Faculty of Science and Technology

Assignment Coversheet

Student ID number & Student Name	U3244749 Alexander Zecevic
Unit name	Software Technology 1
Unit number	4483
Unit Tutor	Linda Ma
Assignment name	ST1 Capstone Project
Due date	29th October 2023
Date submitted	29/10/2023

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: A.Z

Date: _____29/10/2023_______

Capstone project stage 1 u3244749

Introduction:

This report details the methods used to perform exploritory data analysis on a given kaggle data set that can be found at https://www.kaggle.com/datasets/alinedobrovsky/plant-disease-classification-merged-dataset. The report will explore the findings of the exploritory data analysis and predictive data analysis and then impliment the data into a GUI using tkinter.

Methodology:

Stage 1 - Exploratory Data Analysis: Perform exploratory data analysis on astronomy images sourced from the dataset. This phase takes place in Google Colab, where we analyze and gain insights from the data.

Stage 2 - Predictive Data Analysis: Predictive data analysis using machine learning platforms such as Keras and Teachable Machine, within the Google Colab environment. Machine learning models are developed to make predictions based on the astronomy image data.

Stage 3 - Implementation as a Desktop Application: The final stage involves the implementation of the software as a desktop application. This implementation is carried out using tkinter and PyCharm, transforming the developed models and analysis into user-friendly, interactive desktop application for users to access and utilise.

Notes: Data set has been reduced due to file size constraints when implimenting GUI. Data set now only contains 100 images from each "Healthy" class of plants.

- Each file has been manually reduced to 100 files for each class.
- "Wheat_Healthy" only has 58 files creating a bias to the other classes
- "Corn_Healthy" was completly annulled due to file compatibility and naming

---

Problem statement in the context of the provided dataset:

- The dataset consists of images covering a varity of different plants
- The images are a varity of different sizes and qualities
- The file paths are seperated by the name of the plant followed by "_Healthy"
- The dataset can be used to develop a plant classifier for the plants included with room for it to be expanded.

Mounting google drive

---

[18]

3s

```
from google.colab import drive
drive.mount('/content/gdrive')
```

output

Mounted at /content/gdrive

Importing relevant python libraries

[19]

0s

```
import tensorflow as tf
import cv2
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
import numpy as np
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
%matplotlib inline
```

Testing file paths by reading and displaying images

cv2.cvtColor(<INSERT IMAGE NAME HERE>, cv2.COLOR_BGR2RGB)

This code is used to translate BRG to RGB due to the fact that openCV and matplotlib use the seperate formats

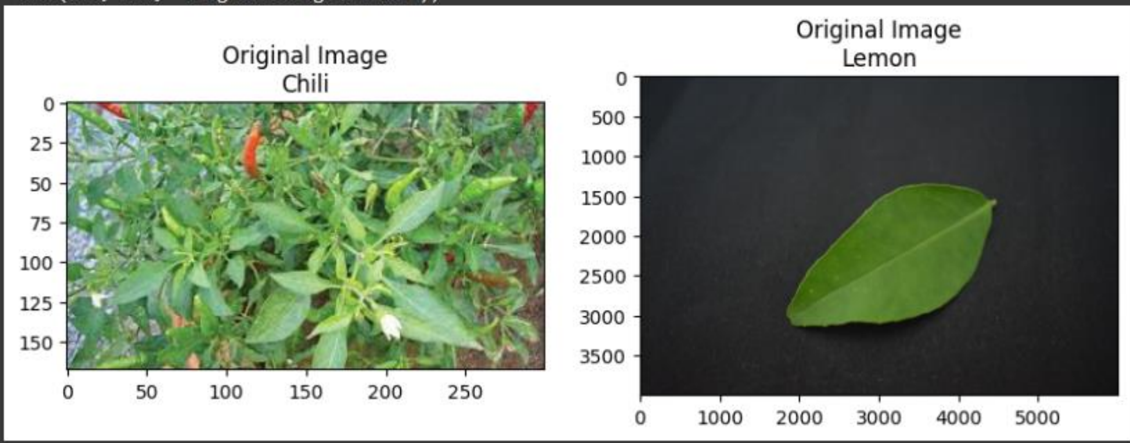
[35]

5s

```
# Read and Dispaly images

img_path_1 = '/content/gdrive/MyDrive/ST1_Capstone/Chili__healthy/Cabai_sehat001.jpg'
img_path_2 = '/content/gdrive/MyDrive/ST1_Capstone/Lemon__healthy/0010_0017.JPG'
img_1 = cv2.imread(img_path_1)
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\nChili')
plt.subplot(122)
plt.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB)),plt.title('Original Image\nLemon')
```

output

```
(<matplotlib.image.AxesImage at 0x7b22a540f520>,
 Text(0.5, 1.0, 'Original Image\nLemon'))
```



The images above are two samples from the several classes selected from the data set. There are two very different images so it makes it easier for a user to see the differences bewtween the images when using exploritory data analysis

---

Gemoetric transformation analysis of images

---

[37]

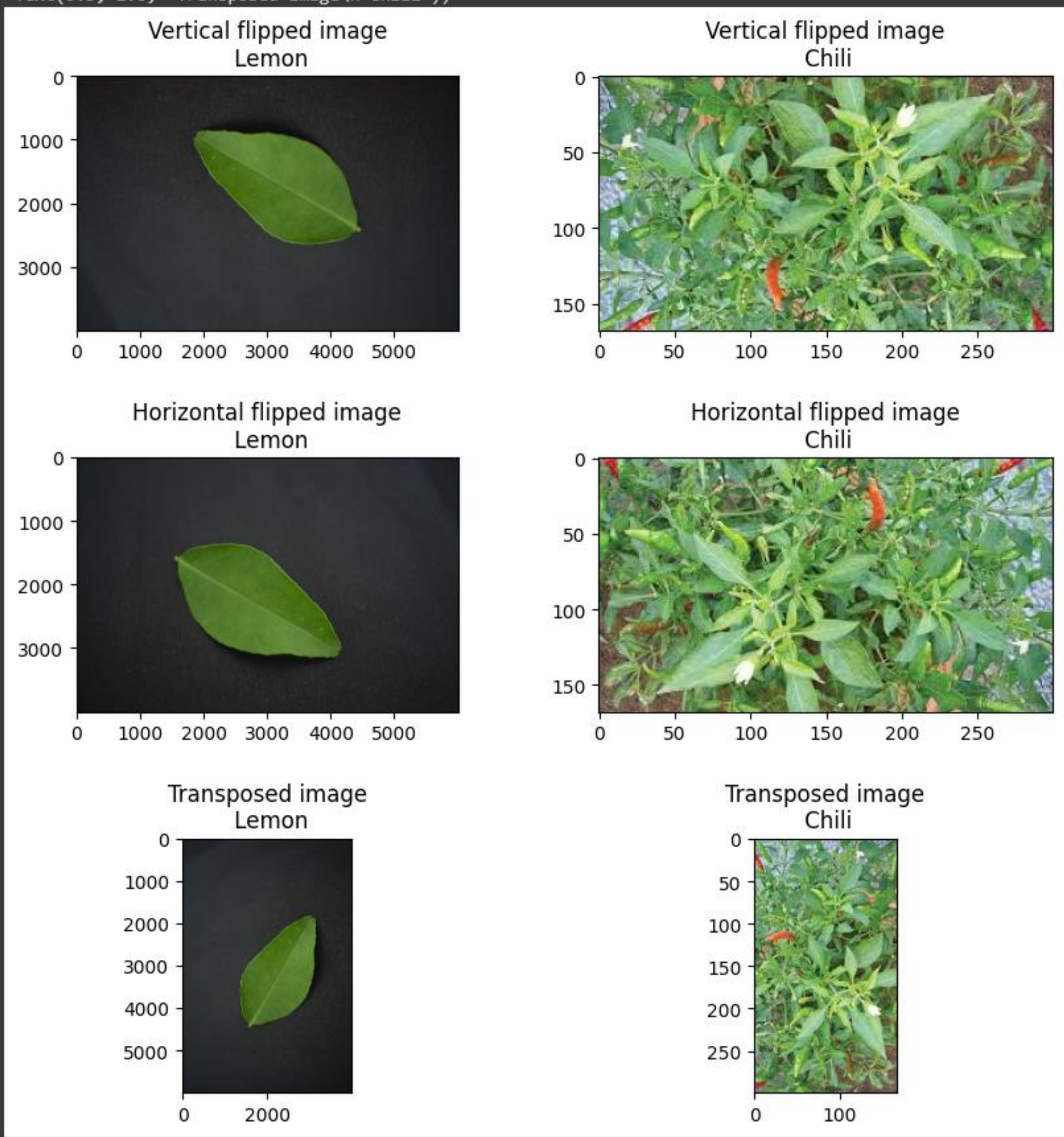
14s

```
img_path_1 = '/content/gdrive/MyDrive/ST1_Capstone/Lemon__healthy/0010_0017.JPG'
img_path_2 = '/content/gdrive/MyDrive/ST1_Capstone/Chili__healthy/Cabai_sehat001.jpg'
img_1 = cv2.imread(img_path_1)
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))
# Increase the spacing between subplots
plt.subplots_adjust(hspace=0.5)

plt.subplot(321)

plt.imshow(cv2.cvtColor(flip_img_v1, cv2.COLOR_BGR2RGB)),plt.title('Vertical flipped image\n Lemon')
plt.subplot(322)
plt.imshow(cv2.cvtColor(flip_img_v2, cv2.COLOR_BGR2RGB)),plt.title('Vertical flipped image\n Chili')
plt.subplot(323)
plt.imshow(cv2.cvtColor(flip_img_h1, cv2.COLOR_BGR2RGB)), plt.title('Horizontal flipped image\n Lem
```

```
on')
plt.subplot(324)
plt.imshow(cv2.cvtColor(flip_img_h2, cv2.COLOR_BGR2RGB)), plt.title('Horizontal flipped image\n Chili'
)
plt.subplot(325)
plt.imshow(cv2.cvtColor(transp_img_1, cv2.COLOR_BGR2RGB)),plt.title('Transposed image\n Lemon')
plt.subplot(326)
plt.imshow(cv2.cvtColor(transp_img_2, cv2.COLOR_BGR2RGB)),plt.title('Transposed image\n Chili')
```

output

```
(<matplotlib.image.AxesImage at 0x7b2294323af0>,
 Text(0.5, 1.0, 'Transposed image\n Chili'))
```



When geometrically manipulated, the images do not look dissimlar from the original. Thus making it easier when attempting to compare the model to a test image

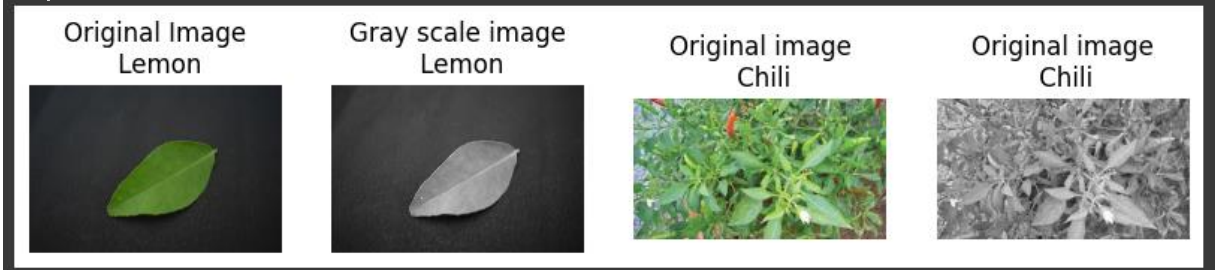
Comparing the images when turned grey

[40]

8s

```
img_path_1 = '/content/gdrive/MyDrive/ST1_Capstone/Lemon__healthy/0010_0017.JPG'
img_path_2 = '/content/gdrive/MyDrive/ST1_Capstone/Chili__healthy/Cabai_sehat001.jpg'
img_1 = cv2.imread(img_path_1)
img_2 = cv2.imread(img_path_2)

fig, (ax1, ax2, ax3,ax4) = plt.subplots(1, 4, figsize=(10, 10))

ax1.imshow(cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB))
ax1.set_title('Original Image\n Lemon')
ax1.set_axis_off()

ax2.imshow(skic.rgb2gray(cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB)), cmap ='gray')
ax2.set_title('Gray scale image\n Lemon')
ax2.set_axis_off()

ax3.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB))
ax3.set_title('Original image\n Chili')
ax3.set_axis_off()


ax4.imshow(skic.rgb2gray(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB)),cmap = 'gray')
ax4.set_title('Original image\n Chili')
ax4.set_axis_off()
```

output



D

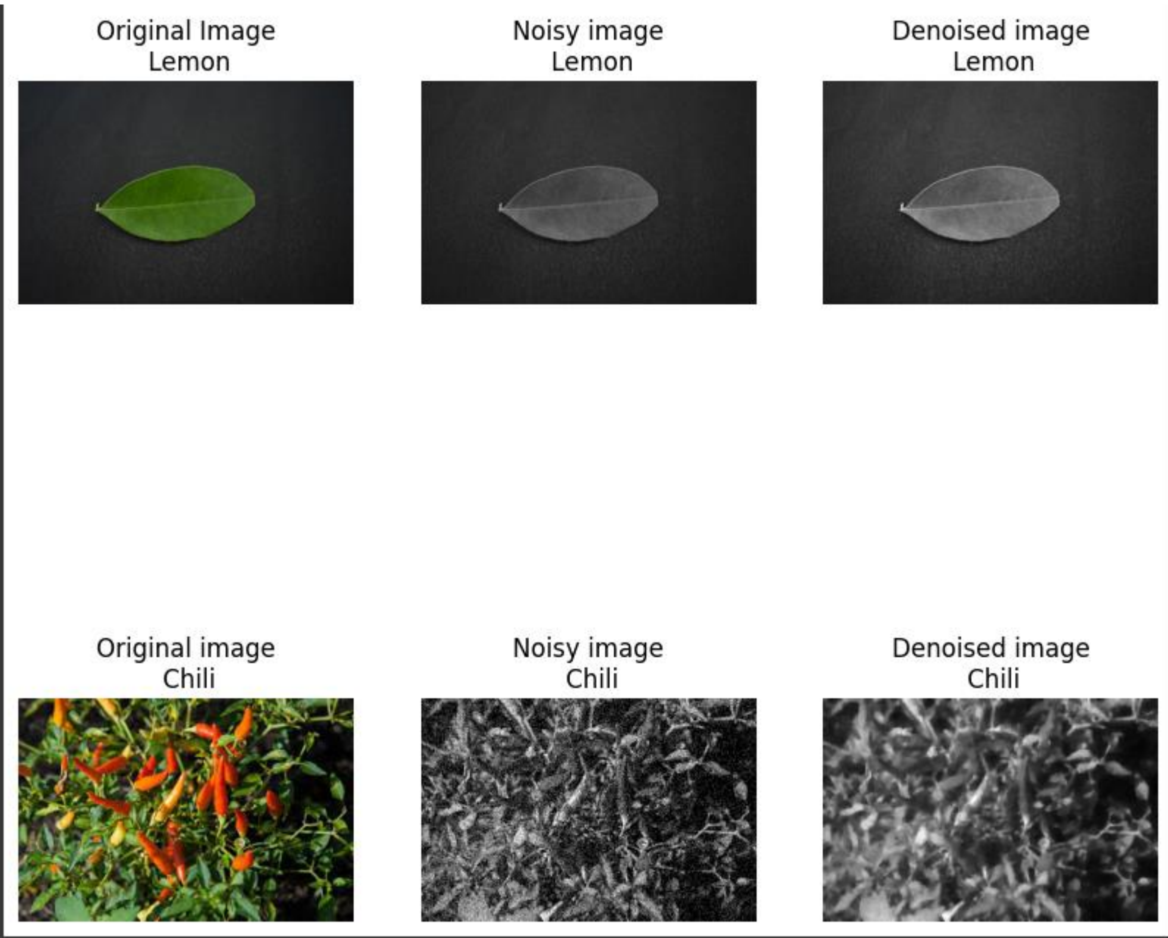
When the images are turned grey, they loose a lot of edge clarity making it harder to tell them apart. In the instances above they are easy to tell apart however, when applied to other images in the dataset it may make the model less confident.

Introducing noise to images

[ ]

```
img_path_1 = '/content/drive/MyDrive/ST1_Capstone/Lemon__healthy/0010_0001.JPG'
img_path_2 = '/content/drive/MyDrive/ST1_Capstone/Chili__healthy/Cabai_sehat005.jpg'
img_1 = cv2.imread(img_path_1)
img_2 = cv2.imread(img_path_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.)


fig, ((ax1, ax2,ax3), (ax4, ax5,ax6)) = plt.subplots(ncols=3,nrows=2,figsize=(10, 10))

ax1.imshow(cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB))
ax1.set_title('Original Image\n Lemon')
ax1.set_axis_off()

ax2.imshow(img_1_n, cmap ='gray')
ax2.set_title('Noisy image\n Lemon')
ax2.set_axis_off()

ax3.imshow(img_1_d, cmap = 'gray')
ax3.set_title('Denoised image\n Lemon')
ax3.set_axis_off()

ax4.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB))
ax4.set_title('Original image\n Chili')
ax4.set_axis_off()

ax5.imshow(img_2_n, cmap ='gray')
ax5.set_title('Noisy image\n Chili')
ax5.set_axis_off()

ax6.imshow(img_2_d, cmap = 'gray')
ax6.set_title('Denoised image\n Chili')
ax6.set_axis_off()
```
output



Again, similar to shifting to grey edge clarity is lost. This is made even more apparent when adding and removing noise from the image

Using canny edge detection to determine class

[17]

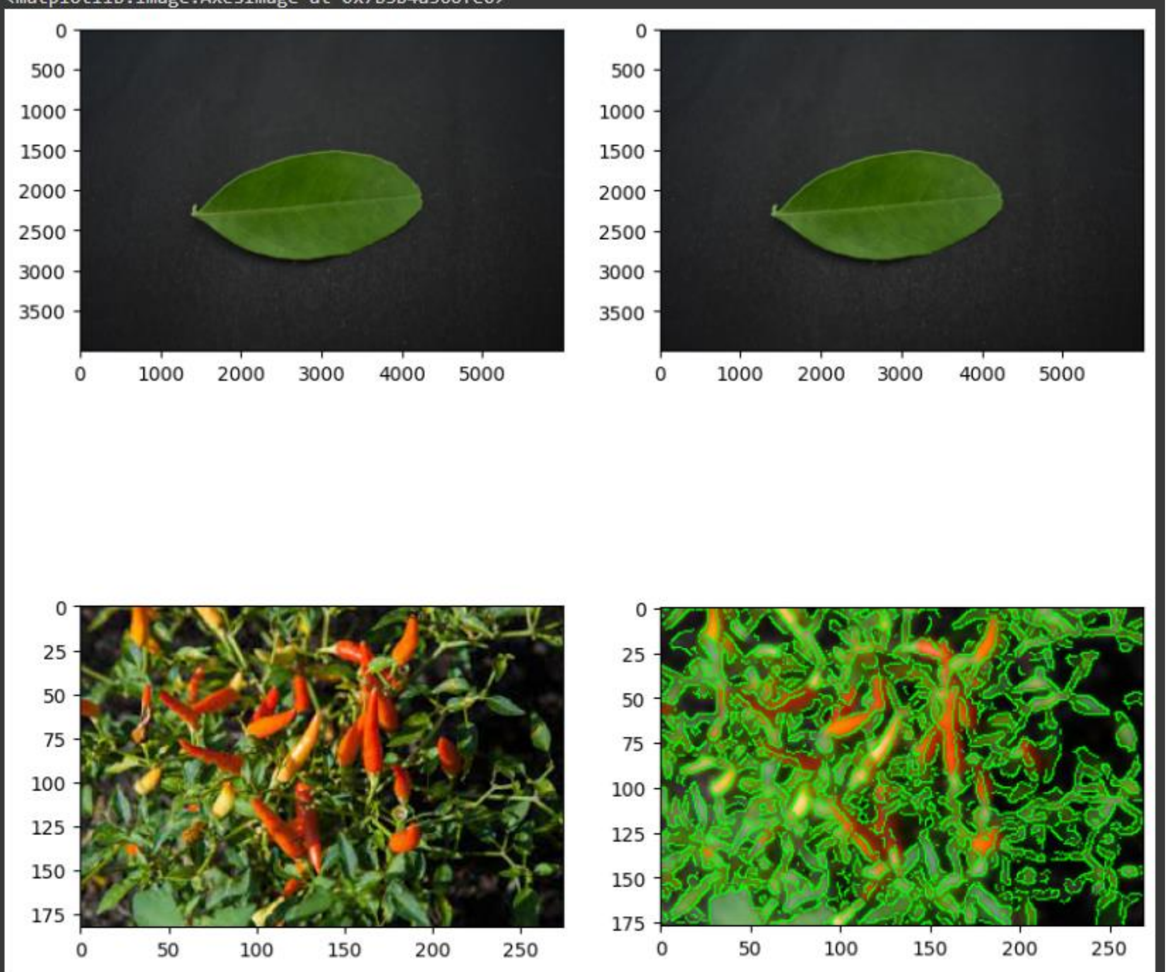
9s

```
img_path_1 = '/content/drive/MyDrive/ST1_Capstone/Lemon__healthy/0010_0001.JPG'
img_path_2 = '/content/drive/MyDrive/ST1_Capstone/Chili__healthy/Cabai_sehat005.jpg'
img_1 = cv2.imread(img_path_1)
img_2 = cv2.imread(img_path_2)

th1=30
th2=60
d=3

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)
```

```
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)

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))
```

output

```
<matplotlib.image.AxesImage at 0x7b3b4a366fe0>
```



The edge detection on the images is highlighted in green in this instance. As far as the chili is concerned the edges are clearly visable where as the leaf has a faint edge recognised along the top of the leaf. This means that the data model maybe more confident with some classes compared to others.

Predictive Data Analysis

[42]

3m

```
train = ImageDataGenerator(rescale = 1/255)
val = ImageDataGenerator(rescale = 1/255)
test = ImageDataGenerator(rescale= 1/255)
d1 = train.flow_from_directory('/content/gdrive/MyDrive/ST1 Capstone/train', target_size = (400,400),
                               batch_size = 5,
                               class_mode = "binary")
d2 = train.flow_from_directory('/content/gdrive/MyDrive/ST1 Capstone/val', target_size=(400,400),
                               batch_size=5,
                               class_mode = "binary")
d3 = train.flow_from_directory('/content/gdrive/MyDrive/ST1 Capstone/test',target_size=(400,400),
                               batch_size=5,
                               class_mode = "binary")
data_model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(16,(3,3),activation = 'relu',input_shape = (400,400,3))
                                         ,tf.keras.layers.MaxPool2D(2,2),
                                         ##
                                         tf.keras.layers.Conv2D(32,(3,3),activation = 'relu'),
                                         tf.keras.layers.MaxPool2D(2,2),
                                         ##
                                         tf.keras.layers.Conv2D(64,(3,3),activation = 'relu'),
                                         tf.keras.layers.MaxPool2D(2,2),
                                         ##
                                         tf.keras.layers.Flatten(),
                                         ##
                                         tf.keras.layers.Dense(512,activation="relu"),
                                         ##
                                         tf.keras.layers.Dense(1,activation='sigmoid')
```

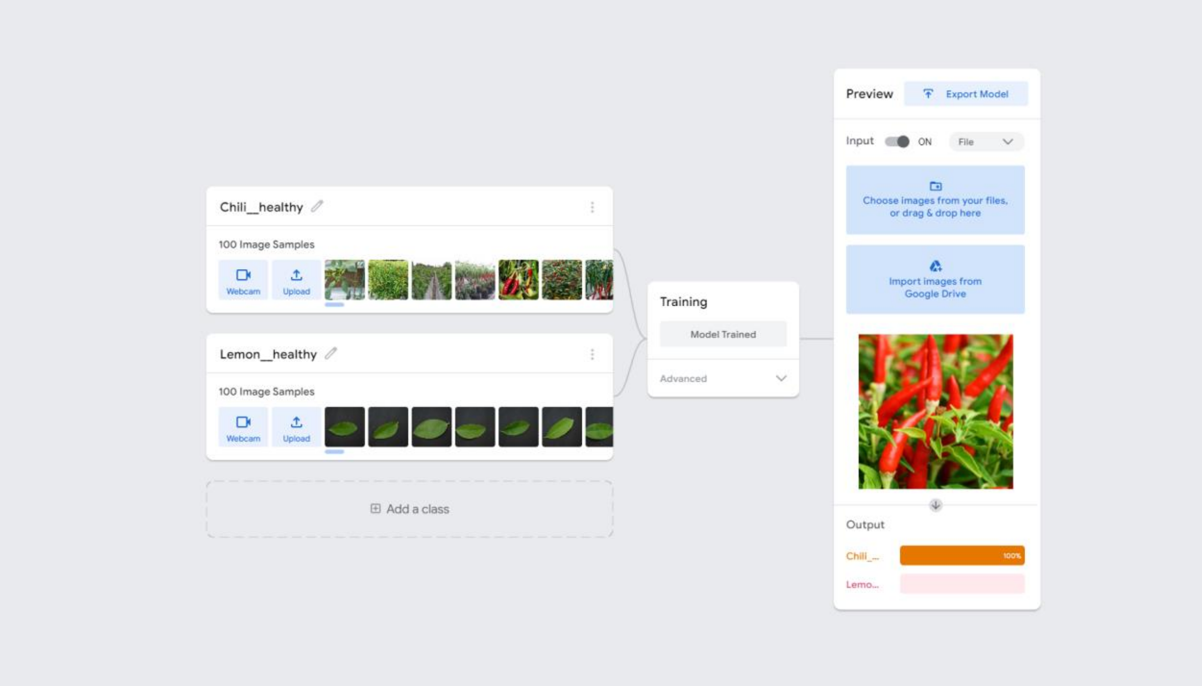
```
                                        ])
from keras.src.callbacks import History
data_model.compile(loss = "binary_crossentropy",
                   optimizer = RMSprop(lr=0.001),
                   metrics = ['accuracy'])
History=data_model.fit(d1,steps_per_epoch=5,epochs = 11,
               validation_data = d2)
```

output
Found 1958 images belonging to 20 classes.
Found 0 images belonging to 0 classes.
Found 0 images belonging to 0 classes.
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
Epoch 1/11
5/5 [==============================] - 18s 2s/step - loss: -27757.6875 - accuracy: 0.0800
Epoch 2/11
5/5 [==============================] - 15s 3s/step - loss: -1016610.6250 - accuracy: 0.1200
Epoch 3/11
5/5 [==============================] - 14s 3s/step - loss: -8761234.0000 - accuracy: 0.0000e+00
Epoch 4/11
5/5 [==============================] - 16s 3s/step - loss: -23822180.0000 - accuracy: 0.0000e+00
Epoch 5/11
5/5 [==============================] - 14s 3s/step - loss: -81140704.0000 - accuracy: 0.0400
Epoch 6/11
5/5 [==============================] - 14s 3s/step - loss: -176373904.0000 - accuracy: 0.0400
Epoch 7/11
5/5 [==============================] - 15s 3s/step - loss: -247174464.0000 - accuracy: 0.0400
Epoch 8/11
5/5 [==============================] - 17s 3s/step - loss: -369944896.0000 - accuracy: 0.0400
Epoch 9/11
5/5 [==============================] - 15s 3s/step - loss: -584582976.0000 - accuracy: 0.0000e+00
Epoch 10/11
5/5 [==============================] - 21s 4s/step - loss: -1088703360.0000 - accuracy: 0.0000e+00
Epoch 11/11
5/5 [==============================] - 14s 3s/step - loss: -1472765440.0000 - accuracy: 0.0400

Using the data generator the identify classes and images.

State 2 Using teachable machine to create data model:

(Example Teachable machine with 2 classes)



Teachable machine provides code for the model:

```
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
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("<IMAGE_PATH>").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]
```

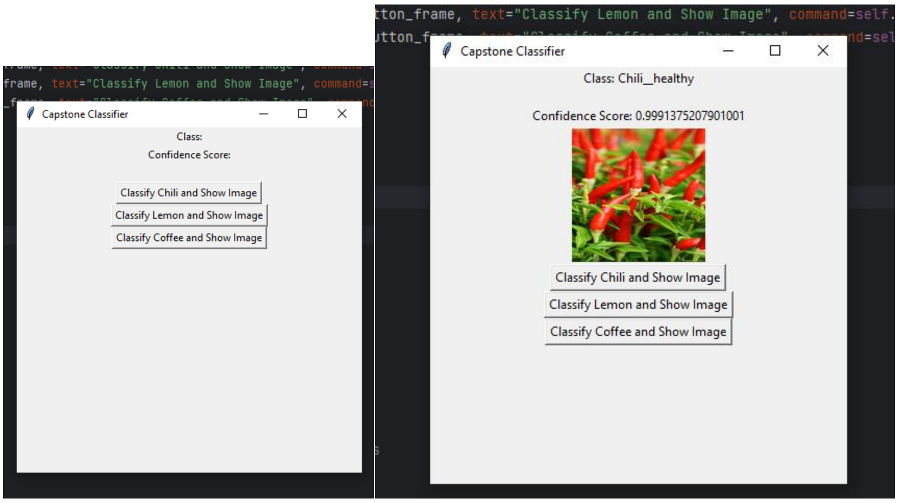
```python
confidence_score = prediction[0][index]

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

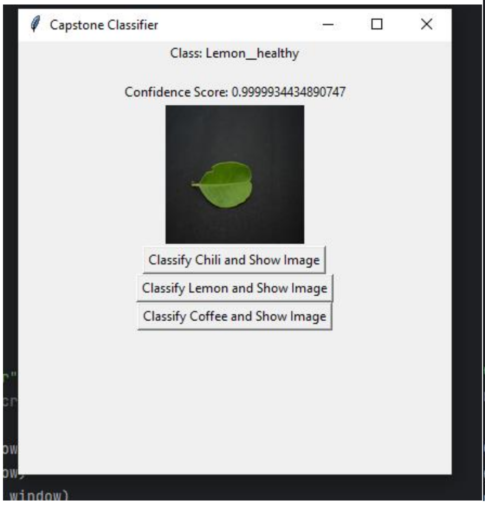
The console interface output looks like so:

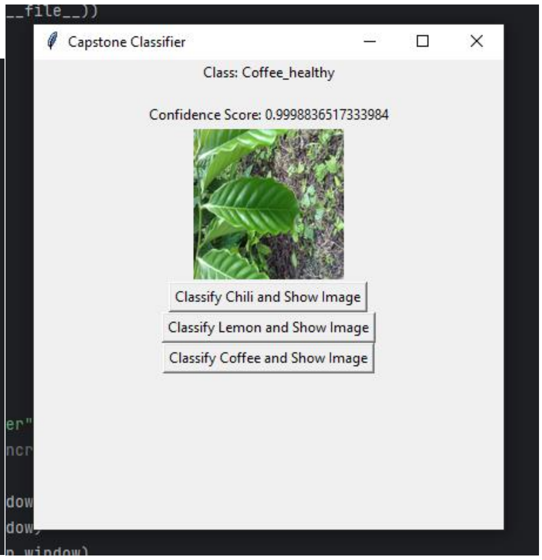
```
1/1 [==============================] - 0s 378ms/step
Class: Chili__healthy
Confidence Score: 0.9991375

Process finished with exit code 0
```

GUI implementation with tkinter:

Using the code from teachable machine. A tkinter GUI was built around 3 test images. The program looks like so:







GUI Code:

```
import tensorflow as tf
from keras.models import load_model
from PIL import Image, ImageTk, ImageOps
import numpy as np
import tkinter as tk
import os

project_folder = os.path.dirname(os.path.abspath(__file__))
# Create a GUI class
class Capstone_GUI:
    def chili_classification(self):
        # Disable scientific notation for clarity
        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
        data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

        # Replace this with the path to your image (escape backslashes)
        image_path = os.path.join(project_folder, "chili_test.jpg")
        image = Image.open(image_path).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.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]

        # Display the class and confidence score
        self.class_label.config(text=f'Class: {class_name[2:]}')
        self.confidence_label.config(text=f'Confidence Score:
{confidence_score}')

        # Display the image
        img = Image.open("chili_test.jpg")
        img = img.resize((128, 128), Image.LANCZOS)
        image = ImageTk.PhotoImage(img)
        self.imageLabel.configure(image=image)
        self.imageLabel.image = image

    def lemon_classification(self):
        # Disable scientific notation for clarity
        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
        data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

        # Replace this with the path to your image (escape backslashes)
        image_path = os.path.join(project_folder, "0010_0014.JPG")
        image = Image.open(image_path).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.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]

        # Display the class and confidence score
```

```
        self.class_label.config(text=f'Class: {class_name[2:]}')
        self.confidence_label.config(text=f'Confidence Score: 
{confidence_score}')

        # Display the image
        img = Image.open("0010_0014.JPG")
        img = img.resize((128, 128), Image.LANCZOS)
        image = ImageTk.PhotoImage(img)
        self.imageLabel.configure(image=image)
        self.imageLabel.image = image

    def coffee_classification(self):
        # Disable scientific notation for clarity
        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
        data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

        # Replace this with the path to your image (escape backslashes)
        image_path = os.path.join(project_folder, "C1P4H1.jpg")
        image = Image.open(image_path).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.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]

        # Display the class and confidence score
        self.class_label.config(text=f'Class: {class_name[2:]}')
        self.confidence_label.config(text=f'Confidence Score: 
{confidence_score}')

        # Display the image
        img = Image.open("C1P4H1.jpg")
        img = img.resize((128, 128), Image.LANCZOS)
        image = ImageTk.PhotoImage(img)
        self.imageLabel.configure(image=image)
        self.imageLabel.image = image
```

```
    def __init__(self):
        # Create the main window.
        self.main_window = tk.Tk()
        self.main_window.title("Capstone Classifier")
        self.main_window.geometry("400x400")  # Increased window height for
displaying the image

        self.image_frame = tk.Frame(self.main_window)
        self.class_frame = tk.Frame(self.main_window)
        self.confidence_frame = tk.Frame(self.main_window)
        self.button_frame = tk.Frame(self.main_window)

        # Create labels
        self.class_label = tk.Label(self.class_frame, text='Class:')
        self.confidence_label = tk.Label(self.confidence_frame,
text='Confidence Score:')
        self.imageLabel = tk.Label(self.image_frame)

        # Create a single button for both functions
        self.chili_button = tk.Button(self.button_frame, text="Classify
Chili and Show Image", command=self.chili_classification)
        self.lemon_button = tk.Button(self.button_frame, text="Classify
Lemon and Show Image", command=self.lemon_classification)
        self.coffee_button = tk.Button(self.button_frame, text="Classify
Coffee and Show Image", command=self.coffee_classification)

        # Pack labels, button, and frames
        self.class_label.pack()
        self.confidence_label.pack()
        self.imageLabel.pack(side=tk.TOP)
        self.chili_button.pack(side=tk.TOP)
        self.lemon_button.pack(side=tk.TOP)
        self.coffee_button.pack(side=tk.TOP)

        self.class_frame.pack()
        self.confidence_frame.pack()
        self.image_frame.pack()
        self.button_frame.pack()

        # Start the GUI main loop
        self.main_window.mainloop()

# Create an instance of the Capstone_GUI class
capstone_gui = Capstone_GUI()
```

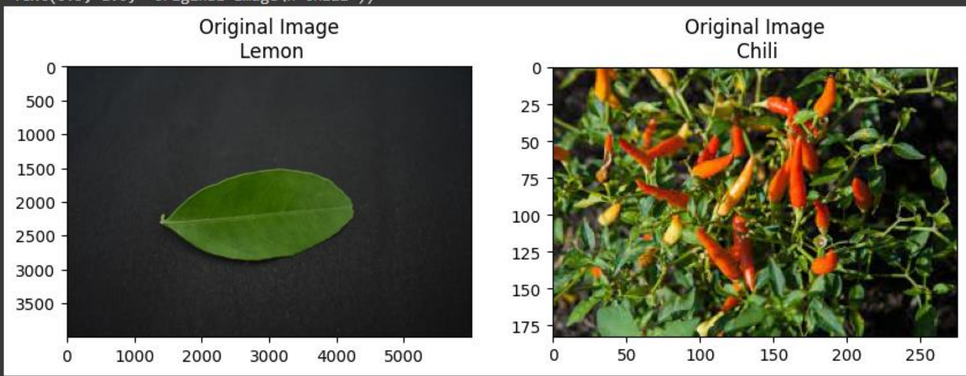
Journal

Week 10:

Started by culling the data down to the healthy classes and then having only 100 photos per class. I then performed various tests for the EDA on the data. The first test was getting the files read from google drive:

```python
#Read and Display Images

img_path_1 = '/content/drive/MyDrive/ST1_Capstone/Lemon__healthy/0010_0001.JPG'
img_path_2 = '/content/drive/MyDrive/ST1_Capstone/Chili__healthy/Cabai_sehat005.jpg'
img_1 = cv2.imread(img_path_1)
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 Lemon')
plt.subplot(122)
plt.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB)),plt.title('Original Image\n Chili')
```

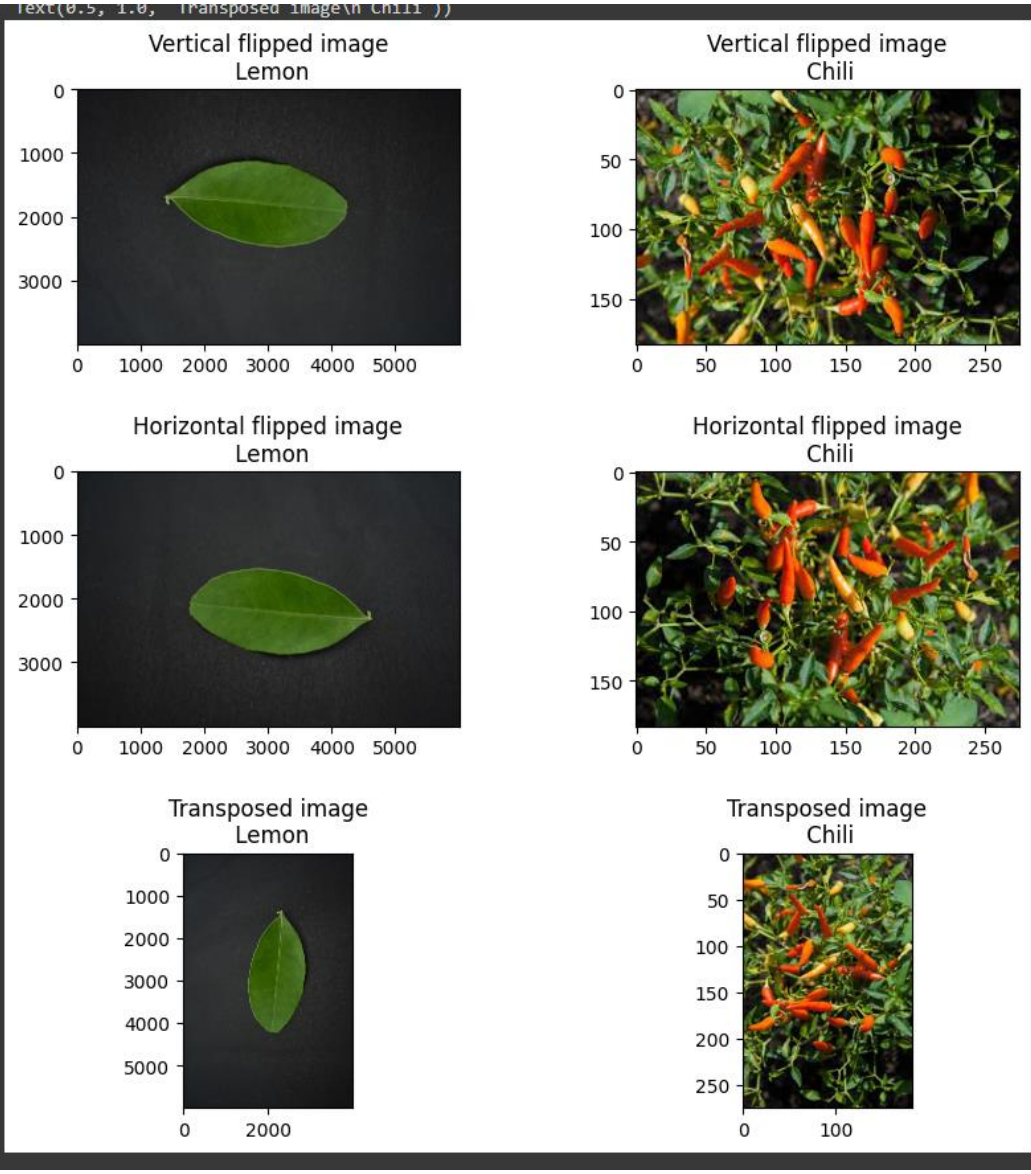
```
(<matplotlib.image.AxesImage at 0x78277cb03010>,
 Text(0.5, 1.0, 'Original Image\n Chili'))
```



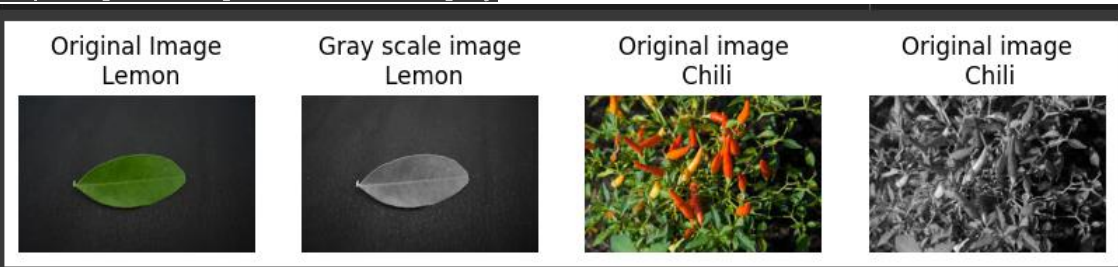
Various other tests were conducted including:

Gemoetric transformation analysis of images

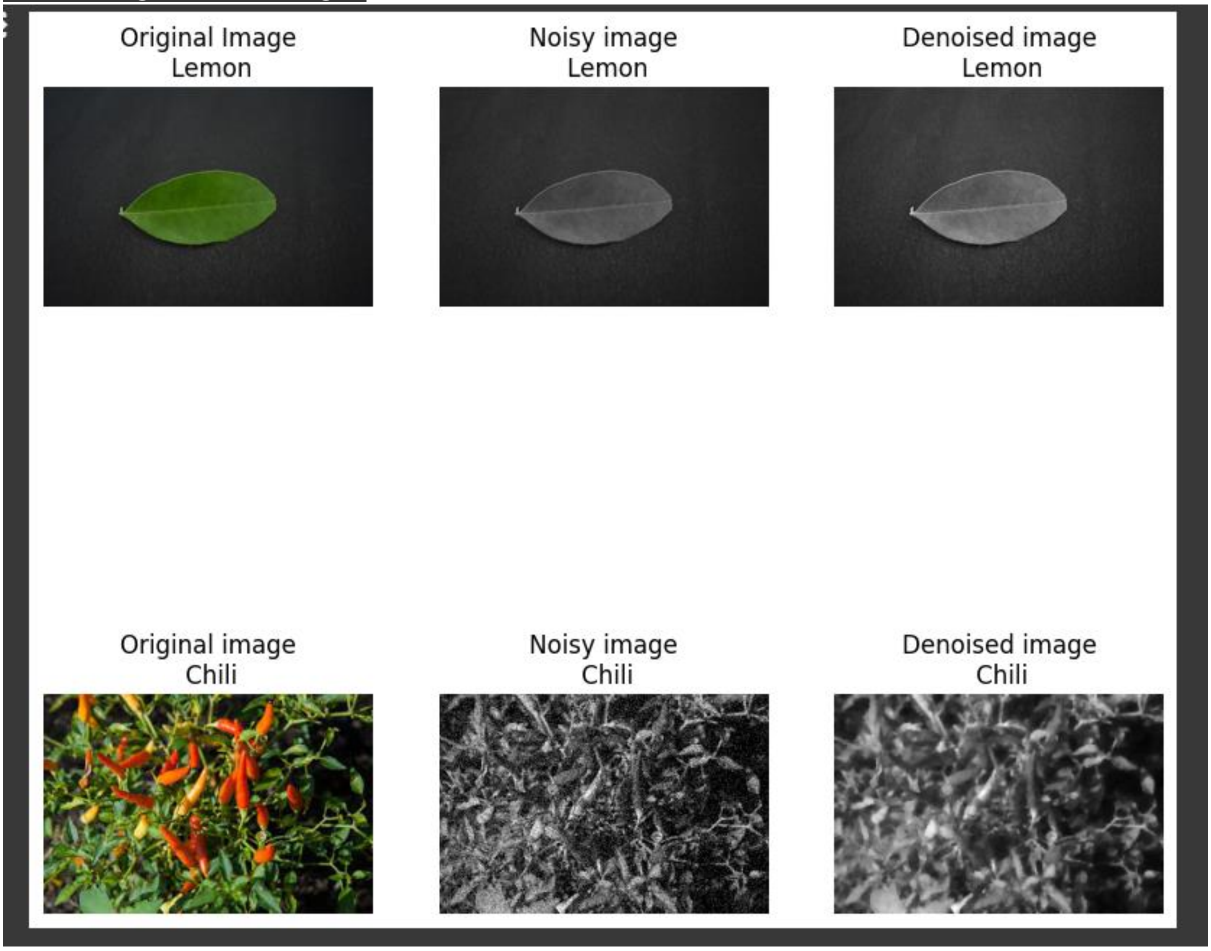
```
Text(0.5, 1.0, 'Transposed image\n Chili')
```



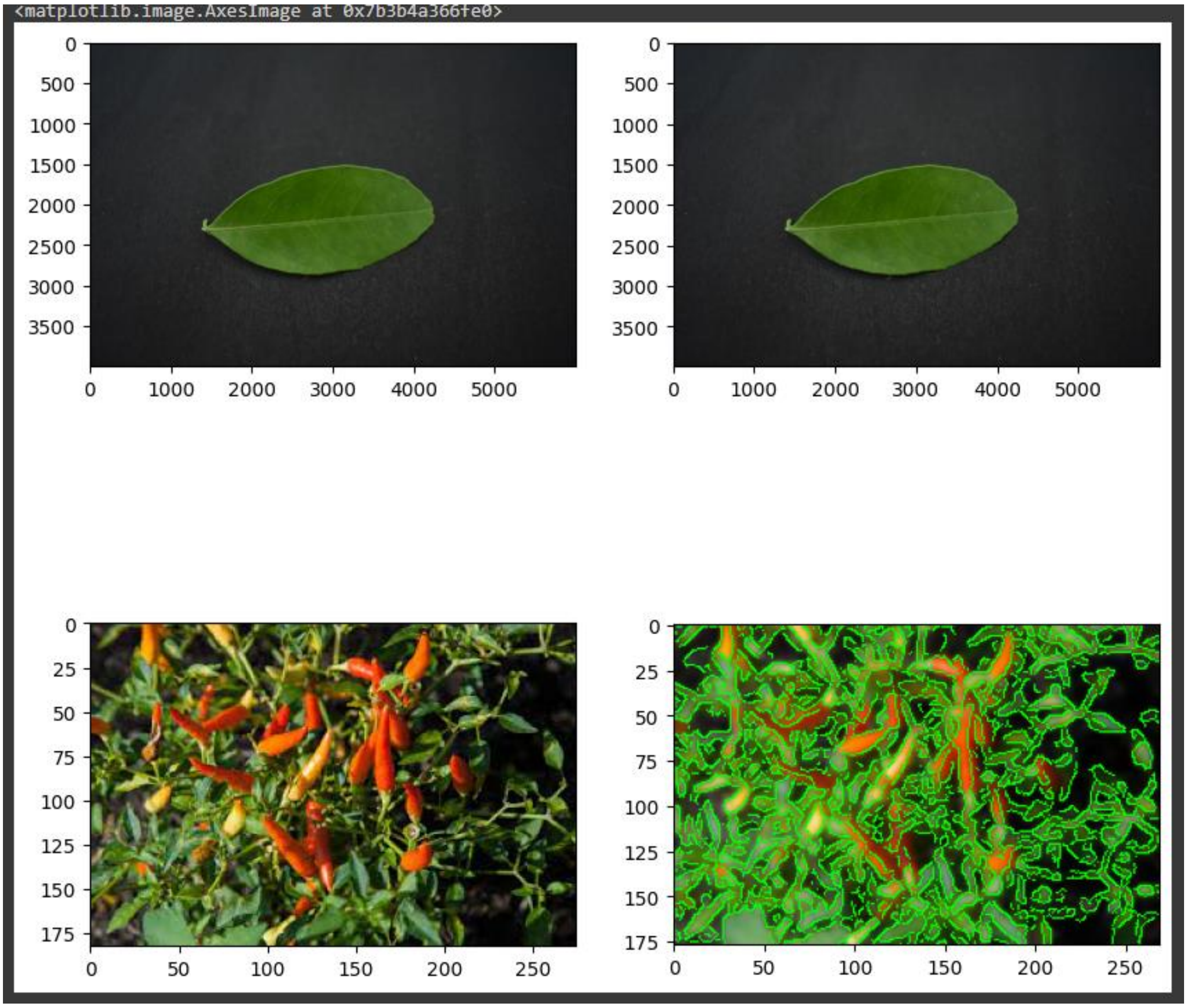
Comparing the images when turned grey



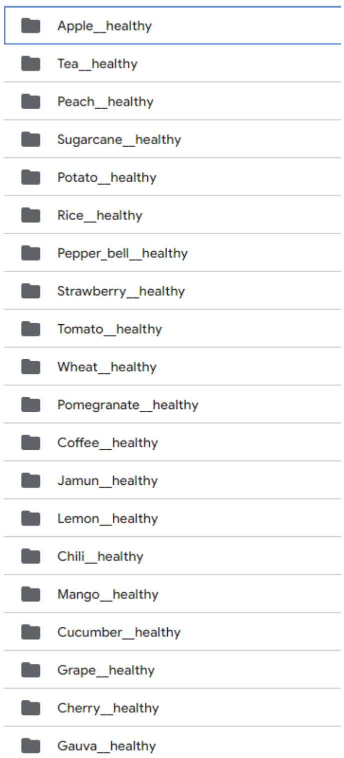
Introducing noise to images



Using canny edge detection to determine class



These tests allowed me to see how the program would interpret the images.



Week 11:

Using teachable machine, I created a model using the most of the following classes to the left of this text. (unfortunately my internet is too slow to be able to process all of the data) this data was then placed into a console program which allowed for an image to be placed into the program and it would print the class and the confidence score. I also performed predictive data analysis using the data image generator using tensorflow.

Predictive Data Analysis

```
train = ImageDataGenerator(rescale = 1/255)
val = ImageDataGenerator(rescale = 1/255)
test = ImageDataGenerator(rescale= 1/255)

d1 = train.flow_from_directory('/content/gdrive/MyDrive/ST1_Capstone/train', target_size = (400,400),
                               batch_size = 5,
                               class_mode = "binary")

d2 = train.flow_from_directory('/content/gdrive/MyDrive/ST1_Capstone/val', target_size=(400,400),
                               batch_size=5,
                               class_mode = "binary")
d3 = train.flow_from_directory('/content/gdrive/MyDrive/ST1_Capstone/test',target_size=(400,400),
                               batch_size=5,
                               class_mode = "binary")
data_model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(16,(3,3),activation = 'relu',input_shape
                                     ,tf.keras.layers.MaxPool2D(2,2),
                                     ##
                                     tf.keras.layers.Conv2D(32,(3,3),activation = 'relu'),
                                     tf.keras.layers.MaxPool2D(2,2),
                                     ##
                                     tf.keras.layers.Conv2D(64,(3,3),activation = 'relu'),
                                     tf.keras.layers.MaxPool2D(2,2),
                                     ##
                                     tf.keras.layers.Flatten(),
                                     ##
                                     tf.keras.layers.Dense(512,activation="relu"),
                                     ##
                                     tf.keras.layers.Dense(1,activation='sigmoid')
                                     ])
from keras.src.callbacks import History

data_model.compile(loss = "binary_crossentropy",
                   optimizer = RMSprop(lr=0.001),
                   metrics = ['accuracy'])
History=data_model.fit(d1,steps_per_epoch=5,epochs = 11,
               validation_data = d2)
```

```
Found 1958 images belonging to 20 classes.
Found 0 images belonging to 0 classes.
Found 0 images belonging to 0 classes.
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
Epoch 1/11
5/5 [==============================] - 18s 2s/step - loss: -27757.6875 - accuracy: 0.0800
Epoch 2/11
5/5 [==============================] - 15s 3s/step - loss: -1016610.6250 - accuracy: 0.1200
Epoch 3/11
5/5 [==============================] - 14s 3s/step - loss: -8761234.0000 - accuracy: 0.0000e+00
Epoch 4/11
5/5 [==============================] - 16s 3s/step - loss: -23822180.0000 - accuracy: 0.0000e+00
Epoch 5/11
5/5 [==============================] - 14s 3s/step - loss: -81140704.0000 - accuracy: 0.0400
Epoch 6/11
5/5 [==============================] - 14s 3s/step - loss: -176373904.0000 - accuracy: 0.0400
Epoch 7/11
5/5 [==============================] - 15s 3s/step - loss: -247174464.0000 - accuracy: 0.0400
Epoch 8/11
5/5 [==============================] - 17s 3s/step - loss: -369944896.0000 - accuracy: 0.0400
Epoch 9/11
5/5 [==============================] - 15s 3s/step - loss: -584582976.0000 - accuracy: 0.0000e+00
Epoch 10/11
5/5 [==============================] - 21s 4s/step - loss: -1088703360.0000 - accuracy: 0.0000e+00
Epoch 11/11
5/5 [==============================] - 14s 3s/step - loss: -1472765440.0000 - accuracy: 0.0400
```

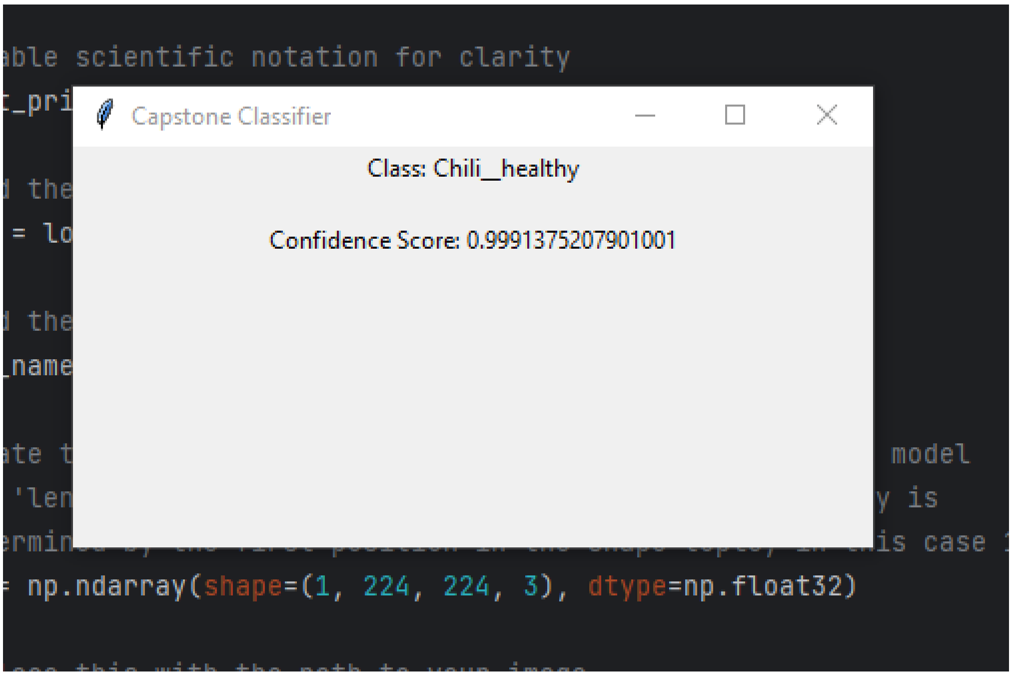
Week 12:

Using the teachable machine file, I was able to successfully use an image of a healthy chili to get an output in the console: I also developed a basic GUI that displayed the class and confidence score.

```
import tensorflow
import keras
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
import tkinter as tk
from tkinter import messagebox

# Disable scientific notation for clarity
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\alexa\Desktop\School\Sem 2 2023\Capstone
stage 2\chili_test.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.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)
# Create a GUI class
class Capstone_GUI:
    def __init__(self):
        # Create the main window.
        self.main_window = tk.Tk()
        self.main_window.title("Capstone Classifier")
        self.main_window.geometry("400x200")

        self.class_frame = tk.Frame(self.main_window)
        self.confidence_frame = tk.Frame(self.main_window)

        # Create labels for class and confidence
        self.class_label = tk.Label(self.class_frame, text=f'Class:
{class_name[2:]}')
```

```
        self.confidence_label = tk.Label(self.confidence_frame, 
text=f'Confidence Score: {confidence_score}')

        # Pack labels into frames
        self.class_label.pack()
        self.confidence_label.pack()

        # Pack frames into the main window
        self.class_frame.pack()
        self.confidence_frame.pack()

        # Start the GUI main loop
        self.main_window.mainloop()

# Create an instance of the Capstone_GUI class
capstone_gui = Capstone_GUI()
```



Week 13:

The GUI has been upgraded significantly so that it can explore other images as well as display the image within the GUI along with its class and confidence score:

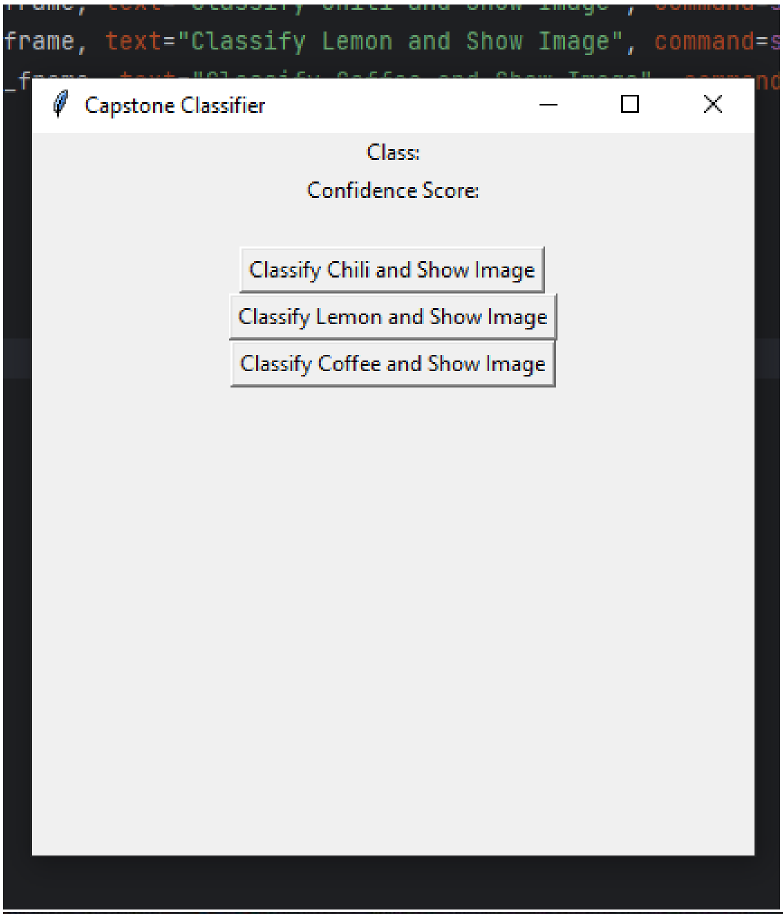
```
import tensorflow as tf
from keras.models import load_model
from PIL import Image, ImageTk, ImageOps
import numpy as np
import tkinter as tk
import os

project_folder = os.path.dirname(os.path.abspath(__file__))
# Create a GUI class
class Capstone_GUI:
    def chili_classification(self):
        # Disable scientific notation for clarity
```

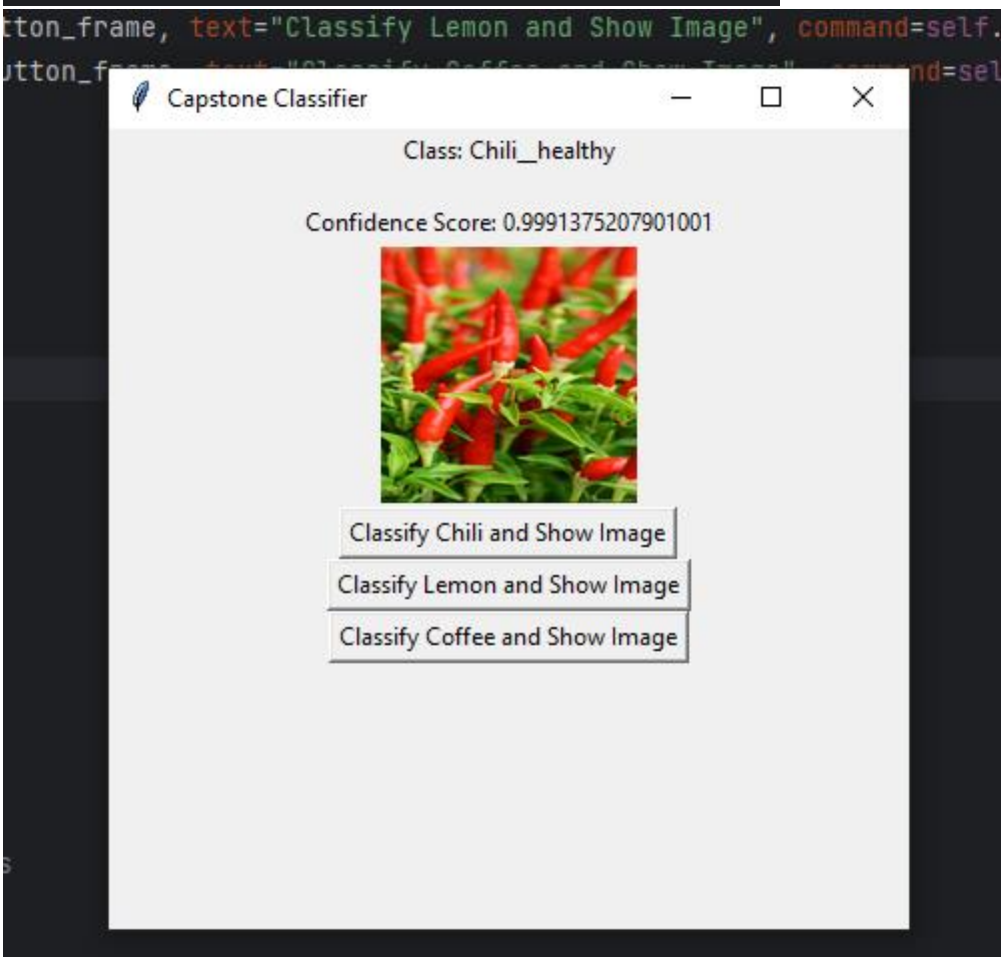
```
        np.set_printoptions(suppress=True)

        # 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
        data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

        # Replace this with the path to your image (escape backslashes)
        image_path = os.path.join(project_folder, "chili_test.jpg")
        image = Image.open(image_path).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.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]

        # Display the class and confidence score
        self.class_label.config(text=f'Class: {class_name[2:]}')
        self.confidence_label.config(text=f'Confidence Score:
{confidence_score}')

        # Display the image
        img = Image.open("chili_test.jpg")
        img = img.resize((128, 128), Image.LANCZOS)
        image = ImageTk.PhotoImage(img)
        self.imageLabel.configure(image=image)
        self.imageLabel.image = image

    def lemon_classification(self):
        # Disable scientific notation for clarity
        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
        data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
```

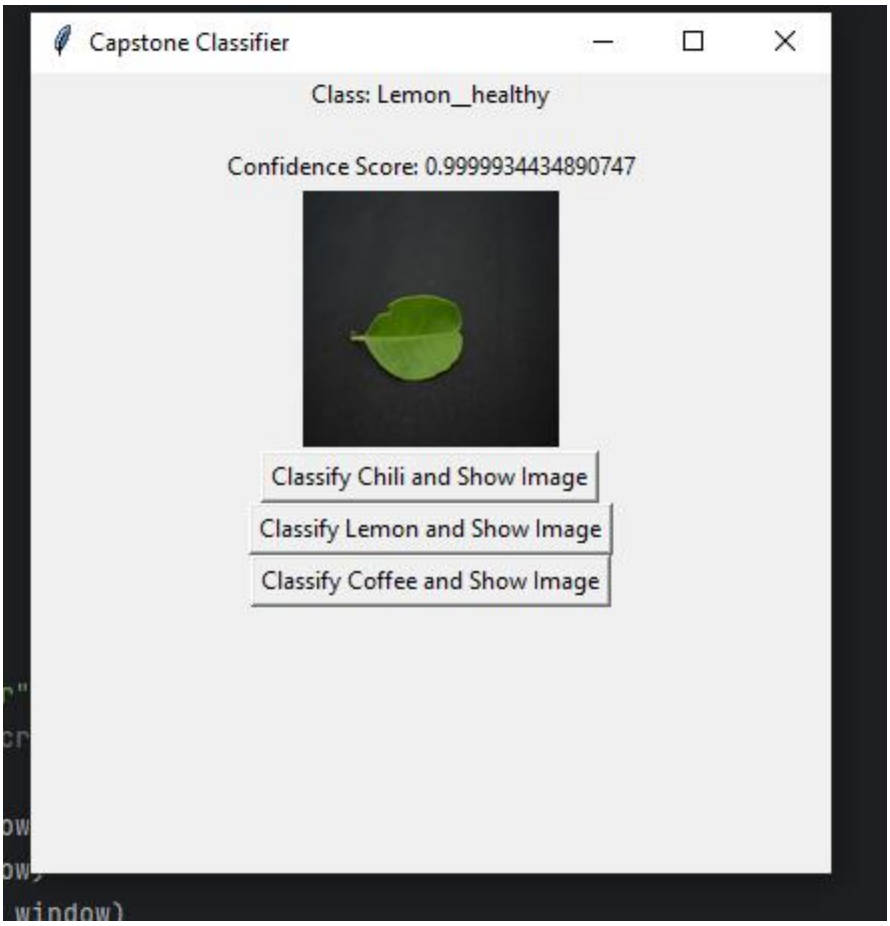
```
        # Replace this with the path to your image (escape backslashes)
        image_path = os.path.join(project_folder, "0010_0014.JPG")
        image = Image.open(image_path).convert("RGB")

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

        # Display the class and confidence score
        self.class_label.config(text=f'Class: {class_name[2:]}')
        self.confidence_label.config(text=f'Confidence Score:
{confidence_score}')

        # Display the image
        img = Image.open("0010_0014.JPG")
        img = img.resize((128, 128), Image.LANCZOS)
        image = ImageTk.PhotoImage(img)
        self.imageLabel.configure(image=image)
        self.imageLabel.image = image

    def coffee_classification(self):
        # Disable scientific notation for clarity
        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
        data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

        # Replace this with the path to your image (escape backslashes)
        image_path = os.path.join(project_folder, "C1P4H1.jpg")
        image = Image.open(image_path).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.LANCZOS)

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

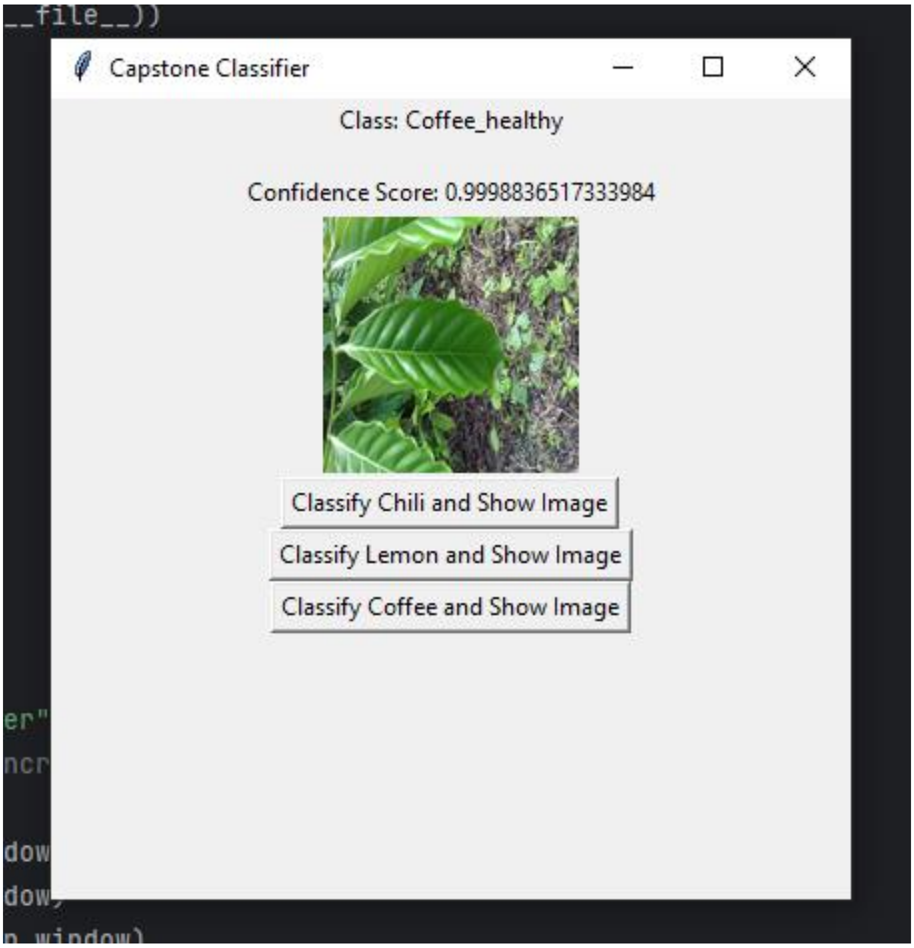
```
        # Normalize the image
        normalized_image_array = (image_array.astype(np.float32) / 127.5) -
1

        # 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]

        # Display the class and confidence score
        self.class_label.config(text=f'Class: {class_name[2:]}')
        self.confidence_label.config(text=f'Confidence Score:
{confidence_score}')

        # Display the image
        img = Image.open("C1P4H1.jpg")
        img = img.resize((128, 128), Image.LANCZOS)
        image = ImageTk.PhotoImage(img)
        self.imageLabel.configure(image=image)
        self.imageLabel.image = image

    def __init__(self):
        # Create the main window.
        self.main_window = tk.Tk()
        self.main_window.title("Capstone Classifier")
        self.main_window.geometry("400x400")  # Increased window height for
displaying the image

        self.image_frame = tk.Frame(self.main_window)
        self.class_frame = tk.Frame(self.main_window)
        self.confidence_frame = tk.Frame(self.main_window)
        self.button_frame = tk.Frame(self.main_window)

        # Create labels
        self.class_label = tk.Label(self.class_frame, text='Class:')
        self.confidence_label = tk.Label(self.confidence_frame,
text='Confidence Score:')
        self.imageLabel = tk.Label(self.image_frame)

        # Create a single button for both functions
        self.chili_button = tk.Button(self.button_frame, text="Classify
Chili and Show Image", command=self.chili_classification)
        self.lemon_button = tk.Button(self.button_frame, text="Classify
Lemon and Show Image", command=self.lemon_classification)
        self.coffee_button = tk.Button(self.button_frame, text="Classify
Coffee and Show Image", command=self.coffee_classification)

        # Pack labels, button, and frames
        self.class_label.pack()
        self.confidence_label.pack()
        self.imageLabel.pack(side=tk.TOP)
        self.chili_button.pack(side=tk.TOP)
        self.lemon_button.pack(side=tk.TOP)
        self.coffee_button.pack(side=tk.TOP)

        self.class_frame.pack()
```

```
        self.confidence_frame.pack()
        self.image_frame.pack()
        self.button_frame.pack()

        # Start the GUI main loop
        self.main_window.mainloop()

# Create an instance of the Capstone_GUI class
capstone_gui = Capstone_GUI()
```


frame, text="Classify Lemon and Show Image", command=s
Capstone Classifier
Class:
Confidence Score:
Classify Chili and Show Image
Classify Lemon and Show Image
Classify Coffee and Show Image


tton_frame, text="Classify Lemon and Show Image", command=self.
Capstone Classifier
Class: Chili__healthy
Confidence Score: 0.9991375207901001
Classify Chili and Show Image
Classify Lemon and Show Image
Classify Coffee and Show Image


Capstone Classifier
Class: Lemon__healthy
Confidence Score: 0.9999934434890747
Classify Chili and Show Image
Classify Lemon and Show Image
Classify Coffee and Show Image



References:

[1] A. Dobrovsky, "Plant Disease Classification merged dataset," Kaggle, https://www.kaggle.com/datasets/alinedobrovsky/plant-disease-classification-merged-dataset (accessed Oct. 29, 2023).