Programming Project #1: Hybrid Images

CS445: Computational Photography

Part I: Hybrid Images

In [1]:

```
import cv2

import numpy as np
from matplotlib.colors import LogNorm
from scipy import signal
import utils
```

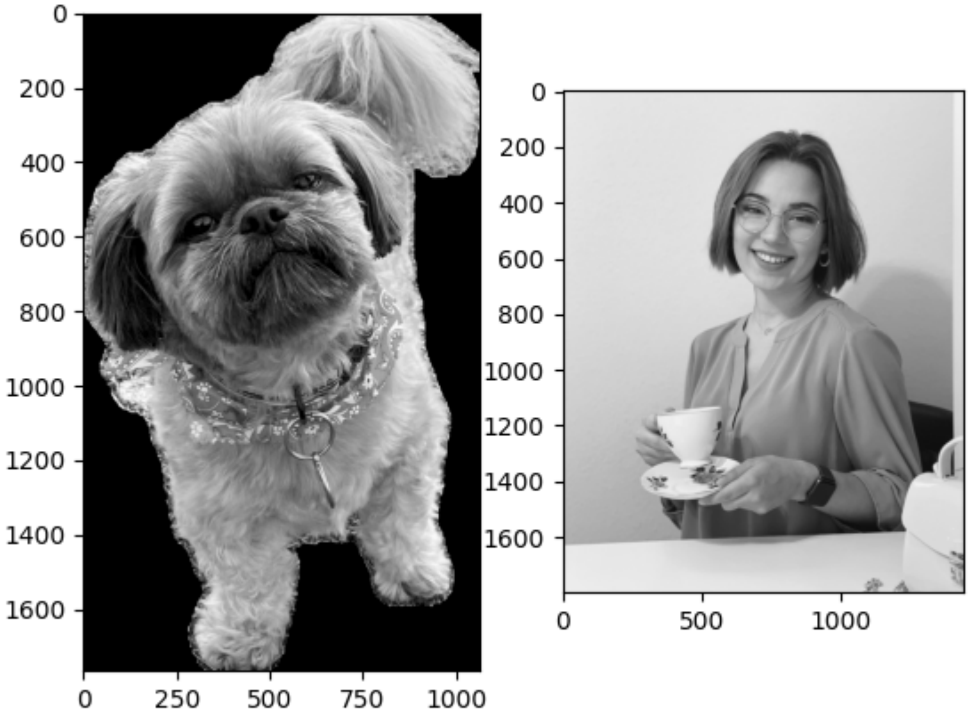
In [2]:

```
# switch from notebook to inline if using colab or otherwise cannot use i
# NOTE: I use ipympl cuz it's supposed to be newer and look nice
%matplotlib ipympl
import matplotlib.pyplot as plt
```

In [3]:

```
datadir = "./"

im1_file = datadir + 'max.png'
im2_file = datadir + 'MK_teacup.JPG'

im1 = np.float32(cv2.imread(im1_file, cv2.IMREAD_GRAYSCALE) / 255.0)
im2 = np.float32(cv2.imread(im2_file, cv2.IMREAD_GRAYSCALE) / 255.0)

fig, ax = plt.subplots(1, 2)
ax[0].imshow(im1, cmap='gray')
ax[1].imshow(im2, cmap='gray')
plt.show()
```

Figure

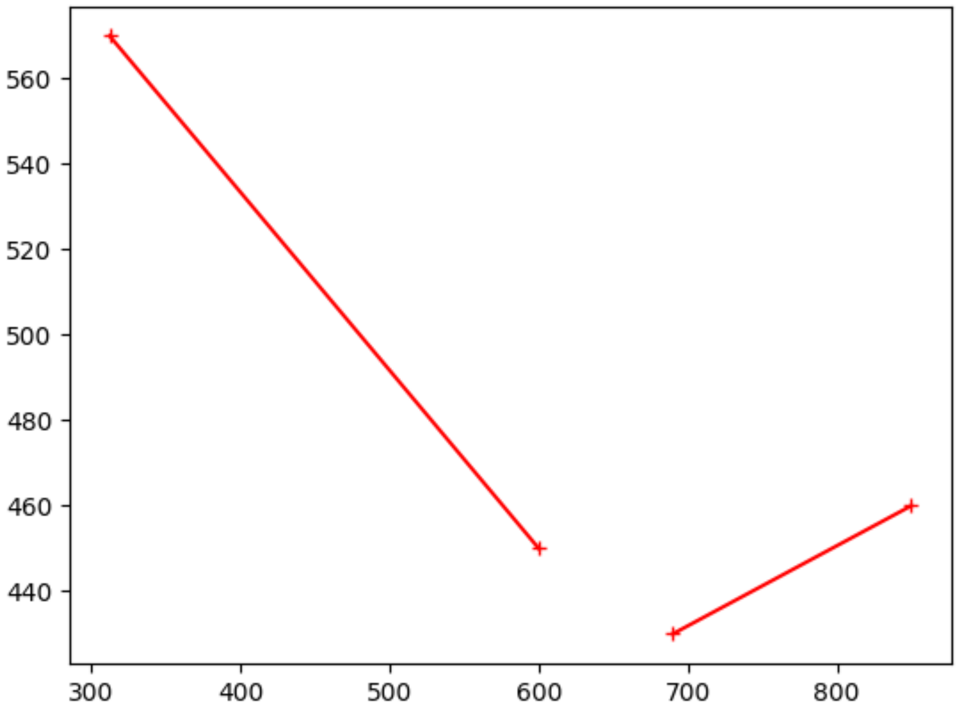


Get Eye Points

In [14]:

```
# Get points for eyes and plot
    # Note: I couldn't get the click program to work so this is manual
plt.figure()

pts_im1 = np.array([[312, 570], [600, 450]])
plt.plot(pts_im1[:,0], pts_im1[:,1], 'r-+')

pts_im2 = np.array([[690, 430], [850, 460]])
plt.plot(pts_im2[:,0], pts_im2[:,1], 'r-+')

plt.show()
```

Figure

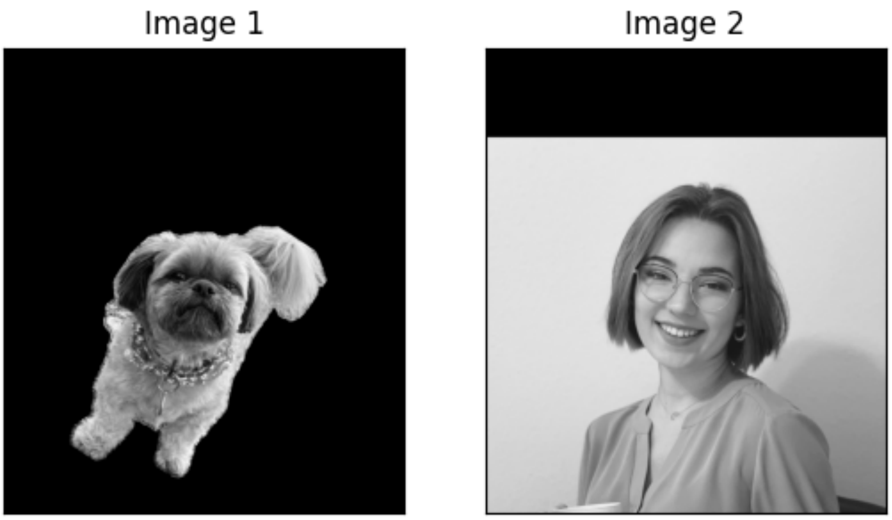


In [5]:
```
im1, im2 = utils.align_images(im1_file, im2_file,pts_im1,pts_im2,save_ima
```

In [6]:
```
# convert to grayscale
im1 = cv2.cvtColor(im1, cv2.COLOR_BGR2GRAY) / 255.0
im2 = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY) / 255.0
```

In [7]:
```
# Images sanity check
fig, axes = plt.subplots(1, 2)
axes[0].imshow(im1,cmap='gray')
axes[0].set_title('Image 1'), axes[0].set_xticks([]), axes[0].set_yticks(
axes[1].imshow(im2,cmap='gray')
axes[1].set_title('Image 2'), axes[1].set_xticks([]), axes[1].set_yticks(
plt.show()
```

Figure



In [8]:

```
# Since I needed 2 gauss filters I just made a function to save some code
def makePaddedGaussFilter(shape, sigma):
    ##### Create Gaussian kernel just like if this was spatial domain
    kernel_size = int(np.ceil(sigma) * 6 + 1)
    if kernel_size % 2 == 0:  # Ensure odd kernel size, or else weird sha
        kernel_size += 1

    gauss_kernel = cv2.getGaussianKernel(kernel_size, sigma)  # 1D kernel
    gauss_kernel = gauss_kernel @ gauss_kernel.T  # 2D kernel by outer pr

    # Create zero matrix to act as padding
    rows, cols = shape
    gaussian_filter = np.zeros((rows, cols), dtype=np.float32)

    # Add the kernel to the center of the zero matrix
    start_row = (rows - kernel_size) // 2
    start_col = (cols - kernel_size) // 2
    gaussian_filter[start_row:start_row + kernel_size, start_col:start_co

    # Upscale the filter to be from 0 to 1 (for mult with fft img)
    return gaussian_filter / np.max(gaussian_filter)
```

In [9]:

```
def hybridImage(im1, im2, sigma_low, sigma_high):
    '''
    Inputs:
        im1:    RGB (height x width x 3) or a grayscale (height x width)
                as a numpy array.
        im2:    RGB (height x width x 3) or a grayscale (height x width)
                as a numpy array.
        sigma_low: standard deviation for the low-pass filter
        sigma_high: standard deviation for the high-pass filter
```

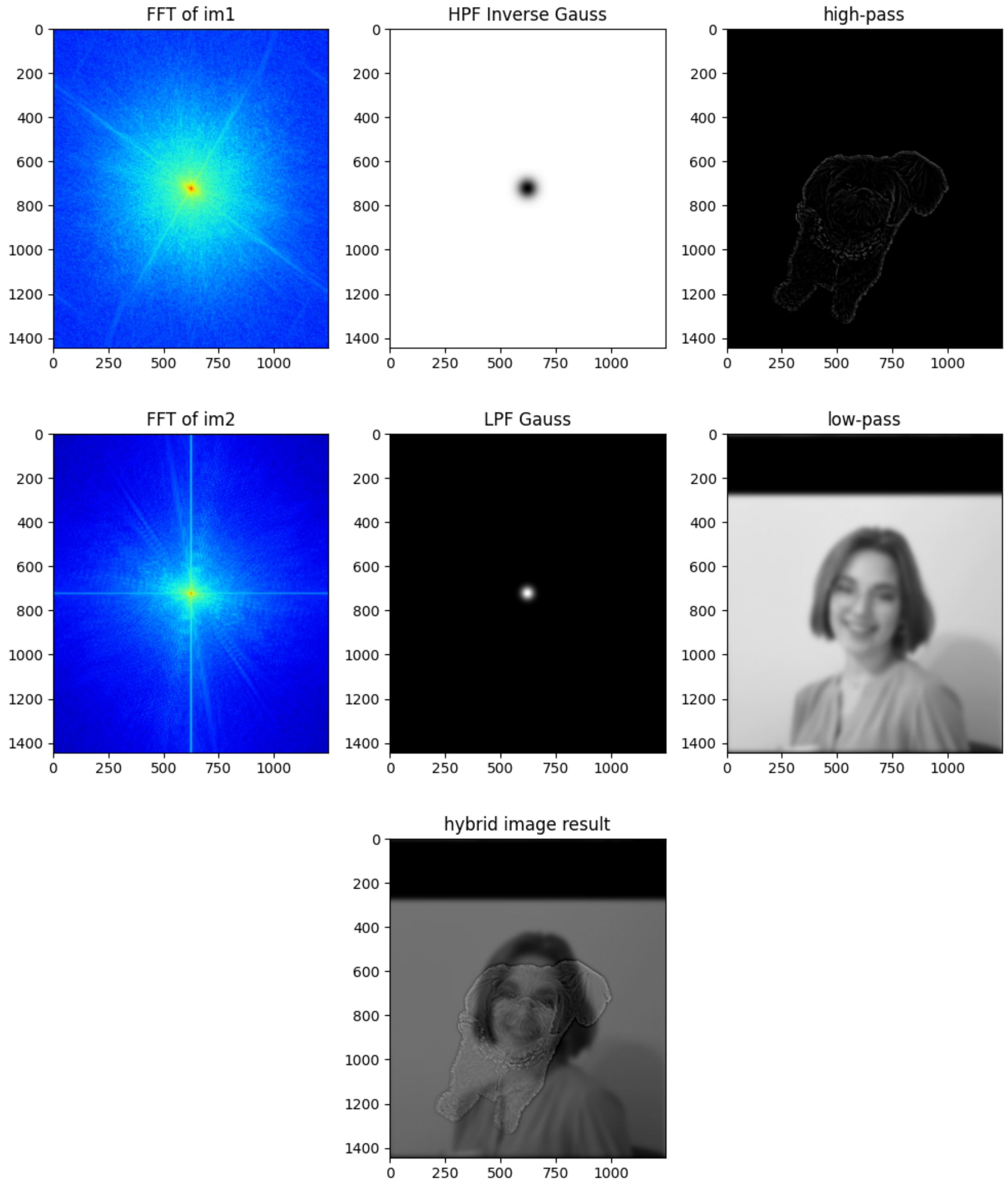
```
Output:
    Return the combination of both images, one filtered with a low-pa
    and the other with a high-pass filter.
'''

fig, ax = plt.subplots(3, 3, figsize=(10, 12))
rows, cols = im1.shape

##### Take FFT of images using numpy
im1_fft = np.fft.fft2(im1)
im1_fft_shifted = np.fft.fftshift(im1_fft)

im2_fft = np.fft.fft2(im2)
im2_fft_shifted = np.fft.fftshift(im2_fft)

##### Plot the spectrums of the images
magnitude_spectrum1 = 20 * np.log(np.abs(im1_fft_shifted) + 1)
ax[0][0].imshow(magnitude_spectrum1, cmap='jet')
ax[0][0].set_title('FFT of im1')

magnitude_spectrum2 = 20 * np.log(np.abs(im2_fft_shifted) + 1)
ax[1][0].imshow(magnitude_spectrum2, cmap='jet')
ax[1][0].set_title('FFT of im2')

##### High Pass Filter
hpf_gauss = makePaddedGaussFilter(im1.shape, sigma_high)
ones_matrix = np.ones((rows, cols), dtype=np.float32)
high_pass_filter = ones_matrix - hpf_gauss

ax[0][1].imshow(high_pass_filter, cmap='gray')
ax[0][1].set_title('HPF Inverse Gauss')

##### Low Pass Filter
low_pass_filter = makePaddedGaussFilter(im2.shape, sigma_low)

ax[1][1].imshow(low_pass_filter, cmap='gray')
ax[1][1].set_title('LPF Gauss')

##### Apply the filters directly in the frequency domain
high_filtered_fft = im1_fft_shifted * high_pass_filter
low_filtered_fft = im2_fft_shifted * low_pass_filter

##### Convert back to spatial domain
# Take only real part (I think this is fine?)
low_img_back = np.real(np.fft.ifft2(np.fft.ifftshift(low_filtered_fft
high_img_back = np.real(np.fft.ifft2(np.fft.ifftshift(high_filtered_f

# Filtered visualizations
print(f"high image range: [{high_img_back.min():.4f}, {high_img_back.
ax[0][2].imshow(high_img_back, cmap='gray', vmin=0, vmax=1)
ax[0][2].set_title('high-pass')

print(f"low image range: [{low_img_back.min():.4f}, {low_img_back.max
ax[1][2].imshow(low_img_back, cmap='gray', vmin=0, vmax=1)
ax[1][2].set_title('low-pass')

##### Combine images
hybrid_image = (low_img_back + high_img_back) / 2.0
ax[2][1].imshow(hybrid_image, cmap='gray', vmin=0, vmax=1)
ax[2][1].set_title("hybrid image result")
```

```
    fig.delaxes(ax[2][0])
    fig.delaxes(ax[2][2])
    plt.tight_layout()
    plt.show()

    return hybrid_image
```

```
In [10]: sigma_low = 20 # choose parameters that work for your images
         sigma_high = 30

         im_hybrid = hybridImage(im1, im2, sigma_low, sigma_high)
```

```
high image range: [-0.5814, 0.7380]
low image range: [-0.0001, 0.9006]
```

Figure



Part II: Image Enhancement

Two out of three types of image enhancement are required. Choose a good image to showcase each type and implement a method. This code doesn't rely on the hybrid image part.

Contrast enhancement

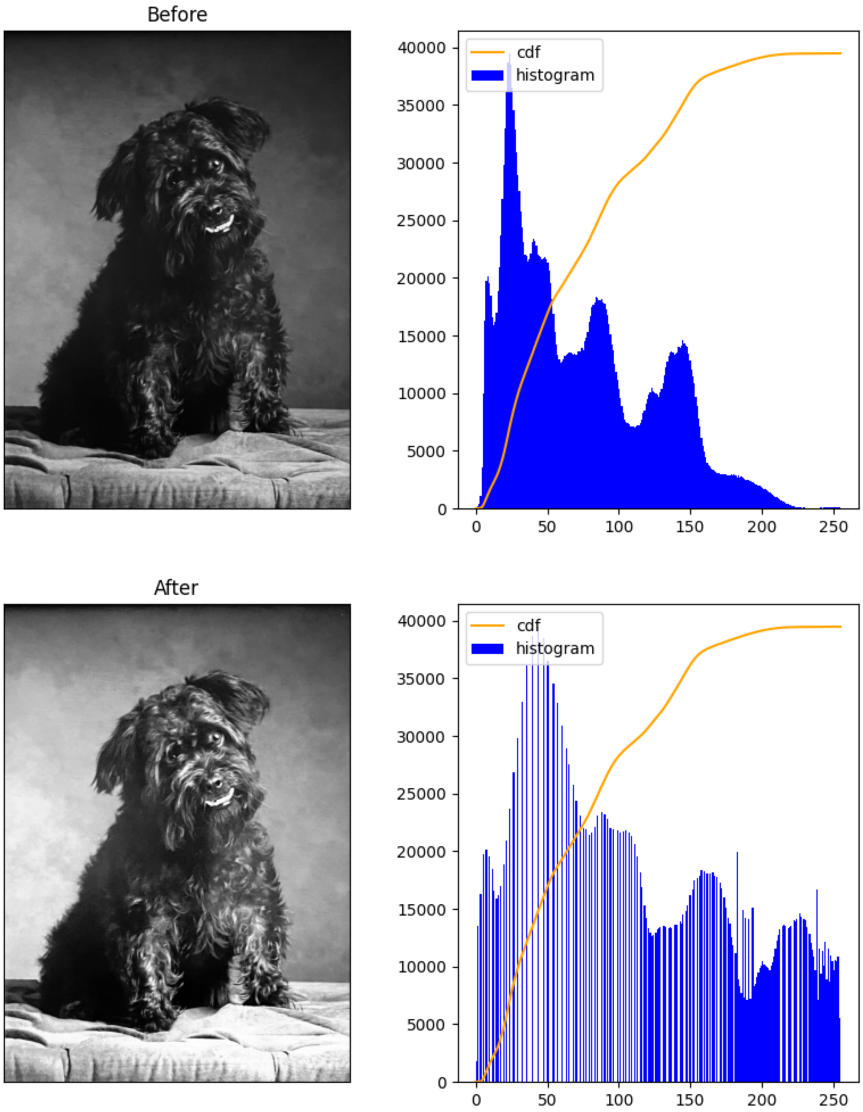
In [11]:

```
# Get the image
im4_file = datadir + "Loki_portrait.jpg"
im4 = cv2.imread(im4_file, cv2.IMREAD_GRAYSCALE)
fig, ax = plt.subplots(2, 2, figsize=(10, 12))

# Display before image
ax[0, 0].imshow(im4, cmap='gray')
ax[0, 0].set_title('Before'), ax[0,0].set_xticks([]), ax[0,0].set_yticks(

# Flatten pixels to 1d array, cuz histogram does not
    # care about dimensions
hist, bins = np.histogram(im4.flatten(), 256, range=[0,255])
cdf = hist.cumsum()
cdf_shrunken = (cdf / cdf.max()) * float(hist.max())

# plot histogram from original image
ax[0, 1].plot(cdf_shrunken, color='orange')
ax[0, 1].hist(im4.flatten(), 256, range=[0,255], color='blue')
ax[0, 1].legend(('cdf','histogram'), loc = 'upper left')

# Equalize the image with built in cv2 function
equ = cv2.equalizeHist(im4)

# Plot the equalized image
ax[1,0].imshow(equ, cmap='gray')
ax[1,0].set_title('After'), ax[1,0].set_xticks([]), ax[1,0].set_yticks([]

# Get the cdf and plot the histogram after equalizing
new_hist, new_bins = np.histogram(equ.flatten(), 256, range=[0,255])
new_cdf = hist.cumsum()
new_cdf_shrunken = (new_cdf / new_cdf.max()) * float(new_hist.max())

ax[1,1].plot(new_cdf_shrunken, color='orange')
ax[1,1].hist(equ.flatten(),256, range=[0,255], color='blue')
ax[1,1].legend(('cdf','histogram'), loc = 'upper left')
```

Out[11]:

```
<matplotlib.legend.Legend at 0x73eb47ca9f40>
```

Figure



Color enhancement

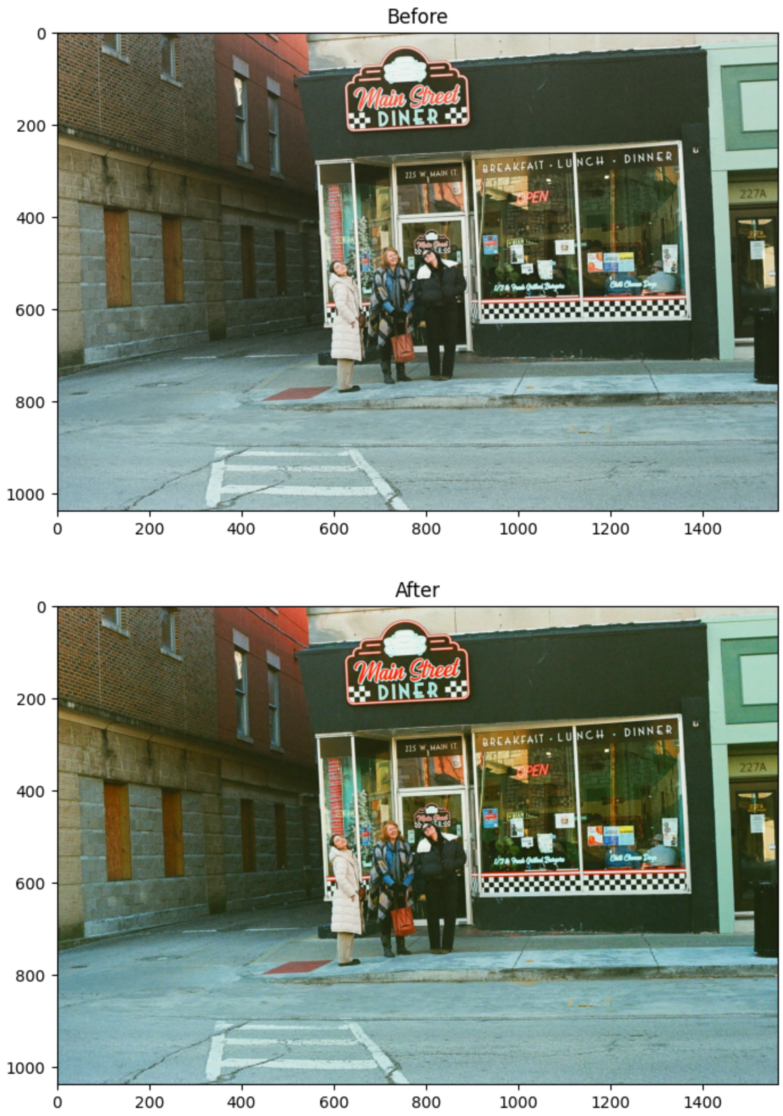
In [12]:

```
# Import image
im5_file = datadir + "gals.jpg"
im5 = cv2.imread(im5_file, cv2.IMREAD_COLOR_RGB)

fig, ax = plt.subplots(2, 1, figsize=(10, 12))

ax[0].imshow(im5)
ax[0].set_title("Before")

# Convert BGR im to the HSV space so we don't mess up luminance
hsv_img = cv2.cvtColor(im5,cv2.COLOR_BGR2HSV)
print(f'shape: {hsv_img.shape}')

# Increase saturation since that won't affect luminance, just chrominance
# Note: I needed to clip values to stay within 0-255 range as recommended
# The part II instructions.
hsv_img[:,:, 1] = np.clip(hsv_img[:,:, 1] * 1.4, a_min=0, a_max=255)

# Need to convert back to BGR for display
color_enhanced_im5 = cv2.cvtColor(hsv_img, cv2.COLOR_HSV2BGR)

ax[1].imshow(color_enhanced_im5)
ax[1].set_title("After")
```

```
shape: (1037, 1565, 3)
```

```
Out[12]:  Text(0.5, 1.0, 'After')
```

Figure



Color shift

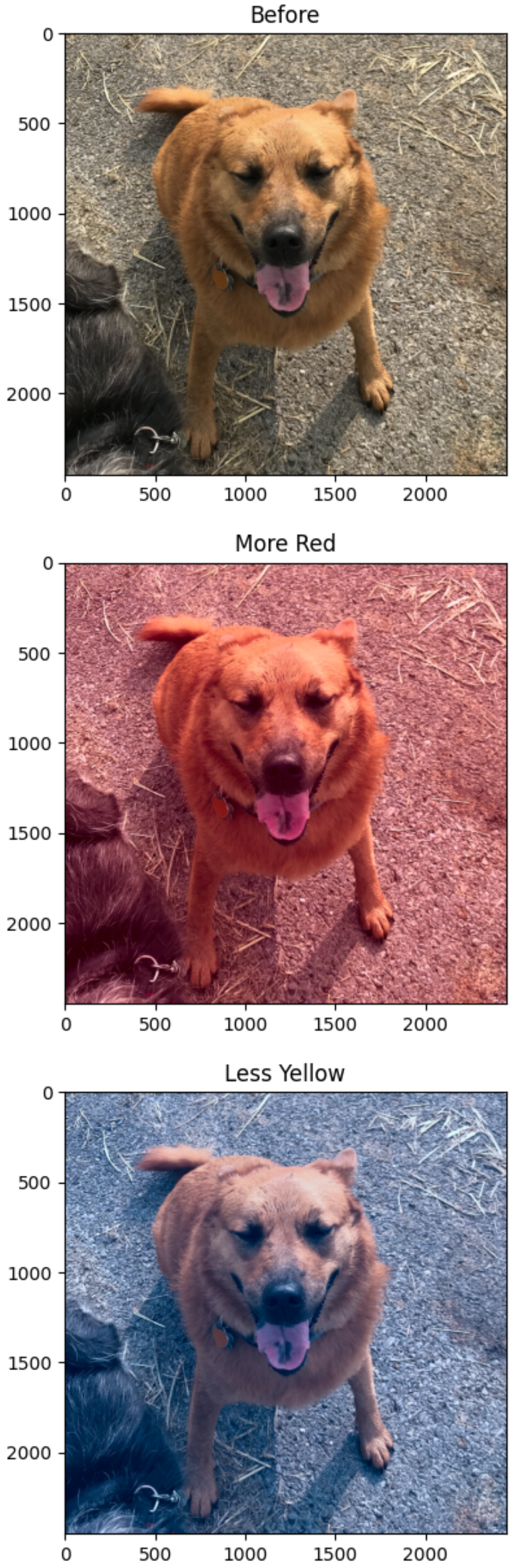
```
In [13]:  # Get the image
im6_file = datadir + "paddy.jpg"
```

```python
im6 = cv2.imread(im6_file, cv2.IMREAD_COLOR_RGB)

# Setup the subplots and print befor eimage
fig, ax = plt.subplots(3, 1, figsize=(8, 15))
ax[0].imshow(im6)
ax[0].set_title("Before")

# Colorshift to LAB for ease of color shifting without affecting luminanc
redshift_lab = cv2.cvtColor(im6, cv2.COLOR_RGB2LAB)
unyellow_lab = cv2.cvtColor(im6, cv2.COLOR_RGB2LAB)

# I was curious so I looked at the min/max channel values in original ima
print(f'min/max in l channel: {np.min(redshift_lab[:, :, 0])}, {np.max(re
print(f'min/max in a channel: {np.min(redshift_lab[:, :, 1])}, {np.max(re
print(f'min/max in b channel: {np.min(redshift_lab[:, :, 2])}, {np.max(re

# Add to each pixel in the a channel to increase red
redshift_lab[:,:,1] += 30
# Keep things in bounds
redshift_lab[:, :, 1] = np.clip(redshift_lab[:, :, 1], a_min=0, a_max=255

# Subtract from each pixel in the b channel to decrease yellow (which is
unyellow_lab[:,:,2] -= 30
# Keep things in bounds
unyellow_lab[:,:,2] = np.clip(unyellow_lab[:,:,2], a_min=0, a_max=255)

# COnvert back to RGB and display
red_shifted_im6 = cv2.cvtColor(redshift_lab, cv2.COLOR_LAB2RGB)
ax[1].imshow(red_shifted_im6)
ax[1].set_title("More Red")

yellow_subtracted_im6 = cv2.cvtColor(unyellow_lab, cv2.COLOR_LAB2RGB)
ax[2].imshow(yellow_subtracted_im6)
ax[2].set_title("Less Yellow")
```

```
min/max in l channel: 0, 255
min/max in a channel: 119, 171
min/max in b channel: 121, 176
```

```
Out[13]: Text(0.5, 1.0, 'Less Yellow')
```

Figure


Before
0
500
1000
1500
2000
0
500
1000
1500
2000
More Red
0
500
1000
1500
2000
0
500
1000
1500
2000
Less Yellow
0
500
1000
1500
2000
0
500
1000
1500
2000