# CS 491/691 Project 4

Elaine Chu and Zachary Black

Fall 2019

**Note: The following functions are implemented with the numpy, matplotlib, and os functions**.

## 1 pca.py

### 1.1 compute_Z(X, centering=True, scaling=False)

This function takes in an input array and two optional boolean parameters. If the first boolean parameter is set to true, then the input array data is centered based on the mean value for each column. If the second boolean parameter is set to true, then each column in the input data is divided by that column's standard deviation. The resultant array after these operations is returned.

```
def compute_Z(X, centering=True, scaling=True):
    Z = []
    if centering:
        column_averages = [compute_mean(X[:, i])
            for i in range(len(X[0]))]
        Z = np.array([subtract(X[:, i], column_averages[i])
            for i in range(len(X[0]))]).transpose()
    if scaling:
        column_std = [compute_std_dev(X[:, i])
            for i in range(len(X[0]))]
        Z = np.array([divide(Z[:, i], column_std[i])
            for i in range(len(Z[0]))]).transpose()

    if centering==False and scaling==False:
        Z = X
    return Z
```

### 1.2 compute_covariance_matrix(Z)

This function takes an input array, Z, as a parameter and returns the result of the calculation of the transpose of Z multiplied by Z.

```
def compute_covariance_matrix(Z):
    zT = np.array(Z).transpose()
```

```
    return np.dot(zT, Z)
```

### 1.3 find_pcs(COV)

This function takes an input array as a parameter which is a covariance matrix. This function computes the eigenvalues and eigenvectors of the covariance matrix. Then the eigenvalues are sorted in descending order and the eigenvectors are also sorted based on their respective eigenvalues.

```
def find_pcs(COV):
    L, PCS = np.linalg.eig(COV)

    idx = L.argsort()[::-1]
    L = L[idx]
    PCS = PCS[:,idx]

    return(L, PCS)
```

### 1.4 project_data(Z, PCS, L, k, var)

This function takes in the Z matrix, eigenvalues, eigenvectors, k amount of principal components, and var a float. This function projects the input data into k dimensions using the most significant eigenvectors.

```
def project_data(Z, PCS, L, k, var):
    eucld = np.linalg.norm(PCS, axis=0)
    PCS_norm = PCS/eucld

    if k==0:
        cuml = 0
        for i in range(len(L)):
            cuml += L[i]
            k += 1
            if (cuml/sum(L) >= var):
                PCS_new = PCS_norm[:,:k]

    if var==0:
        PCS_new = PCS_norm[:,:k]

    feat_vec = PCS_new.T
    Z_adj = Z.T
```

```
    Z_star = np.dot(feat_vec, Z_adj)

    return(Z_star)
```

# 2 compress.py

## 2.1 compress_images(DATA,k)

This function takes flattened image data (DATA) and compresses the image(s) using k number of principal components. The output of this function exports files int a directory named Output. Prior to exporting, all images are reshaped and rescaled.

```
def compress_images(DATA,k):
    DATA = DATA.T
    ## Center data
    mean_mat = np.mean(DATA, axis=0)
    Z = DATA - mean_mat

    ## Compute covariance matrix
    Ztp = Z.T
    COV = np.dot(Ztp,Z)

    ## Calculate eigenvalues L and eigenvectors PCS
    L, PCS = np.linalg.eig(COV)

    idx = L.argsort()[::-1]
    L = L[idx].real
    PCS = PCS[:,idx].real

    ## Project data onto k eigenvectors
    eucld = np.linalg.norm(PCS, axis=0)
    PCS_norm = PCS/eucld

    PCS_new = PCS_norm[:,:k]

    feat_vec = PCS_new.T

    Z_star = np.dot(feat_vec,Ztp)

    ## Return data to original mean
    arr = np.dot(feat_vec.T, Z_star)

    Output = arr.T + mean_mat
```

```
    out_min = np.amin(Output)
    out_max = np.amax(Output)
    Output_norm = (Output - out_min)/(out_max-out_min)  # normalize
    Output_scale = Output_norm * 255  # set to Image values
    Output_scale = Output_scale.astype(np.uint8)  # return to og dtype

    ## Check if Output directory exists
    path = 'Data/Output/'
    if not os.path.exists(path):
        os.mkdir(path)
        print ("Directory", path, " created")
    else:
        print("Directory", path, " already exists")

    ## For each row, reshape into original image size and save
    for j in range(Output_scale.shape[0]):
        curr_image = Output_scale[j,:]
        reshape_image = curr_image.reshape(image_shape)
        output_path = os.path.join(path,'Out'+filenames[j]
            +'_k_'+str(k)+'.png')
        plt.imsave(output_path,reshape_image,
            cmap=plt.get_cmap(name = 'gray'),format='png')

    return(Output)
```

Implementation Steps:

- Transpose DATA so that columns represent dimensions and rows represent each image
- Center DATA by subtracting the mean of each dimension (column) from each value –¿ Z
- PCA: Compute covariance matrix, calculate sorted eigenvalues L and eigenvectors PCS
- Project Z onto k principal components –¿ Z_star
- Return data, Z_star to original mean, rescale output array to reflect image pixel values between 0 and 255
- Check if Output directory already exists, create if not.
- For each scaled image (row), reshape to original image dimensions and export into Output folder as .PNG image with correct name

### 2.2 load_data(input_dir)

This function takes an input directory and outputs the DATA matrix with one flattened image per column and each pixel representing a row.

```
def load_data(input_dir):
    folder = os.fsencode(input_dir)
    global filenames
    filenames =[]

    for file in os.listdir(folder):
        filename = os.fsdecode(file)
        filenames.append(filename)

    ## Test to see dimensions of DATA array
    temp = plt.imread(str(input_dir + filenames[0]))
    global image_shape  # sets image_shape as global variable
    image_shape = temp.shape

    temp_flat = np.ndarray.flatten(temp)

    cols = len(filenames)
    rows = len(temp_flat)

    DATA = np.zeros((rows,cols),dtype=np.uint8)
    # initiate DATA array with zeros of correct size

    for i in range(len(filenames)):
        temp = plt.imread(str(input_dir + filenames[i]))
        temp_flat = np.ndarray.flatten(temp)
        DATA[:,i] = temp_flat

    return(DATA)
```

Implementation Steps:

- Identify folder as the input directory
- Set global variable for filenames to call later
- Initialize empty filenames array
- For each file in the directory, store the file name as filenames
- Read first image to establish image_shape as a global variable to call later
- Initialize empty DATA array with the correct number of rows (pixels of each image) and columns (number of images in directory)
- For each file, flatten the pixels and input into DATA array

### 2.3 Write-Up

Include 5 compressed images, and the original, from the Data/Train directory using $k = [10, 100, 500, 1000, 2000]$ in your write-up.

Figure 1: Original

Figure 2: k=10

Figure 3: k=100

Figure 4: k=500

Figure 5: k=1000

Figure 6: k=2000