
Group Members:

Name: Spencer Lobo **UCID:** 30228989

Name: Anan Ghosh **UCID:** 30220833

Name: Debistuti Kundu **UCID:** 30227696

Name: Himadri Das **UCID:** 30230898

Name: Ariful Islam Anik **UCID:** 30214525

Prerequisite

Running the below code will mount the google drive to the collab notebook. This will ensure the that the Images can be selected and saved to the desired Google Drive Location.

```
from google.colab import drive      # Mount Google Drive to Colab
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
!pip install Pillow                 # Install Pillow library
```

```
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (9.4.0)
```

```
!sudo apt install tesseract-ocr     # Install tesseract-ocr library
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  tesseract-ocr-eng tesseract-ocr-osd
The following NEW packages will be installed:
  tesseract-ocr tesseract-ocr-eng tesseract-ocr-osd
0 upgraded, 3 newly installed, 0 to remove and 15 not upgraded.
Need to get 4,816 kB of archives.
After this operation, 15.6 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tesseract-ocr-eng all 1:4.00~git30-7274cfa-1.1 [1,591 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tesseract-ocr-osd all 1:4.00~git30-7274cfa-1.1 [2,990 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tesseract-ocr amd64 4.1.1-2.1build1 [236 kB]
Fetched 4,816 kB in 0s (11.0 MB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package tesseract-ocr-eng.
(Reading database ... 120899 files and directories currently installed.)
Preparing to unpack .../tesseract-ocr-eng_1%3a4.00~git30-7274cfa-1.1_all.deb ...
Unpacking tesseract-ocr-eng (1:4.00~git30-7274cfa-1.1) ...
Selecting previously unselected package tesseract-ocr-osd.
Preparing to unpack .../tesseract-ocr-osd_1%3a4.00~git30-7274cfa-1.1_all.deb ...
Unpacking tesseract-ocr-osd (1:4.00~git30-7274cfa-1.1) ...
Selecting previously unselected package tesseract-ocr.
Preparing to unpack .../tesseract-ocr_4.1.1-2.1build1_amd64.deb ...
Unpacking tesseract-ocr (4.1.1-2.1build1) ...
Setting up tesseract-ocr-eng (1:4.00~git30-7274cfa-1.1) ...
Setting up tesseract-ocr-osd (1:4.00~git30-7274cfa-1.1) ...
Setting up tesseract-ocr (4.1.1-2.1build1) ...
Processing triggers for man-db (2.10.2-1) ...
```

```
!pip install pytesseract            # Install pytesseract library
```

```
Collecting pytesseract
  Downloading pytesseract-0.3.10-py3-none-any.whl (14 kB)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from pytesseract) (23.2)
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from pytesseract) (9.4.0)
Installing collected packages: pytesseract
Successfully installed pytesseract-0.3.10
```

Importing Library

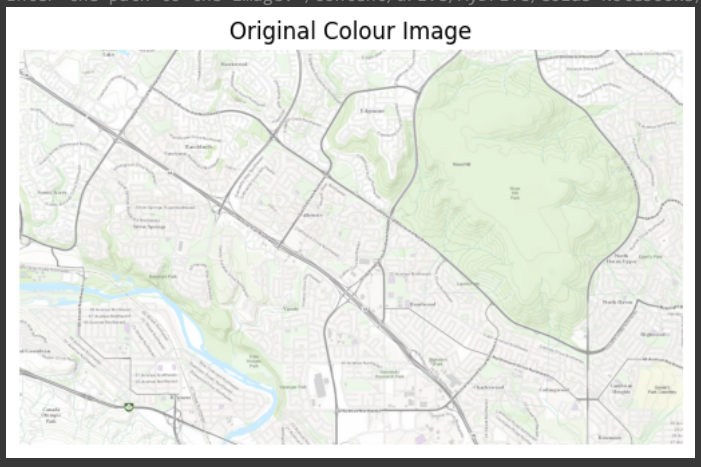
```
from google.colab import drive                  # Connect Google Drive
from matplotlib import pyplot as plt            # Used to Plot Graphs
import numpy as np                              # While working on image array
from PIL import Image                           # Used to read as well as work on an image
from google.colab.patches import cv2_imshow     # Importing CV2 library in google collab it also patches any errors caused by cv2 while ope
import cv2                                      # Importing CV2 library in google collab
import time                                     # Set delays (counter)
import pytesseract                              # Used for text extraction
from scipy import misc,ndimage                  # Importing Scipy library also while applying sharpening of the image
```

Main Code Block

Importing Image from Google Drive

```
drive.mount('/content/drive')                           # Mounting Google Drive
image_path = input(str("Enter the path to the image: "))  # Input the image path of the Google Drive
image = Image.open(image_path)                          # Read the image using PIL
image = np.array(image)                                 # Convert the PIL image to a numpy array
img = Image.open(image_path)                            # Create a copy of the image with img
image = img
org_image = image
image = np.array(img)                                   # Create an array of the image
plt.imshow(img)                                         # Display the Image
plt.title("Original Colour Image")                      # Add Title
plt.axis('off')                                         # Remove the axis
plt.show()                                              # Display
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Enter the path to the image: /content/drive/MyDrive/Colab Notebooks/Nose Hill.png
```



Basic Image Processing

GrayScale of Image

```
def GrayScale(image):                          # Function to Convert image to grayscale
  image = img.convert('L')                     # Convert code to 'L' grayscale
  global ar                                    # Make variable ar global
  ar = np.array(image)                         # Assign the grayscale image array to variable ar
  print("Grayscale Image array")               # Displaying text
  print(ar)                                    # Display the array
  plt.title("Grayscale of Original Image")     # Display the title
  plt.imshow(image)                            # Display the image
  plt.gray()                                   # Display only the grayscale
  plt.axis('off')                              # Removing the axis
  plt.show()                                   # Display the processed image

  return image, ar                             # Return Image and Array
```

Gaussian Blur

```
def GaussianBlur(image, ar):                 # Function to Apply Gaussian Blur on the image
  image = cv2.GaussianBlur(ar, (5, 5), 0)    # Apply Gaussian Blur with a 5x5 kernel
  plt.imshow(image, cmap='gray')             # Display Image with cmap='gray'
  plt.title('Gaussian Blurred Image')        # Add Title to Image
  plt.axis('off')                            # Turn Off the axis
  plt.show()                                 # Display Image

  return image                               # Return Image
```

Image Sharpening

```
def Sharpen_Image(image):                        # Function to Sharpen Image
  image = misc.face(gray=True).astype(float)
  blur = ndimage.gaussian_filter(img, 5)         # Applying Gaussian filter
  plt.imshow(blur)                               # Plot Image
  plt.axis('off')                                # Turn Off the axis
  plt.show()                                     # Display the Image
  blur_G = ndimage.gaussian_filter(blur, 1)
  alpha = int(input('Enter value of alpha (sharpening value): ')) # User Input for the alpha (sharpening value)
  sharp = blur+alpha*(blur-blur_G)               # Blur + the Alpha multiplied with the blur_G
  plt.imshow(sharp)                              # Plot Image
  plt.axis('off')                                # Turn Off the axis
  plt.show()                                     # Display Image
```

Image Rotation

```
def RotateImage(image):                        # Function to Rotate Image
  r = int(input('Enter the rotation degree: ')) # User Input the rotation degree
  rows, cols, _ = image.shape                                  # Extracts the height and width of original image
  rotation_matrix = cv2.getRotationMatrix2D((cols/2, rows/2), r, 1)  # 2D rotation of 45 degrees
  image = cv2.warpAffine(image, rotation_matrix, (cols, rows))       # Apply the 45 degree rotation
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)# Convert BGR format to RGB
  plt.imshow(image)                            # Plot Image
  plt.title('(Rotated Image')                  # Add Title
  plt.axis('off')                              # Removing the axis
  plt.gray()                                   # Display only the grayscale
  plt.show()                                   # Display Image
  return image                                 # Return Image
```

Image Resizing

```
def ResizedImage(image):                          # Function to Resize the image
  x = int(input('Enter Scaling for x-axis: '))   # User Input the X - axis scaling factor
  y = int(input('Enter Scaling for y-axis: '))   # User Input the Y - axis scaling factor
  image = cv2.resize(image, (0, 0), fx=x, fy=y)  # Resize using cv2.resize() the image to half (0.5) both height and width
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convert BGR format to RGB
  plt.imshow(image)                               # Plot Image
  plt.title('Resized Image')                      # Add Title
  plt.axis('off')                                 # Turn off the axis label and title
  plt.gray()                                      # Display only the grayscale
  plt.show()                                      # Display Image
  return image                                    # Return Image
```

Canny Edge Detection

```
def CannyEdge(image):                         # Function to detect edges using canny edge detection
  GrayScale(image)                            # Run the GrayScale(image) function block
  x = int(input('Input lower threshold For Canny Edge Detection: '))   # User Input lower threshold
  y = int(input('Input Upper threshold For Canny Edge Detection: '))   # User Input Upper threshold
  images = cv2.Canny(image, x, y)             # Appling Canny Edge detection with lower threshold and upper threshold
  plt.imshow(images, cmap='gray')             # Plot Image
  plt.title('Edges Image')                    # Add Title
  plt.axis('off')                             # Removing the axis
  plt.show()                                  # Display Image
  return image                                # Return Image
```

Binary Thresholding

```
def BinaryThreshold(image):                   # Function to apply Binary Thresholding on an image
  CannyEdge(image)                            # Run the CannyEdge(image) function block
  x = int(input('Input threshold For Binary Thresholding: '))  # User Input threshold value For Binary Thresholding
  _, image = cv2.threshold(image, x, 255, cv2.THRESH_BINARY)   # Applying binary thresholding on the grayscale image with threshold values
  plt.imshow(image, cmap='gray')              # Plot Image
  plt.title('Binary Thresholding')            # Add Title
  plt.axis('off')                             # Removing the axis
  plt.gray()                                  # Display only the grayscale
  plt.show()                                  # Display Image
  return image                                # Return Image
```

Morphological Operations

Erosion

```
def Erosion(image,ar):                        # Function to apply Ersion on image
  kernel = np.ones((5, 5), np.uint8)
  image = cv2.erode(ar, kernel, iterations=2) # Apply Erosion on the image using cv2
  plt.imshow(image, cmap='gray')              # Plot image as grayscale
  plt.title('Erosion Image')                  # Add Title
  plt.axis('off')                             # Removing the axis
  plt.gray()                                  # Display only the grayscale
  plt.show()                                  # Display Image
  return image                                # Return Image
```

Dilation

```
def Dilation(image, ar):                      # Function to apply Dilation on image
  kernel = np.ones((5, 5), np.uint8)
  image = cv2.dilate(ar, kernel, iterations=2)# Apply Dilation on the image using cv2
  plt.imshow(image, cmap='gray')              # Display image as gray
  plt.title('Dilation Image')                 # Add Title
  plt.axis('off')                             # Removing the axis
  plt.gray()                                  # Display only the grayscale
  plt.show()                                  # Display Image
  return image                                # Return Image
```

Image Restoration

Histogram Analysis

```
def Hist(image):                              # Function to get Histogram Analysis
  GrayScale(image)                            # Run the GrayScale(image) function block
  original_hist = cv2.calcHist([image], [0], None, [256], [0, 256])
  plt.plot(original_hist, color='black')      # plotting an histogram
  plt.title('Image Histogram')                # Adds Title
  plt.xlim([0, 256])                          # x-axis limit
  plt.show()                                  # Display Image
```

Histogram Equalization

```
def Image_restoration(image):                 # Function to apply Image Restoration using histogram equalization
  image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Read the image as grayscale using cv2
  original_hist = cv2.calcHist([image], [0], None, [256], [0, 256])
  plt.plot(original_hist, color='black')      # Plot the original histogram image
  plt.title('Image Histogram')                # Adds Title
  plt.xlim([0, 256])                          # x-axis limit
  plt.show()                                  # Display Image

  equalized_image = cv2.equalizeHist(image)   # Equalize the histogram

  plt.hist(equalized_image.ravel(), 256, [0, 256], color='black')
  plt.title('Equalized Image Histogram')      # Adds Title
  plt.show()                                  # Plot the image

  plt.imshow(equalized_image, cmap='gray')    # Display the equalized image as grayscale
  plt.title('Equalized Image')                # Adds Title
  plt.axis('off')                             # Removing the axis
  plt.show()                                  # Return Image
```

Advanceed Image Processing

Double Threshold

```
def double_threshold(image): # Function for applying Double Threshold
  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)      # Convert the image to grayscale
  blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)   # Apply Gaussian Blur to reduce noise
  low_threshold = int(input("Enter Lower Threshold: "))     # User Input lower threshold value
  high_threshold = int(input("Enter Higher Threshold: "))   # User Input upper threshold value
  _, low_threshold_mask = cv2.threshold(blurred_image, low_threshold, 255, cv2.THRESH_BINARY)         # low Treshold mask
  _, high_threshold_mask = cv2.threshold(blurred_image, high_threshold, 255, cv2.THRESH_BINARY_INV)   # high Treshold mask

  double_thresholded_image = cv2.bitwise_and(low_threshold_mask, high_threshold_mask)   # Combine the two threshold masks
  plt.imshow(image, cmap='gray')              # Display image as grayscale
  plt.title('Double Threshold Result')        # Adds Title
  plt.gray()                                  # Display only the grayscale
  plt.axis('off')                             # Removing the axis
  plt.show()                                  # Display Image

  return image                                # Return Image
```

Prewitt Filter

```
def Prewitt(image, ar):                          # Function for applying Prewitt filters
  ask = int(input("Enter the Type of Prewitt Filter: \n 1: Prewitt Horizontal, 2: Prewitt Vertical, 3: Prewitt +45, 4: Prewitt -45.: ")) # U
  if ask == 1:
      pv = [[-1, -1, -1], [0, 0, 0], [1, 1, 1]]  # Prewitt Horizontal filter
  elif ask == 2:
      pv = [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]  # Prewitt Vertical filter
  elif ask == 3:
      pv = [[-1, -1, 0], [-1, 0, 1], [0, 1, 1]]  # Prewitt +45 filter
  elif ask == 4:
      pv = [[0, 1, 1], [-1, 0, 1], [-1, -1, 0]]  # Prewitt -45 filter
  else:
      print("Invalid operation")                 # Print Invalid Statement
      return None                                # Return none
  pv = np.array(pv)                              # Convert the prewitt filter to array
  print("Prewitt Mask")                          # Print statement
  print(pv)                                      # Print the Prewitt filter
  ar_pv = ndimage.convolve(ar, pv)               # Convolve array with Prewitt filter
  print("After applying Prewitt Mask")           # Print statement
  print(ar_pv)                                   # Print array
  image = Image.fromarray(ar_pv)                 # Image from array
  plt.imshow(image)                              # Displays the image
  plt.title("Image after applying Prewitt Mask") # Adds Title
  plt.axis('off')                                # Removing the axis
  plt.gray()                                     # Display only the grayscale
  plt.show()                                     # Display Image

  return image                                   # Return Image
```

Features and Text Extraction

Feature Extraction

```
def feature_ext(image):                                                    # Function for feature extraction
  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)                     # Convert the image to grayscale
  blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)                  # Apply Gaussian Blur to reduce noise and improve feature ext
  edges = cv2.Canny(blurred_image, 50, 150)                                # Use Canny edge detection to find edges in the image
  contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)   # Find contours in the image
  gis_map_with_contours = image.copy()                                     # Draw the contours on the original image
  cv2.drawContours(gis_map_with_contours, contours, -1, (0, 255, 0), 2)
  plt.imshow(gis_map_with_contours)                                        # Plot Image
  plt.title('Feature extraction')                                          # Add Title
  plt.axis('off')                                                          # Removing the axis
  plt.gray()                                                               # Display only the grayscale
  plt.show()                                                               # Display the Image
```

Text Extraction

```
def text_ext(image):                                                       # Function for text extraction
  pytesseract.pytesseract.tesseract_cmd = r'/usr/bin/tesseract'            # Configure pytesseract to use the tesseract-ocr executable
  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)                     # Convert the image to grayscale
  extracted_text = pytesseract.image_to_string(Image.fromarray(gray_image))     # Perform OCR using pytesseract
  print("Extracted Text:\n", extracted_text)                               # Print the extracted text
```

Executing Image Processing Tasks:

```
import time
while True:
  time.sleep(1)       # Set a deplay for 1 sec
  operation = input( 'Enter Image Processing Option (1-15) \n     "1 : Grayscale Image", "2 : Gaussian Blur on Image", "3 : Rotate Image", "4

  if operation == "1":
    GrayScale(image)
  elif operation == "2":
    GaussianBlur(image, ar)
  elif operation == "3":
    RotateImage(image)
  elif operation == "4":
    Sharpen_Image(image)
```

```
  elif operation == "5":
    ResizedImage(image)
  elif operation == "6":
    CannyEdge(image)
  elif operation == "7":
    BinaryThreshold(image)
  elif operation == "8":
    Erosion(image, ar)
  elif operation == "9":
    Dilation(image, ar)
  elif operation == "10":
    Hist(image)
  elif operation == "11":
    double_threshold(image)
  elif operation == "12":
    Prewitt(image, ar)
  elif operation == "13":
    text_ext(image)
  elif operation == "14":
    feature_ext(image)
  elif operation == "15":
    Image_restoration(image)
  elif operation == "quit":
    break
  else:
    print("Invalid operation")
    continue
# Add a delay of 10 second before the next iteration
time.sleep(1)
```