CS156 - Location Based Assignment

December 12, 2017

For this location based assignment I chose to take pictures of a rail that was just outside Minerva's residence in Seoul. I decided to choose this item since its shape varied based on Minerva students crossing the road; it is made out of plastic and would break every once in a while since students would try to skip over it. I took pictures during different times of the day, and from different angles. This variation between different pictures might have made the task for PCA the more complicated, since the scope of the data (and itd variance) that PCA should project onto 2D is bigger.

In [37]:
```
# coding: utf-8
#156LBA.py
%matplotlib inline

"""
Notice: before running the whole code, delete everything under folders
"resized_LBA_pics/100" and "resized_LBA_pics/300".
"""

from PIL import Image
import PIL.ImageOps

# from collections import defaultdict
from glob import glob
import numpy as np
import pandas as pd

from sklearn.decomposition import PCA
# from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

#Capture all relevant files in folder:
PATH_TO_PICS = glob("/Users/oba2311/Desktop/Minerva/Junior/CS156/LBA/LBA_images/*")

# The pics are originally 4032E3024 (taken with Samsung S8sams) , I'll shrink them to
# and 100 X 100.
```

In [38]:
```
from scipy import misc
def img_to_matrix(filename):
```

```
    """
    takes a filename and turns it into a numpy matrix.
    """
    return misc.imread(filename)
```

In [39]:
```
long_labels = []
import os
for filename in PATH_TO_PICS:
    drive, path = os.path.splitdrive(filename)
    path, filename = os.path.split(filename)
    long_labels.append(filename)
```

In [40]:
```
size_100 = (100, 100)
size_300 = (300, 300)

THREE_HUND_RESU = glob("/Users/oba2311/Desktop/Minerva/Junior/CS156/resized_LBA_pics/30
HUND_RESU = glob("/Users/oba2311/Desktop/Minerva/Junior/CS156/resized_LBA_pics/100?*")

def image_resize_save(f,label):
    i = Image.open(f)
    fn, fext = os.path.splitext(f)
    i.thumbnail(size_300)
    i.save("/Users/oba2311/Desktop/Minerva/Junior/CS156/resized_LBA_pics/300/300_{}".fo
    i.thumbnail(size_100)
    i.save("/Users/oba2311/Desktop/Minerva/Junior/CS156/resized_LBA_pics/100/100_{}".fo
```

In [41]:
```
saved_images = []
for filename, label in zip(PATH_TO_PICS, long_labels):
    saved_file = (image_resize_save(filename, label),filename)
    saved_images.append(filename)
```

In [42]:
```
#Data:
raw_data = [(img_to_matrix(filename),filename) for filename in THREE_HUND_RESU]
```

In [73]:
```
data = np.array([datum for (datum,f) in raw_data])
#squeeze the RGB and width+length into an array to project onto 2D:
data = data.reshape(29,-1)
show_data = pd.DataFrame(data)
#Let's make sure that the data looks like we expect it to look -
#a very long array per image (3RGB dimensions X the width(225) X the length(300) pixels
show_data[:1]
```

Out[73]:

	0	1	2	3	4	5	6	7	8	\
0	49	85	49	53	84	52	56	80	56	

	9	...	202490	202491	202492	202493	202494	202495	202496	\
0	58	...	52	74	53	52	74	53	52	

	202497	202498	202499

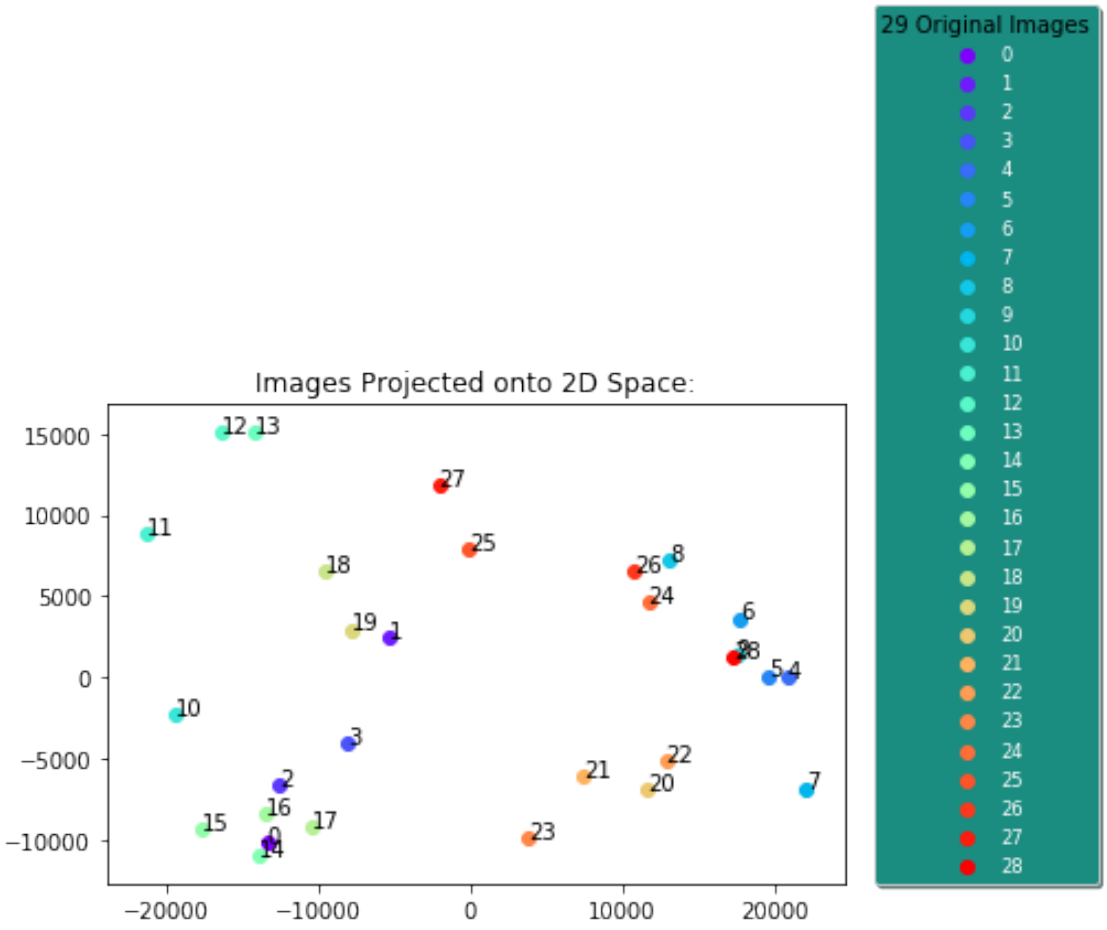
```
          0      74      53      52

          [1 rows x 202500 columns]
```

In [44]:
```
          #PCA:
          """
          PCA computes eigenvectors of the covariance matrix ("principal axes") and sorts them by
          (amount of explained variance). The centered data can then be projected onto these
          principal axes to yield principal components ("scores").
          For the purposes of dimensionality reduction, one can keep only a subset of principal c
          discard the rest, hoping to capture the essence of the data.
          """
          # find the principal components:
          N_COMPONENTS = 2
          X=[]

          pca = PCA(n_components=N_COMPONENTS)
          X = pca.fit_transform(data)
          print X
```

```
[[ -1.34265441e+04  -1.01565249e+04]
 [ -5.36210678e+03   2.50653063e+03]
 [ -1.26163507e+04  -6.65732763e+03]
 [ -8.08426129e+03  -3.99161739e+03]
 [  2.08627987e+04   1.58860678e+01]
 [  1.95673140e+04   1.06362501e+02]
 [  1.77480508e+04   3.63445036e+03]
 [  2.20949812e+04  -6.81082046e+03]
 [  1.30994302e+04   7.24411495e+03]
 [  1.75097629e+04   1.38047062e+03]
 [ -1.94341285e+04  -2.27831080e+03]
 [ -2.13345033e+04   8.84075729e+03]
 [ -1.64257035e+04   1.51414273e+04]
 [ -1.42707538e+04   1.50823930e+04]
 [ -1.39387810e+04  -1.09516824e+04]
 [ -1.77486255e+04  -9.27316254e+03]
 [ -1.34962880e+04  -8.36191971e+03]
 [ -1.04937756e+04  -9.19676705e+03]
 [ -9.62566957e+03   6.52325190e+03]
 [ -7.90124788e+03   2.95312449e+03]
 [  1.16441145e+04  -6.91307160e+03]
 [  7.39752453e+03  -6.06585434e+03]
 [  1.28547198e+04  -5.10971133e+03]
 [  3.78129688e+03  -9.85043949e+03]
 [  1.16981431e+04   4.65049364e+03]
 [ -8.94769428e+01   7.91406631e+03]
 [  1.07482765e+04   6.56074129e+03]
 [ -2.10278113e+03   1.17942303e+04]
```

```
 [  1.73445845e+04   1.26890904e+03]]
```

In [45]:
```
pc_1 = [(x1) for x1,x2 in X]
pc_2 = [(x2) for x1,x2 in X]

# For 5 components:

# pc_1 = [(x1) for x1,x2,x3,x4,x5 in X]
# pc_2 = [(x2) for x1,x2,x3,x4,x5 in X]
# pc_3 = [(x2) for x1,x2,x3,x4,x5 in X]
# pc_4 = [(x2) for x1,x2,x3,x4,x5 in X]
# pc_5 = [(x2) for x1,x2,x3,x4,x5 in X]

fig, ax = plt.subplots()
short_labels = [i for i in range(len(X[:,]))]


# Set colors:
import matplotlib.cm as cm
colors = cm.rainbow(np.linspace(0, 1, len(pc_1)))

for i in range(len(short_labels)):
    ax.scatter(pc_1[i],pc_2[i],c=colors[i], label=short_labels[i])
    ax.annotate(i, (pc_1[i],pc_2[i]))

#Set legend:
legend = ax.legend(loc=(1.04,0), shadow=True, fontsize='small', title = "29 Original Im
# Put a nicer background color on the legend.
plt.setp(legend.get_texts(), color='w')
legend.get_frame().set_facecolor((0.1, 0.55, 0.5))
plt.title("Images Projected onto 2D Space:")
plt.show()
```



In [46]:
```
# To what level does each component explains that data:
print pca.explained_variance_ratio_


#Use inverse transformation on outliers to see the reconstruction:
#for example, let's take images #7 ,#12 , which are further from the cluster.

labels_dict = {}
for i in short_labels:
    labels_dict[i] = long_labels[i]

print "**Outliers:**" ,labels_dict[7] , labels_dict[12]
```

```
[ 0.29417399  0.08795259]
**Outliers:** 20171101_132541.jpg 20171102_205342.jpg
```

"7"

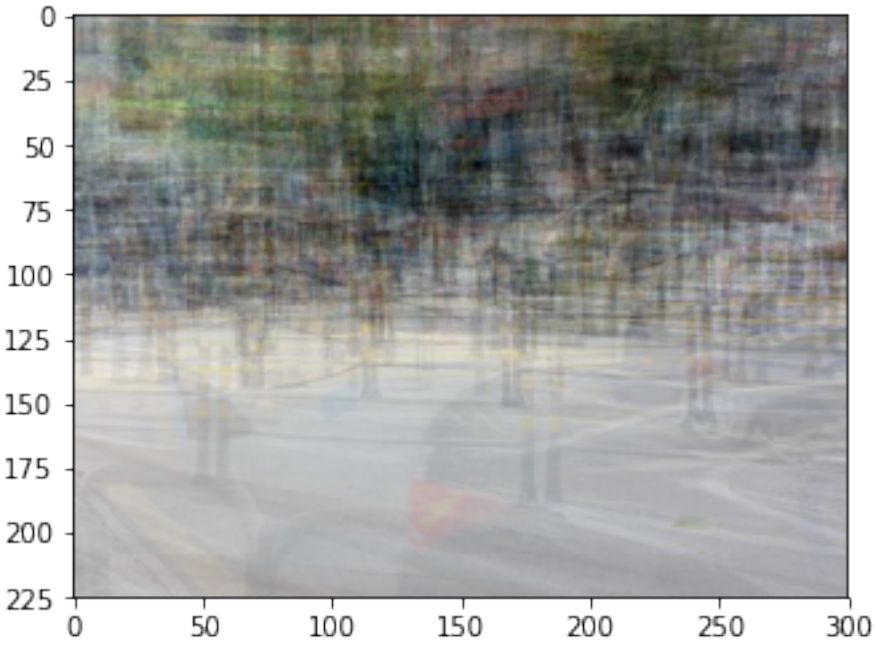
0.0.1 The original image - image 7:

0.0.2 The Reconstructions:

```
In [47]: print "Outlier Before reconstruction:", X[7]

         data_inverse = pca.inverse_transform(X) #6 for 7th image.
         data_inverse = np.clip(data_inverse, 0, 255).astype('uint8')

         # for i in data_inverse:
         image = data_inverse[7]
         print image
         image = image.reshape(225,300,3)
         im = Image.fromarray(image,"RGB")
         plt.figure()
         plt.imshow(im)
```

```
Outlier Before reconstruction: [ 22094.98121012  -6810.8204582 ]
[140 151 148 ..., 167 166 171]
```

```
Out[47]: <matplotlib.image.AxesImage at 0x1a20deac10>
```



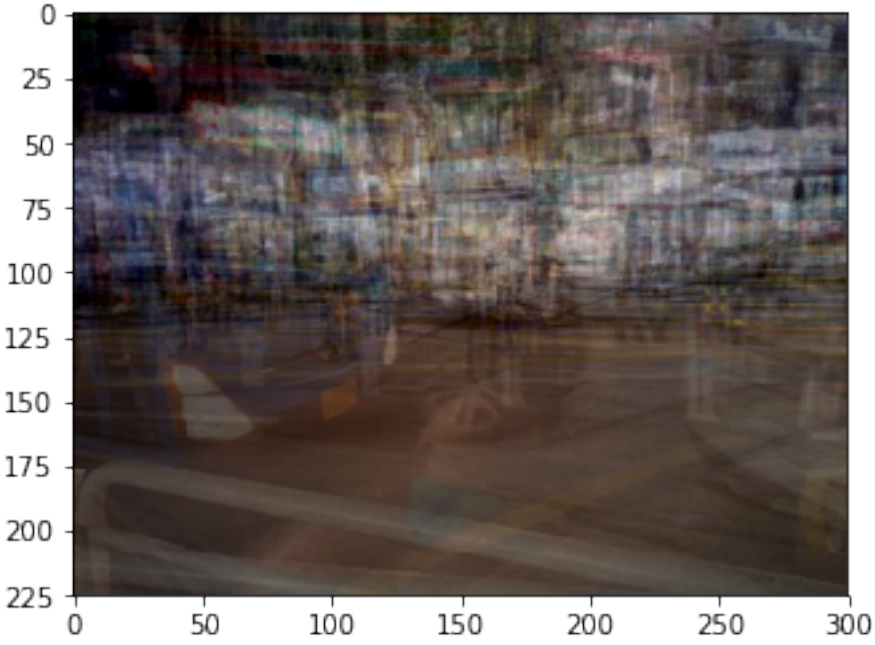
0.0.3 Another Outlier, and Reconstruction - 12:

```
In [51]: print "Outlier Before reconstruction:", X[12]

         data_inverse = pca.inverse_transform(X)
         data_inverse = np.clip(data_inverse, 0, 255).astype('uint8')

         # for i in data_inverse:
         print data_inverse[12]
         image = data_inverse[12]
         print image
         image = image.reshape(225,300,3)
         im = Image.fromarray(image,"RGB")
         plt.figure()
         plt.imshow(im)
```

```
Outlier Before reconstruction: [-16425.7035059   15141.42733493]
(202500,)
[34 29 26 ..., 54 49 39]
```

```
Out[51]: <matplotlib.image.AxesImage at 0x1a2083fd10>
```

"12"



One can see that the reconstructions are indeed a composition of the reconstructed image after projection onton 2D combined with the learning of the PCA algorithm. Technically, what we did is to take the component that best describes the variance of the data, and his orthogonals (remembering that in high-dimensional spaces that are there are many directions which are orthogonal to each other).

Mathemathically, with each one of thos dimensions used, an Eigenvalue is associated. The non-negative eigenvalues are monotonically non-increasing starting from the first component, so we can choose the best ones based on those eigenvalues.