

1. INTRODUCTION

Image classification is the primary domain, in which deep neural networks play the most important role of medical image analysis. The image classification accepts the given input images and produces output classification for identifying whether the disease is present or not. In our model we mainly classify the different types of organs and predict the accuracy.

One of the most imperative problems faced in the domain area of image recognition is the classification of medical images. The major intention of medical image classification is to classify medical images into several elements to assist medical practitioners or physicists in diagnosing disease. Hence, medical image classification is split into two steps. The first and foremost step of medical image classification is to extract the essential features from the acquired input image. The second step in medical image classification is utilizing the features to construct models that classify the image data set. In the recent past, medical practitioners customarily utilized their specialized experience to extract features so that classification of medical images could be performed into several classes. However, this manual medical image classification was found to be highly cumbersome and time consuming.

Medical image classification involves the process of segregating medical-related information into a useful form. Classification of medical images is based on placing image pixels with similar values into groups. With the placement of similar values into groups, common pixels are identified and are denoted by these pixels. Hence, a correctly classified image usually denotes the areas on the ground that share specific features as specified in the classification scheme.

Image classification is where a computer can analyse an image and identify the ‘class’ the image falls under. (Or a probability of the image being part of a ‘class’.) A class is essentially a label, for instance, ‘car’, ‘animal’, ‘building’ and so on. For example, you input an image of a sheep. Image classification is the process of the computer analysing the image and telling you it’s a sheep. (Or the probability that it’s a sheep.) For us, classifying images is no big deal. But it’s a perfect example of Moravec’s paradox when it comes to machines. (That is, the things we find easy are difficult for AI.)

Early image classification relied on raw pixel data. This meant that computers would break down images into individual pixels. The problem is that two pictures of the same thing can look very different. They can have different backgrounds, angles, poses, and etcetera. This made it quite the challenge for computers to correctly ‘see’ and categorise images.

Deep learning is a type of machine learning; a subset of artificial intelligence (AI) that allows machines to learn from data. Deep learning involves the use of computer systems known as neural networks. In neural networks, the input filters through hidden layers of nodes. These nodes each process the input and communicate their results to the next layer of nodes. This repeats until it reaches an output layer, and the machine provides its answer.

There are different types of neural networks based on how the hidden layers work. Image classification with deep learning most often involves convolutional neural networks, or CNNs. In CNNs, the nodes in the hidden layers don’t always share their output with every node in the next layer (known as convolutional layers). Deep learning allows machines to identify and extract features from images. This means they can learn the features to look for in images by analysing lots of pictures. So, programmers don’t need to enter these filters by hand.

Image classification has a few uses — and vast potential as it grows in reliability. Here are just a few examples of what makes it useful. Self-driving cars use image classification to identify what’s around them. I.e., trees, people, traffic lights and so on. Image classification can also help in healthcare. For instance, it could analyse medical images and suggest whether they classify as depicting a symptom of illness. Or, for example, image classification could help people organise their photo collections.

2. IMPLEMENTATION

TECHNOLOGIES USED:

2.1 PYTHON:

What is a Script?

Up to this point, I have concentrated on the interactive programming capability of Python. This is a very useful capability that allows you to type in a program and to have it executed immediately in an interactive mode

Scripts are reusable

Basically, a script is a text file containing the statements that comprise a Python program. Once you have created the script, you can execute it over and over without having to retype it each time.

Scripts are editable

Perhaps, more importantly, you can make different versions of the script by modifying the statements from one file to the next using a text editor. Then you can execute each of the individual versions. In this way, it is easy to create different programs with a minimum amount of typing.

You will need a text editor

Just about any text editor will suffice for creating Python script files.

You can use Microsoft Notepad, Microsoft WordPad, Microsoft Word, or just about any word processor if you want to.

Difference between a script and a program

Script:

Scripts are distinct from the core code of the application, which is usually written in a different language, and are often created or at least modified by the end-user. Scripts are often interpreted

from source code or byte code, whereas the applications they control are traditionally compiled to native machine code.

Program:

The program has an executable form that the computer can use directly to execute the instructions.

The same program in its human-readable source code form, from which executable programs are derived (e.g., compiled)

Python:

What is Python? Chances you are asking yourself this. You may have found this book because you want to learn to program but don't know anything about programming languages. Or you may have heard of programming languages like C, C++, C#, or Java and want to know what Python is and how it compares to "big name" languages. Hopefully I can explain it for you.

Python concepts

If you're not interested in the how's and whys of Python, feel free to skip to the next chapter. In this chapter I will try to explain to the reader why I think Python is one of the best languages available and why it's a great one to start programming with.

- Open-source general-purpose language.
- Object Oriented, Procedural, Functional
- Easy to interface with C/ObjC/Java/Fortran
- Easy-is to interface with C++ (via SWIG)
- Great interactive environment

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive – you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python:

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and UNIX shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features:

Python's features include –

- Easy-to-learn – Python has few keywords, simple structure, and a clearly defined Syntax. This allows the student to pick up the language quickly.
- Easy-to-read – Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain – Python's source code is fairly easy-to-maintain.

- A broad standard library – Python's bulk of the library is very portable and Crossplatform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode – Python has support for an interactive mode which allows interactivetesting and debugging of snippets of code.
- Portable – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable – you can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases – Python provides interfaces to all major commercial databases.
- GUI Programming – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable – Python provides a better structure and support for large programs than shell scripting.
- Apart from the above-mentioned features, Python has a big list of good features, few are listed below –
 - It supports functional and structured programming methods as well as OOP.
 - It can be used as a scripting language or can be compiled to byte-code for building large applications.
 - It provides very high-level dynamic data types and supports dynamic type checking.
 - IT supports automatic garbage collection.
 - It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Dynamic vs. Static

Types Python is a dynamic-typed language. Many other languages are static typed, such as C/C++ and Java. A static typed language requires the programmer to explicitly tell the computer what type of “thing” each data value is.

For example, in C if you had a variable that was to contain the price of something, you would have to declare the variable as a “float” type.

This tells the compiler that the only data that can be used for that variable must be a floatingpoint number, i.e., a number with a decimal point.

If any other data value was assigned to that variable, the compiler would give an error when trying to compile the program.

Python, however, doesn't require this. You simply give your variables names and assign values to them. The interpreter takes care of keeping track of what kinds of objects your program is using. This also means that you can change the size of the values as you develop the program. Say you have another decimal number (a.k.a. a floating-point number) you need in your program.

With a static typed language, you have to decide the memory size the variable can take when you first initialize that variable. A double is a floating-point value that can handle a much larger number than a normal float (the actual memory sizes depend on the operating environment).

If you declare a variable to be a float but later on assign a value that is too big to it, your program will fail; you will have to go back and change that variable to be a double.

With Python, it doesn't matter. You simply give it whatever number you want and Python will take care of manipulating it as needed. It even works for derived values.

For example, say you are dividing two numbers. One is a floating-point number and one is an integer. Python realizes that it's more accurate to keep track of decimals so it automatically calculates the result as a floating-point number

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python Numbers:

Number data types store numeric values. Number objects are created when you assign a value to them

Python Strings:

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

Python Lists:

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type. The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Python Tuples:

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses. The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated. Tuples can be thought of as read-only lists.

Python Dictionary:

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({}) and values can be assigned and accessed using square braces ([]).

Different modes in python:

Python has two basic modes: normal and interactive.

The normal mode is the mode where the scripted and finished .py files are run in the Python interpreter.

Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole

20 Python libraries:

1. Requests. The most famous http library written by Kenneth remits. It's a must have for every python developer.
2. Scrappy. If you are involved in web scraping then this is a must have library for you. After using this library, you won't use any other.
3. Python. A guy toolkit for python. I have primarily used it in place of tinder. You will really love it.
4. Pillow. A friendly fork of PIL (Python Imaging Library). It is more user friendly than PIL and is a must have for anyone who works with images.
5. SQL Alchemy. A database library. Many love it and many hate it. The choice is yours.
6. Beautiful Soup. I know it's slow but this xml and html parsing library is very useful for beginners.

7. Twisted. The most important tool for any network application developer. It has a very beautiful ape and is used by a lot of famous python developers.
8. Numbly. How can we leave this very important library? It provides some advance math functionalities to python.

9. Skippy. When we talk about numbly then we have to talk about spicy. It is a library of algorithms and mathematical tools for python and has caused many scientists to switch from ruby to python.
10. Matplotlib. A numerical plotting library. It is very useful for any data scientist or any data analyser.
11. Pygmy. Which developer does not like to play games and develop them? This library will help you achieve your goal of 2d game development.
12. Piglet. A 3d animation and game creation engine. This is the engine in which the famous python port of mine craft was made
13. Pit. A GUI toolkit for python. It is my second choice after python for developing GUIs for my python scripts.
14. Pit. Another python GUI library. It is the same library in which the famous Bit torrent client is created.
15. Scaly. A packet sniffer and analyser for python made in python.
16. Pywin32. A python library which provides some useful methods and classes for interacting with windows.
17. Notch. Natural Language Toolkit – I realize most people won't be using this one, but it's generic enough. It is a very useful library if you want to manipulate strings. But its capacity is beyond that. Do check it out.
18. Nose. A testing framework for python. It is used by millions of python developers. It is a must have if you do test driven development.
19. Simply. Simply can-do algebraic evaluation, differentiation, expansion, complex numbers, etc. It is contained in a pure Python distribution.
20. I Python. I just can't stress enough how useful this tool is. It is a python prompt on steroids. It has completion, history, shell capabilities, and a lot more. Make sure that you take a look at it.

NumPy:

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called axes. The number of axes is rank.

- Offers Matlab-like capabilities within Python
- Fast array operations
- 2D arrays, multi-D arrays, linear algebra etc.

Matplotlib

- High quality plotting library.

Python class and objects

These are the building blocks of OOP. Class creates a new object. This object can be anything, whether an abstract data concept or a model of a physical object, e.g., a chair. Each class has individual characteristics unique to that class, including variables and methods.

Classes are very powerful and currently “the big thing” in most programming languages. Hence, there are several chapters dedicated to OOP later in the book.

The class is the most basic component of object-oriented programming. Previously, you learned how to use functions to make your program do something.

Now will move into the big, scary world of Object-Oriented Programming (OOP). To be honest, it took me several months to get a handle on objects.

When I first learned C and C++, I did great; functions just made sense for me.

Having messed around with BASIC in the early '90s, I realized functions were just like subroutines so there wasn't much new to learn.

However, when my C++ course started talking about objects, classes, and all the new features of OOP, my grades definitely suffered.

Once you learn OOP, you'll realize that it's actually a pretty powerful tool. Plus, many Python libraries and APIs use classes, so you should at least be able to understand what the code is doing.

One thing to note about Python and OOP: it's not mandatory to use objects in your code in a way that works best; maybe you don't need to have a full-blown class with initialization code and methods to just return a calculation. With Python, you can get as technical as you want.

As you've already seen, Python can do just fine with functions. Unlike languages such as Java, you aren't tied down to a single way of doing things; you can mix functions and classes as necessary in the same program. This lets you build the code

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

Here's a brief list of Python OOP ideas:

- The class statement creates a class object and gives it a name. This creates a new namespace.
- Assignments within the class create class attributes. These attributes are accessed by qualifying the name using dot syntax: `ClassName.Attribute`.
- Class attributes export the state of an object and its associated behaviour. These attributes are shared by all instances of a class.
- Calling a class (just like a function) creates a new instance of the class.
- This is where the multiple copy's part comes in.
- Each instance gets ("inherits") the default class attributes and gets its own namespace. This prevents instance objects from overlapping and confusing the program.
- Using the term `self` identifies a particular instance, allowing for per-instance attributes. This allows items such as variables to be associated with a particular instance.

Inheritance

First off, classes allow you to modify a program without really making changes to it.

To elaborate, by subclassing a class, you can change the behaviour of the program by simply adding new components to it rather than rewriting the existing components.

As we've seen, an instance of a class inherits the attributes of that class.

However, classes can also inherit attributes from other classes. Hence, a subclass inherits from a superclass allowing you to make a generic superclass that is specialized via subclasses.

The subclasses can override the logic in a superclass, allowing you to change the behaviour of your classes without changing the superclass at all.

Operator Overloads

Operator overloading simply means that objects that you create from classes can respond to actions (operations) that are already defined within Python, such as addition, slicing, printing, etc.

Even though these actions can be implemented via class methods, using overloading ties the behaviour closer to Python's object model and the object interfaces are more consistent to Python's built-in objects, hence overloading is easier to learn and use.

User-made classes can override nearly all of Python's built-in operation methods

Exceptions

I've talked about exceptions before but now I will talk about them in depth. Essentially, exceptions are events that modify program's flow, either intentionally or due to errors.

They are special events that can occur due to an error, e.g., trying to open a file that doesn't exist, or when the program reaches a marker, such as the completion of a loop.

Exceptions, by definition, don't occur very often; hence, they are the "exception to the rule" and a special class has been created for them. Exceptions are everywhere in Python.

Virtually every module in the standard Python library uses them, and Python itself will raise them in a lot of different circumstances.

Here are just a few examples:

- Accessing a non-existent dictionary key will raise a Key Error exception.
- Searching a list for a non-existent value will raise a Value Error exception
- Calling a non-existent method will raise an Attribute Error exception.
- Referencing a non-existent variable will raise a Name Error exception.
- Mixing data types without coercion will raise a Type Error exception.

One use of exceptions is to catch a fault and allow the program to continue working; we have seen this before when we talked about files.

This is the most common way to use exceptions. When programming with the Python command line interpreter, you don't need to worry about catching exceptions.

Your program is usually short enough to not be hurt too much if an exception occurs.

Plus, having the exception occur at the command line is a quick and easy way to tell if your code logic has a problem.

However, if the same error occurred in your real program, it will fail and stop working. Exceptions can be created manually in the code by raising an exception.

It operates exactly as a system-caused exceptions, except that the programmer is doing it on purpose. This can be for a number of reasons. One of the benefits of using exceptions is that, by their nature, they don't put any overhead on the code processing.

Because exceptions aren't supposed to happen very often, they aren't processed until they occur.

Exceptions can be thought of as a special form of the if/elif statements. You can realistically do the same thing with if blocks as you can with exceptions.

However, as already mentioned, exceptions aren't processed until they occur; if blocks are processed all the time.

Proper use of exceptions can help the performance of your program.

The more infrequent the error might occur, the better off you are to use exceptions; using if blocks require Python to always test extra conditions before continuing.

Exceptions also make code management easier: if your programming logic is mixed in with error-handling if statements, it can be difficult to read, modify, and debug your program.

User-Defined Exceptions I won't spend too much time talking about this, but Python does allow for a programmer to create his own exceptions.

You probably won't have to do this very often but it's nice to have the option when necessary.

However, before making your own exceptions, make sure there isn't one of the built-in exceptions that will work for you.

They have been "tested by fire" over the years and not only work effectively, they have been optimized for performance and are bug-free.

Making your own exceptions involves object-oriented programming, which will be covered in the next chapter. To make a custom exception, the programmer determines which base exception to use as the class to inherit from, e.g., making an exception for negative numbers or one for imaginary numbers would probably fall under the Arithmetic Error exception class. To make a custom exception, simply inherit the base exception and define what it will do.

Python modules

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module.

Testing code

As indicated above, code is usually developed in a file using an editor.

To test the code, import it into a Python session and try to run it.

Usually there is an error, so you go back to the file, make a correction, and test again.

This process is repeated until you are satisfied that the code works. T

His entire process is known as the development cycle.

There are two types of errors that you will encounter. Syntax errors occur when the form of some command is invalid.

This happens when you make typing errors such as misspellings, or call something by the wrong name, and for many other reasons. Python will always give an error message for a syntax error.

Functions in Python

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function.

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task.

To carry out that specific task, the function might or might not need multiple inputs. When the task is carved out, the function can or cannot return one or more values.

There are three types of functions in python:

Help (), min (), print ().

Python Namespace

Generally speaking, a namespace (sometimes also called a context) is a naming system for making names unique to avoid ambiguity. Everybody knows a name spacing system from daily life, i.e., the naming of people in first name and family name (surname).

An example is a network: each network device (workstation, server, printer,) needs a unique name and address. Yet another example is the directory structure of file systems.

The same file name can be used in different directories, the files can be uniquely accessed via the pathnames.

Many programming languages use namespaces or contexts for identifiers. An identifier defined in a namespace is associated with that namespace.

This way, the same identifier can be independently defined in multiple namespaces. (Like the same file names in different directories) Programming languages, which support namespaces, may have different rules that determine to which namespace an identifier belongs.

Namespaces in Python are implemented as Python dictionaries, this means it is a mapping from names (keys) to objects (values). The user doesn't have to know this to write a Python program and when using namespaces.

Some namespaces in Python:

- global names of a module
- local names in a function or method invocation
- built-in names: this namespace contains built-in functions (e.g., `abs ()`, `camp ()`, ...) and built-in exception names

Garbage Collection

Garbage Collector exposes the underlying memory management mechanism of Python, the automatic garbage collector. The module includes functions for controlling how the collector operates and to examine the objects known to the system, either pending collection or stuck in reference cycles and unable to be freed.

Python XML Parser

XML is a portable, open-source language that allows programmers to develop applications that can be read by other applications, regardless of operating system and/or developmental language.

What is XML? The Extensible Markup Language XML is a markup language much like HTML or SGML.

This is recommended by the World Wide Web Consortium and available as an open standard.

XML is extremely useful for keeping track of small to medium amounts of data without requiring a SQL-based backbone.

XML Parser Architectures and APIs the Python standard library provides a minimal but useful set of interfaces to work with XML.

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

Simple API for XML SAX: Here, you register call-backs for events of interest and then let the parser proceed through the document.

This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.

Document Object Model DOM API: This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical tree – based form to represent all the features of an XML document.

SAX obviously cannot process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files.

SAX is read-only, while DOM allows changes to the XML file. Since these two different APIs literally complement each other, there is no reason why you cannot use them both for large projects.

Python Web Frameworks

A web framework is a code library that makes a developer's life easier when building reliable, scalable and maintainable web applications.

Why are web frameworks useful?

Web frameworks encapsulate what developers have learned over the past twenty years while programming sites and applications for the web. Frameworks make it easier to reuse code for

common HTTP operations and to structure projects so other developers with knowledge of the framework can quickly build and maintain the application.

Common web framework functionality

Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications. These common operations include:

1. URL routing
2. HTML, XML, JSON, and other output format templating
3. Database manipulation
4. Security against Cross-site request forgery (CSRF) and other attacks
5. Session storage and retrieval

Not all web frameworks include code for all of the above functionality. Frameworks fall on the spectrum from executing a single use case to providing every known web framework feature to every developer. Some frameworks take the "batteries-included" approach where everything possibly comes bundled with the framework while others have a minimal core package that is amenable to extensions provided by other packages.

Comparing web frameworks

There is also a repository called [compare-python-web-frameworks](#) where the same web application is being coded with varying Python web frameworks, templating engines and object.

Web framework resources

- When you are learning how to use one or more web frameworks it's helpful to have an idea of what the code under the covers is doing.
- Frameworks is a really well-done short video that explains how to choose between web frameworks. The author has some particular opinions about what should be in a framework. For the most part I agree although I've found sessions and database ORMs to be a helpful part of a framework when done well.

- What is a web framework? Is an in-depth explanation of what web frameworks being and their relation to web servers?
- Jingo vs. Flash vs. Pyramid: Choosing a Python web framework contains background information and code comparisons for similar web applications built in these three big Python frameworks.
- This fascinating blog post takes a look at the code complexity of several Python web frameworks by providing visualizations based on their code bases.
- Python's web frameworks benchmarks are a test of the responsiveness of a framework with encoding an object to JSON and returning it as a response as well as retrieving data from the database and rendering it in a template. There were no conclusive results but the output is fun to read about nonetheless.
- What web frameworks do you use and why are they awesome? Is a language agnostic Reedit discussion on web frameworks? It's interesting to see what programmers in other languages like and dislike about their suite of web frameworks compared to the main Python frameworks.
- This user-voted question & answer site asked "What are the best general purpose Python web frameworks usable in production?" The votes aren't as important as the list of the many frameworks that are available to Python developers.

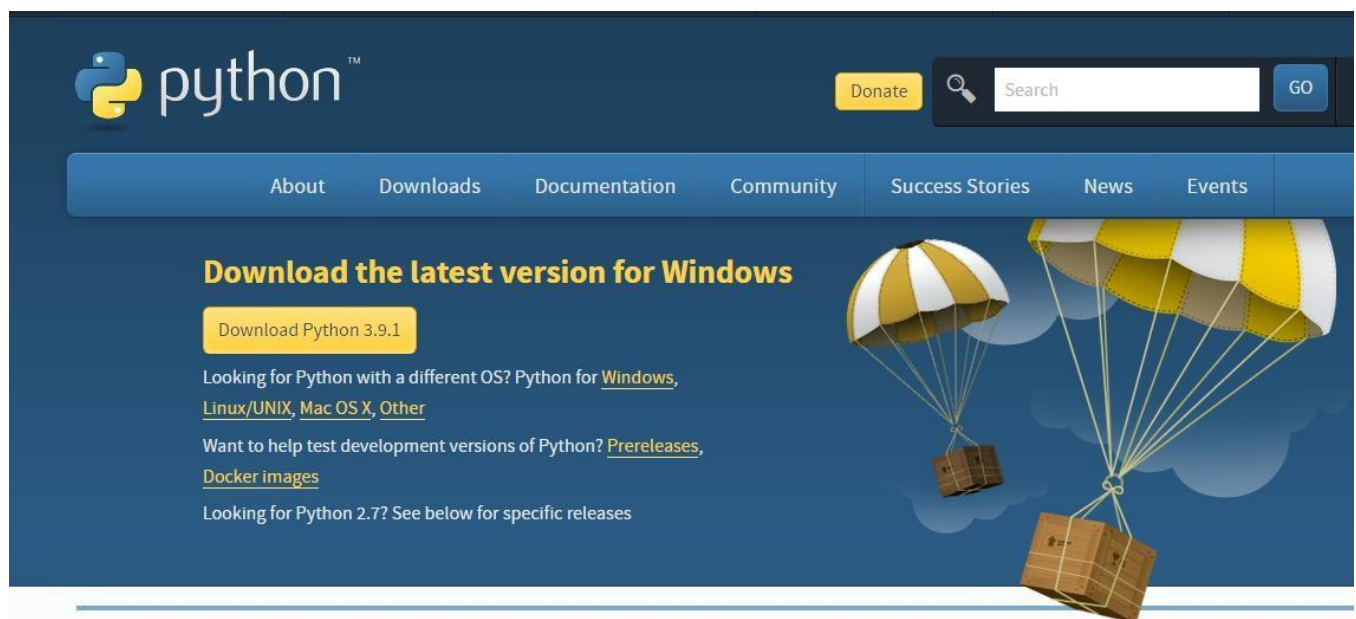
Web frameworks learning checklist

1. Choose a major Python web framework (Jingo or Flask are recommended) and stick with it. When you're just starting it's best to learn one framework first instead of bouncing around trying to understand every framework.
2. Work through a detailed tutorial found within the resource's links on the framework's page.
3. Study open-source examples built with your framework of choice so you can take parts of those projects and reuse the code in your application.
4. Build the first simple iteration of your web application then go to the deployment section to make it accessible on the web.

2.2 SOFTWARE INSTALLATION FOR MACHINE LEARNING PROJECTS

Installing Python:

1. To download and install Python visit the official website of Python <https://www.python.org/downloads/> and choose your version.



2. Once the download is complete, run the exe for install Python. Now click on Install Now.

3. You can see Python installing at this point.
4. When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".

Installing PyCharm:

1. To download PyCharm visit the website <https://www.jetbrains.com/pycharm/download/> and Click the "DOWNLOAD" link under the Community Section.

Download PyCharm

Windows

Mac

Linux

Professional

For both Scientific and Web Python development. With HTML, JS, and SQL support.

Download

Free trial

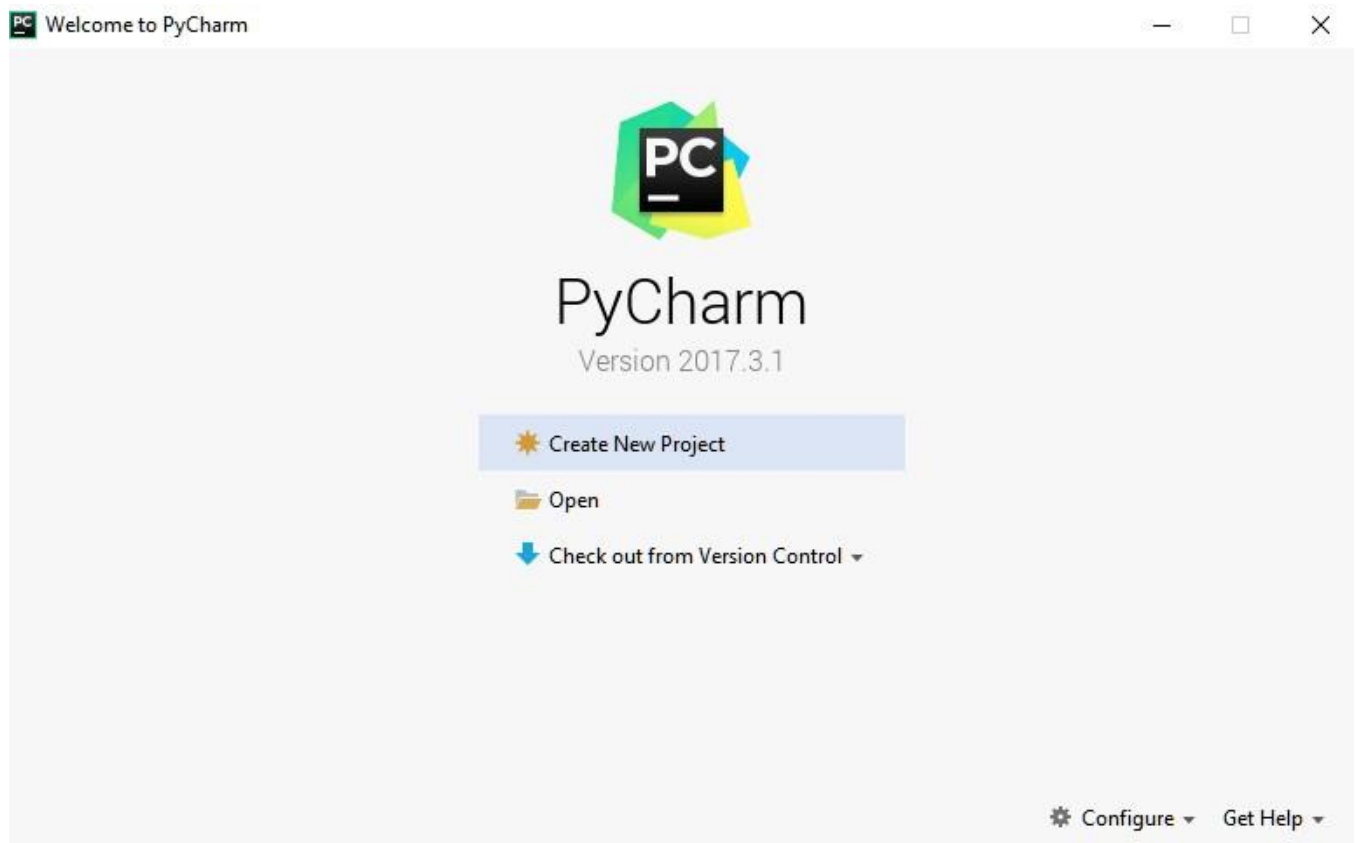
Community

For pure Python development

Download

Free, open-source

2. Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click "Next".
3. On the next screen, Change the installation path if required. Click "Next".
4. On the next screen, you can create a desktop shortcut if you want and click on "Next".
5. Choose the start menu folder. Keep selected Jet Brains and click on "Install".
6. Wait for the installation to finish.
7. Once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the "Run PyCharm Community Edition" box first and click "Finish".
8. After you click on "Finish," the Following screen will appear.



9. You need to install some packages to execute your project in a proper way.
10. Open the command prompt/ anaconda prompt or terminal as administrator.
11. The prompt will get open, with specified path, type “pip install package name” which you want to install (like NumPy, pandas, seaborn, scikit-learn, matplotlib.pyplot) Ex: pip install NumPy

```
C:\WINDOWS\system32>pip install numpy==1.18.5
Collecting numpy==1.18.5
  Downloading numpy-1.18.5-cp36-cp36m-win_amd64.whl (12.7 MB)
    | 12.7 MB 939 kB/s
ERROR: tensorboard 2.0.2 has requirement setuptools>=41.0.0, b
Installing collected packages: numpy
Successfully installed numpy-1.18.5
```


3. TESTING

SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

Types of Tests

Unit testing:

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successful unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g., components in a software system or – one step up – software applications at the company level interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

In future, we aim to explore large-scale image datasets for medical image classification and detection problems. And we can go for another types of pre-trained algorithms that can perform well and gives high classification. By which, we can classify different types of organs and detect the problems easily

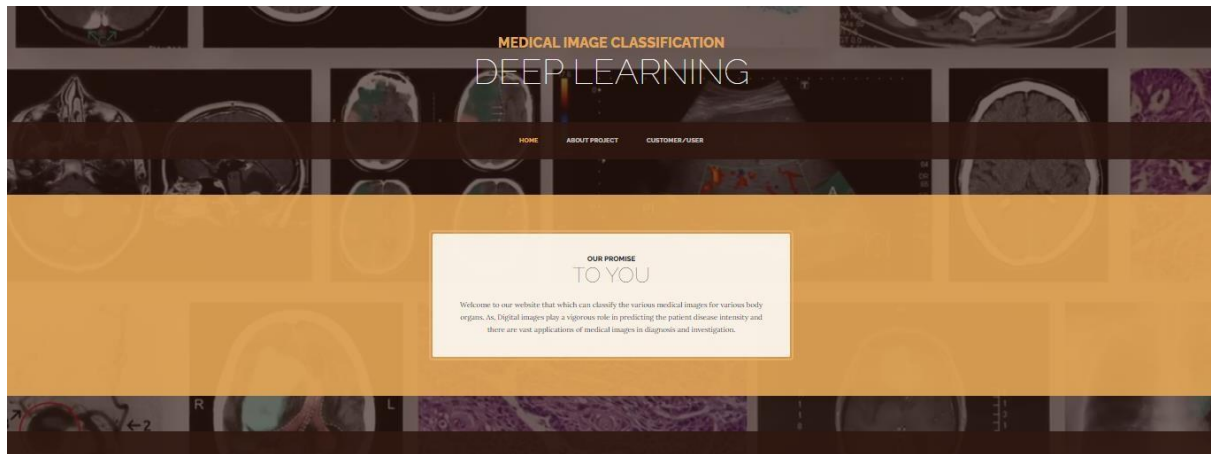
5. CONCLUSION

For classification purpose, deep learning-based framework for medical image classification by training the images is proposed. In this regard, diagnosis is one of the main requirements of the

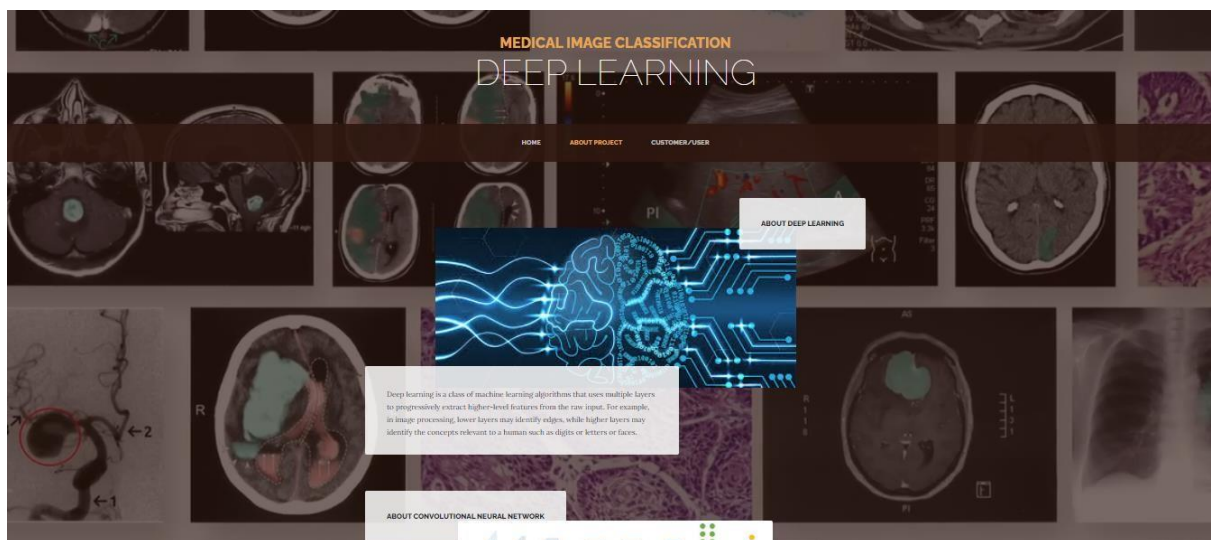
Dept Of MCA **Page 29**

existing era and investigated or examine to specific diseases. Hence, we have proposed a novel deep convolution network-based approach that is assist of doctors and physicians in making reasonable decisions. The results obtained from the proposed method outperformed state-of-the-art methods that is reported for the same dataset.

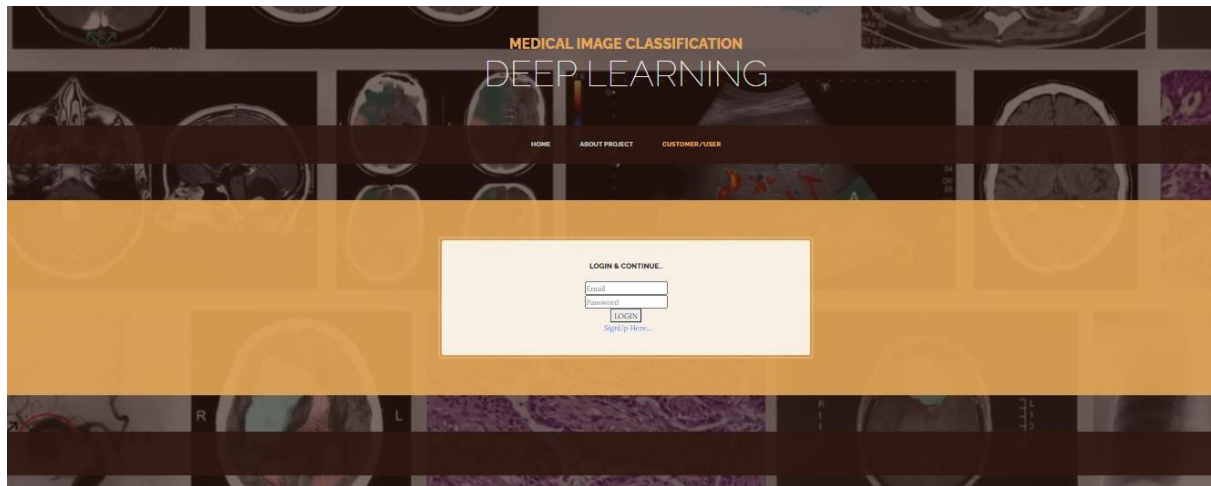
SNAP SHOTS:



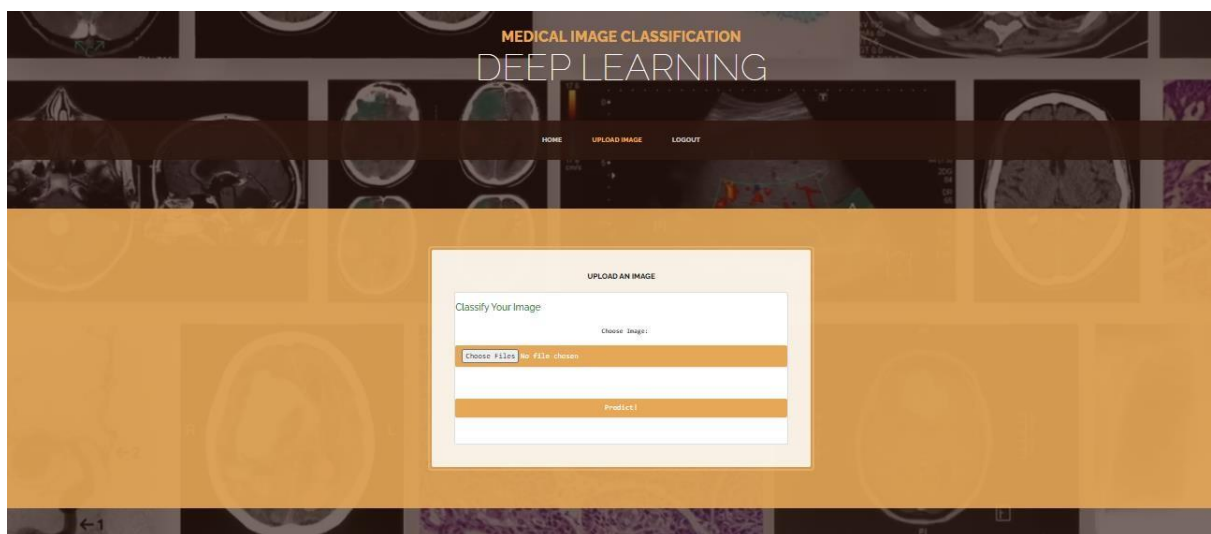
1. Home Page



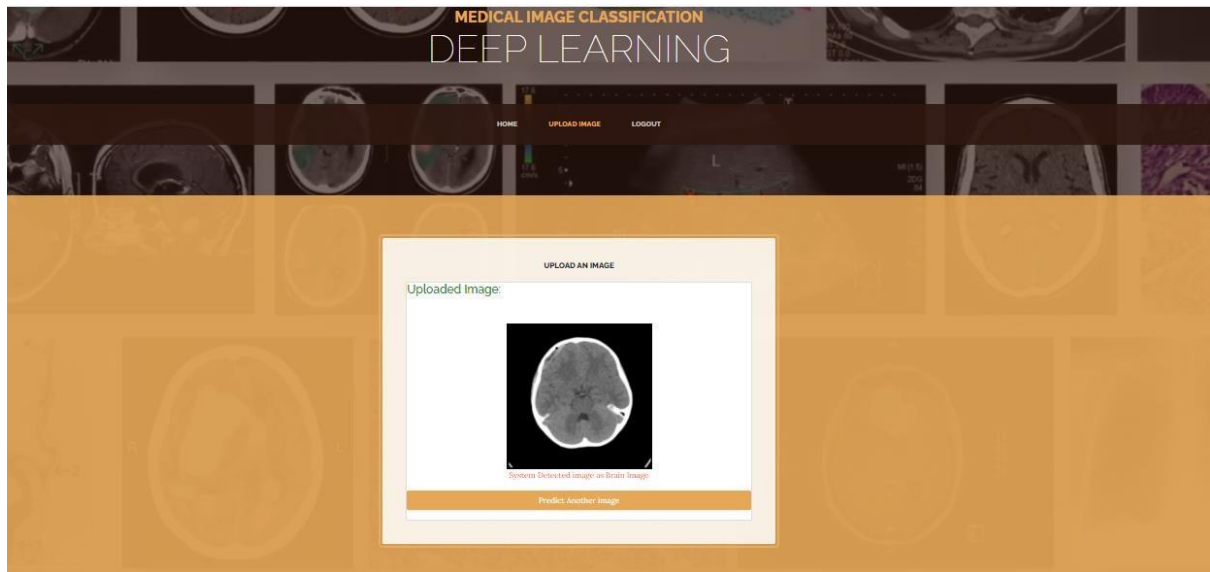
2. About Project



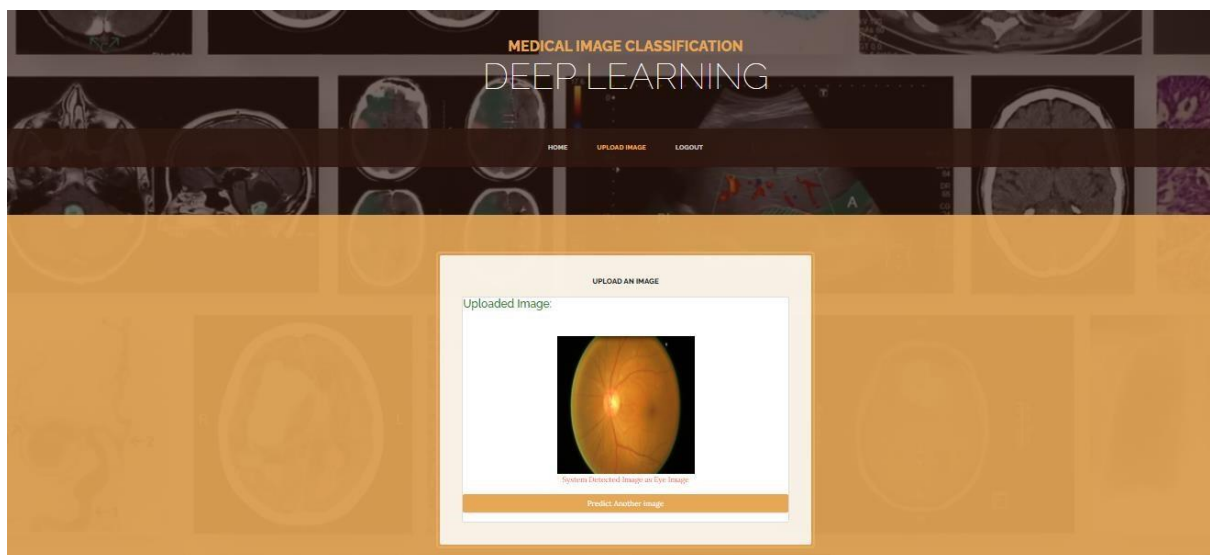
3.Login Page



4.Uploading Image



5. Output image classified as Brain



6. Output image classified as Eye