CS156 - LBA

Akmarzhan Abylay

October 2020

1 Location Based Assignment

For this LBA I made around 30 photos of the view from our apartment in London. They were taken from two different angles during different times of the day as I couldn't align the pictures properly, which is why they are not identical.

1.1 Step 1: Processing

Below I loaded my photos and processed all of them down to a size of 512 by 512.

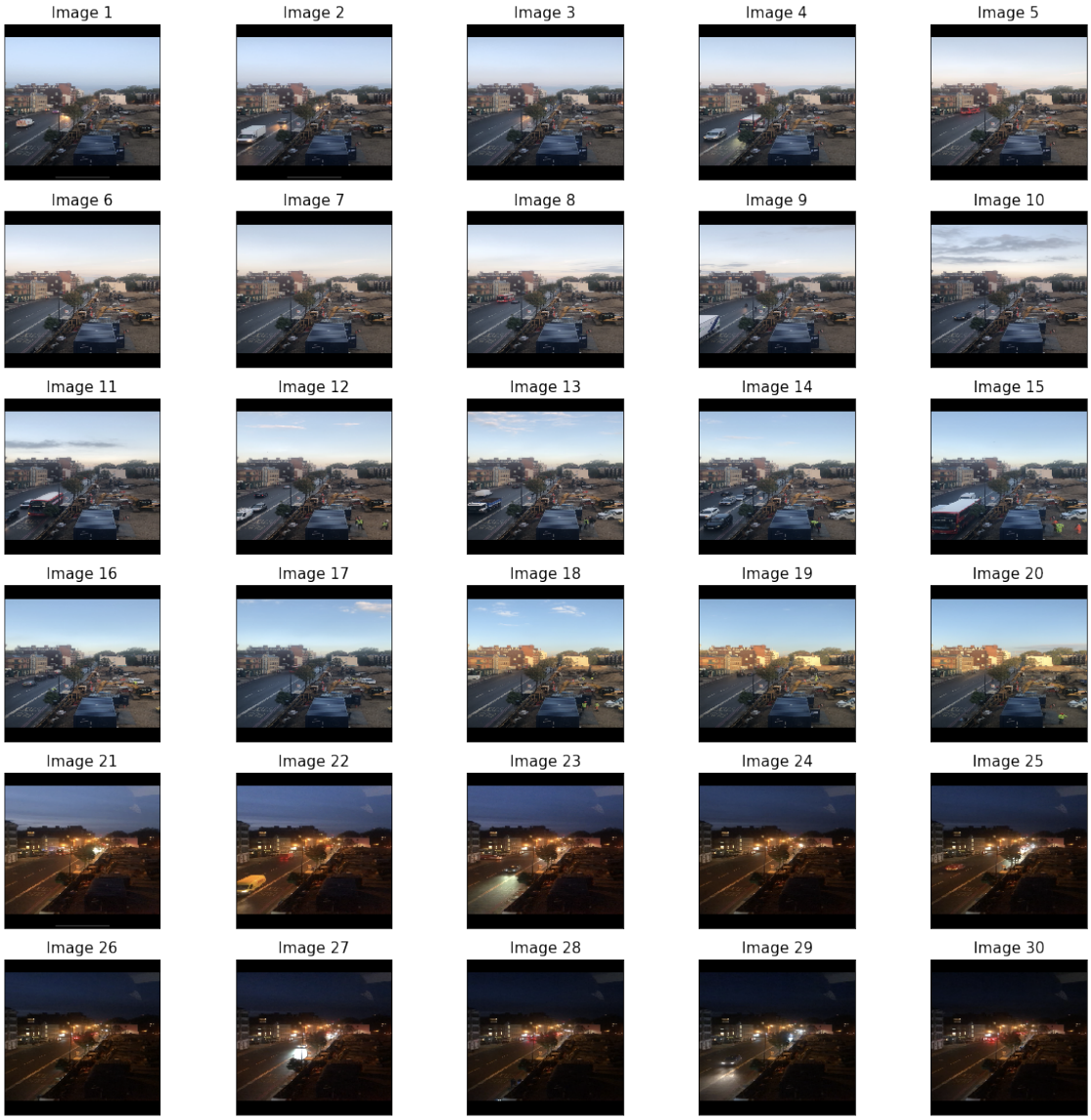
[1]:
```
from glob import glob

london = sorted(glob('london/*')) #loading the photos
```

[2]:
```
from PIL import Image
import numpy as np

width, height = 512, 512

def image_processing(image, width, height): #processing all of the images
    img = Image.open(image)
    pic = np.array(img.resize((width, height)))
    return pic.flatten()

#storing the processed images
data = [image_processing(image, width, height) for image in london]
```

The original pictures look like the following.

[3]:
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(6, 5, figsize=(20, 20))

for i,ax in enumerate(axes.flatten()): #plotting the pictures
    ax.set_title(f'Image {i+1}', fontsize = 15)
    ax.imshow(data[i].reshape(512, 512, 4)) #width, height by a 4 channel image
    ax.set_xticks(())
    ax.set_yticks(())
```

```
plt.show()
```



1.2 Step 2: PCA

Using principal components analysis (PCA), I projected my images down to a 2 dimensional representation. I found out that the explained variance ratio is around 94%, which means that most of the variance is explained by these two selected components.

[4]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
```

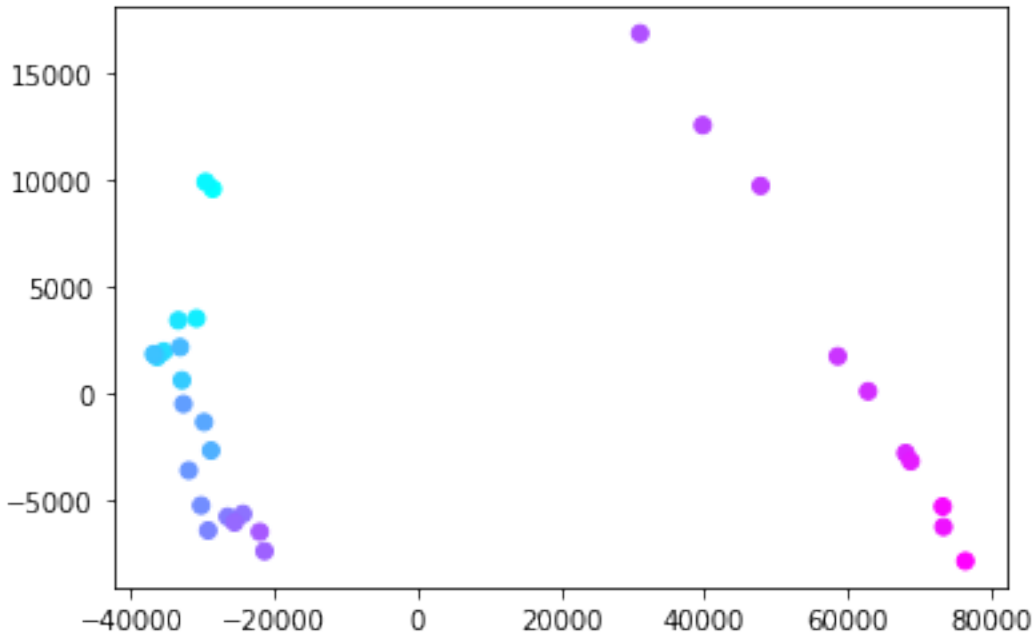
```
projections = pca.fit_transform(data)
print("The explained variance ratio is", np.sum(pca.explained_variance_ratio_[:
 ↪2]))
```

```
The explained variance ratio is 0.9396942561558718
```

1.3 Step 3: 2D scatter

In this step, I visually inspected the 2D locations of each photo in the new space. We can see how all the photos are scattered.

[5]:
```
plt.scatter([point[0] for point in projections], [point[1] for point in␣
 ↪projections],
            cmap = "cool", c = range(len(projections)))
plt.show()
```

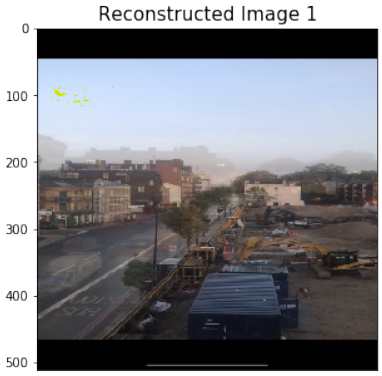


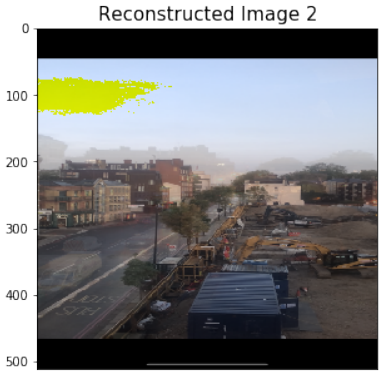
We see that most of the pictures are nearby, but there are two different "clusters". We can see from the color that blue represents the earlier photos (during the day), while pink represents the later ones (night). Also, they were taken during different times (daytime and evening), which is why they might be a little different.

1.4 Step 4: Reconstruction

Below is the reconstruction of each low-dimensional representation. We can see that most of the pictures are very similar with an exception of a few where it couldn't catch the darkness or the clouds.

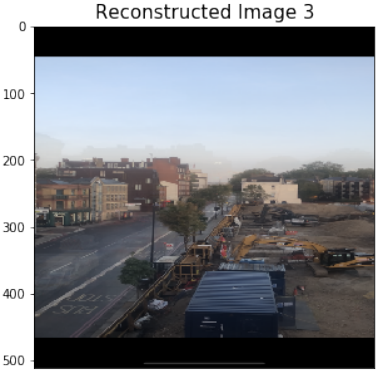
[6]:
```
import warnings
warnings.filterwarnings("ignore")

reconstruction = pca.inverse_transform(projections)

for i in range(len(london)):
    fig, axes = plt.subplots(1, 2, figsize=(20, 5))
    axes[0].set_title(f'Real Image {i+1}', fontsize = 15)
    axes[1].set_title(f'Reconstructed Image {i+1}', fontsize = 15)
    axes[0].imshow(data[i].reshape(512, 512, 4))
    axes[1].imshow(reconstruction[i].reshape(512, 512, 4).astype('uint8'))
    axes[0].set_yticks(())
    axes[1].set_xticks(())
```





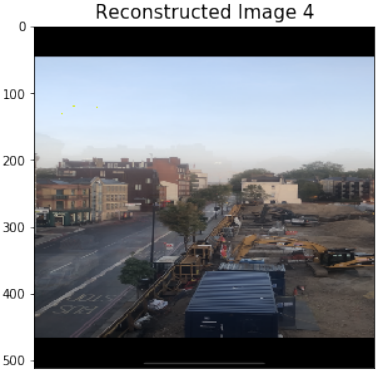
Real Image 3

Reconstructed Image 3



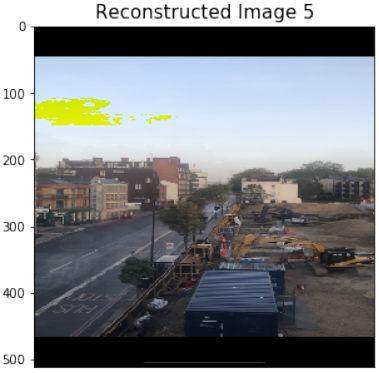
Real Image 4

Reconstructed Image 4

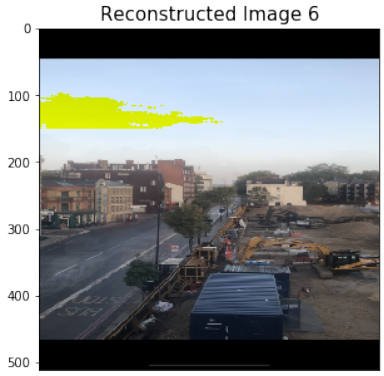


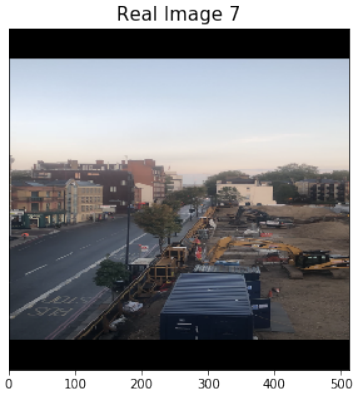
Real Image 5

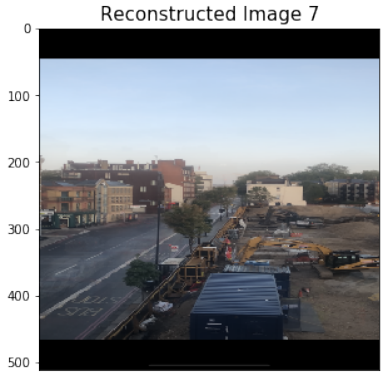
Reconstructed Image 5



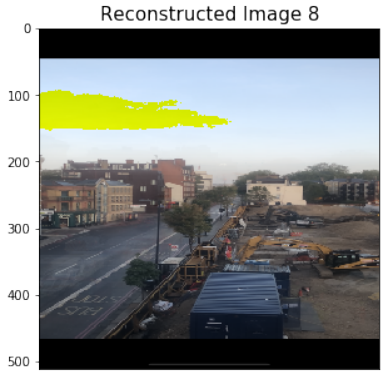

Real Image 6
0
100
200
300
400
500


Reconstructed Image 6
0
100
200
300
400
500

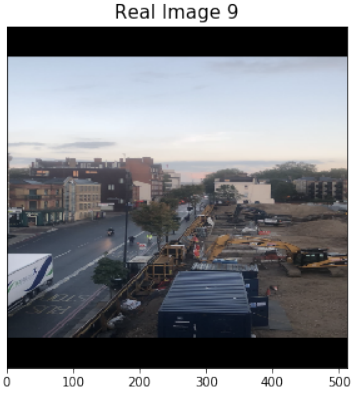

Real Image 7
0
100
200
300
400
500


Reconstructed Image 7
0
100
200
300
400
500

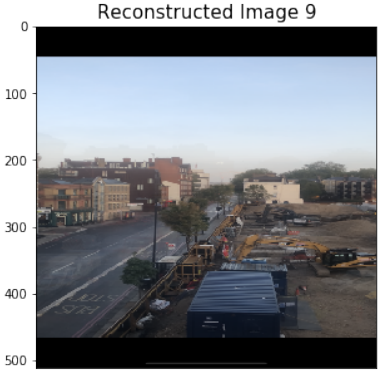

Real Image 8
0
100
200
300
400
500


Reconstructed Image 8
0
100
200
300
400
500

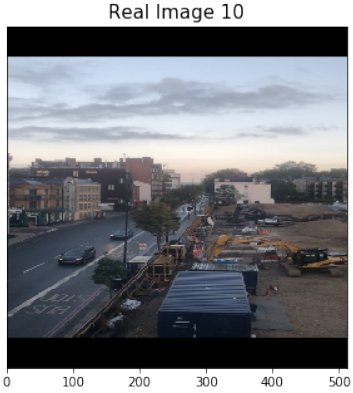
Real Image 9



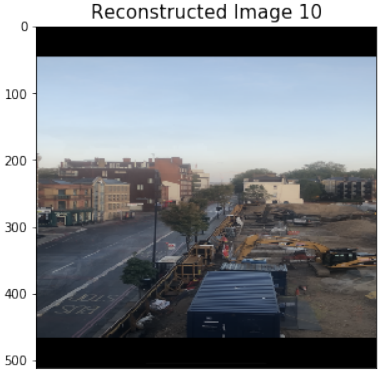
Reconstructed Image 9



Real Image 10

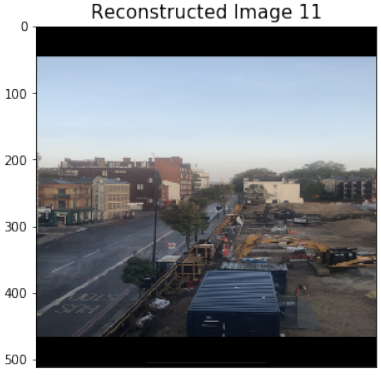


Reconstructed Image 10

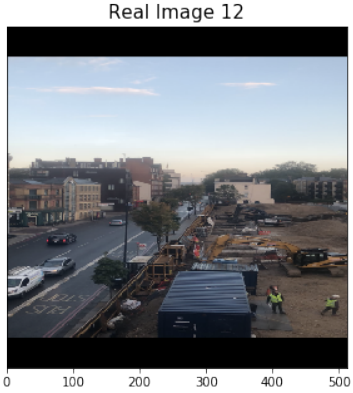


Real Image 11

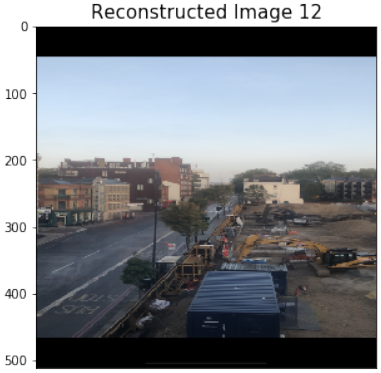
Reconstructed Image 11



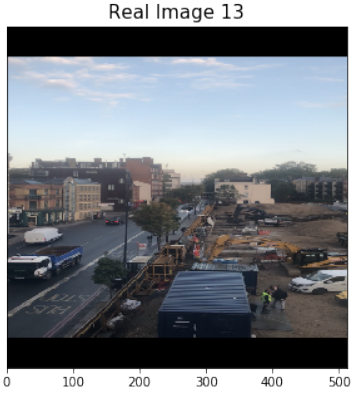
Real Image 12



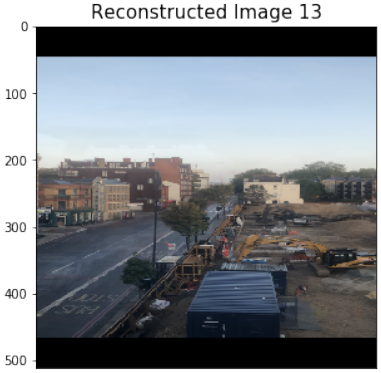
Reconstructed Image 12



Real Image 13

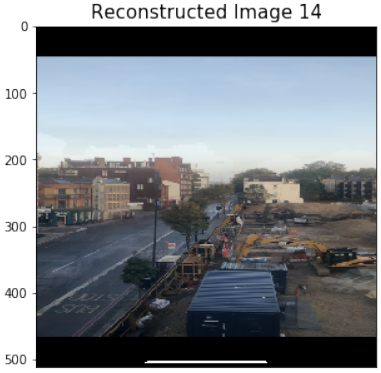


Reconstructed Image 13



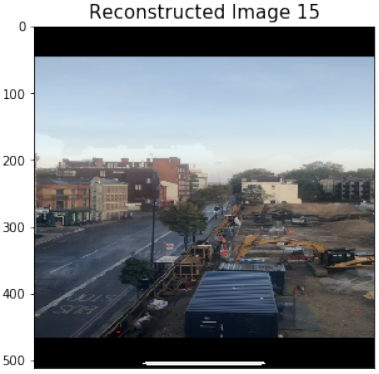
Real Image 14

Reconstructed Image 14



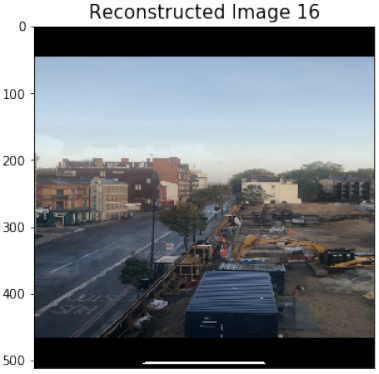
Real Image 15

Reconstructed Image 15



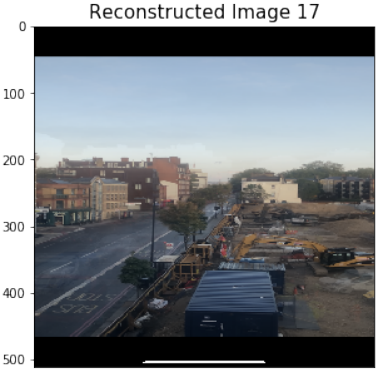
Real Image 16

Reconstructed Image 16



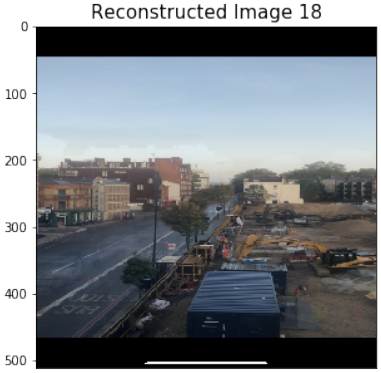
Real Image 17

Reconstructed Image 17



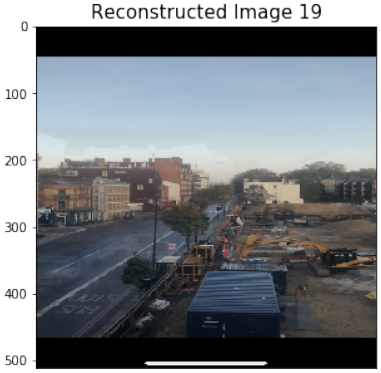
Real Image 18

Reconstructed Image 18



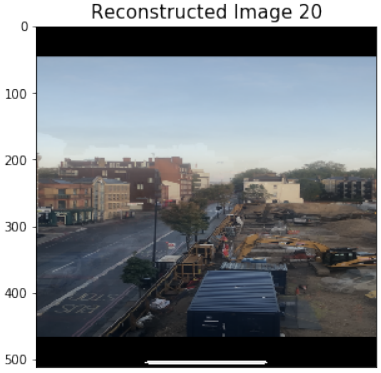
Real Image 19

Reconstructed Image 19

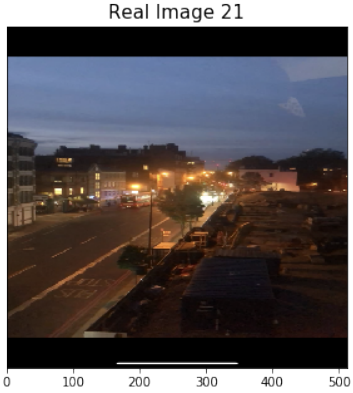


Real Image 20

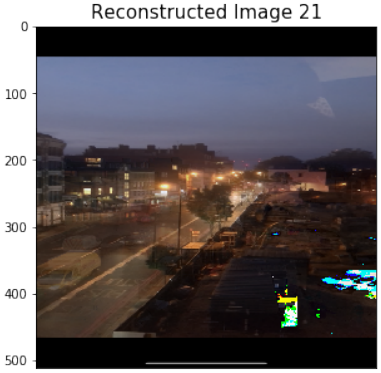
Reconstructed Image 20



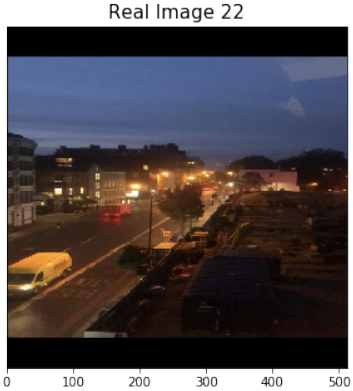
Real Image 21



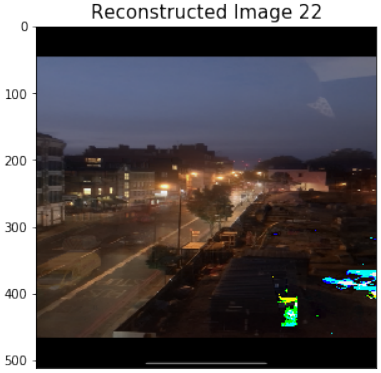
Reconstructed Image 21



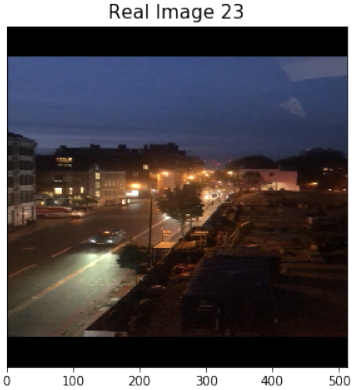
Real Image 22



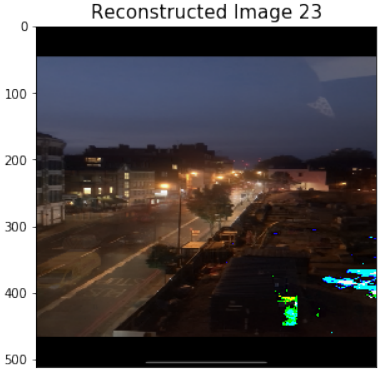
Reconstructed Image 22



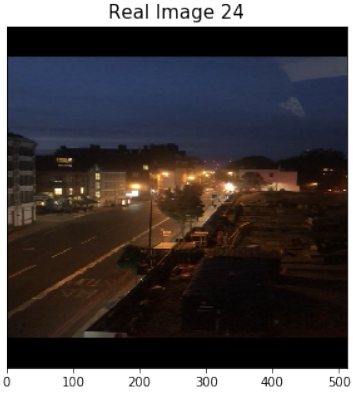
Real Image 23



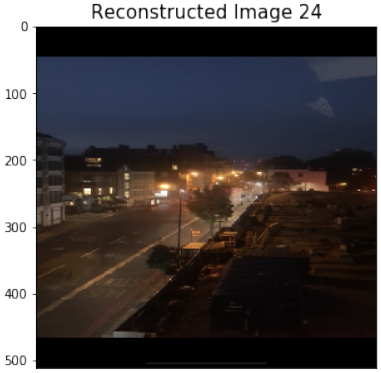
Reconstructed Image 23



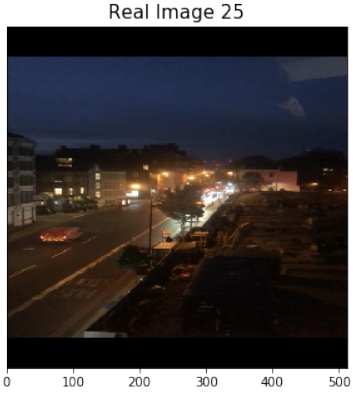
Real Image 24



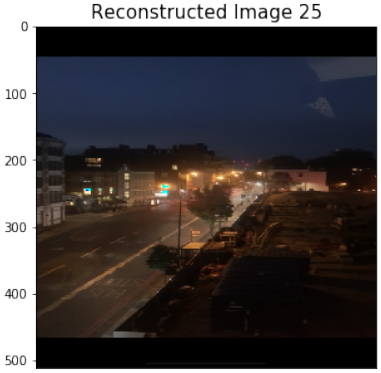
Reconstructed Image 24



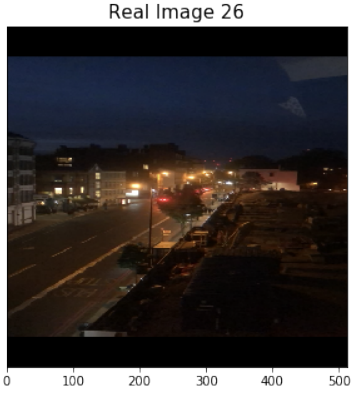
Real Image 25



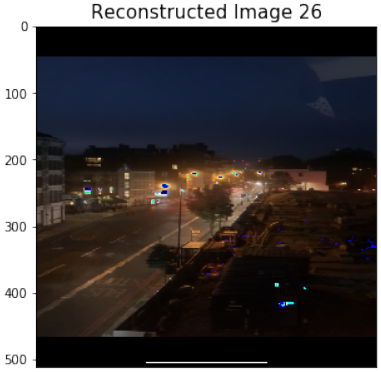
Reconstructed Image 25



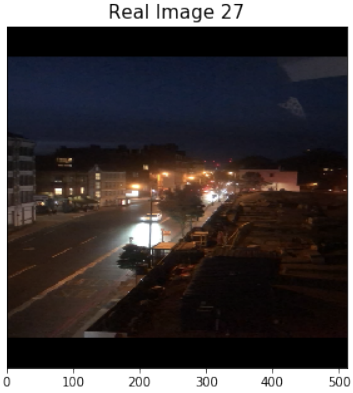
Real Image 26



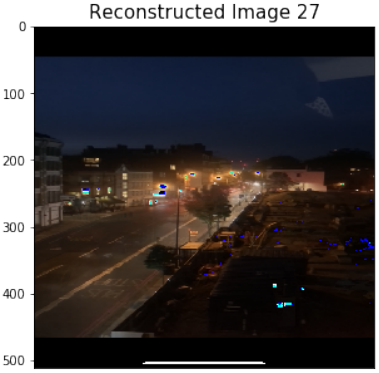
Reconstructed Image 26



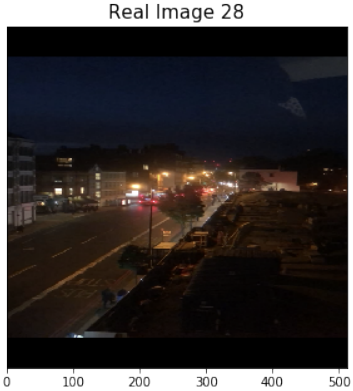
Real Image 27



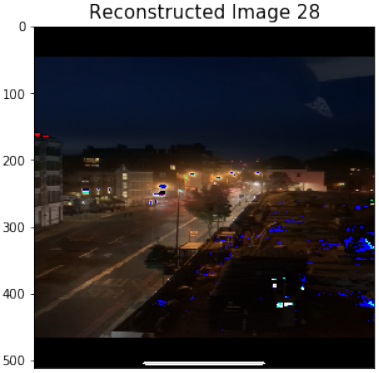
Reconstructed Image 27



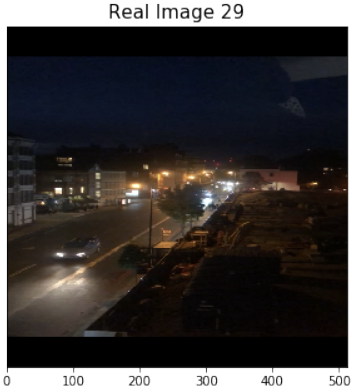
Real Image 28



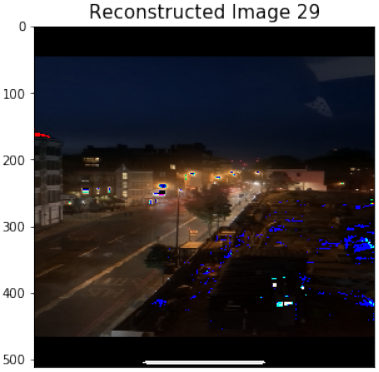
Reconstructed Image 28

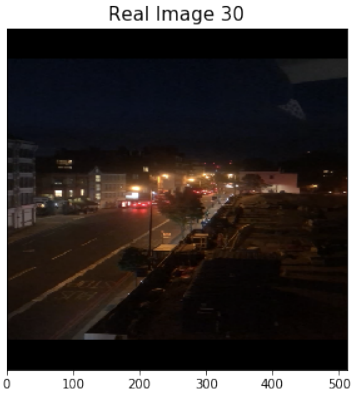


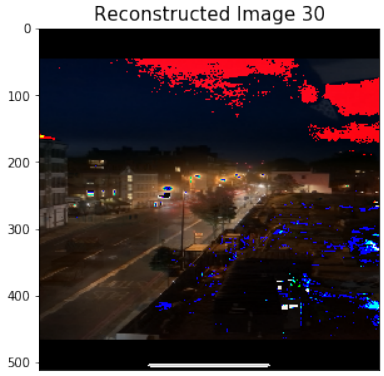
Real Image 29



Reconstructed Image 29



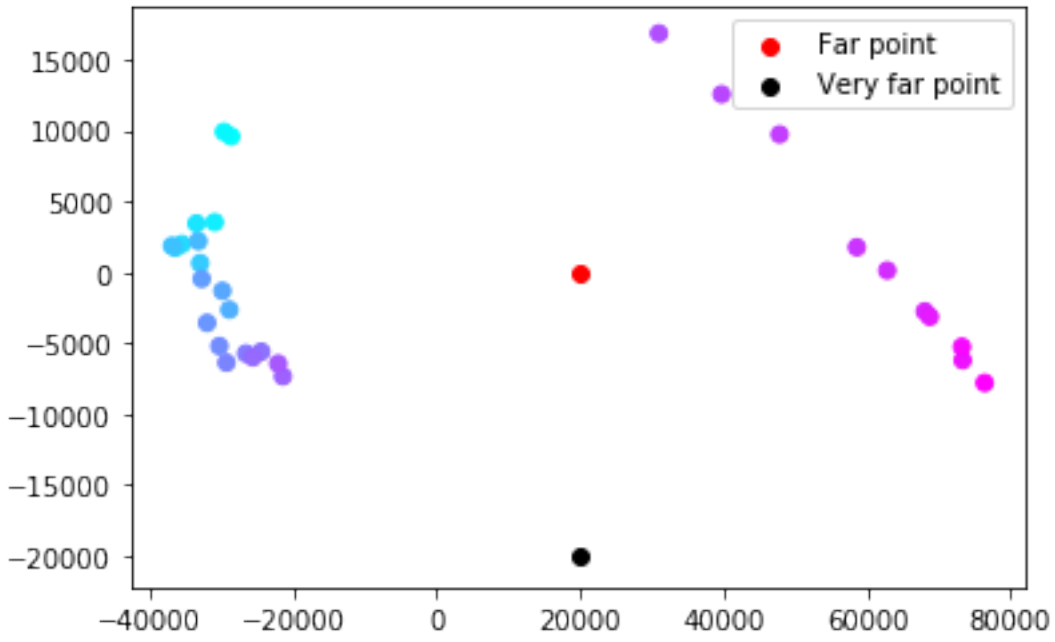




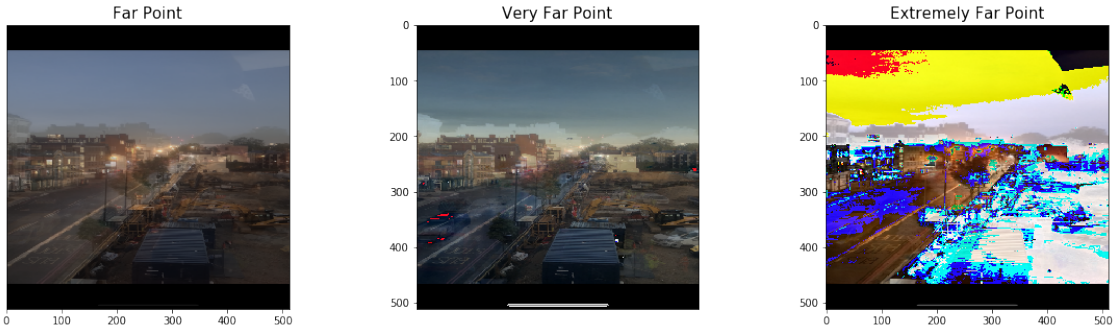
1.5 Step 5: New Point

Finally, I picked two points that were far away from any known location (i.e., (20000, 0) and (20000, -20000)). I also plotted their reconstruction.

[7]:
```
plt.scatter([point[0] for point in projections], [point[1] for point in␣
 ↪projections],
            cmap = "cool", c=range(len(projections)))
plt.scatter(20000, 0, c = 'red', label = "Far point")
plt.scatter(20000, -20000, c = 'black', label = "Very far point")
plt.legend(loc = "best")
plt.show()
```



[8]:
```
far = pca.inverse_transform([20000, 0])
far_plus = pca.inverse_transform([20000, -20000])
extremely_far = pca.inverse_transform([100000, 10000])

fig, axes = plt.subplots(1, 3, figsize=(20,5))
axes[0].set_title(f'Far Point', fontsize = 15)
axes[1].set_title(f'Very Far Point', fontsize = 15)
axes[2].set_title(f'Extremely Far Point', fontsize = 15)
axes[0].imshow(far.reshape(512, 512, 4).astype('uint8'))
axes[1].imshow(far_plus.reshape(512, 512, 4).astype('uint8'))
axes[2].imshow(extremely_far.reshape(512, 512, 4).astype('uint8'))
axes[0].set_yticks(())
axes[1].set_xticks(())
plt.show()
```



As we can see from the reconstructions above, the first far point looks ok. It has this overlap because I used different angles for the photos but otherwise, the colors are fine. The second furthest point looks normal too, although you can see the difference in the photo angles. I also plotted an extremely far point, out of curiosity, and it turned out even worse than the previous one. In conclusion, we can say that the PCA did pretty well, but it is not so applicable to the points further from the original data.