task_1_pattern_matching

October 3, 2023

1 Pattern Matching

1.1 Part 1) Handling strings and basic python operations

In this first task of the lab I am required to load in basic data types and complete some manipulation tasks on them. Specifically I will be working with strings and slicing then changing them into floats and performing mathematical operations on them.

[ ]:
```
statement = "The value of pi is approximately 333 divided by 106" #given␣
 ↪statement from lab script

numerator = float(statement[32:36]) #extracting the numerator and denominator␣
 ↪from the statement by slicing the string and converting to float
denominator = float(statement[47:51])

pi = numerator/denominator

print(f'The approximate value of pi obtained by dividing 333 by 106 is {pi}')
```

```
The approximate value of pi obtained by dividing 333 by 106 is 3.141509433962264
```

1.2 Part 2) loading and manipulating image files

The aims o this task is to load in an image file containing an RGB image of the univeristies logo, crop the image extract the RGB values of the image and then plot the image using matplotlib.

I will complete this task using the numpy and matplotlib libraries. Prior knowledge of slicing and indexing arrays is assumed here.

The cell below imports all the required libraries for this task and sets up the sytle sheet that will be used for all plots in this notebook. (This will be used in the two following notebooks and so will be repeated there)

[ ]:
```
import re
import numpy as np
import os
import matplotlib.pyplot as plt
plt.style.use('../report.mplstyle') #importing the style sheet
```

In the cell below the image files are read in and the shape and RGB values of the image are printed using matplotlib and array indexing to slice the arrays, the image is displayed with 100 pixels cropped from all sides and the axes removed to satisy the aims of this task.

```
[ ]: # while open
rgb_img = plt.imread("patterns/glasgow_rgb.png") #reading in the image
rgb_val = (rgb_img[0,0]) #extracting the rgb values for the top left pixel


print("The shape of the image is", rgb_img.shape, 'The shape here tell us that␣
 ↪the first value `338` is the \nnumber of rows in the image, the second␣
 ↪value`600` is the number of columns and the third value tells us \nhow many␣
 ↪colour channels are present.')
print(f"The RGB values for the pixel in the top left are, {rgb_val}, where the␣
 ↪first \nvalue is the red value, the second value is the green value and the␣
 ↪third value is the blue value.")


plt.figure(figsize=(10,10))
plt.imshow(rgb_img)
plt.ylim(238,100)   #setting the limits of the y axis to crop the image by 100␣
 ↪pixels from the top and bottom
plt.xlim(100,500)   #setting the limits of the x axis to crop the image by 100␣
 ↪pixels from the left and right
plt.axis('off') #removing the axis
plt.show()
```

```
The shape of the image is (338, 600, 3) The shape here tell us that the first
value `338` is the
number of rows in the image, the second value`600` is the number of columns and
the third value tells us
how many colour channels are present.
The RGB values for the pixel in the top left are, [0.         0.22352941
0.39607844], where the first
value is the red value, the second value is the green value and the third value
is the blue value.
```

University of Glasgow
VIA VERITAS VITA

1.3 Part 3) Hidden Instructions

The aims of this task are to sort through images files that have two different patterned naming conventions and then compile the images of each naming convention into one image and find the difference between the two images. This should then reveal a message hidden within the images. To complete this task I will be using functions from the os and re libraries to sort the files into two different folders and then use numpy and matplotlib to compile, find the difference and plot the images.

```
[ ]: # group_1 = (os.listdir("patterns/group_1/"))
     # group_2 = os.listdir("patterns/group_2/")

     # g1 = []
     # for i in np.arange(len(group_1)):

     #     g1.append(plt.imread("patterns/group_1/"+group_1[i]))

     # G1 = (sum(g1))

     # plt.figure(figsize=(12,4))
     # plt.set_cmap("viridis")
     # plt.imshow(G1)

     # plt.show()



     # g2 = []
     # for i in np.arange(len(group_2)):

     #     g2.append(plt.imread("patterns/group_2/"+group_2[i]))

     # G2 = (sum(g2))

     # plt.figure(figsize=(12,4))
     # plt.imshow(G2, interpolation="sinc")
     # plt.show()

     # img_final = abs(abs(G1 - G2))

     # plt.figure(figsize=(24,8))
     # plt.imshow(img_final, interpolation="hamming")

     # plt.show()
```

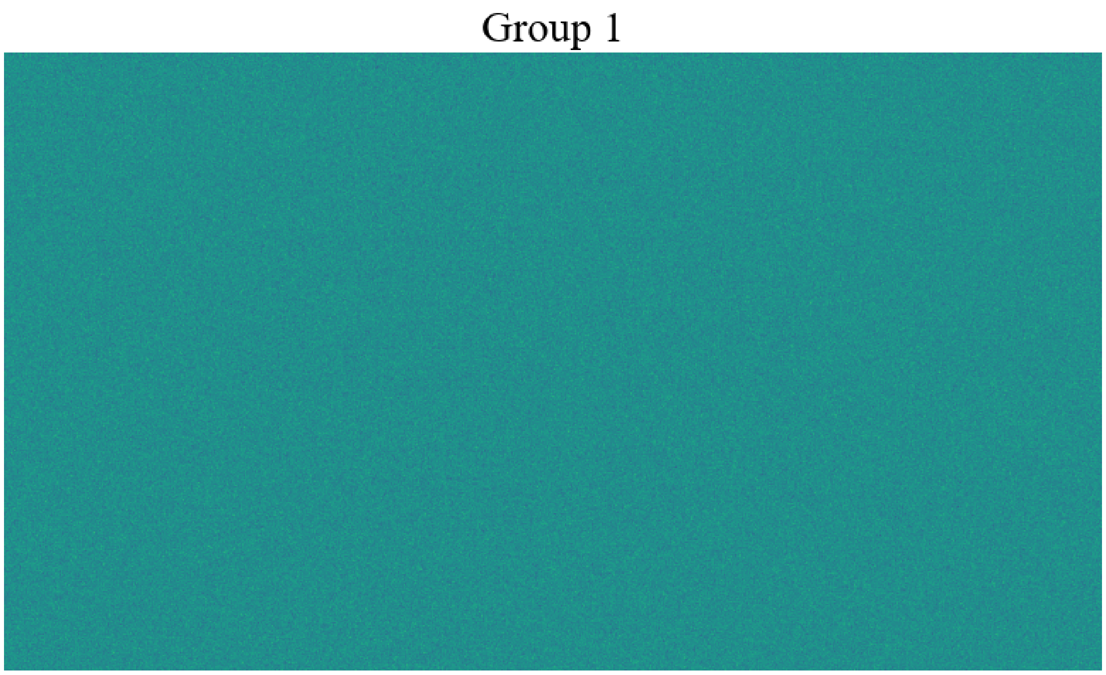
The cell above was an initial attempt at this task, however it was not successful, it produced a final image that still contained a lot of noise and the hidden message was not clear. I have left this cell in the notebook to show my thought process and how I approached the task. I believe it may now work with some refactoring as I later identified that one issue it was not working was that two of the image files were missing from the directory I was searching through to sort them. The cause of this is not downloading them all to my local machine as this code is not running on the JupyterHub server.

The cell below has the final and working attempt of this task, I started by defining a path to the files i needed to sort and creating a list of these file names. This was then used to iterate through first sorting the files of the convention hidden_xxxx.png and second the images called yyy_hidden_xxxx.png (where xxxx is a number and yyy is a string of letters). These were then compiled into two folders and the images were read in and stored in dictionaries. The arrays in these dictionaries were then summed together and the difference of the two was found and plotted. This revealed the message embedded in the images.

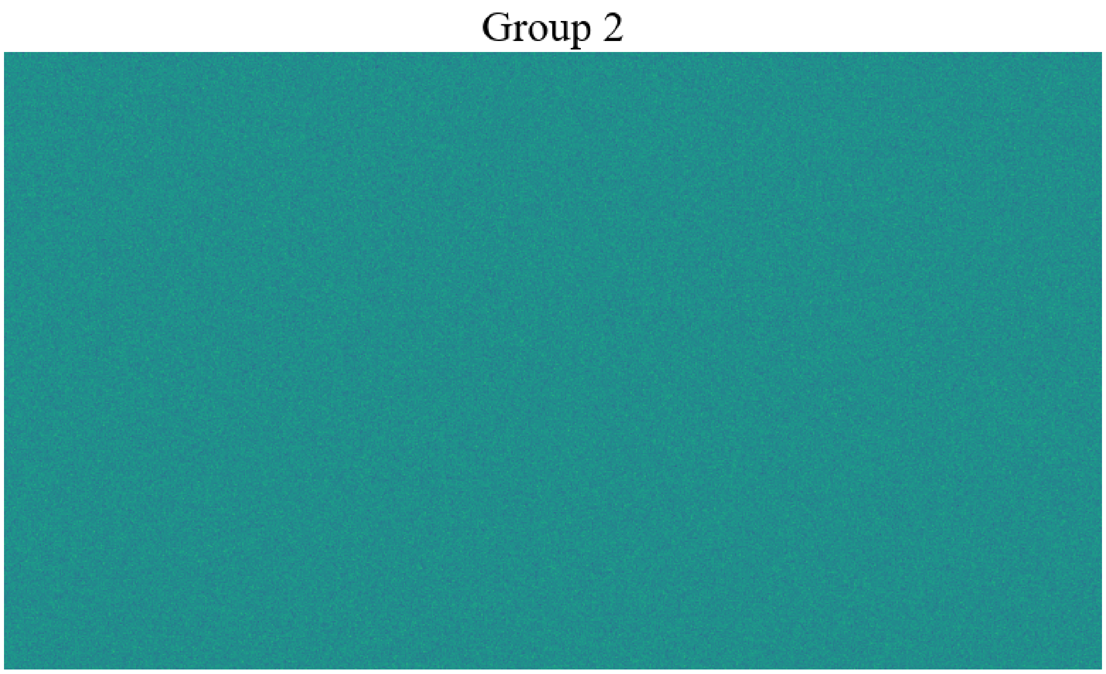
```
[ ]: path = "patterns/"  #define a path to the patterns folder
     files = os.listdir("patterns/") #list all the files in the patterns folder


     for file in files: #sort all the files starting with hidden into group_1
         if file.startswith("hidden"):
             print(file)
             os.rename(path+file, path+'group_1/'+file)

     for file in files: #sort all the files containing hidden into group_2
         if re.search("hidden", file):
             print(file)
             os.rename("patterns/"+file, path+"group_2/"+file)

     group_1 = os.listdir(path+"group_1/")
     group_2 = os.listdir(path+"group_2/") #list all the files in the two groups


     dic_g1 = {} #create two empty dictionaries to store the images
     dic_g2 = {}

     for i in np.arange(len(group_1)):  #read in the images and store them in the␣
      ↪dictionaries with associated keys
         dic_g1[group_1[i]] = plt.imread(path+"group_1/"+group_1[i])

     for i in np.arange(len(group_2)):
         dic_g2[group_2[i]] = plt.imread(path+"group_2/"+group_2[i])


     img_g1 = (np.zeros((1080,1920))) #create two empty arrays to store the images
     img_g2 = (np.zeros((1080,1920)))

     for i in np.arange(len(dic_g1)): #add the images together to create a single␣
      ↪image
```

```
    img_g1 += dic_g1[group_1[i]]

for i in np.arange(len(dic_g2)):
    img_g2 += dic_g2[group_2[i]]

print('This is the image of the ten images in group 1 added together')
plt.figure(figsize=(10,10)) #plot the two seperate images
plt.imshow(img_g1)
plt.axis('off')
plt.title('Group 1')
plt.show()

print('This is the image of the ten images in group 2 added together')
plt.figure(figsize=(10,10))
plt.imshow(img_g2)
plt.axis('off')
plt.title('Group 2')
plt.show()



decrypt_img = img_g1-img_g2 #subtract the two images to decrypt the hidden image

print('This is the final decrypted image revealing the intructions for the next␣
 ↪task')
plt.figure(figsize=(10,10))  #plot the decrypted image
plt.imshow(decrypt_img)
plt.axis('off')
plt.title('Decrypted Image')
plt.show()
```

```
This is the image of the ten images in group 1 added together
```

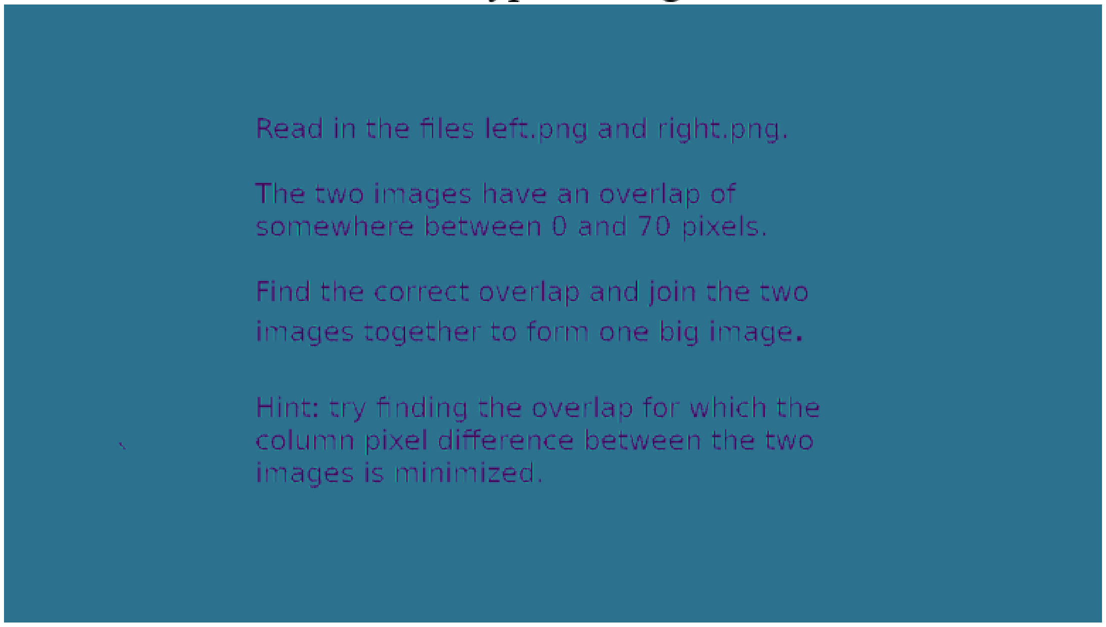


This is the image of the ten images in group 2 added together



This is the final decrypted image revealing the intructions for the next task

Decrypted Image


Read in the files left.png and right.png.

The two images have an overlap of
somewhere between 0 and 70 pixels.

Find the correct overlap and join the two
images together to form one big image.

Hint: try finding the overlap for which the
column pixel difference between the two
images is minimized.

1.4 Part 4) Combining two images and finding where they overlap

The aims for this task are to read in two images left.png and right.png that have x amount of columns of pixels that are the same and find where they overlap then combine them into one image. For this task two images are provided that each have some overlap of eachother and the task is is to find where they overlap and combine them into one image. This was completed by defining a function to find the difference between the two images and iterating through each column of pixels until the lowest difference was found. The images were both cropped to this column and combined using concatenation and finally displayed.

```
[ ]: image_1 = (plt.imread('patterns/left.png'))
     image_2 = (plt.imread('patterns/right.png'))   #Read in the two images and save␣
      ↪them as a variable

     def difference(left, right):  #define a function for the difference of two␣
      ↪arrays
         diffren = left - right
         return  abs(sum(diffren))

     diffr = []  #create an empty array to store the differences
     for i in np.arange(0,70,1):  #loop over the columns of the two images and␣
      ↪calculate the difference
         diff = difference(image_1[:,-i],image_2[:,i])
         diffr.append(diff)
```

```
left_img = image_1[:,:(image_1.shape[1]-np.argmin(diffr))]   #slice the two␣
 ↪images to remove the section that is duplicated
right_img = image_2[:,np.argmin(diffr):]

final_img = np.concatenate((left_img,right_img),axis = 1)  #concatenate the two␣
 ↪images together to create the final image

plt.figure(figsize=(10,10))  #plot the final image
plt.imshow(final_img)
plt.axis('off')
plt.title('Final Image')
plt.show()
```

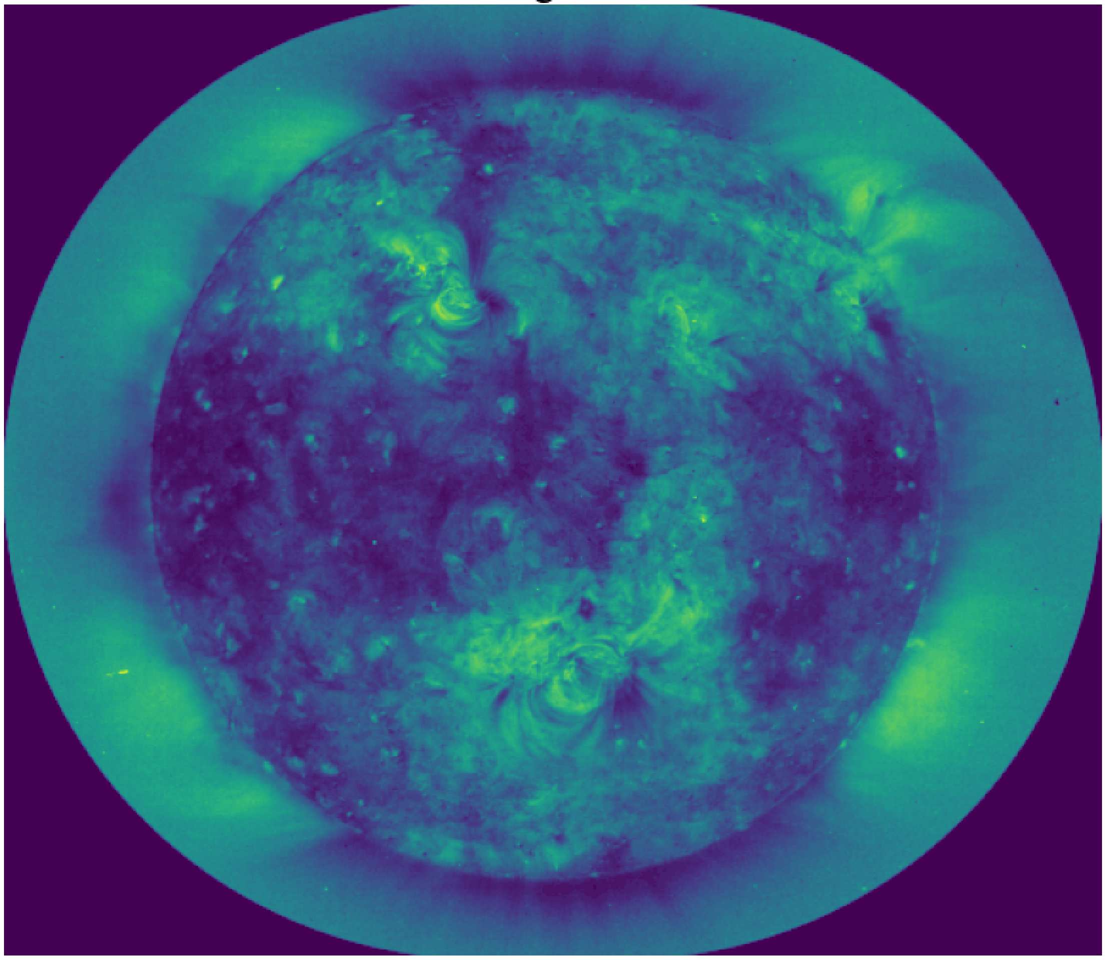
Final Image

1.5 Showimg Generality of finding the overlap of two images

The aims of this task are the same as the previous one but using a more complex set of images to show the generality of the method used to find the overlap of the images. In this next task two images of a star are given that have some overlap and the task is to find where they overlap and combine them, this is done using the same method as above but instead of a single loop finding where the overlap is and interating over the images at the same time one image is held constant while the other is iterated over until they match this is done for both sides then they are stitched together and displayed.

```
[ ]: soho_left = plt.imread("patterns/soho_left.png")
     soho_right = plt.imread("patterns/soho_right.png")   #Read in the two images␣
      ↪and save them as a variable

     diffr =[]
     for i in np.arange(0,200,1):   #loop over the columns of the two images and␣
      ↪calculate the difference in the first image
         for j in np.arange(0,200,1):
             diff = difference(soho_left[:,-i],soho_right[:,j])
             diffr.append(diff)

     left_crop = int(np.floor(np.argmin(diffr)/200))  #find the index of the minimum␣
      ↪difference and slice the images to remove the duplicated section

     diffr =[]
     for i in np.arange(0,200,1):
         for j in np.arange(0,200,1): #loop over the columns of the two images and␣
      ↪calculate the difference in the second image
             diff = difference(soho_left[:,-j],soho_right[:,i])
             diffr.append(diff)

     right_crop = int(np.floor(np.argmin(diffr)/200))

     left_img = soho_left[:,:(soho_left.shape[1]-left_crop)]  #slice the two images␣
      ↪to remove the section that is duplicated
     right_img = soho_right[:,right_crop:]

     final_img = np.concatenate((left_img,right_img),axis=1)  #concatenate the two␣
      ↪images together to create the final image

     plt.figure(figsize=(10,10))   #plot the final image
     plt.imshow(final_img)
     plt.axis('off')
     plt.title('SOHO image of the Sun')
     plt.show()
```

SOHO image of the Sun



1.6 End of Task 1 - Pattern Matching