Computer Vision Lab 2

SIFT related exercises 1

In this and next lab, you will practice some of the implementation details that one may find relevant to SIFT (but we are not trying to implement SIFT in the labs as there are plenty of resources available in libraries or online). The lab only intends to get you familiarized with programming skills with Python and computer vision. We will try Gaussian dilters, DoG, and keypoint localization for this lab, and dominant orientation and histogram of gradients for the next lab.

1. **Gaussian filters, Difference of Gaussian (DoG), for finding scale-space extrema** Create 5 Gaussian filters with increasing standard deviation starting from 0.5. The filter size you use should be 9x9. You can use OpenCV function

```
cv2.getGaussianKernel()
k = cv2.getGaussianKernel(9, sigma)
g = k * np.transpose(k)
```

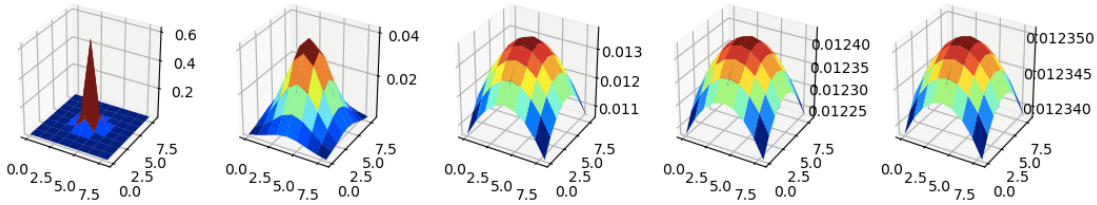
Now H is your g is your gaussian filter and you can visualize it using the following:

```
import from mpl_toolkits.mplot3d import axes3d
X, Y = np.meshgrid(np.linspace(0, 8, num=9), np.linspace(0, 8, num=9))
fig = plt.figure()
plt.title(i)
ax = fig.add_subplot(projection='3d')
surf = ax.plot_surface(X, Y, g, cmap='jet', shade=False)
```

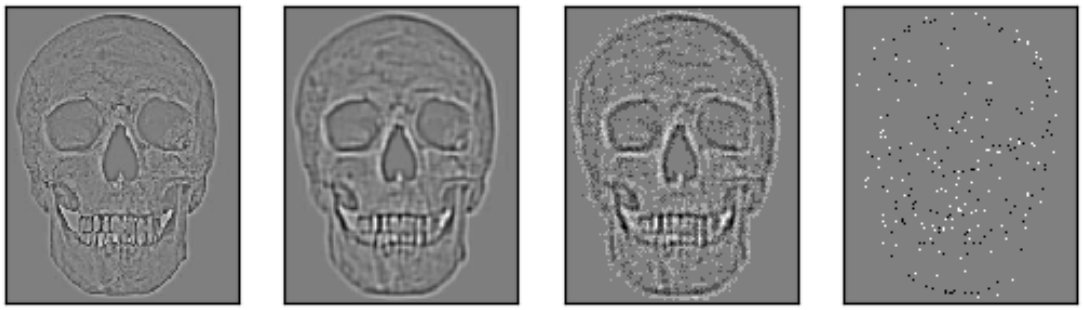
Then filter the input image with cv2.filter2D() with those Gaussian filters created. This can be done by using the following code:

```
imgFiltered = cv2.filter2D(img, -1, g)
```

Then generate DoG by substracting images filtered by Gaussians as described in the lecture notes. Please plot Gaussian filters and DoG for visualization. The figures should look like the following. The Gaussian filters



and DoGs



2. **Key point localization** Next look at examples as how image patches are evaluated with small motion and measure the changes in all directions in terms of the small eigen value of the Hessian matrix. Here you should only consider one patch of size, e.g., 5x5, to demonstrate how this can be done. First, you should compute gradients as we did in last lab. Then for each patch, you should create the Hessian matrix as described in the lecture notes. Then you can use np.linalg.eig() to compute the eigen vectors and eigen values of the Hessian. For example:

```
val, direction = np.linalg.eig(patchHessian)
error = val.min()
directionU = direction[:, np.argsort(val)][0,0]
directionV = direction[:, np.argsort(val)][1,0]
```

Please show the error (smallest eigen value) for all patches.You can also visualize the direction with the smallest change using the following:

```
[X, Y] = np.meshgrid(np.arange(img.shape[1]),
np.arange(img.shape[0]))
U = np.zeros(X.shape)
V = np.zeros(X.shape)
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        U[i,j] = directionU[img.shape[0]-Y[i,j]-1,X[i,j]]
```

```
        V[i,j] = -directionV[img.shape[0]-Y[i,j]-1,X[i,j]]
# for better visualisation, we only display the directions at
every n
# points along both x and y axises.
n = 7
plt.figure()
plt.quiver(X[::n, ::n], Y[::n, ::n], U[::n, ::n], V[::n, ::n])
plt.axis('scaled')
```

Note that here directionU and directionV contains the u and v component of the direction with the smallest change for all the patchs. You can use plt.quiver() function from the matplotlib library for showing arrows for visualizing directions. For better visualization, you should only display arrows at every n pixels. You can use n=7. Also note that in numpy array, the first index is row (so y coordinate) and the positive direction of y coordinate is down (increasing). This is different from normal Math coordinate system (the positive direction of y coordinate is up) that plt.quiver assums. So you need rearrange directionU and directionV for correct visualization as shown in the above code. Your plots should look like the following.