CS216 : Assignment 1

Please edit the cell below to include your name and student ID #

---

name:

SID:

---

0. Python & Jupyter Notebook Tutorials

We will make extensive use of Python's numerical arrays (NumPy) and interactive plotting (Matplotlib) in Jupyter notebooks for the course assignments. This first assignment is intended as a gentle warm up in case you haven't used these tools before. If you haven't used Jupyter before I'd suggest you start by reading through the following tutorials:

https://jupyter-notebook.readthedocs.io/en/stable/notebook.html#starting-the-notebook-server (https://jupyter-notebook.readthedocs.io/en/stable/notebook.html#starting-the-notebook-server)
https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Notebook%20Basics.ipynb (https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Notebook%20Basics.ipynb)
https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Running%20Code.ipynb (https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Running%20Code.ipynb)

If you need a refresher on Python, this is a concise tutorial:

http://cs231n.github.io/python-numpy-tutorial/ (http://cs231n.github.io/python-numpy-tutorial/)

This page gives a good introduction to NumPy and many examples of using NumPy along with Matplotlib:

http://scipy-lectures.org/intro/numpy/array_object.html (http://scipy-lectures.org/intro/numpy/array_object.html)

You should also get comfortable with searching through the documentation as needed

https://numpy.org/doc/stable/reference/index.html (https://numpy.org/doc/stable/reference/index.html)

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)

In [ ]:
```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt

# for problem 1
import os

# for problem 2
from scipy.spatial.distance import cdist

# for problem 3
from scipy.ndimage import gaussian_filter
from scipy.ndimage import zoom
```

1. Image Manipulation

In this exercise you will write code which loads a collection of images (which are all the same size), computes a pixelwise average of the images, and displays the resulting average. You can see some interesting related discussion here http://people.csail.mit.edu/torralba/gallery/ (http://people.csail.mit.edu/torralba/gallery/) and here http://www.salavon.com/work/SpecialMoments/ (http://www.salavon.com/work/SpecialMoments/)

Download the images provided on the Canvas course website for this assignment `averageimage_data.zip` . There are two sets, `set1` and `set2` . Notice that they are not necessarily all the same size within a single set.

1.1 Implementation [18pts]

Write a function in the cell below that loads in one of the sets of images and computes their average. You can use the **os.listdir** to get the list of files in the directory. As you load in the images, you should compute an average image on the fly. Color images are represented by a 3-dimensional array of size (HxWx3) where the third dimension indexes the red, green and blue channels. Since the images are not the same size, you will first need to resize each image to a fixed size (e.g. something like 300x215). You can set the height and width to be size of the first image you load, or if you want to be a bit more clever, make it the average dimensions of the images in the directory.

You should encapsulate your code in a function called **average_image** that takes the image directory as an input and returns the average of the images in that directory. Your function should implement some error checking. Specifically your function should skip over any files in the directory that are not images (**plt.imread** will thrown an **OSError** if the file is not an image). It should also ignore images that are not color images.

NOTE: In general for this course we will think of images as containing floating point values. In practice, when stored in files they are typically quantized and stored as uint8 values. When you load in an image it is usually a good idea to just convert it to a float in the range [0,1] to avoid potential bugs later when you try to do mathematical operations of the pixel values. You can use some code like:

```
if (I.dtype == np.uint8):
   I = I.astype(float) / 256.
```

to carry this out.

In [ ]:
```
def average_image(dirname):
    """
    Computes the average of all color images in a specified directory and returns the result.

    Parameters
    ----------
    dirname : str
        Directory to search for images


    Returns
    -------
    numpy.array (dtype=float)
        HxWx3 array containing the average of the images found

    """

#
## some useful functions for this part
#
## load an image from a file into a numpy array
#
# I = plt.imread(...)
#
## looping over files in a directory
#
#  for file in os.listdir(dirname):
#       filename = os.path.join(dirname,file)
#       if os.path.isfile(filename):
#           try:
#               ...
#            except OSError as err:
#               ...
#               continue
```

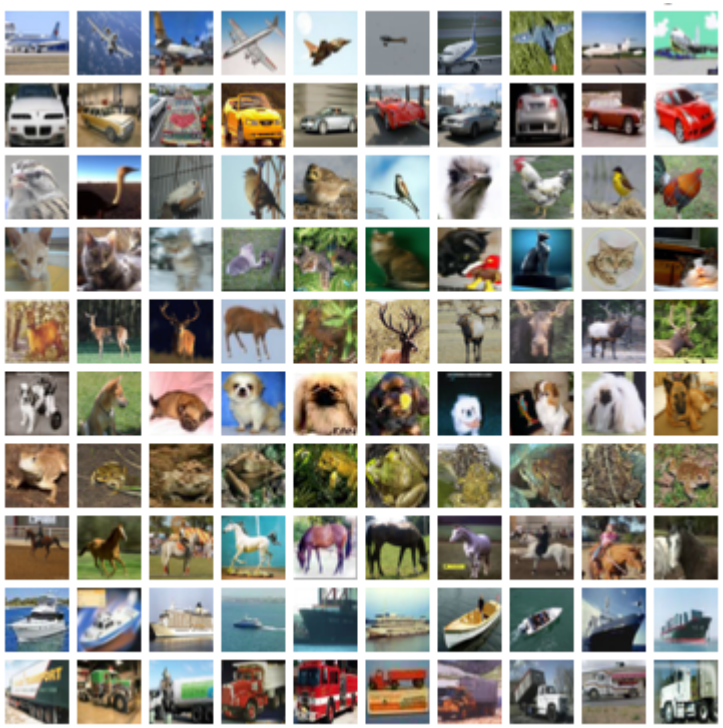
1.2 Visualization [7pts]

Write code below which calls your **average_image()** function twice, once for each set of images. Display the resulting average images. Also display a single example image from each set for comparison. Discuss briefly what you see. In what ways does the average differ from the individual examples?

In [ ]:
```
#
## plt.imshow(..)
## plt.show(..)
#
```

[add your discussion here]

2. Image Classification



In this problem you will write code that will classify small 32x32 color images into one of 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). You should implement your functions "from scratch" (i.e. don't use other libraries beyond the ones we've already imported).

You will images from here: CIFAR-10 (http://www.cs.toronto.edu/~kriz/cifar.html), using the link for the Python CIFAR-10 dataset. Please refer to the webpage for the description of the dataset format. I suggest you first debug your code on a subset of the training and testing data, and then run it on all the data. If you are running into memory or computational issues, it is fine to use a subset of the data, but please make note of this in your writeup.

2.1 Load in the data

To make sure that you understand the data format, write code to display the first airplane in the test set.

In [ ]:

```
```

2.2 Nearest Neighbor [15pts]

Write a function that uses nearest-neighbor (NN) classification to predicts a label for a test image by returning the label of which ever training images is closest (measured using the Euclidean distance between the vector of pixel values). You can use the SciPy function ***scipy.spatial.distance.cdist*** to efficiently compute the pairwise distances between the test examples and all training examples.

In [ ]:

```
def nn_classify(....):
    """
    Given a set of training images with a label for each image and
    set of test images, this function performs nearest-neighbor
    classification and returns a set of labels for the test images

    Parameters
    ----------
    ???


    Returns
    -------
    ???


    """
```

2.3 Evaluation Metrics [10pts]

Write a function to compute a class confusion matrix. The confusion matrix will be an array where entry (i,j) is the fraction of times an image of class 'i' was predicted to be class 'j'. The rows of the matrix should sum to 1. Display this matrix as an image using ***imshow(cm, interpolation='nearest', cmap=cmap)*** with an appropriate colormap. You should also display a ***colorbar*** to help interpreting the values. Finally, print out the average classification accuracy across all classes (the average of the diagonal values in this matrix).

In [ ]:
```
def evaluate(true_labels,predicted_labels,k):
    """
    Given a set of ground-truth labels, a set of predicted labels
    and the number of classes (i.e. NumPy arrays with integers in [0,..,k-1])
    compute and display the class confusion matrix and the average
    classification accuracy.

    Parameters
    ----------
    ???


    Returns
    -------
    ???


    """
```

In [ ]:
```
```

2.4 Experimentation [10pts]

Experiment with several different choices of distances besides Euclidian distance (e.g., cosine, correlation). You can do this by passing different options to ***cdist*** when computing the distance. For each one you try, display the class confusion and average accuracy. To implement this in a clean way, you may want to modify your **nn_classify** function to take the metric as a parameter.

In [ ]:
```
```

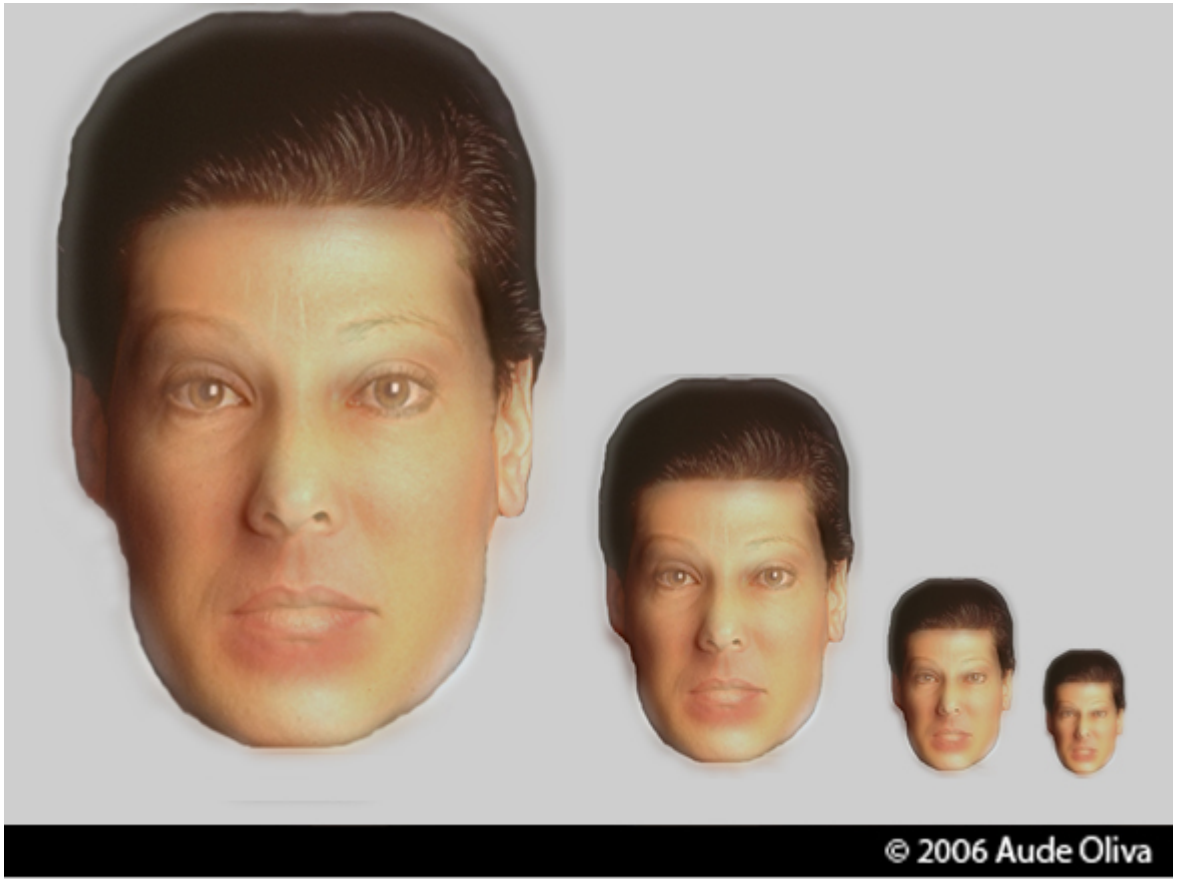
In [ ]:
```
```

2.5 Discussion [5pts]

Which metric performs the best? Provide some explanation as to why you think this is the case. What classes are most commonly confused? Display a few examples of pairs of images from commonly confused classes and explain why the confusion might occur.

Type *Markdown* and LaTeX: α^2

3. Hybrid Images



The goal of this problem is to write an image filtering function and use it to create hybrid images (https://en.wikipedia.org/wiki/Hybrid_image) using a simple version of the technique described in this SIGGRAPH 2006 paper by Oliva, Torralba, and Schyns (https://dl.acm.org/doi/10.1145/1141911.1141919). Hybrid images are static images that change in interpretation as

a function of the viewing distance. The basic idea is that high frequency tends to dominate perception when it is available, but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different viewing distances.

3.1 Image Filtering [15pts]

Implement a function that takes a color image as input and splits it into a low-frequency component and a high-frequency component. To get the low-frequency component, you can blur the image using a Gaussian filter using ***scipy.ndimage.gaussian_filter*** where the **sigma** parameter of the Gaussian controls the amount of blur. To get the high-frequency component, you can simply take the original image and subtract out the low-frequency component.

In [ ]:
```
def split_bands(image,sigma):
    """
    Given an image, split the image into two frequency bands
    and return the two components (low,high)

    Parameters
    ----------
    image : numpy.array (dtype=float)
        a tuple of two HxWx3 arrays containing low/high freq components respectively

    sigma : float
        the width of the gaussian filter

    Returns
    -------
    tuple of numpy.array (dtype=float)
        two HxWx3 arrays containing low/high freq components respectively

    """
```

3.2 Visualization [5pts]

Experiment with your ***split_bands*** function on an image of your choosing which contains a mix of content. Display the high and low frequency component for several different settings of **sigma**. Make sure and experiment to find what values of **sigma** are "too large" or "too small" to be useful and then select a few intermediate values that span this range.

You will encounter a challenge here which is that while the low-frequency image will contain all values in the range [0,1], the high-frequency band image in general will contain negative values which **matplotlib** will not be able to display. For visualization purposes, you can compute an offset value and scale in order to map the values back to [0,1].

In [ ]:
```
# code to visualize
```

3.3 Analysis [10pts]

What is the relation between the value of **sigma** and the content of your high and low-pass images? Ideally you should be able to come up with a formula for the frequency (measured in units of cycles-per-pixel) at which the Gaussian filter with a given **sigma** (measured in units of pixels) reduces the amplitude by 0.5. This is sometimes known as the crossover point between your low-pass and high-pass filter.

If you can't figure out how to determine this analytically, you are welcome to determine it experimentally by synthesizing images whose brightness is a **sin** function with a specific frequency, e.g.

```
x = (2*np.pi/25)*np.arange(100)
I = np.sin(x+x.reshape(100,1))   % 100x100 image with sinusoidal brightness
```

and then empirically determining the value of sigma you need to choose in order to have your low- and high-pass outputs have equal amplitudes.

....your answer/explanation/experiment here...

3.4 Hybrid Image Generation [5pts]

Choose two images for which you would like to make a hybrid. The images should be fairly closely related (e.g., two different faces, a cat and dog face, two different words of text etc.). You will also need to adjust the images so that they are the same size and are well aligned (e.g., for faces the eyes should be at the same location). You can do this in code or in a photo editing tool.

You can then run your ***split_bands*** function on both images. Finally produce your two hybrid images by adding the low-frequency content of one image with the high-frequency of the other. Make sure and display your two input images and the two resulting hybrid images in your notebook. As before, you may need to offset the result to keep all the values positive.

As suggested by Figure 5 in the Oliva et. al. paper, you may be able to get better results by actually running your split bands function with two different sigmas in order to insert a larger gap between the high and low frequency bands. Feel free to experiment to get a good result.

In [ ]: