ST1(4483) / ST1G(8995) Capstone Project

Capstone Project submission due date: Week 13 Friday 11.59 pm
This assignment is worth 40 marks

Faculty of Science and Technology

Assignment Coversheet

Student ID number & Student Name	U3253277, Shiva Banda
Unit name	Software Technology 1
Unit number	4483
Unit Tutor	Ms. Linda Ma
Assignment name	ST1 Capstone Project – Semester 1 2023
Due date	Week 13 Friday 11.59 pm
Date submitted	Week 13 Friday 11.59 pm

You must keep a photocopy or electronic copy of your assignment.

Student declaration

I certify that the attached assignment is my own work. Material drawn from other sources has been appropriately and fully acknowledged as to author/creator, source and other bibliographic details.

Signature of student: Shiva Banda **Date: 29/10/23**

Introduction:

This paper addresses the Python Capstone Project for the ST1 unit within the parameters stated in the assignment handout. I choose to work on the project with the dataset that was provided to me. The dataset "Diabetic Retinopathy" is accessible on Kaggle data repositories [1].

Diabetic Retinopathy (DR) is the second major cause of blindness among diabetic patients, posing a tremendous issue in diabetic patient management. This devastating disorder is frequently caused by uncontrolled blood sugar levels over an extended period, causing in gradual damage to the fragile blood vessels inside the retina. The development of specific lesions is a hallmark of DR, with Microaneurysms (MAs) being one of the first visible indications. Early detection and treatment are important for preventing serious vision impairment or blindness caused by DR.

Medical imaging and ophthalmology have made amazing advances in recent years, particularly in the development of computer-aided diagnostic algorithms. These powerful algorithms make use of digital fundus pictures to detect MAs and other early indicators of DR. The AGAR300 dataset published here is a substantial contribution to the field, serving as a vital resource for academics and healthcare professionals engaged to the fight against DR. This dataset contains 28 colour fundus pictures that were methodically collected with a 45-degree field-of-view and a high quality of 2448x3264 pixels.

The AGAR300 dataset is a must-have for benchmarking and verifying MA detection systems. This resource may be used by researchers, medical practitioners, and the scientific community to advance the state of the art in early DR diagnosis. The AGAR300 dataset results and insights have the potential to revolutionise diabetic patient care and contribute to the prevention of vision loss, furthering the search for appropriate DR management and treatment techniques.

Methodology:

The software platform development methodology follows an organised approach divided into three essential stages:

1. Algorithm Design and Evaluation:

This first level is concerned with the design and development of decision assistance algorithms. It begins with a thorough exploratory data analysis (EDA) to obtain insights into the dataset's features and identify useful patterns.

Multiple algorithms are created and evaluated using predictive analytics approaches. These algorithms are designed to solve real-world problems efficiently and accurately. Furthermore, to select the best-performing algorithm among the choices, rigors assessment measures are used. This algorithm will act as the software platform's base.

2. Software Tool development:

The best-performing algorithm is then implemented as a desktop software utility using the Tkinter framework in the second stage. Tkinter provides a user-friendly graphical interface that makes user interaction easier.

To improve usability and accessibility, the software product has a simple and easy user interface.

To assure the tool's dependability and accuracy in real-world circumstances, extensive testing and quality assurance techniques are implemented.

3. Development via the web or the cloud:

The software tool is finally ready for deployment as a web or cloud-enabled platform. This change improves accessibility and scalability.

The tool has been optimised for online use, making it available to a greater audience through web browsers or cloud-based solutions.

This complete approach assures an organised and efficient development process, resulting in a strong software platform that handles real-world challenges, aids decision-making, and adapts to the changing demands of its users.

Data Description:

The AGAR300 dataset is a critical resource for the development and assessment of computer-aided algorithms aiming to detect early indications of Diabetic Retinopathy, notably Microaneurysms (MAs). diabetes Retinopathy is the second biggest cause of blindness among diabetes individuals, making early detection and screening crucial for minimising vision loss.

Image Content: The AGAR300 dataset contains 28 colour fundus images. These high-resolution photos, obtained at a resolution of 2448x3264 pixels, show the retinal vasculature and accompanying anomalies, including the existence of Microaneurysms.

Data Source: The photos are collected via Fundus photography machines, assuring standardised and consistent image quality. They are collected with a 45-degree field of view, allowing for thorough retinal examination.

This information not only assists in early DR diagnosis, but it also acts as a vital resource for the scientific community, allowing for the development of novel methods to diabetic patient care. The results obtained utilising the AGAR300 dataset have been recorded in an article published by Elsevier in "Biomedical Signal Processing and Control." Emphasising its importance in medical imaging and ophthalmology.

Logbook Journal:

Week 10: Initial Development, the dataset was looked at, I worked on understanding the dataset and what I am working with. I was then looking at the EDA task and started working on immediately. By the end of the week, I had finished most of the EDA Tasks Done and moved onto to updating my report simultaneously.	Week 11: I worked on my PowerPoint and report further, updated both and then did additional research as I had issues with the dataset but still stuck to it, on the coding aspect I had just started my PDA task.
Week 12: During this week I had completed my GUI and PDA task and was further improving the code where needed and was updating my progress onto the doc simultaneously.	Week 13: During this week I finalised all my progress finished up the code for the assignment and finished my week 10-13 challenges and some finishing touches to my assignment.

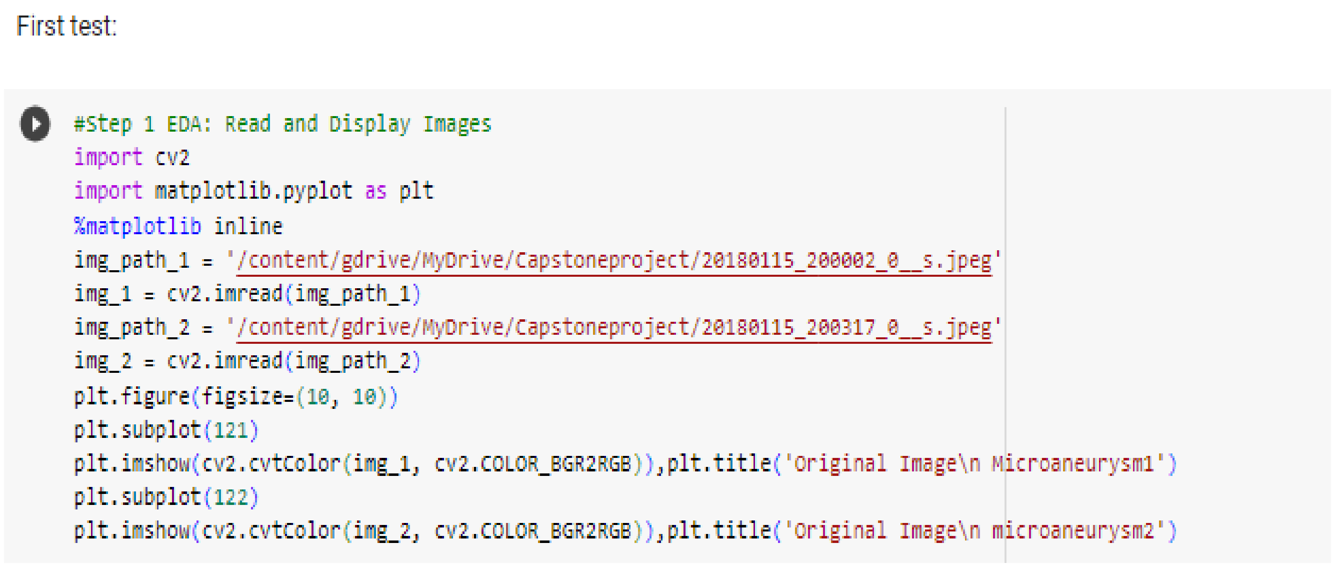
Exploratory Data Analysis and Visualization Task:

The first part of the assignment is exploratory data analysis (EDA), which lays the groundwork for the future analytical procedures. It entails systematically examining and interpreting the dataset to identify relevant patterns, trends, and insights. EDA gives a complete view of the data's properties and informs crucial choices for the assignment through summary statistics, visualisations, and data manipulation.

Google Colab was chosen as the experimental environment because it includes virtual hardware and resources, removing the need for extra physical gear. It may be run simply from a web browser, making it a very accessible option. Python was used as the programming language for creating scripts, which worked well within the online Jupyter notebook environment provided by Google Colab. This was accomplished by utilising a free Google account and essentially storing all notebook data on Google Drive, all without the need for additional setups. Before beginning exploratory data analysis, the Python libraries necessary for EDA were loaded, and the dataset was obtained using the Python script below.

First Test:

Code:



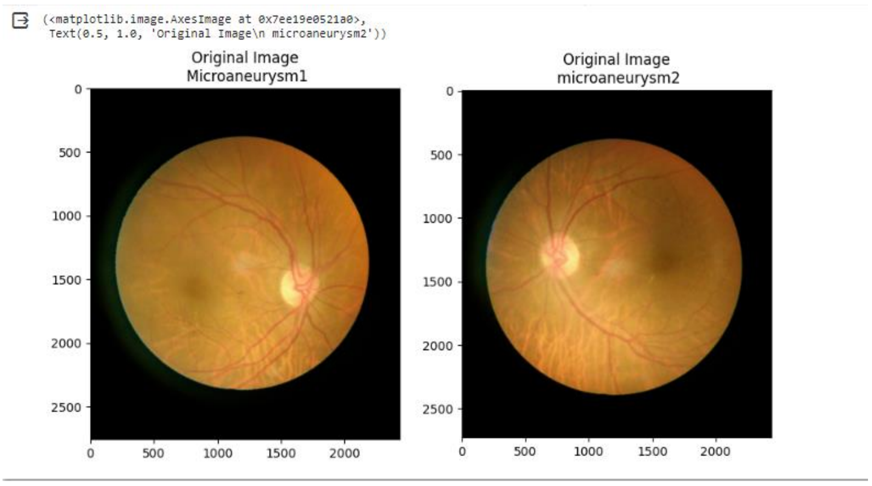
First test:

```
#Step 1 EDA: Read and Display Images
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
img_path_1 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200002_0__s.jpeg'
img_1 = cv2.imread(img_path_1)
img_path_2 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200317_0__s.jpeg'
img_2 = cv2.imread(img_path_2)
plt.figure(figsize=(10, 10))
plt.subplot(121)
plt.imshow(cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB)),plt.title('Original Image\n Microaneurysm1')
plt.subplot(122)
plt.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB)),plt.title('Original Image\n microaneurysm2')
```

The first test just focuses on Reading The right file Paths and Displaying images. File Path for this test was from google drive. The two images used for this test was Microaneurysm2, microaneurysm1.

Testing:

The result just displays the two images with its titles.

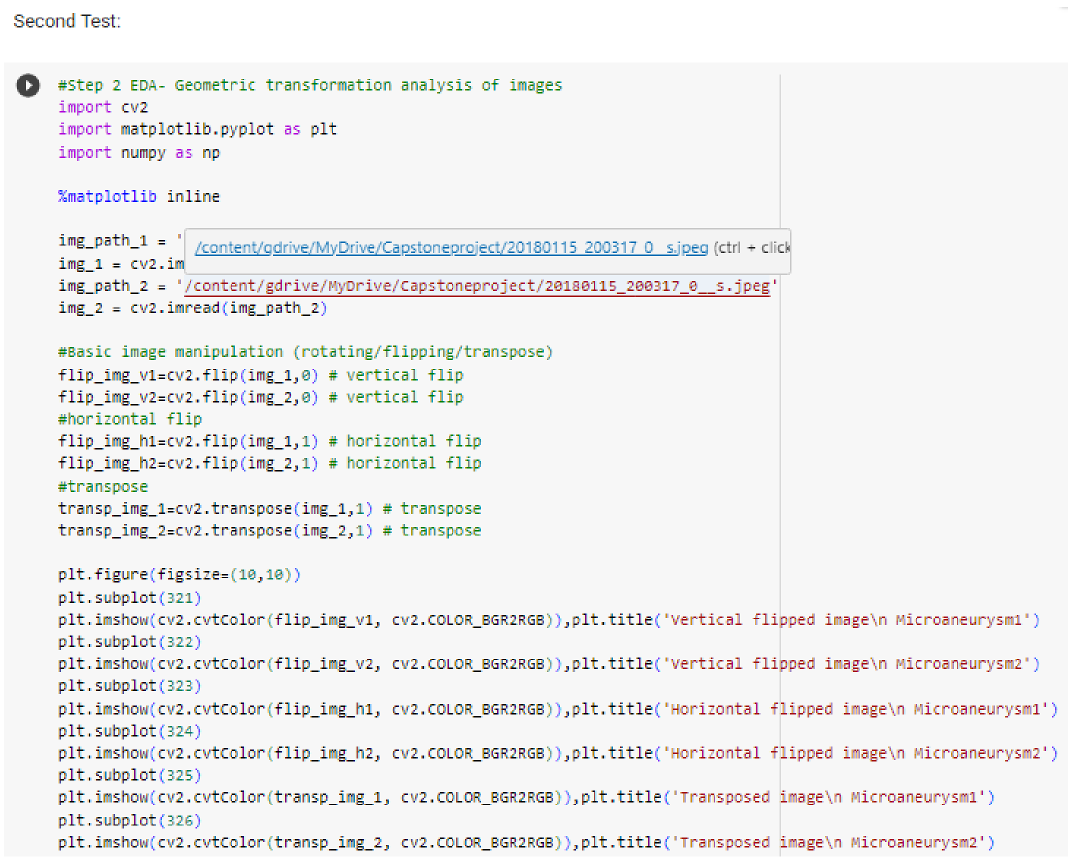


```
(<matplotlib.image.AxesImage at 0x7ee19e0521a0>,
 Text(0.5, 1.0, 'Original Image\n microaneurysm2'))
```

Second test:

The second test just builds off the foundation provided by the first test, the code displays the images, reads the file paths, and then geometrically manipulates the images using the code below.

Code:

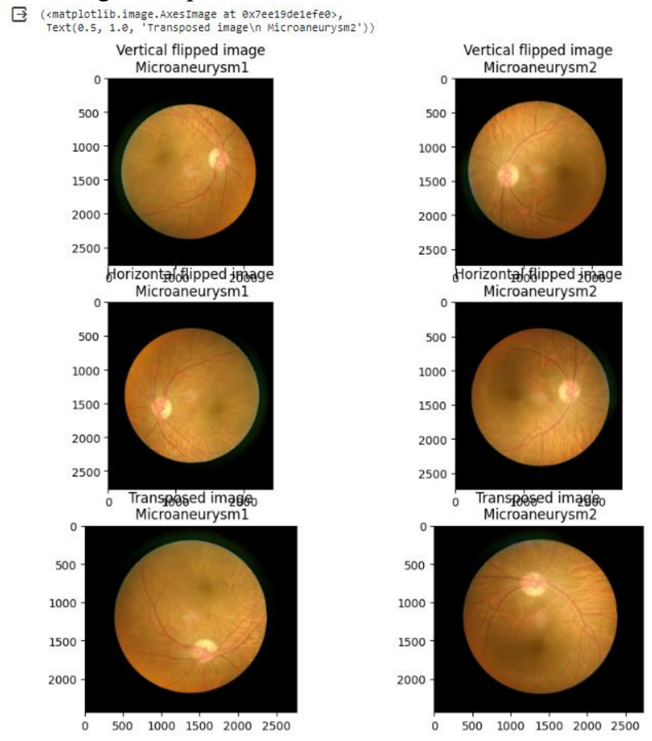


Second Test:

```
#Step 2 EDA- Geometric transformation analysis of images
import cv2
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline

img_path_1 = ' /content/gdrive/MyDrive/Capstoneproject/20180115 200317 0  s.jpeg (ctrl + click
img_1 = cv2.im
img_path_2 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200317_0__s.jpeg'
img_2 = cv2.imread(img_path_2)

#Basic image manipulation (rotating/flipping/transpose)
flip_img_v1=cv2.flip(img_1,0) # vertical flip
flip_img_v2=cv2.flip(img_2,0) # vertical flip
#horizontal flip
flip_img_h1=cv2.flip(img_1,1) # horizontal flip
flip_img_h2=cv2.flip(img_2,1) # horizontal flip
#transpose
transp_img_1=cv2.transpose(img_1,1) # transpose
transp_img_2=cv2.transpose(img_2,1) # transpose

plt.figure(figsize=(10,10))
plt.subplot(321)
plt.imshow(cv2.cvtColor(flip_img_v1, cv2.COLOR_BGR2RGB)),plt.title('Vertical flipped image\n Microaneurysm1')
plt.subplot(322)
plt.imshow(cv2.cvtColor(flip_img_v2, cv2.COLOR_BGR2RGB)),plt.title('Vertical flipped image\n Microaneurysm2')
plt.subplot(323)
plt.imshow(cv2.cvtColor(flip_img_h1, cv2.COLOR_BGR2RGB)),plt.title('Horizontal flipped image\n Microaneurysm1')
plt.subplot(324)
plt.imshow(cv2.cvtColor(flip_img_h2, cv2.COLOR_BGR2RGB)),plt.title('Horizontal flipped image\n Microaneurysm2')
plt.subplot(325)
plt.imshow(cv2.cvtColor(transp_img_1, cv2.COLOR_BGR2RGB)),plt.title('Transposed image\n Microaneurysm1')
plt.subplot(326)
plt.imshow(cv2.cvtColor(transp_img_2, cv2.COLOR_BGR2RGB)),plt.title('Transposed image\n Microaneurysm2')
```

Testing:

The images are presented as the result of the code and then altered in various ways.



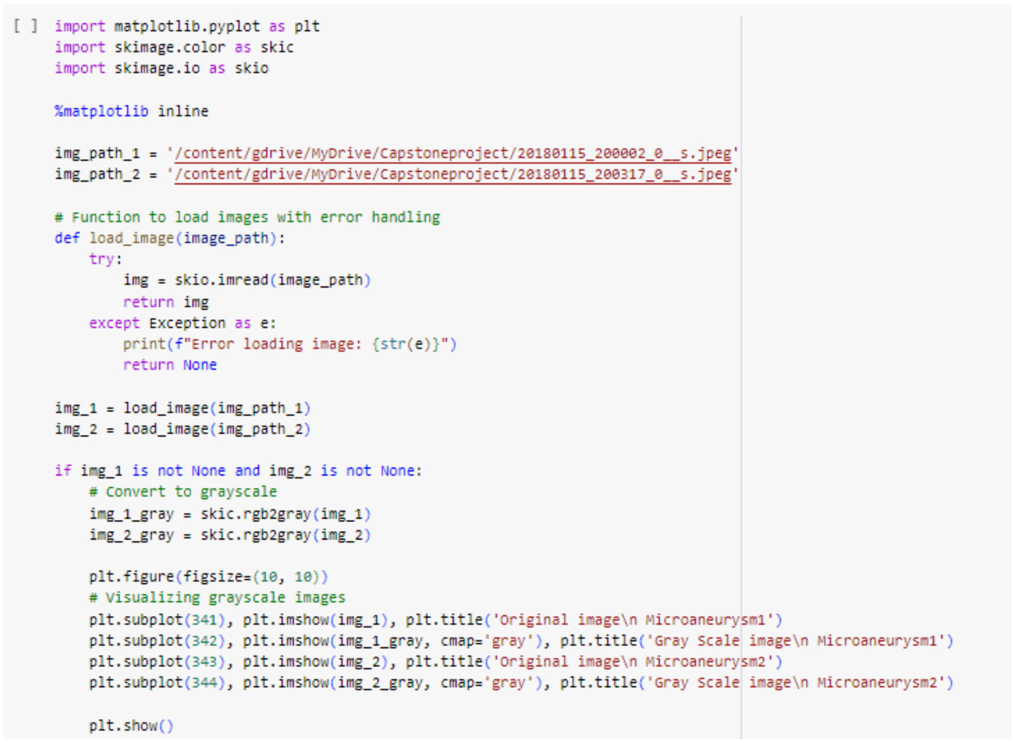
```
(<matplotlib.image.AxesImage at 0x7ee19de1efe0>,
 Text(0.5, 1.0, 'Transposed image\n Microaneurysm2'))
```

Third test:

This test follows the first test’s format and proceeds to manipulate the colour of the images changing it to grey.

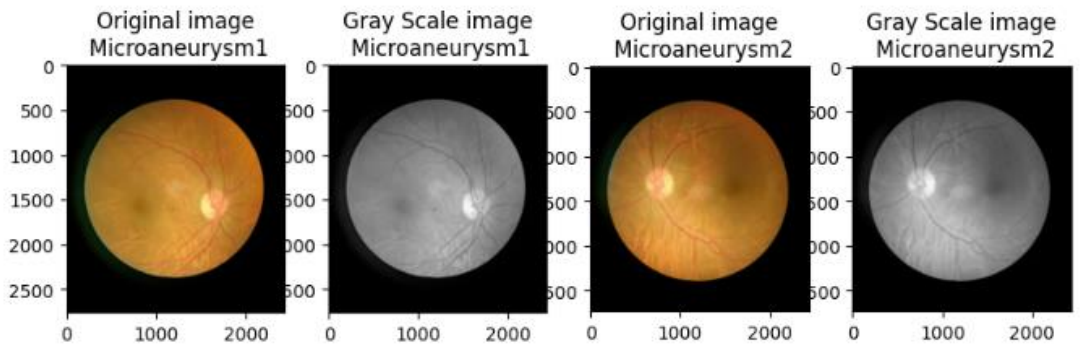
Code:

Third Test:



```
[ ] import matplotlib.pyplot as plt
    import skimage.color as skic
    import skimage.io as skio

    %matplotlib inline

    img_path_1 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200002_0__s.jpeg'
    img_path_2 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200317_0__s.jpeg'

    # Function to load images with error handling
    def load_image(image_path):
        try:
            img = skio.imread(image_path)
            return img
        except Exception as e:
            print(f"Error loading image: {str(e)}")
            return None

    img_1 = load_image(img_path_1)
    img_2 = load_image(img_path_2)

    if img_1 is not None and img_2 is not None:
        # Convert to grayscale
        img_1_gray = skic.rgb2gray(img_1)
        img_2_gray = skic.rgb2gray(img_2)

        plt.figure(figsize=(10, 10))
        # Visualizing grayscale images
        plt.subplot(341), plt.imshow(img_1), plt.title('Original image\n Microaneurysm1')
        plt.subplot(342), plt.imshow(img_1_gray, cmap='gray'), plt.title('Gray Scale image\n Microaneurysm1')
        plt.subplot(343), plt.imshow(img_2), plt.title('Original image\n Microaneurysm2')
        plt.subplot(344), plt.imshow(img_2_gray, cmap='gray'), plt.title('Gray Scale image\n Microaneurysm2')

        plt.show()
```

Testing:

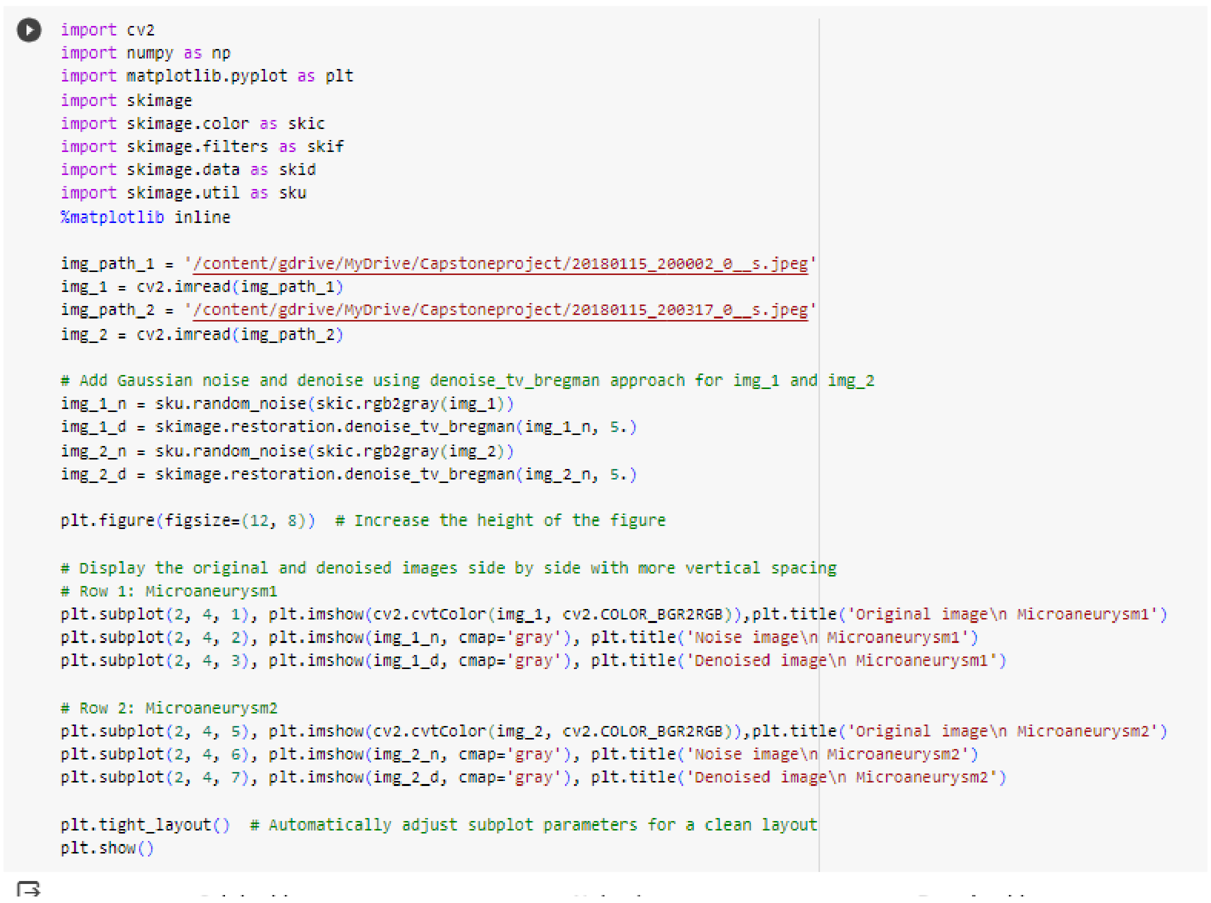


Original image Microaneurysm1
Gray Scale image Microaneurysm1
Original image Microaneurysm2
Gray Scale image Microaneurysm2
0
500
1000
1500
2000
2500
0
1000
2000

Fourth Test:

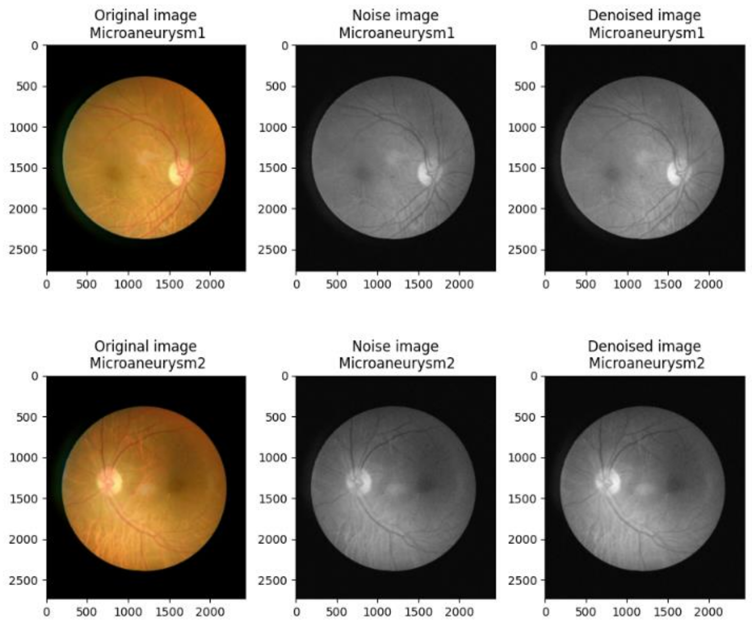
The fourth test was designed to reduce the quality of the images to see how the program interprets the image.

Code:



```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import skimage
import skimage.color as skic
import skimage.filters as skif
import skimage.data as skid
import skimage.util as sku
%matplotlib inline

img_path_1 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200002_0__s.jpeg'
img_1 = cv2.imread(img_path_1)
img_path_2 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200317_0__s.jpeg'
img_2 = cv2.imread(img_path_2)

# Add Gaussian noise and denoise using denoise_tv_bregman approach for img_1 and img_2
img_1_n = sku.random_noise(skic.rgb2gray(img_1))
img_1_d = skimage.restoration.denoise_tv_bregman(img_1_n, 5.)
img_2_n = sku.random_noise(skic.rgb2gray(img_2))
img_2_d = skimage.restoration.denoise_tv_bregman(img_2_n, 5.)

plt.figure(figsize=(12, 8))  # Increase the height of the figure

# Display the original and denoised images side by side with more vertical spacing
# Row 1: Microaneurysm1
plt.subplot(2, 4, 1), plt.imshow(cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB)),plt.title('Original image\n Microaneurysm1')
plt.subplot(2, 4, 2), plt.imshow(img_1_n, cmap='gray'), plt.title('Noise image\n Microaneurysm1')
plt.subplot(2, 4, 3), plt.imshow(img_1_d, cmap='gray'), plt.title('Denoised image\n Microaneurysm1')

# Row 2: Microaneurysm2
plt.subplot(2, 4, 5), plt.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB)),plt.title('Original image\n Microaneurysm2')
plt.subplot(2, 4, 6), plt.imshow(img_2_n, cmap='gray'), plt.title('Noise image\n Microaneurysm2')
plt.subplot(2, 4, 7), plt.imshow(img_2_d, cmap='gray'), plt.title('Denoised image\n Microaneurysm2')

plt.tight_layout()  # Automatically adjust subplot parameters for a clean layout
plt.show()
```

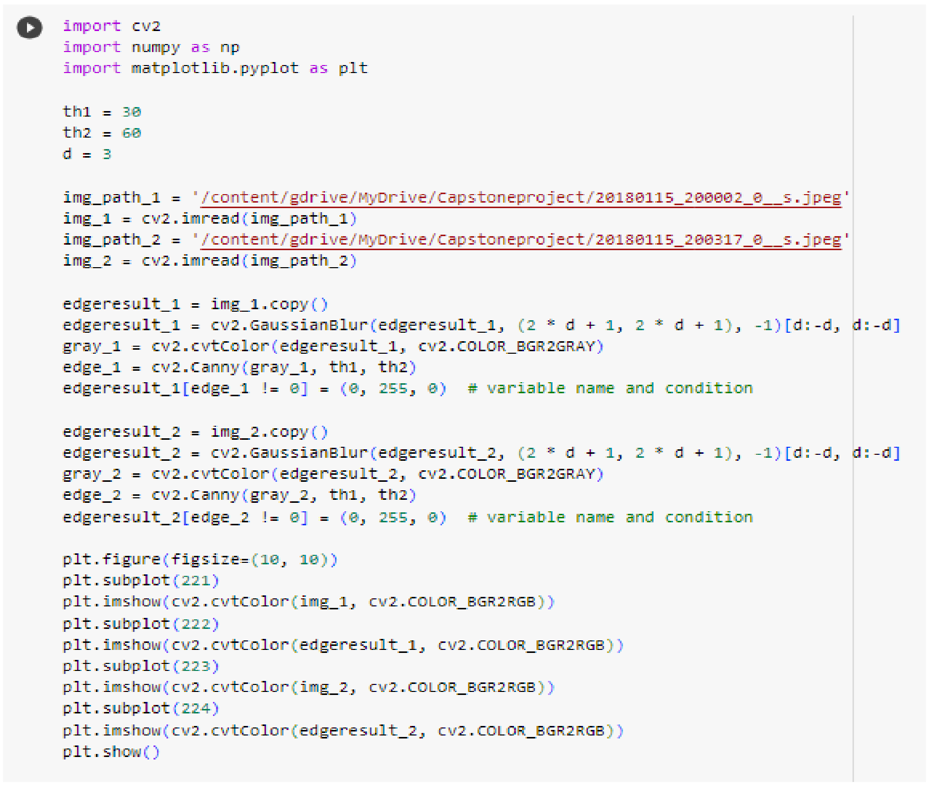
Testing:



Fifth test:

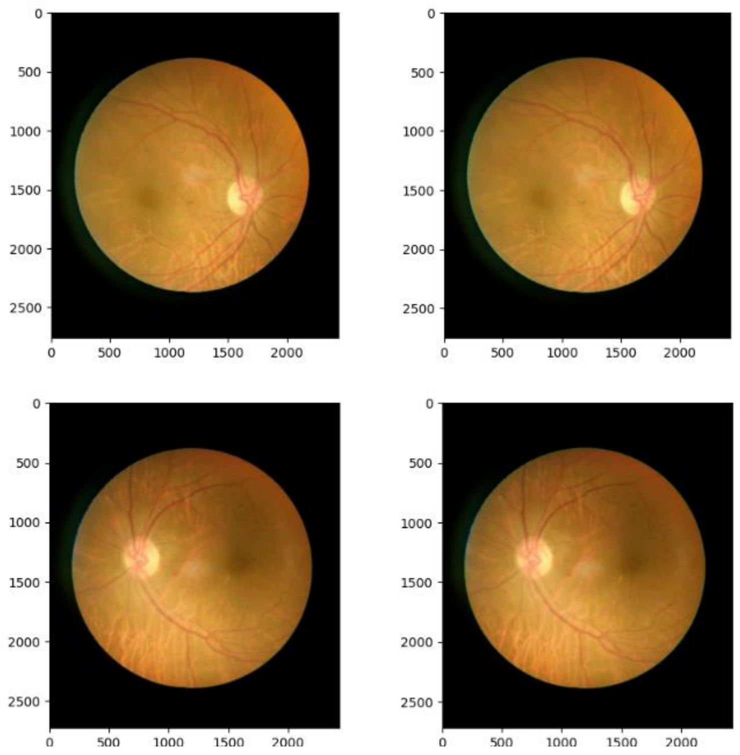
The last stage involves employing clever edge detection to draw on the program's visible edges. This may be used to discover edge patterns inside an image and help distinguish between classes. Can also be used to altered pictures to establish image class by seeing if the edges match.

Code:



```
import cv2
import numpy as np
import matplotlib.pyplot as plt

th1 = 30
th2 = 60
d = 3

img_path_1 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200002_0__s.jpeg'
img_1 = cv2.imread(img_path_1)
img_path_2 = '/content/gdrive/MyDrive/Capstoneproject/20180115_200317_0__s.jpeg'
img_2 = cv2.imread(img_path_2)

edgeresult_1 = img_1.copy()
edgeresult_1 = cv2.GaussianBlur(edgeresult_1, (2 * d + 1, 2 * d + 1), -1)[d:-d, d:-d]
gray_1 = cv2.cvtColor(edgeresult_1, cv2.COLOR_BGR2GRAY)
edge_1 = cv2.Canny(gray_1, th1, th2)
edgeresult_1[edge_1 != 0] = (0, 255, 0)  # variable name and condition

edgeresult_2 = img_2.copy()
edgeresult_2 = cv2.GaussianBlur(edgeresult_2, (2 * d + 1, 2 * d + 1), -1)[d:-d, d:-d]
gray_2 = cv2.cvtColor(edgeresult_2, cv2.COLOR_BGR2GRAY)
edge_2 = cv2.Canny(gray_2, th1, th2)
edgeresult_2[edge_2 != 0] = (0, 255, 0)  # variable name and condition

plt.figure(figsize=(10, 10))
plt.subplot(221)
plt.imshow(cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB))
plt.subplot(222)
plt.imshow(cv2.cvtColor(edgeresult_1, cv2.COLOR_BGR2RGB))
plt.subplot(223)
plt.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB))
plt.subplot(224)
plt.imshow(cv2.cvtColor(edgeresult_2, cv2.COLOR_BGR2RGB))
plt.show()
```

Testing:



Predictive Analysis Task:

In the framework of this research, a predictive analytic task was done to develop a data model using Teachable Machines. This model can take an input image and provide information on its class, as well as display the image and a confidence score. A notable problem in this job was the availability of a single-class dataset, which required the construction of several classes within this constrained dataset. The emphasis was on determining the degree of diabetic retinopathy, which led to the formation of classifications that classified it as either moderate diabetic retinopathy or severe diabetic retinopathy.

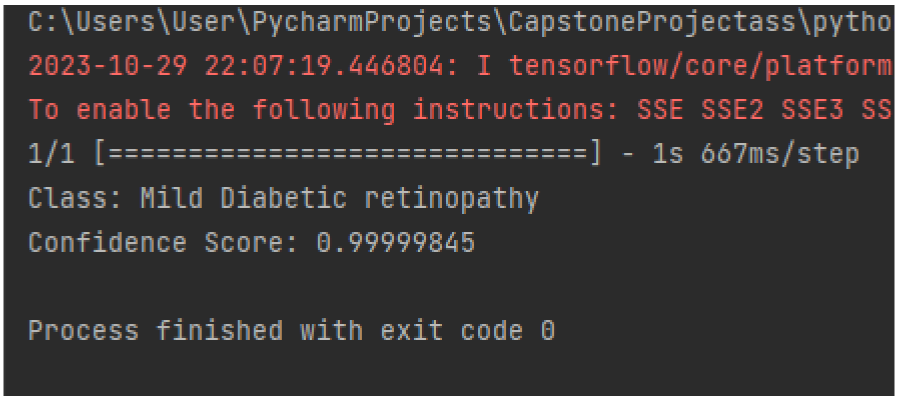
Interface Development, Coding:

To turn this effort into a working software platform, code was written in the Keras deep learning framework. The programme was meant to receive photos and offer real-time classification results, therefore a pre-trained model was loaded. Because of its integration with virtual hardware and resources, Google Colab was chosen as the experimental environment, reducing the requirement for extra physical gear. The categorisation procedure was created and executed using Python scripts, which were seamlessly integrated into online Jupyter notebooks.

```
from keras.models import load_model  # TensorFlow is required for Keras to
work
from PIL import Image, ImageOps  # Install pillow instead of PIL
import numpy as np

# Disable scientific notation for clarity
```

```python
np.set_printoptions(suppress=True)

# Load the model
model = load_model("keras_Model.h5", compile=False)

# Load the labels
class_names = open("labels.txt", "r").readlines()

# Create the array of the right shape to feed into the keras model
# The 'length' or number of images you can put into the array is
# determined by the first position in the shape tuple, in this case 1
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

# Replace this with the path to your image
image = Image.open(r"C:\Users\User\PycharmProjects\pythonProject23\2 (2) - Copy.JPG").convert("RGB")

# resizing the image to be at least 224x224 and then cropping from the center
size = (224, 224)
image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)

# turn the image into a numpy array
image_array = np.asarray(image)

# Normalize the image
normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1

# Load the image into the array
data[0] = normalized_image_array

# Predicts the model
prediction = model.predict(data)
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

# Print prediction and confidence score
print("Class:", class_name[2:], end="")
print("Confidence Score:", confidence_score)
```

Testing:



```
C:\Users\User\PycharmProjects\CapstoneProjectass\pytho
2023-10-29 22:07:19.446804: I tensorflow/core/platform
To enable the following instructions: SSE SSE2 SSE3 SS
1/1 [==============================] - 1s 667ms/step
Class: Mild Diabetic retinopathy
Confidence Score: 0.99999845

Process finished with exit code 0
```

The GUI:

This code generates a graphical user interface (GUI) application for a Diabetic Retinopathy Severity Classifier using Tkinter. It loads a pre-trained deep learning model and uses it to categorise diabetic retinopathy retinal pictures. Users may pick a picture from the project directory by clicking the "Classify Image" button, and the programme will offer real-time estimates about the severity of the retinopathy, as well as a confidence score. The image, classification result, and score are all shown in a fixed window size of 400x400 pixels by the GUI. This application streamlines the process of detecting diabetic retinopathy in medical photographs, allowing for earlier diagnosis and better healthcare decisions.

Coding:

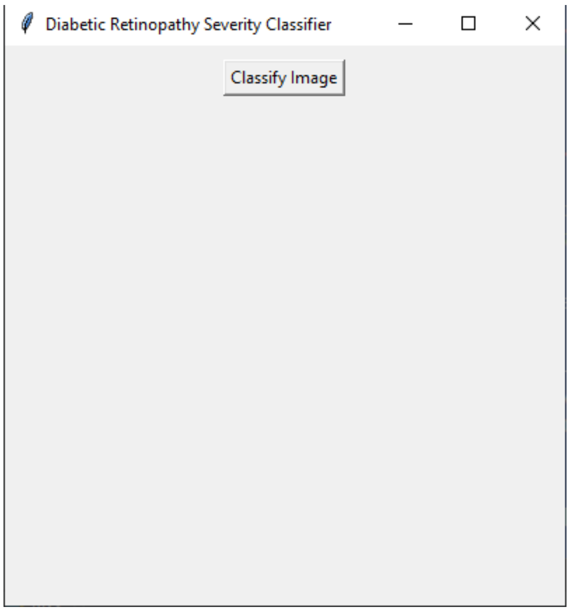
```python
import tkinter as tk
from keras.models import load_model
from PIL import Image, ImageOps, ImageTk  # Import ImageTk
import numpy as np

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)
model = load_model("keras_Model.h5", compile=False)
class_names = open("labels.txt", "r").readlines()

def make_prediction():
    data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

    # Load the image from the project directory
    image = Image.open("IMG-20180327-WA0043.jpg").convert("RGB")
    size = (224, 224)
    image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)
    image_array = np.asarray(image)
    normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1
    data[0] = normalized_image_array
    prediction = model.predict(data)
    index = np.argmax(prediction)
    class_name = class_names[index]
    confidence_score = prediction[0][index]

    # Display prediction and confidence score in the GUI
    result_label.config(text=f"Class: {class_name[2:]}")
    score_label.config(text=f"Confidence Score: {confidence_score:.4f}")

    # Display the image in the GUI using ImageTk
    image.thumbnail((400, 400))
    photo = ImageTk.PhotoImage(image)
    image_label.config(image=photo)
    image_label.image = photo

# Create the main application window
app = tk.Tk()
app.title("Diabetic Retinopathy Severity Classifier")

# window size to 400x400
app.geometry("400x400")

# Button to classify the image
classify_button = tk.Button(app, text="Classify Image",
command=make_prediction)
classify_button.pack(pady=10)

# Labels
result_label = tk.Label(app, text="", font=("terminal", 16))
```

```
result_label.pack()
score_label = tk.Label(app, text="", font=("terminal", 16))
score_label.pack()

image_label = tk.Label(app)
image_label.pack()

# Start the application
app.mainloop()
```

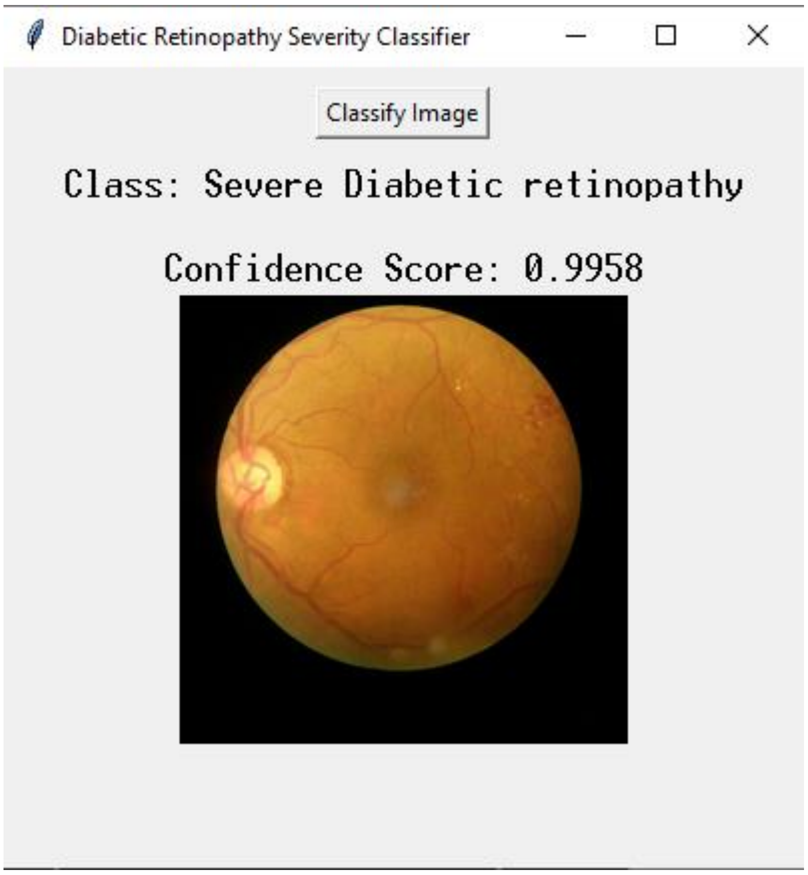
Testing:

The graphical user interface (GUI) shows a small window with a "Classify Image" button. This button allows users to pick a image from the python project files for examination. The GUI then displays the selected image alongside the identified diabetic retinopathy severity categorization and a matching confidence score. This user-friendly interface simplifies the evaluation of retinal pictures, assisting diabetes patients with early diagnosis and medical decision-making.

Stage one result:



Final Result:



Class: Severe Diabetic retinopathy

Confidence Score: 0.9958

Conclusion:

Finally, this Python Capstone Project for the ST1 unit concentrating on Diabetic Retinopathy addresses a vital healthcare issue. diabetes Retinopathy is the major cause of blindness in diabetes individuals, highlighting the need of early identification. With 28 high-quality fundus photos, the AGAR300 dataset is a great resource for academics and healthcare practitioners attempting to tackle this illness. The technique offers an organised and efficient approach to tackling this healthcare dilemma, including algorithm design, software development, and online deployment. Exploratory data analysis and predictive analytics allow for a more in-depth understanding of the information as well as the creation of a sophisticated software platform. The user-friendly GUI streamlines the diagnosis of diabetic retinopathy, providing early identification and informed healthcare decisions.

References:

[1]. “Diabetic retinopathy,” *Kaggle*, Jun. 21, 2023.
https://www.kaggle.com/datasets/eishkaran/diabetes-using-retinopathy-prediction?select=AGAR300%28IEEE+DataPort%29