

task 1.1


the emboss effect In the Data folder for this exercise, you will find the intensity image portrait.png Read it into a numpy array arrF and print the shape of this array to determine its number of rows and columns . **The result is:**

```
<class 'numpy.ndarray'>
```

```
(256, 256)
```





In [*]:



```
from PIL import Image
from numpy import asarray
from scipy.ndimage import correlate
import matplotlib.pyplot as plt
import numpy as np
import scipy.ndimage as img



# Load the image and convert into
# numpy array
img = Image.open(r"portrait.png")


# asarray() class is used to convert
# PIL images into NumPy arrays
arrF = asarray(img)

# <class 'numpy.ndarray'>
print(type(arrF))

# shape
print(arrF.shape)

plt.imshow(arrF/255, 'gray')

plt.show()
def embossV1(arrF): 
    M, N = arrF.shape
    arrG = np.zeros((M,N))
    for i in range(1,M-1):
        for j in range(1,N-1):
            arrG[i,j] = 128 + arrF[i+1,j+1] - arrF[i-1,j-1]
            arrG[i,j] = np.maximum(0, np.minimum(255, arrG[i,j]))
    return arrG
z = embossV1(arrF) 
plt.imshow(z)
plt.title('embossv1')
plt.show()

def embossV2(arrF): 
    M, N = arrF.shape
    arrG = np.zeros((M,N))
    arrG[1:M-1,1:N-1] = 128 + arrF[2:,2:] - arrF[:-2,:-2]
    arrG = np.maximum(0, np.minimum(255, arrG))
    return arrG
h = embossV2(arrF) 
plt.imshow(h)
plt.show()

def embossV3(arrF): 
    mask = np.array([[ -1, 0, 0],
                      [ 0, 0, 0],
                      [ 0, 0, +1]])
    arrG = 128 + correlate(arrF,mask,output=None,mode='reflect',cval=0.0,origin=0)
    arrG = np.maximum(0, np.minimum(255, arrG))
    return arrG
l = embossV3(arrF) 
plt.imshow(l)
```

```
plt.show()

def embossV4(arrF):
    arrG = 128 + arrF[2:,2:] - arrF[:-2,:-2]
    arrG[arrG < 0] = 0
    arrG[arrG > 255] = 255
    return arrG
d = embossV4(arrF)
plt.imshow(d)
plt.show()

f, axarr = plt.subplots(2,2)
axarr[0,0].imshow(z)
axarr[0,0].set(title='embossV1')

axarr[0,1].imshow(h)
axarr[0,1].set(title='embossV2')

axarr[1,0].imshow(l)
axarr[1,0].set(title='embossV3')

axarr[1,1].imshow(d)
axarr[1,1].set(title='embossV4')
```

Type *Markdown* and LaTeX: α^2

In []:

In []:

In []:

Does the result you obtain from embossV4 differ from the results produced by the other methods? It should!
Discuss the difference!

All of the four methods of embossing from the end effect point of view giving the same result even when the embossV4 method is easier to implement but I can not see any differences in the outcome image. Maybe we can find out more when we compute the run time later.

Type *Markdown* and LaTeX: α^2

task 1.2

timing the emboss effec

In []:

```
from numpy import asarray
from scipy.ndimage import correlate
import matplotlib.pyplot as plt
import numpy as np
import scipy.ndimage as img
import timeit, functools
import imageio

# Load the image and convert into
# numpy array

def imageread(imagename):
    return imageio.imread(imagename)#.astype(arrtype)

# asarray() class is used to convert
# PIL images into NumPy arrays
img= imageread(r"portrait.png")
arrF = asarray(img)

# <class 'numpy.ndarray'
print(type(arrF))

# shape
print(arrF.shape)
#put 'gray' 'seismic'... or dont put any thing to get it with color
plt.imshow(arrF/255,'spring')
plt.show()

def embossV1(arrF): #first method for embossing using nasted for loop
    M, N = arrF.shape
    arrG = np.zeros((M,N))
    for i in range(1,M-1):
        for j in range(1,N-1):
            arrG[i,j] = 128 + arrF[i+1,j+1] - arrF[i-1,j-1]
            arrG[i,j] = np.maximum(0, np.minimum(255, arrG[i,j]))
    return arrG
z= embossV1(arrF) # call the function
plt.imshow(z)
plt.show()

def embossV2(arrF): #second method for embossing
    M, N = arrF.shape
    arrG = np.zeros((M,N))
    arrG[1:M-1,1:N-1] = 128 + arrF[2:,2:] - arrF[:-2,:-2]
    arrG = np.maximum(0, np.minimum(255, arrG))
    return arrG
h= embossV2(arrF) #call function
plt.imshow(h)
plt.show()

def embossV3(arrF): # third method using correlate
    mask = np.array([[ -1, 0, 0],
                     [ 0, 0, 0],
                     [ 0, 0, +1]])
    arrG = 128 + correlate(arrF,mask,output=None,mode='reflect',cval=0.0,origin=0)
```

```

    arrG = np.maximum(0, np.minimum(255, arrG))
    return arrG
l= embossV3(arrF) # calling the function embossV3
plt.imshow(l)
plt.show()

def embossV4(arrF):
    arrG = 128 + arrF[2:,2:] - arrF[:-2,:-2]
    arrG[arrG< 0] = 0
    arrG[arrG>255] = 255
    return arrG
d= embossV4(arrF)
plt.imshow(d)
plt.show()

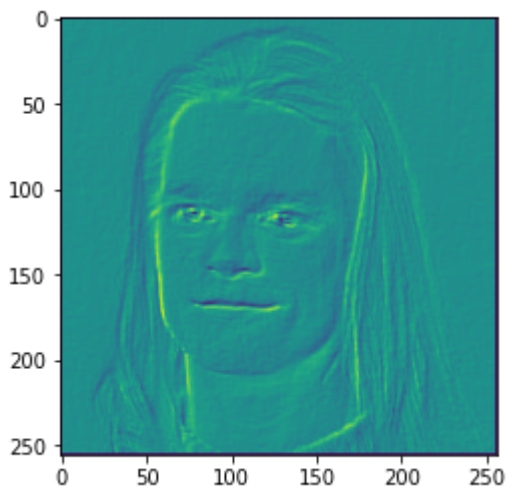
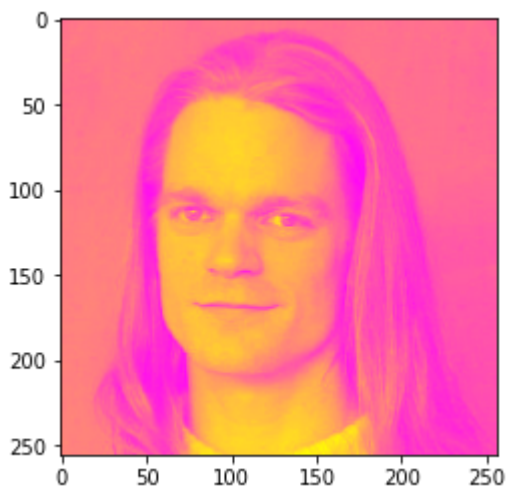
mtds = [embossV1, embossV2, embossV3, embossV4]
nRep = 5
nRun = 100
for mtd in mtds:
    ts = timeit.Timer(functools.partial(mtd, arrF)).repeat(nRep, nRun)
    print (min(ts) / nRun)

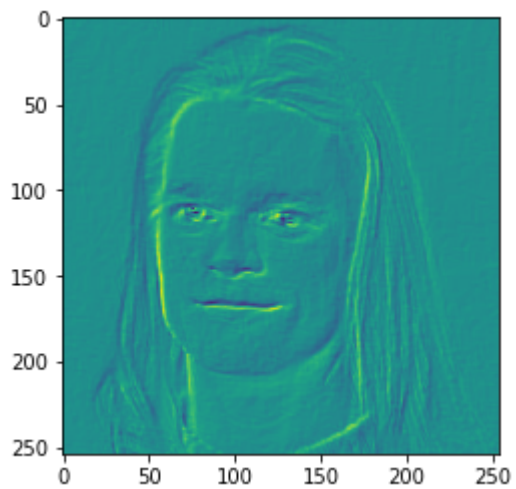
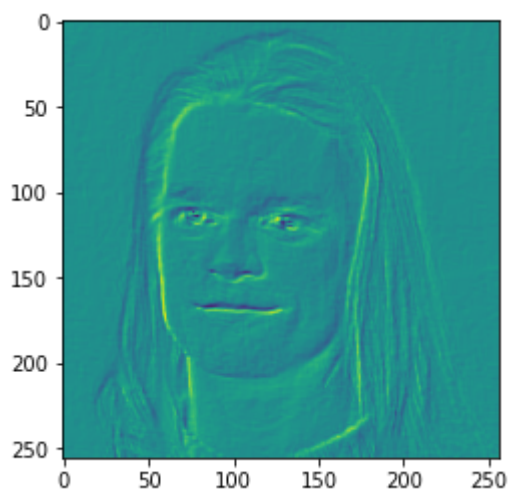
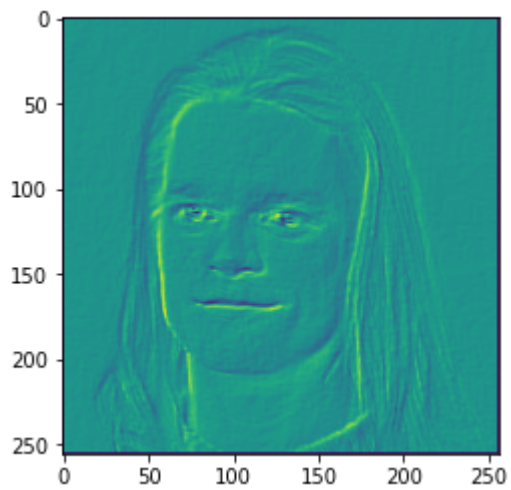
```

```

<class 'numpy.ndarray'>
(256, 256)

```





0.4649694043299999
0.00045484693999998112
0.00113669985999999078
3.34811299995863e-05

task 1.2(a): The run time of the four methods using to emboss images applied to the* portrait.png* is:

- 0.4649694043299999

- 0.0004548469399998112
- 0.0011366998599999078
- 3.34811299995863e-05

task 1.2(b): The run time of the four methods of embossing applied to the intensity image *asterix.png* is:

- 2.64169001785
- 0.0023521718100005273
- 0.005632684060001339
- 0.00026252051000028585

task 1.2(c):

After observing the run time of the embossing methods applied to the two given images *portrait.png* and *asterix.png* we can figure out that the size of the image plays a big role here ,so that

$\text{size(asterix.png)} > \text{size(portrait.png)} \rightarrow \text{run time(asterix.png)} > \text{run time (portrait.png)}$

Not to mention that the run time of the same image differs from a method to another for example, the run time of *embossV1* is the highest because this method uses nested *for_loops* in contrast to the *embossV4* ambossing method which is the quickest one. From all that, it is obvious that getting the best performance in image processing needs some requirements such as:

1. basic coding skills
2. good logical skills
3. Analysing ability in order to analyse a problem you may force and then decide what tools or algo you need to use to solve it.
4. good knowledge in Linear Algebra and calculus.

.....

task 1.3

working with RGB color image

In []:

```
from numpy import asarray
from scipy.ndimage import correlate
import matplotlib.pyplot as plt
import numpy as np
import scipy.ndimage as img
import timeit, functools
import imageio
import matplotlib

# Load the image and convert into
# numpy array
def imageread(image_name, pilmode='RGB'):
    return imageio.imread(image_name, pilmode='RGB').astype(arrtype)

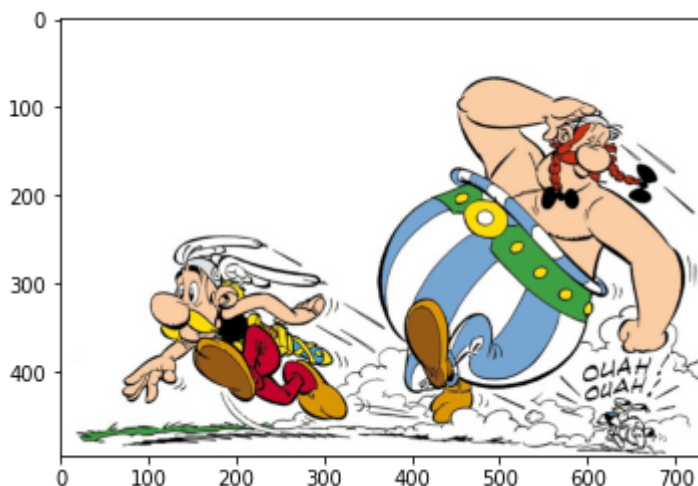
# asarray() class is used to convert
# PIL images into NumPy arrays
img = imageread(r"asterixRGB.png", 'RGB')
arrF = asarray(img)

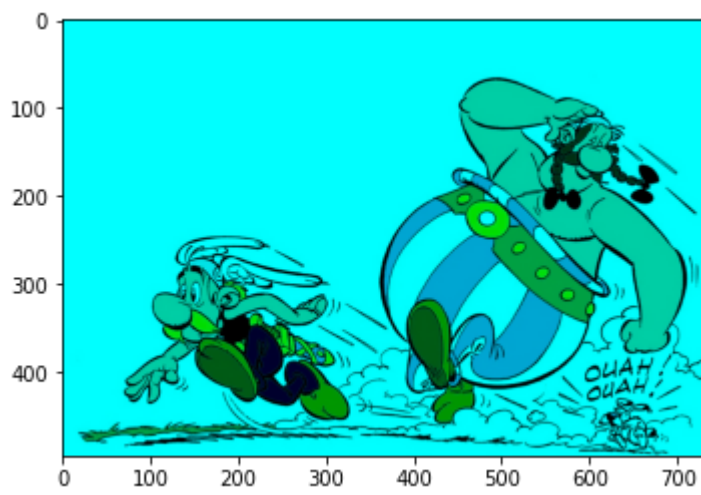
# <class 'numpy.ndarray'
print(type(arrF))

# shape
print(arrF.shape)
plt.imshow(arrF/255)
plt.show()
#copy
arrG = np.copy(arrF)
#shadow
arrG[:, :, 0] = 0
plt.imshow(arrG)
plt.show

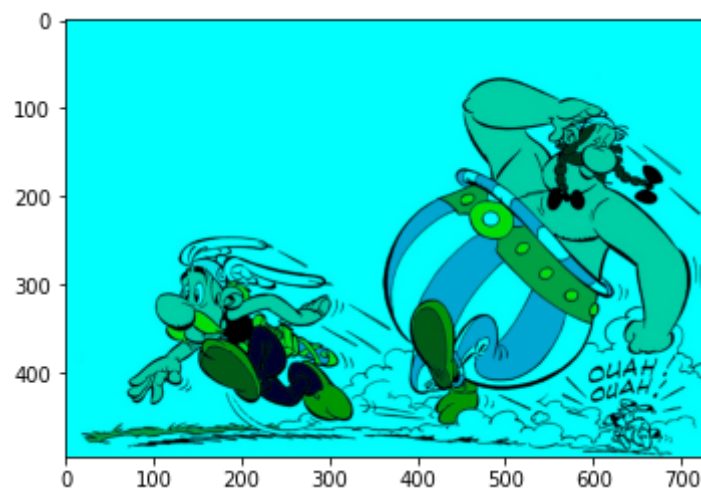
from PIL import Image
im = Image.fromarray(arrG)
im.save("your_file.png")
```

```
<class 'numpy.ndarray'>
(496, 730, 3)
```





My result PNG file:



Type *Markdown* and LaTeX: α^2