**Software Tech Report Capstone Project**

**Student ID: u3260636**

**Student Name: Kuenga Palden**

**Unit Name: Software Tech**

**Unit Number: 4483**

**Unit tutor: Linda Ma**

**Assignment Name: ST1 Capstone Project – Semester 2 2023**

**Due date: 29/10/2023**

**Date Submitted: 29/10/2023.**

Student declaration

I certify that the attached assignment is my work. Material drawn from other sources has been  
appropriately and fully acknowledged as to author/creator, source, and other bibliographic details.  
Signature of student: KP Date: 20/10/2023.

**Table of Contents**

Table of Contents

[Introduction and Background 3](#_Toc148043639)

[Dataset Description 3](#_Toc148043640)

[Methodology 4](#_Toc148043641)

[Stage 1: Exploratory Data Analysis Stage 4](#_Toc148043642)

[Exploratory Data Analysis 6](#_Toc148043643)

[Stage 1: EDA (Exploratory Data Analytics) 17](#_Toc148043644)

[Stage 2: PDA (Predictive Data Analytics) 18](#_Toc148043645)

[Stage 2 PDA with Teachable Machine 19](#_Toc148043647)

[Stage 3A Deployment/Implemenation with Teachable Machine withGoogle 22](#_Toc148043648)

[Stage 3B Deployment/Implemenation with TkInter 26](#_Toc148043649)

[Conclusions 27](#_Toc148043650)

[References 27](#_Toc148043651)

[Log Book 27](#_Toc148043652)

# Introduction and Background

This report describes the details of Python Capstone Project for ST1 unit within the scope of the project requirements provided in the assignment handout. The allocated dataset for this particular capstone project is a binary skin classification dataset available from the publicly published dataset website Kaggle.

Skin Cancer Image Dataset is a comprehensive collection of JPEG files designed for binary classification tasks. It contains a diverse set of images, each of which is categorized as either cancerous or non-cancerous skin conditions. This dataset is invaluable for researchers, medical professionals, and machine learning practitioners who are interested in developing and evaluating models for skin cancer detection. It enables medical professionals to easily identify the more serious cancerous cases, to identify and prevent any further damage to the skin / cause further infection in the affected area.

The capstone project aims to present the findings in the form of a data-driven scientific approach, which involves EDA (Exploratory Data Analysis), PDA (Predictive Analytics Development), and implementation in the form of tinker, streamilt, or Flask. By using this Scientific data-driven approach we can classify and identify which of these images present a cancer and see if the accuracy is high/low.

# Dataset Description

The dataset is available publicly in Kaggle licensed by CC0: Public Domain and is updated/uploaded by Kyle Graupe (02/2023)

**Data Description:**

Type of data: JPG Varies between each image (200 x 200, 380 x 380, 500 x 550, 1090 x 1180) Data format: JPG.

Number of Images: A total of 288 Images (Non-Cancer Testing folder containing 162 Images)

Number of classes: 2 classes in skin data with an additional 2 folders within cancer or non-cancerous each labelled training and testing

Applicability: The dataset is suitable for distinguishing the differences between cancerous and non-cancerous skin diseases (two-class prediction).

# Methodology

The methodology used for developing the software platform involves 3 stages as outlined below:

Stage 1: Exploratory Data Analysis for leaf images from the dataset (Google Collab/PyCharm (Python Console App).

Stage 2: Predictive analytics development using machine learning platform/tools (using teachable machine learning)

Stage 3: Implementation and Deployment of the software technology tool for real-world field testing (using PyCharm IDE (Python Console App/Python Skinter GUI App)

## 

## Stage 1: Exploratory Data Analysis Stage

Stage 1 is the most important preliminary stage, and the purpose of exploratory data analysis is to obtain a thorough understanding of data and inform about the choice of predictive analytics algorithms to be used, and the expected performance of the software tool in real-world settings.

### Exploratory Data Analysis

The first phase of the software development activity involved understanding the data, basic exploratory data analysis, and visualization. Google Collab was chosen as the experimental environment. Before the exploratory data analysis can begin, some of the steps required are:

We downloaded the dataset that was allocated to us through the Kaggle link below:

<https://www.kaggle.com/datasets/kylegraupe/skin-cancer-binary-classification-dataset>

We used Google Collab for Our EDA; the steps below outline the step by step for the EDA analysis:

1. Upload the Data to Google Drive.
2. We then must link the drive with the Google collab.
3. Then we must import libraries that perform EDA.
4. We then must move/change the working folder (Assignment) to Google Drive where we linked the Google Collab notebook.
5. The steps above can be done using the following codes.

# from google collab we import the drive

from google.colab import drive

drive.mount('/content/gdrive')

# Then we import the os environment to the drive

import os

os.environ['KAGGLE\_CONFIG\_DIR'] = "content/gdrive/MyDrive/Kaggle\_Assesment"

# We can check and see if the os is in the right environment

%cd /content/gdrive/MyDrive/Kaggle\_Assesment

/content/gdrive/MyDrive/Kaggle\_Assesment

# We download the dataset directly within Kaggle using the Kaggle link.

!kaggle datasets download -d kylegraupe/skin-cancer-binary-classification-dataset

Downloading skin-cancer-binary-classification-dataset.zip to /content/gdrive/MyDrive/Kaggle\_Assesment

80% 18.0M/22.5M [00:00<00:00, 89.9MB/s]

100% 22.5M/22.5M [00:00<00:00, 88.0MB/s]

# We then unzip the downloaded dataset.

!unzip \\*.zip  && rm \*.zip

Archive: skin-cancer-binary-classification-dataset.zip

replace Skin\_Data/Cancer/Testing/1714-02.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename:

# Install TensorFlow (very important)

!pip install tensorflow==2.9.1

# We then have to import the following libraries.

import os

import time

import shutil

import pathlib

import itertools

import cv2

import numpy as np

import pandas as pd

import seaborn as sns

sns.set\_style('darkgrid')

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, classification\_report

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import Adam, Adamax

from tensorflow.keras.metrics import categorical\_crossentropy

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import (Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, BatchNormalization)

from tensorflow.keras import regularizers

# We must then import ignore warnings

import warnings

warnings.filterwarnings("ignore")

print ('modules loaded')

modules loaded

# We must write code that reads data and stores it in the data frame using the link to the working folder

data\_dir = '/content/gdrive/My Drive/Kaggle\_Assesment/Skin\_Data'

filepaths = []

labels = []

folds = os.listdir(data\_dir)

for fold in folds:

    foldpath = os.path.join(data\_dir, fold)

    filelist = os.listdir(foldpath)

    for file in filelist:

        fpath = os.path.join(foldpath, file)

        filepaths.append(fpath)

        labels.append(fold)

# We then concatenate the data paths with labels into one dataframe

Fseries = pd.Series(filepaths, name= 'filepaths')

Lseries = pd.Series(labels, name='labels')

df = pd.concat([Fseries, Lseries], axis= 1)

# We can then finally proceed to do the 5 EDA Questions:

#EDA Q1:How is the data distribution

# After the steps above are followed we can check the data distribution

df.head(5)

|  |  |  |
| --- | --- | --- |
| **0** | /content/gdrive/My Drive/Kaggle\_Assesment/Skin... | Cancer |
| **1** | /content/gdrive/My Drive/Kaggle\_Assesment/Skin... | Cancer |
| **2** | /content/gdrive/My Drive/Kaggle\_Assesment/Skin... | Non\_Cancer |
| **3** | /content/gdrive/My Drive/Kaggle\_Assesment/Skin... | Non\_Cancer |

# This showcases the following data folder and the distribution

# We can even do the tail end of the data distribution

df.tail(5)

|  |  |  |
| --- | --- | --- |
| **0** | /content/gdrive/My Drive/Kaggle\_Assesment/Skin... | Cancer |
| **1** | /content/gdrive/My Drive/Kaggle\_Assesment/Skin... | Cancer |
| **2** | /content/gdrive/My Drive/Kaggle\_Assesment/Skin... | Non\_Cancer |
| **3** | /content/gdrive/My Drive/Kaggle\_Assesment/Skin... | Non\_Cancer |

# The data distribution is the same because the data folders are small and not high in number.

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 4 entries, 0 to 3

Data columns (total 2 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 filepaths 4 non-null object

1 labels 4 non-null object

dtypes: object(2)

memory usage: 192.0+ bytes

# This code outlines the 4 non-null folders in the file paths and the labels section and shows how much memory usage it consumes.

#EDA Q2: How do images from different classes look like (Read and Display Images)

# To find out and display the two images in the different folders we have to write the following code:

import cv2

import matplotlib.pyplot as plt

%matplotlib inline

img\_path\_1 ='/content/gdrive/MyDrive/Kaggle\_Assesment/Skin\_Data/Cancer/Training/1703.JPG'

img\_1 = cv2.imread(img\_path\_1)

img\_path\_2 ='/content/gdrive/My Drive/Kaggle\_Assesment/Skin\_Data/Non\_Cancer/Testing/2566-03.JPG'

img\_2 = cv2.imread(img\_path\_2)

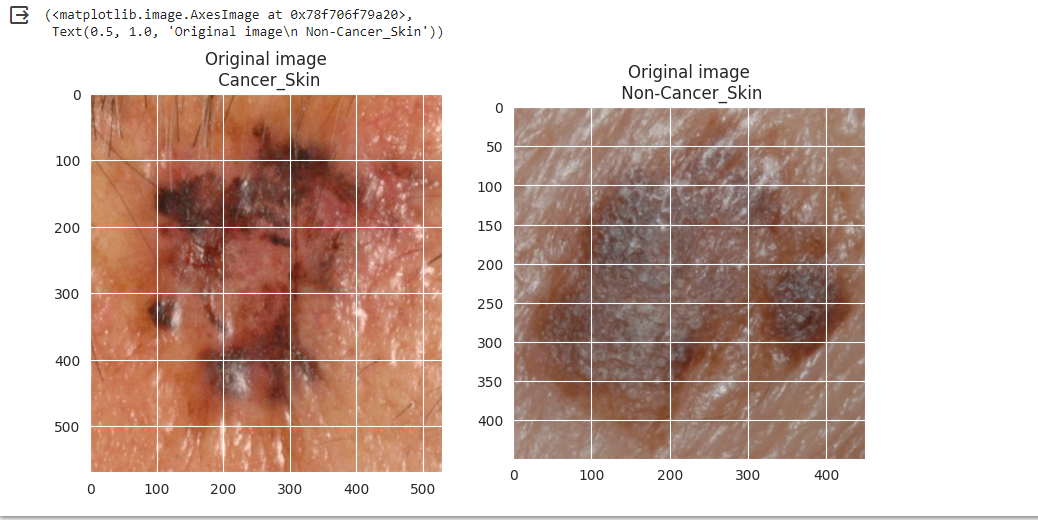
plt.figure(figsize=(10, 10))

plt.subplot(121)

plt.imshow(cv2.cvtColor(img\_1, cv2.COLOR\_BGR2RGB)),plt.title('Original image\n Cancer\_Skin')

plt.subplot(122)

plt.imshow(cv2.cvtColor(img\_2, cv2.COLOR\_BGR2RGB)),plt.title('Original image\n Non-Cancer\_Skin')



# Thus, the images are both being displayed one from each of the classification (Cancer and Non cancer folders)

# EDA Q3 - How does the images from different classes look like with geometrical transformations (vertical flipping, horizontal flipping, transposing)

# To answer this EDA Question, we must input the following code

import cv2

import matplotlib.pyplot as plt

%matplotlib inline

img\_path\_1 ='/content/gdrive/MyDrive/Kaggle\_Assesment/Skin\_Data/Cancer/Training/1703.JPG'

img\_1 = cv2.imread(img\_path\_1)

img\_path\_2 ='/content/gdrive/My Drive/Kaggle\_Assesment/Skin\_Data/Non\_Cancer/Testing/2566-03.JPG'

img\_2 = cv2.imread(img\_path\_2)

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 Cancer Testing')

plt.subplot(322)

plt.imshow(cv2.cvtColor(flip\_img\_v2, cv2.COLOR\_BGR2RGB)),plt.title('Vertical flipped image\n Non-Cancer Training')

plt.subplot(323)

plt.imshow(cv2.cvtColor(flip\_img\_h1, cv2.COLOR\_BGR2RGB)), plt.title('Horizontal flipped image\n Cancer Testing')

plt.subplot(324)

plt.imshow(cv2.cvtColor(flip\_img\_h2, cv2.COLOR\_BGR2RGB)), plt.title('Horizontal flipped image\n Non-Cancer Training')

plt.subplot(325)

plt.imshow(cv2.cvtColor(transp\_img\_1, cv2.COLOR\_BGR2RGB)),plt.title('Transposed image\n Cancer Testing')

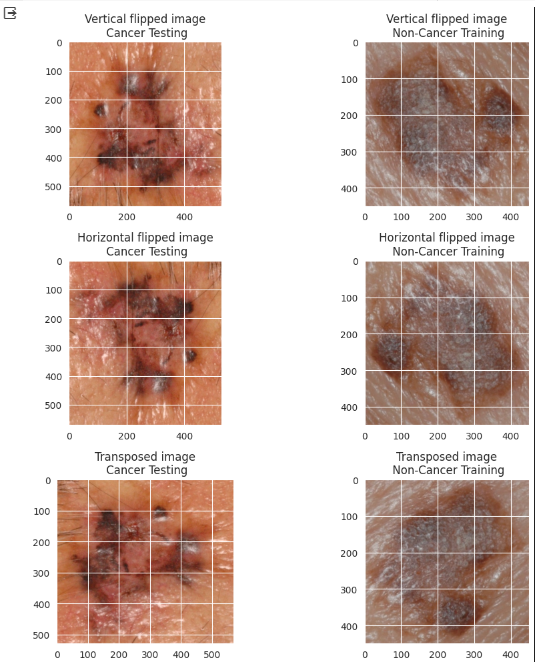
plt.subplot(326)

plt.imshow(cv2.cvtColor(transp\_img\_2, cv2.COLOR\_BGR2RGB)),plt.title('Transposed image\n Non-Cancer Training')

plt.tight\_layout()

plt.show()

# This code inturn outputs the following 6 images:



# And so, the images have been deployed and we can see the different angle and geometric transformations.

#Step 4 EDA: What is impact of noising and denoising operations on image #quality (aka Colour and Texture Analysis)

#Conversion to Gray scale image needed for colour and texture analysis

# To answer this code we have to input the following code:

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/Kaggle\_Assesment/Skin\_Data/Cancer/Training/1703.JPG'

img\_1 = cv2.imread(img\_path\_1)

img\_path\_2 ='/content/gdrive/My Drive/Kaggle\_Assesment/Skin\_Data/Non\_Cancer/Testing/2566-03.JPG'

img\_2 = cv2.imread(img\_path\_2)

#gray scale conversion

img\_1\_gray = skic.rgb2gray(img\_1)

img\_2\_gray = skic.rgb2gray(img\_2)

# We 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.)

#Noise reduction using Gaussian Blur

d=3

img\_1\_blur3 = cv2.GaussianBlur(skic.rgb2gray(img\_1), (2\*d+1, 2\*d+1), -

1)[d:-d,d:-d]

img\_2\_blur3 = cv2.GaussianBlur(skic.rgb2gray(img\_2), (2\*d+1, 2\*d+1), -

1)[d:-d,d:-d]

img\_1\_blur6 = cv2.GaussianBlur(skic.rgb2gray(img\_1), (2\*d+1, 2\*d+1), -

1)[d:-d,d:-d]

img\_2\_blur6 = cv2.GaussianBlur(skic.rgb2gray(img\_2), (2\*d+1, 2\*d+1), -

1)[d:-d,d:-d]

plt.figure(figsize=(10,10))

#VisualisingGray scale images visualisation

plt.subplot(341), plt.imshow(cv2.cvtColor(img\_1, cv2.COLOR\_BGR2RGB)),plt.title('Original image\n Cancer Training')

plt.subplot(342), plt.imshow(img\_1\_gray, cmap = 'gray'),plt.title('GrayScale image\n Cancer Training')

plt.subplot(343), plt.imshow(cv2.cvtColor(img\_2, cv2.COLOR\_BGR2RGB)),plt.title('Original image\n Non-Cancer Testing')

plt.subplot(344), plt.imshow(img\_2\_gray, cmap = 'gray'),plt.title('GrayScale image\n Non-Cancer Testing')

#Visualising Noising-Denoising images

plt.subplot(345), plt.imshow(img\_1\_n,cmap = 'gray'), plt.title('Noise added image\n Cancer Training')

plt.subplot(346), plt.imshow(img\_1\_d,cmap = 'gray'),plt.title('Denoised image\n Cancer Training')

plt.subplot(347), plt.imshow(img\_2\_n,cmap = 'gray'),plt.title('Noise added image\n Non-Cancer Testing')

plt.subplot(348), plt.imshow(img\_2\_d,cmap = 'gray'),plt.title('Denoised image\n Non-Cancer Testing')

#Visualising Noise Reduction with Gaussian Blurring

plt.subplot(349), plt.imshow(img\_1\_blur3,cmap = 'gray'), plt.title('Blurred image(d=3)\n Cancer Training')

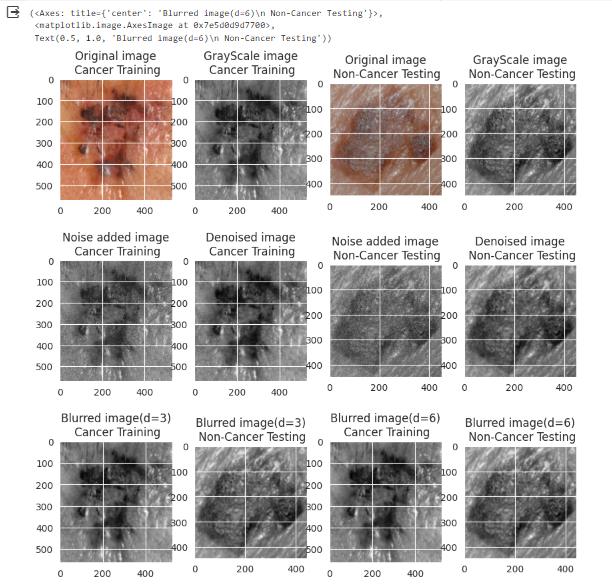
plt.subplot(3,4,10), plt.imshow(img\_2\_blur3,cmap ='gray'),plt.title('Blurred image(d=3)\n Non-Cancer Testing')

plt.subplot(3,4,11), plt.imshow(img\_1\_blur6,cmap ='gray'),plt.title('Blurred image(d=6)\n Cancer Training')

plt.subplot(3,4,12), plt.imshow(img\_2\_blur6,cmap ='gray'),plt.title('Blurred image(d=6)\n Non-Cancer Testing')

# See Page 13 for the image output

# This batch of code will output the following images



#Step 6 EDA- How discriminative are the salient features such as edges and #corners for images corresponding to each class

#Conversion to Gray scale image needed for extracting edges and corners

# To answer this EDA question we have to input the following codes:

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/Kaggle\_Assesment/Skin\_Data/Cancer/Training/1703.JPG'

img\_1 = cv2.imread(img\_path\_1)

img\_path\_2 ='/content/gdrive/My Drive/Kaggle\_Assesment/Skin\_Data/Non\_Cancer/Testing/2566-03.JPG'

img\_2 = cv2.imread(img\_path\_2)

#Sobel edge detector

#edge detector works on gray scale images

sobel\_img\_1=cv2.cvtColor(img\_1,cv2.COLOR\_BGR2GRAY)

sobel\_img\_2=cv2.cvtColor(img\_2,cv2.COLOR\_BGR2GRAY)

sobelx\_img\_1 = cv2.Sobel(sobel\_img\_1,cv2.CV\_64F,1,0,ksize=9)

sobely\_img\_1 = cv2.Sobel(sobel\_img\_1,cv2.CV\_64F,0,1,ksize=9)

sobelx\_img\_2 = cv2.Sobel(sobel\_img\_2,cv2.CV\_64F,1,0,ksize=9)

sobely\_img\_2 = cv2.Sobel(sobel\_img\_2,cv2.CV\_64F,0,1,ksize=9)

#Canny edge detector

#threshold selection

th1=30

th2=60

# Canny recommends threshold 2 is 3 times threshold 1

# you could try experimenting with this...

d=3

# gaussian blur

# this takes pixels in edgeresult where edge non-zero and colours them bright green

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)

#Corner detector

#detecting corners for image\_1

harris\_1=img\_1.copy()

#greyscale it

gray = cv2.cvtColor(harris\_1,cv2.COLOR\_BGR2GRAY)

gray = np.float32(gray)

blocksize=4 #

kernel\_size=3 # sobel kernel: must be odd and fairly small

# run the harris corner detector

dst = cv2.cornerHarris(gray,blocksize,kernel\_size,0.05)

# parameters are blocksize, Sobel parameter and Harris threshold

#result is dilated for marking the corners, this is visualisation related and just makes them bigger

dst = cv2.dilate(dst,None)

#we then plot these on the input image for visualisation purposes, using bright red

harris\_1[dst>0.01\*dst.max()]=[0,0,255]

#detecting corners for image\_2

harris\_2=img\_2.copy()

#greyscale it

gray = cv2.cvtColor(harris\_2,cv2.COLOR\_BGR2GRAY)

gray = np.float32(gray)

blocksize=4 #

kernel\_size=3

# sobel kernel: must be odd and fairly small

# run the harris corner detector

dst = cv2.cornerHarris(gray,blocksize,kernel\_size,0.05)

# parameters are blocksize, Sobel parameter and Harris threshold

#result is dilated for marking the corners, this is visualisation related and just makes them bigger

dst = cv2.dilate(dst,None)

#we then plot these on the input image for visualisation purposes, using bright red

harris\_2[dst>0.01\*dst.max()]=[0,0,255]

#Visualisng Edges and Corners

plt.figure(figsize=(10,10))

#Visualising Sobel Edges

plt.subplot(341), plt.imshow(sobelx\_img\_1, cmap =

'gray'),plt.title('Horizontal edges\n Cancer Training')

plt.subplot(342), plt.imshow(sobely\_img\_1, cmap =

'gray'),plt.title('Horizontal edges\n Cancer Training')

plt.subplot(343), plt.imshow(sobelx\_img\_2, cmap =

'gray'),plt.title('Vertical edges\n Non-Cancer Testing')

plt.subplot(344), plt.imshow(sobely\_img\_2, cmap =

'gray'),plt.title('Vertical edges\n Non-Cancer Testing')

#Visualising Canny Edges

plt.subplot(345), plt.imshow(cv2.cvtColor(img\_1, cv2.COLOR\_BGR2RGB)),plt.title('Original image\n Cancer Training')

plt.subplot(346), plt.imshow(edgeresult\_1, cmap = 'gray'),plt.title('Canny edges\n Cancer Training')

plt.subplot(347), plt.imshow(cv2.cvtColor(img\_2,  cv2.COLOR\_BGR2RGB)),plt.title('Original image\n Non-Cancer Testing')

plt.subplot(348), plt.imshow(edgeresult\_2, cmap = 'gray'),plt.title('Vertical edges\n Non-Cancer Testing')

#Visualising Corners

plt.subplot(349), plt.imshow(cv2.cvtColor(img\_1,

cv2.COLOR\_BGR2RGB)),plt.title('Original image\n Cancer Training')

plt.subplot(3,4,10), plt.imshow(cv2.cvtColor(harris\_1,

cv2.COLOR\_BGR2RGB)),plt.title('Image with Corners\n Cancer Training')

plt.subplot(3,4,11), plt.imshow(cv2.cvtColor(img\_2,

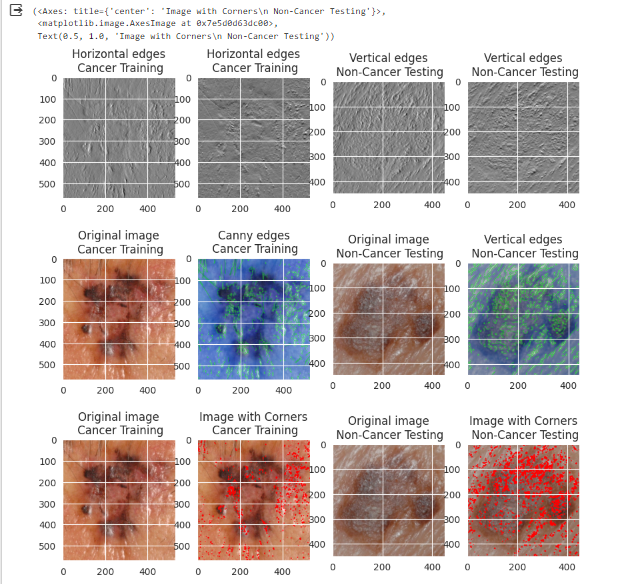
cv2.COLOR\_BGR2RGB)),plt.title('Original image\n Non-Cancer Testing')

plt.subplot(3,4,12), plt.imshow(cv2.cvtColor(harris\_2,

cv2.COLOR\_BGR2RGB)),plt.title('Image with Corners\n Non-Cancer Testing')

# Proceed to Page 17 for the Image output

# This batch of codes outputs the following images to answer the EDA Question



The 5 EDA questions that have been written and answered contributes to understanding how the data is being managed and how change in angles contribute to our understanding of the disease. Looking at data at point face sometimes is not enough and sometimes it requires a deeper internal understanding to achieve a desirable outcome.

## Stage 2: PDA (Predictive Data Analytics)

For the PDA we are going to use a Teachable machine to give us the necessary graph, accuracy per class, and the confusion matrix. To use the Teachable machine with the dataset the following steps need to be followed:

1. Open the Teachable machine and make an account (Same account as the Google Collab account)
2. Upload the image folders (cancer and non-cancer folders into the teachable machine
3. Once the images are put into classes 1 and 2 press the train model button
4. Wait for the train model to finish training the models.
5. Once the model is trained press on advanced and press the Accuracy per matrix, confusion matrix accuracy, and losses per epoch and screenshot them.
6. If the Teachable model (any image) Is saved to the drive then paste the URL link

<https://drive.google.com/file/d/1S_idsazSbHTIiQc5uAfj307UN0xvPHos/view?usp=share_link>

If the steps above have been followed, then this is the result of the output of under-the-hood calculation.

A screenshot of a cell phone

Description automatically generated

A diagram of a class

Description automatically generated with medium confidence

A graph with a line graph

Description automatically generated

A graph with a line

Description automatically generated

The resulting images above shows the accuracy of the images in determining whether it is cancerous or non-cancerous. For example, if I was to calculate the accuracy of one of the images the tm machine would identify whether cancer or not (using percentage). This is especially useful in identifying and accurately pin pointing which images are cancerous or not.

## Stage 3A Deployment/Implementation with teachable machine

The deployment of the capstone can be done with a teachable machine using the Keras model to classify the image. This can be done by using the following code:

1. Download the Keras model and unzip it
2. On the teachable machine press the import model and copy the Python code in Google Collab.
3. Make sure the unzipped keras and text label file are correctly placed in the right folder.
4. Then paste the code that is found in the TM import model.
5. from keras.models import load\_model  # TensorFlow is required for Keras to work
6. from PIL import Image, ImageOps  # Install pillow instead of PIL
7. import numpy as np
8. # Disable scientific notation for clarity
9. np.set\_printoptions(suppress=True)
10. # Load the model
11. model = load\_model("/content/gdrive/MyDrive/Kaggle\_Assesment/content/gdrive/MyDrive/Kaggle\_Assesment/keras\_model.h5", compile=False)
12. # Load the labels
13. class\_names = open("/content/gdrive/MyDrive/Kaggle\_Assesment/content/gdrive/MyDrive/Kaggle\_Assesment/labels.txt", "r").readlines()
14. # Create the array of the right shape to feed into the keras model
15. # The 'length' or number of images you can put into the array is
16. # determined by the first position in the shape tuple, in this case 1
17. data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
18. # Replace this with the path to your image
19. image = Image.open("/content/gdrive/MyDrive/Kaggle\_Assesment/Skin\_Data/Cancer/Training/1157-01.JPG").convert("RGB")
20. # resizing the image to be at least 224x224 and then cropping from the center
21. size = (224, 224)
22. image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)
23. # turn the image into a numpy array
24. image\_array = np.asarray(image)
25. # Normalize the image
26. normalized\_image\_array = (image\_array.astype(np.float32) / 127.5) - 1
27. # Load the image into the array
28. data[0] = normalized\_image\_array
29. # Predicts the model
30. prediction = model.predict(data)
31. index = np.argmax(prediction)
32. class\_name = class\_names[index]
33. confidence\_score = prediction[0][index]
34. # Print prediction and confidence score
35. print("Cancer:", class\_name[2:], end="")
36. print("Confidence Score:", confidence\_score)

# This will output the following:

output

1/1 [==============================] - 1s 1s/step

Cancer: Class 1

Confidence Score: 0.99980074

# The confidence class and the cancer type is shown, and we can also show the image too using the following code

from warnings import filterwarnings

import tensorflow as tf

from tensorflow import io

from tensorflow import image

from matplotlib import pyplot as plt

filterwarnings("ignore")

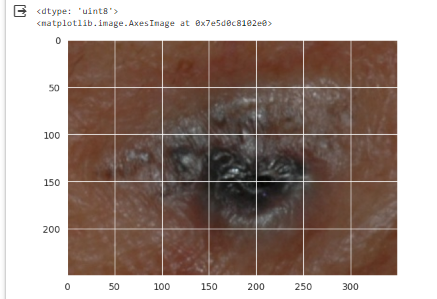
tf\_img = io.read\_file("/content/gdrive/MyDrive/Kaggle\_Assesment/Skin\_Data/Cancer/Training/1157-01.JPG")

tf\_img = image.decode\_png(tf\_img, channels=3)

print(tf\_img.dtype)

plt.imshow(tf\_img)

# plt.show()



This is the image outputted, and it coincides with the confidence score.

## Stage 3B Deployment/Implementation with TkInter

The next stage is implementing the capstone using Tkinter (vs code). The following code will be the implementing code. One thing to remember is to have the code align with where the images are stored. This means if u made a python file in vs code, then have the .py in the same folder as the skin images. The following code will do the implementation.

from tkinter import filedialog

from tkinter import \*

import tkinter as tk

from PIL import ImageTk, Image, ImageOps  # Install pillow instead of PIL

from keras.models import load\_model  # TensorFlow is required for Keras to work

import numpy as np

# load the trained model

model = load\_model('keras\_model.h5')

# Load the labels

label\_path = 'labels.txt'

class\_names = open(label\_path, "r").readlines()

test\_image\_path = 'test-image.jpg'

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

# initialize GUI

top = tk.Tk()

top.geometry('800x600')

top.title('Skin Cancer Classification')

top.configure(*background*='#CDCDCD')

label = Label(top, *background*='#CDCDCD', *font*=('arial', 15, 'bold'))

output\_image = Label(top)

*def* classify(*test\_image\_path*):

    global label\_packed

    disp\_string = ''

    image = Image.open(*test\_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]

    confidence\_score = prediction[0][index]

    # Display prediction and confidence score

    disp\_string += "\nClass:" + str(class\_name[2:])

    disp\_string += "\nConfidence Score:" + str(confidence\_score)

    # label.configure(foreground='#011638', text=class\_name)

    label.configure(*foreground*='#011638', *text*=disp\_string)

*def* show\_classify\_button(*file\_path*):

    classify\_b = Button(top, *text*="Classify Image", *command*=*lambda*: classify(*file\_path*), *padx*=10, *pady*=5)

    classify\_b.configure(*background*='#364156', *foreground*='white', *font*=('arial', 10, 'bold'))

    classify\_b.place(*relx*=0.79, *rely*=0.46)

*def* upload\_image():

    try:

        file\_path = filedialog.askopenfilename()

        uploaded = Image.open(file\_path)

        uploaded.thumbnail(((top.winfo\_width() / 2.25), (top.winfo\_height() / 2.25)))

        im = ImageTk.PhotoImage(uploaded)

        output\_image.configure(*image*=im)

        output\_image.image = im

        label.configure(*text*='')

        show\_classify\_button(file\_path)

    except:

        pass

upload = Button(top, *text*="Upload an image", *command*=upload\_image, *padx*=10, *pady*=5)

upload.configure(*background*='#364156', *foreground*='white', *font*=('arial', 10, 'bold'))

upload.pack(*side*=BOTTOM, *pady*=50)

output\_image.pack(*side*=BOTTOM, *expand*=True)

label.pack(*side*=BOTTOM, *expand*=True)

heading = Label(top, *text*="Skin Cancer Classification", *pady*=20, *font*=('arial', 20, 'bold'))

heading.configure(*background*='#CDCDCD', *foreground*='#364156')

heading.pack()

top.mainloop()

The code above gives the following output. The images below shows that using the implementation tkinter, the system is able to identify the cancer class and the accuracy of each individual cancer images.

A close-up of a skin cancer

Description automatically generated

A close-up of a skin cancer

Description automatically generated

## Conclusions

This capstone report was done using Google Collab, visual studio code for the implementation, teachable machine for the PDA. The dataset used for the project was Binary Skin Classification and throughout the project utilizing the EDA in Stage 1, PDA in stage 2, and Implementation in Stage 3, we have garnered valuable insight into understanding the binary classification of skin cancers. As we can deduct from the confidence score of 99% of the images that have been classified, it is safe to confirm that this would be an invaluable tool for medical practitioners to identify skin cancer and prevent further damage to patients in the future.

## References

K. Graupe, “Skin Cancer Binary Classification,” Kaggle , Feb. 2023. https://www.kaggle.com/datasets/kylegraupe/skin-cancer-binary-classification-dataset (accessed Oct. 18, 2023).

## Appendix 1: Logbook

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Week | Planned  Activities | Tasks  Completed | Problems  Faced | Further Comments |
| Week 10 | Develop the EDA Questions and determine what to do for the project | Developed 3 EDA questions and wanted to use Google Collab | None | None |
| Week 11 | Implement the Google Collab with Google Drive and start importing | Started the Google Collab linking with the g drive and got the 5 EDA questions | It took a while for me to link the Google Collab and understand how the linking worked, but it worked | None |
| Week 12 | I finished the EDA and the PDA using the tm and I plan to do my ptinker implementation on the first day of week 13 | Finished the EDA Analysis and the PDA analysis using teachable machine | The images were colored blue and did not have it in the original color when I imported the images, but I managed to fix and have the color as the image intended. | None |
| Week 13 | I finished my ptinker implementation and managed to get the classification working | Finished the ptinker implementation and the report of the capstone | The image was not loading however after 3 hours I got the classification working and finished the project on time | None |