Hong Seh Looi
frank@minerva.kgi.edu

The program resizes input images into 512 pixel in width and 384 in height to preserve the original aspect ratio of the photo (original pixel 4032 × 3024). We also store images as an array, which is an appropriate data structure for sklearn PCA's fit and transform method.

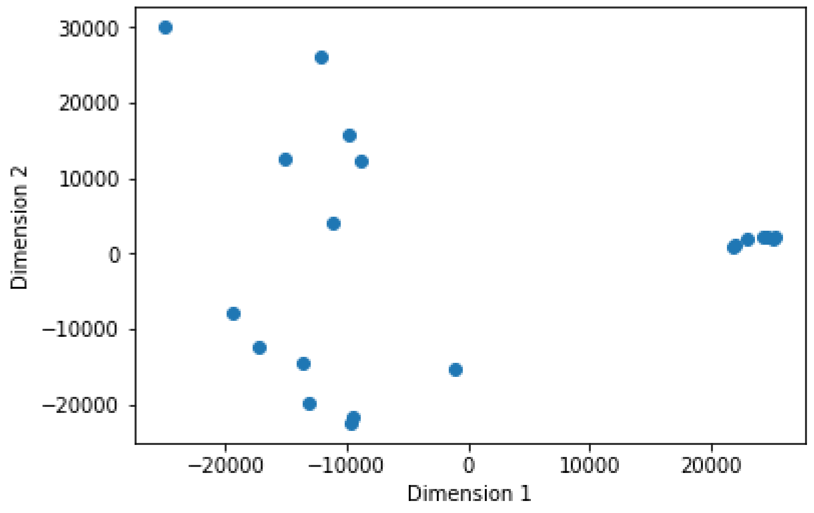
```
from PIL import Image, ImageOps
import glob
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

%matplotlib inline

def img_to_array(filename):
    """Takes an image file, resizes to 512 by width or height
    (whichever larger) and convert to array"""
    img = Image.open(filename)
    desired_size = 512
    old_size = img.size
    ratio = float(desired_size)/max(old_size)

    # preserving image aspect ratio
    new_size = tuple([int(x*ratio)for x in old_size])
    img = img.resize(new_size,Image.ANTIALIAS)

    # converting to array
    img = list(img.getdata())
    img = map(list, img)
    img = np.array(img)
    img_wide = img.reshape(1,-1)
    return img_wide[0]

def plot_img_from_arr(arr):
    """Reconstruct image from 1D array"""
    img = Image.new('RGB',(512,384))
    a1 = np.empty((196608,), dtype=object)
    a1[:]=[tuple([int(j) for j in i]) for i in arr.reshape(196608, 3)]
    img.putdata(a1)
    return np.asarray(img)

# Accessing 20 photos for the LBA into an array
photos = glob.glob('LBA/*.jpg')
raw_data = [img_to_array(photo) for photo in photos]
```

The original images are visualized by the code below:

```python
# Show 5 original images
for i in range(5):
    pil_im = Image.open(photos[i*4], 'r')
    plt.imshow(np.asarray(pil_im),aspect='auto')
    plt.axis('off')
    plt.show()
```

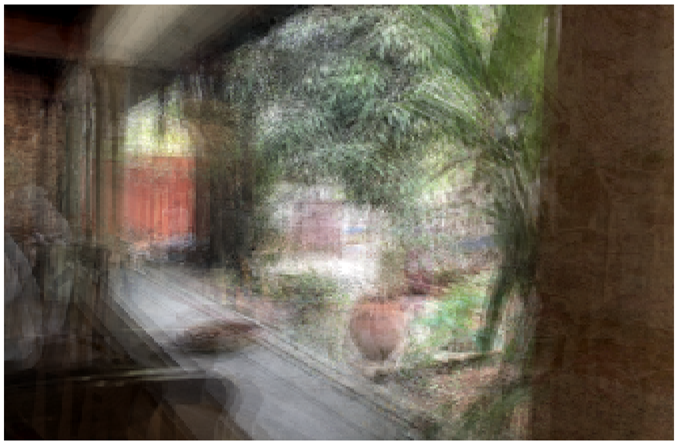
The following code use sklearn's PCA to reduce dimensions of the images into two dimensions. We can use a scatter plot to visualize the eigenvalues of each image. The two dimensions are mutually uncorrelated.

```
# Fitting PCA with 2 components
pca = PCA(n_components = 2)
pca_data = pca.fit_transform(raw_data)
pca_inverse = pca.inverse_transform(pca_data)

# Visualize 2D data as a scatterplot
plt.scatter(*zip(*pca_data))
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.show()
```



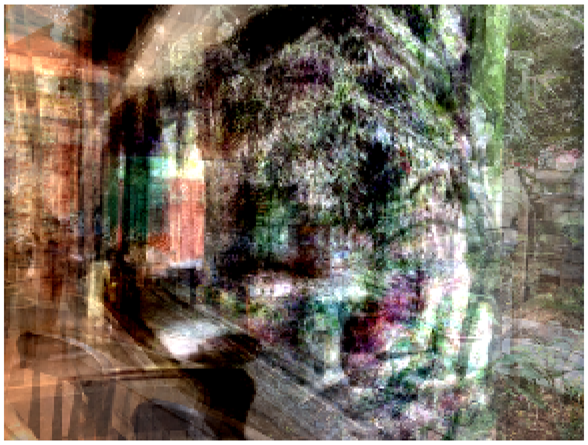
We can also reconstruct the transformed images to see how interesting they would look.

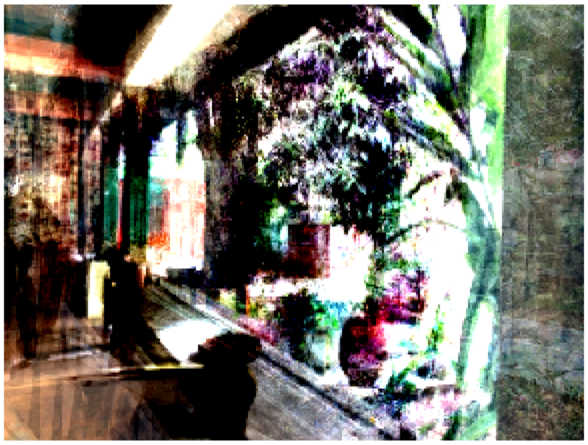
```
# Show 5 transformed images
for image in range(5):
    plt.imshow((np.asarray(plot_img_from_arr(pca_inverse[image*4]))),aspect='auto')
    plt.axis('off')
    plt.show()
```

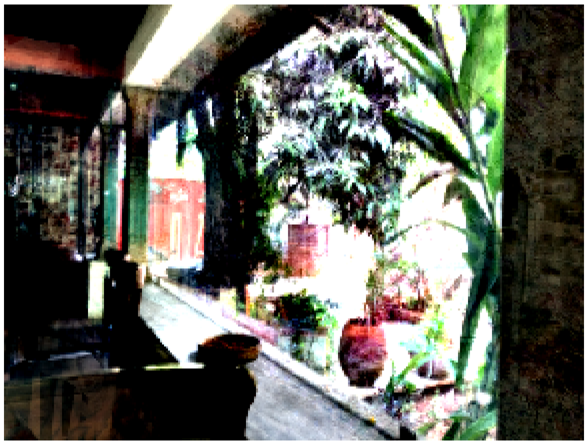


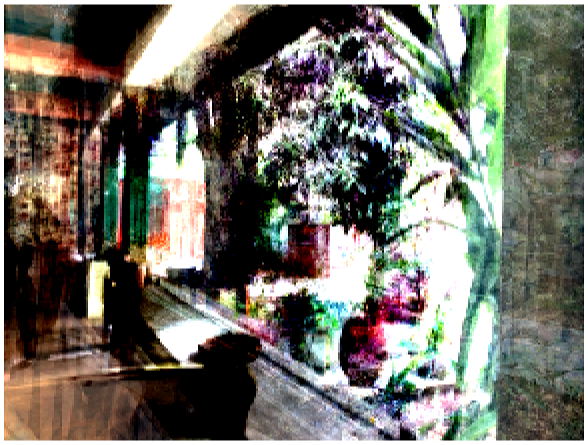
The cool part of this algorithm is that we can also come up with two random numbers and reconstruct an image using the fitted PCA.

```
from random import randint

random_points = [] # list of random points

# Generate 5 sets of random PCA data and show the image
for i in range(5):
    random_x = randint(-100000,100000)
    random_y = randint(-100000,100000)

    random_point = (random_x, random_y)
    random_points.append(random_point)

    random_inverse = pca.inverse_transform(random_point)

    plt.imshow((np.asarray(plot_img_from_arr(random_inverse))))
    plt.axis('off')
    plt.show()
```







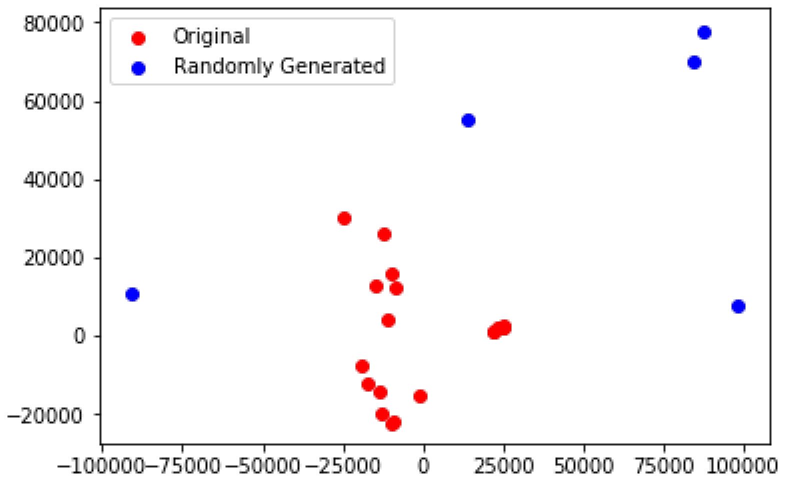


Interestingly, our reconstructed images still look similar to the original, even though they are randomly generated and far from any known location (Appendix 1). The object feature is still somewhat preserved, and we can still recognize it as the outdoor garden from the original picture, but with many visual details left out. The merit of using PCA is that we can easily recreate images from a small number of information, for example if we are trying to manipulate the sharpness and hue of the outdoor garden picture for creative art, or if we want to preprocess images to ease clustering of different human faces based on principal components even if our data is unlabelled (ie unsupervised learning).

The two visual dimensions chosen by the PCA seems to be sharpness and shadows. In other words, these dimensions are the two components that explained the largest variance in the images, and are mutually uncorrelated. We can see that there are different degrees of sharpness for each image, influenced by noise. One can observe this from the first to the fifth image on the PCA transformed images. The second visual feature is the different shadows from each image. Our image tends to have many bright and dark spots, and the fifth image from the above randomly generated image reproduces this quite well. On the other hand, the first randomly generated image did not.

Appendix 1: Visualizing the points that are far away from any known location

```
# Plot original image data and randomly generated data
fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(*zip(*pca_data),color='r',label='Original')
ax1.scatter(*zip(*random_points),color='b',label='Randomly Generated')
plt.legend(loc='best')
plt.show()
```


Original
Randomly Generated