

University of Essex

Galaxy Type Recognition

Final Report

Jack Smith

Student ID: 1504270

Supervisor: Alba Garcia

Second assessor: Adrian Clark

Course: BSc Computer Science

April 2018

Acknowledgements:

Alba Garcia: My project supervisor who provided feedback and opinions based on all material of the project and provided assistance towards the completion of my project

Professor Adrian Clark: Taught the CE316 module “Computer Vision” which provided me with the necessary aspects of computer vision and OpenCV to complete my project. Adrian also provided some guidance on the project, giving his opinions on how I should approach certain challenges.

Abstract:

There are billions of galaxies in the universe and none are the same! Most galaxies however will fall into 2 different general categories, Elliptical & Spiral. Studying these galaxies and classifying their type helps further our understanding of how the universe formed and how we came to exist in it.

My project was to create an application that uses a variety of computer vision techniques such as, canny edge detectors, contour tracing, and template matching to automatically classify images of galaxies into their specific type. Using a data set of images from the Sloan Sky Survey, the application uses the previously mentioned techniques to sort the galaxies into spirals and elliptical galaxies. The application can then display the results in a variety of different outputs such as the console, the user interface, or exported to a text file.

The application is written in Python[9] using OpenCV[8] and allows users to import files, then run a variety of computer vision techniques to extract features from the images, and to automatically classify the users selected images into their specific types. Some examples of feature extraction include Contour tracing and Template matching.

This application allows private or public Institutions/ Individuals who are working with images from the Sloan Sky Survey[7] to automatically classify galaxy images to an accuracy of 80% without having any other previous knowledge of the subject. This application will also allow users save time when it comes to classifying many galaxies as it's all done automatically meaning they can leave it running in the background while they work on other things.

Table of Contents:

1: List of Symbols

2: Main text

2.1: Context

I: What is the problem?

II: Why is the problem important?

III: Sustainability issues

IV: Legal Issues

V: Ethical Issues

VI: Relevant Intellectual Property

2.2: Project aims and objectives

I: Aims

II: Objectives

2.3 Technical Documentation

I: Technical Documentation Introduction

II: User Interface

A: The main application window

B: File Explorer

C: Application Widgets

1: Image Window

2: List Box

3: Buttons

4: Toolbars

5: Labels

6: Help menu

7: Application widgets conclusion

III: Classification

A: Histograms

B: Template Matching

C: Edge Detection

	D: Contouring
	E: K-Means
	F: Circle Detection
	G: Miscellaneous Classification Components
IV: Output/ Exporting Components	
	A: Accuracy Checkers
	B: Console Output
	C: Exporting to a text file
V: Class Diagram	
VI: Further Resources Worth Noting	
3: Project Planning	
	3.1: Introduction
	3.2: Project Planning Methodology
	3.3: Keeping Momentum
	3.4: Adapting to change
	3.5: Dealing with risks
	3.6: Reflection on the achievements made
	3.7: What I have learnt
	3.8: How suitable was the waterfall methodology
4: Conclusion	
5: References	

(1) List of Symbols:

- PIL: Python Imaging Library
- GUI: Graphical User Interface
- UI: User Interface
- CLI: Command Line Interface

(2)Main Text:

(2.1) Context:

(i) What is the problem?

My Project was to create an application that will take a dataset of images containing galaxies from the Sloan Sky Survey. (The Sloan Digital Sky Survey is a ground based optical telescope in New Mexico, United States. It operates in the visible spectrum, the infrared spectrum, and Ultraviolet <http://www.sdss.org>)[7]. It then processes these images and sorts them into their specific types. There are many different sub-types of galaxies, but I have chosen to sort them into the 2 main general categories of either Spiral or Elliptical.

To overcome this problem, I had to use a variety of computer vision techniques and application development techniques to produce a piece of software that would not only be an application that allows for user-friendly operation, but to be able to tie those application approaches into the computer vision algorithms that will classify the images.

Eventually I produced an application that can successfully classify galaxies within an average range of 70-80% and then output them to a user using the application to display the results on-screen, by exporting them to a text file, or by printing the results in the console. The method I used to classify the galaxies will be discussed in further detail later in the report.

(ii) Why is the problem important?

It is important to understand the difference between the galaxies types because it helps us to understand our place in the universe, what kind of galaxy we belong to, how neighbouring galaxies have impacted us over the years and in the future, and how we came to exist in our galaxy.

Classifying galaxies by their type also helps us to explain how the universe formed in the way that it has. This information can also provide us with some insight into how we came to be in this universe too. Looking back on this we can gain insight into life in the universe and whether some galaxies are potentially more hospitable to life similar to that of life on Earth.

Also creating an application that can automatically sort these galaxies will provide beneficial and time saving to scientific institutions across the world as manually classifying these images is a time-consuming process will require multiple peer assessments on images that are harder to classify than others. Computer vision also gives us an insight into these images that the human eye would never be capable of such as edge detection methods and average colour levels which can be used in the classification process.

Also developing this as an application will allow citizen scientists/ hobbyists or students to understand the process behind classifying galaxies and for them to understand how computer vision techniques allow us to peer into the universe like never before using algorithms that change the way we understand and see galaxies.

(iii) Sustainability issues:

Due to the nature of project and algorithms involved, sustainability might prove to be a minor issue due to the fact that once an algorithm has been proven to be accurate, it will always remain so regardless of newer features. However, should a newer algorithm prove to be more accurate than its predecessor, it should be implemented in such a way the application will use the newer, more accurate algorithm by default to ensure the highest overall accuracy when it comes to classifying these galaxies.

(iv) Legal Issues:

Due to the nature of the project being open-source and using royalty images provided by the Sloan Sky Survey. Legal issues should pose no immediate threat. However, if somebody were to use the application on images that are privately owned and then released said images to the world without accreditation to the original author would cause a breach in copyright laws.

If the application is used with images where the user has permission from the original author or uses open source/ royalty free assets for classification, legal issues are irrelevant.

(v) Ethical issues:

Due to the nature of the project, ethical issues are irrelevant.

(vi) Relevant Intellectual Property:

Whilst crowdsourced, Galaxy zoo proves to be a relevant intellectual property as it was the source of the Sloan Sky Survey images that have been peer assessed and pre-sorted by a team of citizen scientist by hand. My application aspires to achieve the same goals, but automatically instead of manually by a team of thousands.

(2.2) Project aims and objectives:

(i) Aims:

The aim of my project is to create an application that will automatically sort images of galaxies into their correct types. The 2 types I'm sorting the galaxies into are Spiral and Elliptical galaxies. This application then has to output the results to the user.

(ii) Objectives:

Below is a breakdown of the objectives that I had to accomplish in order to achieve the final aim of my project:

- Creating the interface/ GUI for the user
 - The first objective was to create a user interface that allows user to import their own images into the application for them to use. This included creating a list of all images that they had imported into the application, a file explorer interface that allowed them to select the files they wished to import into the application, and an image window that would display the users currently selected image.
- Classification
 - The next objective was to create the classification code that would process the image and analyse the image to assess whether it is a Spiral galaxy or an Elliptical galaxy. This step required a lot of trial and error to find the classification algorithm that was the most accurate.

- Accuracy testing
 - The third objective was to create a method that could perform mass accuracy testing on all images that the user imported, this would allow me and the users to assess how accurate the software is at assessing the performance of the application.
- Exporting
 - The final objective for my application was to create a method that would export the classification results to a text file that would allow the user to upload them to an external site or store them in a cloud service/ server.

(2.3) Technical Documentation:

(i) Technical documentation introduction:

For the technical documentation, I will be discussing all major and minor components that were implemented in the final release of the application. This will include; reasons as to why I implemented them, how I implemented them and how they work inside the application, do they work as intended, was there any modifications to the components throughout the lifetime of the implementation phase, and why these modifications were made.

Below I will discuss the 3 major components of my application and all their sub-components that go towards the finished, complete application.

(ii) User interface:

To start off with I developed the user interface that would allow the user to import their own files using a file explorer system instead of having to do everything command line. This makes it easier for citizen scientists and novice programmers to have a user-friendly experience with the application that doesn't come with the command line approach. To assist me with the design of the GUI, I had help from the online tutorial provided by Tkdocs [1]

(a) The main application window:

The application's interface was designed in Python 2.7 using the Tkinter framework with Python Imaging Library. The main window was constructed in its own class called "App" that takes an argument called "Frame" which will allow other functions to easily call this class to initialize functions using "Self" instead of having to create a multitude of class objects for the user interface. This helps to keep the application fast at runtime whilst calling the classification code later via objects only when they're needed.

Outside of the class, I have created a few variables that ensure the application window cannot be resized due to limitations with the GUI elements on smaller and larger window sizes. I've also set the class name to the root which allows me to call Tkinter functions without having to prefix "root." each time, I set the size of the window to 1200 pixels wide and 800 pixels high, set the background colour to a medium grey, instantiated some class objects to call classification code with. Set the application window to loop and check for input infinitely until the application has closed and then the root will be destroyed on close.

Inside the class, I have used python's "_init_" function which allows me to initialise all the Tkinter widgets. Inside this method, I've instantiated some global variables such as the list box variable which will be used to store all the image filenames imported by the user. This is because this variable needs to be accessed from anywhere in the class to allow the other functions to access the filenames from any location, so making it global achieves this.

```

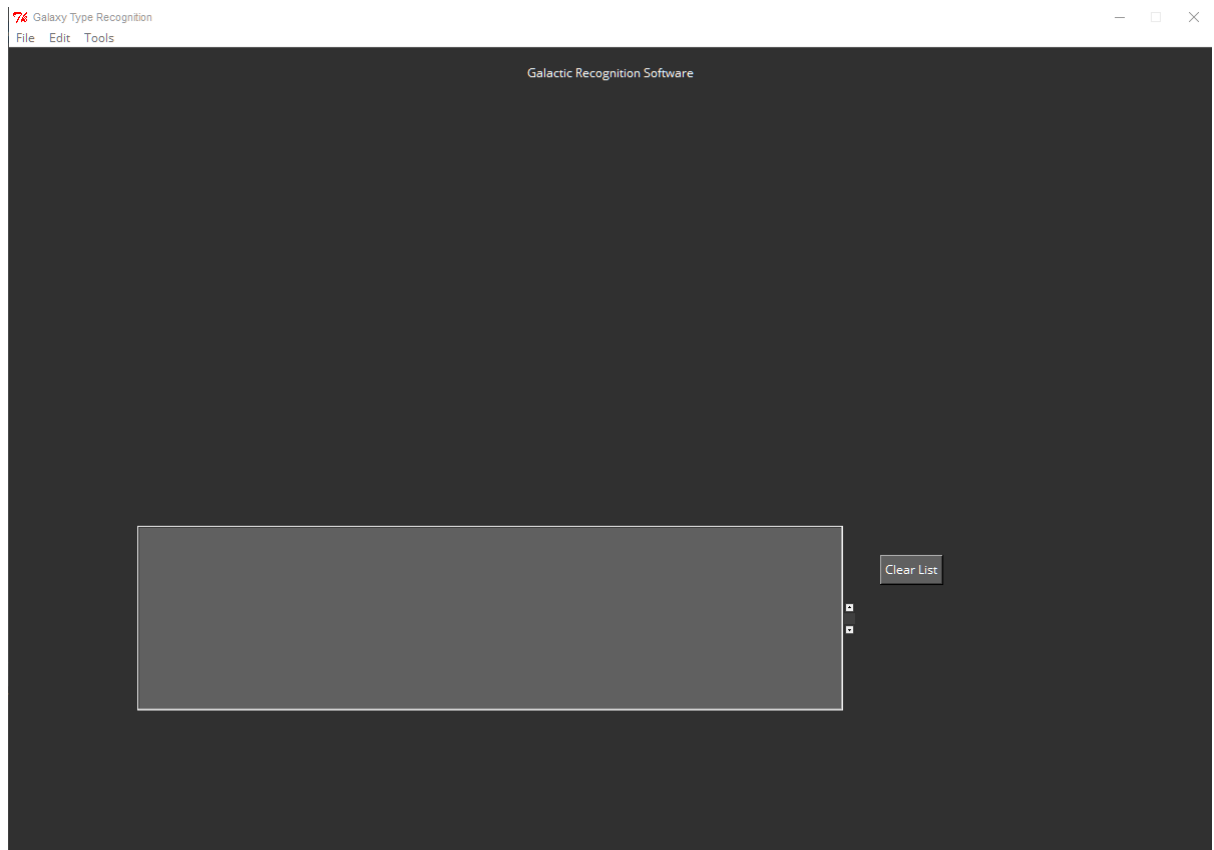
1.      # -----
2.      # Initialising frame
3.      # -----
4.      def __init__(self, master=None):
5.          Frame.__init__(self, master)
6.          self.pack(fill=BOTH, expand=1)
7.          global runonce
8.          runonce = 0
9.          global listbox
10.         listbox = Listbox(self, width=100, background="#666", fg="white")
11.         global fileList
12.         fileList = []
13.         global contourEstimateCount
14.         contourEstimateCount = 0
15.         self.createWidgets()
16.
17.
18.
19.     # -----
20. # Assigning name to frame
21. # -----
22. root = Tk()
23. root.resizable(False,False)
24. app = App(master=root)
25. root.minsize(width=1200, height=800)
26. Style().configure(root, background="#666")
27. classification = Tool.classification()
28. redundant = Tool.redundantClassification()
29. hist = Tool.histograms()
30. app.mainloop()
31. root.destroy()

```

(The initialisation method and program code to initialise the interface and its widgets)

Another global variable instantiated inside the initialisation is the file list which contains the same data as the list box variable, just as a default python list instead of the list box variable. This will be used for the mass accuracy testing method as it will allow it to iterate through each file name to check the accuracy of the classification results for each image and then provide an average.

Below is an image showing the application window. The focus of the image is the background and edges which shows the size of the window and then the background of the application, the actual content inside the window is created later.



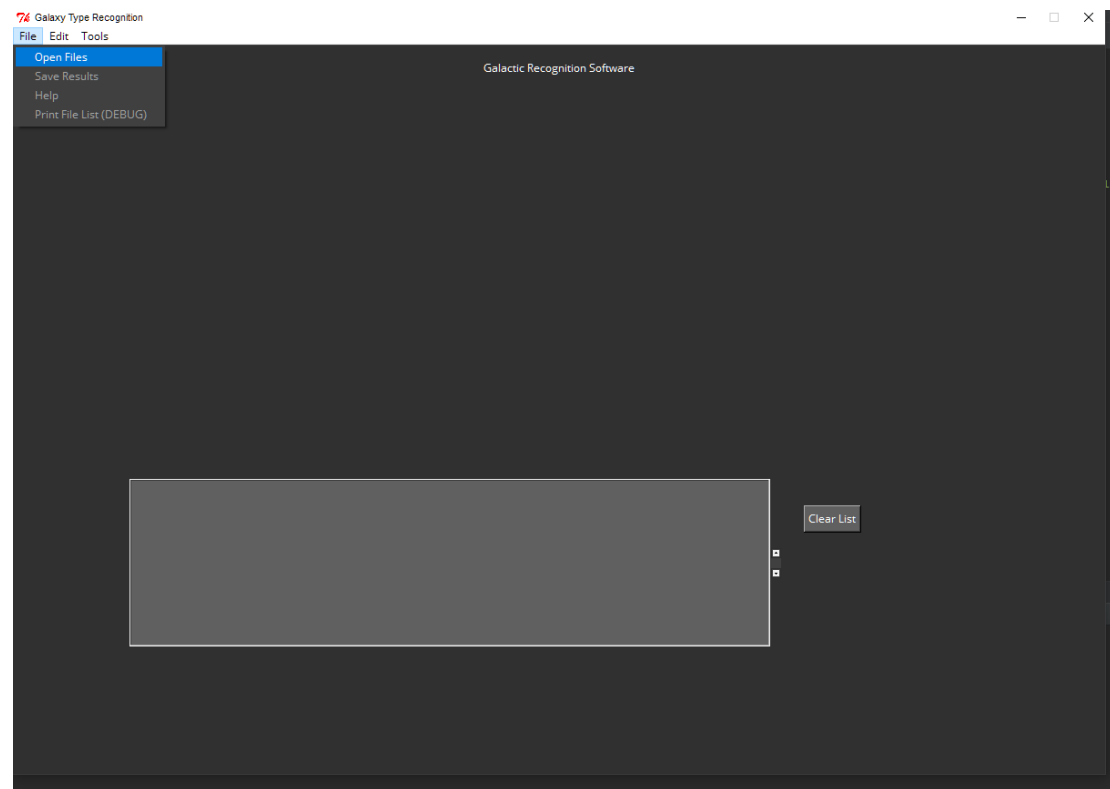
(Figure 1: Application window)

This component I believe works as intended as it's mainly just there to provide as a frame in which I can insert new elements into such as the list box, buttons, images, etc... It was modified throughout its development however, originally the window could be resized, but this was disabled as it would cause issues with the elements inside the window overlapping each other, removing this element all-together removes any chance of this happening. The background colour was also white, but I felt changing it to a dark grey made it more aesthetically pleasing.

(b) File Explorer:

The applications file explorer allows users to select files to import into the application using windows' default file exploring window. I feel this is more aesthetically pleasing than using a command line approach to import files as novice users or citizen scientists may not necessarily know how to operate the command line. Giving them access to a GUI approach will ensure that these users are accounted for and for experienced users, this is just considered standard giving the application a more polished feel.

I created a method called “open” which is called when the user clicks on “Open files” in the “File” tool bar.



(Figure 2: Open File Button Location)

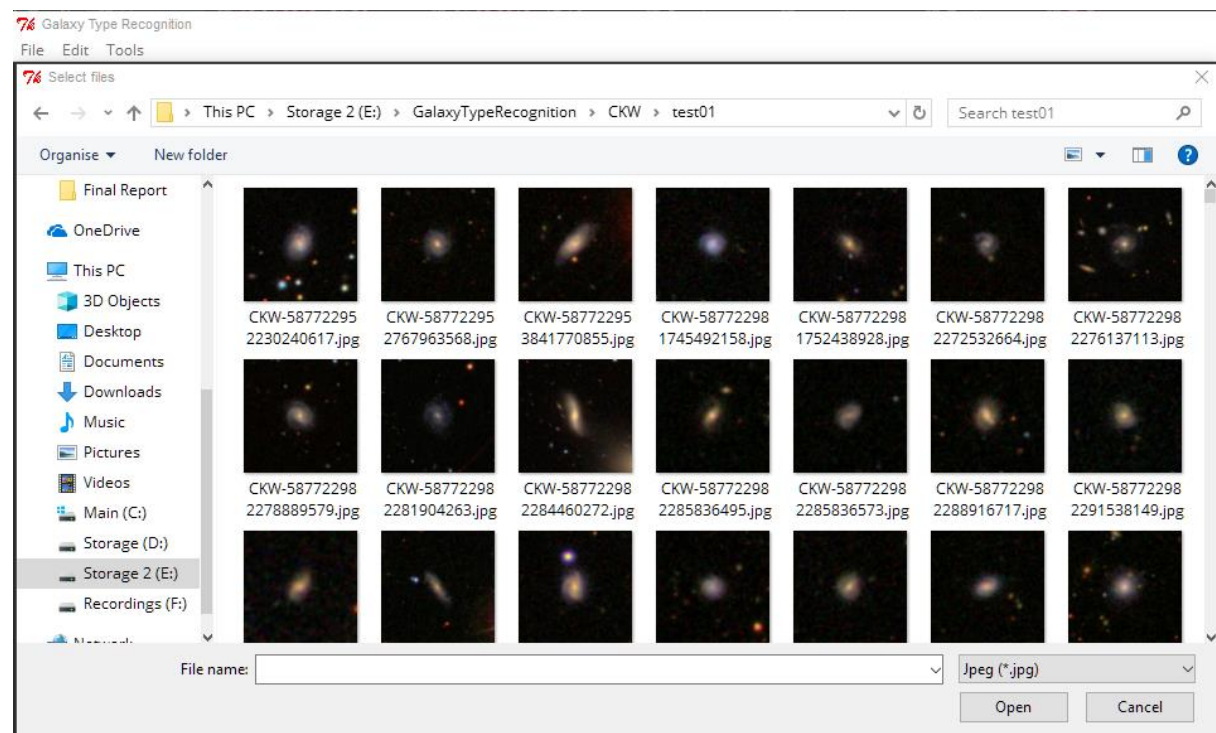
This method works by opening a Tkinter file dialogue which allows users to import jpg, png, or other files. The default format is JPG as this will remove most other clutter such as text documents. The selected files are then stored in a list called file. I then create a new list called “FileName”. I then append the filenames imported from the file explorer into this new list and then into the global “Filelist” variable so I can call the filenames individually. The first filename in the list is set to the default image so the user knows that their image import was successful, and it will insert all filenames into the list box too.

```
1. def open(self):
2.     file = tkFileDialog.askopenfilenames(parent=self, title='Select files', defaultextension=".jpg", filetypes=(("Jpeg", "*.jpg"), ("Png", "*.png"), ("All files", "*.*")))
3.
4.     fileName = []
5.
6.     for i in self.tk.splitlist(file):
7.         fileName.append(i)
8.         fileList.append(i)
9.
10.    if file != None:
11.        self.displayIm(fileName[0])
12.        self.currentFile(fileName[0])
13.        for item in self.tk.splitlist(file):
14.            listBox.insert(END, item)
```

(The open function)

This method works as intended as it allows users to import images using the graphical interface and then scan through their imported files in a list format instead of having to use a command line interface which will be more time consuming than the command line approach. This component had no major modifications.

Below is an image highlighting the file explorer window along with a preview of some of the images that the user might import into the application. Note that the file explorer is similar to the default Windows file explorer, this is due to Tkinter implementing a “file dialogue” that is similar to that implemented in Windows 7 and above.



(Figure 3: File Explorer)

(c) Application widgets:

The interface widgets are the individual elements that you will find on the window such as buttons, images, labels, and list boxes. These are the filling of the interface that are used to display the information being passed through the application. These are vital to the interface, without them the interface has no reason to exist. I decided to keep the widgets simple and clean to allow for a smooth experience using the application which is “Clutter-free”. Once again, I followed the tutorial available from Tkdocs [1] to provide me information how each widget works. I will break down this sub-component into micro-components to allow for a better explanation on how they each work.

(1) Image window:

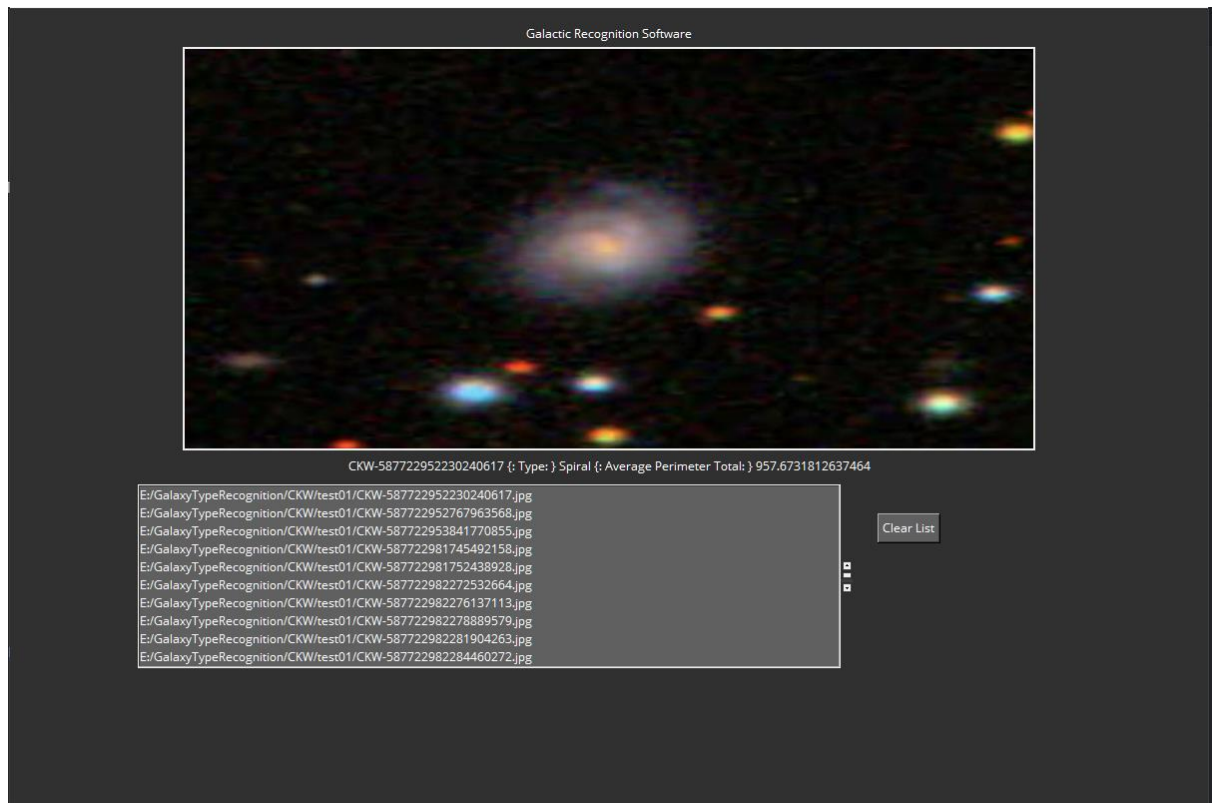
The image window is used to display the users currently selected image, this is used to give a visual representation of the selected image and provide the user with a preview should they wish to scan through the list visually instead of simply going by the file names.

This works by using Python imaging library which can display images inside Tkinter using a label. Instead of using the label for text, you pass an image argument with the image name. When the user double clicks on an item in the list box or presses the return key on an item inside the list box, the “changeIm” function is called which takes the file name as an argument and then gets converted to the Tkinter image format and calls a second function called “displayIm”. This second function takes the Tkinter file and opens it using PIL, resizes the image, converts the filename to actual image data, and assigns this image to the label which will display the new image. This function also calls the default classification function on the selected image and displays the results in another label located underneath the image.

```
1. def changeIm(self,im):
2.
3.     newIm = im.widget
4.     sel = newIm.curselection()
5.     value = newIm.get(sel[0])
6.     label3.config(text="")
7.
8.     self.displayIm(value)
9.
10.
11.
12.     #self.init(value)
13.
14. # -----
15. # Changes the image to the selected image in the display box
16. # -----
17. def displayIm(self, file):
18.
19.     self.currentFile(file)
20.
21.     im = Image.open(file)
22.     im = im.resize(size, Image.ANTIALIAS)
23.     im2 = ImageTk.PhotoImage(im)
24.
25.
26.     label2 = Label(self, image=im2)
27.     label2.image = im2
28.     label2.place(relx=.5, rely=.3, anchor="center")
29.     global label3
30.     label3 = Label(self, text=self.detect(), background="#333", fg="white")
31.     label3.place(relx=.5, rely=.571, anchor="center")
```

Overall this component works as intended as it successfully displays the users currently selected image. Throughout the implementation phase, this component has been updated. I implemented the default classification function after the initial implementation of the display window to allow the classification code to be run on the current image. I’ve also changed aesthetic features such as the border colour to match the background and the location of the image window to give it more space between the other elements.

Below is an image of the display window with an example galaxy image loaded as the selected image.



(Figure 4: Image window)

(2) List Box

The list box is used to display all the files that the user has imported into the application from the file manager. The list box can also be used to select which image to run the classification code on by double clicking or pressing the return key on the highlighted item in the list box. I decided that a list box would be best for this as it separates all the imported files into individual file names which the user can use to select/ browse through the set of imported data.

Setting the list box up was simple, it's packed onto the frame to make it visible, and then I bound the double left click action and return actions to call the function "changelm" which was previously mentioned in the image box component.

```
1. listbox.place(relx=.4, rely=.71, anchor="center")
2. scroll['command'] = listbox.yview
3. scroll.place(relx=.698, rely=.71, anchor="center")
4. listbox.bind("<Double-Button-1>", self.changeIm)
5. listbox.bind("<Return>", self.changeIm)
```

I believe that this component works as intended as it allows the user to browse through the list box to see their set of imported images and select individual images from that list to set as their active image. There was a couple of modifications that were made, such as adding the return key as an action which I thought would be easier to go through the images one-by-one instead of left clicking all the time. I also modified it by adding a small scroll bar but couldn't get the sizing to work correctly on it.

Below is an image of the list box and scrollbar with a sample set of images imported.

```
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722952230240617.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722952767963568.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722953841770855.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722981745492158.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722981752438928.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982272532664.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982276137113.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982278889579.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982281904263.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982284460272.jpg
```

(Figure 5: Listbox)

(3) Buttons

In the end, I opted only to have one button on the user interface, however there used to be buttons for all the major tool bar items. The one button I left behind was the “clear list” button as I felt it made it easier for debugging items and to let users delete the current list and start again without having to relaunch the application. The reason I removed most of the buttons from the application was to give it a more aesthetic feel as well as a cleaner, less cluttered look. I will include before and after screenshots too.

The buttons are also quite simple to implement. You simply create the button object and set the command of the object to the function you wish to call, in my case the function was the “clearList” function. You then pack the button onto the frame and place it where you’d like it to be.

```
1. clear = Button(self, text="Clear List", command=self.clearList, background="#666", fg="white")
2. clear.pack()
3. clear.place(relx=.75, rely=.65, anchor="center")
```

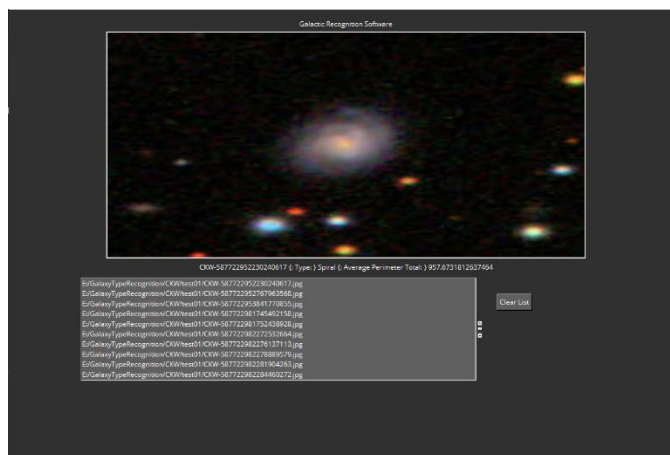
The clear list function which it calls is used to remove all objects from the list box component, so the user can start again without having to restart the application. The clear list function works by deleting all items in the list box starting at position 0 and ending as the “END” position of the list. I then also do the same for the file list global variable and clear label 3 which is the label that displays the classification details.

```
1. def clearList(self):
2.     listbox.delete(0,END)
3.     fileList[:] = []
4.
5.     label3.config(text="")
```

I believe that this function partially works. Personally, I’m not overly fond of this feature as I feel it looks a bit tacky and outdated. I would redesign the buttons to be cleaner and more uniform and remove them as they currently are. However, for ease of access and debugging purposes, I’m willing to leave the clear list button on the interface for the mean time and I’m considering adding the “Open files” button back in to streamline the process even further. As previously mentioned, this component has undergone a vast change in the form of moving most of the buttons into the tool bar at the top of the application. Attached below are the before and after shots.



(Figure 6: Buttons before adding toolbar)



(Figure 7: Buttons after adding toolbar)

(4) Toolbar

The toolbar component was implemented to allow for a clean and streamlined approach to function calling. Instead of having buttons all over the application, I decided to move most of the callable function buttons to the tool bar as it makes for easier selection of the applications processes. For example, the open files button was moved to the “File” toolbar instead of being displayed on screen. As previously mentioned, I believe this gives the application a cleaner look keeping focus on the elements of the interface that matter.

Whilst simple to implement, it does make the code look quite cluttered for the toolbars. This is because each element in the toolbar requires its own entry. However, this is still cleaner and more efficient than buttons as its 1/3 of the code that would be required to implement the buttons. Also, due to the modularity of toolbars, it’s easier to add and remove features on them without having to make space for new buttons so you can implement a lot more classification functions than you could have with buttons.

To implement the toolbars, you first need to create the Menu Bar which is attached to the root element being the frame of the application. You can then add menus to the menu bar and add commands to them which are the elements you wish to add to the drop-down list. Each menu has its own dropdown bar which can be used to access the individual function calls. These menus are then packed onto the menu bar and given a label that provides a brief description of what the menu list contains. For example, the “File” menu. Finally, you configure the roots menu to the name of you tool bar.

```
menubar = Menu(root)

fileMenu = Menu(menubar, tearoff=0)
fileMenu.add_command(label="Open Files", command=self.open)
fileMenu.add_command(label="Save Results", command=self.save)
fileMenu.add_command(label="Help", command=self.help)
fileMenu.add_command(label="Print File List (DEBUG)", command=self.printFileList)

editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Clear List", command=self.clearList)
editmenu.add_command(label="Check Spiral Accuracy", command=self.checkSpiral)
editmenu.add_command(label="Check Elliptical Accuracy", command=self.checkElliptical)
editmenu.add_command(label="Check Spiral Accuracy (K-Means)", command=self.checkSpiralK)
editmenu.add_command(label="Check Elliptical Accuracy (K-Means)", command=self.checkEllipticalK)
editmenu.add_command(label="Close application", command=sys.exit)

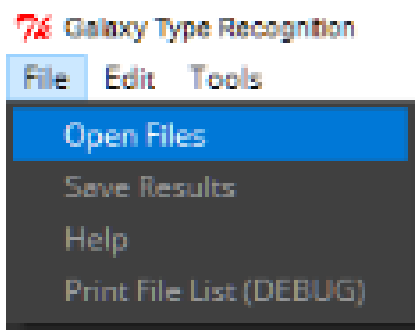
toolsmenu = Menu(menubar, tearoff=0)
toolsmenu.add_command(label="Generate Histogram(Current Image)", command=self.init)
toolsmenu.add_command(label="Generate Colour Histogram(Current Image)", command=self.colourHist)
toolsmenu.add_command(label="Generate Default Classification Details(Current Image)", command=self.detect)
toolsmenu.add_separator()
toolsmenu.add_command(label="Perform Contour Tracing On Current Image (Colour)", command=self.contourColour)
toolsmenu.add_command(label="Perform Template Matching On Current Image (Colour)", command=self.templateMatch)
toolsmenu.add_separator()
toolsmenu.add_command(label="Perform Edge Detection On Current Image", command=self.cannyEdge)
toolsmenu.add_command(label="Fill Edge Detection On Current Image", command=self.fillCanny)
toolsmenu.add_command(label="Perform Edge Detection Inside Template Match On Current Image", command=self.cannyInsideTemplate)
toolsmenu.add_command(label="Count Number Of Contours Inside Template Area On Current Image", command=self.contourCountTemplateArea)
toolsmenu.add_command(label="Estimate Average Contour Count", command=self.contourEstimate)
toolsmenu.add_command(label="Classify Galaxy Based On Average Contour Count", command=self.classifyOverAverage)

toolsmenu.add_separator()
toolsmenu.add_command(label="Perform Circle Detection on current image", command=self.houghCircle)
toolsmenu.add_command(label="Perform HOG current image", command=self.hog)
toolsmenu.add_separator()
toolsmenu.add_command(label="Perform K-Means Quantization on image", command=self.kmeans)
toolsmenu.add_command(label="Contour K-Means image", command=self.contourK)
toolsmenu.add_command(label="Calculate average colour of K-means", command=self.kColourMean)

menubar.add_cascade(label="File", menu=fileMenu)
menubar.add_cascade(label="Edit", menu=editmenu)
menubar.add_cascade(label="Tools", menu=toolsmenu)

root.config(menu=menubar)
```

(Figure 8: Toolbar code)



(Figure 9: Toolbar Options)

Personally, I feel the toolbars achieve their job of making the application feel a lot cleaner than the buttons. They're more uniform to the application and give a more professional feel than the buttons did. They also make it easier to implement modularity into the application by being able to add and remove classification features without having to modify the existing structure of the interface. Due to the nature of the toolbars modularity, they were constantly modified throughout development when adding and removing classification methods.

(5) Labels:

The labels in the interface are there to provide text information about processes or the application itself. For example, the title label is used to provide the name of the application and the results label “Label3” is used to display information about the classification results to the user.

The implementation for the labels was straight forward and simple as they’re mainly just text information that must be displayed. The results label was a bit more complex as it required some string formatting to display in the desired format. To create the label, I call the default detect function which returns a string with the short file name (Without the path to the file), the galaxy type, and the average perimeter total for that image.


```
1. global label3
2. label3 = Label(self, text=self.detect(), background="#333", fg="white")
3. label3.place(relx=.5, rely=.571, anchor="center")
```

(Result label instantiation and initialisation)

```
1. filenameShort = os.path.splitext(os.path.split(filename)[1])[0]
2.
3. returnString = filenameShort, ": |Type|: ", type, ": |Average Perimeter Total|: ", total
4.
5. file.write("\n"+str(returnString))
```

(Fragment of the detect function which formats the string to the desired structure.)

This component works as intended as it successfully displays results on the interface. However, there is an issue with the format of the string, it shows curly braces around the actual string content, this is due to a limitation with the label string format as it separates strings from variables with a comma instead of the + symbol which makes python treat the string as an object. There were no major modifications to this component throughout development.



```
CKW-587722982278889579 {: Type: } Spiral {: Average Perimeter Total: } 1030.9503401606537
```

(Figure 10: Classification Label)

(6) Help Menu

The help menu is a separate pop-up window from the user interface that provides the user with some general advice on how to operate the application. Due to time constraints and priority's, this feature was mostly neglected throughout development meaning it's very basic and outdated. I decided to keep the feature in however as it still does provide some useful information on how to navigate around the interface.

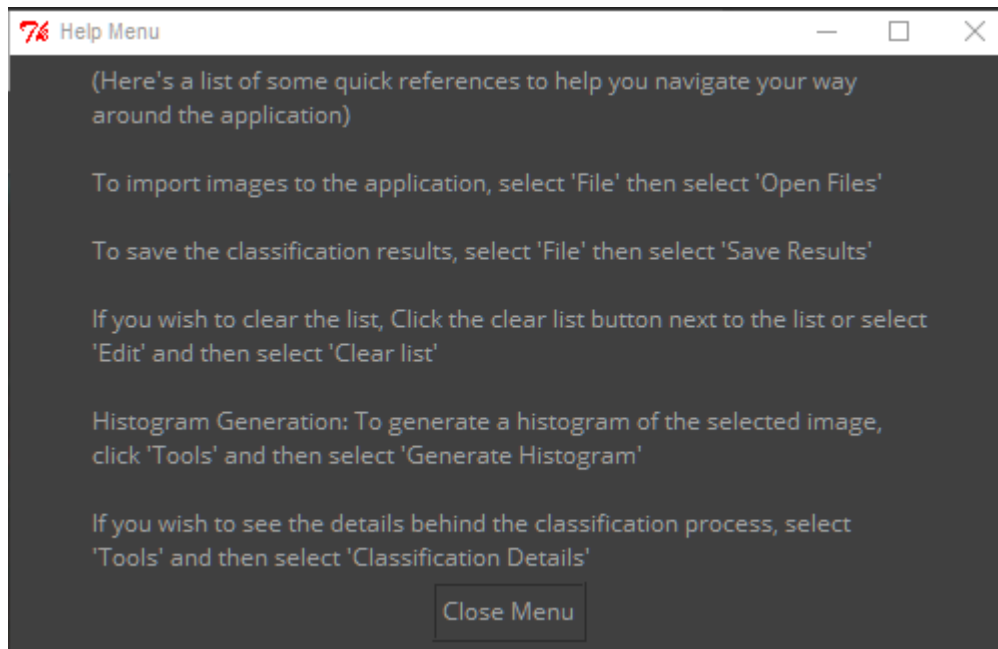
The implementation for the help menu was straight forward, I had to follow the tutorial from Tkdocs [1] to be able to implement a pop-up window but most of the other content is straightforward inside the pop-up. It works a separate function that is called from the help option in the file toolbar menu. It creates a new window on the top level which is 500 pixels wide and 300 high. the message is then displayed on this window

which contains some basic information on how to navigate around the application. When the user presses the Close button in the top right. The window is destroyed.

```
1. def help(self):
2.     top = Toplevel()
3.     top.title("Help Menu")
4.     top.geometry('500x300')
5.
6.     msg = Message(top, text="(Here's a list of some quick references to help
    you navigate your way around the application)"
7.         "\n\nTo import images to the application, select
    'File' then select 'Open Files'"
8.         "\n\nTo save the classification results, select
    'File' then select 'Save Results'"
9.         "\n\nIf you wish to clear the list, Click the cl
    ear list button next to the list or select 'Edit' and then select 'Clear lis
    t'"
10.        "\n\nHistogram Generation: To generate a histogr
    am of the selected image, click 'Tools' and then select 'Generate Histogram'
    "
11.        "\n\nIf you wish to see the details behind the c
    lassification process, select 'Tools' and then select 'Classification Detail
    s'")
12.     msg.pack()
13.
14.     exButton = Button(top, text="Close Menu", command=top.destroy)
15.     exButton.pack()
```

Personally, I feel this component could have done with some updating before the final release, but it took a back seat due to the classification components taking priority over this one. However, in its current state it still does offer some use to the user by providing some basic insight on how to navigate/ operate the applications interface. This component had no major modifications throughout development due to other components taking priority.

Below is a screenshot showing the help screen. I opted to use the dark theme again for this window to keep continuity between the 2 interfaces and for aesthetic reasons.



(Figure 11: Help menu window)

(7) Application widgets conclusion:

Overall, I feel that all the widgets come together to create a complete, functional user interface that is aesthetically pleasing. The widgets not only provide the user with feedback on how the application is working, and what the application is working on but provide the user with a clean way to see this data represented in a graphical format that is more user-friendly than a command line interface. If I could change anything, I would ensure that the help window is brought up to date with current features to provide the user with more insight into how each option works and what it does, and I would add another button on the interface to allow the user to quickly open new files instead of having to go into the toolbar to select new images.

(iii) Classification:

Next, I will discuss the classification technical information. This section will be discussing the individual components of the classification methods, how these components worked, whether the components work as intended, the modifications made to these components and why these modifications were made, and planned components that were never actually implemented and why they were not implemented.

This section has utilised a lot of information from resources such as a paper called the “Morphological classification of galaxies into spirals and non-spirals” [2] Where they discuss some feature extraction methods letting me of certain features of each galaxy to look out for. Another great source of information was the “Galaxy classification using pattern recognition methods” paper by Harvard University [3] which discusses template matching.

(A) Histograms:

To start off my basic classification, I created some grey level histograms and RGB colour histograms to discover the deviation in the various grey & colour levels of each image. The idea to use histograms came from Professor Adrian Clarke’s notes [4] where he discusses that histograms are the simplest way to characterize an image to discover if it’s over/ under exposed.

Although this feature was not used much after the more complex classification methods were implemented, they show some interesting information into the average colour distribution and exposure of the image, so I decided that they should be kept in.

The implementation of this feature spans across 2 different functions, the first called “Init”, converts the image to a CV2 image so it can be edited. I then take the number of columns, rows, and colour channels. The cv2 image is then passed through the “Hist” function which sets the range of the colour from 0 to 256 and then for each column and row and c being in range of 0 to 256, it will combine all colour channels and divide them by 3 to get the average grey level of the image being the exposure. This result is then plotted in the histogram.

```
1. def init(self):
2.     image = cv2.imread(filename)
3.
4.     global ny, nx, nc
5.     ny, nx, nc = image.shape
6.     x,h = self.hist(image)
7.     self.plotHist(x,h,filename)
```

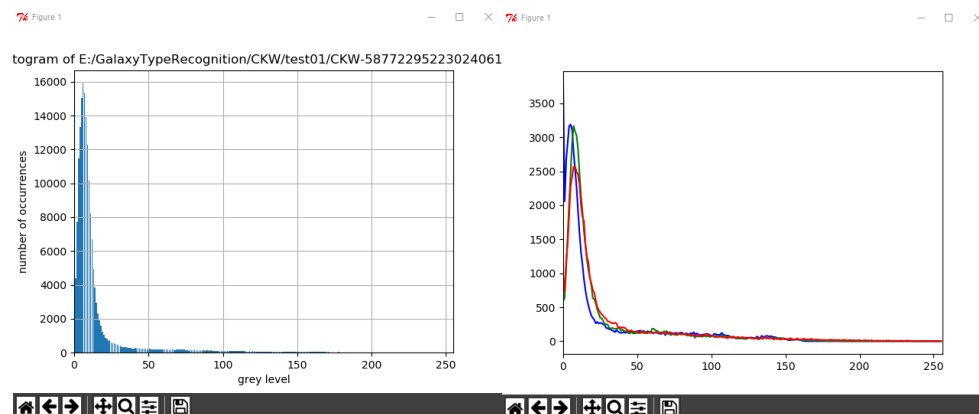
```
1. def hist(self, im):
2.
3.     maxgrey = 256
4.
5.     # ab = numpy.ndarray(maxgrey)
6.     # for i in range(0, maxgrey):
7.     #     ab[i] = i
8.     ab = range(0, 256)
9.
10.    h = np.zeros(maxgrey)
11.    for y in range(0, ny):
12.        for x in range(0, nx):
13.            greyLevel = 0.0
14.            for c in range(0, nc):
15.                # v = im[y, x, c]
16.                greyLevel += im[y, x, c]
17.                # h[v] += 1
18.            greyLevel /= 3
19.            h[int(greyLevel)] += 1
20.
21.    return ab, h
```

The colour histogram works by using 3 characters to assign the colours of blue, green, and red. Then for the enumeration of colour, it will plot each colours distribution on the histogram with a range of 0 to 256 and plot this colour to the chart.

```
1. def colourHist(self, im):
2.
3.     colour = ('b', 'g', 'r')
4.     for i, col in enumerate(colour):
5.         hist = cv2.calcHist([im], [i], None, [256], [0, 256])
6.         plt.plot(hist, color=col)
7.         plt.xlim([0, 256])
8.     plt.show()
```

Overall, I feel this module functions as intended as it successfully displays the average grey level histogram and colour histogram providing the user with an average idea of the exposure of the image and the colour distribution of the selected image. This component was modified throughout the implementation phase which was the addition of the colour histogram, initially this component only consisted of the grey level histogram.

Below I've attached 2 images highlighting the grey level histogram first and then the colour histogram. The grey level shows that the image is under-exposed. However, this is due to the large quantity of black in the image of the surrounding space next to the galaxy. The colour histogram shows that the blues and greens are the most common colour for that image.



(Figures 12 & 13: Grey-level histogram (Left) & Colour histogram(Right))

(B) Template Matching:

As P.Burda and J.V.Feitzinger has great success with pattern recognition in their project where they classified galaxies using pattern recognition techniques [3], I opted to take a similar approach using the template matching feature implemented in OpenCV. The feature works by using an example image that I know to be a galaxy (the template) and then use this in OpenCV's template matching feature on the users selected image to see if it could find a comparison between the 2 images. If it finds that the 2 images are similar enough in features, it will have matched the template. I've also used the template matching method to crop the images down as to only focus on the key point of the image instead of grabbing all the dark space around the galaxy. This will help to increase the accuracy of the results by remove a lot of the artefacts that exist in the image that could be detected by the various edge detection and contouring components.

The template matching components is called by a variety of other classification methods as it's also used to crop the image down to the centre to remove irrelevant areas of the image. It works by taking the users selected image as the argument, it then converts the original image to grey, it then will use OpenCV's matchTemplate function to match the grey image to the template provided. If it finds a match it will draw a box around the galaxy in the image showing that the template has been found. It will then crop the image down to the centre as all the images have the focus galaxy in the middle, (The cropping still happens to an image that has not matched). The cropped image is then returned.

```
1. def selectTemplate(self, im):
2.     gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
3.     template = cv2.imread("template2.png", 0)
4.     template2 = cv2.imread("template3.png", 0)
```

```

5.     template3 = cv2.imread("template4.png", 0)
6.     w, h = template.shape[::-1]
7.     w2, h2 = template2.shape[::-1]
8.     w3, h3 = template3.shape[::-1]
9.     res = cv2.matchTemplate(gray, template, cv2.TM_CCOEFF_NORMED)
10.    res2 = cv2.matchTemplate(gray, template2, cv2.TM_CCOEFF_NORMED)
11.    res3 = cv2.matchTemplate(gray, template3, cv2.TM_CCOEFF_NORMED)
12.
13.    threshold = 0.9
14.    loc = np.where(res >= threshold)
15.    loc2 = np.where(res2 >= threshold)
16.    loc3 = np.where(res3 >= threshold)
17.    x = 0
18.    y = 0
19.    x2 = 0
20.    y2 = 0
21.    x3 = 0
22.    y3 = 0
23.
24.    isGalaxy = 0
25.    for pt in zip(*loc[::-1]):
26.        isGalaxy = 1
27.        x = pt[0]
28.        y = pt[1]
29.
30.    for pt2 in zip(*loc2[::-1]):
31.        isGalaxy = 2
32.        x2 = pt2[0]
33.        y2 = pt2[1]
34.
35.    for pt3 in zip(*loc3[::-1]):
36.        isGalaxy = 3
37.        x3 = pt3[0]
38.        y3 = pt3[1]
39.
40.    if isGalaxy == 0:
41.        print "No matching template, Assumption made that galaxy takes centre frame
42.        "
43.        return im[96:130 + h, 90:90 + w]
44.
45.    if isGalaxy == 1:
46.        print "Galaxy template 1"
47.        return im[96:130 + h, 90:90 + w]
48.
49.        #return im[y:y + h, x:x + w]
50.
51.    if isGalaxy == 2:
52.        print "Galaxy template 2"
53.        return im[96:130 + h, 90:90 + w]
54.
55.        #return im[y2:y2 + h, x2:x2 + w]
56.
57.    if isGalaxy == 3:
58.        print "Galaxy template 3"
59.        return im[96:130 + h, 90:90 + w]
60.
61.        #return im[y3:y3 + h, x3:x3 + w]

```

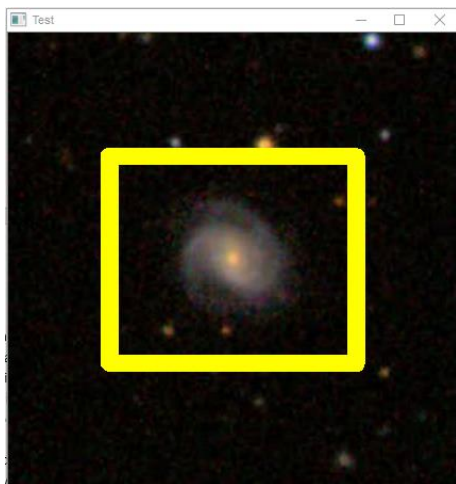
Overall, I think the template matching didn't work the way I wished, as it would be too broad in its matching, even matching galaxies of the opposite type provided. However, it was useful in teaching me to crop the image down to the centre to get the focus of the image instead of grabbing the unnecessary content around the edges of the image. This method has had a variety

of modifications throughout development. Initially, the component wouldn't crop the image down to the centre, this was then applied to images that had a template match and was then provided for images that didn't match a template too, and this is due to all images having the focus galaxy in the centre frame.

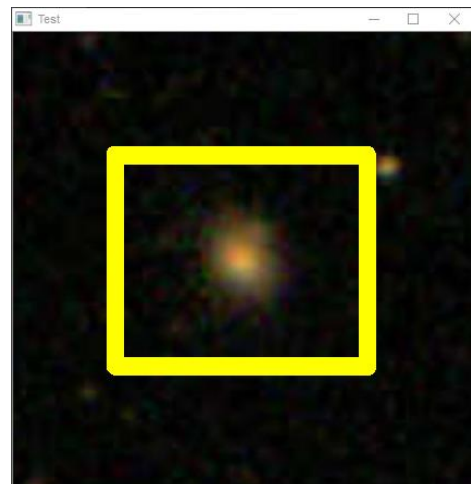
Below is an image showing an example of the template matching on a randomly selected spiral galaxy, it successfully matches a template, but it will also match an elliptical galaxy using the same template.



(Figure 14: The template that will be used to check for similarities.)



(Figure 15: Spiral Template Match)



(Figure 16: Elliptical Template Match)

(C) Edge Detection:

As mentioned by D.S.Dhami in the morphological classification of galaxies paper[2], the different galaxies have a variety of different features that can be used to aid in the classification process. To make a start on the feature extraction, I implemented the basic edge detection component that allows the application to process the images edges by finding sharp contrasts in colour. This feature is useful as it shows the average edge distribution in the image which allows the user to see the shape difference between spiral and elliptical in more detail. Spirals being a more open distribution of edges compared the more closed nature of the edges in the elliptical galaxies.



(Figure 17: Spiral Edge Detection)



(Figure 18: Elliptical Edge Detection)

The edge detection components have a couple of different implementation methods spread across a couple functions, one's takes the file name of the currently selected image and another takes a pre-processed image and runs the edge detection component on that method and returns that image to be used further in other classification components. The CannyEdge method is takes the currently selected file and converts it to an OpenCV image, blurs the image via a median blur to remove most of the noise from the image and then uses OpenCV's denoising algorithm to remove the rest of the noise in the image. The edge detection is then applied in the form of a canny edge detector implemented by OpenCV which works by detecting sharp colour contrasts in the image. The CannyEdgeArg function is the same except for that fact that it returns the image instead of displaying it.

There is also another method that runs the canny edge detector inside the cropped image previously mentioned. This function called "CannyInTemplate" takes the users selected image, passes it through the selectTemplate function, then through the cannyEdgeArg function and displays the result.

```
1. def cannyEdge(self, filename):
2.     im = cv2.imread(filename)
3.     im = cv2.medianBlur(im, 5)
4.     im = cv2.fastNlMeansDenoisingColored(im, None, 10, 10, 7, 21)
5.     edge = cv2.Canny(im, 50, 50)
6.     cv2.imshow('Canny Edge Detection', edge)
```



```

7.     cv2.waitKey(0)
8.
9.     def cannyEdgeArg(self,im):
10.         im = cv2.medianBlur(im, 5)
11.         im = cv2.fastNlMeansDenoisingColored(im, None, 10, 10, 7, 21)
12.         edge = cv2.Canny(im, 50, 50)
13.         return edge

```

Overall, I feel this component functions as intended as it accurately identifies the edges of the image without capturing any noise in the image. The edge also works as a feature extraction method as it can be used to accurately display graphical information about the edge distribution of both spiral and elliptical galaxies showing average shape patterns between them. This method has been modified throughout development by the addition of the CannyEdgeArg function which returns the result for further classification methods. The denoising algorithm was also tweaked to provide the most accurate denoising of the image without removing too much of the detail. Another modification was the addition of the CannyInTemplate function which allows the user to run the canny edge detection on the cropped image to remove further unwanted features of the image.

Above, you can find some images showing the results of the edge detection methods for both spiral galaxies and elliptical galaxies. These results can either be used as a visual representation or passed through further classification components for a more accurate result, these methods will be discussed in further detail later.

(D) Contouring:

Carrying on with the success that P.Burda and J.V.Feitzinger had with pattern recognition[3], the next component is the contouring function which will use the arcLength OpenCV function to draw lines around the edges of the image, if used in conjunction with the edge detecting method it can be used to accurately classify galaxies. This component has been used to contour the edges found by the canny edge detector, these contours can then be used to calculate the average perimeter length of all contours which can then in turn be used to classify the galaxy based on the overall average perimeter length of all contours in the image. The contouring has also been used for a graphical representation of the edges in the image overlaid with the original image. This shows a more realistic representation of the edges as you can still see the original image behind the edges. I discovered most of the techniques regarding contouring from the OpenCV tutorial page [5].

This component had a variety of different functions implemented that all performed the same general contouring classification but either took different arguments or provided different results based on different parameters, Firstly was the initial contouring method which has now been renamed to “contouringOld” which takes the selected image and converts it to an OpenCV image, and then uses an outdated denoising technique called opening which creates spaces between some of the pixels to remove noise in a form of blurring. However, it wasn’t overly efficient. Next the function would run an edge detection method on the image and then use OpenCV’s findContours feature which will trace the edges found by the canny edge detector. Next, I store all the counters found in a list and count the number of contours for that image. If

the contour count was over 15 then I said the galaxy was Spiral, otherwise it was elliptical. However. This method proved to be inaccurate as both galaxy types would routinely return the same number of contours in the image regardless of its type.

Since the previous method was too inaccurate, I decided to tweak the function slightly using the code I already had as a framework on which I can expand upon to receive more accurate results. I streamlined the function by removing most of the pre-processing into other functions. The new “contouring” function starts off by finding contours straight away and then stores all these contours in a list and then counts all contours in the image, the contours and the contour count are then returned. These variables are mainly used in a function called “detect” which is my default classification algorithm which I’ve assigned to the most accurate classification method. This method will take the selected image and then run template matching and edge detection the image. The function then called the contouring method and stores the results into variables of the same name of those returned previously. For each contour, I then calculate the perimeter and add it to a perimeter total by using OpenCV’s arc length function which calculates the average perimeter in pixels. (I’ve multiplied this result by 100 to make it easier to work with.) I then work out the average perimeter length of all contours in the image if it is greater than 850 it’s classified as a spiral, otherwise it’s classified as an elliptical. The result is then formatted into a string and displayed to the user.

```
1. def contouring(self, im):
2.     _, contours, hierachy = cv2.findContours(im.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
3.
4.     contourList = []
5.     for contour in contours:
6.         contourList.append(contour)
7.
8.     contourCount = 0
9.     for item in contourList:
10.         contourCount += 1
11.
12.     return contourList, contourCount
```

```
1. def detect(self):
2.
3.     im = cv2.imread(filename)
4.     newIm = self.selectTemplate(im)
5.
6.     edge = self.cannyEdgeArg(newIm)
7.
8.     contourList, contourCount = self.contouring(edge)
9.
10.    perim = 0
11.    for contour in contourList:
12.        perim += cv2.arcLength(contour, True)
13.
14.    total = perim / contourCount * 100
15.    type = ""
16.    if total > 850:
17.        type = "Spiral"
18.        print total, "= Spiral"
19.    else:
20.        type = "Elliptical"
21.        print total, "= Elliptical"
22.
```

```

23.     filenameShort = os.path.splitext(os.path.split(filename)[1])[0]
24.
25.     returnString = filenameShort, ": Type: ", type, ": Average Perimeter Total: ",
        total
26.
27.     return returnString

```

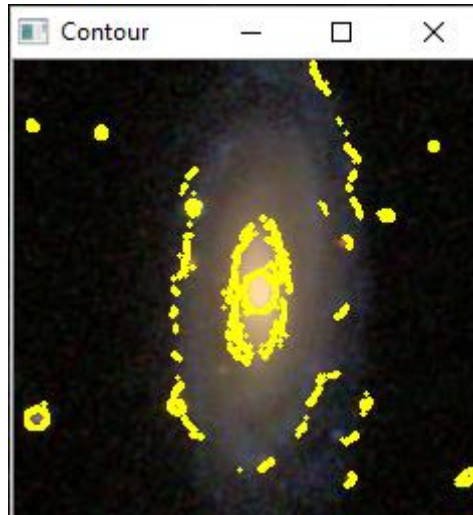
I found this function to be more accurate than the function that just classified them on the contour count alone as it provides a clearer pattern between the different types. This is due to the fact that the spiral galaxies generally cover a larger surface area than ellipticals and that the arms of spiral galaxies will increase the average perimeter area compared to that of ellipticals. This provides an accuracy of around 75% on average with both galaxy types. However, I've noticed some discrepancies with ellipticals that are of a higher image quality than over ellipticals as they tend to produce a higher average contour perimeter than the lower quality images.

Overall, I believe the function works very well to aid in the completion of my aim as it successfully contours the edges of an image and then uses these contours to calculate the average perimeter length of the contours to use for classification purposes with a high accuracy. In future I would look further into a floating pivot point instead of the fixed 850 as it might provide more accurate results. I trialled this method, but the results proved inconclusive with the datasets provided due to large discrepancies with image quality and pc processing power. This component has had some major modifications throughout the development phase including complete reworks of how the component used the contours found in the image and a restructure of the code to make it more streamlined and object orientated friendly.

Below are some images showing the results of this classification method. The first image shows the text representation of the detect functions result. The second image shows the visual representation of the results, which is the contours overlaying the original image.



(Figure 19: Text representation of the results)



(Figure 20: Visual representation of contour tracing)

(E) K-Means:

After doing some further research into further pattern recognition techniques, I found a paper online called the “Application of K-Means method to pattern recognition in on-line cable partial discharge monitoring” [6] Where they discuss how they use K-Means in pattern recognition. Although they use it for different purposes, the same general principle applies.

The K-Means component was used as an alternative to the contouring method to see if I could try an alternative approach to classification, My K-Means quantization works by averaging all colours in the image and then using this to provide a visual representation of that average colour. I thought it might be useful for detecting galaxy types as spirals tend to be blue and ellipticals tend to be more of a brown/ yellow colour. However, due to the image quality, sometimes spirals can come out brown and ellipticals can appear blue. I decided to trial the component to see if it works and it tends to work quite well for elliptical galaxies but struggles with the spiral galaxies.

This component has 3 main functions, the first simply called “KMeans” will reshape the image then will run OpenCV’s “random centers” function to run on several clusters in the image and will apply the K-Means to those clusters. It will then convert the centre back into uint8. K is the number of clusters selected. The lower the cluster, the higher the average it will cover. I’ve set K to 2 so it covers a large area in the small image. The second method was trialling the contouring on the K-Means as it averages the colour in the image leaving less gaps between the contours. However, as it averages the colours, the average perimeter length became too distorted to accurately measure any difference between types. It works like the method mentioned in the contouring section, it’s only difference is that it uses a pivot point of 225 instead of 850 due to the smaller number of contours overall.

```
1. def kMeans(self,im):
2.
3.     z = im.reshape((-1,3))
4.
5.     z = np.float32(z)
6.
7.     crit = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
```

```

8.     k = 2
9.     ret, label, center = cv2.kmeans(z,k,None,crit,10,cv2.KMEANS_RANDOM_CENTERS)
10.
11.     center = np.uint8(center)
12.     res = center[label.flatten()]
13.     res2 = res.reshape((im.shape))
14.     #cv2.imshow('K-Means', res2)
15.     #cv2.waitKey(0)
16.
17.     return res2
18.
19. def contourKMeans(self,im):
20.
21.     edge = self.cannyEdgeArg(im)
22.
23.     contourList, contourCount = self.contouring(edge)
24.
25.     perim = 0
26.     for contour in contourList:
27.         perim += cv2.arcLength(contour, True)
28.
29.     total = contourCount
30.
31.     print total
32.
33.     if total < 225:
34.         print (total , ": Spiral")
35.     else:
36.         print (total , ": Elliptical")
37.
38.     # print contourCount
39.     cv2.drawContours(im, contourList, -1, (0, 255, 255), 2)
40.     cv2.imshow('Contour', im)

```

The final method was an average colour checker which finds the average colour for each row in the image and then only selects the blue channel of that image. If the average blue level was over 17 (very dark blue) it would be classified as a spiral, under 17 would be classified as an elliptical. However, spiral galaxies tend to merge their core into the image which is a gold colour which provided inconclusive results. This method however, worked very well on elliptical galaxies.

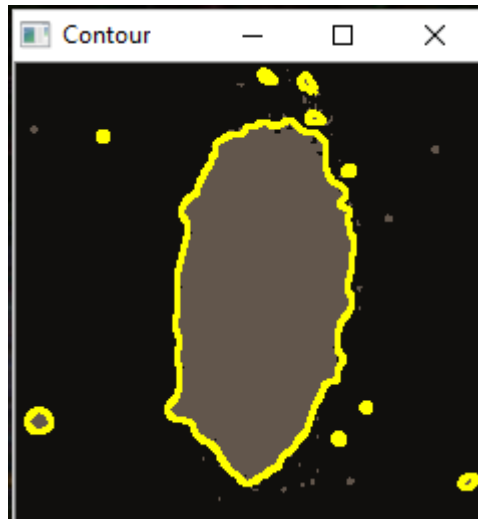
```

1. def kAvgColour(self, im):
2.     avgPerRow = np.average(im, axis=0)
3.     avg = np.average(avgPerRow, axis=0)
4.     print avg
5.
6.     if avg[0] < 17:
7.         print "Elliptical"
8.     else:
9.         print "Spiral"
10.
11.     return avg[0]

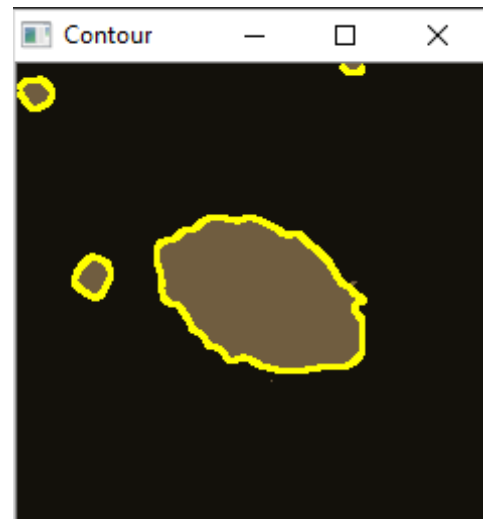
```

Overall, I believe this module was a good method to trial for an alternative approach to classifying galaxies. However, it just didn't prove accurate enough to accurately provide any clear differences between the elliptical galaxies and the spiral galaxies. This component has some minor modifications throughout the implementation phase, most of it being trial and error on the colour values and the K variable.

Below is an image detailing the graphical representation of the K-Means averaging on a spiral galaxy. Note that the majority of the image is brown which is due to the core of the image being merged with the blue gas around the edges. Also note how the contour in this image is closed in comparison to the canny edge contouring, this will increase the perimeter size but due to the normalized colour, the detailing of the arms is lost meaning perimeter lengths tend to be very similar regardless of type. Also note how the elliptical quantization is more golden than its spiral counterpart.



(Figure 21: Spiral Quantization)



(Figure 22: Elliptical Quantization)

(F) Circle Detection:

I also trialled another component that uses Hough circle detection methods to find evidence of circles in images. It works similarly to canny edge detection method by finding sharp gradients in colour, however this time it will only work if the edges are found within a closed/ semi-closed radius. I thought that this might work as the centre of the spiral galaxies tend to change sharply in colour which could be used for classification purposes through the aforementioned technique. I implemented this method after research into other pattern recognition techniques. I found a paper titled “A Fast randomized circle detection algorithm” [10] where they discuss how circle detection can be used for pattern recognition.

It works by taking the selected file and performing a median blur on the image to remove noise, it is then converted to black and white and then passed through OpenCV’s HoughCircles function which uses the gradient technique to find edges within a radius. I’ve then counted the number of circles found in the image and drawn a circle around the detected radius and places a point at the centre of the radius. I then print the number of circles found and return the graphical representation to the user.

```
1. def houghCirlce(self, filename):
2.     im = cv2.imread(filename, 0)
3.     im = cv2.medianBlur(im, 5)
4.     cim = cv2.cvtColor(im, cv2.COLOR_GRAY2BGR)
5.     circles = cv2.HoughCircles(im, cv2.HOUGH_GRADIENT, 1, 20, param1=50, param2=29,
6.                               minRadius=0, maxRadius=0)
```

```

7.     count = 0
8.     # circles = np.around(circles)
9.     if circles is not None:
10.        for i in circles[0, :]:
11.            count += 1
12.            # draw the outer circle
13.            cv2.circle(cim, (i[0], i[1]), i[2], (0, 255, 0), 2)
14.            # draw the center of the circle
15.            cv2.circle(cim, (i[0], i[1]), 2, (0, 0, 255), 3)
16.
17.     print count
18.
19.     cv2.imshow('detected circles', cim)
20.     cv2.waitKey(0)

```

Overall, this method works but not well enough to use for classification purposes, I thought since spiral galaxies have the brighter contrast in the centre, it would detect this as a circle compared to the faded edges of the elliptical galaxies that fade out with less of a sharp gradient. With further tweaking this method could be used efficiently to find major differences between the types, but due to the general poor image quality, this task would take too much time to tweak to be accurate enough compared to the contour tracing method. Due to this, no major modifications were made to this component as it would not prove beneficial to the implementation phase. However, the function can still be run from the tools section as a feature extraction method.

Below is screenshot showing the graphical results of the circle detection component for both elliptical and spiral galaxies.



(Figure 23: Elliptical Circle Detection)



(Figure 24: Spiral Circle Detection)

(G) Miscellaneous Classification Components:

(1) Fill Canny

Fill canny was an attempt to close off most of the open edges from the canny edge detector but I couldn't get kernel to accurately fill most of the edges without being too excessive. This component was abandoned not long after but was left in as a test feature

(2) Total Contours

Total contours are the same as the default classification method but is just used to print the total number of contours in the image for debugging purposes.

(3) Contour Estimate

This component was used to find the average contour count for all imported images, this average would then be used as the pivot point for an alternative perimeter

average function. Initially it provided promising results but was too inconsistent due to rogue images that contain a lot of artefacts would ruin the average result.

It worked by calling the total contours function for each image imported and adding it to the total to find the average contour count of all images. The classifyOverAverage function would then run the exact same code as the fixed-point classifier but instead changes this fixed point to equal the estimate count. If the average perimeter of the selected images is greater than the average of all images minus 5%, it's considered a spiral galaxy, otherwise it's considered a spiral.

```
1. def contourEstimate(self, fileList):
2.     total = 0
3.     imCount = 0
4.     for file in fileList:
5.         imCount += 1
6.         total += self.totalContours(file)
7.     global contourEstimateCount
8.     contourEstimateCount = total / imCount
9.     print contourEstimateCount
10.
11.     return contourEstimateCount
12.
13. def classifyOverAverage(self, filename):
14.     im = cv2.imread(filename)
15.     newIm = self.selectTemplate(im)
16.
17.     edge = self.cannyEdgeArg(newIm)
18.
19.     contourList, contourCount = self.contouring(edge)
20.
21.     perim = 0
22.     for contour in contourList:
23.         perim += cv2.arcLength(contour, True)
24.
25.     total = perim / contourCount * 100
26.     estimateCount = contourEstimateCount
27.     if estimateCount == 0:
28.         estimateCount = self.contourEstimate()
29.
30.     if (total > estimateCount - (estimateCount * 0.05)):
31.         print total, "= Spiral"
32.     else:
33.         print total, "= Elliptical"
```

Overall, this method showed initial promise but was ruined by rogue images that contained a lot of artefacts manipulating the overall average. Also, due to poor image quality, the results proved inconclusive overall. If the images were of a higher quality, I believe this function would be the default classification method for the application, the only downside is that this is quite computationally expensive to run as it must pre-process all the contours beforehand. In future I would create a method that allows the user to enter their own pivot point too to speed up the rate at which this method could be run. This component had no major modifications due to it being a trial method.

(iv) Output/ Exporting Components:

The next objective of my project was to implement a way of displaying the accuracy of the classification methods on the application or in the console. This would be useful as it would allow the user to test how accurate the results of their test have been. Initially this is for debug purposes as we already know what the galaxy type is beforehand, but once proven this feature could be removed. Another form of outputting would be the information displayed in the console such as detailed results of the accuracy testing or the classification results. Another format is the labels which have been previously mentioned in the user interface section [2.3.ii]. The final output method is exporting the classification results to the text file so that the user can send the results to other users or for storage.

This module is heavily tied with the classification section as it must recall most of the functions from that section to be able to display and export the information in a format that makes sense to the average user of the application. I will go into further detail about why each of the components in this section were used, how they were implemented, the overall summary of that component and the modifications made to that component.

(A) Accuracy Checkers:

I needed a way quickly assessing the accuracy of my classification results without having to do it manually for each file imported into the application, the easiest way to do this was to create a function that would check the accuracy of my classification results automatically. I decided on implementing an accuracy checker for both individual galaxy types. This means that the images imported must be of all one type, but it will sift out the negative results from all images of that type. This made debugging my functions a lot easier as the application would quickly evaluate the accuracy of each image imported into the application.

The component has utilised 3 main functions, 2 being the classifiers for each type of galaxy and then 1 main classifier that will run the classification code on each galaxy and return the type detected to the classifier. It works using the code from the average perimeter length function but just returns the galaxy type instead of the average contour number. This galaxy type is then cross-referenced with the current classifier being used and if it matches the type, it will increment the number of correct type matches, the negative file names will be added to a list. The total accuracy is then calculated by the correct number of galaxies detected over the total number of galaxies multiplied by 100 to give the result as a percentage. This accuracy is displayed on the main label in the UI and the console. The negative file names are then printed in the console to inform me and the user of which files provided negative results.

```
1. def returnClassifier(self,file):
2.     im = cv2.imread(file)
3.     newIm = self.selectTemplate(im)
4.
5.     edge = self.cannyEdgeArg(newIm)
6.
7.     contourList, contourCount = self.contouring(edge)
8.
9.     perim = 0
10.    for contour in contourList:
11.        perim += cv2.arcLength(contour, True)
12.
13.    total = perim / contourCount * 100
14.    galType = 0
```

```

15.     if total > 850:
16.         galType = 1
17.         # print total, "= Spiral"
18.     else:
19.         galType = 2
20.         # print total, "= Elliptical"
21.
22.     # print contourCount
23.     # cv2.drawContours(newIm, contourList, -1, (0, 255, 255), 2)
24.     # cv2.imshow('Contour', newIm)
25.     return galType
26.
27. def checkSpiral(self, fileList):
28.     total = 0.0
29.     correctType = 0.0
30.     negativeFiles = []
31.     for file in fileList:
32.         total += 1.0
33.         isSpiral = self.returnClassifier(file)
34.         if isSpiral == 1:
35.             print "Spiral"
36.             correctType += 1.0
37.         else:
38.             negativeFiles.append(file)
39.
40.     print "Total: ", total
41.     print "Spiral: ", correctType
42.     accuracy = correctType / total * 100
43.     print "=====
===="
44.     print "Accuracy of spiral galaxy detection in a dataset of ", total, " All S
piral Images"
45.     print "Accuracy: ", accuracy, "%"
46.     print "=====
===="
47.     print "Negative results are in the following images:"
48.     for item in negativeFiles:
49.         print item
50.     printString = "Accuracy: ", accuracy, "%"
51.     return printString

```

I also implemented another 2 methods for the K-Means accuracy testing as it proved to be initially promising, but mass testing showed that it was not a viable approach for classification.

Overall, I believe this component proved to be incredibly beneficial, if not vital to the success of the classification phase as it allowed me to quickly reference the accuracies saving a great deal of time when it came to the trial and error stage of implementation. Various modifications have been made to adjust it to the current default classification method and implementing a second accuracy tester purely for K-Means. In future I would have implemented an accuracy tester that could check both types at the same time, this could be done by checking the filename as the files come pre-named with their correct type.

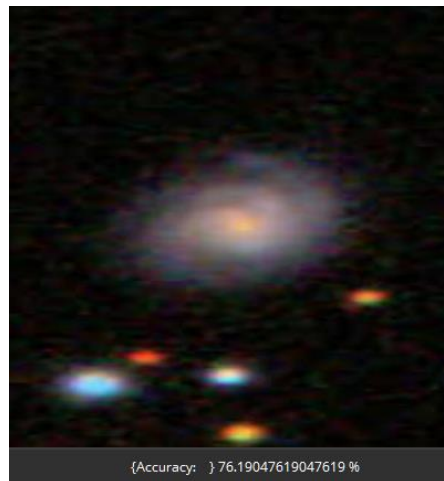
I've included some images showing the various outputs for the accuracy testing phase, both the label format and then the console output which shows which files have been incorrectly classified. Common results show that the average for spiral galaxies is in the 75-80% region whereas the average elliptical accuracy is wilder, being in the 65-80% region.

```

Total:      21.0
Spiral:     16.0
=====
Accuracy of spiral galaxy detection in a dataset of  21.0  All Spiral Images
Accuracy:    76.1904761905 %
=====
Negative results are in the following images:
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722953841770855.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982285836573.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982297764190.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982298616261.jpg
E:/GalaxyTypeRecognition/CKW/test01/CKW-587722982299074952.jpg

```

(Figure 25: Spiral Console Accuracy Output)



(Figure 26: Spiral Label Accuracy Output)

(B) Console Output:

I used the console to display the detailed information from the classification processes. The reason for this is to reduce the amount of information cluttering the interface. Sending it to the console means that the detailed information still gets displayed, but in a way that doesn't clutter up the application and allows users to focus more on their workspace.

The console is used by a variety of different methods, as seen before with the accuracy testers, but it's also used by the default classification method. It in this case the console data is simplified as it informs the use if a template has been matched, the average perimeter of the contours, and the predicted type. Another method that uses the console is the print list debug function which will print all images currently imported to the application. The various classification components that are not default will also send all their data to the console to avoid cluttering the interface.

Overall, I feel the use of the console is more beneficial towards the in-depth detail that the accuracy testers need to provide, and also more appropriate for the larger quantities of data such as the various debug functions and secondary classification methods.

(C) Exporting to a text file:

An important part of the classification is being able to export your data into a format that can be stored or sent to other people for further analysis. Being able to export the data found by the classification results was a vital part of the application development. For this I developed a component that would export the classification results of all imported images to a text file. The text file is then automatically saved to the root folder of the application.

The function named “Save” is responsible for the exporting of classification results. It works by first creating a time stamp which will be used as the file name. (Time stamp format: day/ month/ year – hour/ minute). Then for each file imported to the application, it will run the classification code and store the string result in a new line of the file. This simple function allows the application to store the results in raw text format.

```

1. def save(self):
2.     timestr = time.strftime("%d %m %Y - %H %M")
3.
4.     file = open(str(timestr) + ".txt", "w")
5.
6.     file.write("=====
7.     file.write("\nClassification Results for: " + timestr)
8.     file.write("\n=====
9.
10.    for img in fileList:
11.        im = cv2.imread(img)
12.        newIm = self.selectTemplate(im)
13.
14.        edge = self.cannyEdgeArg(newIm)
15.
16.        contourList, contourCount = self.contouring(edge)
17.
18.        perim = 0
19.        for contour in contourList:
20.            perim += cv2.arcLength(contour, True)
21.
22.        total = perim / contourCount * 100
23.        total = round(total, 2)
24.        type = ""
25.        if total > 850:
26.            type = "Spiral"
27.            print total, "= Spiral"
28.        else:
29.            type = "Elliptical"
30.            print total, "= Elliptical"
31.
32.        # print newIm.std()
33.
34.        filenameShort = os.path.splitext(os.path.split(filename)[1])[0]
35.
36.        returnString = filenameShort, ": |Type|: ", type, ": |Average Perimeter
37.        Total|: ", total
38.        file.write("\n"+str(returnString))

```

Overall, I believe that this component functions as intended as it allows the user to successfully export their classification results to a text file which can be used for storage or for transferring the results to other people. There were no major modifications to this component. However, in future I would take more of an object orientated approach with this method by calling the classification functions instead of reinserting all the code again.

```

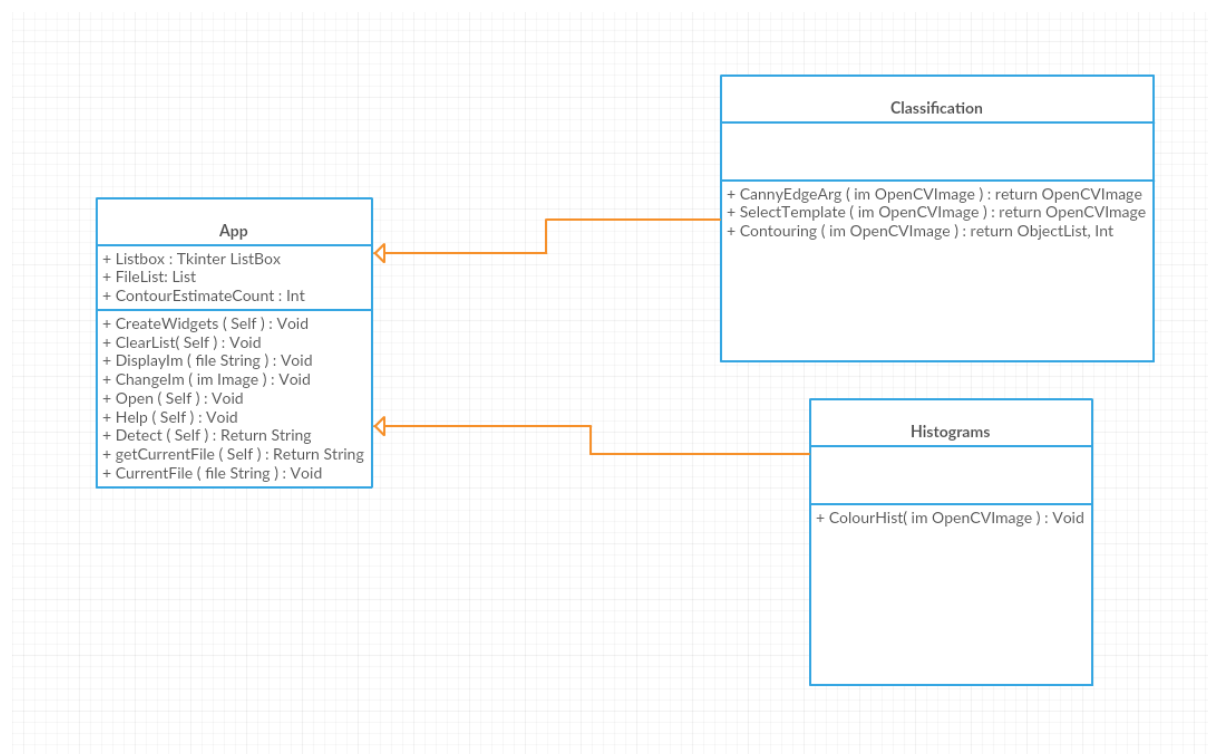
=====
Classification Results for: 01 03 2018 - 17 19
=====
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 921.66)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 1058.27)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 1388.0)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 1005.84)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 966.88)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 1072.96)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 902.68)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 978.06)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 981.17)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 1149.77)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 997.13)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 1094.86)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 1011.99)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 1407.83)
('ACW-587722982270304366', ': |Type|: ', 'Spiral', ': |Average Perimeter Total|: ', 946.7)
=====

```

(Figure 27: exported classification results of 15 spiral images from the 1st of March 2018 at 17:19.)

(v) Class Diagram:

Below, is an image showing the class structure of my application, I have removed secondary classification methods in order to keep the class diagram streamlined. The app classes call the classification class and histogram class to retrieve functions from them to be used for the graphical representation. So, the App class extends the Classification class, and the Histograms class.



(Figure 28: Class diagram of my application)

(vi) Further Resources worth noting:

1. A paper from a collection of physicists where they talk about the morphological classification of galaxies in the galaxy zoo [11] challenge mention a machine learning approach to classification but due to time constraints, something similar could not be implemented.
2. Another user from the galaxy zoo challenge [12] who used a deep learning algorithm to successfully classify galaxies with an accuracy of 99% had an interesting approach, but due to limitation with python and time constraints, it could never be implemented.
3. A web article called the “Automatic detection of galaxy type from galaxy image datasets” [13] where they discuss some interesting techniques to automatic detection.

(3) Project Planning:

(3.1) Introduction:

In this section, I will discuss:

- The project planning method that I used for the development of my project.
- How I kept momentum on the project.
- How I adapted to change.
- How I dealt with risks.
- Reflection on the achievements made.
- What I have learnt.
- How suitable was the waterfall method to completing the project?

(3.2) Project planning methodology:

For my project I opted to use the waterfall method as I felt the linear approach to development would allow me to gather more information in the requirements and design phase that I could use for implementation. I also opted to use waterfall as I have had previous experience with the methodology in previous years and have a broader understanding of it compared to agile.

(3.3) Keeping Momentum:

I kept momentum during my project by working on each task set in the Gantt chart one at a time. I would work my way through the tasks on the Gantt chart logging them in the logbook providing a brief explanation of what I've achieved over a 2-week period. Some weeks were not as productive as others due to other university commitments. Overall, I kept a steady momentum throughout the majority of the project, as the implementation phase came in the momentum increased slightly to adapt to the new workload.

(3.4) Adapting to change:

Adapting to change was a very important part of my project. I had to adapt my workload during the implementation phase to meet the time frame and also what was possible within the framework that I had created, I constantly changed and adapted the techniques I implemented to provide the best accuracy on results and also to efficiently manage my workload around the time I allocated to each task. An example of this would be where I stripped the feature that would extract specific features from the galaxies due to time limitations and library limitations with OpenCV, I invested the extra time this gave me into researching new classification techniques such as the average perimeter area technique used in the final release.

(3.5) Dealing with risks:

Throughout the implementation of my project, I had to deal with a variety of risks that would arise. Solving these risks would allow for more accurate results and make the application more "User-Friendly". The major risk for me was the poor image quality and the artefacts that appear in the

images. These were solved by cropping the image down to the centre to remove unwanted areas of the image and also to use the canny edge detector to discover the edges of the image regardless of the quality, doing this allows the contouring to work on just the edges of the image instead of having to work with a poor quality colour image.

(3.6) Reflection on the achievements made:

Overall, I believe I have achieved the main aim of my project by creating an application that automatically classifies galaxies by their type to a high degree of accuracy. The application has also been provided inside a user interface that allows users to easily select which classification method they wish to test and see the results of this method inside the interface, allowing for a streamlined user-friendly approach for users of any level. This application also then outputs this information in a variety of different formats automatically, or a specific option selected by the user such as, Labels, the console, or even a text document.

(3.7) What I have learnt:

Throughout the development of the project, I feel the biggest thing I have learnt is the ability to solve problems strategically through prioritising specific aspects that show strong potential. For example, when I saw that the contouring approach showed strong potential, I prioritised this approach for classification and made further modifications based on previous findings to create the default classification method available in the final release of the application.

The project has also strengthened my skills in event driven programming by creating the user interface, teaching me skills that will carry over onto future application driven projects. The project has taught me that I should thread new events to avoid creating issues with the application crashing and to implement warning messages and help menus in new pop-up windows.

The most important thing I have learnt throughout this project however, is the ability to prioritise tasks based on the current workload and their potential. The project has taught me that it is okay to abandon ideas that might not work, or ideas that sound good but are unrealistic in the timeframe given.

(3.8) How suitable was the waterfall methodology?

As I was creating the requirements for my project and building the application over the course of a set time. I felt that the waterfall method was the best approach for structuring my project. This allowed me to spend an allocated amount of time on each phase of the project. This linearity allowed for a well structured and streamlined approach to development allowing me to plan and gather requirements efficiently without having to micromanage the other stages of development throughout the year, I could pour 100% of my effort into each stage individually. I feel proof that this method was the best for my project is evident in achieving the main aim of my project.

(4) Conclusion:

Overall, I feel have achieved the main aim of my project which was to create an application that will automatically classify images of galaxies into their respective galaxy types. These types being either Spiral or Elliptical. The objectives completed that go towards completing the aim have been completed in such a way that I believe they meet a high degree of accuracy in displaying the information to the user and do so in a manner that is user-friendly and concise.

The applications objectives which were split into 3 stages (User interface, Classification, Output/Export) are achieved in a way that allows the user to interact with the interface to import a number of images containing galaxies from the Sloan Sky Survey and run the classification code on these images to automatically identify their type. These results can then be exported to an external test file which can be sent to other people or simply be kept for storage.

I am happy with the work I have achieved over the period of my project. I have learnt many new skills and techniques that I can implement into further projects to come and also general working life within the software industry. I have also expanded upon skills, teaching me new approaches I can take to achieving specific goals in relation to application development and project management.

I am also happy with the overall results. My application can predict a galaxies type with an accuracy of 75%. I am also happy with the way that the interface turned out. I believe the interface is of a high quality that will allow users of any skill level to use the application to identify galaxies without needing too much prior knowledge of the subject.

My projects outcome has differed somewhat to the original plan on how the classification would work. Due to time constraints, some features of the application had to be stripped, such as the specific feature extraction method, (However, this was more due to technical limitations) and sub-type classification which would classify whether spiral galaxies were edge on, clockwise, or anti-clockwise.

In future, I would like to implement a basic form of deep learning for my application as I feel it would provide more accurate results than image processing could ever do. Reading up on how other users have achieved this goal through deep learning, I see it as the future for automatic classification. I would also like to implement a way of export the results to a database, so the results are automatically stored in a more data friendly format, this could be done with a search engine such as Kibana. In future I would also like to spend a bit more time refining the interface to give it a more modern feel. While the interface is user-friendly, it's not very pretty and I feel like making it more aesthetically pleasing will only increase the user-friendliness.

(5) References:

- [1] TKDocs, Tkinter User Interface Tutorials, www.tkdocs.com
- [2] D.S.Dhami, Morphological classification of galaxies into spirals and non-spirals, Indiana University, May 2015
- [3] P.Burda & J.v.Feitzinger, Galaxy classification using pattern recognition methods, Harvard university, February 1992, Pages 697 – 705
- [4] A.Clark, CE316 Computer Vision Lecture Notes, University of Essex, September 2017 – December 2017
- [5] OpenCV, OpenCV Python Tutorials, <https://docs.opencv.org>
- [6] X.Peng, C.Zhou, D.M.Hepburn, M.D.Judd & W.H.Siew, Application of K-Means method to pattern recognition in On-Line Cable Partial Discharge Monitoring, Glasgow Caledonian University & University of Strathclyde, May 2013
- [7] Sloan Digital Sky Survey, Sloan Sky Survey Image Datasets, <http://www.sdss.org/>
- [8] OpenCV, OpenCV Computer vision library for Python, <https://opencv.org/>
- [9] Python, Python 2.7.14, <https://www.python.org/>
- [10] L.Jia, C.Z.Peng, H.M.Liu & Z.H.Wang, A fast randomized circle detection algorithm, Henan polytechnic University & North-western Polytechnical University, December 2011,
- [11] H.Dickinson, L.Fortson, C.Lintott, C.Scarlata, K.Willett, M.Beck & M.Galloway, GALAXY ZOO: MORPHOLOGICAL CLASSIFICATION OF GALAXY IMAGES FROM THE ILLUSTRIS SIMULATION, University of Minnesota, January 2018
- [12] GitHub Username: Benanne, My solution for the galaxy zoo challenge, April 2014
- [13] M.A.El Aziz, Automatic detection of galaxy type from galaxy image datasets, Nature.com, November 2016