Location-based assignment

This is a location-based assignment in San Francisco. I chose an interesting symbol in front of the place I usually pick up lunch from.

1. Resize image to 512 pixels

In [ ]:
```
# Import libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from glob import glob
from PIL import Image
from resizeimage import resizeimage

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [ ]:
```
photos = glob('*.jpeg')
PHOTO_SIZE = 512
```

In [ ]:
```
photos_flattened = []
# for each image path
for path in photos:
    # open it as a read file in binary mode
    with open(path, 'r+b') as f:
        # open it as an image
        with Image.open(f) as image:
            # resize the image to be more manageable
            cover = resizeimage.resize_cover(image, [PHOTO_SIZE, PHOTO_SIZE])
            # flatten the matrix to an array and append it to all flattened images
            photos_flattened.append(np.array(cover).flatten())
```

In [ ]:
```
# Flatten it once more
photos_flattened = np.asarray(photos_flattened)
```

In [ ]:
```
X = np.stack(i for i in photos_flattened)
```

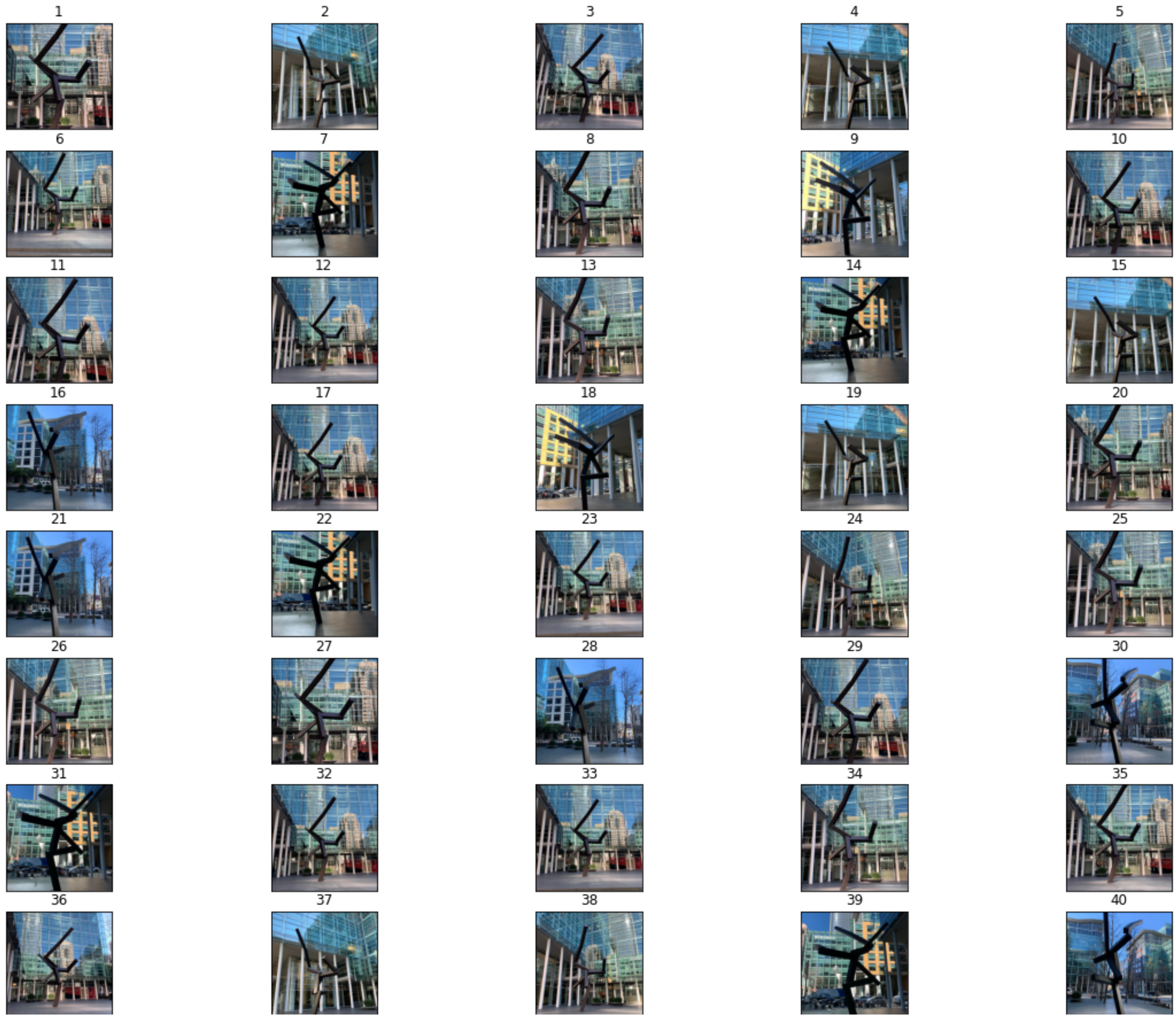
```
/home/khanh/Documents/cs156-pcw-khanhtmn/venv/lib/python3.8/site-packages/IPython/core/interactiveshell.p
y:3191: FutureWarning: arrays to stack must be passed as a "sequence" type such as list or tuple. Support
for non-sequence iterables such as generators is deprecated as of NumPy 1.16 and will raise an error in th
e future.
  if await self.run_code(code, result, async_=asy):
```

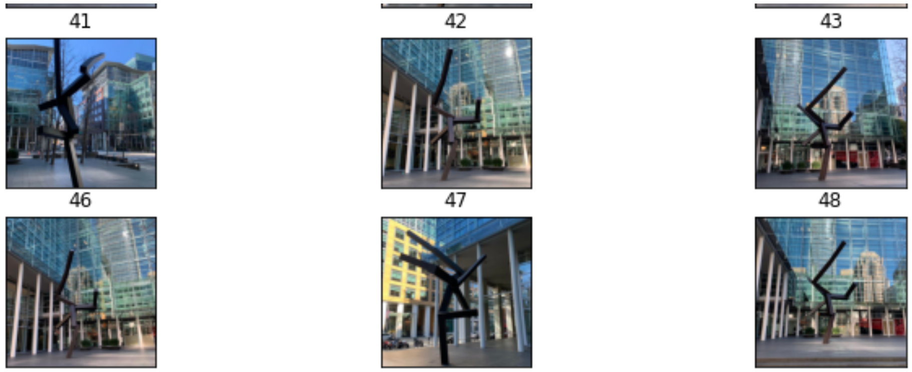
In [ ]:
```
X.shape
```

Out[ ]:
```
(48, 786432)
```

Let's take a look at the original photos. I took them at different angles!

In [ ]:
```
# Original photos

nrows = len(X)//5 + 1
ncols = 5

plt.figure(figsize=[20, 20])
plt.title("Original photos")
for i in range(len(X)):
    plt.subplot(nrows, ncols, i+1, xticks=[], yticks=[])
    plt.imshow(X[i].reshape(PHOTO_SIZE, PHOTO_SIZE, 3).astype(np.uint8))
    plt.title(f"{i+1}")
plt.show()
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40



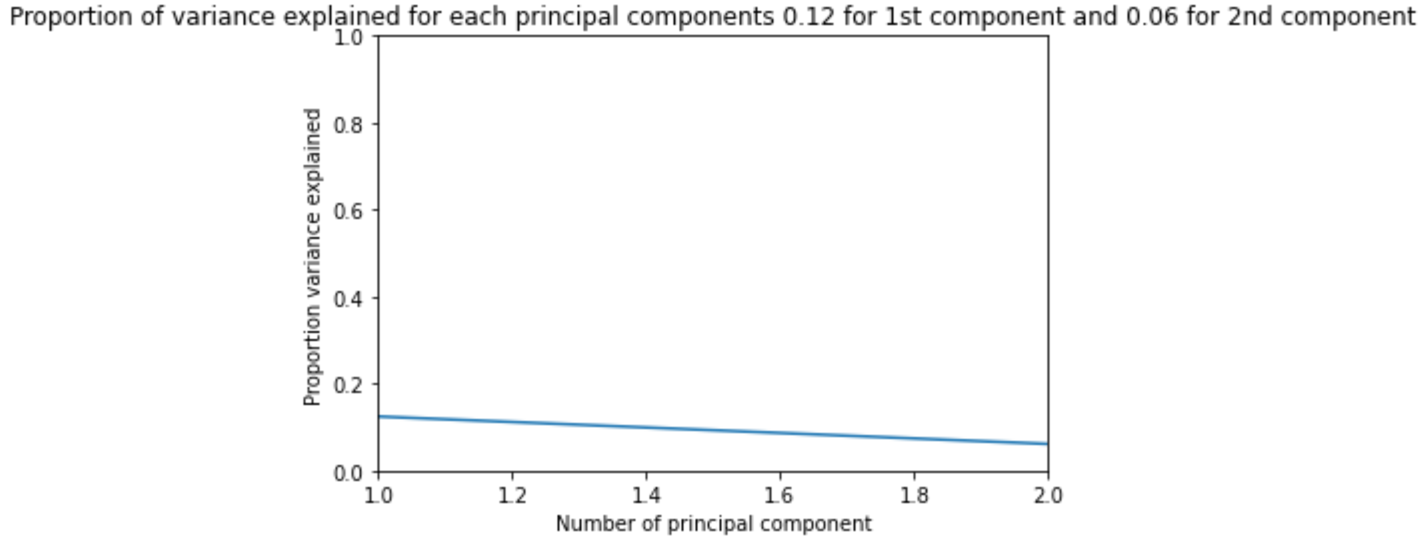
2. Use PCA to project the images down to a 2D representation

In [ ]:
```
pca = PCA(n_components=2)
pca.fit(X)
X_2d = pca.transform(X)
print(X_2d.shape)
```

```
(48, 2)
```

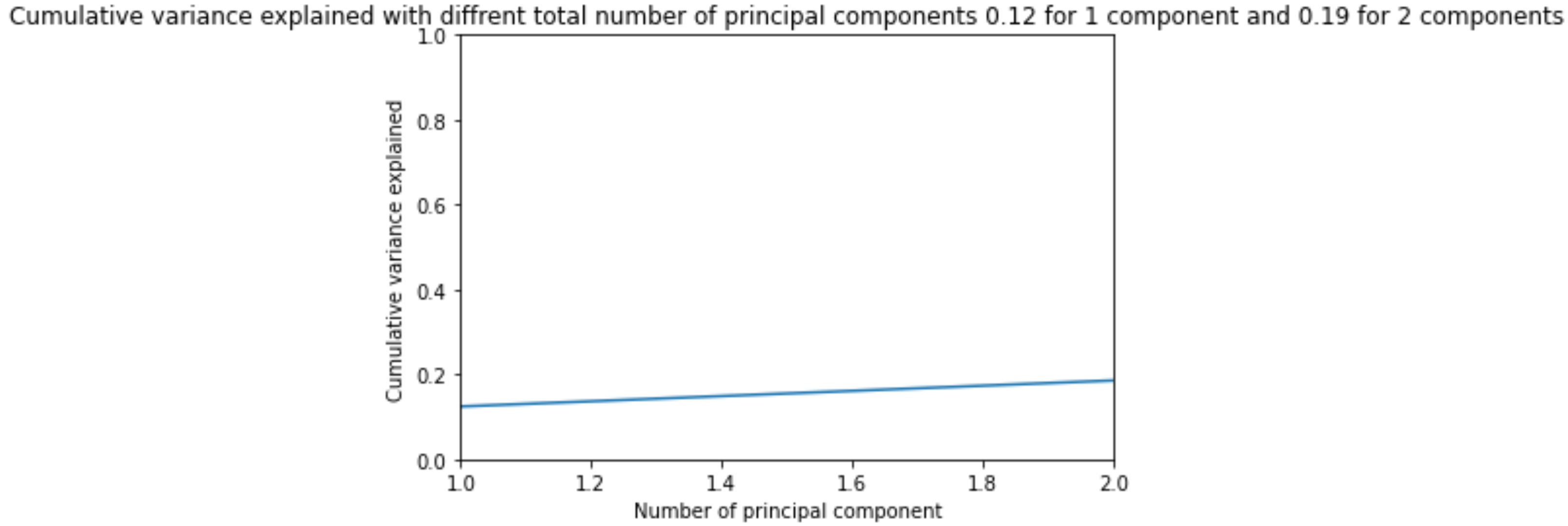
In [ ]:
```
variance_explained = pca.explained_variance_ratio_
```

In [ ]:
```
plt.plot(range(1,3,1), variance_explained)
plt.title(f'Proportion of variance explained for each principal components {round(variance_explained[0], 2
plt.xlabel('Number of principal component')
plt.ylabel('Proportion variance explained')
plt.ylim(0, 1)
plt.xlim([1, 2])
plt.show()
```



In [ ]:
```
cum_sum_variance_explained = np.cumsum(variance_explained)
```

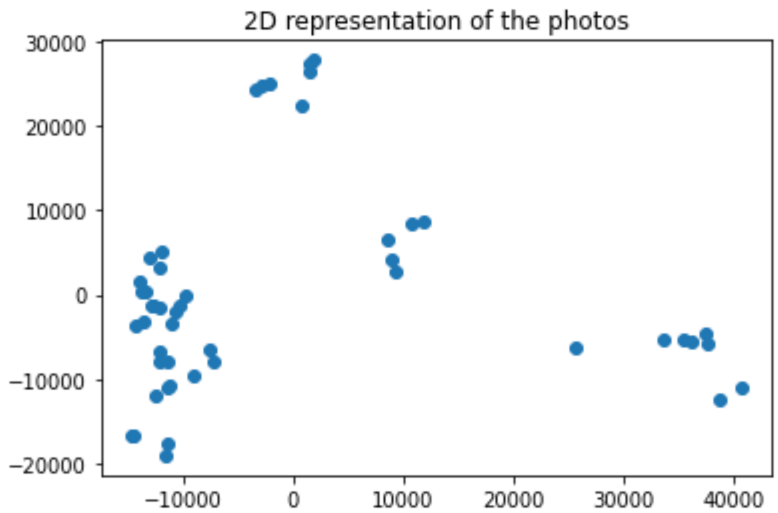
In [ ]:
```
plt.plot(range(1,3,1), cum_sum_variance_explained)
plt.title(f'Cumulative variance explained with diffrent total number of principal components {round(cum_su
plt.xlabel('Number of principal component')
plt.ylabel('Cumulative variance explained')
plt.ylim(0, 1)
plt.xlim([1, 2])
plt.show()
```



It seems like the two components together can only explain 19% variation in the data, which makes sense given that our object has a very unusual shape. In addition, the photos were taken from a lot of angles!

3. Visually inspect the 2D locations in the new space

In [ ]:
```
plt.scatter(X_2d[:,0], X_2d[:,1])
plt.title(f'2D representation of the photos')
plt.show()
```



Seems like the 2D locations cluster into 4 different groups, which were also the different angles (front, back, left, right) that I took the photos from!

We can also represent the information in a table:

In [ ]:
```
pca_df = pd.DataFrame(data=X_2d, columns=['1', '2'])
pca_df.head()
```

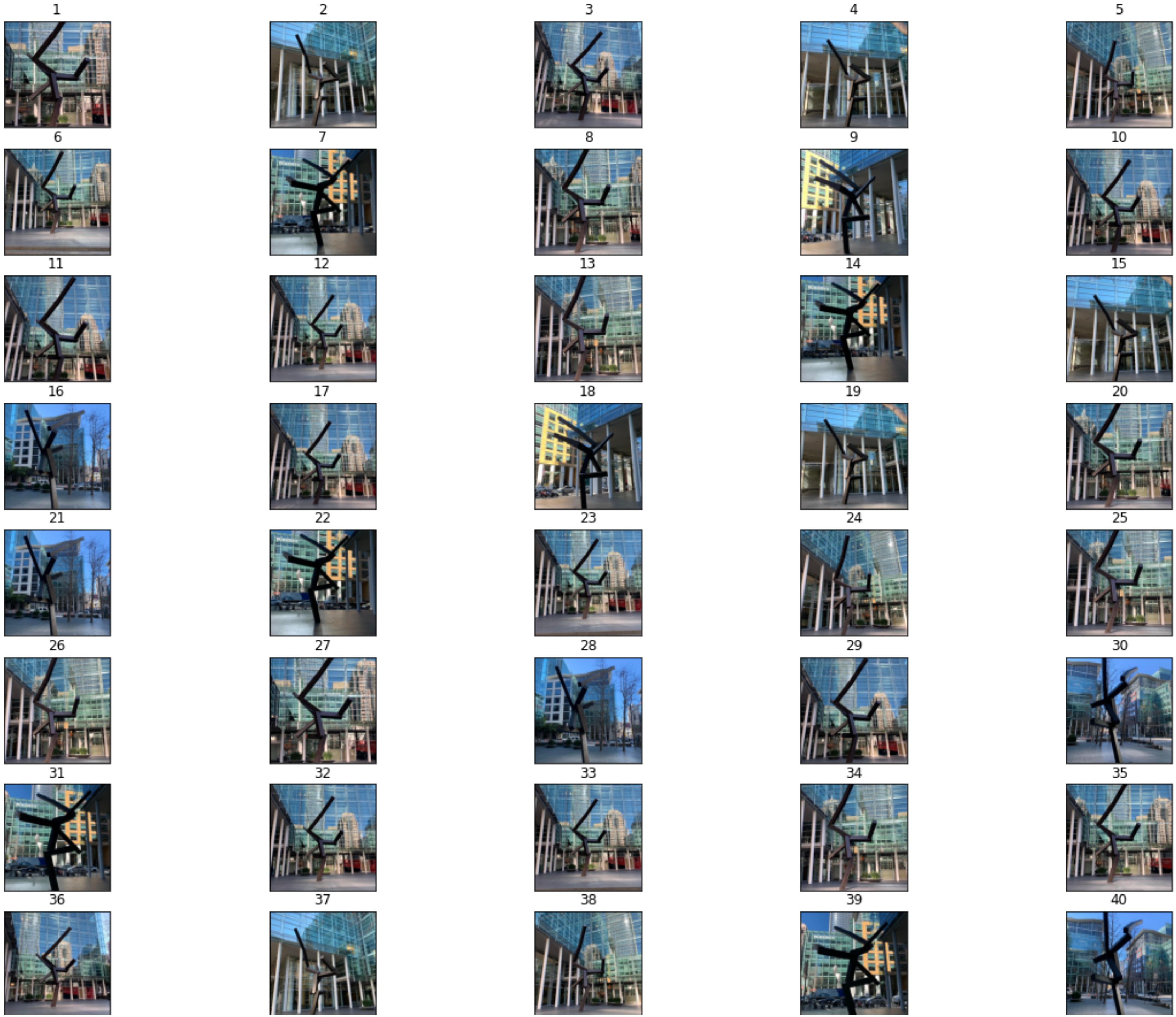
Out[ ]:

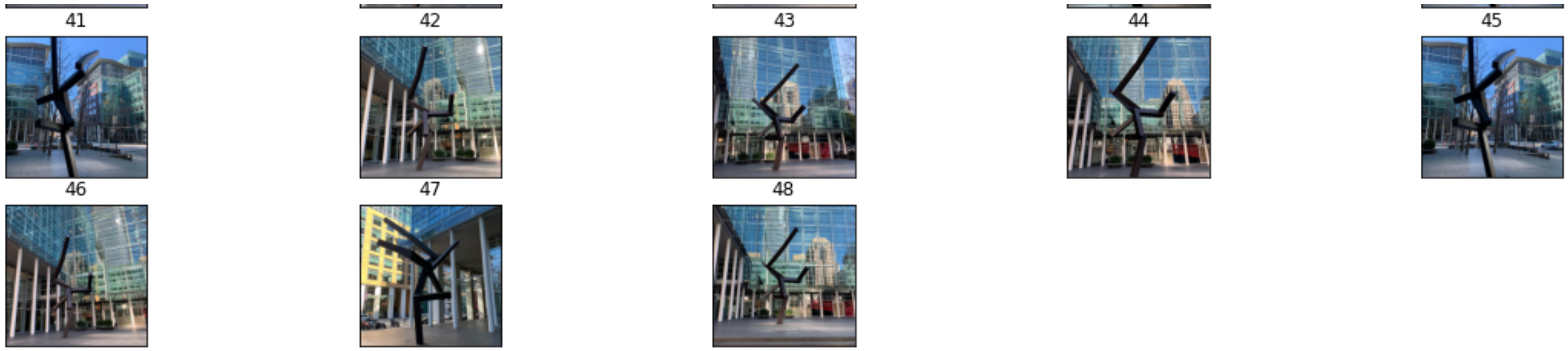
	1	2
0	-7246.326090	-7918.640678
1	9350.144377	2814.114686
2	-9705.062618	-124.327807
3	10766.790330	8391.582263
4	-10653.344788	-1874.696437

4. Show reconstruction from each low-dimensional representation

Let's view the original photos again to compare with the reconstructed photos

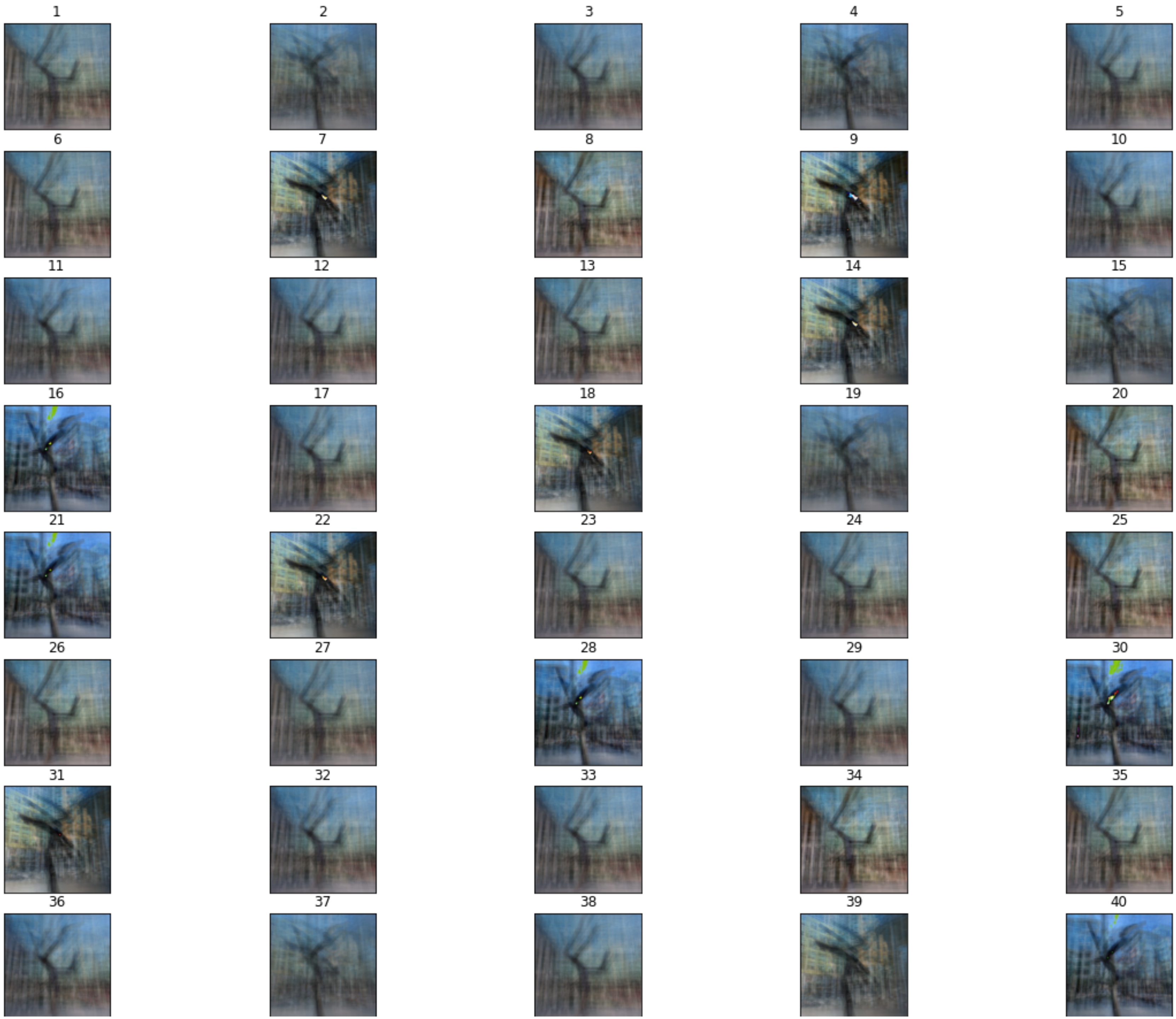
In [ ]:
```
# Original photos

nrows = len(X)//5 + 1
ncols = 5

plt.figure(figsize=[20, 20])
plt.title("Original photos")
for i in range(len(X)):
    plt.subplot(nrows, ncols, i+1, xticks=[], yticks=[])
    plt.imshow(X[i].reshape(PHOTO_SIZE, PHOTO_SIZE, 3).astype(np.uint8))
    plt.title(f"{i+1}")
plt.show()
```

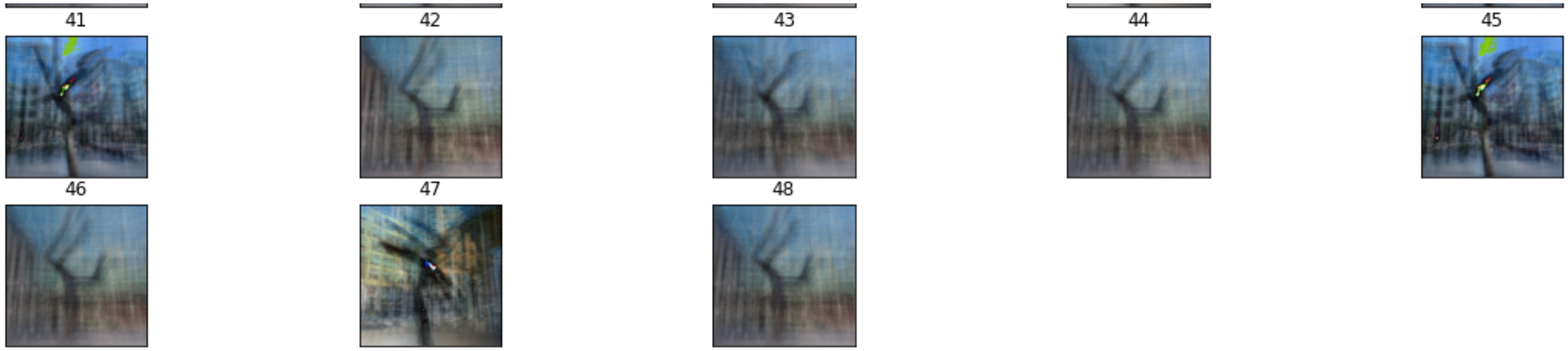

1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
26 27 28 29 30
31 32 33 34 35
36 37 38 39 40



In [ ]:
```
reconstructed = pca.inverse_transform(X_2d)
```

In [ ]:
```
# Reconstructed photos

nrows = len(X)//5 + 1
ncols = 5

plt.figure(figsize=[20, 20])
for i in range(len(X)):
    plt.subplot(nrows, ncols, i+1, xticks=[], yticks=[])
    plt.imshow(reconstructed[i].reshape(PHOTO_SIZE, PHOTO_SIZE, 3).astype(np.uint8))
    plt.title(f"{i+1}")
plt.show()
```

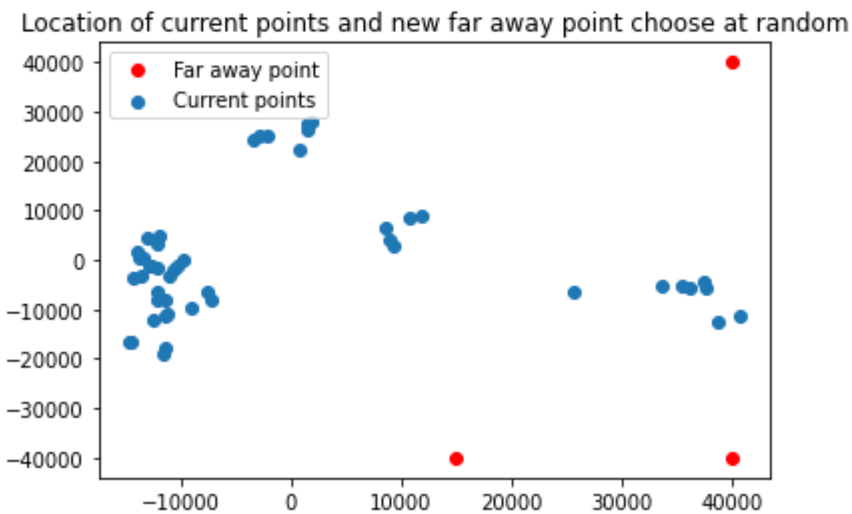

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40



5. Plot the reconstruction from a point far away

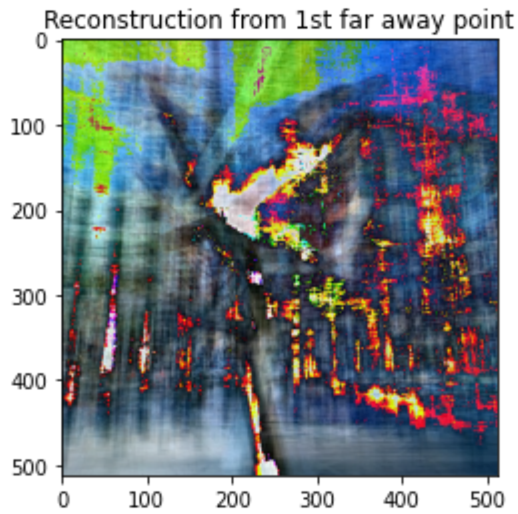
Let's pick 3 far away points to compare their reconstructions

In [ ]:
```
far_away_points = np.array([[40000, 40000], [40000, -40000], [15000, -40000]])

plt.plot(far_away_points[:, 0], far_away_points[:, 1], 'ro', label="Far away point")
plt.scatter(X_2d[:, 0], X_2d[:, 1], label="Current points")
plt.title("Location of current points and new far away point choose at random")
plt.legend()
plt.show()
```



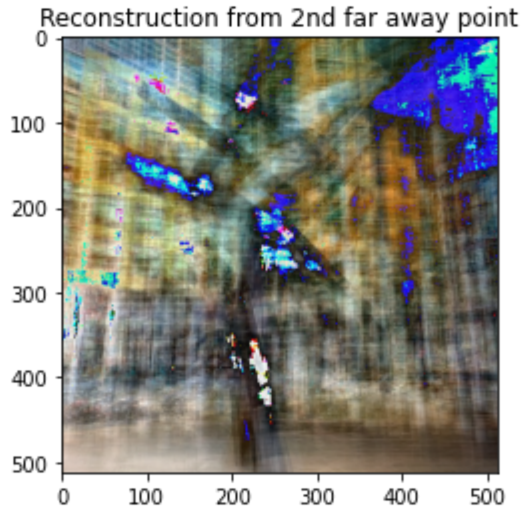
In [ ]:
```
current_far_away_point = np.reshape(np.array(far_away_points[0]), (1, -1))
far_away_point_inversed = pca.inverse_transform(current_far_away_point)

plt.title("Reconstruction from 1st far away point")
plt.imshow(np.reshape(far_away_point_inversed, (PHOTO_SIZE, PHOTO_SIZE, 3)).astype(np.uint8))
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fd01e4e2dc0>
```



In [ ]:
```
current_far_away_point = np.reshape(np.array(far_away_points[1]), (1, -1))
far_away_point_inversed = pca.inverse_transform(current_far_away_point)

plt.title("Reconstruction from 2nd far away point")
plt.imshow(np.reshape(far_away_point_inversed, (PHOTO_SIZE, PHOTO_SIZE, 3)).astype(np.uint8))
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fd01e60dc10>
```

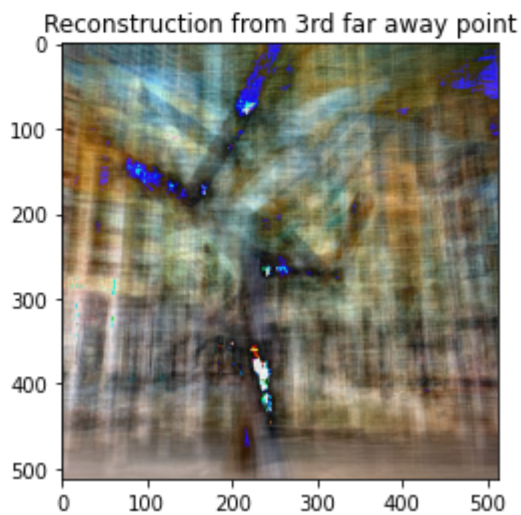


In [ ]:

```
current_far_away_point = np.reshape(np.array(far_away_points[2]), (1, -1))
far_away_point_inversed = pca.inverse_transform(current_far_away_point)

plt.title("Reconstruction from 3rd far away point")
plt.imshow(np.reshape(far_away_point_inversed, (PHOTO_SIZE, PHOTO_SIZE, 3)).astype(np.uint8))
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7fd01e38c9d0>
```



There's a little difference between the reconstruction at different far away points, perhaps in lighting!

Reflection

- Overall, not so much variation is explained (~19%) with 2 components.
- Because of the low variation, the reconstruction is not really close to the original one. The reconstruction also got distracted because of the background building noise.