CSCE460301 - Fundamental of Computer Vision (Spring 2025) Instructions

Short programming example (130 points)

Each of the following short programming worth 10 points.

Please use the provided “hokiebird.jpg” as your input.

1) Plot the R, G, B values along the scanline on the 250th row of the image.
   - Save your plot as “01_scanline.png”.

```
#first assigment computer vistion
#question 1: Plot the R, G, B values along the scanline
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

# Load the image
#The image is loaded as a PIL Image object.
#The image is in RGB format (Red, Green, Blue) by default.
#the main purpose is to open the file to read the image
Image=Image.open("/content/hokiebird.jpg")

#conver the image to RGB (if it's not already in RGB format)
Image=Image.convert('RGB')

#Get the dimantion of the image
width, height = Image.size

#dealing with the second which is scanline
scanline=[Image.getpixel((x,249)) for x in range(width)]

#seperate the R, G, B arrays
#this apply for the library PIL
R=[scanline[i][0] for i in range(len(scanline))]
G=[scanline[i][1] for i in range(len(scanline))]
B=[scanline[i][2] for i in range(len(scanline))]
```

```
#plot the R, G, B values
plt.figure(figsize=(10,5))
plt.plot(R, label='R')
plt.plot(G, label='G')
plt.plot(B, label='B')
plt.title('R, G, B Values along the 250th Scanline')
plt.xlabel('Pixel Position')
plt.ylabel('Pixel Intensity')
plt.legend()
plt.grid(True)


#save the plot
plt.savefig("01_scanline.png")
```

2) Stack the R, G, B channels of the *hokiebird* image vertically. This will be an image with width of 600 pixels and height of 1116 pixels.

- Save the composite image as “02_concat_rgb.png”.

```
#first assigment computer vistion
#question 2 R, G, B channels of the hokiebird image vertically
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

# Load the image
image1 = Image.open("/content/hokiebird.jpg")  # Use a different
variable name (e.g., 'image')

# Convert the image to RGB (if it's not already in RGB format)
#this will applay only with PIL librarry not cv2
image1 = image1.convert('RGB')

# Get the dimensions of the image
width, height = image1.size

# Split the image into R, G, B
# Convert PIL image to NumPy array for OpenCV
R, G, B = cv.split(np.array(image1))
```

```
# Create a new blank image to hold the stacked channels
blank_image = Image.new('RGB', (600, 1116))  # Use the Image class to
create a new blank image

# Paste each channel into the new image
blank_image.paste(Image.fromarray(R), (0, 0))        # Paste Red
channel at the top
blank_image.paste(Image.fromarray(G), (0, 372))     # Paste Green
channel in the middle
blank_image.paste(Image.fromarray(B), (0, 744))     # Paste Blue
channel at the bottom

# Save the composite image
blank_image.save("02_concat_rgb.png")
```

3) Load the input color image and swap its red and green color channels.

- Save the image as “03_swapchannel.png”.

```
#first assigment computer vistion
#question 3 :swaping the red and green color channels
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt


#The image is loaded as a NumPy array in BGR format (Blue, Green,
Red).
#The shape of the array is (height, width, channels), where channels
is typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range
0-255).
image=cv.imread("/content/hokiebird.jpg")


#Make a copy to from the orginal image
New_image=image.copy()

#This line swaps the Green and Red channels of the image.
#the , is conncatination
#New_image geern chanel will be replaced by the red chanel of the
orginal image
```

```
#New_image red chanel will be replaced by the green chanel of the
orginal image
New_image[:, :, 1], New_image[:, :, 2] = image[:, :, 2], image[:, :,
1]


#This line saves the modified image (New_image) to a file named
03_swapchannel.png.
cv.imwrite('03_swapchannel.png', New_image)
```

4) Convert the input color image to a grayscale image.

- Save the grayscale image as “04_grayscale.png”.

```
#first assigment computer vistion
#question 4 :swaping the red and green color channels
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt


#The image is loaded as a NumPy array in BGR format (Blue, Green,
Red).
#The shape of the array is (height, width, channels), where channels
is typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range
0-255).
image=cv.imread("/content/hokiebird.jpg")


#Make a copy to from the orginal image
New_image=image.copy()


# Convert the color image to grayscale using OpenCV
New_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

# Save the grayscale image as "04_grayscale.png"
cv.imwrite('04_grayscale.png', New_image)
```

5) Take the R, G, B channels of the image. Compute an average over the three channels. Note that you may need to do the necessary typecasting (uint8 and double) to avoid overflow.

- Save the averaged image as “05_average.png”.

```
#first assigment computer vistion
#question 5 : 1- Compute an average over the three channels , 2 - save
the result as an (uint8) grayscale image
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

#The image is loaded as a NumPy array in BGR format (Blue, Green,
Red).
#The shape of the array is (height, width, channels), where channels
is typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range
0-255).
image=cv.imread("/content/hokiebird.jpg")

# Split the image into R, G, B channels
R, G, B = cv.split(image)


# Convert the channels to double to avoid overflow
R1 = R.astype(np.double)
G1 = G.astype(np.double)
B1 = B.astype(np.double)

# Compute the average of the three channels
average_channel = (R1 + G1 + B1) / 3

# Convert back to uint8 (grayscale image)
average_channel = average_channel.astype(np.uint8)

# Save the averaged image as "05_average.png"
cv.imwrite('05_average.png', average_channel)
```

6) Convert the input color image to ycbcr color space.

- Save the *y* component as “06_y_ycbcr.png”.

```
#first assigment computer vistion
#question 6 : Convert the input color image to ycbcr color space.
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

#The image is loaded as a NumPy array in BGR format (Blue, Green,
Red).
#The shape of the array is (height, width, channels), where channels
is typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range
0-255).
image=cv.imread("/content/hokiebird.jpg")

# Convert the image from BGR to YCbCr color space
ycbcr_image = cv.cvtColor(image, cv.COLOR_BGR2YCrCb)

# Split the YCbCr image into its components (Y, Cb, Cr)
Y, Cb, Cr = cv.split(ycbcr_image)

# Save the Y component as "06_y_ycbcr.png"
cv.imwrite('06_y_ycbcr.png', Y)
```

7) Convert the input color image to a cie_xyz color space.

- Save the *y* component as “07_y_xyz.png”.

```
#first assigment computer vistion
#question 7 :  Stack the grayscale, y_ycbcr, y_xyz components of the
hokiebird image horizontally.
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt
```

```
#The image is loaded as a NumPy array in BGR format (Blue, Green,
Red).
#The shape of the array is (height, width, channels), where channels
is typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range
0-255).
image=cv.imread("/content/hokiebird.jpg")

# Convert the image from BGR to CIE XYZ color space
xyz_image = cv.cvtColor(image, cv.COLOR_BGR2XYZ)

# Split the XYZ image into its components (X, Y, Z)
X, Y, Z = cv.split(xyz_image)

# Save the Y component as "07_y_xyz.png"
cv.imwrite('07_y_xyz.png', Y)
```

8) Stack the *grayscale*, *y_ycbcr*, *y_xyz* components of the *hokiebird* image horizontally. This will be an image with width of 1800 pixels and height of 372 pixels.

- Save the composite image as “08_concat_grey.png”.

```
#first assigment computer vistion
#question 8 : Convert the input color image to a cie_xyz color space.
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

# Load the grayscale image
#he cv2.IMREAD_GRAYSCALE flag ensures the image is loaded as a single-
channel grayscale image.
gray_image = cv.imread('/content/04_grayscale.png',
cv.IMREAD_GRAYSCALE)

# Load the Y component of YCbCr
y_ycbcr = cv.imread('/content/06_y_ycbcr.png', cv.IMREAD_GRAYSCALE)

# Load the Y component of CIE XYZ
y_xyz = cv.imread('/content/07_y_xyz.png', cv.IMREAD_GRAYSCALE)
```

```
#The np.hstack() function from NumPy is used to stack the three images
horizontally.
#The images must have the same height for horizontal stacking to work.
#The resulting composite image will have a width equal to the sum of
the widths of the three images
# height equal to the height of the individual images.

composite_image = np.hstack((gray_image, y_ycbcr, y_xyz))

# Save the composite image as "08_concat_grey.png"
cv.imwrite('08_concat_grey.png', composite_image)
```

9) Take the grayscale image in (4), obtain the negative image (i.e., mapping 255 to 0 and 0 to 255).

- Save the image as “09_negative.png”.

```
#first assigment computer vistion
#question 9 :  Stack the grayscale, y_ycbcr, y_xyz components of the
hokiebird image horizontally.
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

# Load the grayscale image
#he cv2.IMREAD_GRAYSCALE flag ensures the image is loaded as a single-
channel grayscale image.
gray_image = cv.imread('/content/04_grayscale.png',
cv.IMREAD_GRAYSCALE)


# Obtain the negative image
#The negative of the grayscale image is obtained by subtracting each
pixel value from 255.
#This operation is performed element-wise on the NumPy array
representing the image.
negative_image = 255 - gray_image

# Save the negative image as "09_negative.png"
cv.imwrite('09_negative.png', negative_image)
```

10) First, crop the original *hokiebird* image into a squared image of size 372 x 372. Then, rotate the image by 90, 180, and 270 degrees and stack the four images (0, 90, 180, 270 degrees) horizontally.

- Save the image as “10_rotation.png”.

```
#first assigment computer vistion
#question 10 :  1-  crop the original hokiebird image into a squared
image of size 372 x 372.
#2- Then, rotate the image by 90, 180, and 270 degrees and stack the
four images (0, 90, 180, 270 degrees) horizontally.
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

#The image is loaded as a NumPy array in BGR format (Blue, Green,
Red).
#The shape of the array is (height, width, channels), where channels
is typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range
0-255).
image=cv.imread("/content/hokiebird.jpg")

#Make a copy to from the orginal image
New_image=image.copy()


# Crop the image to a square of size 372x372
#image.shape returns the dimensions of the image as a tuple.
#For a color image, it returns (height, width, channels).
#For a grayscale image, it returns (height, width).
#If image.shape is (400, 600, 3) (height=400, width=600, channels=3),
then image.shape[:2] gives (400, 600).

height, width = New_image.shape[:2]
start_x = (width - 372) // 2
start_y = (height - 372) // 2
cropped_image = New_image[start_y:start_y+372, start_x:start_x+372]

# Rotate the cropped image by 90, 180, and 270 degrees
#The cropped image is rotated by 90°, 180°, and 270° using
cv2.rotate().
#The rotation flags used are:
```

```
#cv2.ROTATE_90_CLOCKWISE: Rotates the image 90° clockwise.
#cv2.ROTATE_180: Rotates the image 180°.
#cv2.ROTATE_90_COUNTERCLOCKWISE: Rotates the image 90°
counterclockwise (equivalent to 270° clockwise).
rotated_90 = cv.rotate(cropped_image, cv.ROTATE_90_CLOCKWISE)
rotated_180 = cv.rotate(cropped_image, cv.ROTATE_180)
rotated_270 = cv.rotate(cropped_image, cv.ROTATE_90_COUNTERCLOCKWISE)

# Stack the four images horizontally
composite_image = np.hstack((cropped_image, rotated_90, rotated_180,
rotated_270))

# Save the composite image as "10_rotated_stack.png"
cv.imwrite('10_rotated_stack.png', composite_image)
```

11) Create another image with the same size as the *hokiebird* image. First, initialize this image as zero everywhere. Then, for each channel, set the pixel values as 255 when the corresponding pixel values in the *hokiebird* image are greater than 127.

- Save the image as “11_mask.png”.

```
#first assigment computer vistion
#question 11 :  1. kreate a mask image witht he same dimentions of
orginal video AND CHANELES
#first intialize it with 0
```

```
import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

#The image is loaded as a NumPy array in BGR format (Blue, Green,
Red).
#The shape of the array is (height, width, channels), where channels
is typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range
0-255).
image=cv.imread("/content/hokiebird.jpg")

#Make a copy to from the orginal image
New_image=image.copy()
```

```
# Get the dimensions of the hokiebird image
#.shape is for returning the dimensions of the image as a tuple
(height, width, channels).
#These dimensions are stored in variables height, width, and channels
#so we can create a new image of the same size.
height, width, channels = New_image.shape


# Initialize the new image with zeros (black)
#np.zeros((height, width, channels), dtype=np.uint8) creates a new
NumPy array filled with zeros.
#This represents a black image of the same size as the hokiebird
image.
#dtype=np.uint8 ensures that the pixel values are integers between 0
and 255 (standard for 8-bit images).
mask_image = np.zeros((height, width, channels), dtype=np.uint8)


# Set pixel values to 255 where the corresponding hokiebird pixel
values are greater than 127
#creates a boolean mask (a True/False array) where:
#-->True indicates that the pixel value in the hokiebird image is
greater than 127.
#-->False indicates that the pixel value is less than or equal to 127.
#uses this boolean mask to set the corresponding pixels in the
mask_image to 255 (white).
mask_image[New_image > 127] = 255

# Save the resulting image as "11_mask.png"
cv.imwrite('11_mask.png', mask_image)
```

12) Report the mean R, G, B values for those pixels marked by the mask in (11).

```
#first assigment computer vistion
#question 12 :  Report the mean R, G, B values for those pixels marked by
the mask in 11 (the previous )

import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt
```

```
#The image is loaded as a NumPy array in BGR format (Blue, Green, Red).
#The shape of the array is (height, width, channels), where channels is
typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range 0-
255).
image=cv.imread("/content/hokiebird.jpg")


# Load the mask image
#This is the mask image loaded in grayscale mode using
cv2.IMREAD_GRAYSCALE.
#It is a 2D NumPy array with shape (height, width), where each pixel
value is either 0 (black) or 255 (white).
mask_image = cv.imread('/content/11_mask.png', cv.IMREAD_GRAYSCALE)  #
Read as grayscale

# Ensure the mask is binary (0 or 255)
#cv2.threshold() is used to apply a threshold:
#Pixels with values greater than 127 are set to 255.
#Pixels with values less than or equal to 127 are set to 0.

#The result is a binary mask where:
#255 represents the pixels of interest (marked by the mask).
#0 represents the background (pixels to ignore).
#By using _, we explicitly tell Python to ignore the first return value
#and only store the second return value (the thresholded image) in
mask_image.
_, mask_image = cv.threshold(mask_image, 127, 255, cv.THRESH_BINARY)

#mask_image == 255: This creates a boolean array (True/False) where:
#True corresponds to pixels in the mask that are 255 (white).
#False corresponds to pixels in the mask that are 0 (black).
#hokiebird_image[mask_image == 255]: This uses the boolean array to index
into the original image (hokiebird_image).
#It selects only the pixels where the mask is 255.
#The result (masked_pixels) is a 2D NumPy array where:
#Each row represents a pixel.
#Each column represents the B, G, and R channels (in that order).

# Use the mask to select pixels from the original image
masked_pixels = image[mask_image == 255]

# Calculate the mean R, G, B values
mean_r = np.mean(masked_pixels[:, 2])  # Red channel
mean_g = np.mean(masked_pixels[:, 1])  # Green channel
```

```
mean_b = np.mean(masked_pixels[:, 0])  # Blue channel

# Print the results
print(f"Mean R value: {mean_r}")
print(f"Mean G value: {mean_g}")
print(f"Mean B value: {mean_b}")
```

13) Take the grayscale image in (3). Create and initialize another image as all zeros. For each 5 x 5 window in the grayscale image, find out the maximum value and set the pixels with the maximum value in the 5x5 window as 255 in the new image.

- Save the result image as “13_nonmax.png”.

```
 #first assigment computer vistion
#question 13 : 1- Take the grayscale image in (3).
#2- Create and initialize another image as all zeros.
#3- For each 5 x 5 window in the grayscale image,
#find out the maximum value and set the pixels with the maximum value in
the 5x5 window as 255 in the new image.

import numpy as np
import scipy as sp
from scipy import signal
import cv2 as cv
from PIL import Image
import matplotlib.pyplot as plt

#The image is loaded as a NumPy array in BGR format (Blue, Green, Red).
#The shape of the array is (height, width, channels), where channels is
typically 3 for color images (BGR).
#The pixel values are stored as integers (usually uint8 in the range 0-
255).
image=cv.imread("/content/03_swapchannel.png")



# Get the dimensions of the hokiebird image
#.shape is for returning the dimensions of the image as a tuple (height,
width, channels).
#These dimensions are stored in variables height, width, and channels
#so we can create a new image of the same size.
height, width, channels = image.shape
```

```
# Initialize the new image with zeros (black)
#np.zeros((height, width, channels), dtype=np.uint8) creates a new NumPy
array filled with zeros.
#This represents a black image of the same size as the hokiebird image.
#dtype=np.uint8 ensures that the pixel values are integers between 0 and
255 (standard for 8-bit images).
Newimage = np.zeros((height, width, channels), dtype=np.uint8)

# Define the window size (5x5)
window_size = 5

# Iterate over the image in 5x5 windows
#Two nested loops are used to iterate over the image in 5x5 windows.
#The outer loop (y) iterates over the rows (height), and the inner loop
(x) iterates over the columns (width).
#The loops stop at height - window_size + 1 and width - window_size + 1
to ensure the 5x5 window stays within the image boundaries.

for y in range(0, height - window_size + 1):
    for x in range(0, width - window_size + 1):
        # Extract the 5x5 window
        #xtracts a 5x5 block of pixels from the grayscale image, starting
at position (y, x).
        window = image[y:y + window_size, x:x + window_size]

        # Find the maximum value in the window
        #computes the maximum pixel value in the 5x5 window.
        max_value = np.max(window)

        # Find the coordinates of the maximum value in the window
        #np.where(window == max_value) returns the coordinates of all
pixels in the window that have the maximum value.
        #max_coords is a tuple containing two arrays:
            #The first array contains the row indices.
            #The second array contains the column indices.
        max_coords = np.where(window == max_value)

        # Set the corresponding pixels in the output image to 255
        for i in range(len(max_coords[0])):
            Newimage[y + max_coords[0][i], x + max_coords[1][i]] = 255

# Save the result image
cv.imwrite('13_nonmax.png', Newimage
```