Jianqiu (Tony) Xu
CSC 391: Computer Vision
Dr. Pauca
Feb. 3rd, 2019

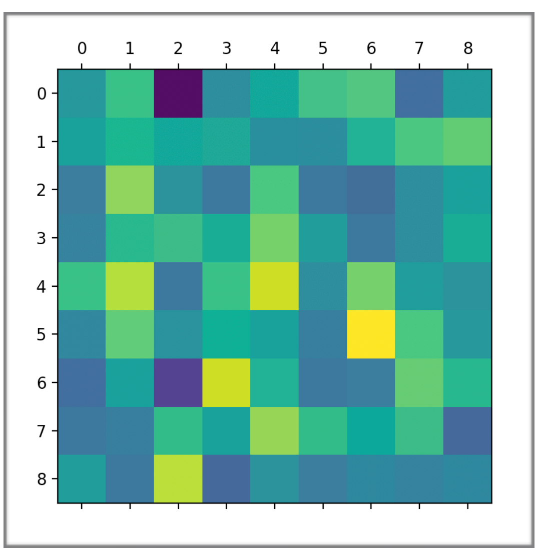
# Project # 1: Spatial and Frequency Filtering

1. Spatial Filtering

```
# =========================== Spatial Filtering ==============================================

# Define a square filter of any size k x k
# Create a 2-D box filter of size k x k and scale so that sum adds up to 1
size = 9
flt = np.random.randn(size, size)
flt = flt/flt.sum()

# Show the generated filter
plt.matshow(flt)
plt.show()

filterImg = np.zeros(img.shape, np.float64)  # array for filtered image
# Apply the filter to each channel
filterImg[:, :, :] = cv2.filter2D(img[:, :, :], -1, flt)
cv2.imshow('Filtered', filterImg.astype(np.uint8))
cv2.imwrite('img-flt.jpg',filterImg.astype(np.uint8))
```

I used the original puppy picture “DSC_9259.JPG” as I implement the self-created spatial filter. By defining a square filter with any size k (defined by size = k), I can apply the filter to each channel of the picture.

This is a 9 x 9 spatial filter randomly generated by my code. After implementing the filter to the original picture of puppy, the result is shown below.




Randomly Generated Filter (9 x 9)

Original Puppy

Filtered Puppy

2. Smoothing, Denoising and Edge Detection

(1) Smoothing and Denoising

```
# ========================== Smoothing, denoising and Edge Detection ========================

# Apply both Gaussian filter and Median filters to original image
# Change the (k, k) to adjust filter size
imgNoisy = cv2.imread('DSC_9259-0.40.JPG')
blur = cv2.GaussianBlur(imgNoisy,(3,3),0)
median = cv2.medianBlur(imgNoisy,3)

cv2.imshow('Original', imgNoisy)
cv2.imshow('Gaussian', blur)
cv2.imshow('Median', median)

# Apply canny to original puppy and noisy pussy
cannyImg = cv2.Canny(img, 50, 200)
cannyNoisy = cv2.Canny(imgNoisy, 200, 350) # If too much noise, increase value
cv2.imshow('Canny1', cannyImg)
cv2.imshow('Canny2', cannyNoisy)

# Apply canny to landscape image
land = cv2.imread('window-00-04.jpg')
cannyLand = cv2.Canny(land, 50, 240)
cv2.imshow('Canny Land', cannyLand)
```

Original Puppy

Noisy Puppy

The noisy image I chose is "DSC_9259-0.40.JPG" and implemented both gaussian filter and median filter with various size value (3 x 3, 9 x 9, 27 x 27).

Gaussian (3 x 3)

Median (3 x 3)

Gaussian (9 x 9)

Median (9 x 9)

Gaussian (27 x 27)

Median (27 x 27)

The outputs from three 3 different sized filters are shown above. The tradeoff between denoise and resolution is clear for both cases.

As gaussian filter is applied to original image with 3 x 3, 9 x 9, and 27 x 27, the effect of denoising gets more obvious while the resolution of the picture become more blur. When median filter is applied, the tradeoff between resolution and denoise is similar to that of gaussian filter. However, gaussian filter performs better in denoising the image while still relatively retaining the resolution.

(2) Edge Detection

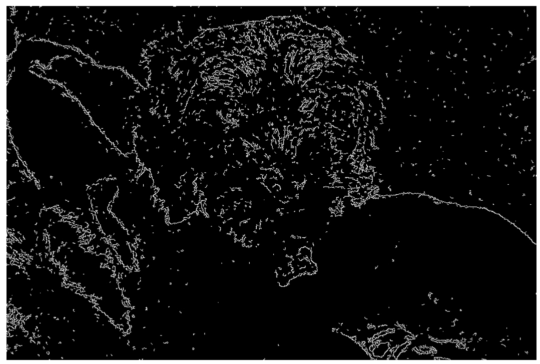
```
# Apply canny to original puppy and noisy pussy
cannyImg = cv2.Canny(img, 50, 200)
cannyNoisy = cv2.Canny(imgNoisy, 200, 350) # If too much noise, increase value
cv2.imshow('Canny1', cannyImg)
cv2.imshow('Canny2', cannyNoisy)
cv2.imwrite('CannyPuppy.jpg', cannyImg)
cv2.imwrite('CannyPuppyNoisy.jpg', cannyNoisy)

# Apply canny to landscape image
land = cv2.imread('window-00-04.jpg')
cannyLand = cv2.Canny(land, 50, 240)
cv2.imshow('Canny Land', cannyLand)
cv2.imwrite('cannyEdge.jpg', cannyLand)
```

In this part of the code, I experimented the Canny edge detectors on the puppy image data, both original and noisy.

Original Puppy

Noisy Puppy

Original Puppy (Canny)

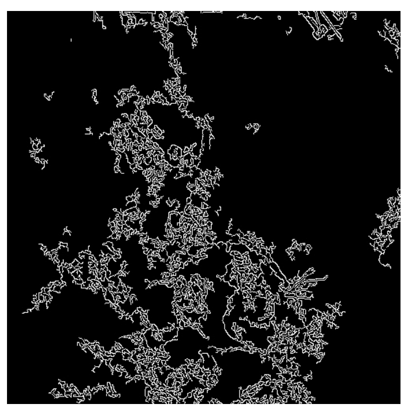


Noisy Puppy (Canny)

We can see that Canny edge detectors has a great result on detecting the edge of the original puppy image when I command "cv2.Canny (img, 50, 200)". However, as Canny filter is implemented on the noisy image with the same command, the result not only contains the edges but also many noise.

Therefore, I adjusted the input value by changing the command to "cv2.Canny (img, 200, 350)" so that I can reduce most of the lower frequency noise and get a clearer edge.

Original Landscape



Landscape (Canny)

I also applied the Canny edge detectors to nature images. The filter can be used to highlight the landscape, but we have to adjust the code to get rid of the noise by setting Canny to a appropriate range

3. Frequency Analysis

(1) Implementation

```
# ========================== Frequency Analysis 4.1 ==========================================

# # Convert the image into grayscale image
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imwrite("grayImage.jpg", grayImg)
# create the x and y coordinate arrays (here we just use pixel indices)
xx, yy = np.mgrid[0:grayImg.shape[0], 0:grayImg.shape[1]]

# Take the 2-D DFT and plot the magnitude of the corresponding Fourier coefficients
F2_grayImg = np.fft.fft2(grayImg.astype(float))

Y = (np.linspace(-int(grayImg.shape[0]/2), int(grayImg.shape[0]/2)-1, grayImg.shape[0]))
X = (np.linspace(-int(grayImg.shape[1]/2), int(grayImg.shape[1]/2)-1, grayImg.shape[1]))
X, Y = np.meshgrid(X, Y)
# Plot the magnitude as image
plt.show()

# Plot the magnitude and the log(magnitude + 1) as images (view from the top)

# Standard plot: range of values makes small differences hard to see
magnitudeImage = np.fft.fftshift(np.abs(F2_grayImg))
magnitudeImage = magnitudeImage / magnitudeImage.max()   # scale to [0, 1]
magnitudeImage = ski.img_as_ubyte(magnitudeImage)
cv2.imshow('Magnitude plot', magnitudeImage)
cv2.imwrite('MagPlot.jpg', magnitudeImage)

# Log(magnitude + 1) plot: shrinks the range so that small differences are visible
logMagnitudeImage = np.fft.fftshift(np.log(np.abs(F2_grayImg)+1))
logMagnitudeImage = logMagnitudeImage / logMagnitudeImage.max()   # scale to [0, 1]
logMagnitudeImage = ski.img_as_ubyte(logMagnitudeImage)
cv2.imshow('Log Magnitude plot', logMagnitudeImage)
cv2.imwrite('LogMagPlot.jpg', logMagnitudeImage)
```

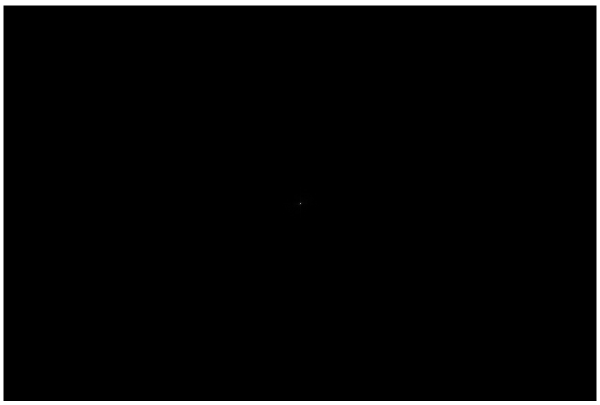
In this part of the code, I first convert “Img” into a grayscale image “grayImg” before processing.

Grayscale Image

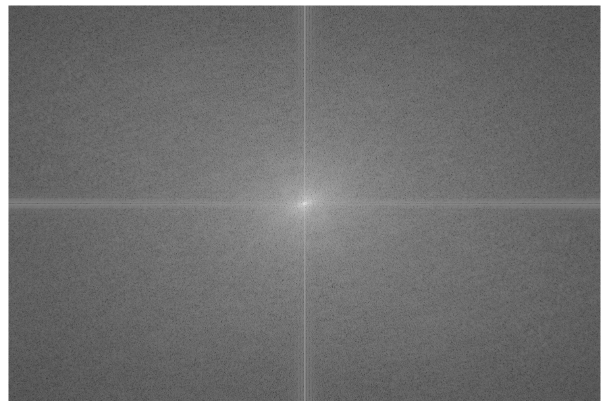
Then, I applied a 2-D Fourier Transformation to visualize the magnitude of the Fourier coefficients of the gray image. The 2-D image of magnitude is shown below.

The low frequencies is in the center of the plot, which shows that low frequencies has a great proportion in the overall frequencies of the image.

I also applied the log of the magnitude+1 to better observe the contribution of mid and high frequencies to your image.



Magnitude



log(Magnitude + 1)

(2) Frequency Analysis

4. Frequency Filtering