Assignment 2 - Exercise 1

Lukas Eder

In [1]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
image = cv.imread("test_img.png")
dim = (448,336)
image = cv.resize(image, dim)
```

In [3]:
```
cv.imshow("OpenCV Image Reading", image)

cv.waitKey(0)
```

Out[3]:

Assignment 2 -Exercise 2 (Gabor Filter)

Lukas Eder

1 Code

In [1]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
image = cv.imread("test_img.png")
dim = (448,336)
image = cv.resize(image, dim)
```

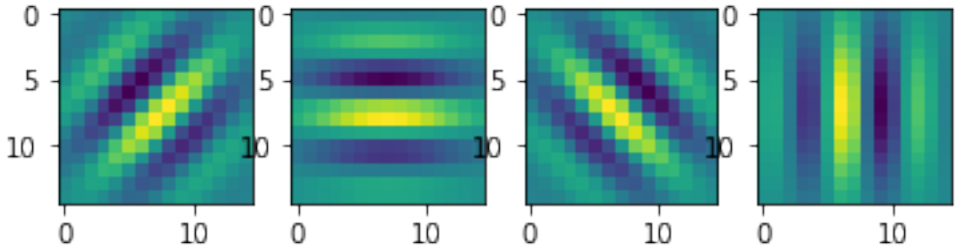
In [3]:
```
k_size = (15,15)
  sigma = 3.0
  theta = 1*np.pi/4
  wavelength = 5.0
  gamma=0.9
  psi = 0.8


  g_kernel1 = cv.getGaborKernel(k_size, sigma, theta, wavelength, gamma,
  psi, ktype=cv.CV_32F)
  g_kernel2 = cv.getGaborKernel(k_size, sigma, 2*theta, wavelength, gamma,
  psi, ktype=cv.CV_32F)
  g_kernel3 = cv.getGaborKernel(k_size, sigma, 3*theta, wavelength, gamma,
  psi, ktype=cv.CV_32F)
  g_kernel4 = cv.getGaborKernel(k_size, sigma, 4*theta, wavelength, gamma,
  psi, ktype=cv.CV_32F)

  filtered_img1 = cv.filter2D(image, cv.CV_8UC3, g_kernel1)
  filtered_img2 = cv.filter2D(image, cv.CV_8UC3, g_kernel2)
  filtered_img3 = cv.filter2D(image, cv.CV_8UC3, g_kernel3)
  filtered_img4 = cv.filter2D(image, cv.CV_8UC3, g_kernel4)


  filter_max1 = np.maximum(image, filtered_img1)
  filter_max2 = np.maximum(filter_max1, filtered_img2)
  filter_max3 = np.maximum(filter_max2, filtered_img3)
  filter_max4 = np.maximum(filter_max3, filtered_img4)
```

In [4]:
```
plt.figure()
```

```
            f, axarr = plt.subplots(1,4)


            axarr[0].imshow(g_kernel1)
            axarr[1].imshow(g_kernel2)
            axarr[2].imshow(g_kernel3)
            axarr[3].imshow(g_kernel4)
```

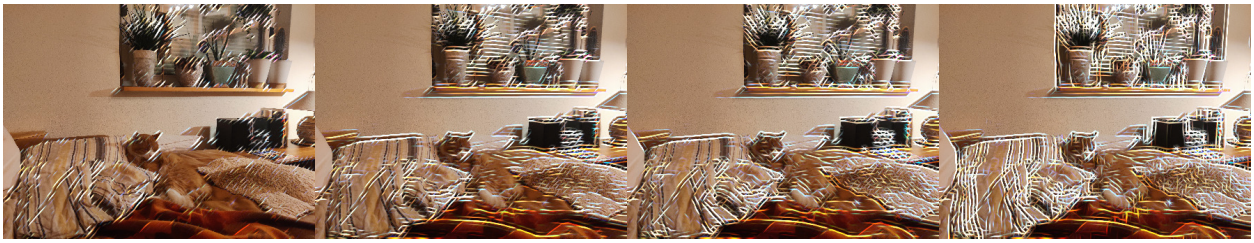
Out[4]: <matplotlib.image.AxesImage at 0x14e238869e8>

<Figure size 432x288 with 0 Axes>



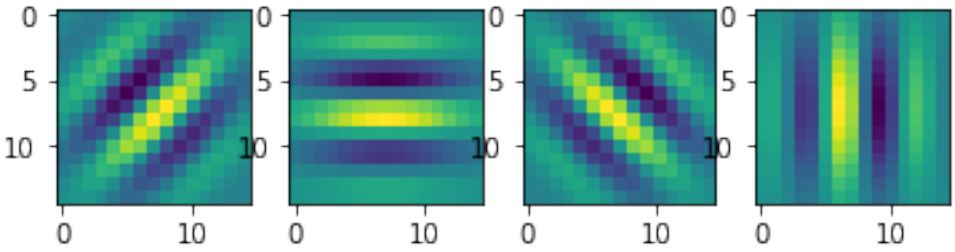

```
In [5]: filters = cv.hconcat([filter_max1, filter_max2, filter_max3, filter_max4])
            cv.imshow('filters', filters)

            cv.waitKey(0)
```

Out[5]: In order Applying filter 1 to 4 on top of each other:

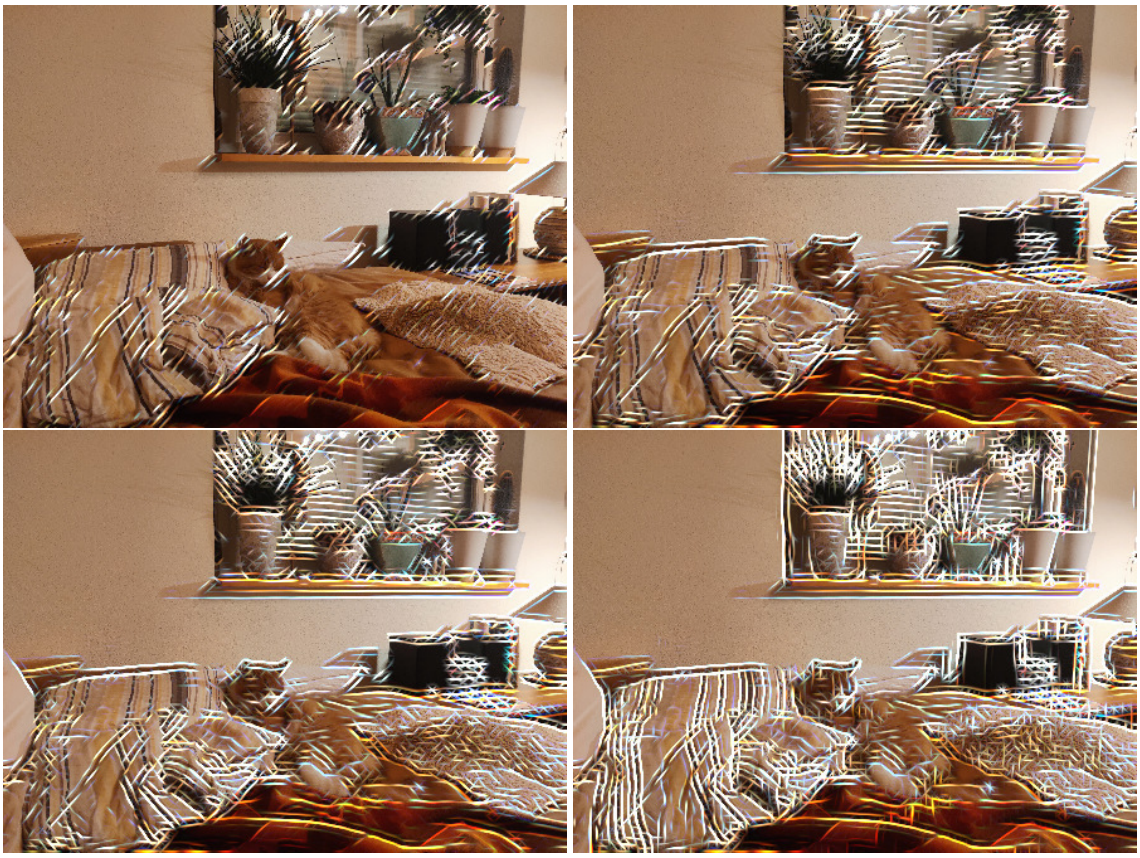


2 different parameter sets for $g(x, y, \lambda, \theta, \psi, \sigma, \gamma)$, where $g_n = g(x, y, \lambda, n * \theta, \psi, \sigma, \gamma)$

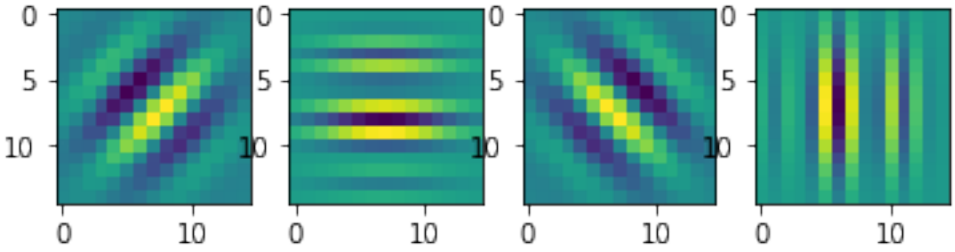
2.1 $g(15, 15, 5.0, \frac{\pi}{4}, 0.8, 3.0, 0.9)$



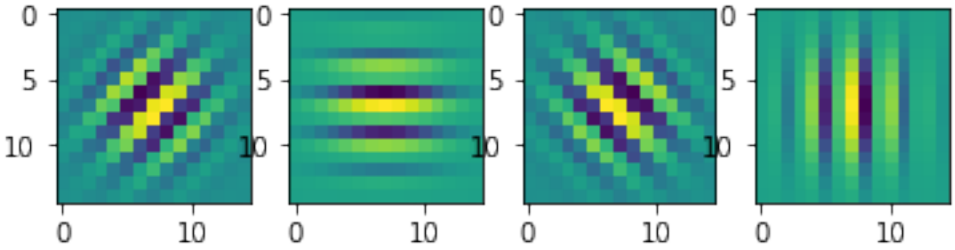
Besides the above shown kernels for g_n for $n = 1, 2, 3, 4$ with values of θ of $\frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi$ in the respective order one can also look at the corresponding image that has been produced by combining and using the maximum pixel value of each image generated by them. In the following images one can observe, which gabor filter (by looking at their orientation) has been added.



2.2 $g(15, 15, \frac{\pi}{5}, \frac{\pi}{4}, 0.8, 3.5, 0.9)$



2.3 $g(15, 15, 3.0, \frac{\pi}{4}, 0.8, 3.0, 0.9)$



Assignment 2 - Exercise 3 Canny Edge detector

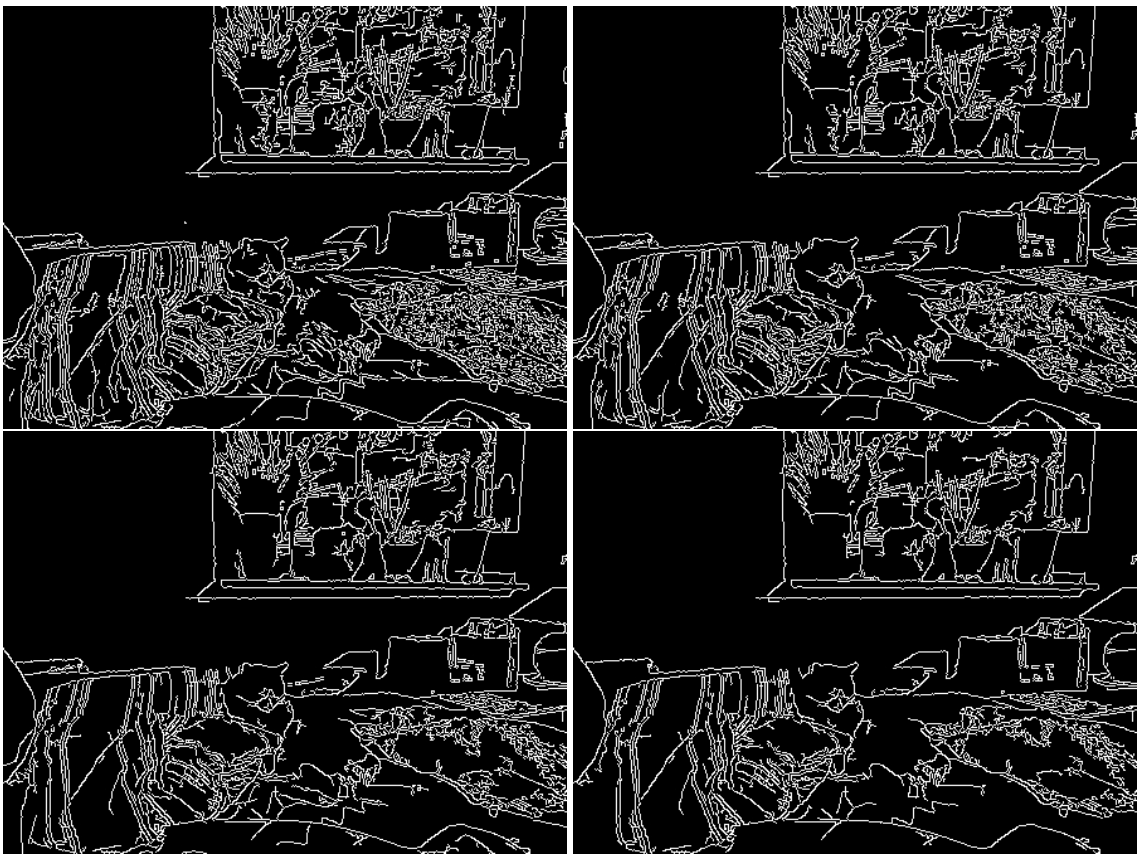
Lukas Eder

In [1]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
img = cv.imread("test_img.png")
dim = (448,336)
img = cv.resize(img, dim)
```

In [3]:
```
edges = cv.hconcat([cv.Canny(img,100,i) for i in [200,300,400,500]])
cv.imshow('filters', edges)

cv.waitKey(0)
plt.show()
```

Out[5]:



If we take a look a these results that show 4 images after edge detection with 4 different increasing values for the upper bound, we see that the results are very similar to the ones of the Gabor filter. However some small details may differ for example the texture of the lamp on the far right is either more detailed with Canny or less detailed than the results using Gabor.

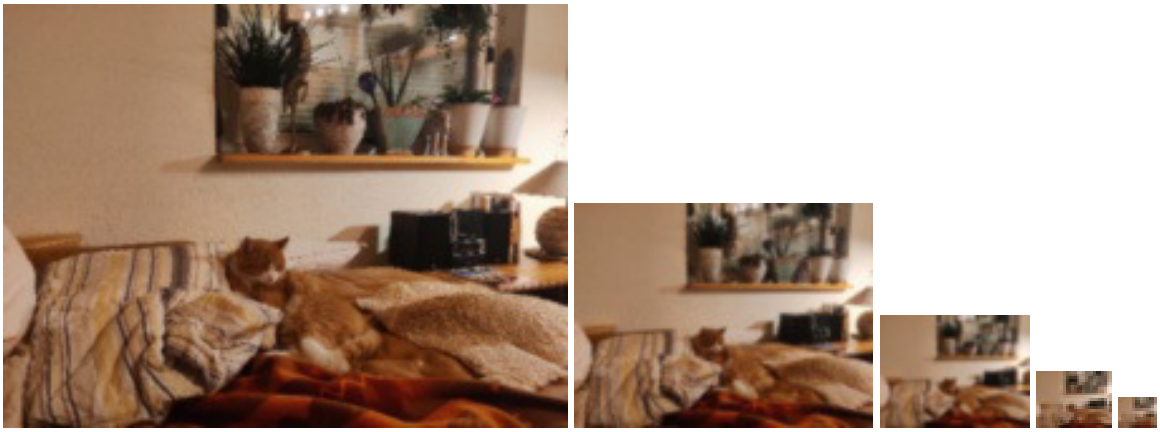
Assignment 2 - Exercise 4 Pyramids

Lukas Eder

1 Code

```
In [1]: import cv2 as cv
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: image = cv.imread("test_img.png")
        dim = (448,336)
        image = cv.resize(image, dim)

In [3]: img = image.copy()
        gaussian_pyr = [img]
        for i in range(6):
            img = cv.pyrDown(gaussian_pyr[i])
            gaussian_pyr.append(img)


In [4]: laplacian_pyr = [gaussian_pyr[-1]]
        for i in range(5,0,-1):
            size = (gaussian_pyr[i - 1].shape[1], gaussian_pyr[i - 1].shape[0])
            gaussian_expanded = cv.pyrUp(gaussian_pyr[i], dstsize=size)
            laplacian = cv.subtract(gaussian_pyr[i-1], gaussian_expanded)
            laplacian_pyr.append(laplacian)

In [28]: def write_to_file(pyr,filetyp,filename):
             for i in range(len(pyr)):
                 name = filename+str(i)+"."+filetype
                 cv.imwrite(filename_gaussian, pyr[i])
             return True

In [29]: write_to_file(gaussian_pyr, "jpg", "gaussian")
         write_to_file(laplacian_pyr, "jpg", "laplacian")

Out[29]: True
```

2 Results



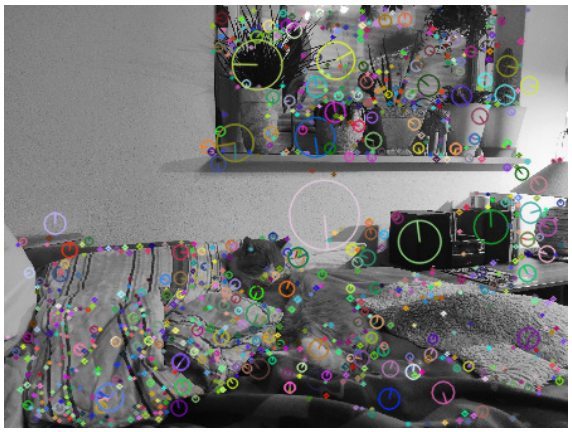
Assignment 2 - Exercise 5 FeatureExtraction

Lukas Eder

1 Code

1.1 SIFT

```
In [1]: import cv2 as cv
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: image = cv.imread("test_img.png")
        dim = (448,336)

        image = cv.resize(image, dim)
        gray= cv.cvtColor(image,cv.COLOR_BGR2GRAY)

In [3]: sift = cv.SIFT_create()
        kp, des = sift.detectAndCompute(gray,None)
        image=cv.drawKeypoints(gray,kp,image,flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

In [4]: cv.imshow("OpenCV Image Reading", image)

        cv.waitKey(0)
```



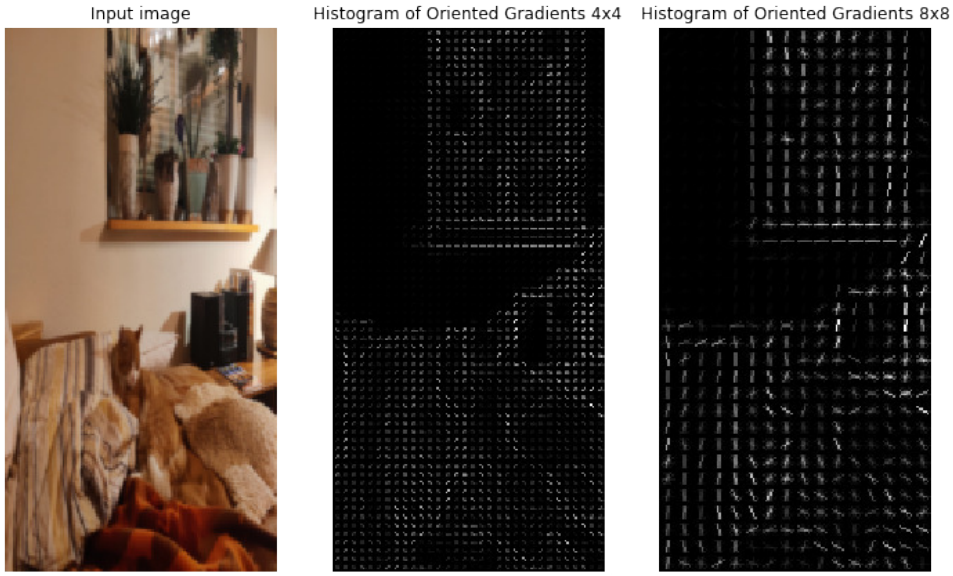
1.2 HOG - using scikit-image

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from skimage.io import imread, imshow
        from skimage.transform import resize
        from skimage.feature import hog
        from skimage import exposure
```

```
In [2]: image = imread('test_img.png')
        # height and width are flipped with scikit-image compared to opencv
        # so (height,width) is used
        # height should be double of width for HOG
        image = resize(image, (256,128))
```

```
In [3]: # set visualize to true to obtain  "hog_image"
        # source: https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.ho
        # cells_per_block= (2,2) = (2*8, 2*8) = (16, 16)
        feature_matrix4x4, hog_image4x4 = hog(image, orientations=9, pixels_per_cell=(4, 4),
                            cells_per_block=(2, 2), visualize=True)

        feature_matrix8x8, hog_image8x8 = hog(image, orientations=9, pixels_per_cell=(8, 8),
                            cells_per_block=(2, 2), visualize=True)
```

```
In [4]: fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 9))
        ax1.imshow(image, cmap=plt.cm.gray)
        ax1.set_title('Input image')
        ax2.imshow(hog_image4x4, cmap=plt.cm.gray)
        ax2.set_title('Histogram of Oriented Gradients 4x4')
        ax3.imshow(hog_image8x8, cmap=plt.cm.gray)
        ax3.set_title('Histogram of Oriented Gradients 8x8')
        plt.show()
```



1.3 Comment on the Results

In the result of HOG on can clearly see how the program detected some objects in the image for example the window frame but also the radio on the shelf in the background. Also both detectors chose similar points to be important as said earlier the frame, radio, flower pot and the pillow on the couch. And both ignore the white wall, with little to no features.

2 Briefly describe in your own words the HOG descriptor

The first requirement that is set to apply to the HOG descriptor is to resize the input image so that it follows a 1:2 width to height ratio. Preferably to make calculations easier the height/width should be a multiple of the latter used cells and blocks, however this is not mandatory, since the calculations is being done by the computer anyway. The next step that has to be done here is to calculate the Gradient in x and y directions for every pixel present in the image. Following that one has to calculate each pixels magnitude and orientation/direction using our previously calculated gradients. To get a pixels magnitude one simply has to perform Pythagoras theorem in the following form $magnitude = \sqrt{G_x^2 + G_y^2}$, where G_d denotes the Gradient in direction d of the given pixel. As well as the pixels orientation, which can be done using the following formula $\theta = arctan(\frac{G_y}{G_x})$. As the name of the method suggests, namely Histogram of Gradients, we have to create such Histograms. First we will do this for every single cell and then in the end for the entire image in a normalized fashion. The way to create Histograms in HOG is to first specify a number of bins, where each bin has equal range. Now we create a Histogram based on the pixels orientation, where we add it's magnitude as value, to be precise we use a weighed method, where depending on how close it's orientation is to the specific bin we add more to it. Since a orientation at most lays between to bins we can use the following formula $magnitude(bin) += \frac{bin-\theta}{\Delta bin} mod \Delta bin)$(where Δbin is the distance between to bins, mod Δbin has to be used here since the orientation θ can be greater or smaller than the bins label) ,to add a pixels value to the histogram. The next step is to do this for every $n * n$ cell, which will each produce a $\#bins * 1$ matrix. Next one should normalize these matrices by using blocks that should be multiples of them. As used in the implementation the block size is twice that of the cells. Meaning on block consist of 4 cells, so a total of $\#bins * 4$ values from the underlying matrices. Which can be normalized by for example using the sum of squares. Now that we have features for all blocks in the image we have to combine them in to one featurematrix.