The Reader

If you find any error, mistake or have any suggestions for the improvement of this text, please email me at sajidiqbal.pk@gmail.com

2 Chapter Two

Solved Programs

This is the complement document of theory chapter, Chapter 2. In this work, we will present the application of image processing concepts introduced in theory part. It is tried to present, most of the things in Python as doing the things from scratch help the learner to have in depth understanding. However, OpenCV APIs are used where complex concepts are introduced and where we think that the student will not be able to implement the algorithms at this early stage.

The programs are listed in same order as referenced in theory part. The APIs, either from OpenCV or any other library, if are used in program, they are explained first followed by their implementation. The reader is encouraged to consult documentation of relevant APIs available online on their respective sites.

2.0 Creating gray scale matrix

Tensor: An array could have different dimensions. 1D array with only one member is known as “**scalar**”, 1D array with more than one values is called “**Vector**”, a 2D array is known as matrix, 3D array is called “**cube**”. It is difficult to find names for higher dimensional arrays so they are given a single name called “**Tensor**”. To generalize the term, scalar, Vector, Cube are also called **Tensors**. So whenever in this document, we use term tensor, you should know that we are talking about an array of some dimension.

APIs used

- **np.zeros(*shape*, *dtype=float*, *order='C'*)**

  This API creates a tensor of specified shape and each matrix member is assigned value zero.

  Parameters
  - **Shape:** This parameter specifies the shape of tensor to be generated i.e. rows and columns
  - **dtype:** this parameter specifies the data type of tensor to be generated. If no datatype is given, by default “float” is used.
  - **Order:** Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

  Returns:
  - **ndarray:** n-dimensional array of zeros with given shape, data type and order.

- **print(*arguments*)**

  This is versatile method of python that can print data of multiple type. In short whatever you give it, it prints.

- **img.shape**

This is the property of matrix (image) that tells the number of elements in each dimension. For example, if img is matrix of 16x16, the output of this statement will be 16 16.

Parameters

- This is property so no parameters are there

Returns:

- **1D array:** The returned array contains the count of elements in each dimension i.e. [16,16].

- **object.astype(<type>)**

This general object method converts the data type of object. For example, if datatype of object is float, you can convert it to integer.

Parameters

- **dtype**: new data type

Returns:

- **Object:** This returns a similar object whose data type is changed. For example, the object (tensor) on which this method is called has float data type and new data type is integer, so returned object (tensor) will have integer data type.

- **cv.resize(src,dsize,dst,fx,fy,interpolation)**

This API changes the size of src image to the specified size $dsize$.

Parameters

- **src**: [required] source image
- **dsize**: [required] desired size for the output image
- **fx**: [optional] scale factor along the horizontal axis
- **fy**: [optional] scale factor along the vertical axis
- **interpolation**: [optional] flag that tells which interpolation method to be used. At this stage you need not be concerned with interpolation.

Returns:

- It returns a resized image as specified by the $dsize$ parameter.

- **cv.imshow(winname,image)**

This method creates an image window that shows the image “$image$”.

Parameters

- **winname**: This is string data that specifies the name of windows.
- **Image**: This is image matrix to be displayed as image

Returns:

- This OpenCV API returns nothing.

- **cv.waitKey(<delay>)**

The function $waitKey$ waits for a key event infinitely (when $\boldsymbol{delay \leq 0}$) or for delay milliseconds, when it is positive. Since the OS has a minimum time between switching threads, the function will not wait exactly delay ms, it will wait at least delay ms, depending on what else is running on your computer at that time.

Parameters

- **delay**: This number specifies the time for which image window will be on display. If 0 time is specified, the windows stays on display till the user herself closes it.

Returns:

- **keycode**: It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.

P2-00: Grayscalematrix.py

```
1  import cv2 as cv
2  import numpy as np
3  img=np.zeros([16,16])
4  print(img.shape)
5  for i in range(16):
6      for j in range(16):
7          img[i,j]=i*16+j
8  print(img)
9  img=img.astype(np.uint8)
10 cv.imshow("GrayMatrix 16x16",img)
11 img=cv.resize(img,(256,256))
12 cv.imshow("GrayMatrix 256x256",img)
13 cv.waitKey(0)
```

Program Description

We explain the program line by line.

- ***Line* 1, 2**, imports the library files that we want to use in our program. Statement 1 serves two purposes 1-it imports library named "***cv2***" and renames it as "***cv***" to use in our program. This renaming is done in order to avoid typing long library names on every use in the program. So, using this way, we provide a short name to our library. On line 2 *numpy* library is imported and its reference name is shortened to *np*.
- ***Line* 3**: We create of matrix of 16 rows and 16 columns. Each element of this matrix is assigned zero value. As we have not provided the data type of matrix to be created so its default data type is float.
- ***Line* 4**: It prints the *shape* of our matrix. The shape is collection of two values i.e. rows and columns. In our case it is 16,16.
- ***Line* 5, 6, 7**: The nested loop assigns values ranging from 0-255. The matrix after value assignment looks like the following matrix

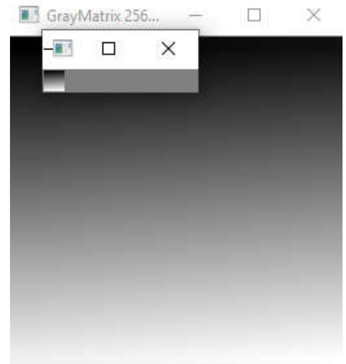
$$A = \begin{bmatrix} 0 & \mathbf{1} & \dots & \mathbf{15} \\ \mathbf{16} & \mathbf{17} & \dots & \mathbf{31} \\ \dots & \dots & \dots & \dots \\ 240 & 241 & \dots & 255 \end{bmatrix}$$

- ***Line* 8**: This statement prints all the values of newly generated matrix i.e. the matrix as shown above
- ***Line* 9**: OpenCV requires that the data type should be integer of the image to be displayed. So here we convert the data type of matrix from float to integer.
- ***Line* 10** This line uses the OpenCV API *cv.imshow* to show the matrix as image.
- ***Line* 11**: As the generated matrix has very small size and can't be visualize properly. So, we increase its size from 16x16 to 256x256.

- ***Line* 12** This line uses the OpenCV API $cv.imshow$ to show the enlarged matrix as image.
- ***Line* 13**: OpenCV method "***waitKey***" is used to stop/ hold the output windows.

Output

(16, 16)
[[0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15.]
[16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31.]
....



You can see that 16x16 window is very small and is shown above 256x256 windows however the shades of both are same.

2.1 Loading and Displaying an image in OpenCV

Before looking at python programs, we will provide some details about key APIs used in the programs.

- **cv.imread(path,flag)**

This method reads an image whose path is given in path parameter.

Parameters

- **path**: It is complete path of the image to be read including file name extension
- **flag**: [optional] this value could be any one of these
  - **cv.IMREAD_COLOR**: It loads a color image. Any transparency of image will be neglected. It is the default flag.
  - **cv.IMREAD_GRAYSCALE**: Loads the image in grayscale mode.
  - **cv.IMREAD_UNCHANGED:** Loads image as such. No information present in image is lost.

Returns:

- Numpy array containing the pixel values of the image.

2.1.1

P2-01: LoadDisplayImage.py

```
1 import cv2 as cv
2 filepath="Ch2-images/child_1.jpg"
3 img=cv.imread(filepath)
4 cv.imshow("The child",img)
5 cv.waitKey(0)
```

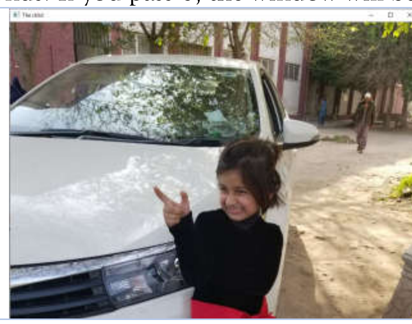
Program Description

We explain the program line by line.

- ***Line* 1**, imports the library files that we want to use in our program. Statement 1 serves two purposes 1-it imports library named "***cv2***" and renames it as "***cv***" to use in our program. This renaming is done in order to avoid typing long library names on every use in the program. So, using this way, we provide a short name to our library.
- ***Line* 2**, declares a string constant in our program. Remember that in python, we do not need to specify the variable type. Here when you assign some value to a variable, the variable determines its type from

the assigned value. So "$\boldsymbol{filepath}$" is our string constant that contains the path of the file. In our case, we have a directory named "$\boldsymbol{Ch2 - Images}$" in the directory containing our script file "$\boldsymbol{LoadDisplayImage.py}$".

- ***Line*3**: we call here OpenCV method "$\boldsymbol{imread(filepath)}$". This method gets file path as string parameter and reads the image. The image may be color or grayscale. We suppose here that you have grayscale image. The read image is then assigned to a variable $\boldsymbol{img}$.
- ***Line* 4**: We call OpenCV method "$\boldsymbol{imshow}$()" to display an image. This method takes two values as parameters. First is the name of window and other is the image data to be displayed.
- ***Line* 5**: OpenCV method "$\boldsymbol{waitKey}$" is used to stop/ hold the output windows. It gets one parameter that is number of milli seconds. If you pass 0, the window will be displayed until user terminates it.



2.2 Displaying image pixel values and other features

We can show the image matrix in python using *print* command. In addition to printing the values of image matrix, you can find lot of other information Following program shows the different features associated with image

P2-02: DisplayFeatures.py

```
1  import cv2 as cv
2  filepath="Ch2-images/child_1.jpg"
   img=cv.imread(filepath,0)
3  print(img)
4  print(img.shape)
5  print(img.dtype)
   print(img.size)
6  print(img[0,:])
7  print(img[:,0])
   print(img[0,0])
```

Output

[[78 75 84 ... 237 249 231] [78 75 84 ... 241 226 201] [79 75 85 ... 235 185 161]	(1280, 960) uint8 1228800 ....

Program Description

As before we explain the program line by line.

- ***Line*1**: importing the library
- ***Line*2**: string constant containing path of image

- $Line3$: Reading the image using $\boldsymbol{OpenCV}$ method $\boldsymbol{imread}()$. “0” value specifies that image should be read as grayscale image.
- $Line4$: Displaying the image matrix using python print method. As most of the image is black so in output section, we see most of the values as zero.
- $Line5$: $\boldsymbol{img.shape}$ displays the shape of image i.e. number of rows, columns and channels as (rows, columns, channels). In our case it is $(\mathbf{1280}, \mathbf{960},)$.
- $Line6$: It prints the data type of $\boldsymbol{img}$ variable
- $Line7$: It prints the size of image. In our case it is $\mathbf{1280 * 960 = 1228800}$
- $Line8$: this statement prints first row values. First row and all column values
- $Line9$: This statement prints values of all rows and first column only
- $Line10$: This line prints the value of first pixel of an image

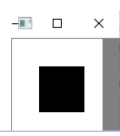
2.3 Declaring a value matrix and showing as an image

Similarly, you can declare a matrix and display it as an image. Following program shows this concept. We do not explain this code as the reader is expected to understand it.

P2-03: ValuesAsImage.py

```
1 import cv2 as cv
2 import numpy as np
3 img=np.zeros([100,100])
4 img[:,:]=255
5 img=img.astype(np.uint8)
6 img[30:80,30:80]=0
7 cv.imshow("Array image",img)
8 cv.waitKey(0)
```

Output



Program Description:

As before we explain the program line by line.

- $Line1, 2$: importing $\boldsymbol{opencv}$ and $\boldsymbol{numpy}$ libraries
- $Line3$: declaring a matrix $\boldsymbol{img}$ of shape $\mathbf{100x100}$ and filling it with zero values
- $Line4$: Assigning the value 255 (white) to all rows and columns of image
- $Line5$: $\boldsymbol{img.astype(np.uint8)}$ This method of $\boldsymbol{numpy}$ changes the data type of image from $\boldsymbol{float64}$ to $\boldsymbol{uint8}$.
- $Line6$: change values in $\boldsymbol{img}$ variable to zero. The new zero value is assigned to rows ranging from 30-80 and columns ranges from 30-80.
- $Line7$: As before this statement shows the matrix as image
- $Line8$: pausing the screen until user closes it.

Pictorial output of matrix is shown above. Where you can see that all image is white except the part of image that is set to zero (black)

2.4 Spatial and Intensity Resolution of an image

P2-04: Resolution.py

```python
1 import numpy as np
2 import cv2 as cv
3 filepath="Ch2-images/child_1.jpg"
4 img=cv.imread(filepath)
5 print("Spatial Resolution::",img.shape)
6 print("Intensity Resolution::",np.min(img),np.max(img))
```

Output

```
Spatial Resolution:: (1280, 960, 3)
Intensity Resolution:: 0 255
```

Program Description

- *Line1, 2*: importing ***opencv*** and ***numpy*** libraries
- *Line3*: declaring file path
- *Line4*: Reading an image using ***OpenCV***
- *Line5*: printing the spatial resolution
- *Line6*: printing the intensity resolution by getting minimum and maximum intensity level in the image

2.5 Finding 4-adjacency in an image

P2-05: P2-05-4Adjacency.py

```python
1 import cv2 as cv
2 filepath="Ch2-images/cameraman.jpg"
3 img=cv.imread(filepath,0)
4 row=int(input("Enter x-coordinate of reference pixel::"))
5 col=int(input("Enter y-coordinate of reference pixel::"))
6 print(img[row-1,col])
7 print(img[row+1,col])
8 print(img[row,col-1])
9 print(img[row,col+1])
```

Output

```
Enter x-coordinate of reference pixel::10
Enter y-coordinate of reference pixel::30
168
168
167
168
```

Program Description

- *Line1*: importing ***opencv*** library
- *Line2*: declaring file path
- *Line3*: Reading an image as gray scale using ***OpenCV***
- *Line4, 5*: Reading x,y coordinate values from user
- *Line6 – 9*: Printing the 4-neighbour of reference pixel

2.6 Finding 8-adjacency in an image

P2-05: P2-06-8Adjacency.py

If you find any error, mistake or have any suggestions for the improvement of this text, please email me at sajidiqbal.pk@gmail.com

```
import cv2 as cv
filepath="Ch2-images/cameraman.jpg"
img=cv.imread(filepath,0)
row=int(input("Enter x-coordinate of reference pixel::"))
col=int(input("Enter y-coordinate of reference pixel::"))
for r in range(row-1,row+2):
    for c in range(col-1,col+2):
        print(r,c,img[r,c])
```

Output

```
Enter x-coordinate of reference pixel::5
Enter y-coordinate of reference pixel::5
4 4 157
4 5 158
4 6 158
5 4 157
5 5 158
5 6 158
6 4 157
6 5 158
6 6 158
```

Program Description

The reader is required to understand this program himself

2.7 Finding m-adjacency in an image

P2-07: P2-07-m-Adjacency.py

```
import numpy as np
img=np.asarray([[0,1,1],
                [0,1,0],
                [0,0,1]])
v=np.asarray([1])
p_c=p_r=1 # position of p pixel
q_c=p_c+1 # column position of q pixel
q_r=p_r-1 # row position of q pixel
def is_in_n4(p_c,p_r,q_c,q_r): # check if coordinates of q in N4(p)?
    if q_c==p_c-1 and q_r==p_r:     return 1 # left neighbor
    elif q_c==p_c+1 and q_r==p_r:   return 1 # right neighbor
    elif q_c==p_c and q_r==p_r-1:   return 1 # top neighbor
    elif q_c==p_c and q_r==p_r+1:   return 1 # bottom neighbor
    else:                           return -1 # otherwise
def is_in_nd(p_c,p_r,q_c,q_r): # check if coordinates of q in ND(p)?
    if q_c==p_c-1 and q_r==p_r-1:   return 1 # left-top neighbor
    elif q_c==p_c+1 and q_r==p_r+1: return 1 # bottom-right neighbor
    elif q_c==p_c+1 and q_r==p_r-1: return 1 # top-right neighbor
    elif q_c==p_c-1 and q_r==p_r+1: return 1 # bottom-left neighbor
    else:                           return -1
```

```python
def get_n4(p_c,p_r): # get N4(p)
    nlist=[]                    # create empty list
    nlist.append([p_r,p_c-1])   # Add left neighbor to list
    nlist.append([p_r,p_c+1])   # Add right neighbor to list
    nlist.append([p_r-1,p_c])   # Add top neighbor to list
    nlist.append([p_r+1,p_c])   # Add bottom neighbor to list
    return nlist                # return N4(p) list
def get_intersection(n4p,n4q): # finding intersection of N4(p) and N4(q)
    intersection=[]             # Creating empty list
    for i in range(len(n4p)):   # iterating all N4(p)
        if n4p[i] in n4q:       # does n4q contains n4p(i) element
            intersection.append(n4p[i]) # if yes, add in list
    return intersection         # return the list
# Now we apply m-adjacency checks
if (is_in_n4(p_c,p_r,q_c,q_r)==1): # condition-1:: q is in N4(p)
    print("m-adjacency")
if (is_in_nd(p_c,p_r,q_c,q_r)==1): # condition-2(A):: q is in ND(p)
    n4p=get_n4(p_c,p_r)            # condition-2(B) get N4(p)
    n4q=get_n4(q_c,q_r)            #                get N4(q)
    intersection=get_intersection(n4p,n4q)#         get N4(p) ∩ N4(q)
# No pixel in intersection should have intensity value from set V
    for i in range(len(intersection)): )# Iterate every pixel
        for j in range(len(v)):  # Check for every level of intensity
            r,c=intersection[i]  # get coordinates of ith common element
            val=v[j]             # get jth intensity value from V
            pixval=img[r,c]      # get intensity value of ith common pixel
            if val==pixval:      # if ith common pixel value is from V
                print("no m-adjacency") # No adjacency exists
                break
```

2.8 Distance Calculation

P2-08: P2-08-DistanceCalculation.py

```python
# The program is easy to understand, hence not explained
import numpy as np
row1=100
col1=100
row2=150
col2=150
def Eucladian_Distance(p_x,p_y,q_x,q_y):
    Eu_dist=np.sqrt(np.square(p_x-q_x)+np.square(p_y-q_y))
    return Eu_dist

```

```
def CityBlock_Distance(p_x,p_y,q_x,q_y):
    City_dist=np.abs(p_x-q_x)+np.abs(p_y-q_y)
    return City_dist

def Chessboard_Distance(p_x,p_y,q_x,q_y):
    a=np.abs(p_x-q_x)
    b=np.abs(p_y-q_y)
    if a==b: return a
    else:
        Chess_dist=np.max(a,b)
    return Chess_dist

print("Eucladian
Distance:",Eucladian_Distance(row1,col1,row2,col2))
print("City Block
Distance:",CityBlock_Distance(row1,col1,row2,col2))
print("Chessboard
Distance:",Chessboard_Distance(row1,col1,row2,col2))
```

Output

Euclidian Distance: 70.71067811865476
City Block Distance: 100
Chessboard Distance: 50

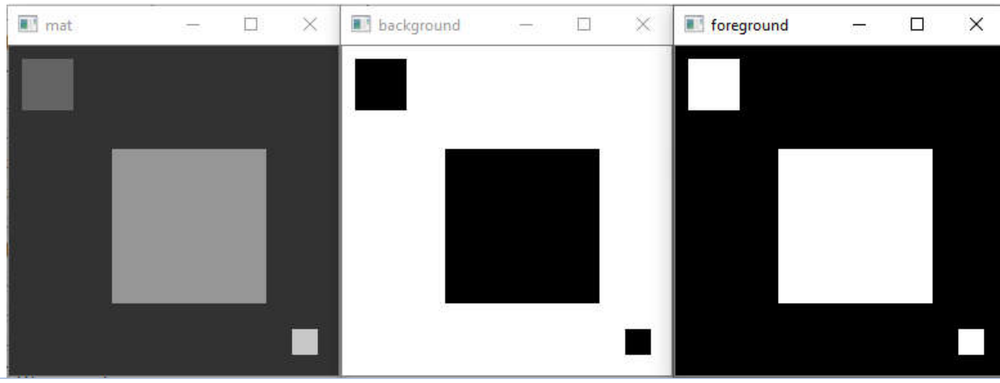
2.9 Foreground background extraction

P2-09: P2-09-ImageForegroundBackground.py

```
import numpy as np
import cv2 as cv
img=np.zeros([256,256],np.uint8)
bg=np.zeros([256,256],np.uint8)
fg=np.zeros([256,256],np.uint8)
img[:,:]=50
img[10:50,10:50]=100
img[80:200,80:200]=150
img[220:240,220:240]=200
idxs=np.where(img==50)
bg[idxs]=255
idxs1=np.where(img==100)
idxs2=np.where(img==150)
idxs3=np.where(img==200)
fg[idxs1]=255
fg[idxs2]=255
fg[idxs3]=255
cv.imshow("mat",img)
```

```
19 cv.imshow("background",bg)
20 cv.imshow("foreground",fg)
21 cv.waitKey(0)
```

Output:



Program Description

- $Line1-2$: library import
- $Line3-5$: declaring three matrices of size 256x256 populated with zero values
- $Line6$: We set pixel values of all image as 50. So that every pixel with value 50 will be part of background
- $Line7-9$: We select here different rectangle areas of matrix and set it with different values i.e. 100, 150, 200. It makes three foreground objects in our image as shown above in first output windows.
- $Line10-11$: At line 10, we get indices of all pixels that have value 50, it means indices of all background pixels. At line 11, in matrix $\boldsymbol{bg}$ that consists of zero values, set all pixels to 255 whose indices are given in $\boldsymbol{idxs}$.
- $Line12-13$: Get indices of pixels that have values 100,150 and 200 in variables $\boldsymbol{idxs1}, \boldsymbol{idxs2}, \boldsymbol{idxs3}$
- $Line15-17$: In matrix $\boldsymbol{fg}$ that consists of all zeros, change the pixels values to 255 of those pixels whose indices are given in $\boldsymbol{idxs1}, \boldsymbol{idxs2}, \boldsymbol{idxs3}$
- $Line18-20$: Show the matrices. All three are shown in output

2.10 Arithmetic Operations

P2-10: P2-10-ArithematicOperations.py

```
import cv2 as cv
img=cv.imread("ch2-images/cameraman.jpg")
img=img[:,:,0]
print("Original Pixel Value at location (10,10):",img[10,10])
img_a=img+10
print("Updated Pixel Value at location (10,10):",img_a[10,10])
img_b=img-10
print("Updated Pixel Value at location (10,10):",img_b[10,10])
img_c=img*10
print("Updated Pixel Value at location (10,10):",img_c[10,10])

```

If you find any error, mistake or have any suggestions for the improvement of this text, please email me at
sajidiqbal.pk@gmail.com

12	`img_d=img/10` `print("Updated Pixel Value at location (10,10):",img_d[10,10])`
Output	

Original Pixel Value at location (10,10): 159
Updated Pixel Value at location (10,10): 169
Updated Pixel Value at location (10,10): 149
Updated Pixel Value at location (10,10): 54
Updated Pixel Value at location (10,10): 15.9

2.11 Image Subtraction

- **np.random.randint(*low*, *high=None*, *size=None*, *dtype='l'*)**

  Return random integers from *low* (inclusive) to *high* (exclusive).

  Parameters
  - **low**: Lowest (signed) integer to be drawn from the distribution
  - **high**: [optional] If provided, one above the largest (signed) integer to be drawn from the distribution
  - **shape**: Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. Default is None, in which case a single value is returned.
  - **Dtype**: [optional] Desired dtype of the result. All dtypes are determined by their name, i.e., ‘int64’, ‘int’, etc, so byteorder is not available and a specific precision may have different C types depending on the platform. The default value is ‘np.int’.

  Returns:
  - Creates a tensor of size specified filled with random values ranging from low-high.

- **np.abs(***x, *args,**kwargs***)**

  Calculate the element wise absolute value of input tensor

  Parameters
  - **x**: Input array (tensor)
  - **high**: [optional] If provided, one above the largest (signed) integer to be drawn from the distribution
  - *args: arguments. Will be explained later when used.
  - ****kwargs**: Key word arguments. Not need to be explained here.

  Returns:
  - Returns the elementwise absolute values of input tensor

- **np.hstack(***tup***)**

  Stack arrays in sequence horizontally (column wise). Following examples shows the horizontal stacking operation.

  $$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, B = \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 7 & 8 & 9 \\ 4 & 5 & 6 & 10 & 11 & 12 \end{bmatrix}$$

  Parameters
  - **tup**: sequence of arrays or tensors. The arrays must have the same shape along all but the second axis, except 1-D arrays which can be any length.

  Returns:
  - The array formed by stacking the given arrays.

P2-11: P2-11-ImageSubtraction.py

```python
import cv2 as cv
import numpy as np
filepath="Ch2-images/child_small.jpg"
img=cv.imread(filepath,0)
r,c=img.shape
# creating a matrix of random variable between values 0-50 and specified shape.
noise=np.random.randint(0,50,[r,c],np.uint8)
img2=img+noise
diff=np.abs(img-img2)
```

```
all_images=np.hstack((img,img2,diff))
cv.imshow("The child",all_images)
cv.waitKey(0)
```

Output

2.12 Image Negative

P2-12: P2-12-ImageSubtraction.py

```
import cv2 as cv
import numpy as np
filepath="Ch2-images/child_small.jpg"
img=cv.imread(filepath,0)
img2=255-img
two_images=np.hstack((img,img2))
cv.imshow("A)Original Image B)Its Negative",two_images)
cv.waitKey(0)
```

Output

2.13 Noise removal using image averaging

P2-13: P2-13-NoiseAveraging.py

```
import numpy as np
import cv2 as cv
filepath="Ch2-images/child_1.jpg"
img=cv.imread(filepath,0)
```

```
print(img.shape)
noisy_imgs=[]
for i in range(10):
    noise=np.random.randint(i,(i+1)*5,[1280,960])
    im_noise=img+noise
    noisy_imgs.append(im_noise)
new_img=np.zeros([1280,960])
for i in range(len(noisy_imgs)):
    new_img+=noisy_imgs[i]
    nimg=noisy_imgs[i].astype(np.uint8)
    cv.imshow("imge",nimg)
    #cv.waitKey(0)
img_f=new_img/10
img_f=img_f.astype(np.uint8)
cv.imshow("Noise Removed",img_f)
cv.waitKey(0)
```

Output: A) Noisy Image B)Noise Removed Image

2.14 Set Operations

- **np.where(*condition*[, *x*, *y*])**

  Return indexes of elements chosen from *x* or *y* depending on *condition*. Following example shows the use:

$$A = \begin{bmatrix} 4 & 5 & 6 \\ 2 & 1 & 2 \end{bmatrix}$$

$$np.where(A == 2) \;\; returns\ two\ arrays$$

$$[1 \quad 1]$$

$$[0 \quad 2]$$

  It shows that first occurrence of 2 is at (1,0) and second is at (1,2)

  Parameters

  - **condition**: It is condition to be checked.
  - **X,Y**: [optional] these are the values from which elements are to be chosen based on condition

  Returns:

  - Two arrays. First contains the row indexes and other contains the column indexs.

P2-14: P2-14-SetOperations.py

```python
import cv2 as cv
import numpy as np
filepath1="Ch2-images/eye_small.jpg"
filepath2="Ch2-images/flower.jpg"
img1=cv.imread(filepath1,0)
img2=cv.imread(filepath2,0)
# Union
img_union=img1
idxs=np.where(img2>img1)
img_union[idxs]=img2[idxs]
cv.imshow("Union",img_union)
# Intersection
img_intersection=img1
idxs=np.where(img2<img1)
img_intersection[idxs]=img2[idxs]
cv.imshow("Intersection",img_intersection)
# Set Complement
img_comp=255-img1
cv.imshow("Complement",img_comp)
# Set Difference
img_diff=img2-img1
cv.imshow("Difference",img_diff)
cv.waitKey(0)
```

2.15 Average filter for smoothing

P2-15: P2-15-SmoothingMeanModeMedianFilter.py

```
import cv2 as cv
import numpy as np
filepath="Ch2-images/child_1.jpg"
#img=cv.imread(filepath,0)
img=np.asarray([[1,1,1,1],[2,2,2,2],[3,3,3,3],[4,4,4,4]])
height,width=img.shape
new_img=np.zeros([height,width])
padded_img=np.zeros([height+2,width+2])
padded_img[1:-1,1:-1]=img
def get8Adj(row,col):
    adj8=[]
    for r in range(row-1,row+2):
        for c in range(col-1,col+2):
            adj8.append([r,c])
    return adj8
def getAverage(img,n8):
    sum=0
    for i in range(len(n8)):
        r,c=n8[i]
        sum+=img[r,c]
    return sum/9

def getMedian(img,n8):
    #implement it yourself
    pass

def getMode(img,n8):
    #implement it yourself
    pass

for i in range(height):
     for j in range(width):
         n8=get8Adj(i+1,j+1)
         new_img[i,j]= getAverage(padded_img,n8)

np.set_printoptions(2)
print(new_img)
```

```
Output:
[[0.67 1.   1.   0.67]
 [1.33 2.   2.   1.33]
 [2.   3.   3.   2.  ]
 [1.56 2.33 2.33 1.56]]
```

2.16 Image Rotation

- **cv.getRotatedMatrix2D(center, angle, scale)**

  The function calculates the following matrix:

$$Rotation\ Matrix = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  However OpenCV calculates the rotation matrix in some modified way

  Parameters
  - **center**: Center of the rotation in the source image.
  - **angle**: Rotation angle in degrees. Positive values mean counter-clockwise rotation (the coordinate origin is assumed to be the top-left corner)
  - **scale**: Isotropic (invariant to rotation) scale factor.

  Returns:
  - rotated matrix with center at "center" parameter.

- **cv.warpAffine(src, M, dsize, dst, flags, borderMode,borderValue)**

  Applies an affine transformation to an image.

  Parameters
  - **src**: input image or tensor
  - **M**: 2x3 transformation matrix
  - **dsize**: size of output image
  - **flags**: combination of interpolation methods and the optional flag WARP_INVERSE_MAP that means that M is the inverse transformation (dst→src).
  - **borderMode**: pixel extrapolation method when borderMode=BORDER_TRANSPARENT, it means that the pixels in the destination image corresponding to the "outliers" in the source image are not modified by the function.
  - **borderValue**: value used in case of a constant border; by default, it is 0.

  Returns:
  - returns the image upon which the affine has been applied

P2-16: P2-16-Rotation.py

```python
import cv2 as cv
import numpy as np
filepath="Ch2-images/child_3.jpg"
img=cv.imread(filepath,0)
h,w=img.shape
center = (h / 2, w / 2)
angle30,angle60,angle90 = 30,60,90
scale = 1.0
M = cv.getRotationMatrix2D(center, angle30, scale)
rotated30 = cv.warpAffine(img, M, (h, w))
M = cv.getRotationMatrix2D(center, angle60, scale)
rotated60 = cv.warpAffine(img, M, (h, w))
M = cv.getRotationMatrix2D(center, angle90, scale)
rotated90 = cv.warpAffine(img, M, (h, w))
```

```
all_images=np.hstack((rotated30,rotated60,rotated90))
cv.imshow("The child",all_images)
cv.waitKey(0)
```

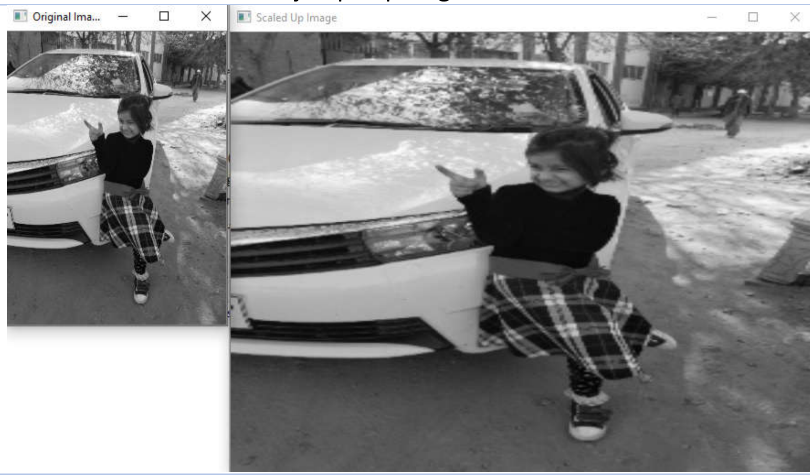
Output:

2.17 Scaling

P2-17: P2-17-Scaling.py

```
import numpy as np
filepath="Ch2-images/child_3.jpg"
img=cv.imread(filepath,0)
h,w=img.shape
# scale up
img_scaledup=cv.resize(img,(2*h,2*w))
#scale down
nh=int(h/2)
nw=int(w/2)
img_scaleddown=cv.resize(img,(nh,nw))
cv.imshow("Original Image",img)
cv.imshow("Scaled Up Image",img_scaledup)
cv.imshow("Scaled Down Image",img_scaleddown)
cv.waitKey(0)
```

Output:

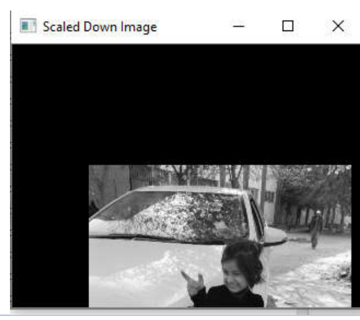


2.18 Translation

P2-18: P2-18-Translation.py

```python
import cv2 as cv
import numpy as np
filepath="Ch2-images/child_3.jpg"
img=cv.imread(filepath,0)
h,w=img.shape
translation_matrix = np.float32([ [1,0,70], [0,1,110] ])
img_translation = cv.warpAffine(img,translation_matrix, (h, w))
cv.imshow("Scaled Down Image",img_translation)
cv.waitKey(0)
```

Output:



The Reader

If you find any error, mistake or have any suggestions for the improvement of this text, please email me at sajidiqbal.pk@gmail.com

2.19 Sheer Transform

- **cv.getAffineTransform(src, dst)**

  Calculates an affine transform from three pairs of the corresponding points.:

  Parameters

  - **src**: Coordinates of triangle vertices in the source image.
  - **dst**: Coordinates of the corresponding triangle vertices in the destination image.

  Returns:

  - returns an image upon which the transform has been applied

P2-19: P2-19-SheerTransform.py

```
import cv2
import numpy as np
import cv2 as cv
filepath="Ch2-images/child_3.jpg"
img=cv.imread(filepath,0)
rows, cols = img.shape
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1]])
dst_points = np.float32([[0,0], [int(0.6*(cols-1)),0],
[int(0.4*(cols-1)),rows-1]])
affine_matrix = cv2.getAffineTransform(src_points, dst_points)
img_output = cv2.warpAffine(img, affine_matrix, (cols,rows))
cv2.imshow('Original Image', img)
cv2.imshow('Affine Transformed Image', img_output)
cv2.waitKey()
```

Output:

2.20 Image Registration

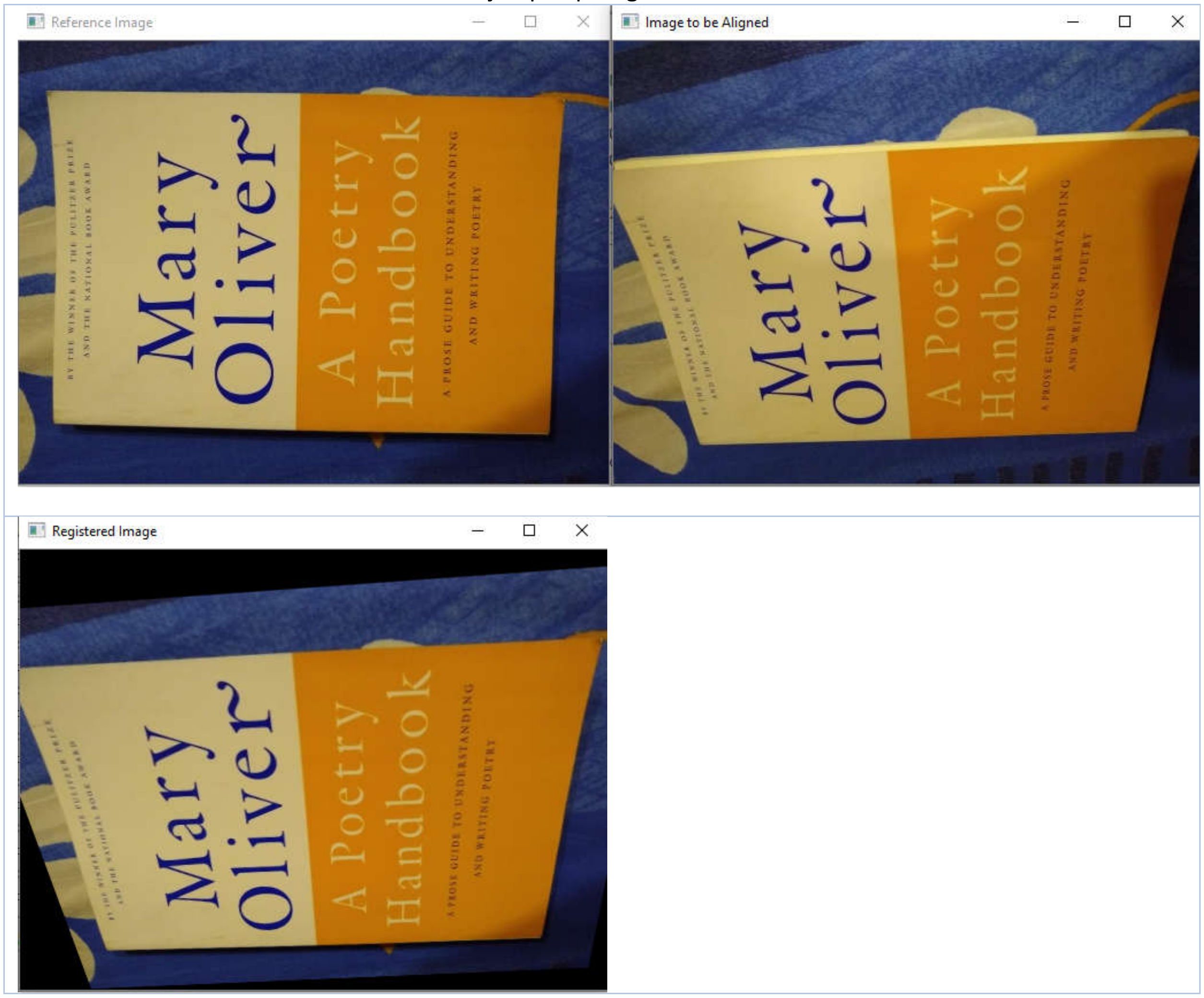
This program is listed in order to give you the idea of image registration. It contains some advanced concepts that are not explained. You will come across these concepts in later chapters.

The Reader

P2-19: P2-20-ImageRegistration.py

```
import cv2 as cv
import numpy as np
img1_color = cv.imread("Ch2-images/book1.jpg") # Image to be aligned.
img2_color = cv.imread("Ch2-images/book2.jpg") # Reference image.
img1 = cv.cvtColor(img1_color, cv.COLOR_BGR2GRAY) # convert to grayscale
img2 = cv.cvtColor(img2_color, cv.COLOR_BGR2GRAY) # convert to grayscale
height, width = img2.shape
orb_detector = cv.ORB_create(5000)   # Create ORB detector with 5000 features.
kp1, d1 = orb_detector.detectAndCompute(img1, None) # Find keypoints and descriptors.
kp2, d2 = orb_detector.detectAndCompute(img2, None) # Find keypoints and descriptors.

# Match features between the two images. # We create a Brute Force matcher
# with Hamming distance as measurement mode.
matcher = cv.BFMatcher(cv.NORM_HAMMING, crossCheck = True)

matches = matcher.match(d1, d2) # Match the two sets of descriptors.

# Sort matches on the basis of their Hamming distance.
matches.sort(key = lambda x: x.distance)


matches = matches[:int(len(matches)*90)] # Take the top 90 % matches forward.
no_of_matches = len(matches)

# Define empty matrices of shape no_of_matches * 2.
p1 = np.zeros((no_of_matches, 2))
p2 = np.zeros((no_of_matches, 2))

for i in range(len(matches)):
   p1[i, :] = kp1[matches[i].queryIdx].pt
   p2[i, :] = kp2[matches[i].trainIdx].pt

homography, mask = cv.findHomography(p1, p2, cv.RANSAC) # Find the homography matrix.

# Use this matrix to transform the colored image wrt the reference image.
transformed_img = cv.warpPerspective(img1_color,homography, (width, height))


cv.imshow("Reference Image",image2_color)
cv.imshow("Image to be Aligned",image1_color)
cv.imshow("Registered Image",transformed_img)
cv.waitKey(0)
```

Output:


Reference Image
Mary Oliver
A Poetry Handbook
Image to be Aligned
Mary Oliver
A Poetry Handbook
Registered Image
Mary Oliver
A Poetry Handbook

2.21 Normal distribution

- `np.random.normal(loc=m, scale=stddev, size=10000)`

Draw random samples from a normal (Gaussian) distribution.

Parameters

- **loc**: Means "center" of the distribution
- **scale**: Standard deviation (spread or "width") of the distribution.
- **Size**: Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. If size is None (default), a single value is returned if loc and scale are both scalars. Otherwise, np.broadcast(loc, scale).size samples are drawn.

Returns:

- Drawn samples from the parameterized normal distribution.

- `plt.hist(loc=m, scale=stddev, size=10000)`

Plot a histogram. Compute and draw the histogram of *x*. The return value is a tuple (*n*, *bins*, *patches*) or ([*n0*, *n1*, ...], *bins*, [*patches0*, *patches1*,...])

Parameters

- **x:** Input values, this takes either a single array or a sequence of arrays which are not required to be of the same length**.**
- **bins**: If an integer is given, bins + 1 bin edges are calculated and returned
- There are number of parameters, we have explained the important ones only.

Returns:

- **N**: The values of the histogram bins.
- **bins**: The edges of the bins. Length nbins + 1 (nbins left edges and right edge of last bin). Always a single array even when multiple data sets are passed in.
- **patches:** Silent list of individual patches used to create the histogram or list of such list if multiple input datasets.

P2-21: P2-21-NormalDistribution.py

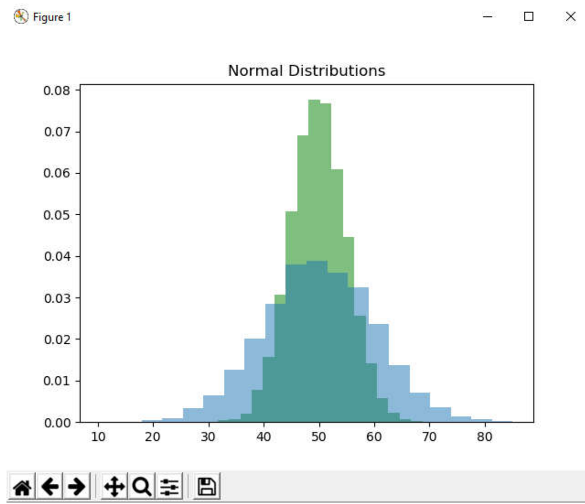
```python
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt
# loc is mean, scale is std dev, size is count of samples
m=50
stddev=5
n = np.random.normal(loc=m, scale=stddev, size=10000)
n2 = np.random.normal(loc=m, scale=stddev+5, size=10000)
n,bins,patches=plt.hist(n, alpha=0.5, bins=20,
density=True,color="green")

n2,bins2,patches2=plt.hist(n2, alpha=0.5, bins=20, density=True)
plt.title("Normal Distributions")

```

The Reader

If you find any error, mistake or have any suggestions for the improvement of this text, please email me at sajidiqbal.pk@gmail.com

```
#plt.plot(range(len(bins)),bins)
plt.show();
```

Output:



2.22 Interpolations

`interpolation=cv.INTER_NEAREST`

This parameter specifies that which interpolation method should be used by the resize algorithm.

P2-22: P2-22-Interpolations.py

```
import cv2 as cv
import numpy as np
filepath="Ch2-images/s.png"
img=cv.imread(filepath,0)
h,w=img.shape
new_size=(2*w,2*h)
img_nn=cv.resize(src=img,dsize=new_size,interpolation=cv.INTER_NEAREST)
img_area=cv.resize(src=img,dsize=new_size,interpolation=cv.INTER_AREA)
img_bits=cv.resize(src=img,dsize=new_size,interpolation=cv.INTER_BITS)
img_bits2=cv.resize(src=img,dsize=new_size,interpolation=cv.INTER_BITS2)
img_cubic=cv.resize(src=img,dsize=new_size,interpolation=cv.INTER_CUBIC)
img_lanczos4=cv.resize(src=img,dsize=new_size,interpolation=cv.INTER_LANCZOS4)
img_linear=cv.resize(src=img,dsize=new_size,interpolation=cv.INTER_LINEAR)
img_linear_exact=cv.resize(src=img,dsize=new_size,interpolation=cv.INTER_LINEA
R_EXACT)
all_imgs=np.hstack((img_nn,img_area,img_bits,img_bits2,img_cubic,img_lanczos4,
img_linear,img_linear_exact))
cv.imshow("Original Image",img)
cv.imshow("NN,Area,Bites,Bites2,Cubic,Lanczos4,Linear,Linear_exact",all_imgs)
cv.waitKey(0)
```

Output:

You can see that some interpolation method are good whereas some do not perform better.