in reforming medical services through proficient and computerized sickness arrangement strategies.

Data science is a field that uses sensible procedures, cycles, and systems to isolate data and encounters from coordinated and formless data and relate data and critical pieces of information shape data over a broad extent of use scops. The term "data science" was tracked back to 1974, when Peter Naur put forward it as an alternative name for programming. In the year 1996, the Worldwide Association of Request Social Orders transformed into the chief get-together to expressly feature data science as a subject.

AI refers to the knowledge displayed by machines as opposed to the natural insight showed by humans or other living things. Textbooks on driving artificial intelligence describe the field as the research of "smart specialists"; any system which detects its environment and takes action increases the probability that it will achieve its goals. A few well-known narratives use the term "man-made reasoning" to describe computers that mimic "mental" skills that people relate to the human psyche, such as "learning" and "critical thinking," but popular computer scientists reject this interpretation.

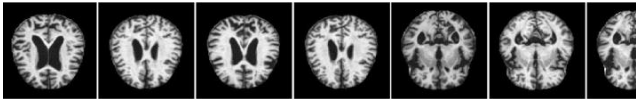
Information researchers use a variety of AI algorithms to identify Python designs that result in significant discoveries. Without a doubt, these various computations could be classified into 2 groups based on how they "learn" about data to create predictions: guided

and unassisted learning. The most popular method for predicting the class of given data that is of interest is order. Classes are sometimes referred to as classifications, targets, or marks. Description The task of estimating a planning capability from input variables (X) to discrete result variables (y) is known as predictive displaying. Characterization in AI and measurements is a directed learning technique where the computer program learns from the data it receives and then utilizes this knowledge to order novel perception. It is possible for this informational index to be multi-class as well as bi-class (e.g., to identify whether the email is spam or not, or to determine if the person is male or female). Order issues include, but are not limited to, acknowledgment of discourse, penmanship, biometric distinguishing proof, report characterization, and so forth.

II. PREPARING DATASET.

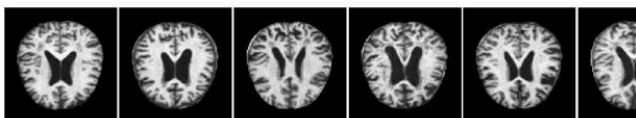
The data is collected from the Kaggle which has image records of features extracted. The intention of this information base is to figure whether a person has cerebrum infection, taking into account unequivocal intelligent evaluations related to the dataset. A few objectives were set on the decision of these events from a more significant informational grouping. The various factors found by this process are brain tumor, Alzheimer, glioma, meningioma.

```
TRAINING DATA FOR ALZHEI_HILD_DEMENTED:
===== Images in: DATASET/TRAIN/ALZHEI_HILD_DEMENTED
Images_count : 326
Min_width : 128
Max_width : 128
Min_height : 128
Max_height : 128
```



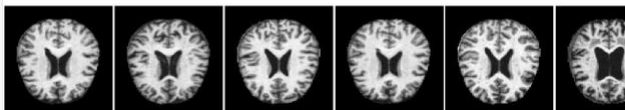
```
TRAINING DATA FOR ALZHEI_MODERATE_DEMENTED:
```

```
===== Images in: DATASET/TRAIN/ALZHEI_MODERATE_DEMENTED
Images_count : 304
Min_width : 128
Max_width : 128
Min_height : 128
Max_height : 128
```



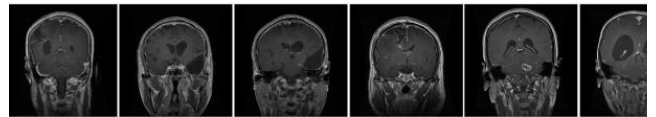
```
TRAINING DATA FOR ALZHEI_NON_DEMENTED:
```

```
===== Images in: DATASET/TRAIN/ALZHEI_NON_DEMENTED
Images_count : 319
Min_width : 128
Max_width : 128
Min_height : 128
Max_height : 128
```



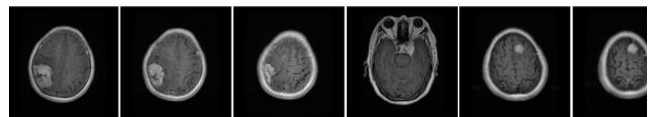
```
TRAINING DATA FOR GLIOMA:
```

```
===== Images in: DATASET/TRAIN/GLIOMA
Images_count : 319
Min_width : 512
Max_width : 512
Min_height : 512
Max_height : 512
```



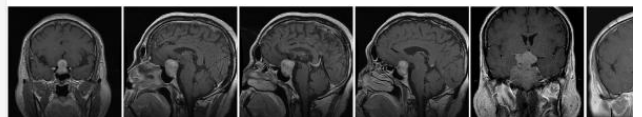
```
TRAINING DATA FOR MENINGIOMA:
```

```
===== Images in: DATASET/TRAIN/MENINGIOMA
Images_count : 319
Min_width : 200
Max_width : 1149
Min_height : 207
Max_height : 1019
```



```
TRAINING DATA FOR PITUITARY:
```

```
===== Images in: DATASET/TRAIN/PITUITARY
Images_count : 320
Min_width : 201
Max_width : 940
Min_height : 202
Max_height : 900
```



III. PROBLEM DESCRIPTION

The existing system faces significant limitations, primarily confined to basic database operations for patient images without the integration of advanced technologies like CNN and ANN. The absence of image preprocessing further impedes the system's effectiveness. Its implementation lacks sophistication, neglecting deployment procedures and relying on a simplistic approach.

Critical processes such as image preprocessing, data visualization, and analysis with plots and graphs are omitted. The system falls short in comparing different architectures, resulting in suboptimal accuracy and performance levels. Essential metrics for evaluation are also lacking. In summary, the system's deficiencies, ranging from the absence of advanced technologies and preprocessing to the lack of comprehensive analysis, contribute to its low accuracy and performance. A targeted solution must address these issues for improved system efficacy.

IV. PROPOSED SYSTEM

The proposed framework for ordering phases of cerebrum cancer and Alzheimer's infection uses Convolutional Brain Organization (CNN) designs to accomplish precise and proficient conclusions. Utilizing the force of profound learning, the framework processes clinical imaging information, for example, X-ray checks, to distinguish examples related to various phases of these neurological circumstances. The CNN models are

prepared on different datasets, integrating an assortment of picture highlights and qualities well-defined for every illness. Through this methodology, the framework means to improve symptomatic accuracy via consequently perceiving unpretentious subtleties demonstrative of changing sickness stages, eventually helping medical services experts in pursuing opportune and informed therapy choices for patients with cerebrum cancers or Alzheimer's illness.

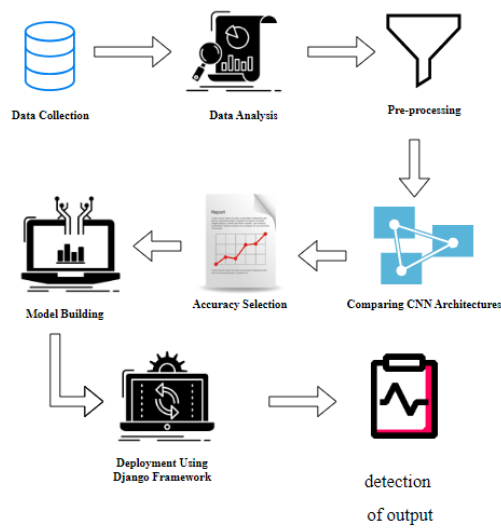


Fig 1. Architecture - proposed model

V. WORKFLOW DIAGRAM

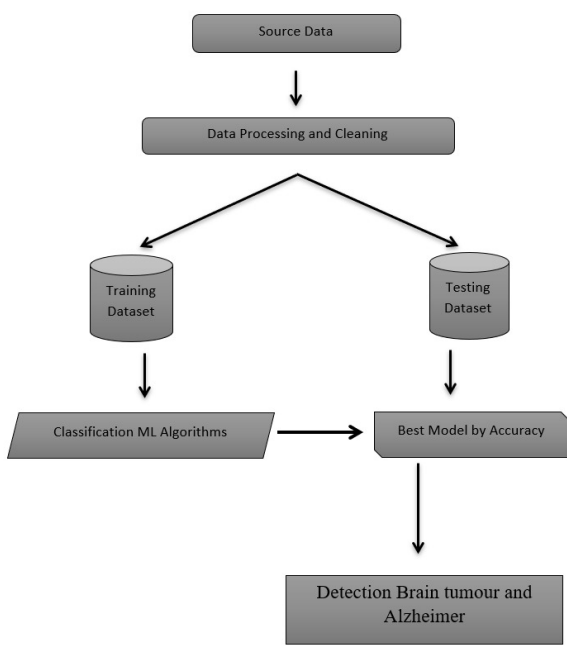


Fig 2. Workflow

VI. DATA PREPROCESSING

To get the fumble speed of the AI model, endorsement techniques in AI are used which is near the substantial mistake track of the data set. We might not need the endorsement strategies on the spoiled chance that this data limit is acceptably gigantic to be logical of this all-inclusive community. In any case, to co-function the preliminary information that might not be a confirmed expert of the quantity about tenants in the given data set in obvious conditions. Duplicate the value & depiction of the documentation type if it is an integer or floating variable to see the missing regard. While adaptive model volatile curb, an impartial examination of a manner reasonable over the preparing data set given using the case about data.

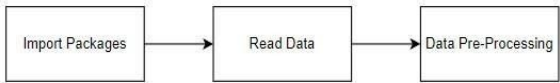


Fig 3. Data pre-processing diagram

VII. DATA VISUALIZATION

Bivariate process. The means and procedures Bringing in libraries, stacking the given information. Reviewing variable obvious proof of the data shape, and type, and looking over the missing attributes, and duplicate respects. An underwriting dataset is the delineation of information tried not to set up your models that are used to present an analyzed model capacity while adapting models and frameworks that could be used to use the support and examine data set while evaluating the models. Data purification/organization by relabeling the given data set and unhanding the part. To isolate the single variate, dual variate, and for data purify do differentiate from one information base to another set. The key objective of information data is to see and discard stumbles and idiosyncrasies to develop the worth of the dataset in appraisal and autonomous courses.

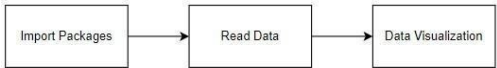


Fig 4. Data Visualization diagram

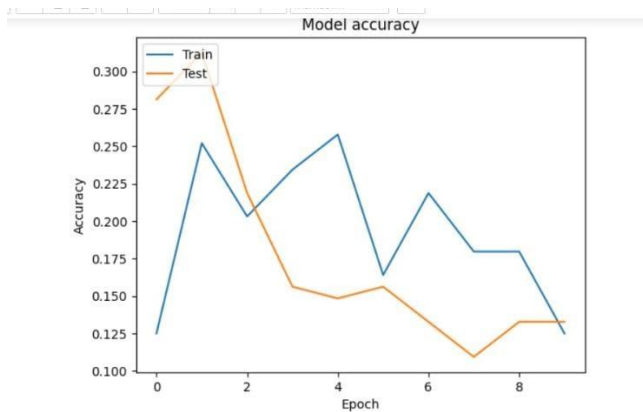


Fig 5. Data in graph - manual net

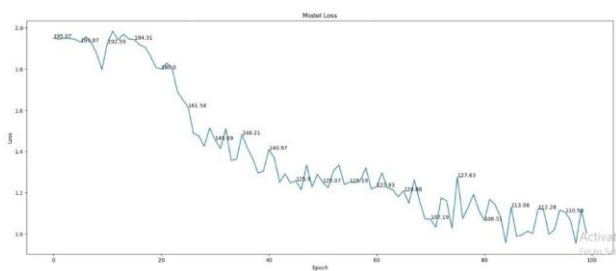


Fig 6. Data in graph – Google net

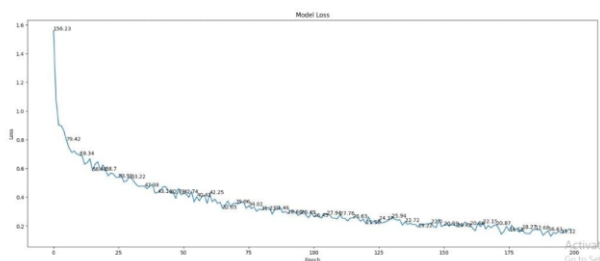


Fig 7. Data in graph - Lenet

VIII. ALGORITHM AND FORECAST AS BEST EXACTNESS RESULT

- Support Vector Machine
- Random forest
- Logistic
- Decision tree

IX. FORECAST OUTCOME BY PRECISION

Certifiable Positive Rate(TPR) = $TP / (TP + FN)$
 Bogus Positive rate(FPR) =

$FP / (FP + TN)$ Precision = $(TP + TN) / (TP + TN + FP + FN)$ Exactness = $TP / (TP + FP)$ Review = $TP / (TP + FN)$ General Equation:

F-Measure = $2TP / (2TP + FP + FN)$

F1-Score Equation:

F1 Score = $2 * (Recall * Accuracy) / (Review + Accuracy)$

X. CONCLUSION

All in all, the venture zeroed in on improving brain cancer order through the execution of Convolutional Brain Organization (CNN) models. By utilizing progressed profound learning strategies, the model exhibited huge steps in precisely recognizing and ordering cerebrum growths from clinical imaging information. The use of CNNs demonstrated instrumental in removing perplexing highlights and examples, empowering more exact and effective growth characterization. This examination adds to the continuous endeavors to work on symptomatic exactness and smooth out the dynamic cycle in the field of clinical imaging, at last propelling the possibilities for early location and treatment of cerebrum growths. The progress of this task highlights the capability of CNN designs in upsetting clinical picture examination and highlights their significance in the domain of medical care advancement.

XI. FUTURE WORK

Future work in cerebrum cancer characterization includes utilizing Convolutional Brain Organization (CNN) structures for improved precision in clinical imaging. These particular profound learning models succeed in design acknowledgment, demonstrating significance in the examination and arrangement of cerebrum cancer pictures. The objective is to work on symptomatic accuracy and therapy arranging by tackling the force of CNNs, subsequently adding to headways in clinical picture examination for neuro-oncology.

REFERENCES

- [1] Velmurugan, A., J. Albert Mayan, R. Mohana Prasad, and A. Yovan Felix. "Implementing health care center using hadoop for analysis and prediction of diseases." International Journal of Engineering and Advanced Technology (IJEAT) 8, no. 6 (2019): 3456-3459.

- [2] Gowri, S., Revathy, S., Vigneshwari, S., Jabez, J., Felix, Y., Srinivasulu, S. (2020). Clustering of Various Diseases by Collagen Gene Using the Positional Factor. In: Smys, S., Iliyasu, A.M., Bestak, R., Shi, F. (eds) New Trends in Computational Vision and Bio-inspired Computing. ICCVBIC 2018. Springer, Cham.
https://doi.org/10.1007/978-3-030-41862-5_80
- [3] G. Gasparoni *et al.*, "DNA methylation analysis on purified neurons
and glia dissects age and alzheimer's disease-specific changes
in the human cortex," *Epigenetics Chromatin*, vol. 11, no. 1, pp. 1–19,
2018.
- [4] M. Xu *et al.*, "A systematic integrated analysis of brain expression
profiles reveals yap1 and other prioritized hub genes as important
upstream regulators in alzheimer's disease," *Alzheimer's Dementia:*
J. Alzheimer's Assoc., vol. 14, pp. 215–229, 2018.
- [5] Y. Xiong *et al.*, "Diseasemeth version 2.0: A major expansion and
update of the human disease methylation database," *Nucleic Acids*
Res., vol. 45, pp. D888–D895, 2017.
- [6] D. Zhang *et al.*, "Multimodal classification of
Alzheimer's disease
and mild cognitive impairment," *NeuroImage*, vol. 55, no. 3,
pp. 856–867, 2011.
- [7] H.-I. Suk, C.-Y. Wee, S.-W. Lee, and D. Shen,
"Supervised discriminative
group sparse representation for mild cognitive
impairment diagnosis," *Neuroinformatics*, vol. 13, pp. 277–295,
2014.
- [8] Y. Shi, H.-I. Suk, Y. Gao, and D. Shen, "Joint coupled-
feature
representation and coupled boosting for AD diagnosis,"
in Proc.
IEEE Conf. Comput. Vis. Pattern Recognit., 2014, pp. 2721–2728.
- [9] H.-I. Suk and D. Shen, "Deep learning-based feature
representation
for AD/MCI classification," in Proc. Int. Conf. Med. Image
Comput. Comput.-Assist. Interv., 2013, pp. 583–590.
- [10] S. Korolev, A. Safiullin, M. Belyaev, and Y. Dodonova,
"Residual
and plain convolutional neural networks for 3D brain MRI
classification,"
in Proc. IEEE Int. Symp. Biomed. Imag., 2017, pp. 835–
838.
- [11] R. E. Marioni *et al.*, "GWAS on family history of
Alzheimer's disease," *Translational psychiatry*, vol. 8, no. 1, pp. 1-7, 2018.
- [12] B. W. Kunkle *et al.*, "Genetic meta-analysis of
diagnosed Alzheimer's disease identifies new risk loci and
implicates A β , tau, immunity and lipid processing,"
Nature genetics, vol. 51, no. 3, pp. 414-430, 2019.
- [13] F. Zhuang *et al.*, "A comprehensive survey on
transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43-76, 2020.
- [14] B.-L. Romero-Rosales, J.-G. Tamez-Pena, H. Nicolini,
M.-G. Moreno-Treviño, and V. Trevino, "Improving
predictive models for Alzheimer's disease using GWAS
data by incorporating misclassified samples modeling,"
PloS one, vol. 15, no. 4, p. e0232103, 2020.
- [15] M. Maciukiewicz *et al.*, "GWAS-based machine
learning approach to predict duloxetine response in
major depressive disorder," (in eng), *J Psychiatr Res*,
vol. 99, pp. 62-68, Apr 2018, doi:
10.1016/j.jpsychires.2017.12.009.
- [16] J. V. Sanchez-Mut *et al.*, "DNA methylation map of
mouse and human brain identifies target genes in
alzheimer's disease," *Brain : A J. Neurol.*, vol. 136, pp. 3018–3027, 2013.
- [17] A. D. Francesco *et al.*, "Involvement of 5-lipoxygenase
in
alzheimer's disease: A role for dna methylation,"
J.Alzheimer's
Dis., vol. 37, pp. 3–8, 2013.
- [18] Y. Hou *et al.*, "Changes in methylation patterns of
multiple genes from peripheral blood leucocytes of
alzheimer's disease patients," *Acta Neuropsychiatrica*, vol. 25, pp. 66–76, 2013.
- [19] A. Iwata *et al.*, "Altered CPG methylation in sporadic
alzheimer's disease is associated with app and MAPT
dysregulation," *Hum. Mol. Genet.*, vol. 23, pp. 648–656,
2014.

RE-2022-216157-plag-report

ORIGINALITY REPORT

7%

SIMILARITY INDEX

5%

INTERNET SOURCES

3%

PUBLICATIONS

3%

STUDENT PAPERS

PRIMARY SOURCES

1

www.philstat.org.ph

Internet Source

4%

2

Submitted to University for Business and Technology

Student Paper

1%

3

Submitted to Symbiosis International University

Student Paper

1%

4

cronfa.swan.ac.uk

Internet Source

1%

5

Palvai Sai Kumar Reddy*, S. Christy. "Prediction of Best Traffic Route using Supervised Classification Machine Learning", International Journal of Recent Technology and Engineering (IJRTE), 2020

Publication

1%

Exclude quotes On

Exclude matches Off

Exclude bibliography On