Project

on

IMAGE PROCESSING TECHNIQUES FOR SKIN CARE DETECTION

Submitted in partial fulfillment of the

requirement for the award of the degree of

Bachelor Of Technology in Computer Science & Engg.

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Dr. Bassety Malligarjuna
Associate Professor

Submitted By

Abhishek Tripathi(18SCSE1010057)
Ankit Sharma(18SCSE1010055)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA

DECEMBER, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"IMAGE PROCESSING TECHNIQUES FOR SKIN CARE DETECTION"** in partial fulfillment of the requirements for the award of the Bachelor of Technology submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of month, Year to Month and Year, under the supervision of Name… Designation, Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

Abhishek Tripathi(18SCSE1010057)
Ankit Sharma(18SCSE1010055)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. Bassety Malligarjuna
Associate Professor

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of Abhishek Tripathi(18SCSE1010057), Ankit Sharma(18SCS1010055) has been held on ________________ and his/her work is recommended for the award of Bachelor of Technology in Computer Science & Engg.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021
Place: Greater Noida

Abstract

Skin diseases are more common than other diseases. Skin diseases may be caused by fungal infection, bacteria, allergy, or viruses, etc. The advancement of lasers and Photonics based medical technology has made it possible to diagnose the skin diseases much more quickly and accurately. But the cost of such diagnosis is still limited and very expensive. So, image processing techniques help to build automated screening system for dermatology at an initial stage. The extraction of features plays a key role in helping to classify skin diseases. Computer vision has a role in the detection of skin diseases in a variety of techniques. Due to deserts and hot weather, skin diseases are common in Saudi Arabia. This work contributes in the research of skin disease detection. We proposed an image processing -based method to detect skin diseases. This method takes the digital image of disease effect skin area, then use image analysis to identify the type of disease. Our proposed approach is simple, fast and does not require expensive equipment other than a camera and a computer. The approach works on the inputs of a color image. Then resize the of the image to extract features using pretrained convolutional neural network. After that classified feature using Multiclass SVM. Finally, the results are shown to the user, including the type of disease, spread, and severity. The system successfully detects 3 different types of skin diseases with an accuracy rate of 100%.

Contents

List of Table

List of Figures

CHAPTER-1 Introduction

Machine learning algorithms have the potential to be invested deeply in all fields of medicine, from drug discovery to clinical decision making, significantly altering the way medicine is practiced. The success of machine learning algorithms at computer vision tasks in recent years comes at an opportune time when medical records are increasingly digitalized. The use of electronic health records (EHR) quadrupled from 11.8% to 39.6% amongst offce-based physicians in the US from 2007 to 2012. Medical images are an integral part of a patient's EHR and are currently analyzed by human radiologists, who are limited by speed, fatigue, and experience. It takes years and great financial cost to train a qualified radiologist, and some health-care systems out source radiology reporting to lower cost countries such as India via tele-radiology. A d5elayed or erroneous diagnosis causes harm to the patient. Therefore, it is ideal for medical image analysis to be carried out by an automated, accurate and effcient machine learning algorithm. Medical image analysis is an active field of research for machine learning, partly because the data is relatively structured and labelled, and it is likely that this will be the area where patients first interact with functioning, practical artificial intelligence systems. This is significant for two reasons. Firstly, in terms of actual patient metrics, medical image analysis is a litmus test as to whether artificial intelligence systems will actually improve patient outcomes and survival. Secondly, it provides a testbed for human-AI interaction, of how receptive patients will be towards health- altering choices being made, or assisted by a non-human actor.

Computer aided detection (CAD) of brain tumor is a preferred tool for non-invasively diagnosing brain tumor. The brain images are obtained using Magnetic Resonance Imaging (MRI), which are prone to noise and artefacts such as labels and intensity variations during acquisition [2]. In addition, there are many structures in the brain image such as cerebrospinal fluid, grey matter, and white matter and skull tissues apart from the tumor. A generic CAD brain tumor detection process follows the following steps: pre-processing the image to remove noise and artefacts, segmenting the pre-processed image to identify possible tumor regions, extracting useful features from the tumor regions and classifying whether or not tumor is present.

Purpose of the project

The tremendous success of machine learning algorithms at image recognition tasks in recent years intersects with a time of dramatically increased use of electronic medical records and diagnostic imaging. This review introduces the machine learning algorithms as applied to medical image analysis, focusing on convolutional neural networks, and emphasizing clinical aspects of the field.   The purpose of this project is to address the automatic brain tumor detection using image processing tools and to reduce the computation time of the steps involved so that a brain MRI image can be identified as malignant or benign in the least computation time possible.

A. Problem statement

Medical images are an integral part of a patient's EHR(electronic health records) and are currently analyzed by human radiologists, who are limited by speed, fatigue, and experience. It takes years and great financial cost to train a qualified radiologist, and some health-care systems out source radiology reporting to lower cost countries such as India via tele-radiology. A delayed or erroneous diagnosis causes harm to the patient. Therefore, it is ideal for medical image analysis to be carried out by an automated, accurate and efficient machine learning algorithm.

B. Solution for the problem statement

Detection, sometimes known as Computer-Aided Detection is a keen area of study as missing a lesion on a scan can have drastic consequences for both the patient and the clinician. This project is contributing in the field of image processing by introducing a model which can automate the diagnosis of tumor more accurately, efficiently and in less time.

Here we will use classification technique with proposing model of Deep Neural Network with grey scaled segmentation technique which will improve the diagnosis process of brain tumor for the effective and timely treatment.

So, one can detect the patient's tumor is cancerous or not just by inputting the MRI image with ease.

C. Introduction of Medical Imaging

Medical imaging is the technique and process of creating visual representations of the interior of a body for clinical analysis and medical intervention, as well as visual representation of the function of some organs or tissues (physiology). Medical imaging seeks to reveal internal structures hidden by the skin and bones, as well as to diagnose and treat disease.

Review of Literature

Several researchers have proposed image processing-based techniques to detect the type of skin diseases. Here we briefly review some of the techniques as reported in the literature.

In [1], a system is proposed for the dissection of skin diseases using color images without the need for doctor intervention. The system consists of two stages, the first the detection of the infected skin by uses color image processing techniques, k -means clustering and color gradient techniques to identify the diseased skin and the second the classification of the disease type using artificial neural networks. The system was tested on six types of skin diseases with average accuracy of first stage 95.99% and the second stage 94.016%.

In the method of [2], extraction of image features is the first step in detection of skin diseases. In this method, the greater number of features extracted from the image, better the accuracy of system.

The author of [2] applied the method to nine types of skin diseases with accuracy up to 90%.

Melanoma is type of skin cancer that can cause death, if not diagnose and treat in the early stages. The author of [3], focused on the study of various segmentation techniques that could be applied to detect melanoma using image processing. Segmentation process is described that falls on the infected spot boundaries to extract more features.

MACHINE LEARNING ARCHITECTURE

Machine learning (**ML**) is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in a wide variety of applications, such as medical image analysis, email filtering, computer vision, etc., where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Basicaly there are two types of machine learning models-Supervised and Unsupervised.

Supervised Learning Model

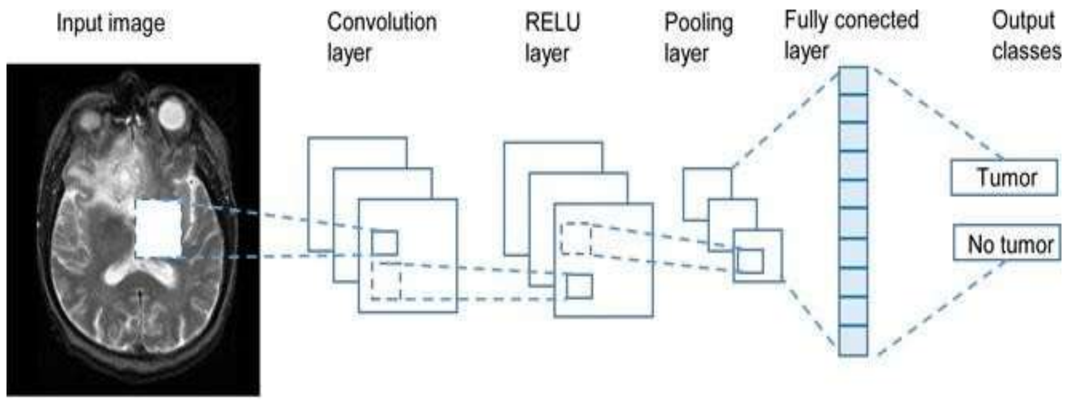
Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and a desired output, also known as a supervisory signal. In the case of semi-supervised learning algorithms, some of the training examples are missing the desired output. In the mathematical model, each training example is represented by an array or vector, and the training data by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.

Classification:

The Classification comes under the supervised machine learning. A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”. A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes. For example, when filtering emails “spam” or “not spam”, when looking at transaction data, “fraudulent”, or “authorized”. In short Classification either predicts categorical class labels or classifies data (construct a model) based on the training set and the values (class labels) in classifying attributes and uses it in classifying new data. There are a number of classification models. Classification models include logistic regression, decision tree, random forest, gradient-boosted tree, multi layer perceptron, one-vsrest, and Naive Bayes.

Convolutional Neural Networks

CNNs are the most researched machine learning algorithms in medical image analysis . The reason for thesis that CNNs preserve spatial relationships when filtering input images. As mentioned, spatial relationships are of cru-cial importance in radiology, for example, in how the edge of a bone joins with muscle, or where normal lung tissue interfaces with cancerous tissue. A CNN takes an input image of raw pixels, and transforms it via Convolutional Layers, Rectified Linear Unit (ReLU) Layer sand Pooling Layers. This feeds into a final Fully Connected Layer which assigns class scores or probabilities, thus classifying the input into the class with the highest probability.



Convolution:

A convolution is defined as an operation on two functions. In image analysis, one function consists of input values(e.g. pixel values) at a position in the image, and the second function is a filter (or kernel); each can be represented as array of numbers. Computing the dot product between the two functions gives an output. The filter is then shifted to the next position in the image as defined by the stride length .The computation is repeated until the entire image is covered,producing a feature (or activation) map. This is a map of where the filter is strongly activated and 'sees' a feature such as a straight line, a dot, or a curved edge. If a photograph of a face was fed into a CNN, initially low-level features such as lines and edges are discovered by the filters. These build up to progressively higher features in subsequent layers, such as nose, eye or ear, as the feature maps become inputs for the next layer in the CNN architecture.
Convolution exploits three ideas intrinsic to perform com- computationally efficient machine learning: sparse connect-tions, parameter sharing (or weights sharing) and equivariant(or invariant) representation. Unlike some neural net-works where every input neuron is connected to every output neuron in the subsequent layer, CNN neurons have sparse connections, meaning that only some inputs are connected to the next layer. By having a small, local receptive field(i.e., the area covered by the filter per stride), meaningful features can be gradually learnt,

and the number of weights to be calculated can be drastically reduced, increasing the algorithm's efficiency. In using each filter with its fixed weights across different positions of the entire image, CNNsreduce memory storage requirements. This is known as parameter sharing. This is in contrast to a fully connected neural network where the weights between layers are more numerous, used once and then discarded. Parameter sharing results in the quality of equivariant representation to arise.This means that input translations result in a corresponding feature map translation. The convolution operation is defined by the*symbol. An output (or feature map)s(t) is defined below when input I(t) is convolved with a filter or kernal.

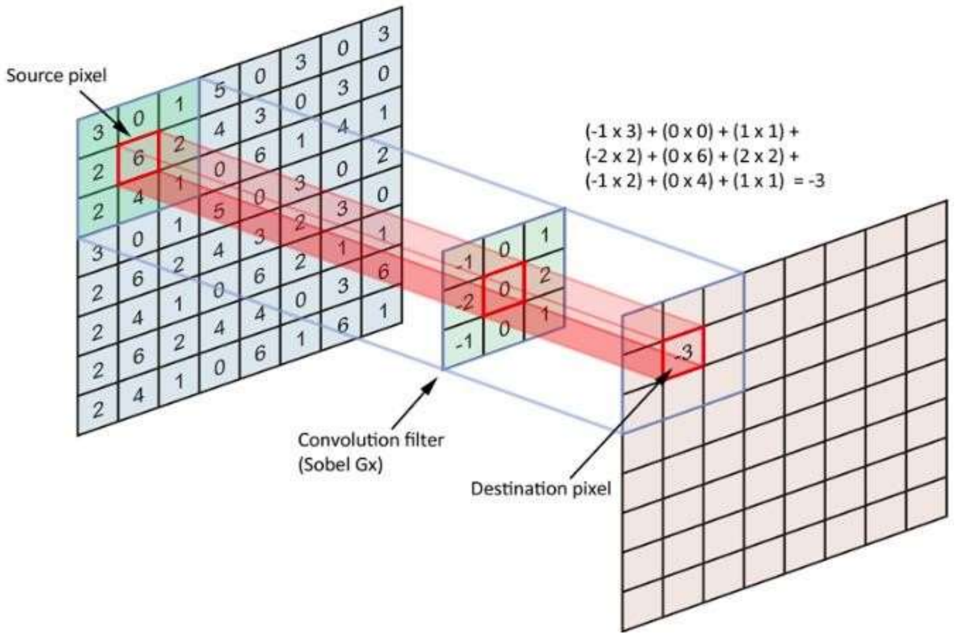

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Fig: Convolutional Layer

Rectified linear unit(ReLU) layer:

The ReLU layer is an activation function that sets negative input values to zero. This simplifies and accelerates calculations and training, and helps to avoid the vanishing gradient problem. Mathematically it is defined as:f(x)=max(0,x).(6)where xis the input to the neuron. Other activation functions include the sigmoid, tanh, leaky RELUs, Randomized RELUs and parametric RELUs

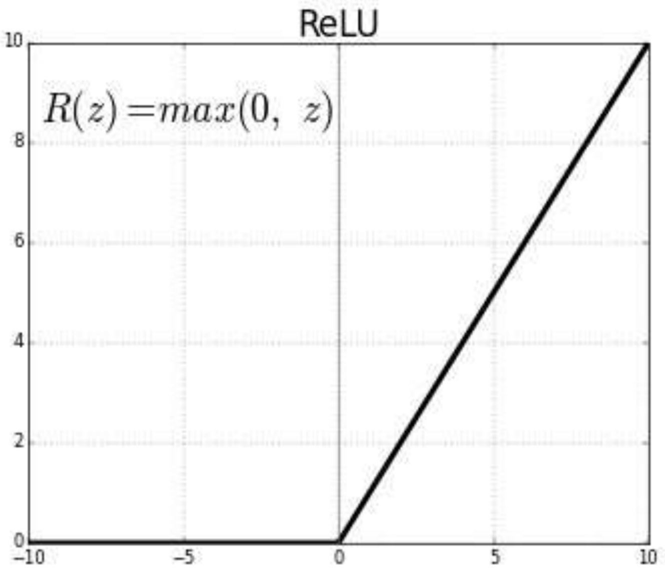


Fig: ReLU Layer

Pooling layer:
The Pooling layer is inserted between the Convolution and RELU layers to reduce the number of parameters to be calculated, as well as the size of the image (width and height, but not depth).

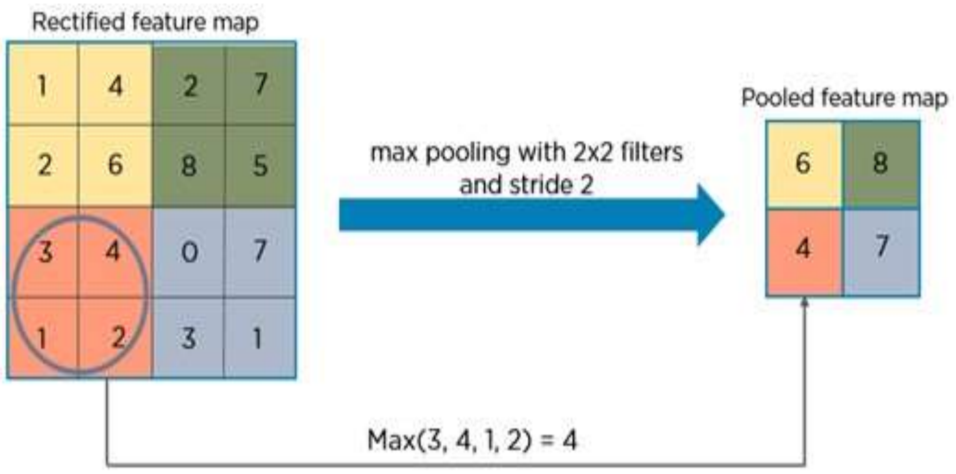


Fig: Pooling Layer

Max-pooling is most commonly used; other pooling layers include Average pooling and L2-normalization pooling. Max-pooling simply takes the largest input value within a filter and discards the other values; effectively it summarizes the strongest activations over a neighborhood. The rationale is that the relative location of a strongly activated feature to another is more important than its exact location
After getting max-pooled matrix, the task is to convert pooled matrix to a single 1D array. This process is called Flattening.

Fully Connected Layer:

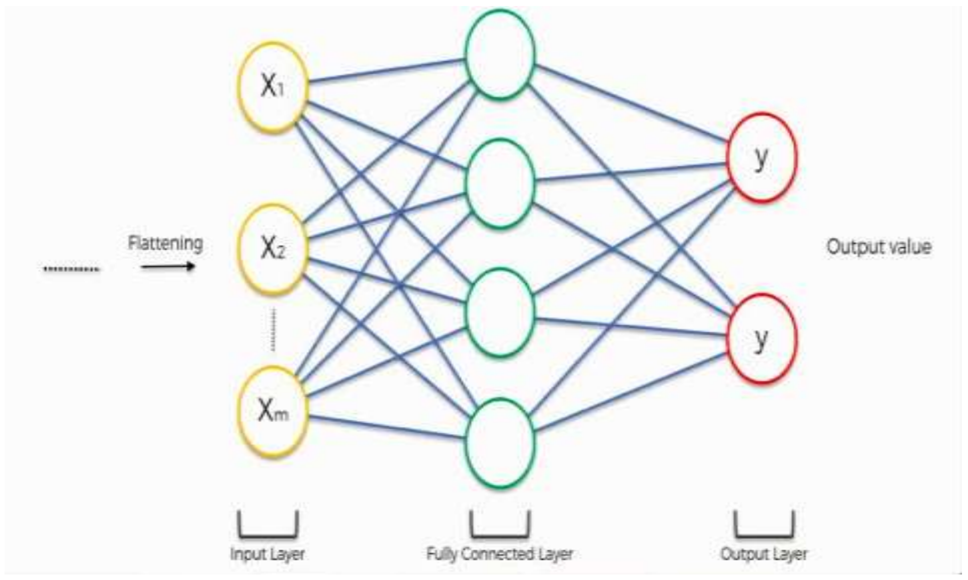


Fig: Fully Connected Layer

The final layer in a CNN is the Fully Connected Layer, meaning that every neuron in the preceding layer is connected to every neuron in the Fully Connected Layer. Like the convolution, ReLU and pooling layers, there can be1 or more fully connected layers depending on the level of feature abstraction desired. This layer takes the output from the preceding layer (Convolutional, ReLU or Pooling) as its input, and computes a probability score for classification into the different available classes. In essence, this layer looks at the combination of the most strongly activated features that would indicate the image belongs to a particular class. For example, on histology glass slides, cancer cells have a high DNA to cytoplasm ratio compared to normal cells. f features of DNA were strongly detected from the preceding layer, the CNN would be more likely to predict the presence of cancer cells. Standard neural network training methods with back propagation [10] and stochastic gradient descent help the CNN learn important associations from training image

Regression

In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors'). More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed. Regression also comes under supervised machine learning .

Unsupervised Learning models

Unsupervised learning is a term used for Hebbian learning, associated to learning without a teacher, also known as self-organization and a method of modeling the probability density of inputs,e.g. K-Means clustering. The cluster analysis is a branch of machine learning that groups the data that has not been labelled, classified or categorized. Instead of responding to feedback, cluster analysis identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data.

A central application of unsupervised learning is in the field of density estimation in statistics, though unsupervised learning encompasses many other domains involving summarizing and explaining data features. It could be contrasted with supervised learning by saying that whereas supervised learning intends to infer a conditional probability distribution conditioned on the label of input data; unsupervised learning intends to infer an a priori probability distribution .

SYSTEM ANALYSIS

STUDY OF THE SYSTEM

1. NumPy
2. Pandas
3. Matplotlib
4. SciKit-Learn
5. TensorFlow
6. Keras

1. NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and FORTRAN code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

2. Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

3. Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hard copy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

4. SciKit – Learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D/3D plotting
- **IPython**: Enhanced interactive console
- **Sympy**: Symbolic mathematics
- **Pandas**: Data structures and analysis
- Extensions or modules for SciPy care conventionally named SciKits. As such, the module provides learning algorithms and is named scikit-learn.

5.Tensorflow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production. TensorFlow was developed by the Google Brain team for internal Google use.

TensorFlow architecture works in three parts:

- Preprocessing the data
- Build the model
- Train and estimate the model

It is called TensorFlow because it takes input as a multi-dimensional array, also known as **tensors**. We can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.
This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

6.Keras

Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow.

It was developed to make implementing deep learning models as fast and easy as possible for research and development.

It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license.

- **Modularity**: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
- **Minimalism**: The library provides just enough to achieve an outcome, no frills and maximizing readability.
- **Extensibility**: New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
- **Python**: No separate model files with custom file formats. Everything is native Python.

We can summarize the construction of deep learning models in Keras as follows:

1. **Define your model**. Create a sequence and add layers.
2. **Compile your model**. Specify loss functions and optimizers.
3. **Fit your model**. Execute the model using data.
4. **Make predictions**. Use the model to generate predictions on new data.

INPUTS AND OUTPUTS

The following some are the projects inputs and outputs.

Inputs:

- Importing the all required packages like Tensorflow, keras ,NumPy, Pandas, Matplotlib, scikit – learn and required machine learning algorithms packages .
- Setting the dimensions of visualization graph.
- Downloading and importing the dataset and convert to data frame (In case of images,we need to convert it into numeric values since ML algorithm work on only numerical data).

Outputs:

- preprocessing the importing data frame for imputing nulls with the related information.
- All are displaying cleaned outputs.
- After applying machine learning algorithms, it will give good results and visualization plots.

Chapter 3 REQUIREMENTS

Hardware Requirements:

RAM: 4GB and Higher
Processor: Intel i3 and above
Hard Disk: 500GB: Minimum

Software Requirements:

OS: Windows or Linux
Python IDE : python 2.7.x and above
Jupyter IDE
Anaconda 3.x

Setup tools and pip to be installed for 3.6 and

above Python Scripting

DIAGRAMS

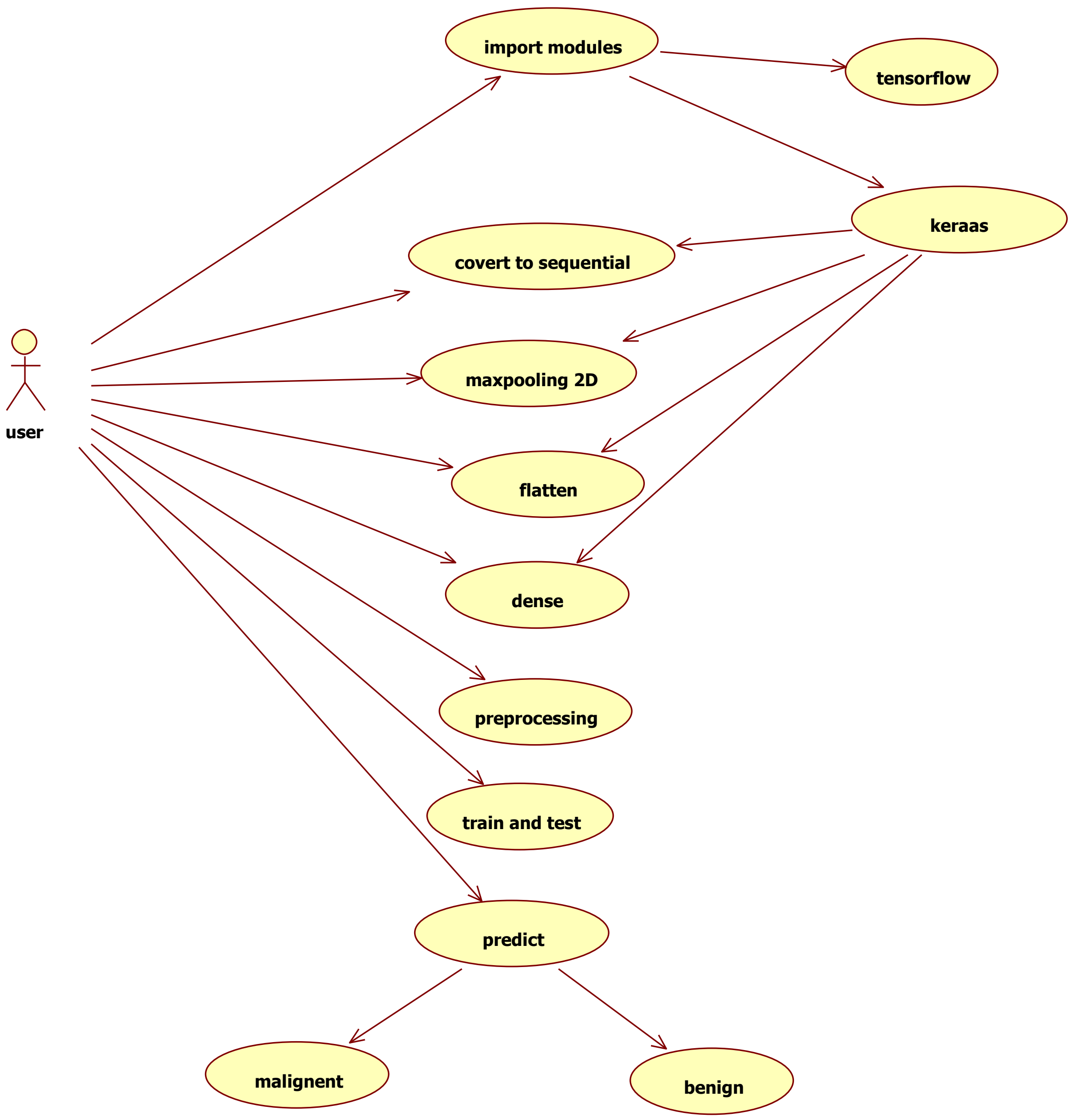
. UML DIAGRAMS

The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as:

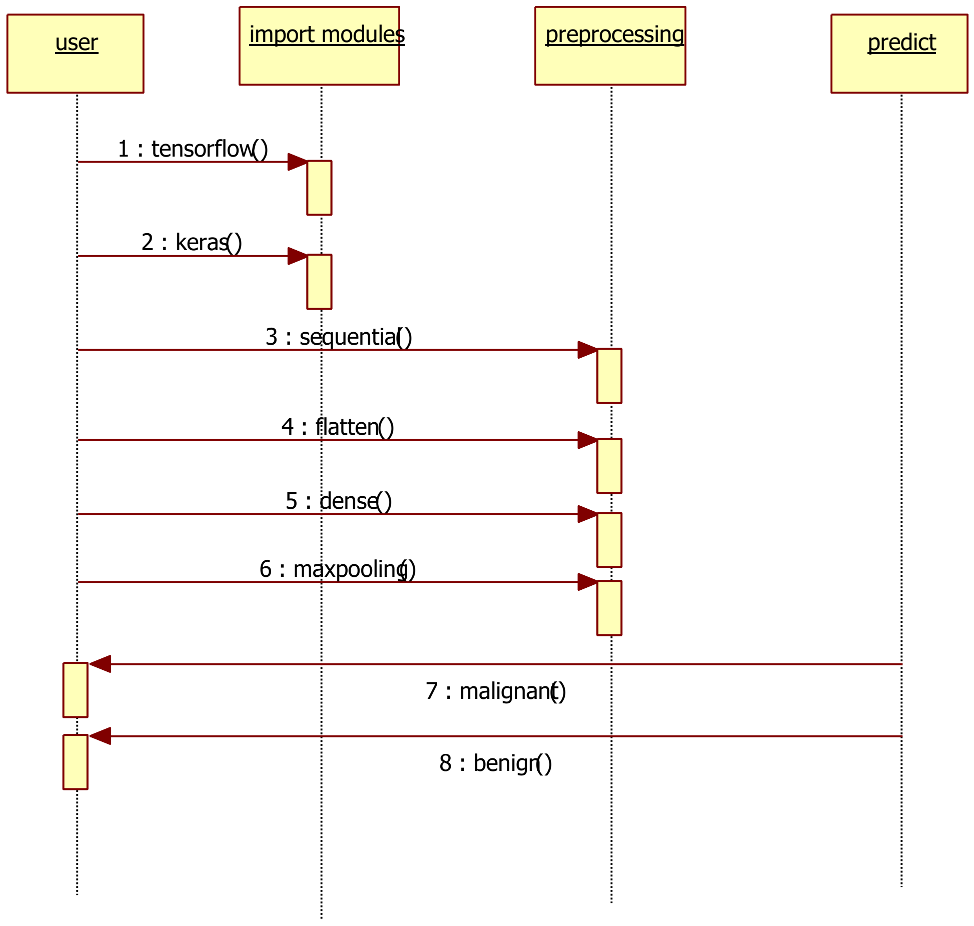
- actors
- business processes
- (logical) components
- activities
- programming language statements
- database schemas, and
- Reusable software components.

UML combines best techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems

Use Case Diagram



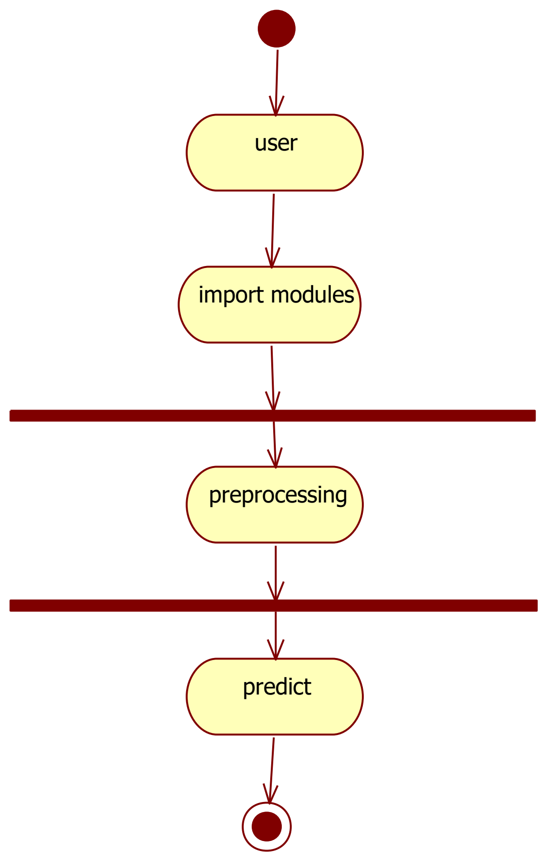
Sequence diagram



Sequence Diagrams Represent the objects participating the interaction horizontally and time vertically. A Use Case is a kind of behavioral classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the subject without reference to its internal structure. These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional behavior and error handling.

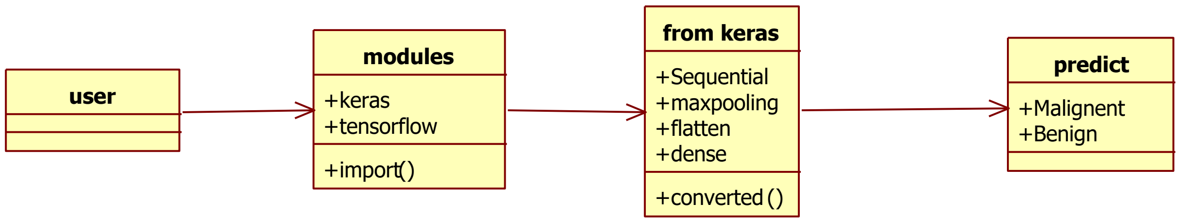
..

Activity diagram



Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

Class Diagram:



The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

METHODOLOGY OF ANALYSIS AND IMPLEMENTATION

Operations and Code details:

Here we are solving an image classification problem, where our goal will be to tell which class the input image belongs to. The way we are going to achieve it is by training an artificial neural network on few brain scan images of benign and malignent type and make the NN(Neural Network) learn to predict which class the image belongs to, next time it sees an image having a malignent or benign i.e. tumor or not.

So coming to the coding part, we are going to use **Keras** deep learning library in python to build our CNN(Convolutional Neural Network).

Here we will need to download two folders and both the folders named " test set " and "training_set" into our working directory which is the training data as well as the test dataset.

First, the folder "**training_set**" contains two sub folders **benign** and **malignent**, each holding 26 images of the respective category. Second, the folder "**test_set**" contains two sub folders **benign and malignent**, each holding 9 images of respective category.

The process of building a Convolutional Neural Network always involves four major steps.

Step - 1 : Convolution

Step - 2 : Pooling

Step - 3 : Flattening

Step - 4 : Full connection

We will be going through each of the above operations while coding our neural network.

First let us import all the required keras packages using which we are going to build our CNN, make sure that every package is installed properly in our machine, there is two ways os using keras, i.e Using Tensorflow backend and by Using Theano backend, all the code remains the same in either cases. I tested the below code using Tensorflow backend.

```
# Importing the Keras libraries and

packages from keras.models import

Sequential from keras.layers import Conv2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense
```

Let us now see what each of the above packages are imported for :

In **line 1**, we've imported Sequential from keras.models, to initialise our neural network model as a sequential network. There are two basic ways of initialising a neural network, either by a sequence of layers or as a graph.

In **line 2**, we've imported Conv2D from keras.layers, this is to perform the convolution operation i.e the first step of a CNN, on the training images. Since we are working on images here, which a basically 2 Dimensional arrays, we're using Convolution 2-D, you may have to use Convolution 3-D while dealing with videos, where the third dimension will be time.

In **line 3**, we've imported MaxPooling2D from keras.layers, which is used for pooling operation, that is the step—2 in the process of building a cnn. For building this particular neural network, we are using a Maxpooling function, there exist different types of pooling operations like Min Pooling, Mean Pooling, etc. Here in MaxPooling we need the maximum value pixel from the respective region of interest.

In **line 4**, we've imported Flatten from keras.layers, which is used for Flattening. Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector.

And finally in **line 5**, we've imported Dense from keras.layers, which is used to perform the full connection of the neural network, which is the step 4 in the process of building a CNN.

Now, we will create an object of the sequential class below:

classifier = Sequential()

Let us now code the Convolution step, actually it is too easy to implement these complex operations in a single line of code in python, because of Keras.

classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

Let's break down the above code function by function. We took the object which already has an idea of how our neural network is going to be(Sequential), then we added a convolution layer by using the "Conv2D" function. The Conv2D function is taking 4 arguments, the first is the number of filters i.e 32 here, the second argument is the shape each filter is going to be i.e 3x3 here, the third is the input shape and the type of image(RGB or Black and White)of each image i.e the input image our CNN is going to be taking is of a 64x64 resolution and "3" stands for RGB, which is a colour img, the fourth argument is the activation function we want to use, here 'relu' stands for a rectifier function.

Now, we need to perform pooling operation on the resultant feature maps we get after the convolution operation is done on an image. The primary aim of a pooling operation is to reduce the size of the images as much as possible. But the key thing to understand here is that we are trying to reduce the total number of nodes for the upcoming layers.

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

We start by taking our classifier object and add the pooling layer. We take a 2x2 matrix we'll have minimum pixel loss and get a precise region where the feature are located. We just reduced the complexity of the model without reducing it's performance.

It's time for us to now convert all the pooled images into a continuous vector through Flattening. Flattening is a very important step to understand. What we are basically doing here is taking the 2-D array, i.e pooled image pixels and converting them to a one dimensional single vector. **classifier.add(Flatten())**

The above code is pretty self-explanatory. We've used flatten function to perform flattening, we no need to add any special parameters, keras will understand that the "classifier" object is already holding pooled image pixels and they need to be flattened.

In this step we need to create a fully connected layer, and to this layer we are going to connect the set of nodes we got after the flattening step, these nodes will act as an input layer to these fullyconnected layers. As this layer will be present between the input layer and output layer, we can refer to it a hidden layer. **classifier.add(Dense(units = 128, activation = 'relu'))**

As we can see, Dense is the function to add a fully connected layer, 'units' is where we define the number of nodes that should be present in this hidden layer, these units value will be always between the number of input nodes and the output nodes but the art of choosing the most optimal number of nodes can be achieved only through experimental tries. Though it's a common practice to use a power of 2. And the activation function will be a rectifier function.

Now it's time to initialise our output layer, which should contain only one node, as it is binary classification. This single node will give us a binary output of either a benign or malignent. **classifier.add(Dense(units = 1, activation = 'sigmoid'))**

We can observe that the final layer contains only one node, and we will be using a sigmoid activation function for the final layer.

Now that we have completed building our CNN model, it's time to compile it.

classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

From above :

- Optimizer parameter is to choose the stochastic gradient descent algorithm.
- Loss parameter is to choose the loss function.
- Finally, the metrics parameter is to choose the performance metric.

It's time to fit our CNN to the image dataset that you've downloaded. But before we do that, we are going to pre-process the images to prevent over-fitting. Overfitting is when you get a great training accuracy and very poor test accuracy due to overfitting of nodes from one layer to another.

So before we fit our images to the neural network, we need to perform some image augmentations on them, which is basically synthesising the training data. We are going to do this using **keras.preprocessing** library for doing the synthesising part as well as to prepare the training set as well as the test test set of images that are present in a properly structured directories, where the directory's name is take as the label of all the images present in it. For example : All the images inside the 'benign' named folder will be considered as benign by keras.

```
train_datagen = ImageDataGenerator(rescale = 1./255,
shear_range = 0.2, zoom_range = 0.2, horizontal_flip =
True)
test_datagen = ImageDataGenerator(rescale = 1./255)
training_set =
train_datagen.flow_from_directory('training_set', target_size =
(64, 64), batch_size = 32, class_mode = 'binary')
test_set = test_datagen.flow_from_directory('test_set',
target_size = (64, 64), batch_size = 32, class_mode =
'binary')
```

We can find the explanation of what each of the above parameters do**,**in the keras documentation page. As a whole of whats happening above is that we are creating synthetic data out of the same images by performing different type of operations on these images like flipping, rotating, blurring, etc.

Now fit the data to our model !

```
classifier.fit_generator(training_set,
steps_per_epoch = 8000,
epochs = 25,
validation_data = test_set,
validation_steps = 2000)
```

In the above code, '*steps_per_epoch*' holds the number of training images, i.e the number of images the training_set folder contains.

And '*epochs*', a single epoch is a single step in training a neural network; in other words when a neural network is trained on every training samples only in one pass we say that one epoch is finished. So training process should consist more than one epochs. In this case we have defined 25 epochs.

Making new predictions from our trained model :

```
import numpy as np
from keras.preprocessing import image
test_image     =
         image.load_img('dataset/single_prediction/benign_or_malignent_1.jpg', target_size
= (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
prediction = 'Benign'
```

else: prediction =
'Malignent'

The test_image holds the image that needs to be tested on the CNN. Once we have the test image, we will prepare the image to be sent into the model by converting its resolution to 64x64 as the model only excepts that resolution. Then we are using ‘predict()’ method on our classifier object to get the prediction. As the prediction will be in a binary form, we will be receiving either a 1 or 0, which will represent a ‘benign’ or ‘Malignent’ respectively.

Medical Image Analysis with CNN:-

<u>The complete code with outputs :</u>

```
import tensorflow as tf import keras
from keras.models import Sequential
from keras.layers import Conv2D from
keras.layers import MaxPooling2D from
keras.layers import Flatten from
keras.layers import Dense

classifier = Sequential()

classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

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

classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

classifier.add(Flatten())

classifier.add(Dense(activation = 'relu',units=128))
classifier.add(Dense(activation = 'sigmoid',units=1))

classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

classifier.summary()
```

o/p:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
```

```
conv2d_2 (Conv2D)           (None, 29, 29, 32)       9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)       0
_________________________________________________________________
flatten_1 (Flatten)         (None, 6272)            0
_________________________________________________________________
dense_1 (Dense)             (None, 128)             802944
_________________________________________________________________
dense_2 (Dense)             (None, 1)               129
=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.preprocessing.image import ImageDataGenerator

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

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set= train_datagen.flow_from_directory('/home/avinash/Deep Learning Applications in/code/Brain_tumor/train/',
                                                 target_size = (64, 64),
batch_size = 32,
class_mode = 'binary')

test_set = test_datagen.flow_from_directory('/home/avinash/Deep                    Learning Applications in/code/Brain_tumor/test/',
                                    target_size=(64, 64),
                          batch_size = 32,
                class_mode = 'binary')
```

```
Found 44 images belonging to 2 classes.
Found 14 images belonging to 2 classes.
```

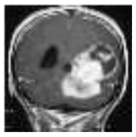
```
classifier.fit_generator(training_set,   steps_per_epoch=None,        epochs=100,    verbose=1,
callbacks=None,         validation_data=test_set,        validation_steps=None,
          class_weight=None, max_queue_size=10, workers=1, use_multiprocessing=False,
shuffle=True, initial_epoch=0)
```

o/p:

```
Epoch 1/100
```

```
2/2 [==============================] - 0s 147ms/step - loss: 0.0388 - acc: 1.0000 -
val_loss: 3.0713 - val_acc: 0.4286
Epoch 2/100
2/2 [==============================] - 0s 151ms/step - loss: 0.0266 - acc: 1.0000 -
val_loss: 3.1552 - val_acc: 0.4286
.
.
.
.
Epoch 92/100
2/2 [==============================] - 0s 111ms/step - loss: 0.0207 - acc: 1.0000 -
val_loss: 3.9695 - val_acc: 0.4286
Epoch 93/100
2/2 [==============================] - 0s 153ms/step - loss: 0.0996 - acc: 0.9287 -
val_loss: 4.0976 - val_acc: 0.4286
Epoch 94/100
2/2 [==============================] - 0s 123ms/step - loss: 0.0684 - acc: 0.9820 -
val_loss: 3.6747 - val_acc: 0.4286
Epoch 95/100
2/2 [==============================] - 0s 135ms/step - loss: 0.0292 - acc: 1.0000 -
val_loss: 3.1225 - val_acc: 0.5714
Epoch 96/100
2/2 [==============================] - 0s 149ms/step - loss: 0.0479 - acc: 0.9820 -
val_loss: 2.8529 - val_acc: 0.5714
Epoch 97/100
2/2 [==============================] - 0s 121ms/step - loss: 0.0752 - acc: 0.9640 -
val_loss: 2.8919 - val_acc: 0.5714
Epoch 98/100
2/2 [==============================] - 0s 142ms/step - loss: 0.0264 - acc: 0.9820 -
val_loss: 3.1539 - val_acc: 0.5714
Epoch 99/100
2/2 [==============================] - 0s 111ms/step - loss: 0.1200 - acc: 0.9646 -
val_loss: 3.7281 - val_acc: 0.5714
Epoch 100/100
2/2 [==============================] - 0s 140ms/step - loss: 0.1041 - acc: 0.9467 -
val_loss: 3.9787 - val_acc: 0.5714

<keras.callbacks.History at 0x7f3c680e7240>
```

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('/home/avinash/Deep Learning Applications in/mri-brain-image_
new/brain-tumors-fig2_large.jpg', target_size = (64, 64)) test_image
```



test_image = image.img_to_array(test_image)

test_image = np.expand_dims(test_image, axis =

0) test_image **o/p:**

```
array([[[[57., 57., 57.],
[11., 11., 11.],
         [11., 11., 11.],
         ...,
         [11., 11., 11.],
         [11., 11., 11.],
         [10., 10., 10.]],

        [[14., 14., 14.],
         [14., 14., 14.],
         [11., 11., 11.],
         ...,
         [10., 10., 10.],
         [10., 10., 10.],
         [ 9.,  9.,  9.]],

        [[12., 12., 12.],
         [10., 10., 10.],
         [11., 11., 11.],
         ...,
         [11., 11., 11.],
         [11., 11., 11.],
         [11., 11., 11.]],

        ...,

        [[63., 63., 63.],
         [60., 60., 60.],
         [93., 93., 93.],
         ...,
         [11., 11., 11.],
         [11., 11., 11.],
         [11., 11., 11.]],

        [[23., 23., 23.],
         [ 5.,  5.,  5.],
         [ 8.,  8.,  8.],
         ...,
         [11., 11., 11.],
         [11., 11., 11.],
         [11., 11., 11.]],
        [[13., 13., 13.],
         [13., 13., 13.],
         [15., 15., 15.],
         ...,
         [ 9.,  9.,  9.],
         [ 9.,  9.,  9.],
         [ 9.,  9.,  9.]]]], dtype=float32)
```

result =

classifier.predict(test_image) result

o/p:array([[1.]], dtype=float32)

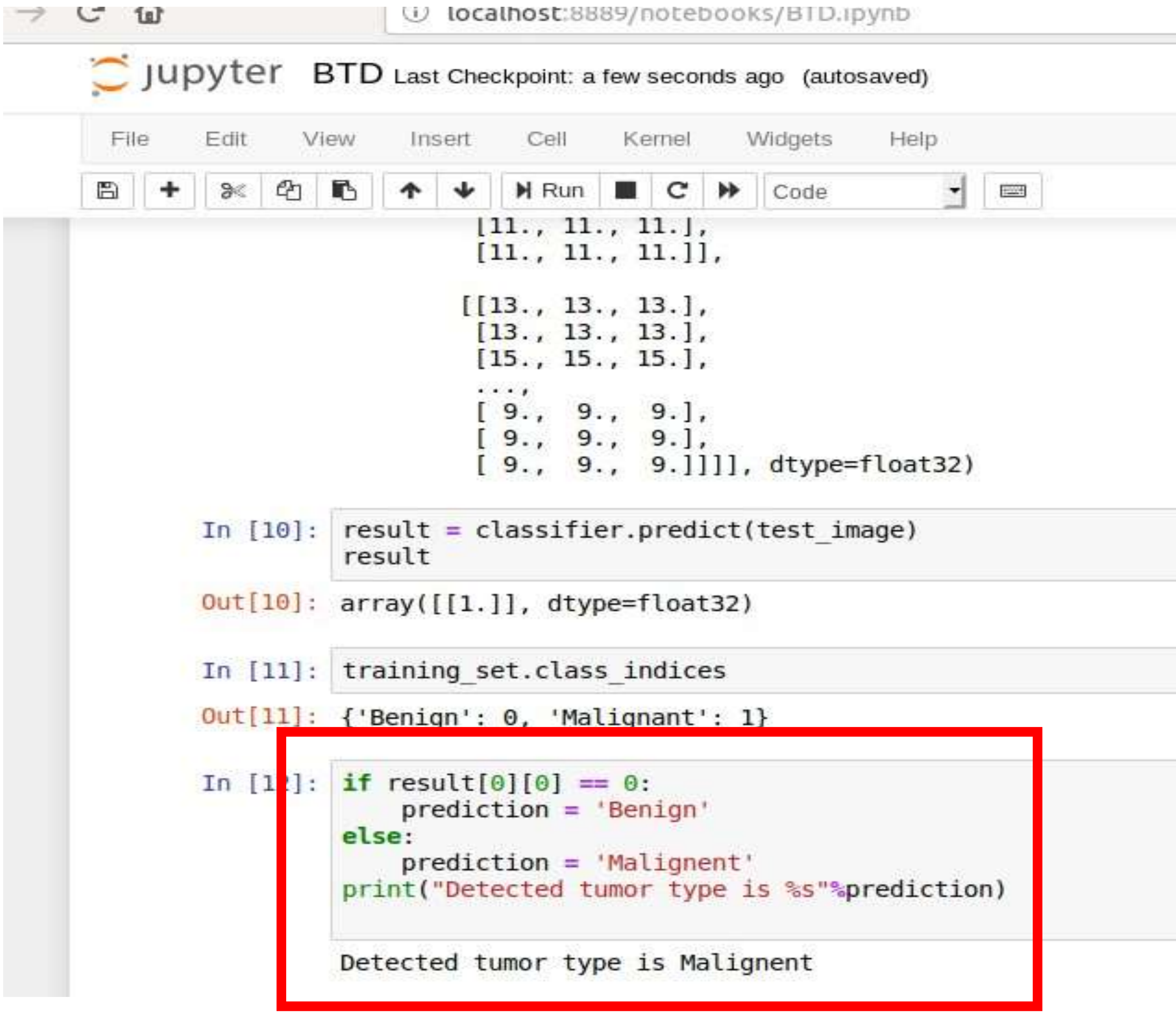
training_set.class_indices

o/p:{'Benign': 0, 'Malignant': 1}

if result[0][0] == 0:   prediction = 'Benign'
else: prediction = 'Malignent'
print("Detected tumor type
is %s"%prediction) **o/p:** Detected tumor type
is Malignent

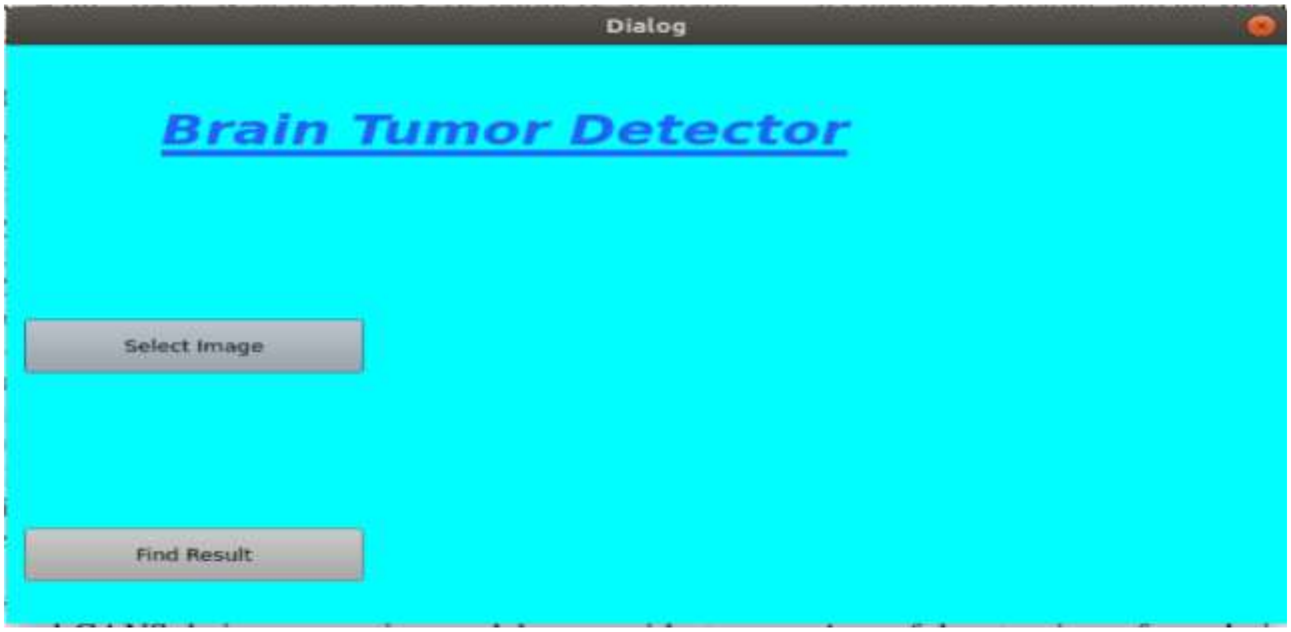
OUTPUT SCREENSHOTS

a.) In Jupyter Notebook


localhost:8889/notebooks/BTD.ipynb
jupyter BTD Last Checkpoint: a few seconds ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
        [11., 11., 11.],
        [11., 11., 11.]],

       [[13., 13., 13.],
        [13., 13., 13.],
        [15., 15., 15.],
        ...,
        [ 9.,  9.,  9.],
        [ 9.,  9.,  9.],
        [ 9.,  9.,  9.]]]], dtype=float32)

In [10]: result = classifier.predict(test_image)
         result

Out[10]: array([[1.]], dtype=float32)

In [11]: training_set.class_indices

Out[11]: {'Benign': 0, 'Malignant': 1}

In [12]: if result[0][0] == 0:
             prediction = 'Benign'
         else:
             prediction = 'Malignent'
         print("Detected tumor type is %s"%prediction)

         Detected tumor type is Malignent
```

b.) In user Interface



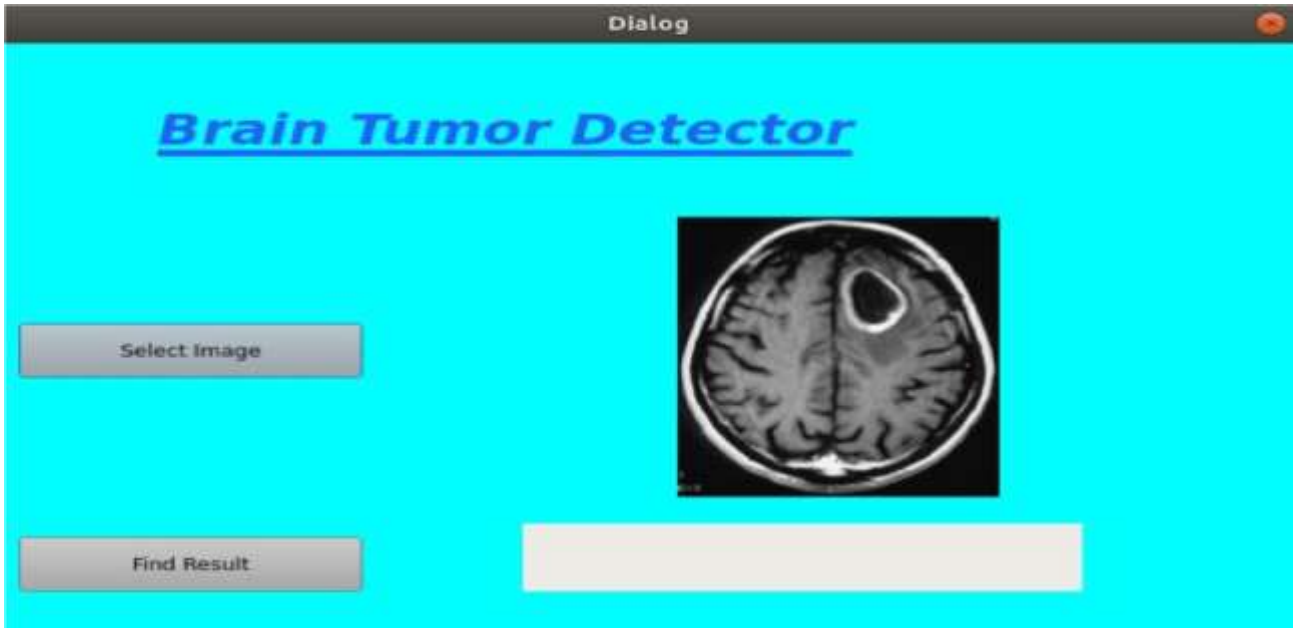


fig2: after selecting the test image

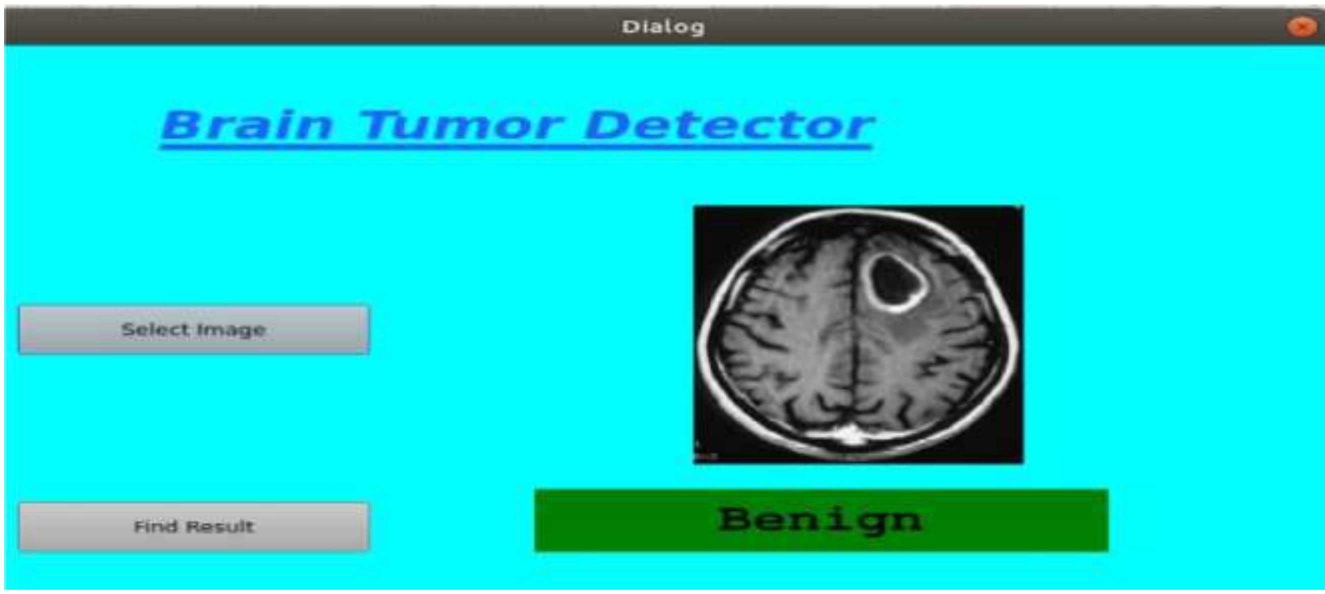
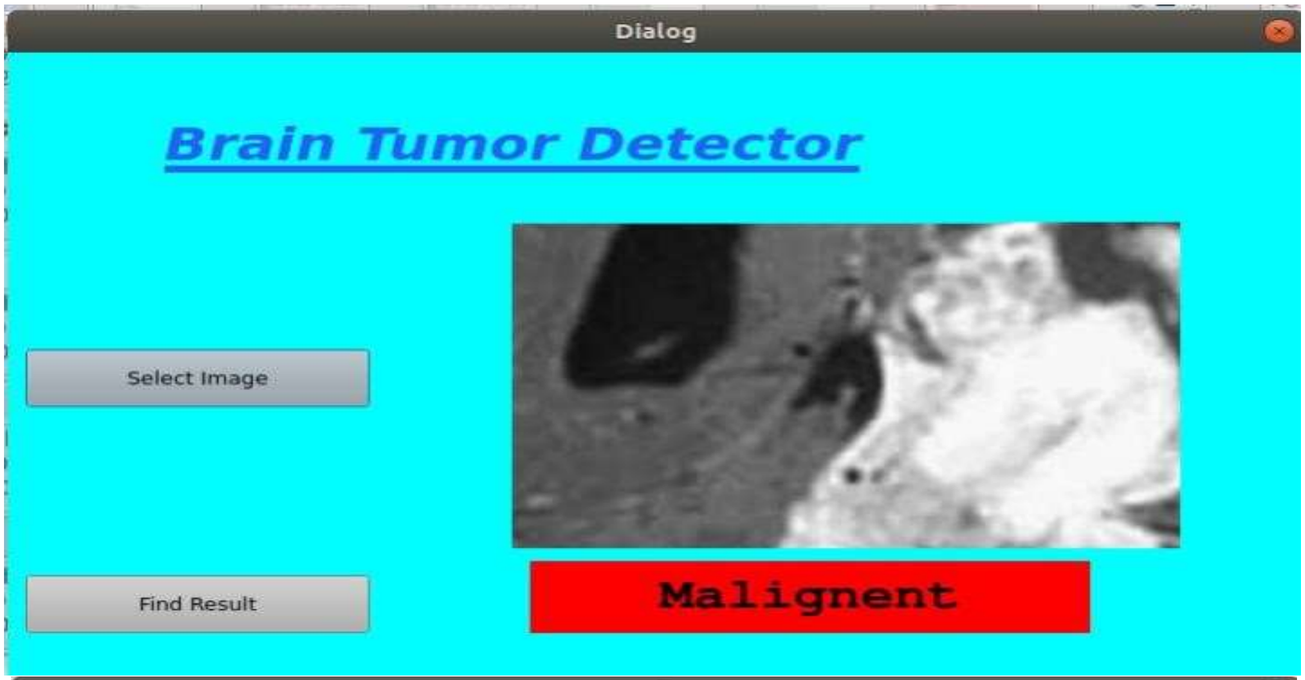
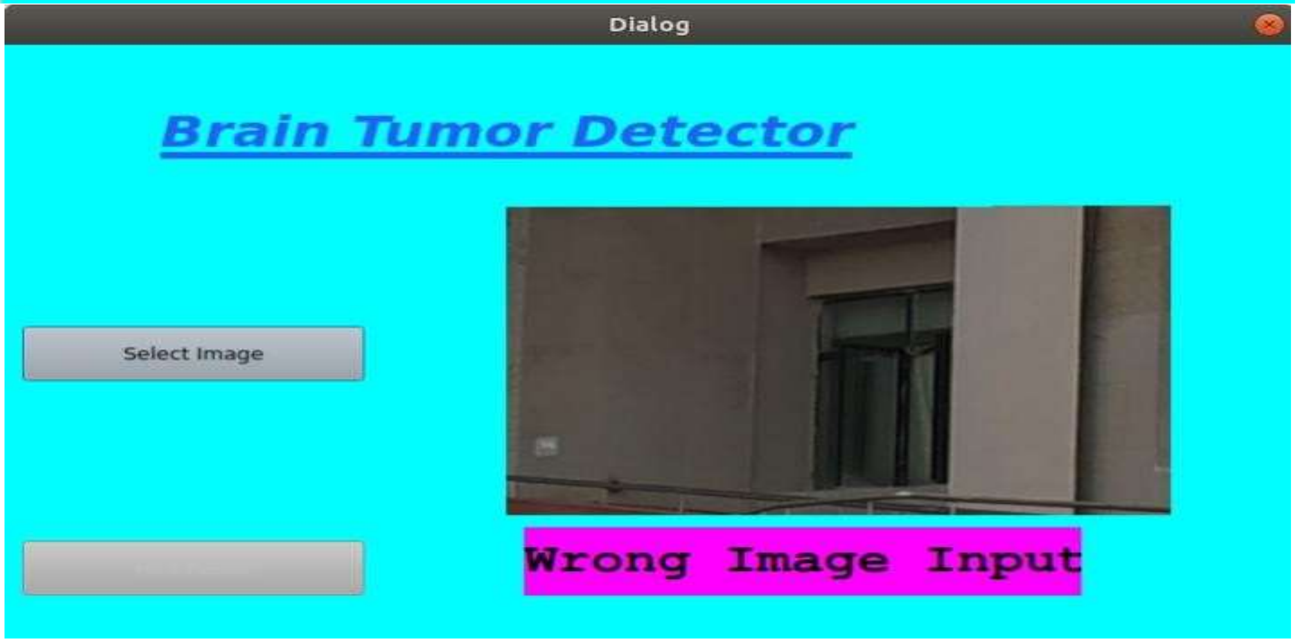


fig3: Output for Benign(Non- Cancerous)





➔ The above screenshots has come after the deployment of Machine Learning model in stand alone user interface for naive user who don’t have the technical knowledge. Model deployment of machine learning is a very tough task.

CONCLUSION

Challenges:-

A recurring theme in machine learning is the limit imposed by the lack of labelled datasets, which hampers training and task performance. Conversely, it is acknowledged that more data improves performance, as Sun et al. shows using an internal Google dataset of 300 million images. In general computer vision tasks, attempts have been made to circumvent limited data by using smaller filters on deeper layers ,with novel CNN architecture combinations , or hyperparameter optimization . In medical image analysis, the lack of data is two-fold and more acute: there is general lack of publicly available data, and high quality labelled data is even more scarce. Most of the datasets presented in this review involve fewer than 100 patients. Yet the situation may not be as dire as it seems, as despite the small training datasets, the papers in this review report relatively satisfactory performance in the various tasks. The question of how many images are necessary for training in medical image analysis was partially answered by Cho et al. . He ascertained the accuracy of a CNN with GoogLeNet architecture in classifying individual axial CT images into one of 6 body regions: brain, neck, shoulder, chest, abdomen, pelvis. With 200 training images, accuracies of 88-98% were achieved on a test set of 6000 images. While categorization into various body regions is not a realistic medical image analysis task, his report does suggest that the problem may be surmountable. Being able to accomplish classification with a small dataset is possibly due to the general intrinsic image homogeneity across different patients, as opposed to the near-infinite variety of natural images, such as a dog in various breeds, colors and poses. VAEs and GANS, being generative models, may sidestep the data paucity problem, by creating synthetic medical data. This was done by Guibas and Virdi, who used a 2 stage GAN to segment and then generate retinal fundus images successfully . Their work was built on the research of Costa et al. , which first described using GANs to generate retinal fundus images. Aside from synthetic data generation, GANs have been used in brain MRI segmentation as well by Moeskops et al. , Kamnitsas et al. and Alex et al.

Data or class imbalance in the training set is also a significant issue in medical image analysis. This refers to the number of images in the training data being skewed towards normal and non-pathological images. Rare diseases are an extreme example of this and can be missed without adequate training examples. This data imbalance effect can be ameliorated by using data augmentation to generate more training images of rare or abnormal data, though there is risk of overfitting. Aside from data-level strategies, algorithmic modification strategies and cost sensitive learning have also been studied . An important, non-technical challenge is the public reception towards their health results being studied by a non-human actor. This situation is not helped by the apocalyptic artificial intelligence scenarios painted by some. Machine learning algorithms have surpassed human performance in image recognition tasks, and it is likely that they will perform better than humans in medical image analysis as well. Indeed, some of the papers in this review report that dermatologists and radiologists have already been bested by machine learning. Yet the question regarding legal and moral culpability arises when a patient is misdiagnosed, or suffers morbidity as a result of AI or AI-assisted medical management. This is accentuated by our inability to fully explain how the black-box of machine algorithms work. However, it is likely that our relationship will continue evolve and recalibrate as AI-based technologies mature and inexorably permeate different facets of our lives.

FUTURE SCOPE

The traditional applications for medical image analysis were discussed . New areas of research include prognostication , content-based image retrieval,image report or caption generation,and manipulation of physical objects with LSTMs and reinforcement learning, involving surgical robots .

A few innovative applications that span across traditional medical image analysis categories are described below. An interesting application was reported by Nie et al. in which GANs were used to generate CT brain images from MRI images. This is remarkable, as it means that patients can potentially avoid the ionizing radiation from a CT scanner altogether, lowering cost and improving patient safety. Nie also exploited the ability of GANs to generate improved, higher resolution images from native images and reduced the blurriness in the CT images. A useful extension of resolution improvement techniques would be applying them to generate MRI images of higher quality. High quality MRI images require high tesla (and correspondingly costlier) MRI scanners. Algorithmically generated high quality MRI images on a lower field-strength scanner would thus lower healthcare costs.

References

[1]. Wang, X. Dong, F. Zhou, L. Cao, and C.-H. Chi, “Coupled attribute similarity learning on categorical data,” IEEE TNNLS, vol. 26, no. 4, pp. 781–797, 2015.

[2]. S. Jian, L. Cao, K. Lu, and H. Gao, “Unsupervised coupled metric similarity for non-iid categorical data,” IEEE TKDE, 2018.

[3] C. Zhu, L. Cao, Q. Liu, J. Yin, and V. Kumar, “Heterogeneous metric learning of categorical data with hierarchical couplings,” IEEE TKDE, 2018.

[4] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” IEEE TPAMI, vol. 35, no. 8, pp. 1798–1828, 2013.

[5] L. Cao, Y. Ou, and S. Y. Philip, “Coupled behavior analysis with applications,” IEEE TKDE, vol. 24, no. 8, pp. 1378–1392, 2012. [6] L. Cao, “Coupling learning of complex interactions,” Information Processing & Management, vol. 51, no. 2, pp. 167–186, 2015.

[7] A. Foss and O. R. Za¨ıane, “A parameterless method for efficiently discovering clusters of arbitrary shape in large datasets,” in Pro- ceedings of ICDM. IEEE, 2002, pp. 179–186.

[8] A. Aizawa, “An information-theoretic perspective of tf–idf mea- sures,” Information Processing & Management, vol. 39, no. 1, pp. 45–65, 2003.

[9] A. Ahmad and L. Dey, “A method to compute distance between two categorical values of same attribute in unsupervised learning for categorical data set,” Pattern Recognition Letters, vol. 28, no. 1, pp. 110–118, 2007. [9] C. Park, D. Kim, J. Oh, and H. Yu, “Predicting user purchase in ecommerce by comprehensive feature engineering and decision boundary focused undersampling,” in RecSys, 2015.

[10] Y. Dong, D. Tao, X. Li, J. Ma, and J. Pu, “Texture classification and retrieval using shearlets and linear regression,” IEEE transactions on cybernetics, vol. 45, no. 3, pp. 358–369, 2015. [11] X. Li, G. Cui, and Y. Dong, “Refined-graph regularization-based nonnegative matrix factorization,” ACM Transactions on Intelligent Systems and Technology (TIST), vol. 9, no. 1, p. 1, 2017.

[12] J. Lee, M. Podlaseck, E. Schonberg, R. Hoch, and S. Gomory, “Analysis and visualization of metrics for online merchandising,” in WEBKDD’99 Workshop.

[13].Muhammad Naeem TahirClassification and characterization of brain tumor MRI by using gray scaled segmenta-tion and DNN