**CHAPTER 1**

**INTRODUCTION**

Ultrasound scans are very important to detect any kind of injury or disease in human body because it used to scan the internal tissues of the body. One major disadvantage of these images is that they include huge amount of noise so doctors face difficulty in finding the exact location of the nerves with in the image. This project aims at identifying the nerves, using canny edge detection. Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations. The general criteria for edge detection include:

* Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible
* The edge point detected from the operator should accurately localize on the center of the edge.
* A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

The project comprises of four modules. The first module deals with adding the patient details and details of ultra sound images to the DB. The second module deals with creating the master control screen, and verification of the existence of image files. The third module deals with Canny edge Detection and generating the edge diagrams. The fourth module deals with identifying ‘Peripheral Neuropathy’ based on patient symptoms and test results.

Tools Used: cv2.canny

Algorithms Used: Canny Edge Detection Algorithm

**Outputs from the project are**

* A set of GUI screens to operate the system
* Plots showing the canny edges in the given Ultra Sound Images
  1. **PROBLEM DEFINITION**

Vision:

* The project aims at developing a tool for Nerve Segmentation in Ultra Sound Images using Canny edge detection.

Mission:

* This tool is developed by using Python along with its layout toolkit PyQt, PyUIC and OpenCV.
  1. **ADVANTAGES**
* The project is useful for doctors in identifying the nerves.
* The project is also useful in analyzing the Ultra Sound images, which are difficult to analyze, because of low quality/noise in the image
* This project finally leads to the improvement of quality of the patients life.
* Technical advantages:
* Using latest Canny edge detection methodology.
* Using Python, which is chosen as the best programming language, by the Programming Community.
* More Functionality can be implemented with less no of lines of code in Python.
* PyQt tool is used to create the Graphical User interfaces.
* All the Front end code is generated automatically by PyUIC.

pace appropriate forin the decreasing order o

**CHAPTER 2**

**SYSTEM ANALYSIS**

**2.1 REQUIREMENT ANALYSIS**

**2.1.1 Hardware Requirements**

1. It requires a minimum of 2.16 GHz processor.

2. It requires a minimum of 4 GB RAM.

3. It requires 64-bit architecture.

4. It requires a minimum storage of 500GB.

**2.1.2 Software Requirements**

1. It requires a 64-bit Ubuntu/windows Operating System.

2. Python Qt Designer for designing user interface.

3. MY SQL server for storing database Entities.

4. Pyuic for converting the layout designed user interface (UI) to python code.

5. Python language for coding.

**2.2 FEASIBILITY ANALYSIS**

As the name implies, a feasibility study is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. It tells us whether a project is worth the investment—in some cases, a project may not be doable. There can be many reasons for this, including requiring too many resources, which not only prevents those resources from performing other tasks but also may cost more than an organization would earn back by taking on a project that isn’t profitable.

**2.2.1 Economical Feasibility**

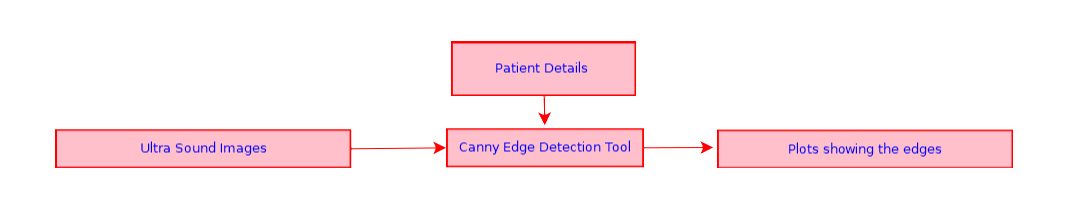
This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility helping decision makers determine the positive economic benefits to the organization that the proposed project will provide. Our project is economically feasible because in this we have used “UBUNTU”, “PYTHON”, “PYQT” designer tool and “PYUIC” which are all available as an open source.

**2.2.2 Technical Feasibility**

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity. Technical feasibility also involves evaluation of the hardware, software, and other technology requirements of the proposed system. A prototype of the tool was developed to verify the technical feasibility. The prototype is working successfully and hence the project is feasible.

**CHAPTER 3**

**SYSTEM DESIGN**

**3.1 SYSTEM DESCRIPTION**

****

**Fig 1: Data Flow Diagram**

Details like Patient Details & Ultra Sound Images are to be provided as Input to the system, using the corresponding user interfaces. The system generates the plots showing canny edges.

The output screens that are created are

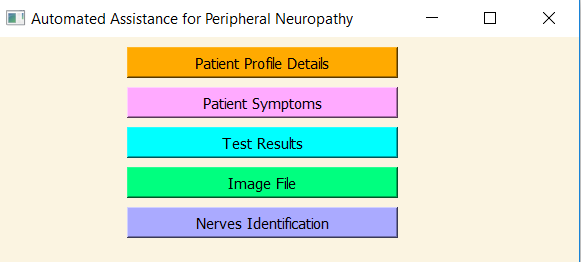


Fig 2. Main GUI Screen

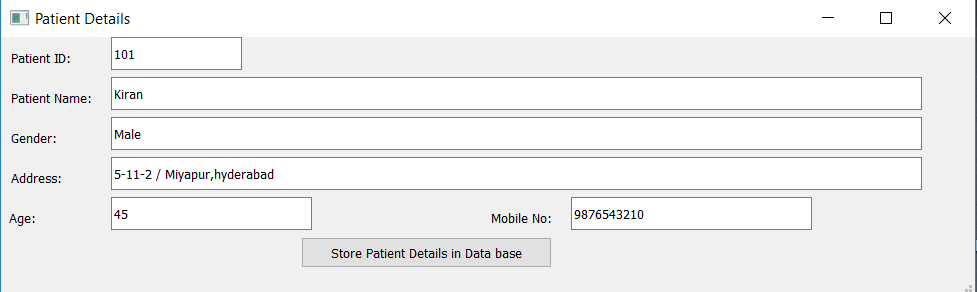


Fig 3.Patient details Screen.

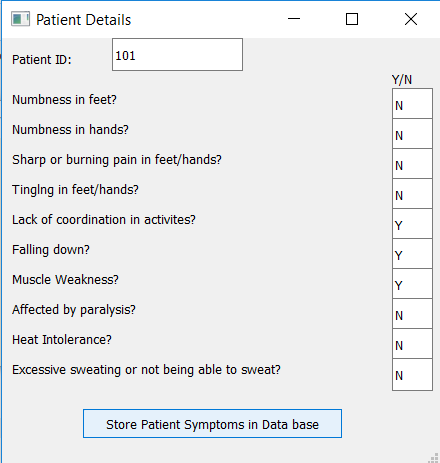


Fig 4: Patient Symptoms Screen.

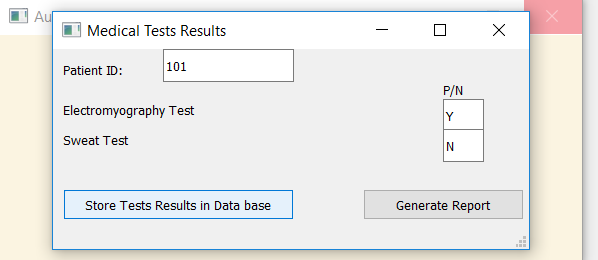


Fig 5. Test details Screen.

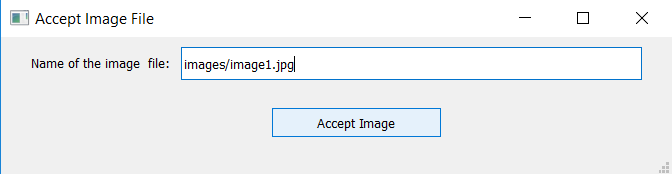


Fig 6. Image Accepting Screen.

**3.2 Tools Used**

**3.2.1.Python Qt Designer**

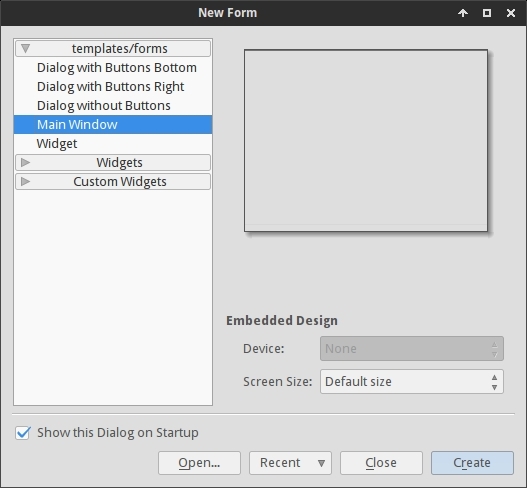
Qt is designed for developing applications and user interfaces once and deploying them across several desktop and mobile operating systems.

The easiest way to start application development with Qt is to [download](http://qt.io/download) and install Qt 5. It contains Qt libraries, examples, documentation, and the necessary development tools, such as the [Qt Creator](http://doc.qt.io/qtcreator/index.html) integrated development environment (IDE).

Qt Creator provides you with tools for accomplishing your tasks throughout the whole application development life-cycle, from creating a project to deploying the application on the target platforms. Qt Creator automates some tasks, such as creating projects, by providing wizards that guide you step-by-step through the project creation process, create the necessary files, and specify settings depending on the choices you make.

The PyQt installer comes with a GUI builder tool called Qt Designer. Using its simple drag and drop interface, a GUI interface can be quickly built without having to write the code. It is however, not an IDE such as Visual Studio. Hence, Qt Designer does not have the facility to debug and build the application.

Creation of a GUI interface using Qt Designer starts with choosing a top-level window for the application.



**Fig 7: Creating GUI Screen**

You can then drag and drop required widgets from the widget box on the left pane. You can also assign value to properties of widget laid on the form.

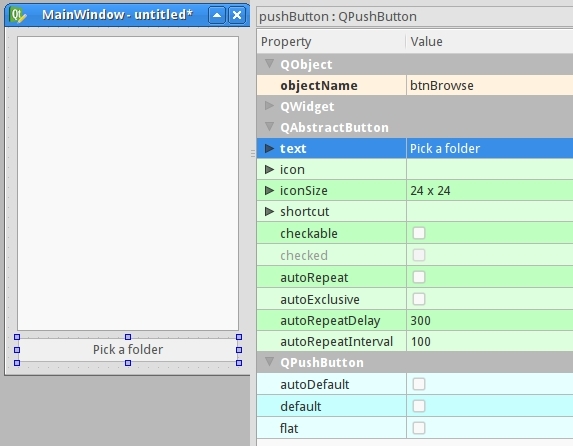
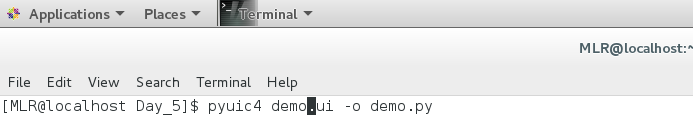


Fig 8: Creating Main Window

The designed form is saved as demo.ui. This ui file contains XML representation of widgets and their properties in the design. This design is translated into Python equivalent by using pyuic4 command line utility. This utility is a wrapper for ui module. The usage of pyuic4 is as follows

****

**Fig 9: Generating .py file from .ui file**

|  |  |
| --- | --- |
| **3.2.2.** | **Widgets** |
|  | **QLabel**  A QLabel object acts as a placeholder to display non-editable text or image, or a movie of animated GIF. It can also be used as a mnemonic key for other widgets. |
|  | **QLineEdit**  QLineEdit object is the most commonly used input field. It provides a box in which one line of text can be entered. In order to enter multi-line text, QTextEdit object is required. |
|  | **QPushButton**  In PyQt API, the QPushButton class object presents a button which when clicked can be programmed to invoke a certain function. |

. Typical buttons are OK, Apply, Cancel, Close, Yes, No and Help.

A command button is rectangular and typically displays a text label describing its action. A shortcut key can be specified by preceding the preferred character with an ampersand in the text. For example:

QPushButton \*button = new QPushButton("&Download", this);

In this example the shortcut is Alt+D. See the QShortcut documentation for details (to display an actual ampersand, use '&&').

Push buttons display a textual label, and optionally a small icon. These can be set using the constructors and changed later using setText() and setIcon(). If the button is disabled the appearance of the text and icon will be manipulated with respect to the GUI style to make the button look "disabled".

A push button emits the signal clicked() when it is activated by the mouse, the Spacebar or by a keyboard shortcut. Connect to this signal to perform the button's action. Push buttons also provide less commonly used signals, for example, pressed() and released().

Command buttons in dialogs are by default auto-default buttons, i.e. they become the default push button automatically when they receive the keyboard input focus. A default button is a push button that is activated when the user presses the Enter or Return key in a dialog. You can change this with setAutoDefault(). Note that auto-default buttons reserve a little extra space which is necessary to draw a default-button indicator. If you do not want this space around your buttons, call setAutoDefault(false).

Being so central, the button widget has grown to accommodate a great many variations in the past decade. The Microsoft style guide now shows about ten different states of Windows push buttons and the text implies that there are dozens more when all the combinations of features are taken into consideration.

The most important modes or states are:

* Available or not (grayed out, disabled).
* Standard push button, toggling push button or menu button.
* On or off (only for toggling push buttons).
* Default or normal. The default button in a dialog can generally be "clicked" using the Enter or Return key.
* Auto-repeat or not.
* Pressed down or not.

As a general rule, use a push button when the application or dialog window performs an action when the user clicks on it (such as Apply, Cancel, Close and Help) and when the widget is supposed to have a wide, rectangular shape with a text label. Small, typically square buttons that change the state of the window rather than performing an action (such as the buttons in the top-right corner of the QFileDialog) are not command buttons, but tool buttons. Qt provides a special class (QToolButton) for these buttons.

If you need toggle behavior (see setCheckable()) or a button that auto-repeats the activation signal when being pushed down like the arrows in a scroll bar (see setAutoRepeat()), a command button is probably not what you want. When in doubt, use a tool button.

A variation of a command button is a menu button. These provide not just one command, but several, since when they are clicked they pop up a menu of options. Use the method setMenu() to associate a popup menu with a push button.

Other classes of buttons are option buttons (see QRadioButton) and check boxes (see QCheckBox).

In Qt, the QAbstractButton base class provides most of the modes and other API, and QPushButton provides GUI logic. See QAbstractButton for more information about the API.

**QLineEdit**

The QLineEdit widget is a one-line text editor.

A line edit allows the user to enter and edit a single line of plain text with a useful collection of editing functions, including undo and redo, cut and paste, and drag and drop.

By changing the echoMode() of a line edit, it can also be used as a "write-only" field, for inputs such as passwords.

The length of the text can be constrained to maxLength(). The text can be arbitrarily constrained using a validator() or an inputMask(), or both. When switching between a validator and an input mask on the same line edit, it is best to clear the validator or input mask to prevent undefined behaviour.

A related class is QTextEdit which allows multi-line, rich text editing.

You can change the text with setText() or insert(). The text is retrieved with text(); the displayed text (which may be different, see EchoMode) is retrieved with displayText(). Text can be selected with setSelection() or selectAll(), and the selection can be cut(), copy()ied and paste()d. The text can be aligned with setAlignment().

When the text changes the textChanged() signal is emitted; when the text changes other than by calling setText() the textEdited() signal is emitted; when the cursor is moved the cursorPositionChanged() signal is emitted; and when the Return or Enter key is pressed the returnPressed() signal is emitted.

When editing is finished, either because the line edit lost focus or Return/Enter is pressed the editingFinished() signal is emitted.

Note that if there is a validator set on the line edit, the returnPressed()/editingFinished() signals will only be emitted if the validator returns QValidator.Acceptable.

By default, QLineEdits have a frame as specified by the Windows and Motif style guides; you can turn it off by calling setFrame(false).

The default key bindings are described below. The line edit also provides a context menu (usually invoked by a right mouse click) that presents some of these editing options.

**QMainWindow**

The QMainWindow class provides a main application window.

Qt Main Window Framework

A main window provides a framework for building an application's user interface. Qt has QMainWindow and its related classes for main window management. QMainWindow has its own layout to which you can add QToolBars, QDockWidgets, a QMenuBar, and a QStatusBar. The layout has a centre area that can be occupied by any kind of widget. You can see an image of the layout below

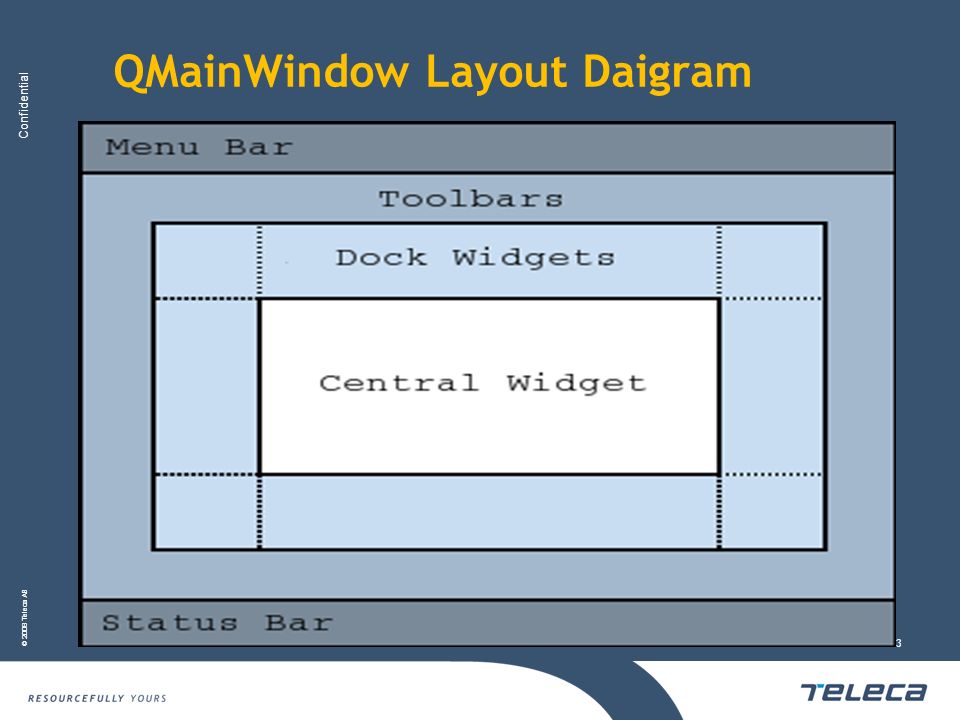


Fig 10: QMainWindow Layout

**Note:** Creating a main window without a central widget is not supported. You must have a central widget even if it is just a placeholder.

Creating Main Window Components

A central widget will typically be a standard Qt widget such as a QTextEdit or a QGraphicsView. Custom widgets can also be used for advanced applications. You set the central widget with setCentralWidget().

Main windows have either a single (SDI) or multiple (MDI) document interface. You create MDI applications in Qt by using a QMdiArea as the central widget.

We will now examine each of the other widgets that can be added to a main window. We give examples on how to create and add them.

**3.2.3.Creating Menus**

Qt implements menus in QMenu and QMainWindow keeps them in a QMenuBar. QActions are added to the menus, which display them as menu items.

You can add new menus to the main window's menu bar by calling menuBar(), which returns the QMenuBar for the window, and then add a menu with QMenuBar::addMenu().

QMainWindow comes with a default menu bar, but you can also set one yourself with setMenuBar(). If you wish to implement a custom menu bar (i.e., not use the QMenuBar widget), you can set it with setMenuWidget().

Creating Main Window Components

A central widget will typically be a standard Qt widget such as a QTextEdit or a QGraphicsView. Custom widgets can also be used for advanced applications. You set the central widget with setCentralWidget().

Main windows have either a single (SDI) or multiple (MDI) document interface. You create MDI applications in Qt by using a QMdiArea as the central widget.

We will now examine each of the other widgets that can be added to a main window. We give examples on how to create and add them.

An example of how to create menus follows:

void MainWindow.createMenus()

{

fileMenu = menuBar()->addMenu(tr("&File"));

fileMenu->addAction(newAct);

fileMenu->addAction(openAct);

fileMenu->addAction(saveAct);

The createPopupMenu() function creates popup menus when the main window receives context menu events. The default implementation generates a menu with the checkable actions from the dock widgets and toolbars. You can reimplement createPopupMenu() for a custom menu.

**3.2.4.Creating Toolbars**

Toolbars are implemented in the QToolBar class. You add a toolbar to a main window with add Toolbar(). You control the initial position of toolbars by assigning them to a specific Qt.ToolBarArea. You can split an area by inserting a toolbar break - think of this as a line break in text editing - with addToolBarBreak() or insertToolBarBreak(). You can also restrict placement by the user with QToolBar.setAllowedAreas() and QToolBar.setMovable().

The size of toolbar icons can be retrieved with iconSize(). The sizes are platform dependent; you can set a fixed size with setIconSize(). You can alter the appearance of all tool buttons in the toolbars with setToolButtonStyle().

An example of toolbar creation follows:

void MainWindow.createToolBars()

{

fileToolBar = add Toolbar(tr("File"));

fileToolBar->addAction(newAct);

isWidgetType() returns whether an object is actually a widget. It is much faster than inherits(“QWidget" ).

Some QObject functions, e.g. children(), objectTrees() and queryList() return a QObjectList. A QObjectList is a QPtrList of QObjects. QObjectLists support the same operations as QPtrLists and have an iterator class, QObjectListIt. To convert the design file to python code saved as design.py, use cd command to change to the directory holding the design.ui file and simply run:

$ pyuic4 design.ui -o design.py

If you want to specify full path for either input or output file you can do that like this:

**$ pyuic4 path/to/design.ui -o output/path/to/design.py**

Writing the code

Now that we have the design.py file with the necessary design part of the application we can create our main application code and logic.

Create a file main.py in the same folder as your design.py file.

Using the design

For the application we'll need the following python modules imported:

**from PyQt4 import QtGui**

**import sys**

We also need the design code we created in the previous steps so add this too:

**import design**

Since the design file will be completely overwritten each time we change something in the design and recreate it we will not be writing any code in it, instead we'll create a new class e.g. ExampleApp that we'll combine with the design code so that we can use all of its features, like this:

**class ExampleApp(QtGui.QMainWindow, design.Ui\_MainWindow):**

**def \_\_init\_\_(self, parent=None):**

**super(ExampleApp, self).\_\_init\_\_(parent)**

**self.setupUi(self)**

In that class we'll interact with the GUI elements, add connections and everything else we need. But first we'll need to initialize that class on our code startup, we'll handle the class instance creation and other stuff in our main() function:

**def main():**

**app = QtGui.QApplication(sys.argv)**

**form = ExampleApp()**

**form.show()**

**app.exec\_()**

And to execute that main function we'll use well known:

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

In the end our whole main.py file looks like this (with short explanations of the code):

from PyQt4 import QtGui # Import the PyQt4 module we'll need

import sys # We need sys so that we can pass argv to QApplication

import design # This file holds our MainWindow and all design related things

class ExampleApp(QtGui.QMainWindow, design.Ui\_MainWindow):

def \_\_init\_\_(self):

super(self.\_\_class\_\_, self).\_\_init\_\_()

self.setupUi(self) # This is defined in design.py file automatically

# It sets up layout and widgets that are defined

def main():

app = QtGui.QApplication(sys.argv) # A new instance of QApplication

form = ExampleApp() # We set the form to be our ExampleApp (design)

form.show() # Show the form

app.exec\_() # and execute the app

if \_\_name\_\_ == '\_\_main\_\_': # if we're running file directly and not importing it

main() # run the main function

Running that will bring up our app running completely from python code!

But clicking button isn't doing anything, so we need to implement those features ourselves.

Implementing functions

(All of the following code is written inside the ExampleApp class)

Let's start with the "Pick a folder" button.

To connect a button event, such as clicked, to a function we use the following code:

**self.btnBrowse.clicked.connect(self.browse\_folder)**

And add it to the \_\_ini\_\_ method of our ExampleApp class so that it's set up when the application starts.

**Code Explanation:**

self.btnBrowse - btnBrowse is the name of the object we defined in Qt Designer. self is self exaplanatory and means that it belongs to current class.

clicked - the event we want to connect. Various elements have various events, for example list widgets have itemSelectionChanged etc.

connect() - used to specify with what we want to connect it with. In our example:

self.browse\_folder - simply a function name that we'll write inside our ExampleApp class:

For getting the directory browser dialog we can use the built in QtGui.QFileDialog.getExistingDirectory method like this:

directory = QtGui.QFileDialog.getExistingDirectory(self,"Pick a folder")

If the user picks a directory the directory variable will be equal to absolute path of the selected directory, otherwise it's None. To avoid running our code any further if the user cancels the folder browse dialog we'll use if directory: statement.

To list the directory contents, we'll need to add os to our imports:

import os

and to get current file list we can use os.listdir(path).

For adding items to the listWidget we use addItem() method on it, and to clear all existing items simply use self.listWidget.clear()

In the end our browse\_folder function looks something like this:

def browse\_folder(self):

self.listWidget.clear()

directory = QtGui.QFileDialog.getExistingDirectory(self,"Pick a folder")

if directory:

for file\_name in os.listdir(directory):

self.listWidget.addItem(file\_name)

Now you can run your app by typing python main.py and you should get the layout you designed and picking the folder will populate list with folder items.

Finished main.py

from PyQt4 import QtGui # Import the PyQt4 module we'll need

import sys # We need sys so that we can pass argv to QApplication

import design # This file holds our Main Window and all design related things

# it also keeps events etc. that we defined in Qt Designer

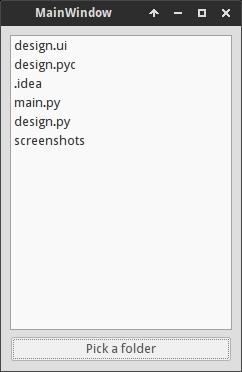


Fig 11.Selecting File

import os # For listing directory methods

class ExampleApp(QtGui.QMainWindow, design.Ui\_MainWindow):

def \_\_init\_\_(self):

super(self.\_\_class\_\_, self).\_\_init\_\_()

self.setupUi(self) # This is defined in design.py file automatically

# It sets up layout and widgets that are defined

self.btnBrowse.clicked.connect(self.browse\_folder) # When the button is pressed

# Execute browse\_folder function

def browse\_folder(self):

self.listWidget.clear() # In case there are any existing elements in the list

directory = QtGui.QFileDialog.getExistingDirectory(self,

"Pick a folder")

# execute getExistingDirectory dialog and set the directory variable to be equal

# to the user selected directory

if directory: # if user didn't pick a directory don't continue

for file\_name in os.listdir(directory): # for all files, if any, in the directory

self.listWidget.addItem(file\_name) # add file to the listWidget

def main():

app = QtGui.QApplication(sys.argv) # A new instance of QApplication

form = ExampleApp() # We set the form to be our ExampleApp (design)

form.show() # Show the form

app.exec\_() # and execute the app

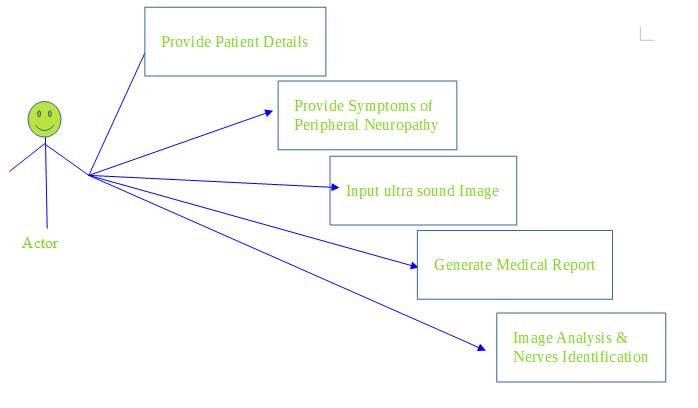
if \_\_name\_\_ == '\_\_main\_\_': # if we're running file directly and not importing it

main() # run the main function

That's the basic logic of using Qt Designer and PyQt to design and develop a GUI application.

**3.3. UML DIAGRAMS**

**3.3.1. Use case diagram**

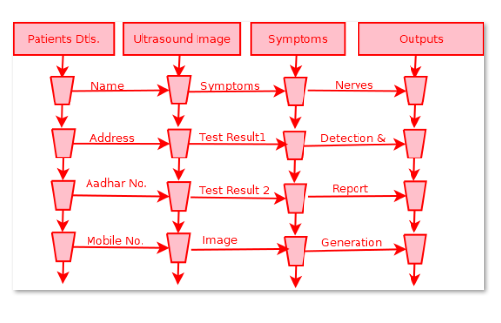


**Fig 12: Use Case Diagram**

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

As we can see the user is interacting with system by a UI through which the customer can perform above mentioned operations like providing patient details,Symptoms details, Images and then generating the report along with the image analysis for identification of nerves.

**3.3.2. Sequence Diagram**

****

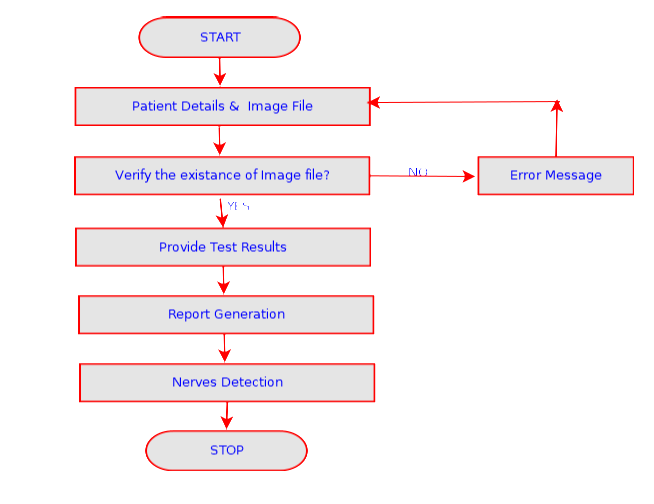
**Fig 13: Sequence diagram**

A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart. A sequence diagram shows object interactions arranged in time sequence.

From above mentioned sequence diagram we have to go in sequence: Enter the needed details as shown in the above figure, Provide the patients details, symptoms, test results and Image as inputs, Identify the nerves and generate the report.

**3.3.3. Activity Diagram**

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. So, the control flow is drawn from one operation to another. In activity diagram we can see that first enter the patient details & image file. The system verifies the existence of the image file. Then generate the .pdf report and identfies the nerves in the image.



**Fig 14. Activity diagram**

**CHAPTER 4**

**IMPLEMENTATION**

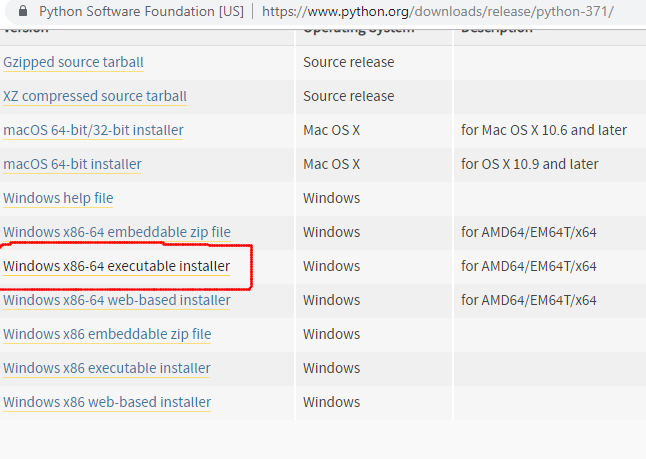
**4.1 SOFTWARE USED**

**4.1.1 Installations.**

1) If the system don't have Python, then install Python by down loading it from the following site.

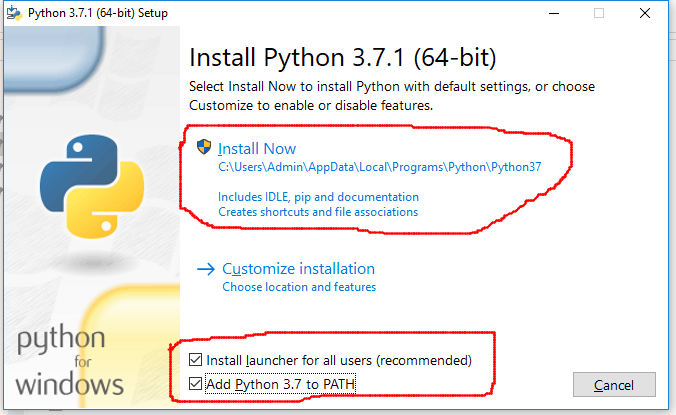
[https://www.python.org/downloads/release/python-371/](https://www.python.org/downloads/release/python-370/)

1.a) Go to the above site, come to the bottom of the page, and click on ‘ Windows x86-64 executable Installer ‘, as shown in the above figure.

1.b) Python Application will be downloaded into the ‘Downloads’ folder as shown in the following figure.

**Fig 15: Downloading Executable Installer**

1.c) Click on python ‘ python-3.7.1-amd64’ to get the following screen. Select both the options at the bottom of the screen, and click on “ Install Now“.

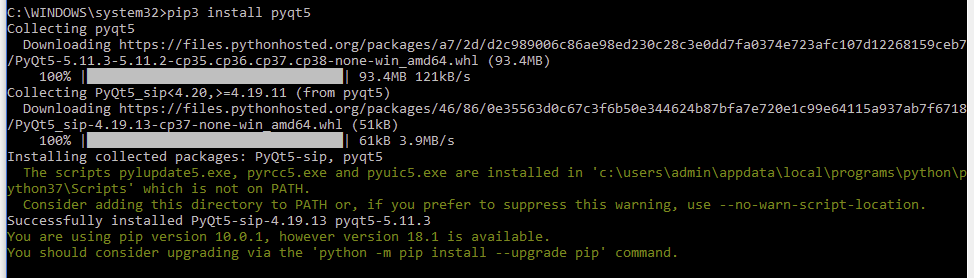
**Fig 16: Installing Python**

1.d) Follow the installation process, and, finish it. After completion, open command prompt, and give the command python -V, as shown below.

2) Close the above command prompt, and re-open it as an administrator, as shown in the following figure.

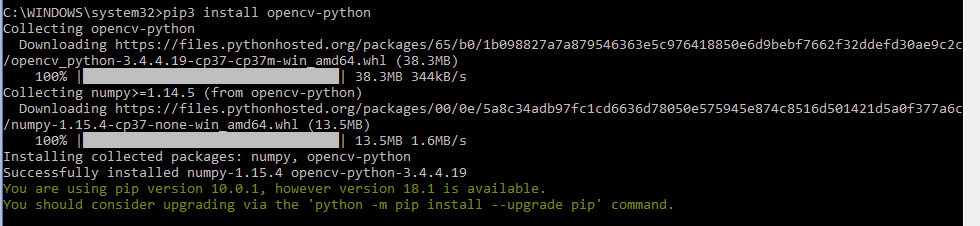
(Note: When you RIGHT-CLICK on command prompt, the submenu will be opened. From there, you can run it as administrator.)

3.a)Execute the command from command prompt: pip3 install wheel. You should get the wheel successfully, installed as shown in the following figure.

**Fig. 17:Installing pyqt5**

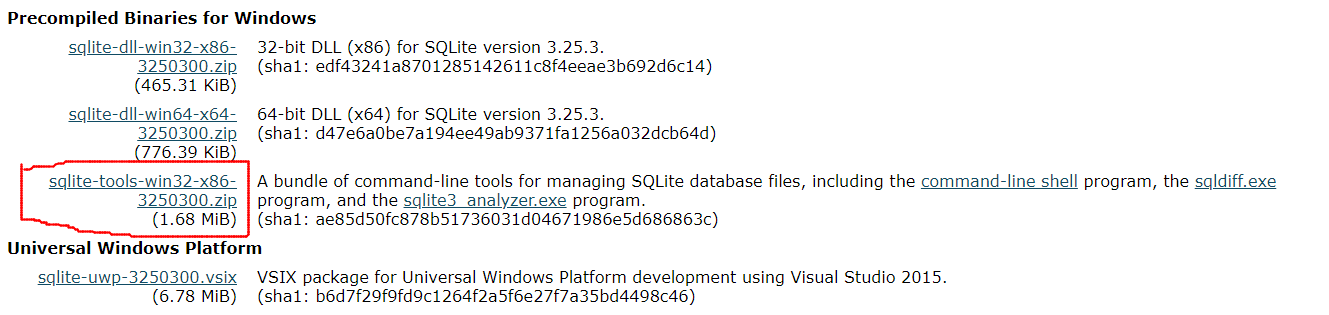
4)Execute the command from command prompt: pip3 install pyqt5. You should get the pyqt5 successfully, installed as shown in the following two figures.

5)Execute the command from command prompt: pip3 install pillow. You should get the pillow successfully, installed as shown in the following figure.

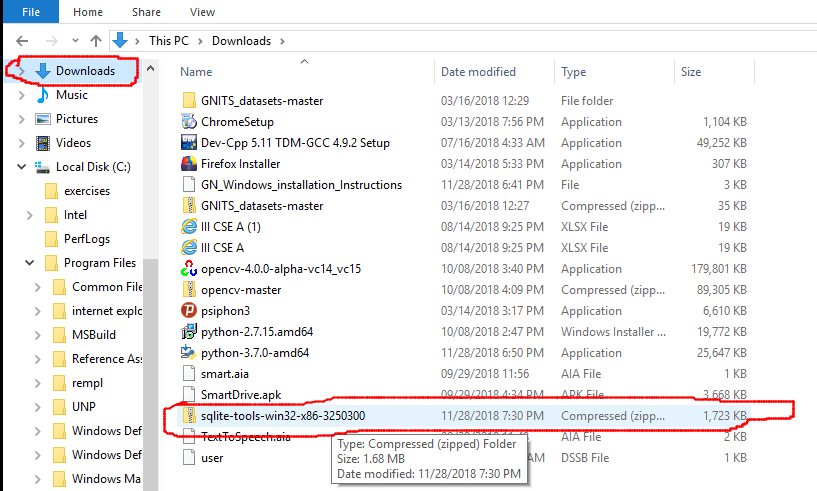
6)Execute the command from command prompt: pip3 install opencv-python. You should get the opencv-python successfully, installed as shown in the following figure.

**Fig 17: Installing open cv**

6.a)Execute the command from command prompt(CMD): pip3 install pyqt5-dev-tools

**** Install sqlite by downloading it from the following site: <https://www.sqlite.org/download.html>. Open this site, go to the bottom, and click on “sqlite-tools-win32-x86-32--------.zip”, as shown in the following figure.

**Fig 18: Downloading sqlite3**

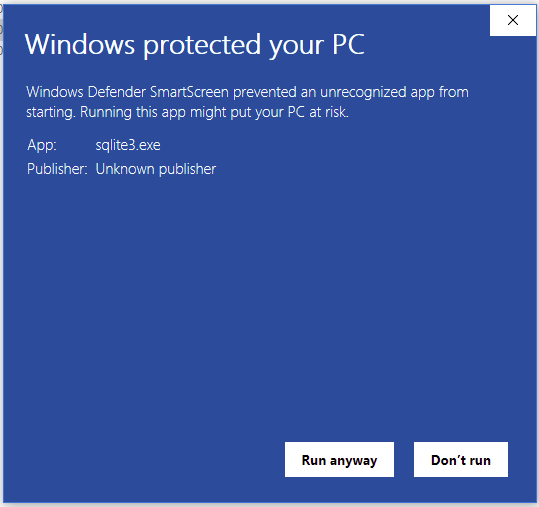
7.a). The sqlite tools, will be downloaded into your downloads folder, as shown in the following figure.

**Fig 19: Installing sqlite3**

7.b) Extract the three files in this compressed file, into a separate folder, and RIGHT CLICK on ‘ sqlite3 ‘ as shown in the following figure.

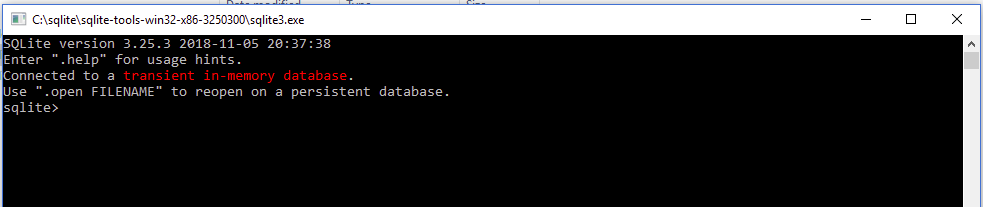
7.c) You are likely to get a message, as shown in the following figure.

7.d) Click on ‘More info’, if you get the above message. Click on ‘ Run anyway’ as shown in the following figure.

****

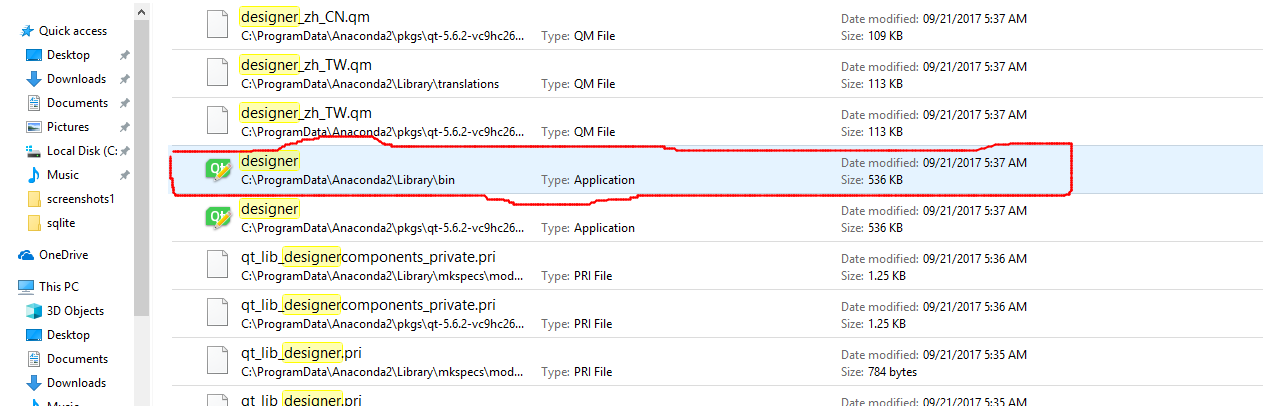
**Fig 20: Run sqlite3.exe**

7.e) You should get a screen as shown below.

**Fig 21:Checking sqlite3 installation**

7.f) Give the command .system dir/p as shown in the below figure. It should display the current directory structure.

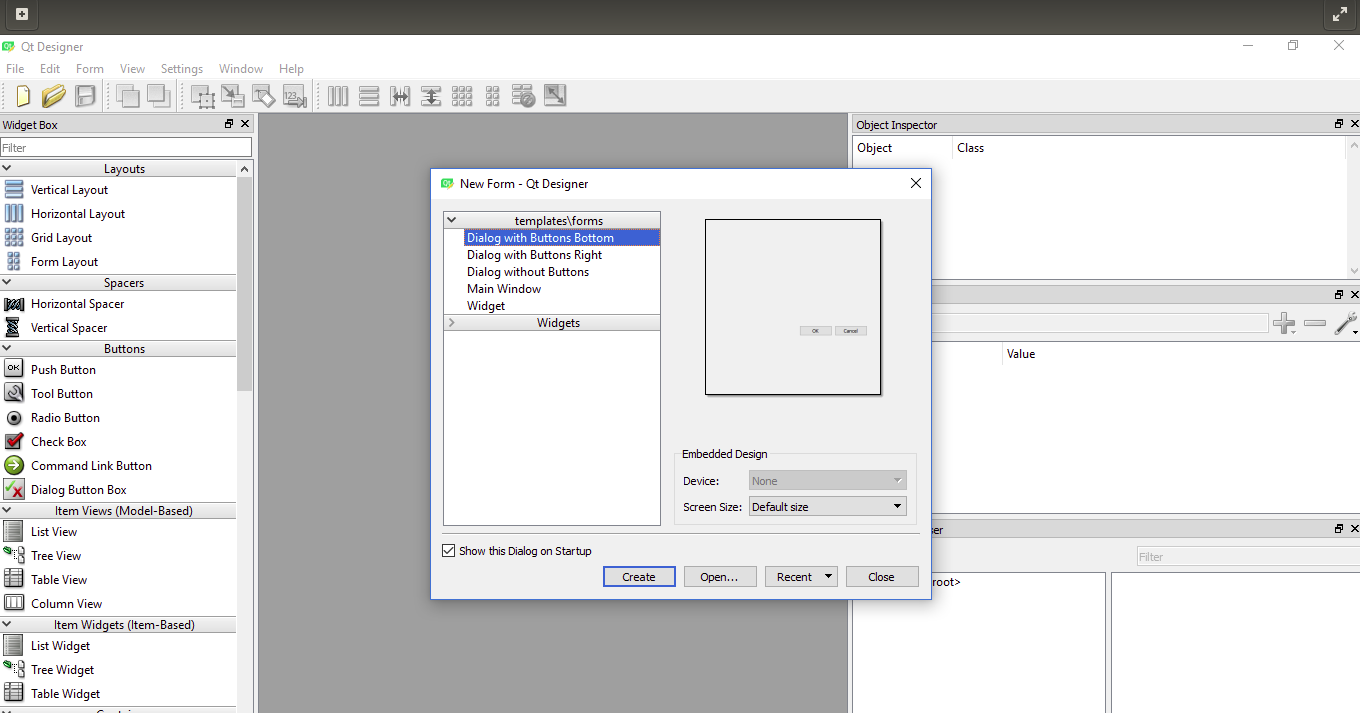
7. g) You can give the command .cd .. to move one level up in the directory. Like wise, .cd ../.. can move you two levels up.

8). Open windows explorer, and search for ‘Designer’. This search will take a few minutes. Find out whether you have Anaconda’s designer, as shown in below figures.

**Fig 22: Opening Qt Designer**

8. a) If you have the designer, as shown in the above figure.

8.b). Click on the above designer. You should get the following.

**Fig 23: Qt Designer**

8.c) If you are able to get the above, then you can ignore the remaining installation steps

**4.1.2 Languages**

Python was conceived in the late 1980s, and its implementation began in December 1989 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing.

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations.

**Features and Philosophy**

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta programming and meta objects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

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

**4.2.Code**

**Images.py**

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(671, 138)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.pushButton = QtWidgets.QPushButton(self.centralwidget)

self.pushButton.setGeometry(QtCore.QRect(270, 70, 171, 31))

self.pushButton.setObjectName("pushButton")

self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit.setGeometry(QtCore.QRect(180, 10, 461, 33))

self.lineEdit.setObjectName("lineEdit")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(30, 10, 171, 31))

self.label.setObjectName("label")

MainWindow.setCentralWidget(self.centralwidget)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Accept Image File"))

self.pushButton.setText(\_translate("MainWindow", "Accept Image"))

self.label.setText(\_translate("MainWindow", "Name of the image file:"))

**Neuro1.py**

import sys

import os

from neuro import \*

from PyQt5 import QtWidgets, QtGui, QtCore

class MyForm(QtWidgets.QMainWindow):

def \_\_init\_\_(self,parent=None):

QtWidgets.QWidget.\_\_init\_\_(self,parent)

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

self.ui.pushButton.clicked.connect(self.psympts)

self.ui.pushButton\_2.clicked.connect(self.ptstrslts)

self.ui.pushButton\_3.clicked.connect(self.imgfls)

self.ui.pushButton\_4.clicked.connect(self.nerveidn)

self.ui.pushButton\_5.clicked.connect(self.pdetails)

def psympts(self):

os.system("python psymptoms1.py")

def pdetails(self):

os.system("python patientdtls1.py")

def ptstrslts(self):

os.system("python ptests1.py")

def imgfls(self):

os.system("python images1.py")

def nerveidn(self):

os.system("python seg1.py -i IMAGES")

# def gnrep(self):

# os.system("python genrep1.py")

if \_\_name\_\_ == "\_\_main\_\_":

app = QtWidgets.QApplication(sys.argv)

myapp = MyForm()

myapp.show()

sys.exit(app.exec\_())

**Patient.py**

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(974, 261)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(0, 170, 91, 21))

self.label.setObjectName("label")

self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit.setGeometry(QtCore.QRect(110, 160, 201, 33))

self.lineEdit.setObjectName("lineEdit")

self.pushButton = QtWidgets.QPushButton(self.centralwidget)

self.pushButton.setGeometry(QtCore.QRect(300, 200, 251, 31))

self.pushButton.setObjectName("pushButton")

self.lineEdit\_3 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_3.setGeometry(QtCore.QRect(110, 0, 131, 33))

self.lineEdit\_3.setObjectName("lineEdit\_3")

self.lineEdit\_4 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_4.setGeometry(QtCore.QRect(110, 40, 811, 33))

self.lineEdit\_4.setObjectName("lineEdit\_4")

self.lineEdit\_5 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_5.setGeometry(QtCore.QRect(110, 80, 811, 33))

self.lineEdit\_5.setObjectName("lineEdit\_5")

self.lineEdit\_6 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_6.setGeometry(QtCore.QRect(110, 120, 811, 33))

self.lineEdit\_6.setObjectName("lineEdit\_6")

self.label\_3 = QtWidgets.QLabel(self.centralwidget)

self.label\_3.setGeometry(QtCore.QRect(10, 10, 81, 21))

self.label\_3.setObjectName("label\_3")

self.label\_4 = QtWidgets.QLabel(self.centralwidget)

self.label\_4.setGeometry(QtCore.QRect(10, 50, 101, 21))

self.label\_4.setObjectName("label\_4")

self.label\_5 = QtWidgets.QLabel(self.centralwidget)

self.label\_5.setGeometry(QtCore.QRect(10, 130, 141, 21))

self.label\_5.setObjectName("label\_5")

self.label\_6 = QtWidgets.QLabel(self.centralwidget)

self.label\_6.setGeometry(QtCore.QRect(10, 90, 91, 21))

self.label\_6.setObjectName("label\_6")

self.label\_2 = QtWidgets.QLabel(self.centralwidget)

self.label\_2.setGeometry(QtCore.QRect(490, 170, 81, 21))

self.label\_2.setObjectName("label\_2")

self.lineEdit\_2 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_2.setGeometry(QtCore.QRect(570, 160, 241, 33))

self.lineEdit\_2.setObjectName("lineEdit\_2")

MainWindow.setCentralWidget(self.centralwidget)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Patient Details"))

self.label.setText(\_translate("MainWindow", " Age:"))

self.pushButton.setText(\_translate("MainWindow", "Store Patient Details in Data base"))

self.label\_3.setText(\_translate("MainWindow", "Patient ID:"))

self.label\_4.setText(\_translate("MainWindow", "Patient Name:"))

self.label\_5.setText(\_translate("MainWindow", "Address:"))

self.label\_6.setText(\_translate("MainWindow", "Gender:"))

self.label\_2.setText(\_translate("MainWindow", "Mobile No:"))

**Patientdtls.py**

import sys

from patient import \*

from PyQt5 import QtWidgets, QtGui, QtCore

import sqlite3

con = sqlite3.connect('neuro1')

class MyForm(QtWidgets.QMainWindow):

def \_\_init\_\_(self,parent=None):

QtWidgets.QWidget.\_\_init\_\_(self,parent)

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

self.ui.pushButton.clicked.connect(self.insertvalues)

def insertvalues(self):

with con:

cur = con.cursor()

aadhar = str(self.ui.lineEdit.text())

pid = str(self.ui.lineEdit\_3.text())

pname = str(self.ui.lineEdit\_4.text())

addr1 = str(self.ui.lineEdit\_5.text())

addr2 = str(self.ui.lineEdit\_6.text())

mobile = str(self.ui.lineEdit\_2.text())

cur.execute('INSERT INTO patient VALUES(?,?,?,?,?,?)',(pid,pname,addr1,addr2,aadhar,mobile))

con.commit()

if \_\_name\_\_ == "\_\_main\_\_":

app = QtWidgets.QApplication(sys.argv)

myapp = MyForm()

myapp.show()

sys.exit(app.exec\_())

**Psymptoms.py**

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(439, 428)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.pushButton = QtWidgets.QPushButton(self.centralwidget)

self.pushButton.setGeometry(QtCore.QRect(80, 370, 261, 31))

self.pushButton.setObjectName("pushButton")

self.lineEdit\_3 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_3.setGeometry(QtCore.QRect(110, 0, 131, 33))

self.lineEdit\_3.setObjectName("lineEdit\_3")

self.lineEdit\_4 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_4.setGeometry(QtCore.QRect(390, 50, 41, 33))

self.lineEdit\_4.setObjectName("lineEdit\_4")

self.label\_3 = QtWidgets.QLabel(self.centralwidget)

self.label\_3.setGeometry(QtCore.QRect(10, 10, 81, 21))

self.label\_3.setObjectName("label\_3")

self.label\_4 = QtWidgets.QLabel(self.centralwidget)

self.label\_4.setGeometry(QtCore.QRect(10, 50, 361, 21))

self.label\_4.setObjectName("label\_4")

self.label\_5 = QtWidgets.QLabel(self.centralwidget)

self.label\_5.setGeometry(QtCore.QRect(390, 30, 41, 21))

self.label\_5.setObjectName("label\_5")

self.lineEdit\_5 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_5.setGeometry(QtCore.QRect(390, 80, 41, 33))

self.lineEdit\_5.setObjectName("lineEdit\_5")

self.lineEdit\_6 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_6.setGeometry(QtCore.QRect(390, 110, 41, 33))

self.lineEdit\_6.setObjectName("lineEdit\_6")

self.lineEdit\_7 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_7.setGeometry(QtCore.QRect(390, 140, 41, 33))

self.lineEdit\_7.setObjectName("lineEdit\_7")

self.lineEdit\_8 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_8.setGeometry(QtCore.QRect(390, 170, 41, 33))

self.lineEdit\_8.setObjectName("lineEdit\_8")

self.lineEdit\_9 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_9.setGeometry(QtCore.QRect(390, 200, 41, 33))

self.lineEdit\_9.setObjectName("lineEdit\_9")

self.lineEdit\_10 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_10.setGeometry(QtCore.QRect(390, 230, 41, 33))

self.lineEdit\_10.setObjectName("lineEdit\_10")

self.lineEdit\_11 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_11.setGeometry(QtCore.QRect(390, 260, 41, 33))

self.lineEdit\_11.setObjectName("lineEdit\_11")

self.lineEdit\_12 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_12.setGeometry(QtCore.QRect(390, 290, 41, 33))

self.lineEdit\_12.setObjectName("lineEdit\_12")

self.lineEdit\_13 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_13.setGeometry(QtCore.QRect(390, 320, 41, 33))

self.lineEdit\_13.setObjectName("lineEdit\_13")

self.label\_6 = QtWidgets.QLabel(self.centralwidget)

self.label\_6.setGeometry(QtCore.QRect(10, 80, 371, 21))

self.label\_6.setObjectName("label\_6")

self.label\_7 = QtWidgets.QLabel(self.centralwidget)

self.label\_7.setGeometry(QtCore.QRect(10, 110, 381, 21))

self.label\_7.setObjectName("label\_7")

self.label\_8 = QtWidgets.QLabel(self.centralwidget)

self.label\_8.setGeometry(QtCore.QRect(10, 140, 371, 21))

self.label\_8.setObjectName("label\_8")

self.label\_9 = QtWidgets.QLabel(self.centralwidget)

self.label\_9.setGeometry(QtCore.QRect(10, 170, 341, 21))

self.label\_9.setObjectName("label\_9")

self.label\_10 = QtWidgets.QLabel(self.centralwidget)

self.label\_10.setGeometry(QtCore.QRect(10, 200, 341, 21))

self.label\_10.setObjectName("label\_10")

self.label\_11 = QtWidgets.QLabel(self.centralwidget)

self.label\_11.setGeometry(QtCore.QRect(10, 320, 321, 21))

self.label\_11.setObjectName("label\_11")

self.label\_12 = QtWidgets.QLabel(self.centralwidget)

self.label\_12.setGeometry(QtCore.QRect(10, 230, 321, 21))

self.label\_12.setObjectName("label\_12")

self.label\_13 = QtWidgets.QLabel(self.centralwidget)

self.label\_13.setGeometry(QtCore.QRect(10, 260, 321, 21))

self.label\_13.setObjectName("label\_13")

self.label\_15 = QtWidgets.QLabel(self.centralwidget)

self.label\_15.setGeometry(QtCore.QRect(10, 290, 321, 21))

self.label\_15.setObjectName("label\_15")

MainWindow.setCentralWidget(self.centralwidget)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Patient Details"))

self.pushButton.setText(\_translate("MainWindow", "Store Patient Symptoms in Data base"))

self.label\_3.setText(\_translate("MainWindow", "Patient ID:"))

self.label\_4.setText(\_translate("MainWindow", "Numbness in feet?"))

self.label\_5.setText(\_translate("MainWindow", "Y/N"))

self.label\_6.setText(\_translate("MainWindow", "Numbness in hands?"))

self.label\_7.setText(\_translate("MainWindow", "Sharp or burning pain in feet/hands?"))

self.label\_8.setText(\_translate("MainWindow", "Tinglng in feet/hands?"))

self.label\_9.setText(\_translate("MainWindow", "Lack of coordination in activites?"))

self.label\_10.setText(\_translate("MainWindow", "Falling down?"))

self.label\_11.setText(\_translate("MainWindow", "Excessive sweating or not being able to sweat?"))

self.label\_12.setText(\_translate("MainWindow", "Muscle Weakness?"))

self.label\_13.setText(\_translate("MainWindow", "Affected by paralysis?"))

self.label\_15.setText(\_translate("MainWindow", "Heat Intolerance?"))

**Ptests.py**

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(476, 200)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.pushButton = QtWidgets.QPushButton(self.centralwidget)

self.pushButton.setGeometry(QtCore.QRect(10, 140, 231, 31))

self.pushButton.setObjectName("pushButton")

self.lineEdit\_3 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_3.setGeometry(QtCore.QRect(110, 0, 131, 33))

self.lineEdit\_3.setObjectName("lineEdit\_3")

self.lineEdit\_4 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_4.setGeometry(QtCore.QRect(390, 50, 41, 33))

self.lineEdit\_4.setObjectName("lineEdit\_4")

self.label\_3 = QtWidgets.QLabel(self.centralwidget)

self.label\_3.setGeometry(QtCore.QRect(10, 10, 81, 21))

self.label\_3.setObjectName("label\_3")

self.label\_4 = QtWidgets.QLabel(self.centralwidget)

self.label\_4.setGeometry(QtCore.QRect(10, 50, 191, 21))

self.label\_4.setObjectName("label\_4")

self.label\_5 = QtWidgets.QLabel(self.centralwidget)

self.label\_5.setGeometry(QtCore.QRect(390, 30, 41, 21))

self.label\_5.setObjectName("label\_5")

self.lineEdit\_5 = QtWidgets.QLineEdit(self.centralwidget)

self.lineEdit\_5.setGeometry(QtCore.QRect(390, 80, 41, 33))

self.lineEdit\_5.setObjectName("lineEdit\_5")

self.label\_6 = QtWidgets.QLabel(self.centralwidget)

self.label\_6.setGeometry(QtCore.QRect(10, 80, 201, 21))

self.label\_6.setObjectName("label\_6")

self.pushButton\_2 = QtWidgets.QPushButton(self.centralwidget)

self.pushButton\_2.setGeometry(QtCore.QRect(310, 140, 161, 31))

self.pushButton\_2.setObjectName("pushButton\_2")

MainWindow.setCentralWidget(self.centralwidget)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Medical Tests Results"))

self.pushButton.setText(\_translate("MainWindow", "Store Tests Results in Data base"))

self.label\_3.setText(\_translate("MainWindow", "Patient ID:"))

self.label\_4.setText(\_translate("MainWindow", "Electromyography Test"))

self.label\_5.setText(\_translate("MainWindow", "P/N"))

self.label\_6.setText(\_translate("MainWindow", "Sweat Test"))

self.pushButton\_2.setText(\_translate("MainWindow", "Generate Report"))

**Seg1.py**

import numpy as np

import argparse

import glob

import cv2

def auto\_canny(image, sigma=0.33):

# compute the median of the single channel pixel intensities

v = np.median(image)

# apply automatic Canny edge detection using the computed median

lower = int(max(0, (1.0 - sigma) \* v))

upper = int(min(255, (1.0 + sigma) \* v))

edged = cv2.Canny(image, lower, upper)

return edged

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-i", "--images", required=True,

help="path to input dataset of images")

args = vars(ap.parse\_args())

# loop over the images

for imagePath in glob.glob(args["images"] + "/\*.jpg"):

# load the image, convert it to grayscale, and blur it slightly

image = cv2.imread(imagePath)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (3, 3), 0)

# apply Canny edge detection using a wide threshold, tight

# threshold, and automatically determined threshold

wide = cv2.Canny(blurred, 10, 200)

tight = cv2.Canny(blurred, 225, 250)

auto = auto\_canny(blurred)

# show the images

cv2.imshow("Original", image)

cv2.imshow("Edges", np.hstack([wide]))

cv2.waitKey(0)

**4.3.Modules Imported**

**OS Module:**

The OS module in Python provides a way of using operating system dependent functionality.

The functions that the OS module provides allows you to interface with the

underlying operating system that Python is running on – be that Windows, Mac or Linux.

**OS functions:**

Executing a shell command

os.system()

Get the users environment

os.environ()

Returns the current working directory.

os.getcwd()

Return the real group id of the current process.

os.getgid()

Return the current process’s user id.

os.getuid()

Returns the real process ID of the current process.

os.getpid()

Set the current numeric umask and return the previous umask.

os.umask(mask)

Return information identifying the current operating system.

os.uname()

Change the root directory of the current process to path.

os.chroot(path)

Return a list of the entries in the directory given by path.

os.listdir(path)

Create a directory named path with numeric mode mode.

os.mkdir(path)

Recursive directory creation function.

os.makedirs(path)

Remove (delete) the file path.

os.remove(path)

Remove directories recursively.

os.removedirs(path)

Rename the file or directory src to dst.

os.rename(src, dst)

Remove (delete) the directory path.

os.rmdir(path)

**Sys Module:**

The sys module provides information about constants, functions and methods of the Python interpreter. dir(system) gives a summary of the available constants, functions and methods. Another possibility is the help() function. Using help(sys) provides valuable detail information.

The module sys informs e.g. about the maximal recursion depth (sys.getrecursionlimit() ) and provides the possibility to change (sys.setrecursionlimit()).

The current version number of Python can be accessed as well by using this module.

Lots of scripts need access to the arguments passed to the script, when the script was started. argvargv (or to be precise sys.argv) is a list, which contains the command-line arguments passed to the script. The first item of this list contains the name of the script itself. The arguments follow the script name.

Every serious user of a UNIX or Linux operating system knows standard streams, i.e. input, standard output and standard error. They are known as pipes. They are commonly abbreviated as stdin, stdout, stderr.

The standard input (stdin) is normally connected to the keyboard, while the standard error and standard output go to the terminal (or window) in which you are working.

These data streams can be accessed from Python via the objects of the sys module with the same names, i.e. sys.stdin, sys.stdout and sys.stderr.

The standard output (stdout) can be redirected e.g. into a file, so that we can process this file later with another program. The same is possible with the standard error stream, we can redirect it into a file as well. We can redirect both stderr and stdout into the same file or into separate files.

**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.

NumPy is module for Python. The name is an acronym for "Numeric Python" or "Numerical Python". It is an extension module for Python, mostly written in C. This makes sure that the precompiled mathematical and numerical functions and functionalities of Numpy guarantee great execution speed.

Furthermore, NumPy enriches the programming language Python with powerful data structures, implementing multi-dimensional arrays and matrices. These data structures guarantee efficient calculations with matrices and arrays. The implementation is even aiming at huge matrices and arrays, better known under the heading of "big data". Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.

SciPy (Scientific Python) is often mentioned in the same breath with NumPy. SciPy needs Numpy, as it is based on the data structures of Numpy and furthermore its basic creation and manipulation functions. It extends the capabilities of NumPy with further useful functions for minimization, regression, Fourier-transformation and many others.

Both NumPy and SciPy are not part of a basic Python installation. They have to be installed after the Python installation. NumPy has to be installed before installing SciPy.

NumPy is based on two earlier Python modules dealing with arrays. One of these is Numeric. Numeric is like NumPy a Python module for high-performance, numeric computing, but it is obsolete nowadays. Another predecessor of NumPy is Numarray, which is a complete rewrite of Numeric but is deprecated as well. NumPy is a merger of those two, i.e. it is built on the code of Numeric and the features of Numarray.

NumPy’s array class is called ndarray. It is also known by the alias array. Note that numpy.array is not the same as the Standard Python Library class array.array, which only handles one-dimensional arrays and offers less functionality. The more important attributes of an ndarray object are:

**ndarray.ndim**

The number of axes (dimensions) of the array.

**ndarray.shape**

The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

**ndarray.size**

The total number of elements of the array. This is equal to the product of the elements of shape.

**ndarray.dtype**

An object describing the type of the elements in the array. One can create or specify dtype’s using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

**ndarray.itemsize**

The size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.

**ndarray.data**

The buffer containing the actual elements of the array. Normally, we won’t need to use this attribute because we will access the elements in an array using indexing facilities.

**CHAPTER 5**

**TESTING**

**5.1 SOFTWARE TESTING**

Software testing is the process of evaluation a software item to detect differences between given input and expected output. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words, software testing is a verification and validation process.

**Verification**

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to.

**Validation**

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

**Basics of software testing**

There are two basics of software testing: Black box testing and white box testing.

**Black box Testing**

Black box testing is a testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

**White box Testing**

White box testing is a testing technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

Black box testing is often used for validation and white box testing is often used for verification.

**5.1.1 Types of testing**

There are many types of testing like

* + Unit Testing
  + Integration Testing
  + Functional Testing
  + System Testing
  + Regression Testing etc.

**Unit Testing**

Unit testing is the testing of an individual unit or group of related units. It falls under the class of white box testing. It is often done by the programmer to test that the unit he/she has implemented is producing expected output against given input.

**Integration Testing**

Integration testing is testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested in integration testing if software and hardware components have any relation. It may fall under both white box testing and black box testing.

**Functional Testing**

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

**System Testing**

System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing.

**Regression Testing**

Regression testing is the testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. It falls under the class of black box testing.

**CHAPTER 6**

**TESTING RESULTS**

**Step 1:**

The below screenshot is the first step in the project. Here first we go to the respective directory where the project is located and check the ui files in it.

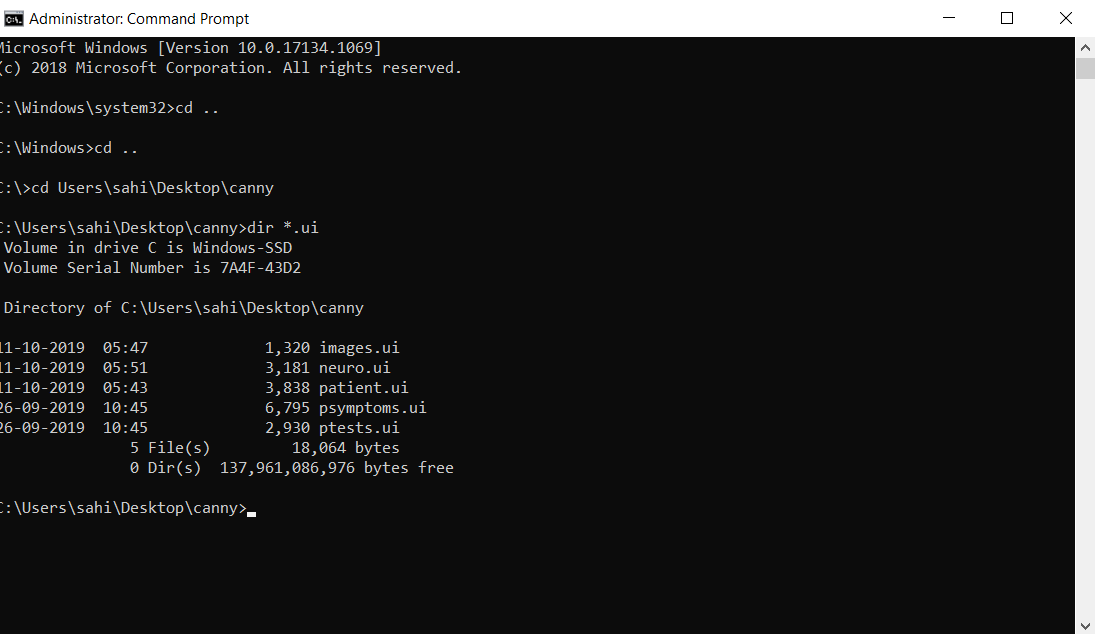
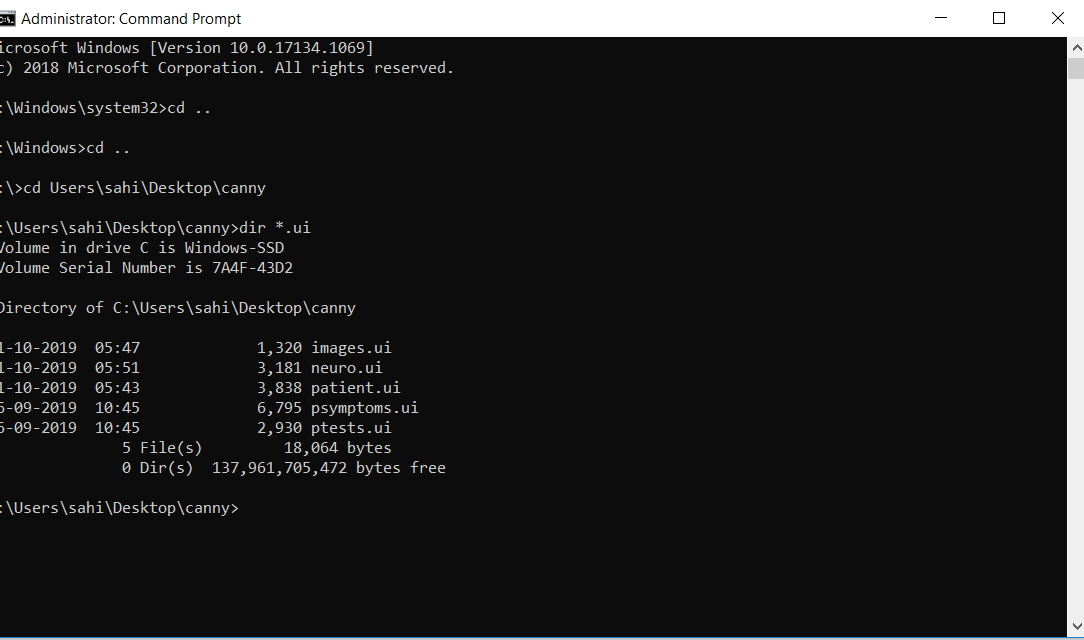


Fig 24: Checking the ui files

**Step 2:**

Now convert all the .ui files to .py files



**Fig 25: Converting .ui to .py file**

**Step 3**:

Now open the neuro1.py file. You can see the main GUI Screen containing the buttons.

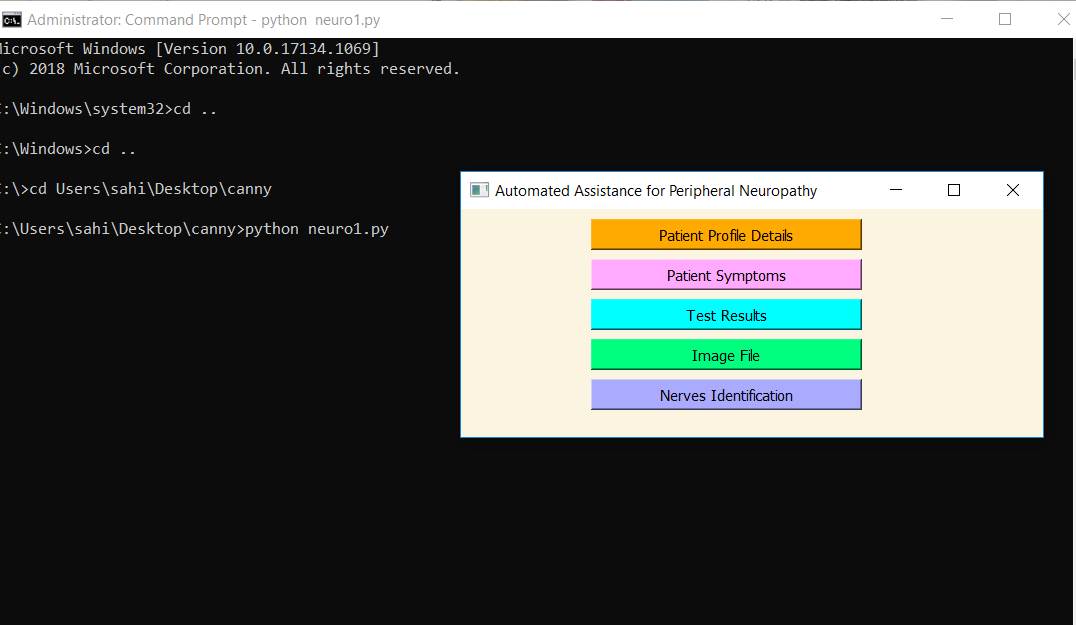


Fig 26: Opening the main window

**Step 4:**

On clicking the first button,we get this patient details window.Fill the patient details and click on Store details in database button.

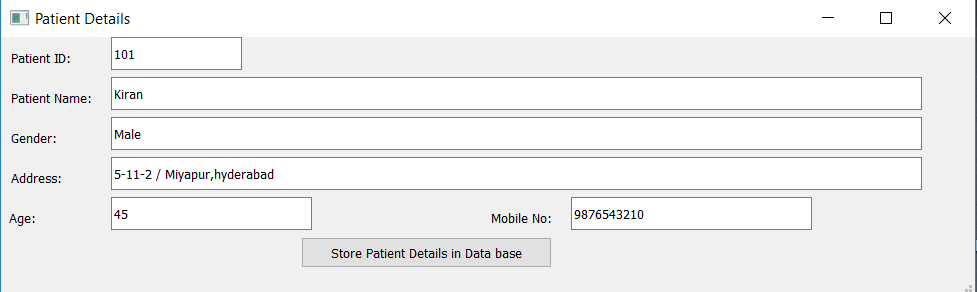


Fig 27: Patient details window

**Step 5:**

Check whether the patient details are inserted in database correctly or not in sqlite3.

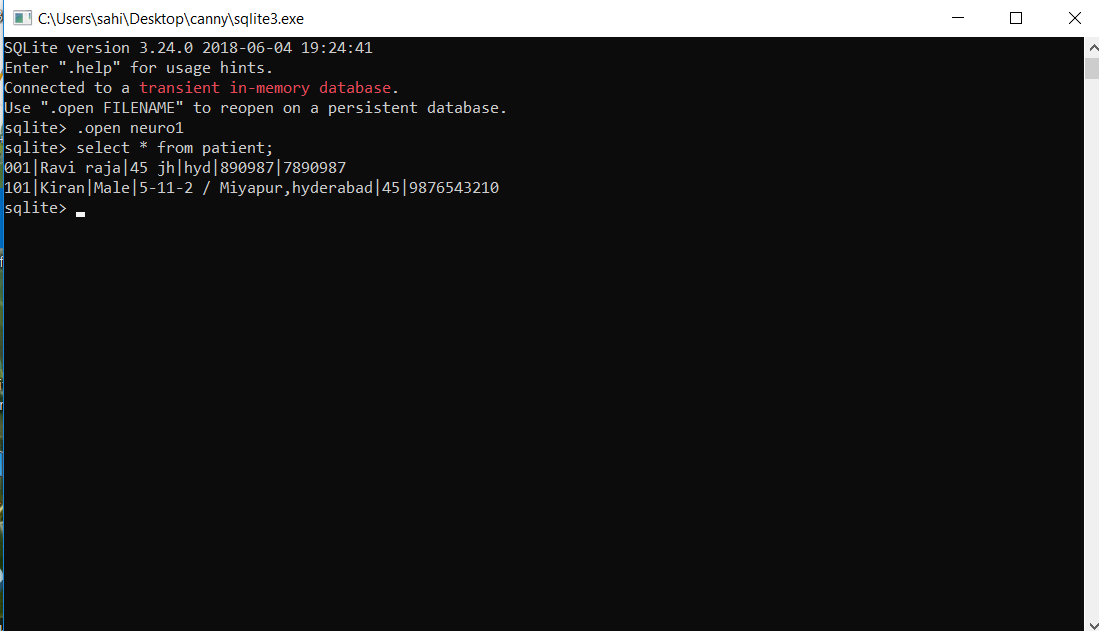


Fig 28; Checking the patient details in database

Step 6:

On clicking the second button in main GUI screen we get this patient symptoms window. Fill all the symptoms as Y or N and click on the Store details in database button.

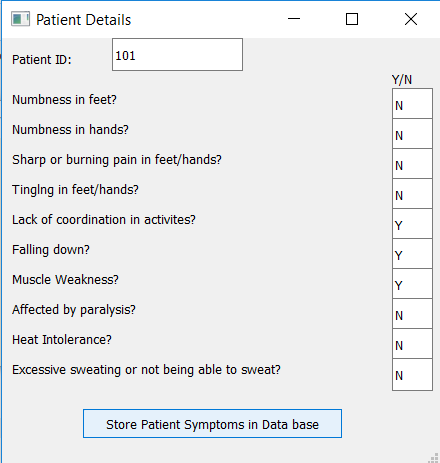
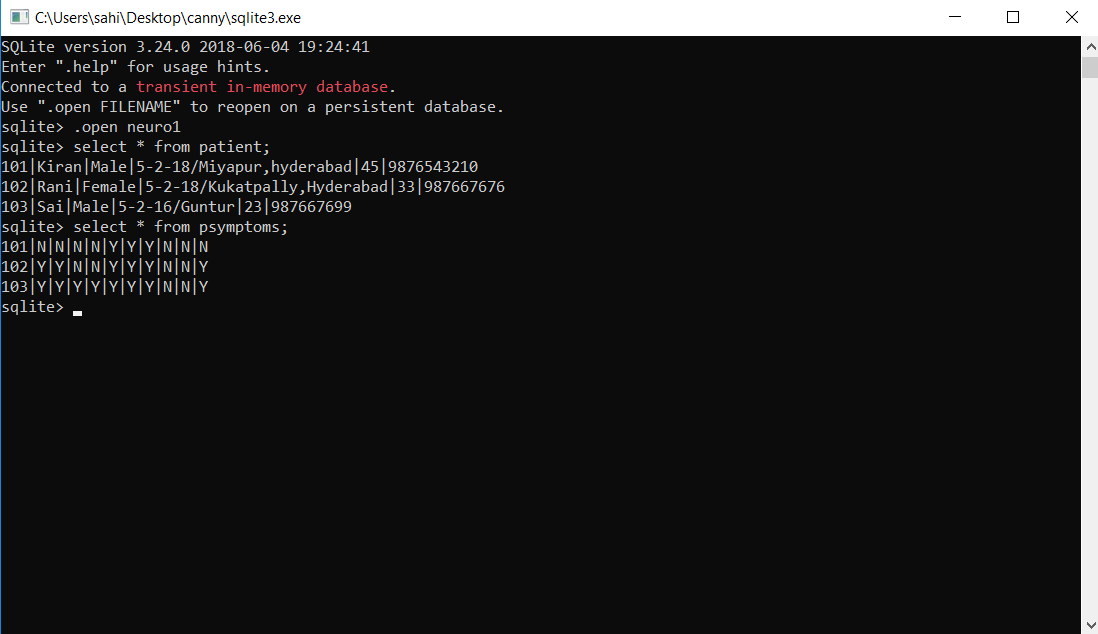


Fig 29: Patient Symptoms Window

**Step 7:**

Check whether the patient symptoms are inserted in database correctly or not in sqlite3.



**Fig 30: Checking the patient symptoms in the database.**

**Step 8:**

On clicking the third button in main GUI screen we get this patient test results window. Fill all the symptoms as Y or N and click on the Store details in database button.

Click on Generate report button.

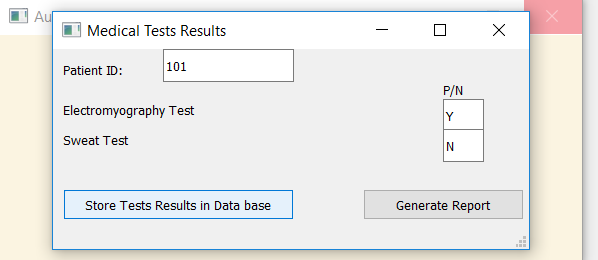


Fig 31: Test results window

**Step 9:**

You can see the medical report generated in the folder where the python files are located.

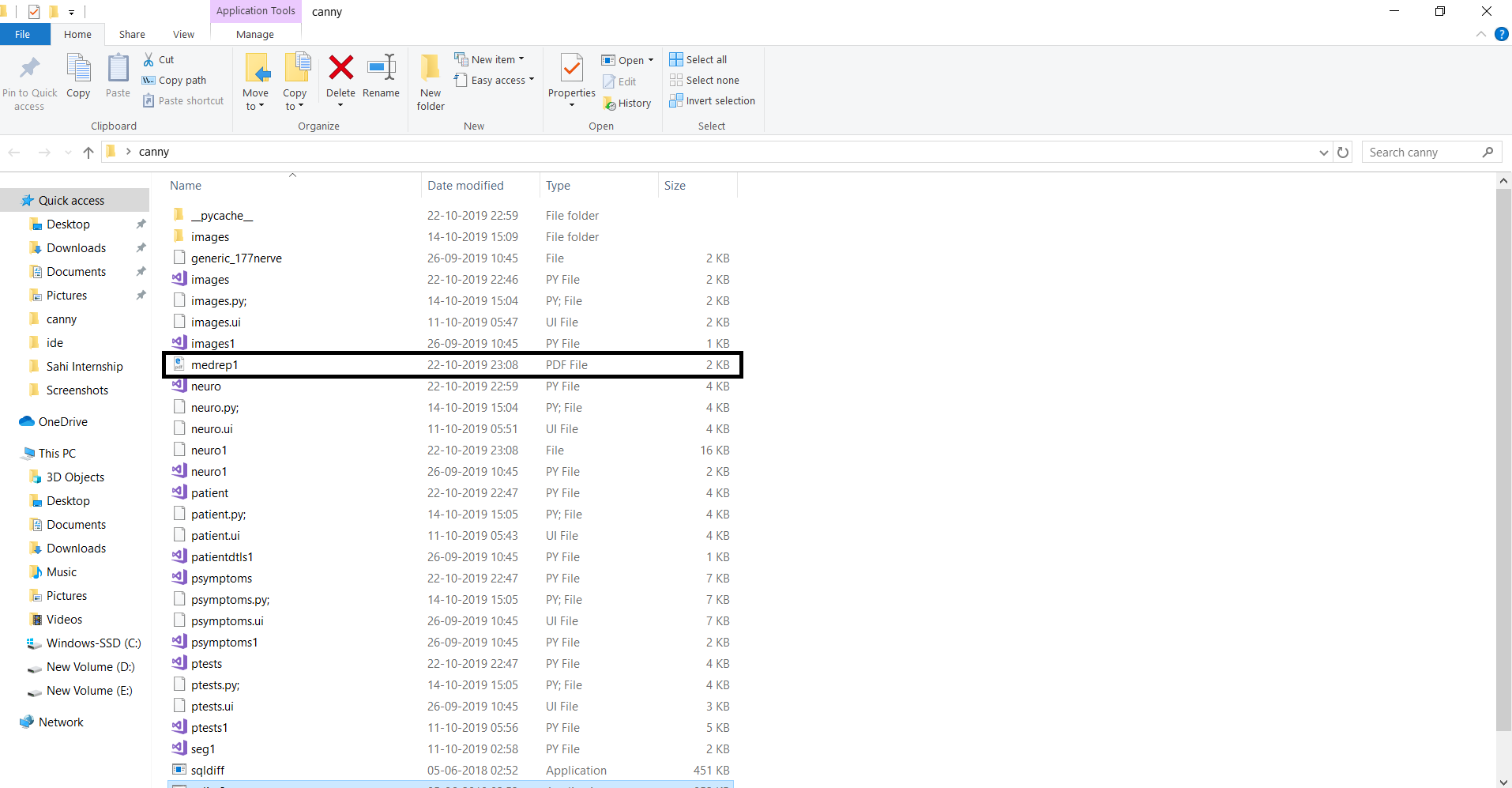


Fig 32: Generating the report

**Step 10:**

On clicking the medical report pdf, you can see the results.

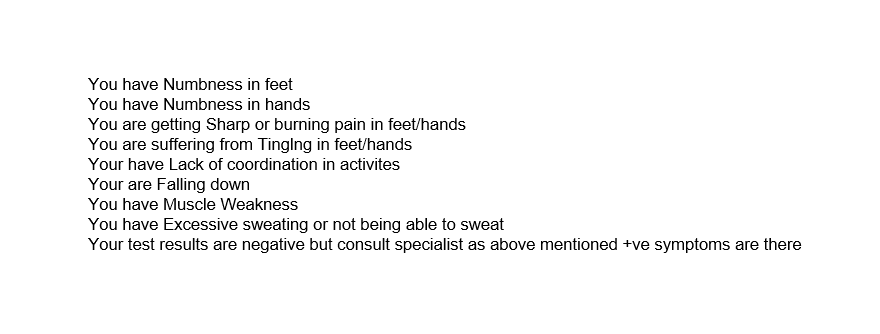


Fig 33: Report contents

**Step 11:**

On clicking the Fourth button in main GUI screen we get this Accept Image window. Write the image path and click on accept image to accept the image.

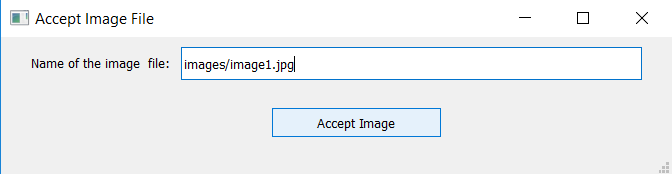


Fig 34: Accept Image File

Step 12:

On clicking the fifth button in main GUI screen we get this nerve detection window.

Here the nerves are detected using canny algorithm.

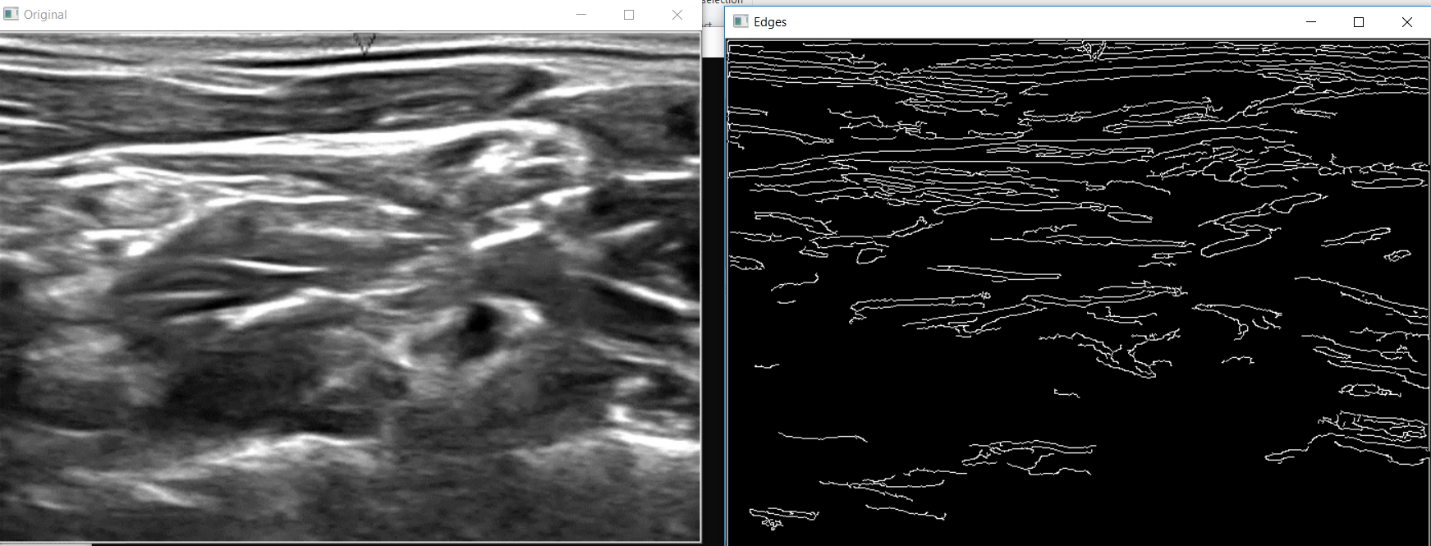


Fig 35: Nerve detection.

**CHAPTER 7**

**CONCLUSION AND FUTURE SCOPE**

**CONCLUSION**

This project entitled **“Nerve Segmentation in Ultra Sound Images”** is useful for doctors in identifying the correct location of the nerves. The project is also useful in analyzing the Ultra Sound images, which are difficult to analyze, because of low quality/noise in the image. This project finally leads to the improvement of quality of the patients life.

**FUTURE SCOPE**

As of now, the system is currently generating the analysis report, based on 10 symptoms and two test results. The system can be further extended to include all symptoms and test results.

**CHAPTER 8**

**REFERENCES**

1.Nerve Segmentation in Ultrasound Images by Vidushi Vashishtha Dr. Aju D International Conference on Innovations in Power and Advanced Computing Technologies

2. D.R. Ashbaugh, Quantitative-Qualitative Friction Ridge Analysis: An Introduction to Basic and Advanced Ridgeology,CRC Press

3. National Research Council of the National Academies, Strengthening Forensic Science in the United States: A Path Forward, National Academies Press,

4.https://www.python.org/

5.https://baoboa/pyqt5/blob/master/pyuic/uic/pyuic.py

6. <https://www.numpy.org/>

7.<https://riverbankcomputing.com/software/pyqt/intro>