**CS 445**

**Tips for Project 1**

* You will first learn all the material you need for the hybrid image part, so do that part early.
* When specifying a Gaussian filter, choose a value for standard deviation and then choose a filter size such that the values near the edges of the filter are near zero
* When displaying FFT images, it may help to specify a scaling range using the 'norm ' argument of matplotlib plt.imshow(im, norm=LogNorm(vmin, vmax))
* With high-pass filtered images or laplacian pyramid images, many of the values will be negative. To display them, you need to rescale the images to range from 0 to 1. numpy.clip and image.astype(numpy.uint8) might be useful.
* If you have trouble with interactive plots or matlibplt, you can use inline and get coordinates from another program (see below). Mac users may find it helps to open your notebooks in Chrome.
* For “Color Shift” B&W, make sure that your solution does not have side effects. E.g. if you make something more red, other pixels should not become less green. Masking is one way to do this, i.e. modifying only the pixels that are more red than green. A good solution requires choosing a good color space and modifying chrominance values. This is the trickiest B&W to get right.

**Google Colab**

Google Colab is a great way to run Jupyter notebooks and complete your projects, but there are some limitations:

* **Timeouts**: If your session is inactive for some time, your session may be stopped. Your notebook edits will have been saved, but you need to re-run everything and lose any temporary files. Using Colab Pro reduces timeout incidences.
* **Session limits**: Colab is limited to 12 hours sessions. Colab Pro is limited to 24 hour sessions.
* **Storage**: File storage in Colab sessions is temporary. One solution is to save your helper code and files in Drive (e.g. store in "/content/drive/My Drive/cs445\_projects/proj1/") and then copy them over. You can connect to Drive with this code:

from google.colab import drive

drive.mount('/content/drive')

* **No interactive matlibplt**: You can’t use '%matlibplt notebook' to show figures that you can interact with. Use '%matlibplt inline' instead and manually enter coordinates. You can get the pixel coordinates using MS Paint or Gimp, for example.
* **Google Colab Pro** gives you access to faster machines, good GPUs, and longer processes. It might be worth the $10 / month.

**Python examples**:

Read image with values scaled from 0 to 1:

np.float32(cv2.imread(imfn, cv2.IMREAD\_GRAYSCALE))/255

np.float32(cv2.imread(imfn))/255

Create a 2D Gaussian kernel:

ksize = np.int(np.ceil(sigma)\*6+1)

fil = cv2.getGaussianKernel(ksize, sigma) # 1D kernel

fil = fil\*np.transpose(fil) # 2D kernel by outer product

Filter

im\_fil = cv2.filter2D(im, -1, fil)

Get the size of a numpy array

data.shape

Resize

im\_big = cv2.resize(im, None, fx=2.0, fy=2.0) # double the size

Display a grayscale image

plt.figure(figsize=(15,15))

plt.imshow(im,cmap='gray')

Display an image within a particular value range

plt.imshow(im,cmap='gray', vmin=0.0, vmax=0.5)

Display a BGR image (remember that cv2 loads in BGR channel order, while plt uses RGB order)

plt.figure(figsize=(15,15))

plt.imshow(im[:, :, [2,1,0]) # maps BGR to RGB

Operators (add, subtract, raise to power, element-wise multiply, transpose)

+, -, \*\*, np.multiply(im1,im2), np.transpose

Get FFT magnitude and display:

fftmag = np.abs(np.fft.fftshift(np.fft.fft2(imc)))

plt.imshow(fftmag,norm=LogNorm(fftmag.min(),fftmag.max()),cmap='jet')

Convert colors

im\_RGB = cv2.cvtColor(im, cv2.COLOR\_BGR2RGB)

im\_gray = cv2.cvtColor(im, cv2.COLOR\_BGR2Gray)

im\_HSV = cv2.cvtColor(im, cv2.COLOR\_BGR2HSV)

im\_LAB = cv2.cvtColor(ime, cv2.COLOR\_BGR2LAB)

im = cv2.cvtColor(im\_LAB, cv2.COLOR\_LAB2BGR)

Manipulate values in a single channel

im\_b = im\_bgr[:,:,0] # select first channel

mask = im\_b < 0.5 # check which values are less than 0.5

im\_b2 = im\_b[mask] / 2 # divide only those values by 2

im\_bgr[:,:,0] = im\_b2 # replace the first channel

Clip grayscale pixel values to 0 to 1 before displaying

plt.imshow(np.clip(im,0,1),cmap='gray')