RETINAL VESSEL SEGMENTATION BY USING (CNN) ALGORITHM.

A PROJECT REPORT

Submitted by

VAJRAM SUMAN	**211419104294**
STEPHEN JAYARAJ.C	**211419104271**
YOGESH.K	**211419104314**

in partial fulfilment for the award

of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

April-2023

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **" RETINAL VESSEL SEGMENTATIONBY USING(CNN) ALGORITHM"** is the Bonafide work of "**VAJRAM SUMAN(211419104294),STEPHENJAYARAJ.C(211419104271),YOGESH .K (211419104314)"** who carried out the project work under my supervision.

SIGNATURE	SIGNATURE
Dr.L.JABASHEELA M.E.,Ph.D.,	**MrM.MAHENDRAN M.Tech..(Ph.D)**
HEAD OF THE DEPARTMENT,	**ASSISTANT PROFESSOR**
DEPARTMENT OF CSE,	DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE	PANIMALAR ENGINEERING COLLEGE
NAZARATHPETTAI,	NAZARATHPETTAI,
POONAMALLEE,	POONAMALLEE,
CHENNAI-600 123.	CHENNAI-600 123.

Certified that the above mentioned students were examined in the End Semester project viva-voice held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

We **VAJRAM SUMAN(211419104294)**, **STEPHEN JAYARAJ.C (211419104271), YOGESH.K (211419104314) hereby declare that this project report titled "RETINAL VESSEL SEGMENTATION BY USING (CNN) ALGORITHM"** under the guidance of **MR.M.MAHENDRAN M.Tech..(Ph.D)** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

VAJRAM SUMAN

STEPHEN JAYARAJ.C

YOGESH.K

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.,** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI**, **Dr.C.SAKTHI KUMAR,M.E.,Ph.D**. and **Dr.SARANYASREE SAKTHI KUMAR B.E.,M.B.A.,Ph.D.,** for providing us with the necessary facilities to under take this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.**who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr.L.JABASHEELA., M.E., Ph.D.,** for the support extended throughout the project.

We would like to thank my project coordinator **Mr.M.MOHAN,M.E.,(Ph.D.)** and **Project Guide Mr.M.MAHENDRAN M.Tech(Ph.D)** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

VAJRAM SUMAN

STEPHEN JAYARAJ .C

YOGESH .K

TABLE OF CONTENTS

ABSTRACT

In medicine, the finding is all around as significant as treatment. Retinal veins are the most effectively noticeable vessels in the entire body, and in this manner, assume a critical part in the conclusion of various illnesses and eye issues. Deliberate and eye illnesses cause morphologic varieties, like the developing, restricting, or spreading of retinal veins. Imaging-based screening of retinal veins assumes a significant part in the recognizable proof and follow-up of eye illnesses. Subsequently, programmed retinal vessel division can be utilized to analyze and screen those sicknesses. Automatic segmentation of blood vessels in fundus images is of great importance as eye diseases as well as some systemic diseases cause observable pathologic modifications. Retinal blood vessel image segmentation is an important step in the ophthalmological analysis. However, it is difficult to segment small vessels accurately because of the low contrast and complex feature information of blood vessels. Segmentation is the first and most important task in computer-based diagnosis of retinal vessel since other tasks are relied mainly on accurately segmented lesions. Convolutional neural networks (CNNs) has achieved great success in the segmentation of retinal vessels.

Keywords: Neural network, U-Net, segmentation

CHAPTER 1

INTRODUCTION

1. INTRODUCTION:

The fundoscopic exam is a procedure that provides necessary information to diagnose different retinal degenerative diseases such as Diabetic Retinopathy, Macular Edema, Cytomegalovirus Retinitis. A highly accurate system is required to segment retinal vessels and find abnormalities in the retinal subspace to diagnose these vascular diseases. Many image processing and machine learning-based approaches for retinal vessel segmentation have so far been proposed. However, such methods fail to precisely pixel-wise segment blood vessels due to insufficient illumination and periodic noises. Attributes like this present in the subspace can create false-positive segmentation. In recent times, UNet based deep learning architectures have become very popular for retinal vessel segmentation. UNet consists of an encoder to capture context information and a decoder for enabling precise localization. Retinal blood vessel image segmentation is an important step in the ophthalmological analysis. However, it is difficult to segment small vessels accurately because of the low contrast and complex feature information of blood vessels. Segmentation is the first and most important task in computer-based diagnosis of retinal vessel since other tasks are relied mainly on accurately segmented lesions. Convolutional neural networks (CNNs) has achieved great success in the segmentation of retinal vessels.

1.1 ARTIFICAL 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, 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 machinesbecome increasingly capable, tasks considered to require "intelligence" are oftenremoved 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, modeling human problem solving, formal logic, large databases of knowledge and imitating animal behavior. 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 centered around 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 amongthe field's long-term goals.

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.

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.

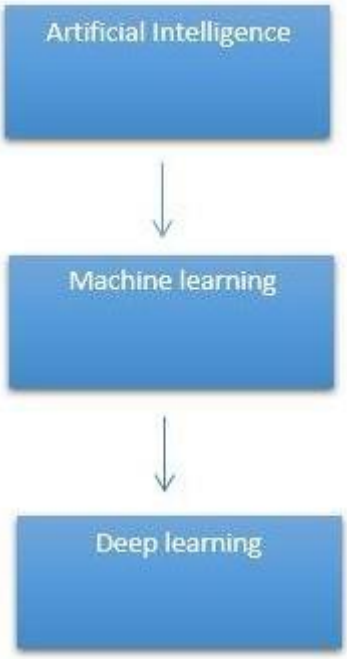
5.1 Learning processes. This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, provide computing devices with step-by-step instructions for how to complete a specific task.

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

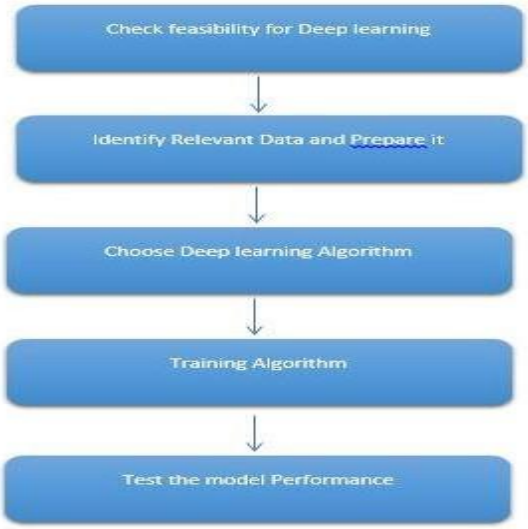
5.3 Self-correction processes. This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

1.2 DEEP LEARNING

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. It's on hype nowadays because earlier we did not have that much processing power and a lot of data. A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. In brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbors. The question here is how it recreates these neurons in a computer. So, it creates an artificial structure called an artificial neural net where we have nodes or neurons. It has some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.



It need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). It needs to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Choose the Deep Learning Algorithm appropriately. Algorithm should be used while training the dataset. Final testing should be done on the dataset



Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi- supervised or unsupervised.

INTERPRETATION

Deep neural networks are generally interpreted in terms of the universal approximation theorem or probabilistic inference.

The classic universal approximation theorem concerns the capacityof feed-forward neural networks with a single hidden layer of finite size to approximate continuous functions. In 1989, the first proof was publishedby George Cybenko for sigmoid activation functions and was generalised to feed-forward multi-layer architectures in 1991 by Kurt Hornik. Recent work also showed that universal approximation also holds for non-bounded activation functions such as the rectified linear unit.

The universal approximation theorem for deep neural networks concerns the capacity of networks with bounded width but the depth is allowed to grow proved that if the width of a deep neural network with ReLU activation is strictly larger than the input dimension, then the network can approximate any Lebesgue integrable function; If the width is smaller or equal to the input dimension, then deep neural network is not a universal approximator.

The probabilistic interpretation derives from the field of machine learning. It features inference, as well as the optimization concepts of training and testing, related to fitting and generalization, respectively. More specifically, the probabilistic interpretation considers the activation nonlinearity as a cumulative distribution function. The probabilistic interpretation led to the introduction of dropout as regularizer in neural networks.

CHAPTER 2

LITERATURE SURVEY

2.1 LITERATURE SURVEY

Title: Emerging insights into the relationship between hyperlipidemia and the risk of diabetic retinopathy

Author: Yuyu Chou1 , Jin Ma1 , Xin Su2* and Yong Zhong1*

Year:2020

Hyperlipidemia is correlated with a series of health problems. Notably, aside from its established role in promoting cardiovascular morbidity and mortality, hyperlipidemia has also been considered for modulating the risk and the severity of multiple metabolic disorders. According to the results of epidemiologic investigations, several certain circulating lipoprotein species are correlated with the prevalence of diabetic retinopathy, suggesting that the physiological and pathological role of these lipoproteins is analogous to that observed in cardiovascular diseases. Furthermore, the lipid-lowering treatments, particularly using statin and fibrate, have been demonstrated to ameliorate diabetic retinopathy. Thereby, current focus is shifting towards implementing the protective strategies of diabetic retinopathy and elucidating the potential underlying mechanisms. However, it is worth noting that the relationship between major serum cholesterol species and the development of diabetic retinopathy, published by other studies, was inconsistent and overall modest, revealing the relationship is still not clarified. In this review, the current understanding of hyperlipidemia in pathogenesis of diabetic retinopathy was summarized and the novel insights into the potential mechanisms whereby hyperlipidemia modulates diabetic retinopathy were put forward.

2.2 Title-DiabeticRetinopathy–AnUnderdiagnosedandUndertreated Inflammatory, Neuro-Vascular Complication of Diabetes

Author - Stephen H. Sinclair 1,2 * and Stanley S. Schwartz 3

Year -2019

Diabetes mellitus is a world-wide epidemic and diabetic retinopathy, a devastating, vision-threatening condition, is one of the most common diabetes-specific complications. Diabetic retinopathy is now recognized to be an inflammatory, neuro-vascular complication with neuronal injury/dysfunction preceding clinical microvascular damage. Importantly, the same pathophysiologic mechanisms that damage the pancreatic β-cell (e.g., inflammation, epigenetic changes, insulin resistance, fuel excess, and abnormal metabolic environment), also lead to cell and tissue damage causing organ dysfunction, elevating the risk of all complications, including diabetic retinopathy. Viewing diabetic retinopathy within the context whereby diabetes and all its complications arise from common pathophysiologic factors allows for the consideration of a wider array of potential ocular as well as systemic treatments for this common and devastating complication. Moreover, it also raises the importance of the need for methods that will provide more timely detection and prediction of the course in order to address early damage to the neurovascular unit prior to the clinical observation of microangiopathy. Currently, treatment success is limited as it is often initiated far too late and after significant neurodegeneration has occurred. This forward-thinking approach of earlier detection and treatment with a wider array of possible therapies broadens the physician's armamentarium and increases the opportunity for prevention and early treatment of diabetic retinopathy with preservation of good vision, as well the prevention of similar destructive processes occurring among other organs.

2.3 Title : Neurovascular Impairment and Therapeutic Strategies in Diabetic Retinopathy

Author :Toshiyuki Oshitari

Year : 2021

Diabetic retinopathy has recently been defined as a highly specific neurovascular complication of diabetes. The chronic progression of the impairment of the interdependence of neurovascular units (NVUs) is associated with the pathogenesis of diabetic retinopathy. The NVUs consist of neurons, glial cells, and vascular cells, and the interdependent relationships between these cells are disturbed under diabetic conditions. Clinicians should understand and update the current knowledge of the neurovascular impairments in diabetic retinopathy. Above all, neuronal cell death is an irreversible change, and it is directly related to vision loss in patients with diabetic retinopathy. Thus, neuroprotective and vasoprotective therapies for diabetic retinopathy must be established. Understanding the physiological and pathological interdependence of the NVUs is helpful in establishing neuroprotective and vasoprotective therapies for diabetic retinopathy. This review focuses on the pathogenesis of the neurovascular impairments and introduces possible neurovascular protective therapies for diabetic retinopathy.

2.3 Title: Diabetic retinopathy: current understanding, mechanisms, and treatment strategies

Author : Elia J. Duh,1 Jennifer K. Sun,2 Alan W. Stitt3

Year : 2017

Diabetic retinopathy (DR) causes significant visual loss on a global scale. Treatments for the vision-threatening complications of diabetic macular edema (DME) and proliferative diabetic retinopathy (PDR) have greatly improved over the past decade. However, additional therapeutic options are needed that take into account pathology associated with vascular, glial, and neuronal components of the diabetic retina. Recent work indicates that diabetes markedly impacts the retinal neurovascular unit and its interdependent vascular, neuronal, glial, and immune cells. This knowledge is leading to identification of new targets and therapeutic strategies for preventing or reversing retinal neuronal dysfunction, vascular leakage, ischemia, and pathologic angiogenesis. These advances, together with approaches embracing the potential of preventative or regenerative medicine, could provide the means to better manage DR, including treatment at earlier stages and more precise tailoring of treatments based on individual patient variations.

2.4 Title : The Evolving Treatment of Diabetic Retinopathy

Author : Sam E Mansour 1,2 David J Browning

Year 2020

With the recent expansion of management options for diabetic retinopathy, optimal sequences of treatment application and combination in specific clinical situations are under investigation. A review and synthesis of the ophthalmologic literature on treatment of diabetic retinopathy was performed to provide perspective on the relative prioritization of the various treatments in the contexts seen in clinical practice. In general, pharmacotherapy is ascendant, particularly with the anti-VEGF class, while laser treatment continues to have lesser roles in specific situations and under certain economic constraints. Surgical intervention continues to be reserved for those situations which fail to respond to pharmacotherapy, laser or combination therapy. Ongoing refinements in the systemic management of both hyperglycemia and hyperlipidemia continue to demonstrate significant benefits for both diabetic retinopathy and diabetic macular edema. Recent developments involving newer retinal diagnostics are proving beneficial in optimizing both initiation and maintenance of therapy. As well, recent advances in novel pharmaceutical agents and ocular drug delivery methods show promise in better controlling the disease as well as reducing the burden of treatment.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Existing System:

Architecture of convolution neural network for diabeticretinopathy (DR-Net) is based on normal convolution (NC). It incurs highcomputational cost as NC uses a multiplicative weight that measures acombined correlation in both cross-channel and spatial dimension of layer'sinputs. This might cause the overall DR-Net architecture to be over-parameterised and computationally inefficient. This paper reports thedevelopment and the experimental results of a new lightweight DR-Net, withDWSC module, called EDR-Net for predicting the presence of referable DRfrom fundus images.

Drawbacks:

- No comparison of architectures.
- It only classifies the data.

3.2 PROPOSED SYSTEM:

This paper presents an image segmentation technique using CNN U-Net architecture for vessel segmentation. This system have many good features among different segmentation schemes. It organizes the image elements into, mathematically and structural form, and makes the problem formulation more flexible, and the computation, more efficient. The proposed method consist of segmentation of image by using a CNN algorithm. The datasets of normal and masked images are used for training the image for segmentation. The best architecture from CNN algorithm is used for training the data and the model is created. Finally the model is used for segmenting the retinal vessel.

Advantages:

- Segments the vessel from the retina.
- Project will be deployed.

CHAPTER 4

FEASIBILITY STUDY

4.FEASIBILITY

Aim; Retinal vessel segmentation is one of the major factors in our healthcaredomain. There are lot of retinal diseases that are actively present in the world.So we can't able to classify the disease easily. So this project can easily classify the disease.

Objectives:

The goal is to develop a deep learning model for retinal vessel segmentation by convolutional neural network algorithm for potentially classifying the results in the form of best accuracy by comparing the CNN architectures.

Scope:

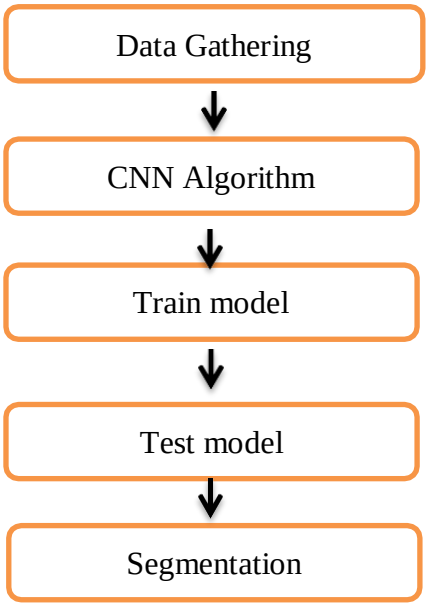
A normal and mask images are collected. We have to train the machine to find the pattern of the vessel. This project is used to find the segmented fundus image. We train to teach the machine to achieve the accuracy and get the possible outcome.

4.1 SPLLITING DATA SET

The data used is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. It has the test dataset (or subset) in order to test our models and it will do this using the Tensorflow library in Python using the Keras method.

4.2 DETECTION MODEL

Deep learning needs data gathering have lot of past image data's. Training and testing this model working and predicting correctly.



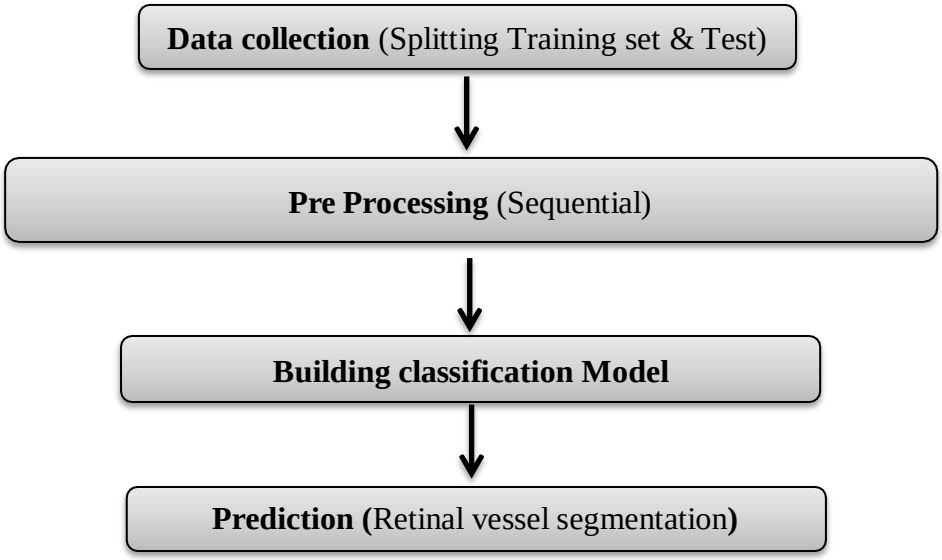


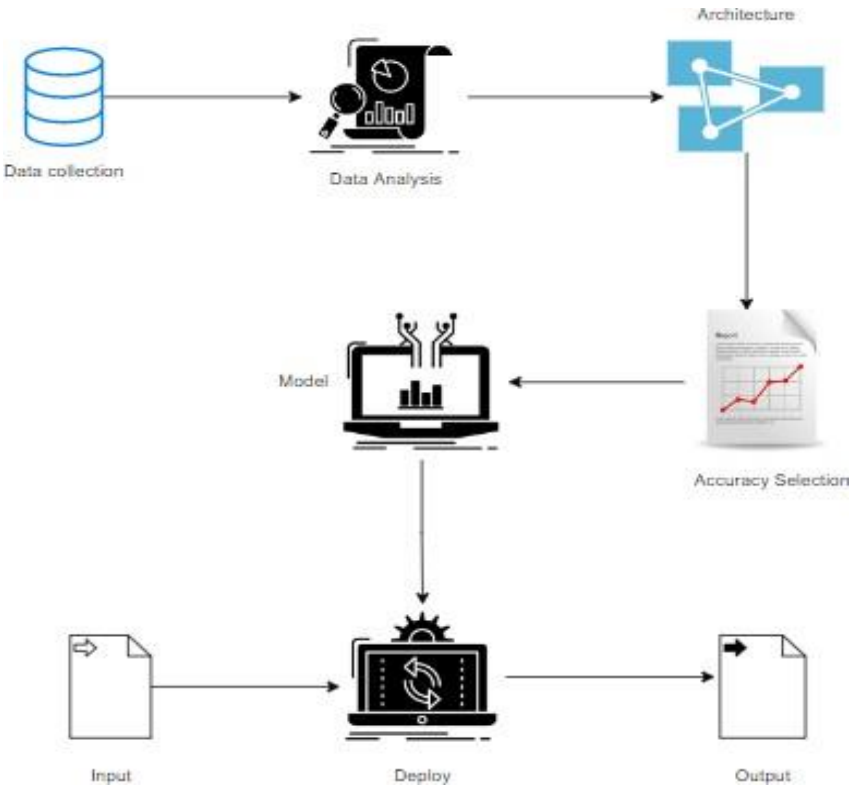
Fig: data flow diagram for CNN model

CHAPTER 5

ARTITECTURE SYSTEM

5.1 SYSTEM ARTITECTURE

Imaging-based screening of retinal veins assumes a significant part in the recognizable proof and follow-up of eye illnesses. Subsequently, programmed retinal vessel division can be utilized to analyze and screen those sicknesses. Automatic segmentation of blood vessels in fundus images is of great importance as eye diseases as well as some systemic diseases cause observable pathologic modifications. Retinal blood vessel image segmentation is an important step in the ophthalmological analysis. However, it is difficult to segment small vessels accurately because of the low contrast and complex feature information of blood vessels. Segmentation is the first and most important task in computer-based diagnosis of retinal vessel since other tasks are relied mainly on accurately segmented lesions. Convolutional neural networks (CNNs) has achieved great success in the segmentation of retinal vessels.

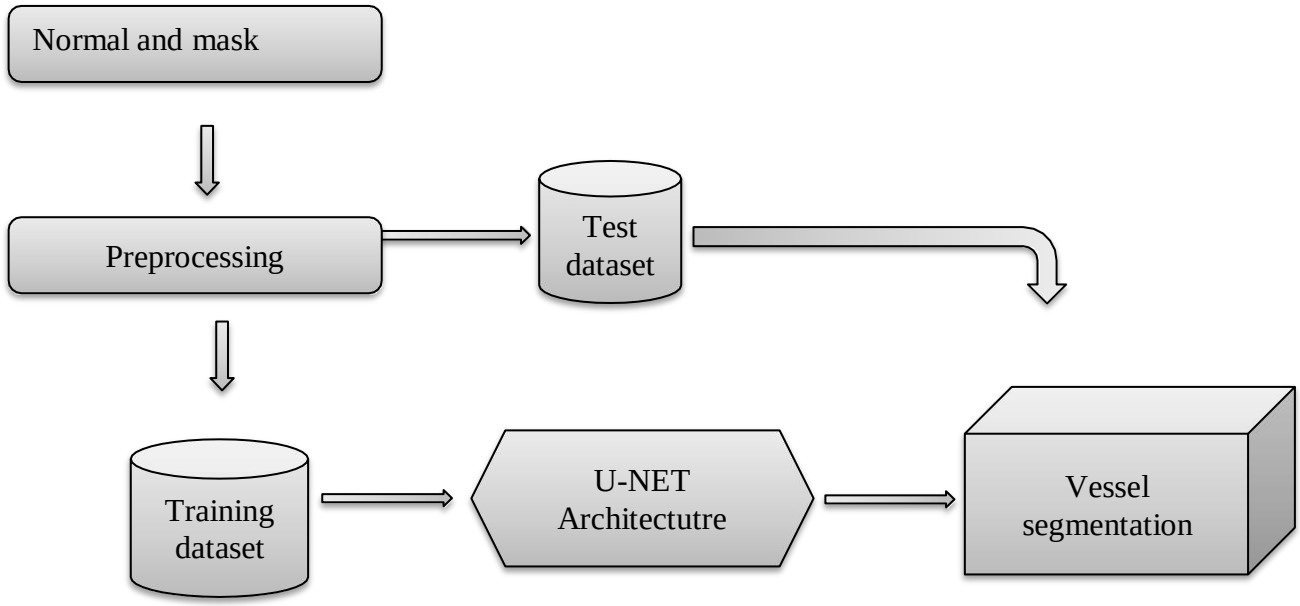


5.2 DESIGN ARTITECTURE

General

Design is meaningful engineering representation of something that is to be built. Software design is a process design is the perfect way to accurately translate requirements in to a finished software product. Design creates a representation or model, provides detail about software data structure, architecture, interfaces and components that are necessary to implement a system.

Data Flow Diagram:



Process of dataflow diagram

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often usedas a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through

the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model. Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top down approach to Systems Design. Symbols and Notations Used in DFDs Using any convention's DFD rules or guidelines, the symbols depict the four components of data flow diagrams.

External entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

Process: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.

Data store: files or repositories that hold information for later use, such as a database table or a membership form.

Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like "Billing details."

DFD levels and layers A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish. DFD Level 0 is also called a Context Diagram.

CHAPTER 6

MODULE

6 LIST OF MODULES :

1. Data Analysis

2. Manual Architecture

3. U-Net Architecture

4. Deployment

MODULE DESCRIPTION

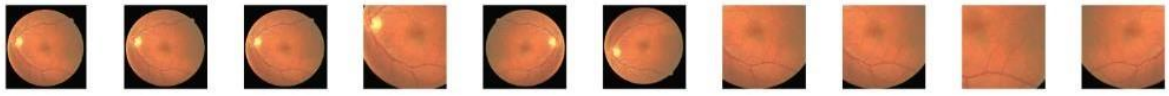
6.1 Data Analysis

Data analysis is the process of cleaning, changing, and processing raw data, and extracting actionable, relevant information that helps businesses make informed decisions. The procedure helps reduce the risks inherent in decision-making by providing useful insights. The data analysis process, or alternately, data analysis steps, involves gathering all the information, processing it, exploring the data, and using it to find patterns and other insights.

In data analysis we analyse the data that how the image data is available. We analyse how many data are available and we check whether the normal data is available corresponding to the mask data.

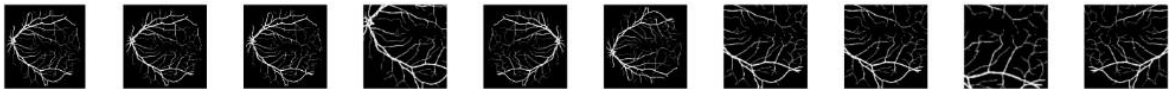
NORMAL IMAGES:

```
Train data for Orinigal Images:

 ====== Images in:  datasets/Images
images_count:   420
min_width:      256
max_width:      584
min_height:     256
max_height:     584
```



MASK IMAGES:

```
Train data for Mask images:

 ====== Images in:  datasets/Masks
images_count:   420
min_width:      256
max_width:      584
min_height:     256
max_height:     584
```



6.2 Manual Architecture and U-Net Architecture:

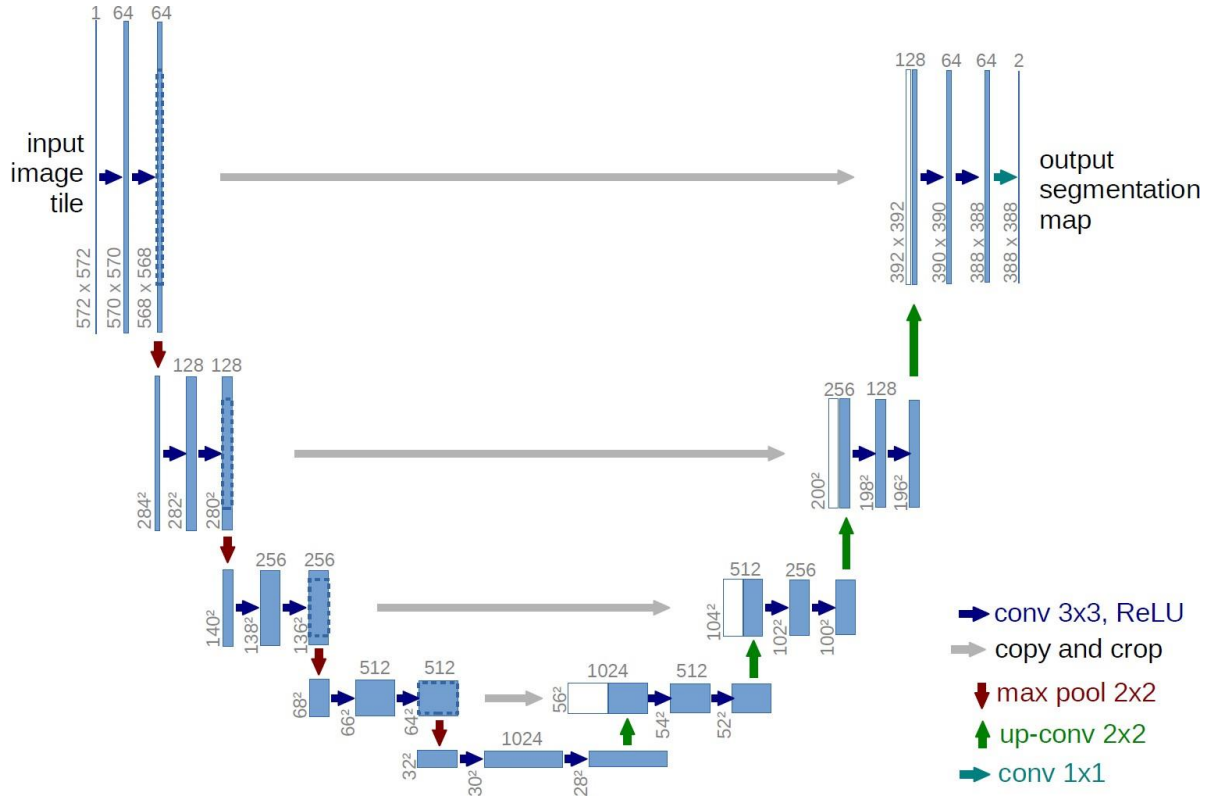
A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate onefrom the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that ofthe connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field.

6.3 U-NET ARCHITECTURE:

The U-Net architecture, first published in the year 2015, has been a revolution in the field of deep learning. The architecture won the International Symposium on Biomedical Imaging (ISBI) cell tracking challenge of 2015 in numerous categories by a large margin. Some of their works include the segmentation of neuronal structures in electron microscopic stacks and transmitted light microscopy images.

With this U-Net architecture, the segmentation of images of sizes 512X512 can be computed with a modern GPU within small amounts of time. There have

been many variants and modifications of this architecture due to its phenomenal success. Some of them include LadderNet, U-Net with attention, the recurrent and residual convolutional U-Net (R2-UNet), and U-Net with residual blocks or blocks with dense connections.



6.4 DEPLOYMENT

In this module, the trained deep learning model is converted into a hierarchical data format file (.h5 file) which is then deployed in our Django framework for providing a better user interface and predicting the output of retinal fundus image.

Django

Django is a high-level Python web framework that enables rapiddevelopment of secure and maintainable websites. Built by experienceddevelopers, Django takes care of much of the hassle of web development, so youcan focus on writingyour app without needing to reinvent the wheel. It is freeand opensource, has a thriving and active community, great documentation, andmany options for free and paid-for support.

Django helps you write software that is:

Complete

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

Versatile

Django can be (and has been) used to build almost any type ofwebsite — from content management systems and wikis, throughto social networks and news sites. It can work with any client-sideframework, and can deliver content in almost any format (includingHTML, RSS feeds, JSON, XML, etc). The site you are currentlyreading is built with Django!Internally, while it provides choices for almost any functionality youmight want (e.g. several popular databases, templating engines, etc.), itcan also be extended to use other components if needed.

Secure

- Django helps developers avoid many common security mistakesby providing a framework that has been engineered to "do theright things" to protect the website automatically.
- For example,Django provides a secure way to manage user accounts andpasswords, avoiding common mistakes like putting sessioninformation in cookies where it is vulnerable (instead cookiesjust containa 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 bysending thepassword through a cryptographic hash function.
- Django can check ifan entered password is correct by running it through the hashfunction and comparing the output to the stored hash value.
- Howeverdue to the "one-way" nature of the function, even if a stored hashvalue is compromised it is hard for an attacker to work out the originalpassword.
- Django enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-siterequest forgery and clickjacking (see Website security for moredetails of such attacks).

Scalable

Django uses a component-based “shared-nothing” architecture (eachpart of the architecture is independent of the others, and can hence bereplaced or changed if needed). Having a clear separation between thedifferent parts means that it can scale for increased traffic by adding hardwareat any level: caching servers, database servers, or application

servers. Someof the busiest sites have successfully scaled Django to meet their demand

Maintainable

Django code is written using design principles and patterns that enourage the creation of maintainable and reusable code. In particular, itmakes use of the Don't Repeat Yourself (DRY) principle so there is nounnecessary duplication, reducing the amount of code. Django also promotesthe grouping of related functionality into reusable "applications" and, at alower level, groups related code into modules (along the lines of the ModelView Controller (MVC) pattern).

Portable

Django is written in Python, which runs on many platforms. Thatmeans thatyou are not tied to any particular server platform, and can runyour applications on many flavours of Linux, Windows, and Mac OS X.Furthermore, Django is well-supported by many web hosting providers, whooften provide specific infrastructure and documentation for hosting Djangosites.

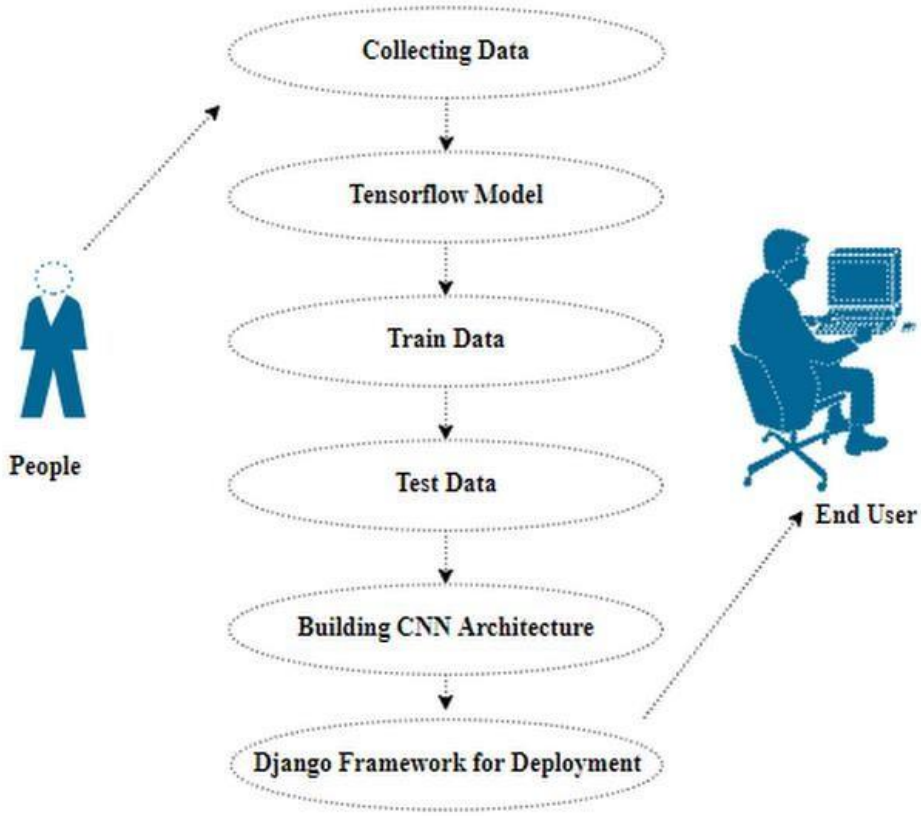
METHODOLOGY

Preprocessing and Training the model (CNN): The datasetispreprocessedsuch as Image reshaping, resizing and conversion to an array form.Similarprocessing is also done on the test image. A dataset consisting ofaboutdifferent mask and normal images is obtained, out of which any imagecan be used as a test image for the software.

CHAPTER 7

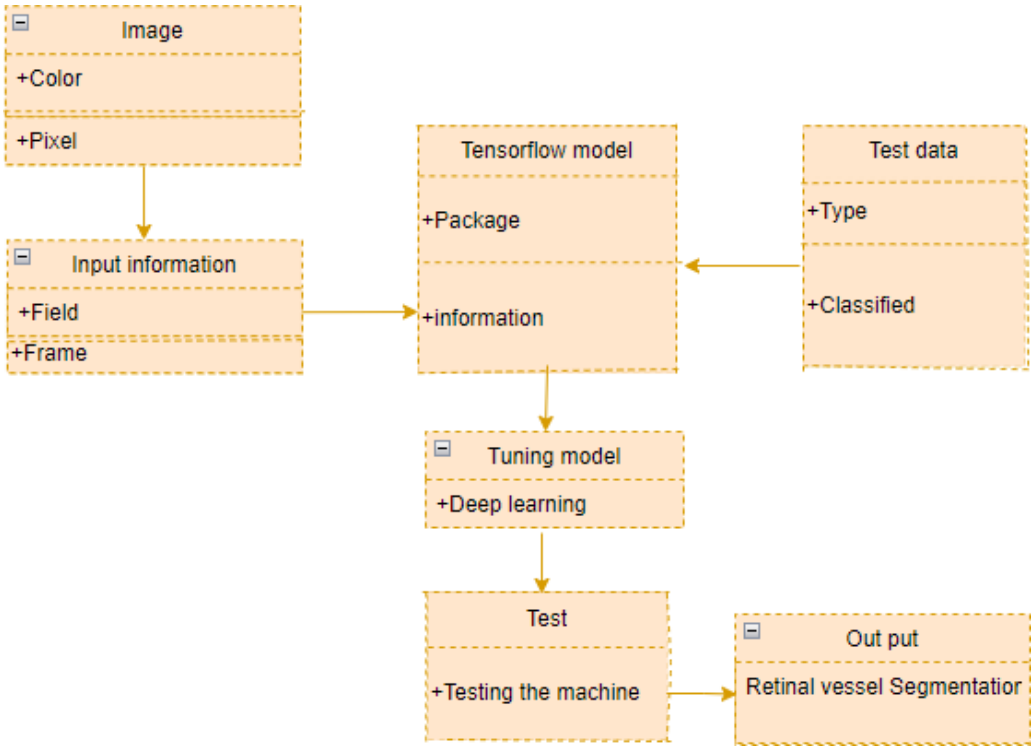
UML DIGRAM

7.1 USECASE DIAGRAM:



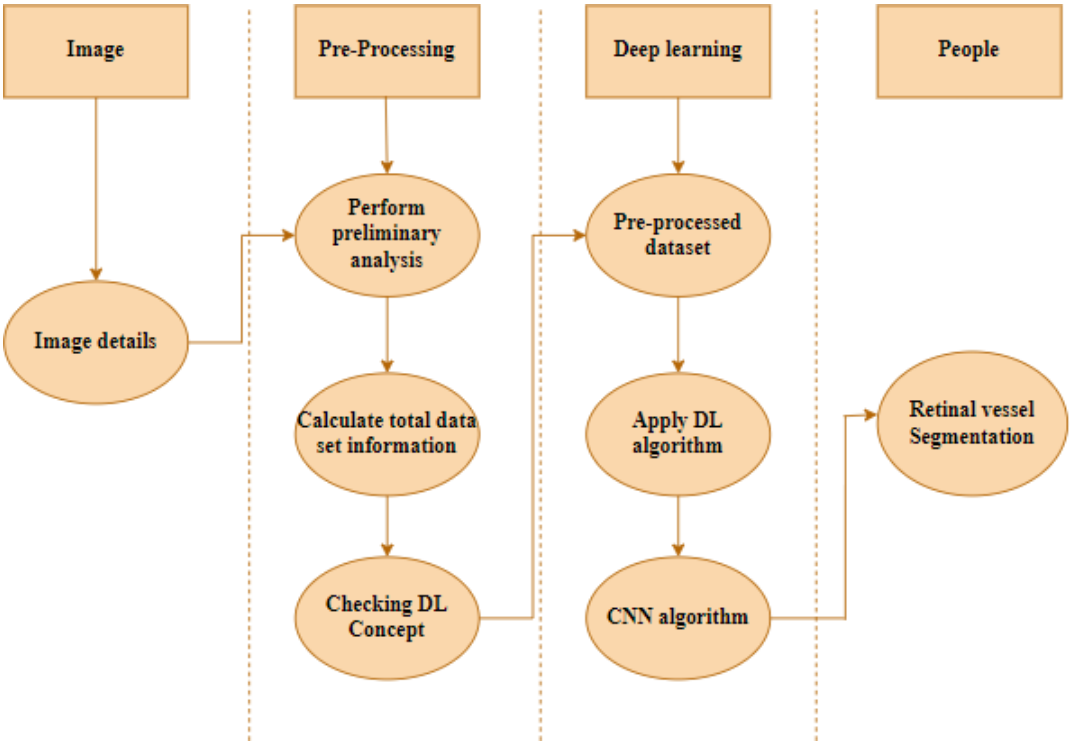
Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

7.2 CLASS DIAGRAM:



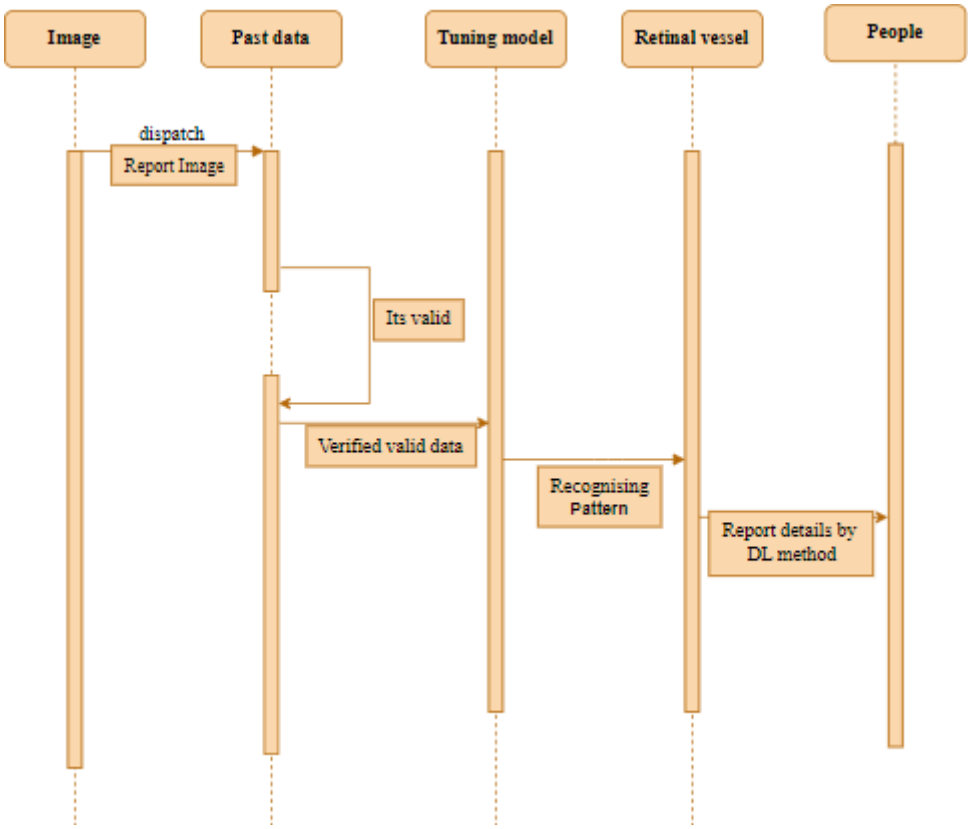
Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final versionthe diagram should be drawn on plain paper and rework as many times as possible to make it correct.

7.3 ACTIVITY DIAGRAM:



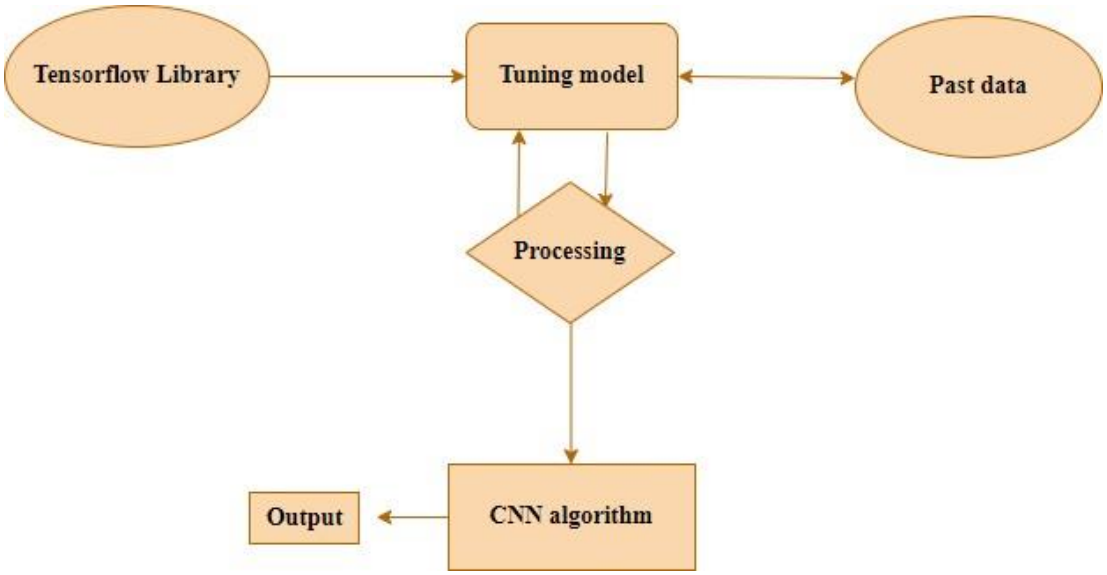
Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

7.4 SEQUENCE DIAGRAM:



Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behaviour within your system. Other dynamic modelling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are in my opinion the most important design-level models for modern business application development.

7.5 ER DIAGRAM:



An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering beneeded later.

Chapter 8

SYSTEM DESCRIPTION

8.1 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.

8.2 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.

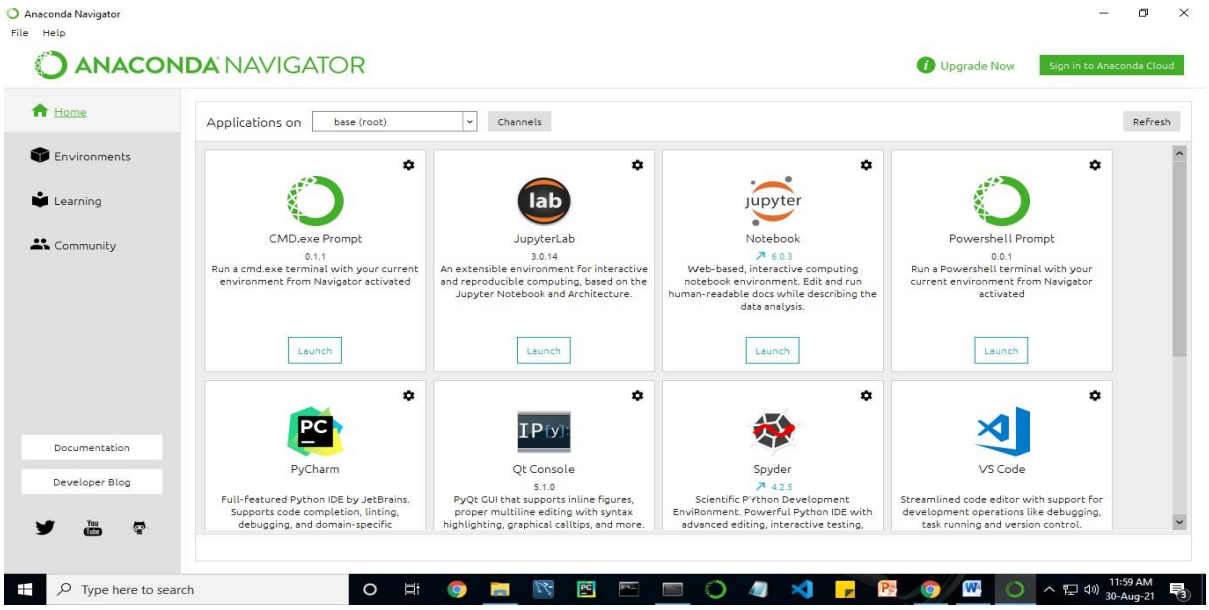
In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

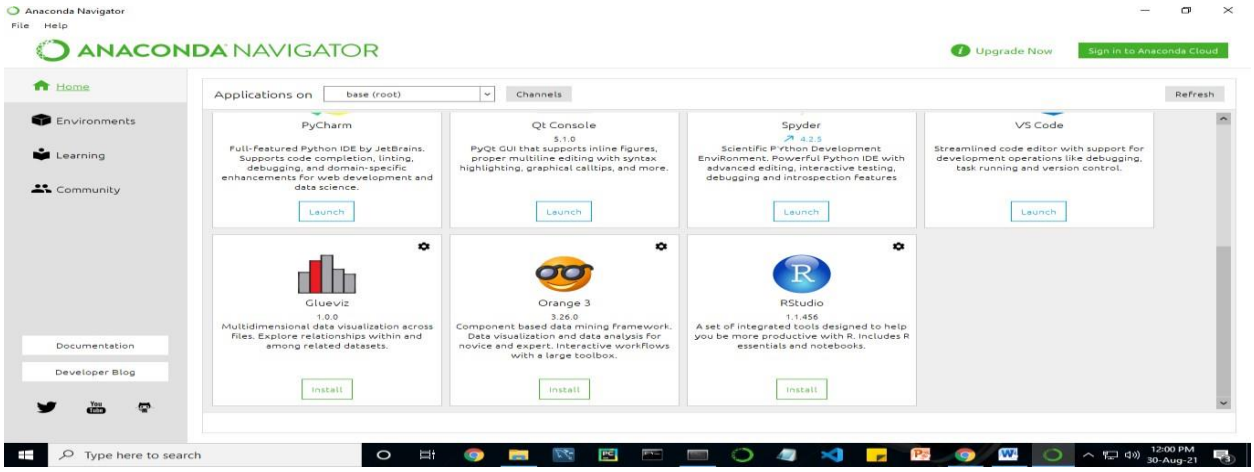
The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

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





8.3 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 *ipython 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 Jupyter 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.

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.

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.

8.4 PYCHARM:

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development. Code faster and with more easily in a smart and configurable editor with code completion, snippets, code folding and split windows support.

PyCharm Features

- **Intelligent Coding Assistance** – PyCharm provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactorings and rich navigation capabilities.
- **Intelligent Code Editor** – PyCharm's smart code editor provides first-class support for Python, JavaScript, CoffeeScript, TypeScript, CSS, popular template languages and more. Take advantage of language-aware code completion, error detection, and on-the-fly code fixes!
- **Smart Code Navigation** – Use smart search to jump to any class, file or symbol, or even any IDE action or tool window. It only takes one click to switch to the declaration, super method, test, usages, implementation, and more.
- **Fast and Safe Refactorings** – Refactor your code the intelligent way, with safe Rename and Delete, Extract Method, Introduce Variable, Inline Variable or Method, and other refactorings. Language and framework-specific refactorings help you perform project-wide changes.
- **Built-in Developer Tools** – PyCharm's huge collection of tools out of the box includes an integrated debugger and test runner; Python profiler; a

built-in terminal; integration with major VCS and built-in database tools; remote development capabilities with remote interpreters; an integrated ssh terminal; and integration with Docker and Vagrant.

- **Debugging, Testing and Profiling** – Use the powerful debugger with a graphical UI for Python and JavaScript. Create and run your tests with coding assistance and a GUI-based test runner. Take full control of your code with Python Profiler integration.
- **VCS, Deployment and Remote Development** – Save time with a unified UI for working with Git, SVN, Mercurial or other version control systems. Run and debug your application on remote machines. Easily configure automatic deployment to a remote host or VM and manage your infrastructure with Vagrant and Docker.
- **Database tools** – Access Oracle, SQL Server, PostgreSQL, MySQL and other databases right from the IDE. Rely on PyCharm's help when editing SQL code, running queries, browsing data, and altering schemas.
- **Web Development** – In addition to Python, PyCharm provides first-class support for various Python web development frameworks, specific template languages, JavaScript, CoffeeScript, TypeScript, HTML/CSS, AngularJS, Node.js, and more.
- **Python Web frameworks** – PyCharm offers great framework-specific support for modern web development frameworks such as Django, Flask, Google App Engine, Pyramid, and web2py, including Django templates debugger, manage.py and appcfg.py tools, special autocompletion and navigation, just to name a few.
- **JavaScript & HTML –** PyCharm provides first-class support for JavaScript, CoffeeScript, TypeScript, HTML and CSS, as well as their modern successors. The JavaScript debugger is included in PyCharm and is integrated with the Django server run configuration.

- **Live Edit** – Live Editing Preview lets you open a page in the editor and the browser and see the changes being made in code instantly in the browser. PyCharm auto-saves your changes, and the browser smartly updates the page on the fly, showing your edits.
- **Scientific Tools** – PyCharm integrates with Ipython Notebook, has an interactive Python console, and supports Anaconda as well as multiple scientific packages including Matplotlib and NumPy.
- **Interactive Python console** – You can run a REPL Python console in PyCharm which offers many advantages over the standard one: on-the-fly syntax check with inspections, braces and quotes matching, and of course code completion.
- **Scientific Stack Support** – PyCharm has built-in support for scientific libraries. It supports Pandas, Numpy, Matplotlib, and other scientific libraries, offering you best-in-class code intelligence, graphs, array viewers and much more.
- **Conda Integration** – Keep your dependencies isolated by having separate Conda environments per project, PyCharm makes it easy for you to create and select the right environment.
- **Customizable and Cross-platform IDE** – Use PyCharm on Windows, Mac OS and Linux with a single license key. Enjoy a fine-tuned workspace with customizable color schemes and key-bindings, with VIM emulation available.
- **Customizable UI** – Enjoy a fine-tuned workspace with customizable color schemes and key-bindings.
- **Plugins** – More than 10 years of IntelliJ platform development gives PyCharm 50+ IDE plugins of different nature, including support for additional VCS, integrations with different tools and frameworks, and editor enhancements such as Vim emulation.
- **Cross-platform IDE** – PyCharm works on Windows, Mac OS or Linux.

8.5 PYTHON

Python isan interpreted high-level general-purposeProgramming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It Supports multiple programming aradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible. Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

History:

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his

"permanent vacation" from his responsibilities as Python's Benevolent Dictator For Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a 5-member "Steering Council" to lead the project. As of 2021, the current members of this council are Barry Warsaw, Brett Cannon, Carol Willing, Thomas Wouters, and Pablo Galindo Salgado.

Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's end-of-life, only Python 3.6.x and later are supported.

Python 3.9.2 and 3.8.8 were expedited as all versions of Python (including 2.7) had security issues, leading to possible remote code execution and web cache poisoning.

Design Philosophy & Feature

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta-programming and meta-objects (magic

methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter, map and reduce functions; list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional toolsborrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible (with modules). This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one— and preferably only one —obvious way to do it" design philosophy. Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the C-Python reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

Python's developers aim to keep the language fun to use. This is reflected in its name a tribute to the British comedy group Monty Python and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (a reference to a Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonistas

Syntax and Semantics :

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are allowed but are rarely, if ever, used. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation :

Main article: Python syntax and semantics & Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the off-side rule, which some other languages share, but in most languages indentation does not have any semantic meaning. The recommended indent size is four spaces.

Statements and control flow :

Python's statements include:

- The assignment statement, using a single equals sign =.
- The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
- The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.

- The while statement, which executes a block of code as long as its condition is true.
- The Try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
- The raise statement, used to raise a specified exception or re-raise a caught exception.
- The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The def statement, which defines a function or method.
- The with statement, which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing resource-acquisition-is-initialization (RAII) - like behavior and replaces a common try/finally idiom.
- The break statement, exits from a loop.
- The continue statement, skips this iteration and continues with the next item.
- The del statement, removes a variable, which means the reference from the name to the value is deleted and trying to use that variable will cause an error. A deleted variable can be reassigned.
- The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The assert statement, used during debugging to check for conditions that should apply.
- The yield statement, which returns a value from a generator function and yield is also an operator. This form is used to implement co-routines.
- The return statement, used to return a value from a function.

- The import statement, which is used to import modules whose functions or variables can be used in the current program.

The assignment statement (=) operates by binding a name as a reference to a separate, dynamically-allocated object. Variables may be subsequently rebound at any time to any object. In Python, a variable name is a generic reference holder and does not have a fixed data type associated with it. However, at a given time, a variable will refer to some object, which will have a type. This is referred to as dynamic typing and is contrasted with statically-typed programming languages, where each variable may only contain values of a certain type.

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will.[80][81] However, better support for co-routine-like functionality is provided, by extending Python's generators. Before 2.5, generators were lazy iterators; information was passed uni-directionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

Expressions :

Some Python expressions are similar to those found in languages such as C and Java, while some are not:

- Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) // and floating-point / division. Python also uses the ** operator for exponentiation.

- From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.
- From Python 3.8, the syntax :=, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.
- In Python, == compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example A<=B<=C.
- Python uses the words and, or, not for or its boolean operators rather than the symbolic &&, ||, ! used in Java and C.
- Python has a type of expression termed a list comprehension as well as a more general expression termed a generator expression.
- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.
- Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages).
- Python makes a distinction between lists and tuples. Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.

- Python features sequence unpacking wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in an identical manner to that forming tuple literals and, as a whole, are put on the left-hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right-hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through and will iterate through it, assigning each of the produced values to the corresponding expression on the left.
- Python has a "string format" operator %. This functions analogously ton printf format strings in C, e.g. “spam=%s eggs=%d” % (“blah”,2) evaluates to “spam=blah eggs=2”. In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. “spam={0} eggs={1}”.format(“blah”,2). Python 3.6 added "f-strings": blah = “blah”; eggs = 2; f‘spam={blah} eggs={eggs}’
- Strings in Python can be concatenated, by "adding" them (same operator as for adding integers and floats). E.g. “spam” + “eggs” returns “spameggs”. Even if your strings contain numbers, they are still added as strings rather than integers. E.g. “2” + “2” returns “2”.

- Python has various kinds of string literals:
  - Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and doublequote marks function identically. Both kinds of string use the backslash (\)as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".

- Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
- Raw string varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.

- Python has array index and array slicing expressions on lists, denoted as a[Key], a[start:stop] or a[start:stop:step]. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements fromthe start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

- List comprehensions vs. for-loops
- Conditional expressions vs. if blocks
- The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain

statements. A particular case of this is that an assignment statement such as a=1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c==1) {…} is syntactically valid (but probably unintended) C code but if c=1: … causes a syntax error in Python.

Methods :

Methods on objects are functions attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, syntactic sugar for Class.method(instance, argument). Python methods have an explicit self parameter access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby). Apart from this Python also provides methods, sometimes called d- under methods due to their names beginning and ending with double-underscores, to extend the functionality of custom class to support native functions such as print, length, comparison, support for arithmetic operations, type conversion, andmany more.

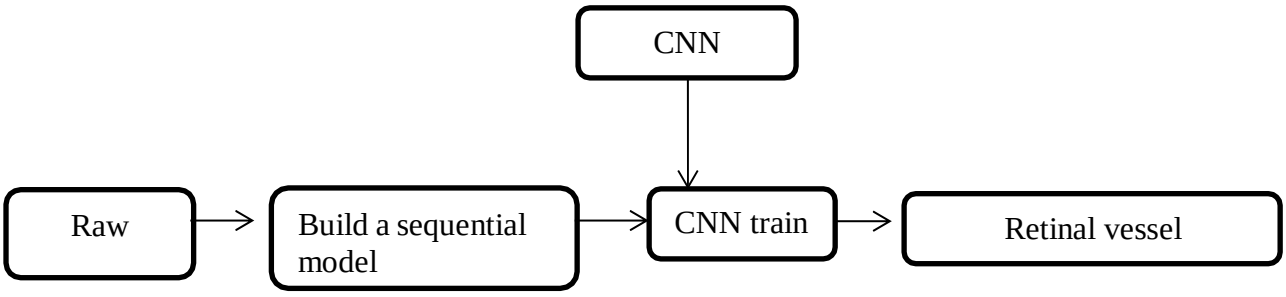
Typing :

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically-typed, Python is strongly-typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

8.6 METHODOLOGY

Preprocessing and Training the model (CNN): The dataset is preprocessed 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 different mask and normal images is obtained, 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 disease 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 brain tumor Classification image contained in the dataset. After successful training and preprocessing, comparison of the test image and trained model takes place to predict the brain tumor.

CHAPTER 9

LAYER

9.1 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.

9.2 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.

9.3 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.

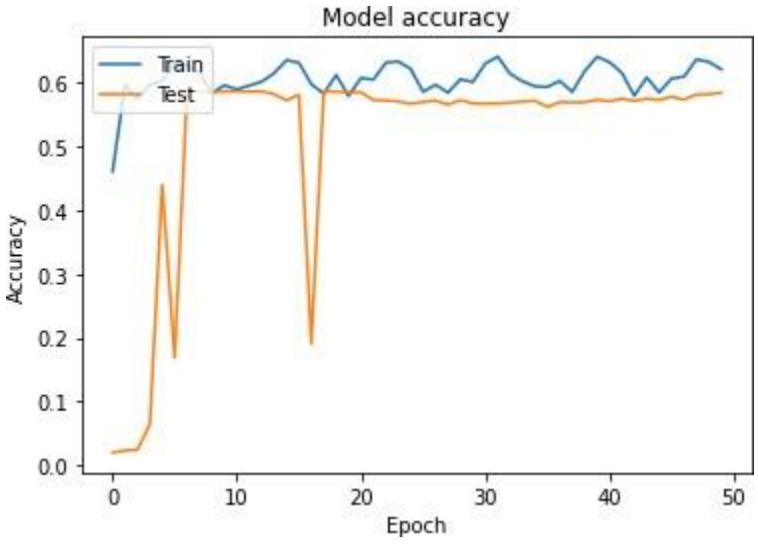
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 conv2d (Conv2D)             (None, 256, 256, 64)      1792      
                                                                 
 batch_normalization (BatchNormalization)  (None, 256, 256, 64)  256       
                                                                 
 activation (Activation)     (None, 256, 256, 64)      0         
                                                                 
 conv2d_1 (Conv2D)           (None, 256, 256, 64)      36928     
                                                                 
 batch_normalization_1 (BatchNormalization)  (None, 256, 256, 64)  256       
                                                                 
 activation_1 (Activation)   (None, 256, 256, 64)      0         
                                                                 
 max_pooling2d (MaxPooling2D)  (None, 128, 128, 64)    0         
                                                                 
 conv2d_2 (Conv2D)           (None, 128, 128, 12       73856     
```

9.4 Input Layer:

Input layer in CNN contain image data. Image data is represented by three dimensional matrixes. It needs to reshape it into a single column. Suppose you have image of dimension 28 x 28 =784, it need to convert it into 784 x 1 before feeding into input.

9.5 Convo Layer:

Convo layer is sometimes called feature extractor layer because features of the image are get extracted within this layer. First of all, a part of image is connected to Convo layer to perform convolution operation as we saw earlier and calculating the dot product between receptive field (it is a local region of the input image that has the same size as that of filter) and the filter. Result of the operation is single integer of the output volume. Then the filter over the next receptive field of the same input image by a Stride and do the same operation again. It will repeat the same process again and again until it goes through the whole image. The output will be the input for the next layer.



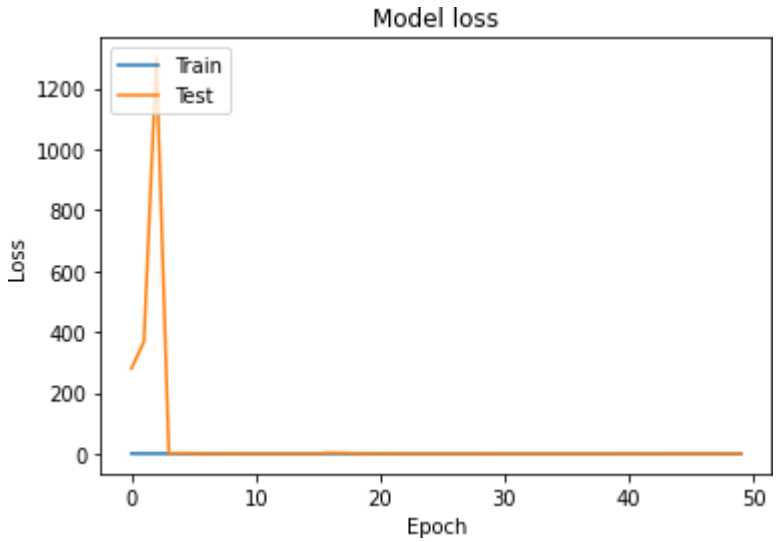
CNN model trained dataset accuracy

Pooling Layer:

Pooling layer is used to reduce the spatial volume of input image after convolution. It is used between two convolution layers. If it applies FC after Convo layer without applying pooling or max pooling, then it will be computationally expensive. So, the max pooling is only way to reduce the spatial volume of input image. It has applied max pooling in single depth slice with Stride of 2. It can observe the 4 x 4 dimension input is reducing to 2 x 2 dimensions.

Softmax / Logistic Layer:

Softmax or Logistic layer is the last layer of CNN. It resides at the end of FC layer. Logistic is used for binary classification and softmax is for multi-classification.



CNN model trained dataset loss values

11.6 Output Layer:

Output layer contains the label which is in the form of one-hot encoded. Now you have a good understanding of CNN.

.Libraries Required:

- ✓ **tensorflow**: Just to use the tensor board to compare the loss and adam curve our result data or obtained log.
- ✓ **keras**: To pre-process the image dataset.
- ✓ **matplotlib**: To display the result of our predictive outcome.
- ✓ **os**: To access the file system to read the image from the train and test directory from our machines.

TensorFlow:

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.

TensorFlow is a software library or framework, designed by the Google team to implement machine learning and deep learning concepts in the easiest manner. It combines the computational algebra of optimization techniques for easy calculation of many mathematical expressions.

Let us now consider the following important features of TensorFlow −

- It includes a feature of that defines, optimizes and calculates mathematical expressions easily with the help of multi-dimensional arrays called tensors.
- It includes a programming support of deep neural networks and machine learning techniques.
- It includes a high scalable feature of computation with various data sets.
- TensorFlow uses GPU computing, automating management. It also includes a unique feature of optimization of same memory and the data used.

CHAPTER 10

CODING

13.CODING:

Module 1:

Data Analysis:

In [ ]:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from os import listdir
import glob
from PIL import Image
import cv2
from random import randrange
from sklearn.model_selection import train_test_split
```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
ori_images = 'datasets/Images'
mask_images = 'datasets/Masks'
```

In [ ]:

```
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(7, n, idx+1)
        img = plt.imread(img_path)
        plt.imshow(img, cmap='gray')
```

```
        plt.axis('off')

    plt.tight_layout()
```

In [ ]:

```
def Images_details_Print_data(data, path):
    print(" ====== Images in: ", path)
    for k, v in data.items():
        print("%s:\t%s" % (k, v))

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  # No image will be bigger than that
    data['max_width'] = 0
    data['min_height'] = 10**100  # No image will be bigger than that
    data['max_height'] = 0


    for f in files:
        im = Image.open(f)
        width, height = im.size
        data['min_width'] = min(width, data['min_width'])
        data['max_width'] = max(width, data['max_height'])
        data['min_height'] = min(height, data['min_height'])
        data['max_height'] = max(height, data['max_height'])

    Images_details_Print_data(data, path)
```

In [ ]:

```
print("")
print("Train data for Orinigal Images:")
print("")
Images_details(ori_images)
print("")
plot_images(ori_images, 10)
```

In [ ]:

```
print("")
print("Train data for Mask images:")
print("")
Images_details(mask_images)
print("")
```

```
plot_images(mask_images, 10)
```
In [ ]:

```
def load_data(split=0.2):
    images = [f for f in glob.glob(ori_images + "/*.jpg", recursive=True)]
    masks = [f for f in glob.glob(mask_images + "/*.jpg", recursive=True)]

    train_x, test_x = train_test_split(images, test_size=split, random_state=42)
    train_y, test_y = train_test_split(masks, test_size=split, random_state=42)

    return (train_x, train_y), (test_x, test_y)
```
In [ ]:

```
dataset_path = "datasets"
(train_x, train_y), (test_x, test_y) = load_data()

print(f"Train: {len(train_x)} - {len(train_y)}")
print(f"Test: {len(test_x)} - {len(test_y)}")
```
In [ ]:

```
train_x
```
In [ ]:

```
index = randrange(0, len(train_x))
check_i = plt.imread(train_x[index])
check_m = plt.imread(train_y[index])
fig = plt.figure(figsize=(10, 7))
fig.add_subplot(1, 2, 1)
plt.imshow(check_i)
plt.axis('off')
plt.title("Original")
fig.add_subplot(1, 2, 2)
plt.imshow(check_i)
plt.axis('off')
plt.title("Mask")
plt.imshow(check_m)
```
In [ ]:

In [ ]:

MODULE – 2:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from os import listdir
import glob
from PIL import Image
import cv2
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation,
MaxPool2D, Conv2DTranspose, Concatenate, Input
from tensorflow.keras.models import Model
```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
ori_images = 'datasets/Images'
mask_images = 'datasets/Masks'
```

In [ ]:

```
def load_data(split=0.2):
    images = [f for f in glob.glob(ori_images + "/*.jpg", recursive=True)]
    masks = [f for f in glob.glob(mask_images + "/*.jpg", recursive=True)]

    train_x, test_x = train_test_split(images, test_size=split, random_state=42)
    train_y, test_y = train_test_split(masks, test_size=split, random_state=42)

    return (train_x, train_y), (test_x, test_y)
```

In [ ]:

```
def read_image(path):
    path = path.decode()
    x = cv2.imread(path, cv2.IMREAD_COLOR)  ## (H, W, 3)
    x = cv2.resize(x, (128, 128))
    x = x/255.0
    x = x.astype(np.float32)
    return x
```

In [ ]:

```
def read_mask(path):
    path = path.decode()
    x = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    x = cv2.resize(x, (128, 128))
    x = x/255.0
    x = x.astype(np.float32)
    x = np.expand_dims(x, axis=-1)
    return x
```

In [ ]:

```
def tf_parse(x, y):
    def _parse(x, y):
        x = read_image(x)
        y = read_mask(y)
        return x, y

    x, y = tf.numpy_function(_parse, [x, y], [tf.float32, tf.float32])
    x.set_shape([128, 128, 3])
    y.set_shape([128, 128, 1])
    return x, y

def tf_dataset(X, Y, batch):
    dataset = tf.data.Dataset.from_tensor_slices((X, Y))
    dataset = dataset.map(tf_parse)
    dataset = dataset.batch(batch)
    dataset = dataset.prefetch(10)
    return dataset
```

In [ ]:

```
(train_x, train_y), (test_x, test_y) = load_data()

print(f"Train: {len(train_x)} - {len(train_y)}")
print(f"Test: {len(test_x)} - {len(test_y)}")
```

In [ ]:

```
batch = 4
train_dataset = tf_dataset(train_x, train_y, batch)
```

```
valid_dataset = tf_dataset(test_x, test_y, batch)
```

In [ ]:

```
def conv_block(inputs, num_filters):
    x = Conv2D(num_filters, 3, padding="same")(inputs)
    x = Activation("relu")(x)

    x = Conv2D(num_filters, 3, padding="same")(x)
    x = Activation("relu")(x)

    return x

def encoder_block(inputs, num_filters):
    x = conv_block(inputs, num_filters)
    p = MaxPool2D((2, 2))(x)
    return x, p

def decoder_block(inputs, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(inputs)
    x = Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x

def build_unet(input_shape):
    inputs = Input(input_shape)

    """ Encoder """
    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)

    """ Bridge """
    b1 = conv_block(p2, 1024)

    """ Decoder """
    d1 = decoder_block(b1, s2, 128)
    d2 = decoder_block(d1, s1, 64)

    """ Outputs """
    outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d2)

    """ Model """
    model = Model(inputs, outputs)
    return model
```

In [ ]:

```
model = build_unet((128, 128, 3))
model.compile(loss="binary_crossentropy", optimizer=Adam(), metrics=['accuracy'])
model.summary()
```

In [ ]:

```
model_path = 'files/model.h5'
callbacks = [
        ModelCheckpoint(model_path, verbose=1, save_best_only=True)
    ]
```

In [ ]:

```
batch_size = 32
num_epoch = 20
```

In [ ]:

```
model.fit(
    train_dataset.repeat(),
    epochs=num_epoch,
    validation_data=valid_dataset.repeat(),
    steps_per_epoch=len(train_x)//batch_size,
    validation_steps=len(test_x)//batch_size,
    callbacks=callbacks
)
```

In [ ]:

```
def graph():
    #Plot training & validation accuracy values
    plt.plot(model.history.history['accuracy'])
    plt.plot(model.history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    # Plot training & validation loss values
    plt.plot(model.history.history['loss'])
    plt.plot(model.history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
```

graph()

In [ ]:

MODULE - 3

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from os import listdir
import glob
from PIL import Image
import cv2
from random import randrange
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool2D, Conv2DTranspose, Concatenate, Input
from tensorflow.keras.models import Model
```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
ori_images = 'datasets/Images'
mask_images = 'datasets/Masks'
```

In [ ]:

```
def load_data(split=0.2):
    images = [f for f in glob.glob(ori_images + "/*.jpg", recursive=True)]
    masks = [f for f in glob.glob(mask_images + "/*.jpg", recursive=True)]

    train_x, test_x = train_test_split(images, test_size=split, random_state=42)
    train_y, test_y = train_test_split(masks, test_size=split, random_state=42)
```

```
    return (train_x, train_y), (test_x, test_y)
```

In [ ]:

```
def read_image(path):
    path = path.decode()
    x = cv2.imread(path, cv2.IMREAD_COLOR)
    x = cv2.resize(x, (256, 256))
    x = x/255.0
    x = x.astype(np.float32)
    return x
```

In [ ]:

```
def read_mask(path):
    path = path.decode()
    x = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    x = cv2.resize(x, (256, 256))
    x = x/255.0
    x = x.astype(np.float32)
    x = np.expand_dims(x, axis=-1)
    return x
```

In [ ]:

```
def tf_parse(x, y):
    def _parse(x, y):
        x = read_image(x)
        y = read_mask(y)
        return x, y

    x, y = tf.numpy_function(_parse, [x, y], [tf.float32, tf.float32])
    x.set_shape([256, 256, 3])
    y.set_shape([256, 256, 1])
    return x, y

def tf_dataset(X, Y, batch):
    dataset = tf.data.Dataset.from_tensor_slices((X, Y))
    dataset = dataset.map(tf_parse)
    dataset = dataset.batch(batch)
    dataset = dataset.prefetch(10)
    return dataset
```

In [ ]:

```
(train_x, train_y), (test_x, test_y) = load_data()

print(f"Train: {len(train_x)} - {len(train_y)}")
print(f"Test: {len(test_x)} - {len(test_y)}")
```

In [ ]:

```
batch = 4
train_dataset = tf_dataset(train_x, train_y, batch)
valid_dataset = tf_dataset(test_x, test_y, batch)
```

In [ ]:

```
def conv_block(inputs, num_filters):
    x = Conv2D(num_filters, 3, padding="same")(inputs)
    x = BatchNormalization()(x
x = Activation("relu")(x)

    x = Conv2D(num_filters, 3, padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    return x

def encoder_block(inputs, num_filters):
    x = conv_block(inputs, num_filters)
    p = MaxPool2D((2, 2))(x)
    return x, p
def build_unet(input_shape):
    inputs = Input(input_shape)

    """ Encoder """
    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    """ Bridge """
    b1 = conv_block(p4, 1024)

    """ Decoder """
    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    """ Outputs """
    outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4)

     model = Model(inputs, outputs)
    return model
```

In [ ]:

```
model = build_unet((256, 256, 3))
model.compile(loss="binary_crossentropy", optimizer=Adam(), metrics=['accuracy'])
model.summary()
```

In [ ]:

```
model_path = 'files/final_model.h5'
callbacks = [
        ModelCheckpoint(model_path, verbose=1, save_best_only=True)
    ]
model.fit(
    train_dataset.repeat(),
    epochs=num_epoch,
    validation_data=valid_dataset.repeat(),
    steps_per_epoch=len(train_x)//batch_size,
    validation_steps=len(test_x)//batch_size,
    callbacks=callbacks
)
```

In [ ]:

```
def graph():
    #Plot training & validation accuracy values
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
    plt.plot(model.history.history['loss'])
    plt.plot(model.history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

graph()
```

In [ ]:

In [ ]:

CHAPTER 11

CONCLUSION

11 . CONCLUTION:

Image segmentation is a very useful task in computer vision that can be applied to a variety of use-cases in medical domain. Here we analyse the data how it is available and train the data. The best architecture is picked form the trained architecture and it is converted into model. The bulid model is used in django framework to segment the retinal fundus image. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex

11.1 FUTURE WORK:

Further improvement in the network's accuracy can be achieved through the following practices. Proper pre-processing techniques can be used to generalize the data. Data augmentation can be done to manipulate the data to get more accuracy. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex

CHAPTER 12

REFERENCE

12.REFERENCE

- [1] P. Saeedi et al., “Global and regional diabetes prevalence estimates for 2019 and projections for 2030 and 2045: Results from the international diabetes federation diabetes atlas,” Diabetes Res. Clin. Pract., vol. 157, 2019, Art. no. 107843.
- [2] T. Y. Wong and C. Sabanayagam, “Strategies to tackle the global burden of diabetic retinopathy: From epidemiology to artificial intelligence,” Ophthalmologica, vol. 243, no. 1, pp. 9–20, 2020.
- [3] C. Bascaran et al., “Effectiveness of task-shifting for the detection of diabetic retinopathy in low-and middle-income countries: A rapid review protocol,” Systematic Rev., vol. 10, no. 1, pp. 1–5, 2021.
- [4] R. Klein, B. E. Klein, and S. E. Moss, “Visual impairment in diabetes,” Ophthalmology, vol. 91, no. 1, pp. 1–9, 1984.
- [5] M. M. Engelgau et al., “The evolving diabetes burden in the United States,” Ann. Intern. Med., vol. 140, no. 11, pp. 945–950, 2004.
- [6] R. Rajalakshmi, R. Subashini, R. M. Anjana, and V. Mohan, “Automated diabetic retinopathy detection in smartphone-based fundus photography using artificial intelligence,” Eye, vol. 32, no. 6, pp. 1138–1144, 2018.
- [7] A. Grzybowski et al., “Artificial intelligence for diabetic retinopathy screening: A review,” Eye, vol. 34, no. 3, pp. 451–460, 2020.

- [8] R. E. Hacisoftaoglu, M. Karakaya, and A. B. Sallam, “Deep learning frameworks for diabetic retinopathy detection with smartphone-based retinal imaging systems,” Pattern Recognit. Lett., vol. 135, pp. 409–417, 2020.
- [9] M. Karakaya and R. E. Hacisoftaoglu, “Comparison of smartphone-based retinal imaging systems for diabetic retinopathy detection using deep learning,” BMC Bioinf., vol. 21, no. 4, pp. 1–18, 2020.
- [10] G. Quellec, K. Charrière, Y. Boudi, B. Cochener, and M. Lamard, “Deep image mining for diabetic retinopathy screening,” Med. Image Anal., vol. 39, pp. 178–193, 2017.
- [11] R. Gargeya and T. Leng, “Automated identification of diabetic retinopathy using deep learning,” Ophthalmology, vol. 124, no. 7, pp. 962–969, 2017.
- [12] M. D. Abràmoff et al., “Improved automated detection of diabetic retinopathy on a publicly available dataset through integration of deep learning,” Invest. Ophthalmol. Vis. Sci., vol. 57, no. 13, pp. 5200–5206, 2016.
- [13] A. Rakhlin, “Diabetic retinopathy detection through integration of deep learning classification framework,” 2018, arXiv:225508.
- [14] V. Gulshan et al., “Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs,” Jama, vol. 316, no. 22, pp. 2402–2410, 2016.
- [15] M. Voets, K. Møllersen, and L. A. Bongo, “Reproduction study using public data of: Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs,” PLoS One, vol. 14, no. 6, 2019, Art. no. e0217541.
- [16] B. Graham, “Kaggle Diabetic Retinopathy Detection competition report.” Kaggle, 2015. Accessed: Jun. 23, 2019, [Online]. Available: https://www.kaggle.com/c/diabetic-retinopathy-detection/discussion/15801#latest-370950.

- [17] M. Antony, and S. Bruggemann, “Kaggle Diabetic Retinopathy Detection Team o_O solution.” Kaggle, 2015. Accessed: Jun. 23, 2019. [Online]. Available:https://www.kaggle.com/c/diabeticretinopathydetection/discussion /15807

- [18] J. Xu, J. Dunavent, and R. Kainkaryam, “Summary of our solution to the kaggle diabetic retinopathy detection competition,” Kaggle, 2015. Accessed: Jun. 23, 2019. [Online]. Available: https://www.kaggle.com/ c/diabetic-retinopathy-detection/discussion/15845
- [19] P. Porwal et al., “Indian diabetic retinopathy image dataset (IDRID): A database for diabetic retinopathy screening research,” Data, vol. 3, no. 3, 2018, Art. no. 25.
- [20] Y. Deng, “Deep learning on mobile devices: A review,” Mobile Multimedia/Image Process., Secur., Appl., vol. 10993, 2019, Art. no. 109930A.
- [21] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, “On the dangers of stochastic parrots: Can language models be too big?,” in Proc. ACM Conf. Fairness, Accountability, Transparency, 2021, pp. 610–623.
- [22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 2818–2826.
- [23] B. Graham, Kaggle Diabetic Retinopathy Detection Competition Report. Coventry, U.K.: Univ. Warwick, 2015.
- [24] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, “Network decoupling: From regular to depthwise separable convolutions,” 2018, arXiv:1808.05517.
- [25] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 1800–1807.

- [26] A. G. Howard et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861. [28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2018, pp. 4510–4520 .