Computer Vision
Degree in Information Technology
2º Semester 2020/2021

Worksheet 2

Goals:

- Image processing – pixel manipulation and geometric operations
- Consolidate Python knowledge to be applied in computer vision

Exercises

Part I – Image Reading and Pixel Manipulation

Import the cv2 library and follow the next steps. Take notes and store the solution of each exercise in a different python file.

1- Run the following code and answer the questions:
   a. What is the width in pixels? How can we get that information in Python?
   b. What is the height in pixels? How can we get that information in Python?
   c. How many channels the image contains? How can we get that information in Python?
   d. In this code, what is the data type of the variable `image`?
   e. How many channels would a Gray image have?
   f. In that situation, what would be the data type of the variable `image`?

```
# Import libraries
import cv2

# Image Reading with imread()
image = cv2.imread("DCT_img.jpg")

print('Width in pixels: ', end='')
print(image.shape[1]) #image width
print('Height in pixels: ', end='')
print(image.shape[0]) #image height
print('Amount of channels: ', end='')
print(image.shape[2])

# Show the image with imshow
cv2.imshow("Window name", image)
cv2.waitKey(0) #waiting to press a key

# Save a new image in your computer with imwrite()
cv2.imwrite("output.jpg", image)
```

Code 1- Image Reading

2- Analyse the next pixel representation in terms of coordinate system and answer the following questions:

a. How many channels we have?

b. What is the pixel in (2,8)?

c. What is the pixel in (7,5)?

d. What is the pixel in (4,5)?

	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
R0	0	0	0	0	0	0	0	0	0	0
R1	0	50	50	50	50	50	50	50	50	0
R2	0	50	100	100	100	100	100	100	50	0
R3	0	50	100	150	150	150	150	100	50	0
R4	0	50	100	150	200	200	150	100	50	0
R5	0	50	100	150	200	200	150	100	50	0
R6	0	50	100	150	150	150	150	100	50	0
R7	0	50	100	100	100	100	100	100	50	0
R8	0	50	50	50	50	50	50	50	50	0
R9	0	0	0	0	0	0	0	0	0	0

Figure 1- Coordinate System for pixels

3- Using the coordinate system for pixels, it is possible to manipulate and read the pixel information individually.

   a. How can we access the actual pixel values in Python? Write and run the following code.

   b. As we could notice, images are Numpy matrixes returned by `imread`. What is the pixel which we are accessing?

   c. The tuple `(b, g, r)` is receiving what?

   d. What is the result?

   e. Which colour is 255, 255, 255?

   f. Analyse the order of the colours. **The order is BGR and not RGB**

   g. Add show image to the code and check if 255,255,255 is white (do not forget to add `waitkey(0)`)

```
import matplotlib.pyplot as plt
import cv2

image = cv2.imread("DCT_img.jpg ")
(b, g, r) = image[0, 0]
print("Pixel at (0, 0) - Red: {}, Green: {}, Blue: {}".format(r, g, b))
```

Code 2- Pixels Analysis

4- Analyse the next piece of code.

a) What does it mean?

b) Add show image to the code and check your conclusions.

```
import cv2

image = cv2.imread("DCT_img.jpg ")
image[10, 10] = (0, 0, 255)
(b, g, r) = image[10, 10]

print("Pixel at (0, 0) - Red: {}, Green: {}, Blue: {}".format(r, g, b))
```

Code 3- Pixel Analysis

5- Run the following piece of code.

a) Analyse the `corner` variable. What does it represent?

b) Increase the `corner` size, *i.e.*, it takes the first 300 pixels of y and 200 of x concerning pixel coordinate system. What is the result, *i.e.*, the new image?

c) Apply other slicing examples to the image matrix according to your will. Ex. 100:200 and 300:400, etc.

d) Save the new image in your computer.

e) What is the goal of the next lines of code?

f) Change the green colour for red.

```
import matplotlib.pyplot as plt
import cv2

image = cv2.imread("DCT_img.jpg ")
corner = image[0:100, 0:100]

cv2.imshow("Corner", corner)
cv2.waitKey(0)

image[0:100, 0:100] = (0, 255, 0)
cv2.imshow("Updated", image)
cv2.waitKey(0)
```

Code 4 - Pixel Manipulation

6- Let us play a bit more with pixels. Run the next code and analyse it. Note that % is the division remainder.

a) Change the colour.

b) Apply `image[y, x] = (x%256,y%256,x%256)` and see the result

c) Substitute the for cycles and see the result:

```
for y in range(0, image.shape[0], 10): #rows
        for x in range(0, image.shape[1], 10):
            image[y:y+5, x: x+5] = (0,255,255)
```

d) Ideas for practical work to do after classes:

a. Create your own formulas and play with the image

b. Experiment the previous exercises with other images

c. Report your ideas to be presented as the first practical work

```
import matplotlib.pyplot as plt
import cv2

image = cv2.imread("DCT_img.jpg ")

for y in range(0, image.shape[0], 1): #rows
     for x in range(0, image.shape[1], 1): #colomns
          image[y, x] = (0,(x*y)%256,0)

cv2.imshow("Modified image", image)
cv2.waitKey(0)
```

Code 5 - Pixel Manipulation

Part II – Drawing

The above exercises provide examples concerning pixel manipulation. Now, we will demonstrate how to draw shapes on images. With slicing it is possible to create squares and rectangles. However, OpenCV provides convenient, easy-to-use methods to draw multiple shapes on an image: `cv2.line`, `cv2.rectangle`, and `cv2.circle`.

1- Write and run the following code in a python file. Consider the next lines and add them as Python comments in the code in the right place:

   a. Initialise our canvas as a 300x300 with 3 channels, Red, Green, and Blue, with a black background

   b. Draw a green line from the top-left corner of our canvas to the bottom-right

   c. Now, draw a 3-pixel thick red line from the top-right corner to the bottom-left

   d. Draw a green 50x50 pixel square, starting at 10x10 and ending at 60x60, 2 pixels thick

   e. Draw another rectangle, this time we will make it red and 5 pixels thick

   f. Let us draw one last rectangle: blue and filled in

```
import matplotlib.pyplot as plt
import numpy as np
import cv2

canvas = np.zeros((300, 300, 3), dtype = "uint8")

green = (0, 255, 0)
cv2.line(canvas, (0, 0), (300, 300), green)
cv2.imshow("Canvas", canvas)
cv2.waitKey(0)

red = (255, 0, 0)
cv2.line(canvas, (300, 0), (0, 300), red, 3)
cv2.imshow("Canvas", canvas)
cv2.waitKey(0)

cv2.rectangle(canvas, (10, 10), (60, 60), green, 2)
cv2.imshow("Canvas", canvas)
cv2.waitKey(0)

cv2.rectangle(canvas, (50, 200), (200, 225), red, 5)
cv2.imshow("Canvas", canvas)
cv2.waitKey(0)

blue = (0, 0, 255)
cv2.rectangle(canvas, (200, 50), (225, 125), blue, -1)
cv2.imshow("Canvas", canvas)
cv2.waitKey(0)
```

Code 6- Drawing shapes

2- Write and run the following code in a python file:

```
import numpy as np
import cv2

canvas = np.zeros((300, 300, 3), dtype = "uint8")
(centerX, centerY) = (canvas.shape[1] // 2, canvas.shape[0] // 2)
white = (255, 255, 255)

for r in range(0, 150, 25):
     cv2.circle(canvas, (centerX, centerY), r, white)

cv2.imshow("Canvas", canvas)
cv2.waitKey(0)
```

Code 7- Drawing shapes

- To draw circles, we must use `cv2.circle` function.
- First, we calculate two variables: `centerX` and `centerY`. These two variables represent the (x, y) coordinates of the centre of the image. We calculate the centre by examining the shape of our NumPy array, and then dividing by two.
- Then, we choose the white colour for the lines.
- We loop over several radius values, starting from 0 and ending at 150 (since the range function is exclusive), incrementing by 25 at each step.
- The first argument is our `canvas`, second argument is the tuple of the centres, third argument is the radius of the circle, the fourth argument is the colour of our circle.

Sometimes, this kind of shape is important to analyse the distance between objects. Play with the values in the code and try to understand all parameters.

3- A useful tool is to write texts on images. For that, we need to take as starting point the pixel coordinate where the text will be introduced. To test this functionality, write a Python program with the following piece of code. Play with texts and positions.

```
import numpy as np
import cv2

image = cv2.imread('DCT_img.jpg')
font = cv2.FONT_HERSHEY_SIMPLEX

cv2.putText(image,'Computer Vision',(0,30), font, 1,(0,0,0),2,cv2.LINE_AA)
cv2.imshow("Image with Text", image)
cv2.waitKey(0)
```

Code 8-Insert Text

4- Ideas for the practical work to do after classes:

   a. Pick images according your tastes

   b. Experiment the previous exercises with other images

   c. Report your ideas to be presented as the first practical work

Part III – Geometric Operations

1- Translation

Translation is the shifting of an image along the x and y axis. Using translation, we can shift an image up, down, left, or right, along with any combination of the above. Write and run the following code in a python file:

```
import numpy as np
import cv2

# Load the image and show it
image = cv2.imread('DCT_img.jpg')
cv2.imshow("Image", image)
cv2.waitKey(0)

# Store height and width of the image
height, width = image.shape[:2]

quarter_height, quarter_width = height / 4, width / 4

T = np.float32([[1, 0, quarter_width], [0, 1, quarter_height]])

# We use warpAffine to transform
# the image using the matrix, T
img_translation = cv2.warpAffine(image, T, (width, height))

cv2.imshow('Translation', img_translation)
cv2.waitKey()
```

Code 9 - Translation

- First, we import the packages we will make use of.
- Then, we define our translation matrix T. This matrix tells us how many pixels to the left or right, and up or down, the image will be shifted.

- Our translation matrix M is defined as a floating-point array. The first row of the matrix is `[1,0, tx]`, where `tx` is the number of pixels we will shift the image left or right. Negative values of `tx` will shift the image to the left and positive values will shift the image to the right.
- Then, we define the second row of the matrix as `[0,1, ty]`, where `ty` is the number of pixels we will shift the image up or down. Negative value of `ty` will shift the image up and positive values will shift the image down.
- The translation takes place using the `cv2.warpAffine` function. The first argument is the image we wish to shift, the second argument is our translation matrix T, and the third argument is the dimensions (width and height) of our image.

2- Rotation

Rotation is rotating an image by some angle θ. Write and run the following code in a Python file. Then, read the next points to understand the code.

```
import numpy as np
import cv2

# Load the image and show it
image = cv2.imread("DCT_img.jpg")
cv2.imshow("Image", image)
cv2.waitKey(0)

#Grab the dimensions of the image and calculate the center of the image
center = (image.shape[1] // 2, image.shape[0] // 2)

#Rotate our image by 45 degrees
M = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
cv2.imshow("rotated", rotated)
cv2.waitKey(0)

#Rotate our image by -90 degrees
M = cv2.getRotationMatrix2D(center, -90, 1.0)
rotated = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
cv2.imshow("rotated", rotated)
cv2.waitKey(0)
```

Code 10 - Rotation

- Import the packages you need.
- When you rotate an image, you need to specify around which point you want to rotate. In most cases, you will want to rotate around the centre of an image; Integer division is used here, denoted as // to ensure we receive whole integer numbers.
- Define a matrix to rotate the image: `cv2.getRotationMatrix2D`. This function has as first argument the point at which you want to rotate the image around. The second argument is the number of degrees, and the third argument is the scale of the image. 1.0 means that we are using the same dimensions of the image.
- Then apply the rotation to your image using the `cv2.warpAffine` method.

3- Resizing

Resizing is changing the size of an image. Write and run the following code in a python file.

- Import the packages
- When resizing an image, you need to keep in mind the aspect ratio of the image. The aspect ratio is the proportional relationship of the width and the height of the image:
    - When you define the new image width of 150 pixels you divide the new width (150) by the old width (`image.shape[1]`).
    - When you define the new image height of 50 pixels you divide the new height (50) by the old height (`image.shape[0]`).
- Compute the new dimensions of the image:
    - The width of the new image will be 150 pixels. The height is the old height multiplied by the ratio and converted to an integer.
    - The height of the new image will be 50 pixels. The width is the old width multiplied by the ratio and converted to an integer.

The resizing is performed by the `cv2.resize` function. The first argument is the image you want to resize and the second is your computed dimensions for the new image. The last parameter is your interpolation method. It is the algorithm working behind the scenes to handle how the actual image is resized.

```
import numpy as np
import cv2

# Load the image and show it
image = cv2.imread("DCT_img.jpg")
cv2.imshow("image", image)
cv2.waitKey(0)

#Let's make our new image have a width of 150 pixels
#Aspect ratio
r = 150.0 / image.shape[1]
dim = (150, int(image.shape[0] * r))

#Resize
resized = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
cv2.imshow("image",resized)
cv2.waitKey(0)

#Let's make our new image have a height of 50 pixels
#Aspect ratio
r = 50.0 / image.shape[0]
dim = (int(image.shape[1] * r), 50)

#Resize
resized = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
cv2.imshow("image",resized)
cv2.waitKey(0)
```

Code 11- Resizing

4- Flipping

Flipping is moving an image by a mirror-reversal of an axis. Write and run the following code in a python file:

```
import numpy as np
import cv2

# Load the image and show it
image = cv2.imread("DCT_img.jpg")
cv2.imshow("image", image)
cv2.waitKey(0)

# Flip the image horizontally
flipped = cv2.flip(image, 1)
cv2.imshow("image",flipped)
cv2.waitKey(0)

# Flip the image vertically
flipped = cv2.flip(image, 0)
cv2.imshow("image",flipped)
cv2.waitKey(0)

# Flip the image along both axes
flipped = cv2.flip(image, -1)
cv2.imshow("image",flipped)
cv2.waitKey(0)
```

Code 12- Flipping

- Flipping an image is accomplished by making a call to the `cv2.flip` function.
- The function requires two arguments: the image you want to flip and a flip code that is used to determine how you are going to flip the image:
    - 1 - flip the image horizontally, around the y-axis.
    - 0 - flip the image vertically, around the x-axis.
    - Negative - flips the image around both axes.

5- Cropping

Cropping is to remove/select parts of an image. We can accomplish image cropping by using NumPy array slicing. Write and run the following code in a python file:

```
import numpy as np
import cv2

# Load the image and show it
image = cv2.imread("DCT_img.jpg")
cv2.imshow("image", image)
cv2.waitKey(0)

#Cropping
# Start at Y=30 and end at Y=190. Start at X=190 and end at X=350
cropped = image[30:190 , 190:350]
cv2.imshow("image",cropped)
cv2.waitKey(0)
```

Code 13-Cropping

- We extract a rectangular region of the image, starting at (190,30) and ending at (350,190).
- Remember that OpenCV represents images as NumPy arrays with the height first and the width second. This means that we need to supply our y-axis values before our x-axis.
- To perform our cropping, NumPy expects four indexes:
    - 1. Start y: The starting y coordinate. In this case, we start at y = 30.
    - 2. End y: The ending y coordinate. We will end our crop at y = 190.
    - 3. Start x: The starting x coordinate of the slice. We start the crop at x = 190.
    - 4. End x: The ending x-axis coordinate of the slice. Our slice ends at x = 350.

6- A bird's-eye view is an elevated view of an object from above, with a perspective as though the observer were a bird. To help with this transformation, download the python file from Moodle (transform.py) which contains the perspective computation.

Then, write and run the following example in a python file:

```
import transform
import numpy as np
import cv2

# Load the image and show it
image = cv2.imread("DCT_img.jpg")
cv2.imshow("image", image)
cv2.waitKey(0)

pts = np.array([(73, 239), (356, 117), (475, 265), (187, 443)])
warped = transform.four_point_transform(image, pts)
cv2.imshow("Original", image)
cv2.imshow("Warped", warped)
cv2.waitKey(0)
```

Code 14 - Bird's eye view

This code first loads the image, then selects 4 points from each corner of the image and obtains the elevated view of our image.

Do not forget! In each exercise play with the values to understand how it works.

Good work!