Tien Li Shen

30930512

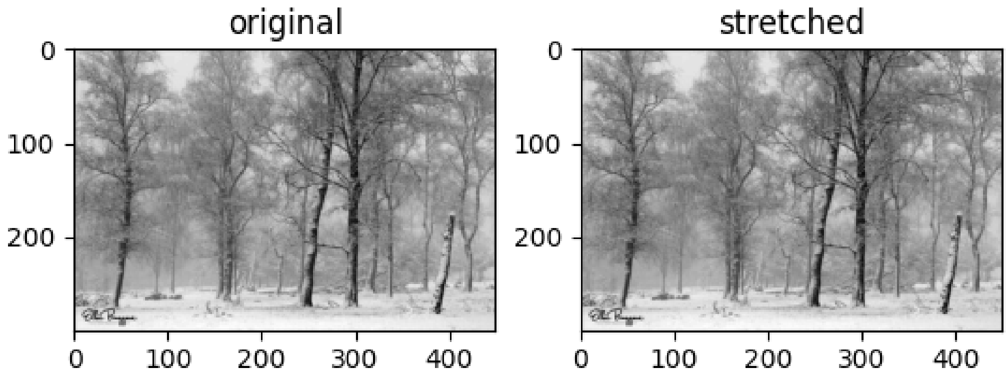
March 1, 2021

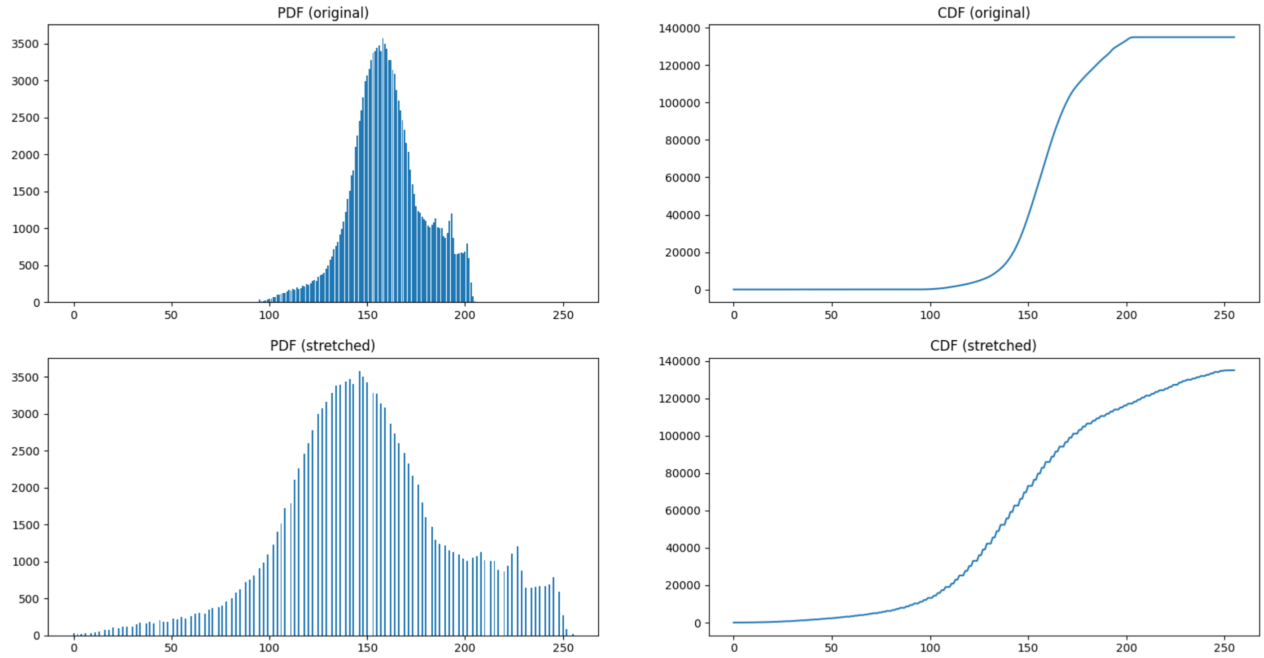
CS 370

Mini Project 3

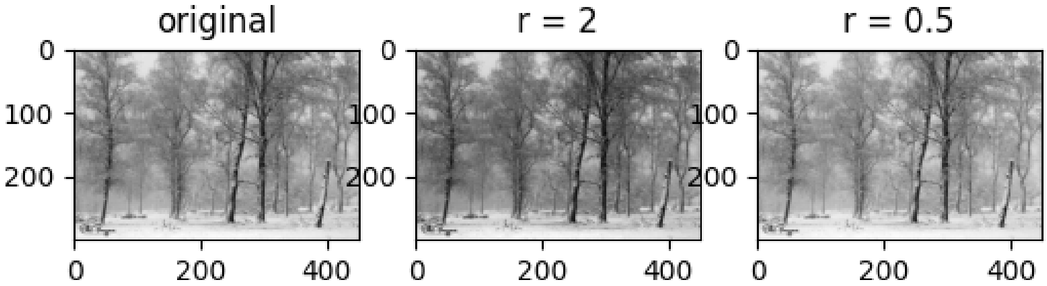
1.

Contrast stretching outputs (images, pdf, and cdf):



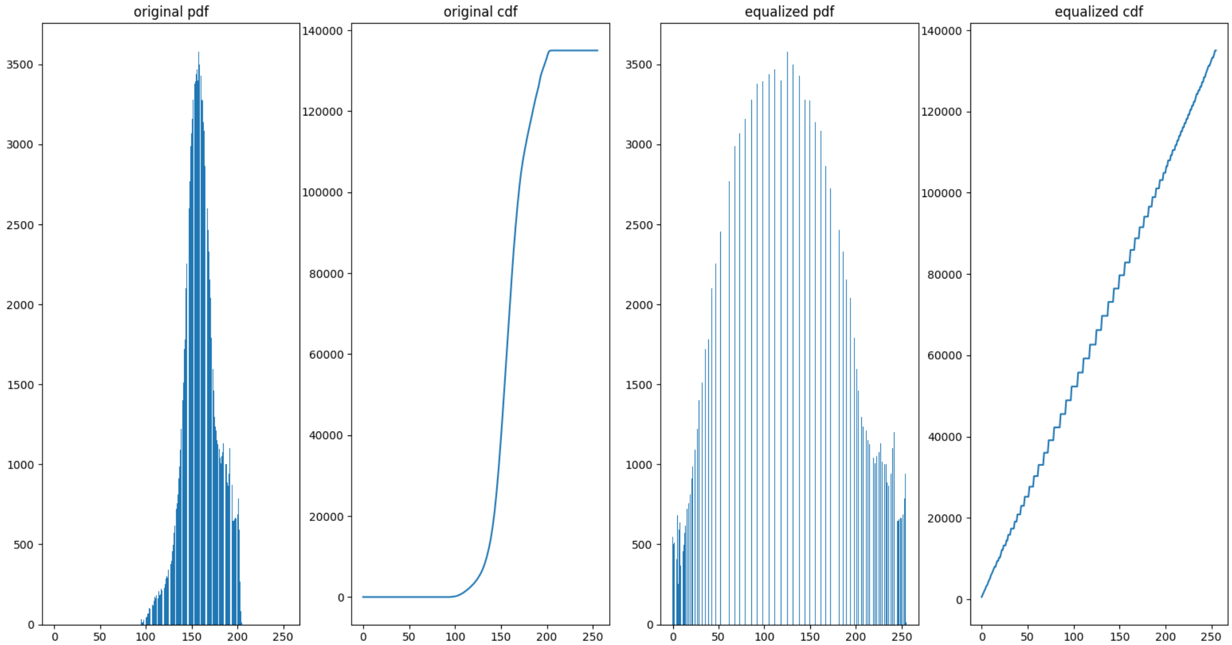


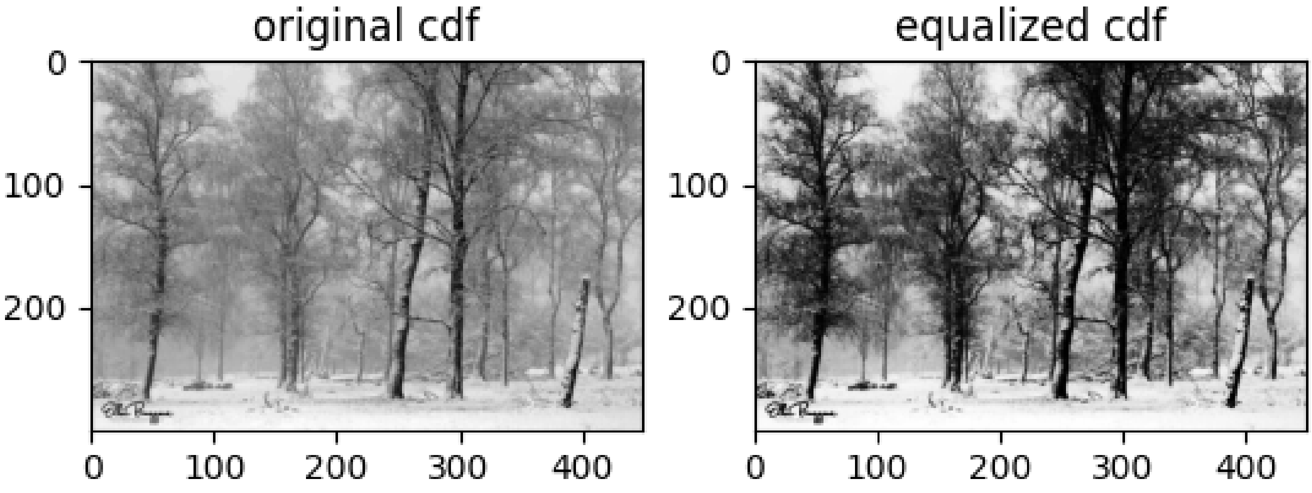
Gamma correction output:



Gamma correction can turn the image brighter or darker. Higher the r value, darker the image, vice versa.

Histogram equalization outputs (pdf, cdf and images):





```
import utils
import numpy as np
import matplotlib.pyplot as plt

def histogram(im):
    """
    input: image with only 1 channel

    return: [pdf, cdf]
    """
    # count the occurance of pixel values and store them in i_arr
    i_arr = np.zeros(256)
    for row in im:
        for pix in row:
            i_arr[pix.astype(int)] += 1

    # optionally convert the PDF to the decimal probability scale
    pix_sum = sum(i_arr)
    pdf = i_arr #/pix_sum

    # declare and compute the cdf
    cdf = np.zeros(256)
    cdf[0] = pdf[0]
    count = 0
    for i in range(len(pdf)): # sum the cdf with a for loop
        count += pdf[i]
        cdf[i] = count


    return [pdf, cdf]

def contrast_stretch(im):
    # declare variables needed
    im_max = np.amax(im)
    im_min = np.amin(im)
    im_diff = im_max - im_min
    im_stretched = np.zeros(im.shape) # placeholder for converted image
    for i in range(im.shape[0]): # loop over rows
        for j in range(im.shape[1]): # loop over columns/each pixels
            im_stretched[i][j] = ((im[i][j]-np.amin(im))/(im_max - im_min) *
255).astype(np.uint8) # apply streching equation to each pixel

    return im_stretched

def compare_stretch(im):
    """
    function to use other functions and get desired results in the desired format for the
assignmnet submission
    """
    im_stretched = contrast_stretch(im)
    pdf, cdf = histogram(im)
    #
    x = [i for i in range(256)]
    plt.subplot(2, 2, 1)
    plt.title("PDF (original)")
    plt.bar(x, pdf)
    plt.subplot(2, 2, 2)
    plt.title("CDF (original)")
    plt.plot(cdf)

    pdf, cdf = histogram(im_stretched)
    #
    x = [i for i in range(256)]
    plt.figure(1)
    plt.subplot(2, 2, 3)
    plt.title("PDF (stretched)")
    plt.bar(x, pdf)
    plt.subplot(2, 2, 4)
    plt.title("CDF (stretched)")
    plt.plot(cdf)

    plt.figure(2)
    plt.subplot(1, 2, 1)
    plt.imshow(im, cmap="gray")
    plt.title("original")
    plt.subplot(1, 2, 2)
    plt.imshow(im_stretched, cmap="gray")
    plt.title("stretched")
    plt.show()

def gamma_correction(im, r):
    """
    apply gamma correction with the provided equation in the assignment PDF
    """
    im_arr = np.array(im)
    im_arr = (im_arr / 255)**r * 255

    return im_arr
def compare_gamma(im):
    """
    function to use other functions and get desired results in the desired format for the
assignmnet submission
    """
    gamma_corrected = gamma_correction(im, 2)
    plt.subplot(1, 3, 1)
    plt.imshow(im, cmap="gray")
    plt.title("original")
    plt.subplot(1, 3, 2)
    plt.imshow(gamma_corrected, cmap="gray")
    plt.title("r = 2")
    plt.subplot(1, 3, 3)
    gamma_corrected = gamma_correction(im, 0.5)
    plt.imshow(gamma_corrected, cmap="gray")
    plt.title("r = 0.5")
    plt.show()

def hist_equalization(cdf, im):
    """
    input: cdf, im

    output: equalize image

    equalize the image by applying cdf equalization to the image, the image can then be
used to obtain a equalized cdf
    """
    N = im.flatten().shape[0]
    print(N)
    print(im.shape)
    count = 1
    cdf_min = 0
    im_eq = np.zeros(im.shape)
    # use a while loop to find the smallest non-zero cdf value (the first value that is
non-zero)
    while(cdf[count-1] <= 0):
        cdf_min = cdf[count]
        count+=1

    # use nested for loop to apply cdf equalization to each pixel
    for i in range(im.shape[0]):
        for j in range(im.shape[1]):
            im_eq[i][j] = ((cdf[im[i][j]] - cdf_min) / (N - cdf_min) * 255).astype("int")

    # cdf_eq = ((cdf - cdf_min) / (N - cdf_min) * 255).astype(int)
    return im_eq

def compare_hist_equalization(im):
    """
    function to use other functions and get desired results in the desired format for the
assignmnet submission
    """
    pdf, cdf = histogram(im)
    im_eq = hist_equalization(cdf, im)
    pdf,  cdf_eq = histogram(im_eq)

    plt.subplot(1, 2, 1)
    plt.plot(cdf)
    plt.title("original cdf")
    plt.subplot(1, 2, 2)
    plt.plot(cdf_eq)
    plt.title("equalized cdf")
    plt.figure(2)
    plt.subplot(1, 2, 1)
    plt.imshow(im, cmap="gray")
    plt.title("original cdf")
    plt.subplot(1, 2, 2)
    plt.imshow(im_eq, cmap="gray")
    plt.title("equalized cdf")
    plt.show()

if __name__ == '__main__':
    """
    Call the functions to generate output plots and images:

        compare_stretch(im)
        compare_gamma(im)
        compare_hist_equalization(im)

    """

    im = plt.imread('../data/forest.png')
    im *= 255
    im = im.astype(np.uint8)
    compare_stretch(im)
```

2.

Gaussian filter output error for sigmas [0, 1, 2, 3]:

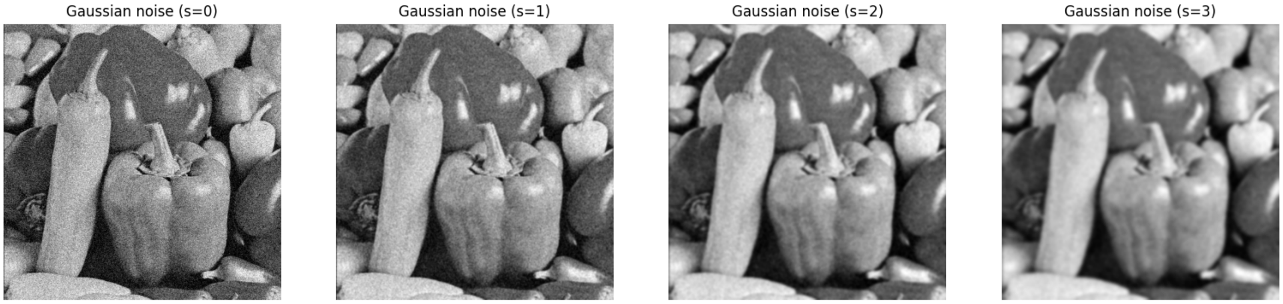
[3450.26391602 458.92755127 532.10723877 775.02160645] (Gaussian noise)

[7339.91308594 882.73419189 747.99725342 963.09112549] (Salt and pepper noise)

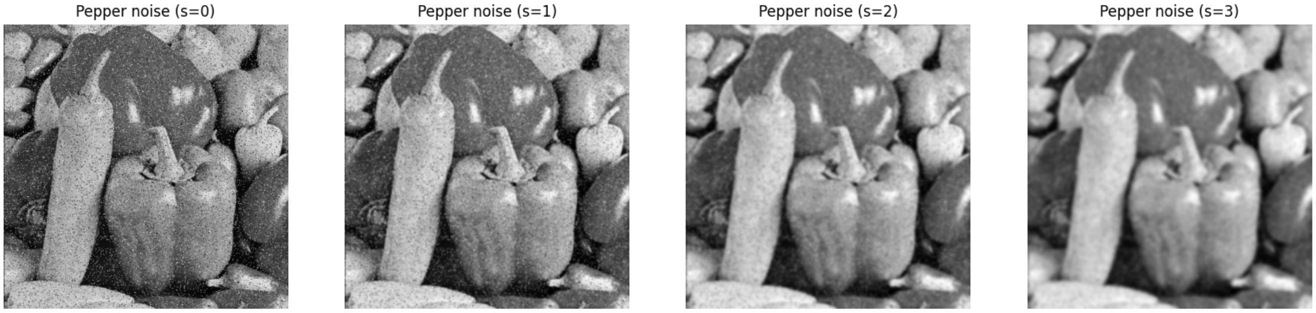
Sigma of 1 minimizes the error for the Gaussian noise image with the lowest error of 458.927 and sigma of 2 minimizes the error in the salt and pepper noise image with the lowest error of 882.734.

Image output from the Gaussian filter:

Gaussian noise



Salt and pepper noise



Pepper noise (s=0)

Pepper noise (s=1)

Pepper noise (s=2)

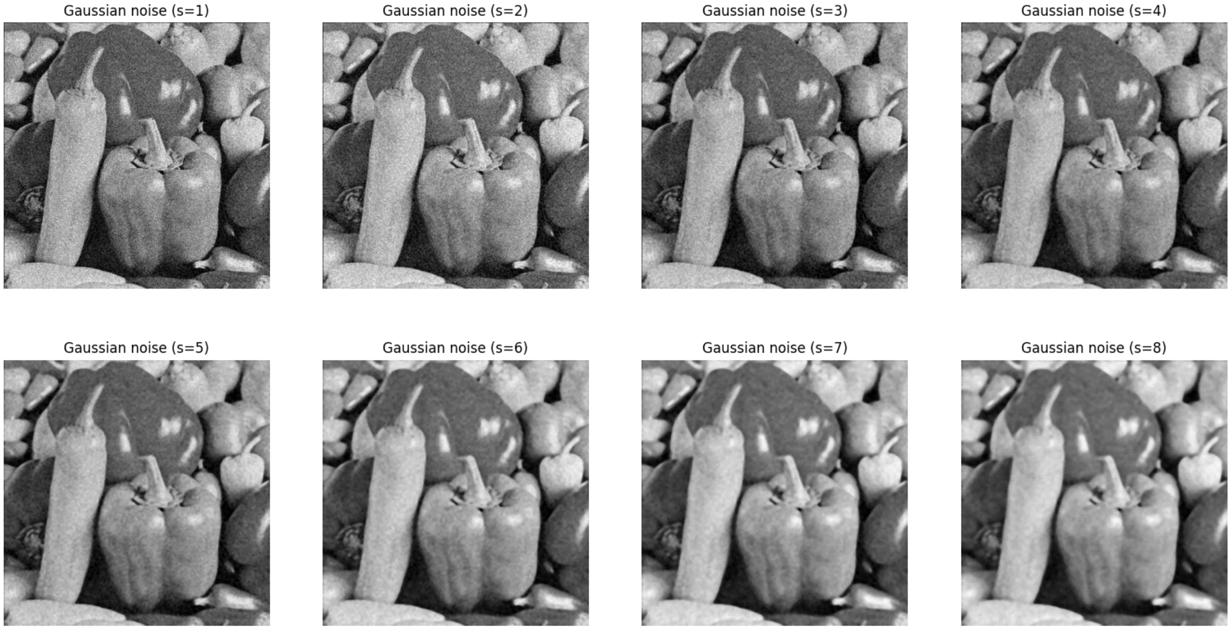
Pepper noise (s=3)

Median filter output error for sigmas [1, 2, 3, 4, 5, 6, 7, 8]

```
[3450.26391602 1875.50195312  728.10736084  751.34545898  548.86029053
  637.41595459  568.28930664  670.59906006]
[7339.91308594 1428.77429199  137.73141479  458.86508179  323.01040649
  483.16812134  405.97351074  548.93188477]
```

Image output from the median filter:

Gaussian noise:



Gaussian noise (s=1)

Gaussian noise (s=2)

Gaussian noise (s=3)

Gaussian noise (s=4)

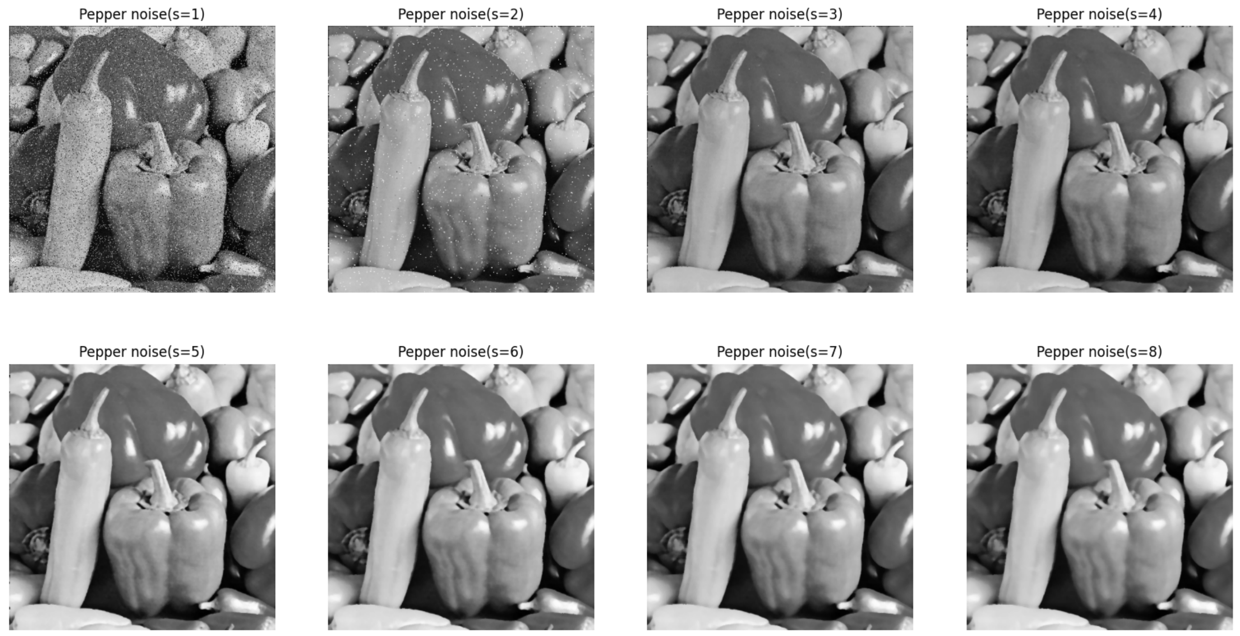
Gaussian noise (s=5)

Gaussian noise (s=6)

Gaussian noise (s=7)

Gaussian noise (s=8)

Salt and pepper noise:



Size of 5 minimizes error for the Gaussian noise image with the lowest error of 548.860 and size of 5 minimizes error for the salt and pepper noise image with the lowest error of 323.010.

Gaussian filter performed better on the Gaussian noise image with the lowest error of 458.927 compared to 548.860 of the median filter.

and median filter performed much better on the salt and pepper noise image with the lowest error of 323.010 compared to 882.734 of the Gaussian filter.

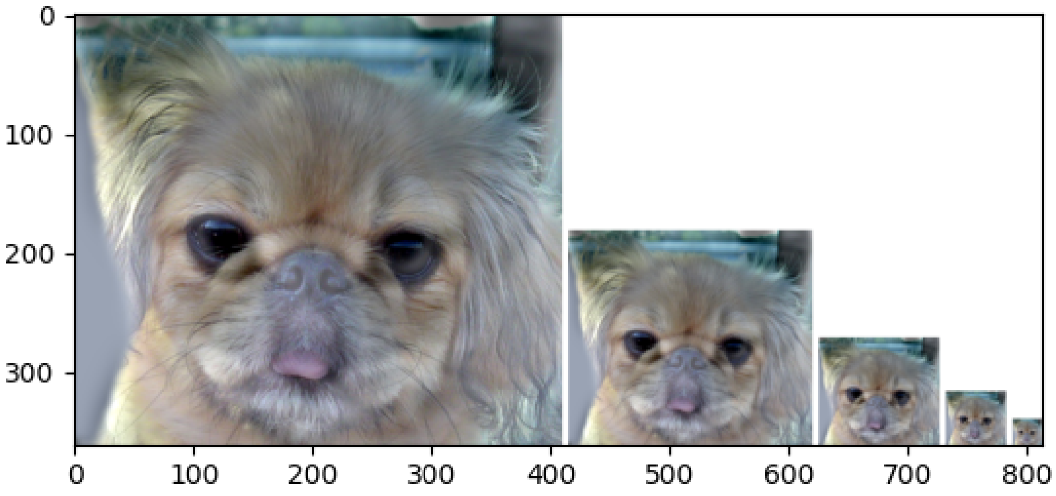
```
import utils
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage

def load_image():
   im1 = plt.imread('../data/peppers.png')
   im2 = plt.imread('../data/peppers_g.png')
   im3 = plt.imread('../data/peppers_sp.png')

   im_123 = np.array([im1, im2, im3])

   return im_123

def gaussian_filter(im_123, sigma):
   plt.figure(1)
```

```
    count = 0

    # apply gaussian filter to Gaussian noise image
    error_g = np.zeros(len(sigma))
    for s in sigma: # use for loop to use gaussian filter on each sigma tested
        im_gau = ndimage.gaussian_filter(im_123[1], sigma = s) # use scipy gaussian filter
on image
        error_g[count] = ((im_123[0] - im_gau) ** 2).sum()  # calculate the error

        # configure the plot
        plt.subplot(1, len(sigma), count+1)
        plt.title("Gaussian noise (s={})".format(s))
        plt.imshow(im_gau, cmap = "gray")
        plt.axis('off')
        count+=1

    # apply gaussian filter to pepper noise image
    plt.figure(2)
    count = 0
    error_sp = np.zeros(len(sigma))

    for s in sigma: # use for loop to use gaussian filter on each sigma tested
        im_gau = ndimage.gaussian_filter(im_123[2], sigma=s) # use scipy gaussian filter on
image
        error_sp[count] = ((im_123[0] - im_gau) ** 2).sum() # calculate the error

        # configure the plot
        plt.subplot(1, len(sigma), count+1)
        plt.title("Pepper noise (s={})".format(s))
        plt.imshow(im_gau, cmap="gray")
        plt.axis('off')
        count += 1
    print(sigma)
    print(error_g)
    print(error_sp)
    plt.show()

def median_filter(im_123, sigma):
    plt.figure(1)
    count = 0

    # apply gaussian filter to Gaussian noise image
    error_g = np.zeros(len(sigma))
    for s in sigma: # use for loop to use median filter on each size tested
        im_gau = ndimage.median_filter(im_123[1], size=s)  # use scipy median filter on
image
        error_g[count] = ((im_123[0] - im_gau) ** 2).sum()  # calculate the error

        # configure the plot
        plt.subplot(2, int(len(sigma) / 2), count + 1)
        plt.title("Gaussian noise (s={})".format(s))
        plt.imshow(im_gau, cmap="gray")
        plt.axis('off')
        count += 1

    # apply gaussian filter to pepper noise image
    plt.figure(2)
```

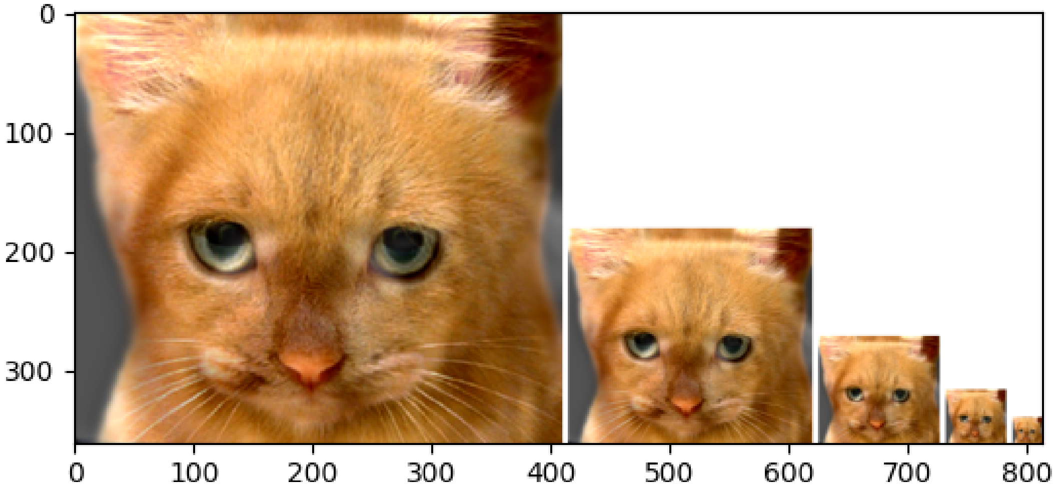
```
   count = 0
   error_sp = np.zeros(len(sigma))
   for s in sigma: # use for loop to use median filter on each size tested
       im_gau = ndimage.median_filter(im_123[2], size=s) # use scipy median filter on
image
       error_sp[count] = ((im_123[0] - im_gau) ** 2).sum()  # calculate the error

       # configure the plot
       plt.subplot(2, int(len(sigma)/2), count + 1)
       plt.title("Pepper noise (s={})".format(s))
       plt.imshow(im_gau, cmap="gray")
       plt.axis('off')
       count += 1
   print(sigma)
   print(error_g)
   print(error_sp)
   plt.show()

if __name__ == "__main__":
   im_123 = load_image()
   sigma = [i for i in range(1,9)]
   sigma = [i for i in range(4)]
   # median_filter(im_123, sigma)
   gaussian_filter(im_123, sigma)
```

3.

Hybrid image of dog and cat with sigma of 4 and 10

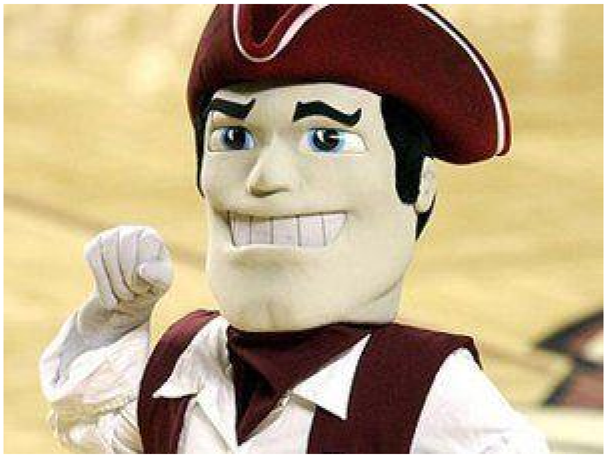


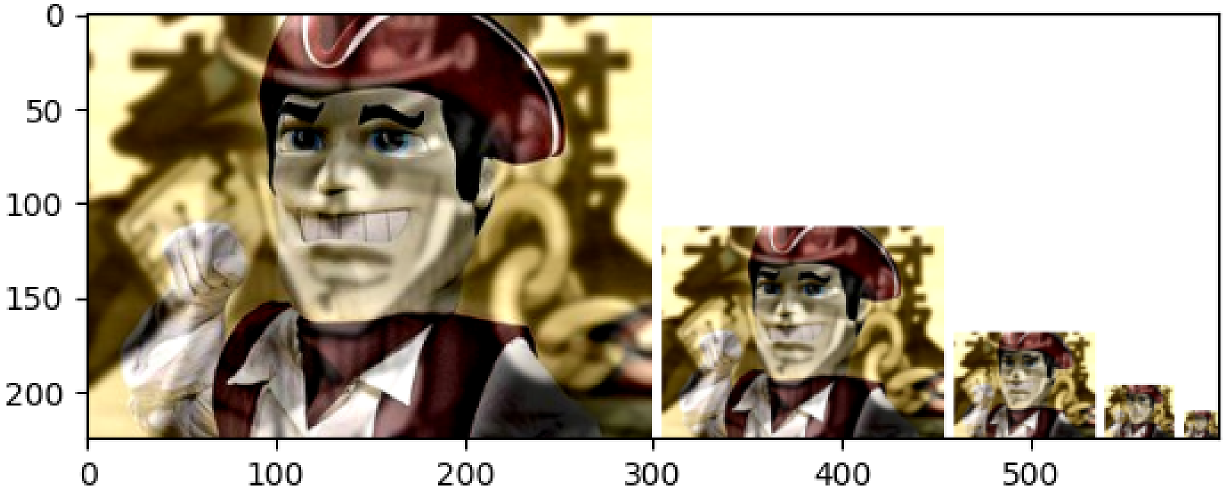
Hybrid image of cat and dog with sigma of 4 and 10 (dog and cat are reversed)



Additionally, I produced an hybrid image combining Sam the Minutemen and Jotaro from Jojo's bizarre adventure. The sigmas are [2, 11] respectively. I scaled, shifted, and cropped the image to get the desired image for hybrid operations

JoJo's
Bizarre
Adventure
STARDUST CRUSADERS





```
import numpy as np
from skimage.transform import resize
from scipy import ndimage
import matplotlib.pyplot as plt
import utils
from PIL import Image


def vis_hybrid_image(hybrid_image):
   scales = 5
   scale_factor = 0.5
   padding = 5

   original_height = hybrid_image.shape[0]
   num_colors = hybrid_image.shape[2]
   output = hybrid_image.copy()
   cur_image = hybrid_image.copy()

   for i in range(1, scales):
       output = np.concatenate((output, np.ones((original_height, padding, num_colors))),
axis=1)

       cur_image = resize(cur_image, (int(scale_factor*cur_image.shape[0]),
           int(scale_factor*cur_image.shape[1])))

       tmp = np.concatenate((np.ones((original_height - cur_image.shape[0],
cur_image.shape[1],
           num_colors)), cur_image), axis=0)

       output = np.concatenate((output, tmp), axis=1)

   return output
```

```
#Function test
def hybridImage(im1, im2, sigma1, sigma2):
   """
   input: 2 colored images, and sigmas for the gaussian filter

   output: the hybrid image

       hybrid image = blurry(I) + sharp(I):

       Ihybrid = blurry(I1; sigma1) + sharp(I2; sigma2) = I1 conv g(sigma) + I2 * I2
conv g(sigma2)

   """

   im1_gau = ndimage.gaussian_filter(im1, sigma1) # apply gaussian filter with sigma1 to
image1 and store it in variable im1_gau
   im2_gau = ndimage.gaussian_filter(im2, sigma2) # apply gaussian filter with sigma2 to
image2 and store it in variable im2_gau
   hybrid_im = im1_gau + im2 - im2_gau # apply the equation: hybrid_image = I1 conv
g(sigma) + I2 * I2 conv g(sigma2)
   hybrid_im = np.clip(hybrid_im, a_min = 0, a_max = 1) # clipping image to min of 0 and
max of 1

   return hybrid_im


if __name__ == '__main__':
   # img = utils.imread('../data/dog.jpg')
   # plt.imshow(vis_hybrid_image(img))
   # plt.show()

   # import images and declare hyperparmeters
   im1 = utils.imread('../data/jotaro.jpg')
   im2 = utils.imread('../data/minutemen.jpg')
   sigma1, sigma2 = [2, 11]

   # scale,shit, and crop image correctly
   scale_factor = 1.3
   im1 = resize(im1, (int(1/scale_factor * im1.shape[0]), int(1/scale_factor *
im1.shape[1]))) # scale image
   #crop image
   x = 115
   y = 30
   im1 = im1[x:225+x,y:300+y,:]

   hybrid_im = hybridImage(im1, im2, sigma1, sigma2) # get hybrid image
   plt.imshow(vis_hybrid_image(hybrid_im))
   plt.show()
```