COMPSCI 289A Homework 7
4/28/2017

Mitch Negus
3032146443

Problem 1

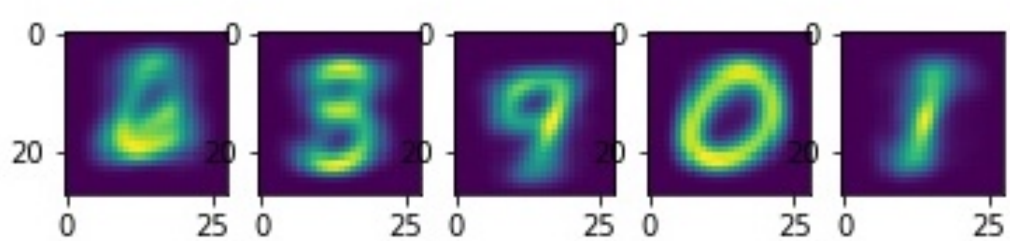
a.)

(See code)

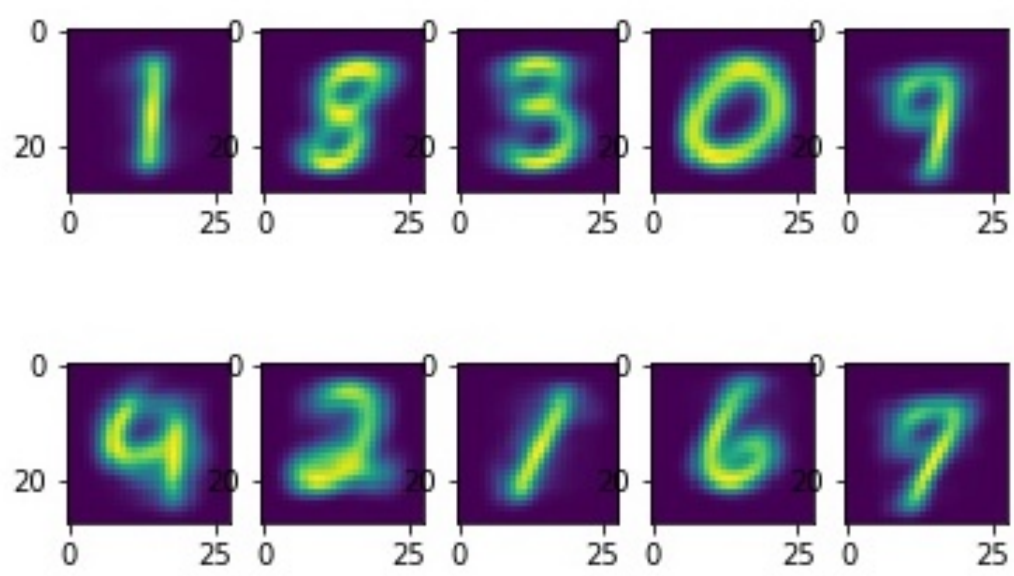
b.)

Visualization of MNIST for *k*-means

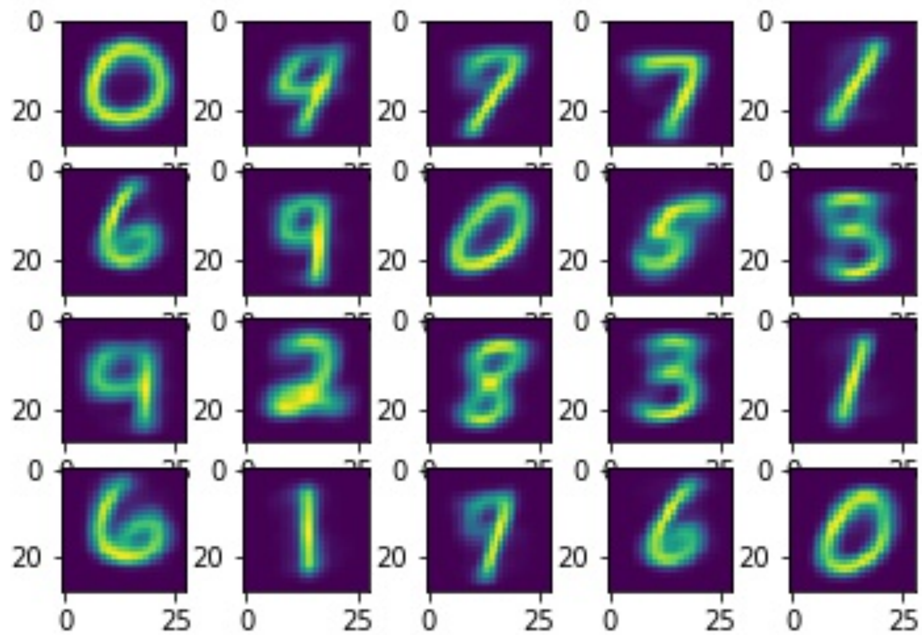
$k = 5$



$k = 10$



$k = 20$



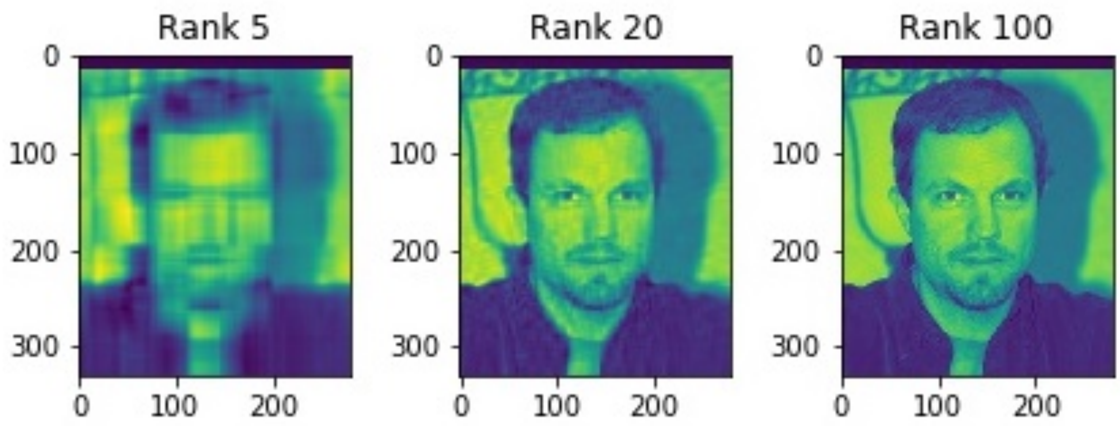
It becomes apparent that the resolution of the numbers that are picked out by the clusters improves. For example, it is difficult to identify a number corresponding to the first cluster in the $k = 5$ set. In the $k = 10$ set we are able to identify most of the numbers (to at least two possible digits) even though not all digits 1-10 are represented. Finally, in the $k = 20$ clustering, we have at least one clustering corresponding to each digit 1-10 and the resolution of each cluster is significantly improved.

Problem 2

a.)

Low-Rank Approximation: Face Image

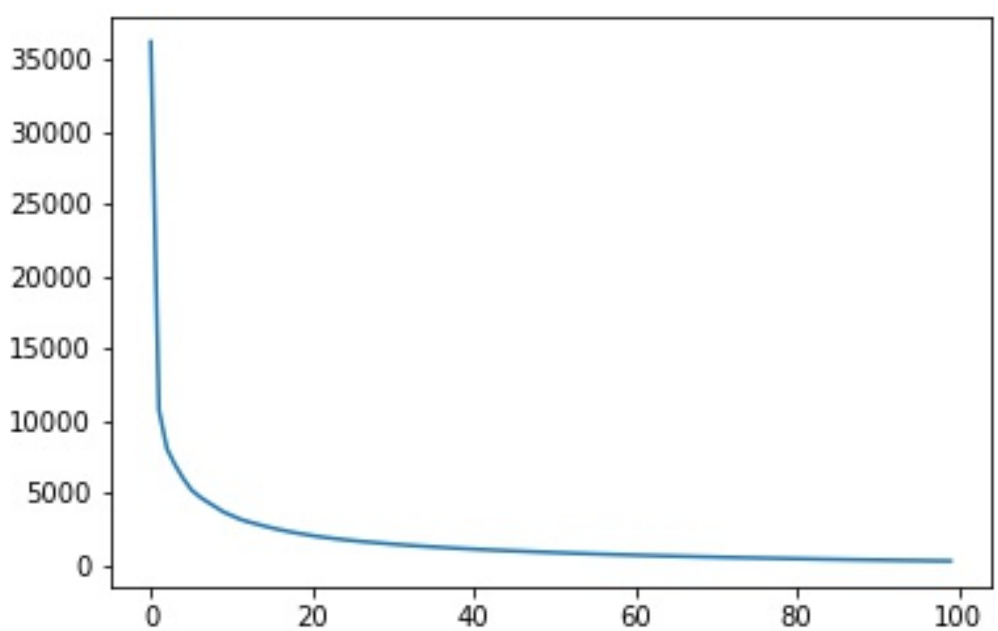
Rank-5, rank-20, and rank 100 approximations:



b.)

Mean-Squared-Error: Face Image

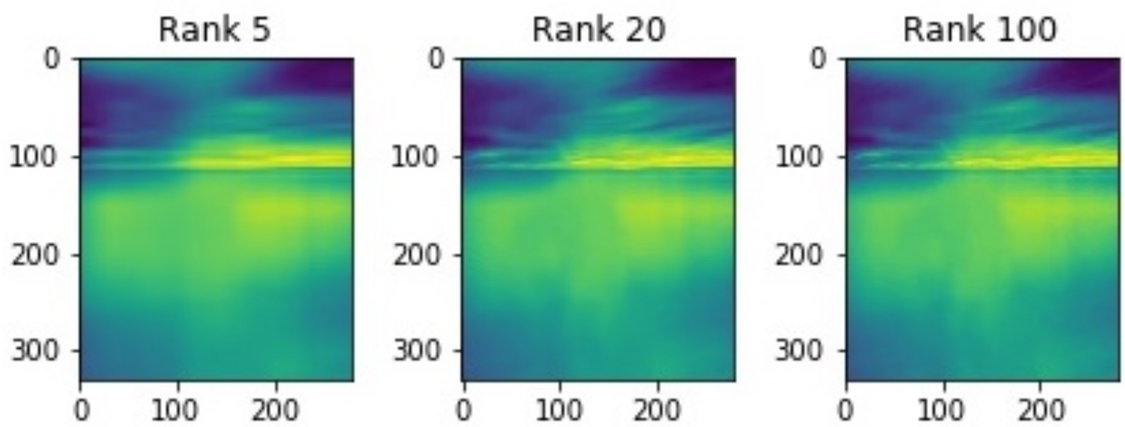
Mean-Squared Error for rank-1 through rank-100 approximations of the face image compared to the original.



c.)

Low-Rank Approximation: Sky Image

Rank-5, rank-20, and rank 100 approximations:



d.)

For the face image, a rank 40 approximation shows significantly degraded resolution (especially in the forehead area), and a rank 45 approximation still seems a bit fuzzy. I find the rank 50 approximation to be sufficiently close for the two images to be considered identical. (See Jupyter notebook for evidence)

For the sky image, the rank 20 image shows a slight degradation in quality in definition of the colors in the bottom left. This effect is significantly reduced in a rank 25 image, and by a rank 30 approximation, the original and the low-rank image are nearly indistiguishable.

The difference between the sky and face images is likely caused by the level of detail. The sky image has less distinct features, so reducing the rank will likely have less impact on the clarity of each of those features. (Additionally, from a biological perspective, the human brain is incredibly adept at identifying other human faces. Even very slight differences in the face image may trigger the brain to reject the images as identical, whereas it is more apt to overlook those slight differences in the sky image.)

[See Jupyter notebook for evidence]

CS289A_HW07_prob1

April 28, 2017

1 Homework 7: Problem 1

1.1 *k*-means Clustering

Programmatic overhead; import this homework's utility module as well as math, numpy, scipy and matplotlib.

```
In [1]: import HW07_utils as ut
        import math
        import numpy as np
        from matplotlib import pyplot as plt
        from scipy import ndimage as ndi
        from scipy import io as spio
```

Load the MNIST data into memory.

```
In [2]: BASE_DIR = '/Users/mitch/Documents/Cal/2_2017_Spring/COMPSCI 289A - Intro t
        DATA_PATH = 'Data/hw7_data/mnist_data/images.mat'
        mnistdataraw = ut.load_data(DATA_PATH,BASE_DIR,'images')
        mnistdata = np.empty((60000,784))
        for i in range(60000):
            mnistdata[i]=np.reshape(mnistdataraw[:,:,i],(784,))
```

```
In [6]: class Cluster:
            """ A class to perform k-means clustering"""

            def __init__(self,k):
                self.k = k
                self.means = None
                self.clusters = None


            def update_means(self):
                """Method to calculate the means of each cluster"""
                for i in range(self.k):
                    cluster_i = np.array(self.clusters[i])
                    mu_i = np.mean(cluster_i,axis=0)
                    self.means[i] = mu_i
```

```
def update_clusters(self):
    """ Method to take in a set of means and reclassify points accordi
    new_clusters = [[] for k_i in range(self.k)]
    clusters_changed = 0

    for i in range(self.k):
        while True:
            try:
                x_j = self.clusters[i].pop(0)
                # Create object to store the i-value of the closest mea
                cluster_index = self.assign_cluster(x_j)

                # Reclassify point
                if cluster_index != i:
                    clusters_changed += 1
                new_clusters[cluster_index].append(x_j)

            except:
                break
    self.clusters = new_clusters

    return clusters_changed


def forgy_init(self,data):
    """
    Execute the Forgy initializaiton method:
            -choose k random sample points from data to be initial mea
    """
    select = np.random.choice(len(data),self.k,replace=False)
    self.means = data[select]

    # Assign points according to these random means
    self.clusters = [[] for k_i in range(self.k)]
    for datapoint in data:
        cluster_index = self.assign_cluster(datapoint)
        self.clusters[cluster_index].append(datapoint)


def assign_cluster(self,x_j):
    """Assign point x_j to the cluster with nearest mean"""
    nearest_mean_index = -1
    nearest_mean_dist = math.inf
    for i in range(self.k):
        # Check the distance of the point to each mean
        mu_i = self.means[i]
        d = np.linalg.norm(x_j-mu_i)
```

```
                if d < nearest_mean_dist:
                    # Reassign closest mean
                    nearest_mean_index = i
                    nearest_mean_dist = d

            return nearest_mean_index


        def lloyd_alg(self):
            """
            Execute Lloyd's algorithm to construct k clusters:
                    -Minimize the sum of squared distances of points from clust
                    -Use k-means heuristic to alternate between updating means
            """
            clusters_changed = 1
            counter = 0
            while clusters_changed != 0:
                print('Iteration', counter)

                self.update_means()
                clusters_changed = self.update_clusters()
                print(clusters_changed)
                counter += 1

            print('Finished')
            for cluster in self.clusters:
                    print(np.shape(cluster))
```

Test the clustering algorithm on a simple dataset.

```
In [7]: simpledata = np.array([[3,10,10],[9,10,10],[9,9,10],[9,4,10],[10,3,10],[4,8
        clustering = Cluster(3)
        clustering.forgy_init(simpledata)
        clustering.lloyd_alg()
        print(np.array(clustering.clusters))
```

```
Iteration 0
1
Iteration 1
0
Finished
(3, 3)
(3, 3)
(3, 3)
[[[ 3 10 10]
  [ 4  8 10]
  [ 2  8 10]]
```

```
 [[ 9 10 10]
  [ 9  9 10]
  [ 9  8 10]]

 [[ 9  4 10]
  [10  3 10]
  [ 8  2 10]]]
```

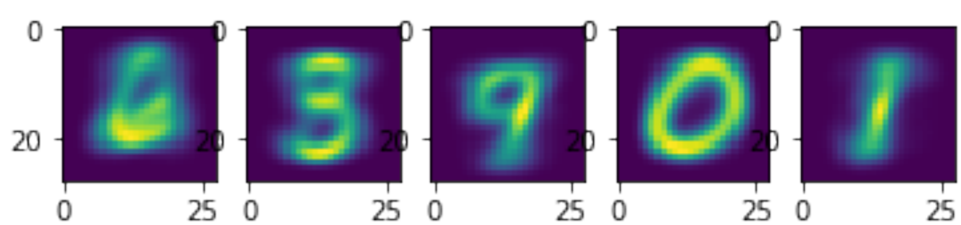
```
In [9]: clustering_k5 = Cluster(5)
        clustering_k5.forgy_init(mnistdata)
        clustering_k5.lloyd_alg()
```

```
Iteration 0
11738
Iteration 1
6207
Iteration 2
4116
Iteration 3
3457
Iteration 4
3170
Iteration 5
3138
Iteration 6
3042
Iteration 7
2661
Iteration 8
2285
Iteration 9
1963
Iteration 10
1740
Iteration 11
1604
Iteration 12
1488
Iteration 13
1457
Iteration 14
1653
Iteration 15
1828
Iteration 16
2045
```

```
Iteration 17
2088
Iteration 18
2104
Iteration 19
1910
Iteration 20
1686
Iteration 21
1479
Iteration 22
1239
Iteration 23
970
Iteration 24
666
Iteration 25
490
Iteration 26
329
Iteration 27
239
Iteration 28
179
Iteration 29
126
Iteration 30
75
Iteration 31
54
Iteration 32
27
Iteration 33
25
Iteration 34
24
Iteration 35
7
Iteration 36
5
Iteration 37
3
Iteration 38
1
Iteration 39
2
Iteration 40
2
```

```
Iteration 41
3
Iteration 42
2
Iteration 43
2
Iteration 44
1
Iteration 45
1
Iteration 46
2
Iteration 47
3
Iteration 48
1
Iteration 49
1
Iteration 50
0
Finished
(10901, 784)
(11630, 784)
(16619, 784)
(5563, 784)
(15287, 784)
```

In [21]:
```
k5 = plt.figure()
for k in range(5):
    cluster = clustering_k5.means[k]
    center = np.reshape(cluster,(28,28))

    k5.add_subplot(1,5,k+1)
    plt.imshow(center)
plt.savefig(BASE_DIR+'Figures/MNIST_k05.jpg')
plt.show()
```



```
In [10]: clustering_k10 = Cluster(10)
         clustering_k10.forgy_init(mnistdata)
         clustering_k10.lloyd_alg()
```

```
Iteration 0
16393
Iteration 1
7565
Iteration 2
5146
Iteration 3
4241
Iteration 4
4000
Iteration 5
3634
Iteration 6
2980
Iteration 7
2302
Iteration 8
1701
Iteration 9
1351
Iteration 10
1042
Iteration 11
883
Iteration 12
754
Iteration 13
734
Iteration 14
675
Iteration 15
651
Iteration 16
674
Iteration 17
713
Iteration 18
718
Iteration 19
765
Iteration 20
816
Iteration 21
825
```

```
Iteration 22
821
Iteration 23
824
Iteration 24
814
Iteration 25
831
Iteration 26
827
Iteration 27
746
Iteration 28
778
Iteration 29
725
Iteration 30
661
Iteration 31
547
Iteration 32
488
Iteration 33
414
Iteration 34
360
Iteration 35
291
Iteration 36
242
Iteration 37
235
Iteration 38
195
Iteration 39
172
Iteration 40
160
Iteration 41
160
Iteration 42
150
Iteration 43
129
Iteration 44
118
Iteration 45
100
```

```
Iteration 46
104
Iteration 47
103
Iteration 48
104
Iteration 49
97
Iteration 50
89
Iteration 51
79
Iteration 52
87
Iteration 53
78
Iteration 54
71
Iteration 55
55
Iteration 56
54
Iteration 57
54
Iteration 58
60
Iteration 59
54
Iteration 60
69
Iteration 61
71
Iteration 62
86
Iteration 63
84
Iteration 64
63
Iteration 65
57
Iteration 66
48
Iteration 67
43
Iteration 68
45
Iteration 69
52
```

```
Iteration 70
63
Iteration 71
51
Iteration 72
49
Iteration 73
64
Iteration 74
67
Iteration 75
73
Iteration 76
84
Iteration 77
104
Iteration 78
103
Iteration 79
110
Iteration 80
107
Iteration 81
112
Iteration 82
99
Iteration 83
91
Iteration 84
84
Iteration 85
76
Iteration 86
79
Iteration 87
72
Iteration 88
69
Iteration 89
72
Iteration 90
50
Iteration 91
29
Iteration 92
25
Iteration 93
22
```

```
Iteration 94
29
Iteration 95
29
Iteration 96
35
Iteration 97
31
Iteration 98
35
Iteration 99
26
Iteration 100
32
Iteration 101
35
Iteration 102
44
Iteration 103
44
Iteration 104
45
Iteration 105
38
Iteration 106
20
Iteration 107
20
Iteration 108
19
Iteration 109
21
Iteration 110
29
Iteration 111
33
Iteration 112
31
Iteration 113
18
Iteration 114
14
Iteration 115
15
Iteration 116
15
Iteration 117
18
```

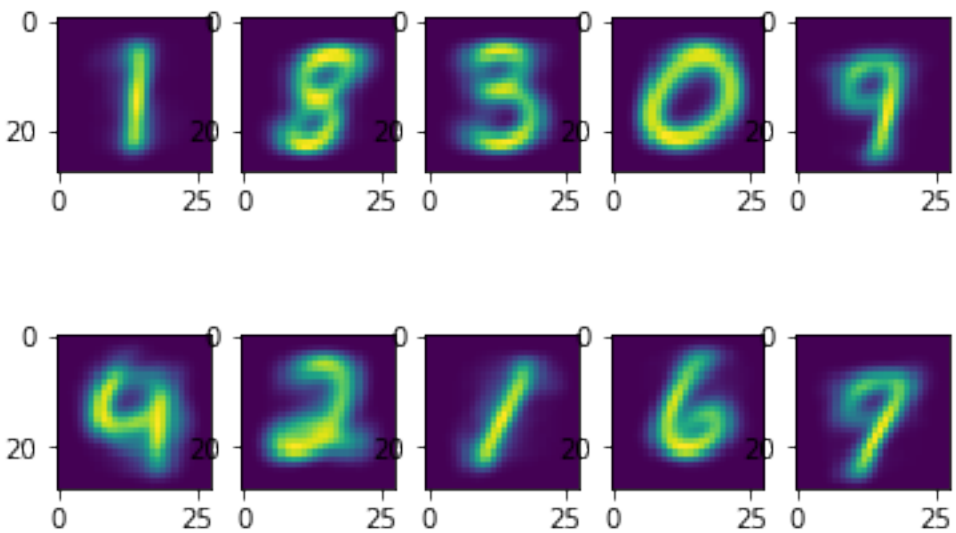
```
Iteration 118
20
Iteration 119
33
Iteration 120
48
Iteration 121
43
Iteration 122
41
Iteration 123
51
Iteration 124
57
Iteration 125
63
Iteration 126
62
Iteration 127
57
Iteration 128
57
Iteration 129
60
Iteration 130
65
Iteration 131
66
Iteration 132
65
Iteration 133
79
Iteration 134
73
Iteration 135
66
Iteration 136
69
Iteration 137
67
Iteration 138
66
Iteration 139
60
Iteration 140
66
Iteration 141
66
```

```
Iteration 142
69
Iteration 143
75
Iteration 144
86
Iteration 145
79
Iteration 146
63
Iteration 147
66
Iteration 148
44
Iteration 149
38
Iteration 150
35
Iteration 151
25
Iteration 152
13
Iteration 153
9
Iteration 154
9
Iteration 155
13
Iteration 156
15
Iteration 157
12
Iteration 158
13
Iteration 159
12
Iteration 160
12
Iteration 161
16
Iteration 162
17
Iteration 163
20
Iteration 164
21
Iteration 165
15
```

```
Iteration 166
11
Iteration 167
10
Iteration 168
9
Iteration 169
7
Iteration 170
9
Iteration 171
8
Iteration 172
5
Iteration 173
12
Iteration 174
10
Iteration 175
16
Iteration 176
21
Iteration 177
20
Iteration 178
18
Iteration 179
16
Iteration 180
12
Iteration 181
7
Iteration 182
12
Iteration 183
16
Iteration 184
11
Iteration 185
10
Iteration 186
12
Iteration 187
5
Iteration 188
2
Iteration 189
1
```

```
Iteration 190
3
Iteration 191
2
Iteration 192
0
Finished
(6311, 784)
(6208, 784)
(7178, 784)
(5043, 784)
(7409, 784)
(5041, 784)
(4793, 784)
(5927, 784)
(5435, 784)
(6655, 784)
```

```
In [19]: k10 = plt.figure()
         for k in range(10):
             cluster = clustering_k10.means[k]
             center = np.reshape(cluster,(28,28))

             k10.add_subplot(2,5,k+1)
             plt.imshow(center)
         plt.savefig(BASE_DIR+'Figures/MNIST_k10.jpg')
         plt.show()
```



```
In [11]: clustering_k20 = Cluster(20)
         clustering_k20.forgy_init(mnistdata)
         clustering_k20.lloyd_alg()
```

```
Iteration 0
18870
Iteration 1
9819
Iteration 2
6728
Iteration 3
4908
Iteration 4
3726
Iteration 5
2961
Iteration 6
2337
Iteration 7
1944
Iteration 8
1743
Iteration 9
1593
Iteration 10
1424
Iteration 11
1201
Iteration 12
1081
Iteration 13
986
Iteration 14
968
Iteration 15
897
Iteration 16
829
Iteration 17
721
Iteration 18
650
Iteration 19
535
Iteration 20
483
Iteration 21
430
```

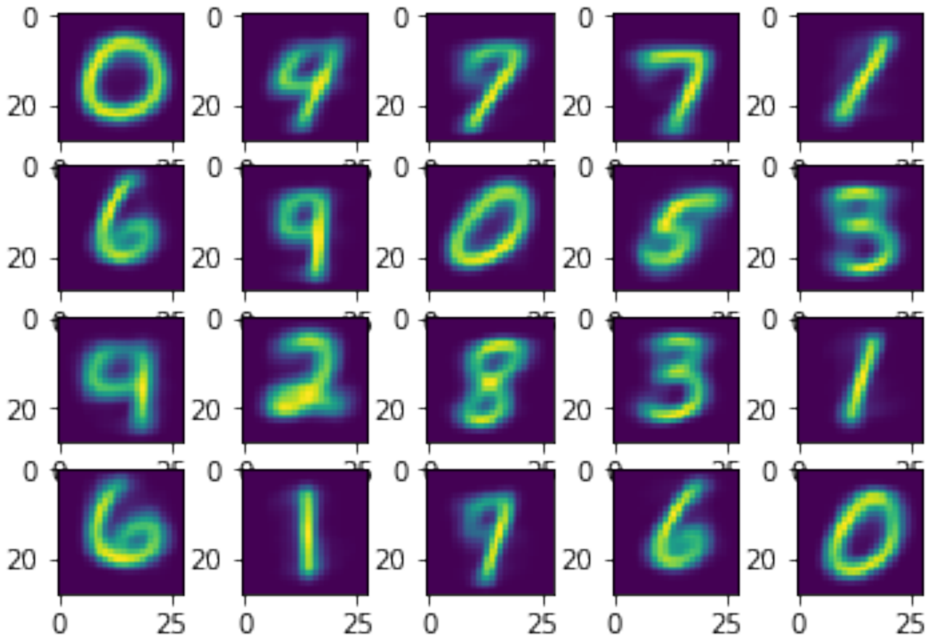
```
Iteration 22
377
Iteration 23
303
Iteration 24
275
Iteration 25
276
Iteration 26
243
Iteration 27
195
Iteration 28
204
Iteration 29
190
Iteration 30
158
Iteration 31
155
Iteration 32
156
Iteration 33
161
Iteration 34
139
Iteration 35
110
Iteration 36
121
Iteration 37
101
Iteration 38
95
Iteration 39
99
Iteration 40
88
Iteration 41
73
Iteration 42
56
Iteration 43
50
Iteration 44
40
Iteration 45
47
```

```
Iteration 46
56
Iteration 47
67
Iteration 48
68
Iteration 49
79
Iteration 50
85
Iteration 51
63
Iteration 52
64
Iteration 53
55
Iteration 54
44
Iteration 55
40
Iteration 56
39
Iteration 57
48
Iteration 58
46
Iteration 59
48
Iteration 60
38
Iteration 61
29
Iteration 62
30
Iteration 63
12
Iteration 64
9
Iteration 65
10
Iteration 66
9
Iteration 67
5
Iteration 68
8
Iteration 69
5
```

```
Iteration 70
4
Iteration 71
2
Iteration 72
4
Iteration 73
7
Iteration 74
9
Iteration 75
7
Iteration 76
7
Iteration 77
11
Iteration 78
8
Iteration 79
6
Iteration 80
5
Iteration 81
4
Iteration 82
3
Iteration 83
4
Iteration 84
6
Iteration 85
7
Iteration 86
6
Iteration 87
9
Iteration 88
9
Iteration 89
8
Iteration 90
8
Iteration 91
8
Iteration 92
7
Iteration 93
2
```

```
Iteration 94
3
Iteration 95
0
Finished
(2222, 784)
(2929, 784)
(2962, 784)
(2377, 784)
(2611, 784)
(2583, 784)
(3917, 784)
(1763, 784)
(3109, 784)
(4386, 784)
(2936, 784)
(3919, 784)
(4390, 784)
(3958, 784)
(3011, 784)
(1739, 784)
(3453, 784)
(3645, 784)
(2310, 784)
(1780, 784)
```

```
In [23]: k20 = plt.figure()
         for k in range(20):
             cluster = clustering_k20.means[k]
             center = np.reshape(cluster,(28,28))

             k20.add_subplot(4,5,k+1)
             plt.imshow(center)
         plt.savefig(BASE_DIR+'Figures/MNIST_k20.jpg')
         plt.show()
```



```
In [ ]:
```

CS289A_HW07_prob2

April 28, 2017

1 Homework 7: Problem 2

1.1 Low-Rank Approximation

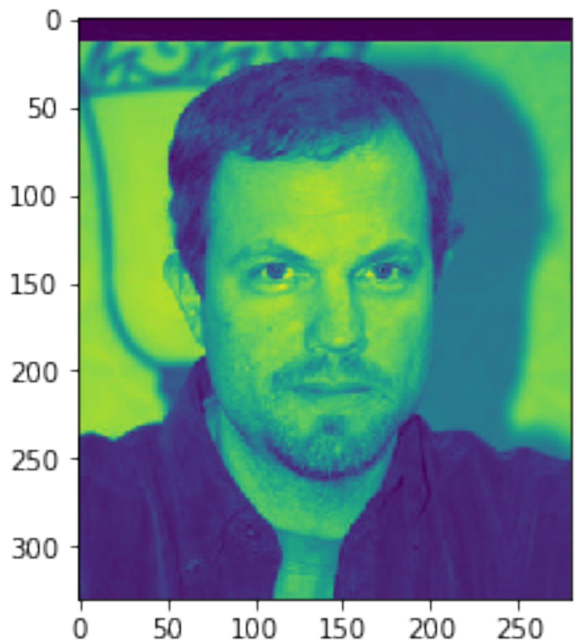
Programmatic overhead; import numpy, scipy and matplotlib.

```
In [1]: import numpy as np
        from scipy import ndimage as ndi
        from matplotlib import pyplot as plt
```

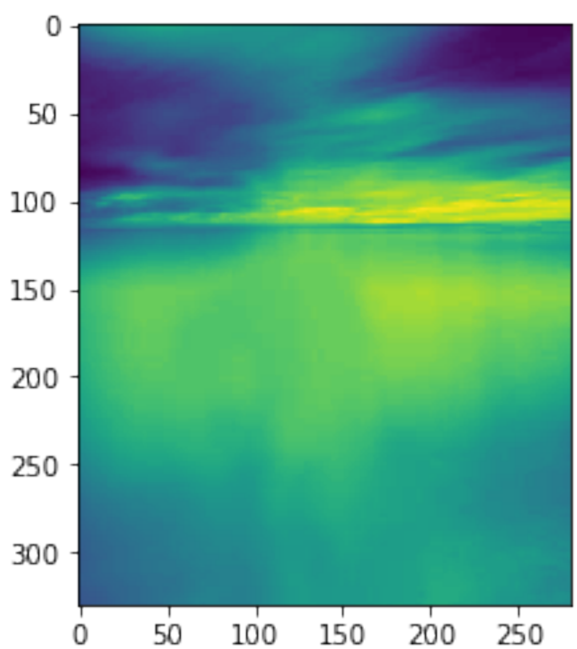
```
In [2]: BASE_DIR = '/Users/mitch/Documents/Cal/2_2017_Spring/COMPSCI 289A - Intro t
```

Load the face and sky images as arrays

```
In [3]: facefile = BASE_DIR+'Data/hw7_data/low-rank_data/face.jpg'
        face = ndi.imread(facefile,flatten=True)
        plt.imshow(face)
        plt.show()
        print(np.shape(face))
        skyfile = BASE_DIR+'Data/hw7_data/low-rank_data/sky.jpg'
        sky = ndi.imread(skyfile,flatten=True)
        plt.imshow(sky)
        plt.show()
        print(np.shape(sky))
```


0
50
100
150
200
250
300
0
50
100
150
200
250

(330, 280)


0
50
100
150
200
250
300
0
50
100
150
200
250

```
(330, 280)
```

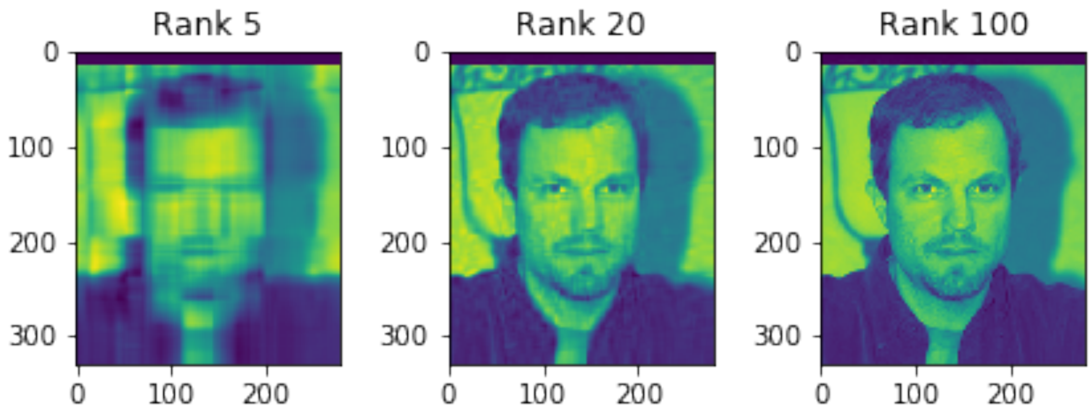
```
In [4]: def SVD(matrix):
            """Perform singluar value decomposition"""
            U,s,V = np.linalg.svd(matrix,full_matrices=0)

            return U,s,V
```

```
In [5]: def LRA(matrix,rank):
            "Generate a low-rank approximation of the input matrix"
            U,s,V = SVD(matrix)
            s_lra = np.zeros(len(s))
            for i in range(rank):
                s_lra[i] = s[i]
            lra = np.dot(np.dot(U,np.diag(s_lra)),V)

            return lra
```

```
In [6]: def MSE(matrix1,matrix2):
            "Calculate the mean squared error between 2 matrices (frobenius norm of
            mse = np.linalg.norm(matrix1-matrix2)

            return mse
```

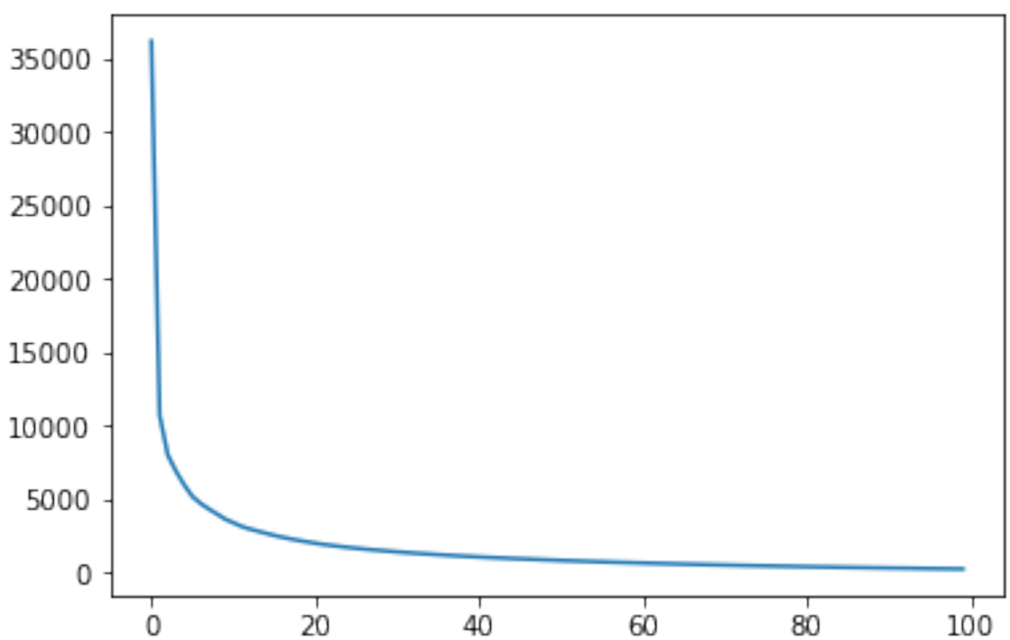
(a) Low-rank approximations of rank 5, 20, and 100 on the face image.

```
In [7]: facefig = plt.figure()
        ranks = [5,20,100]
        for rank_i in range(len(ranks)):
            rank= ranks[rank_i]
            lra = LRA(face,rank)
            facefig.add_subplot(1,len(ranks),rank_i+1)
            plt.imshow(lra)
            plt.title('Rank %i'%rank)
            plt.tight_layout()

        plt.savefig(BASE_DIR+'Figures/face_LRAs.jpg')
        plt.show()
```



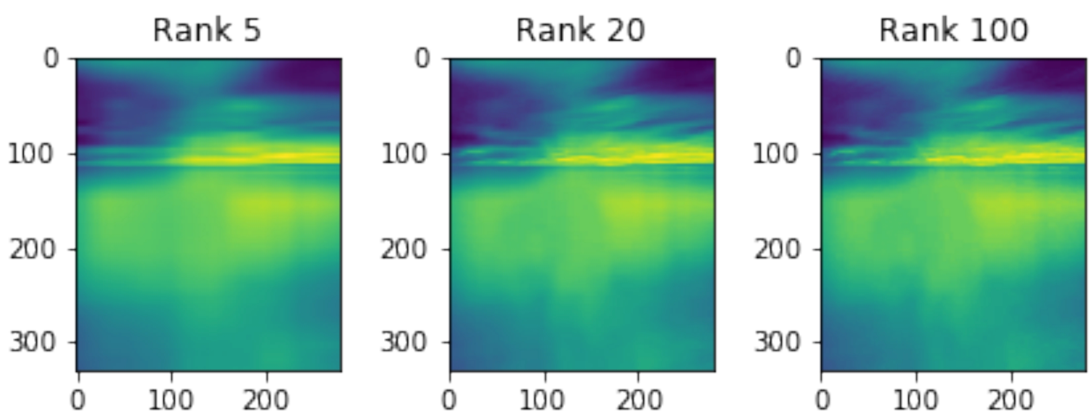
(b) Plot of mean squared error (MSE) for LRA from rank 1-100.

```
In [12]: MSEs = []
         for rank in range(100):
             lra = LRA(face,rank)
             mse = MSE(face,lra)
             MSEs.append(mse)
         plt.plot(range(100),MSEs)
         plt.savefig(BASE_DIR+'Figures/face_MSEs.jpg')
         plt.show()
```



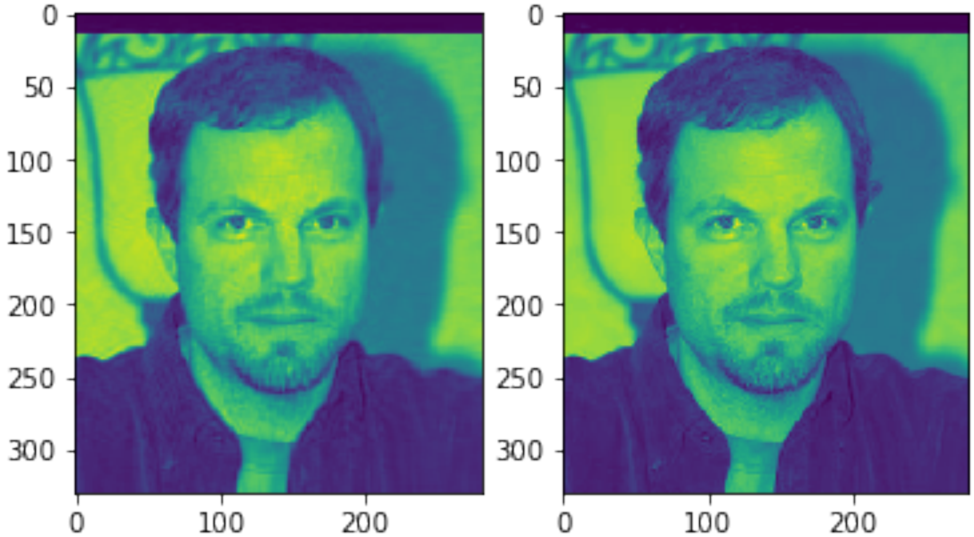
(c) Low-rank approximations of rank 5, 20, and 100 on the face image.

In [9]:
```
skyfig = plt.figure()
ranks = [5,20,100]
for rank_i in range(len(ranks)):
    rank= ranks[rank_i]
    lra = LRA(sky,rank)
    skyfig.add_subplot(1,len(ranks),rank_i+1)
    plt.imshow(lra)
    plt.title('Rank %i'%rank)
    plt.tight_layout()

plt.savefig(BASE_DIR+'Figures/sky_LRAs.jpg')
plt.show()
```

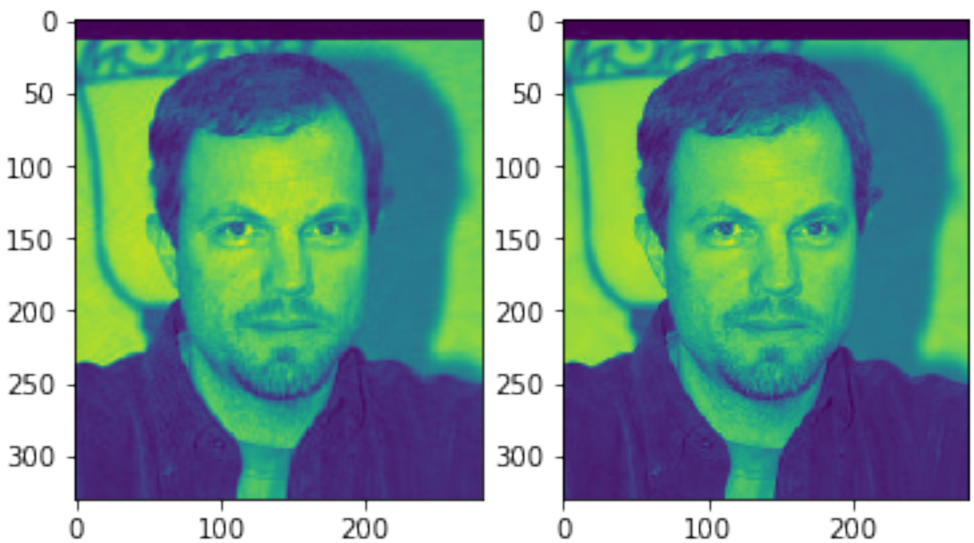


(d) The lowest rank at which the low-rank images are indistinguishable is found by inspection.

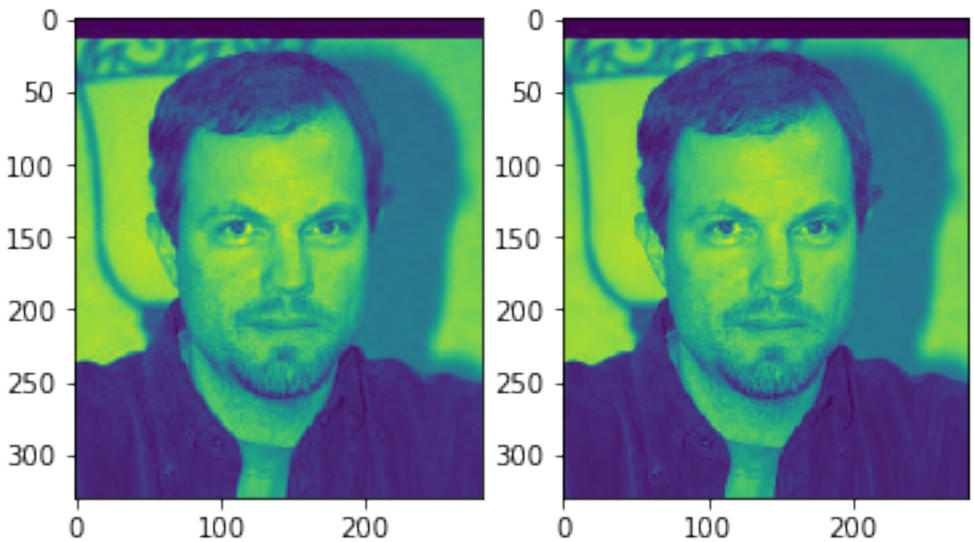
In [10]:
```
for rank in [40,45,50]:
    lra = LRA(face,rank)

    fig = plt.figure()
    fig.add_subplot(1,2,1)
    plt.imshow(lra)
    fig.add_subplot(1,2,2)
    plt.imshow(face)
    plt.show()
    print('Rank',rank,'\n')
```



Rank 40



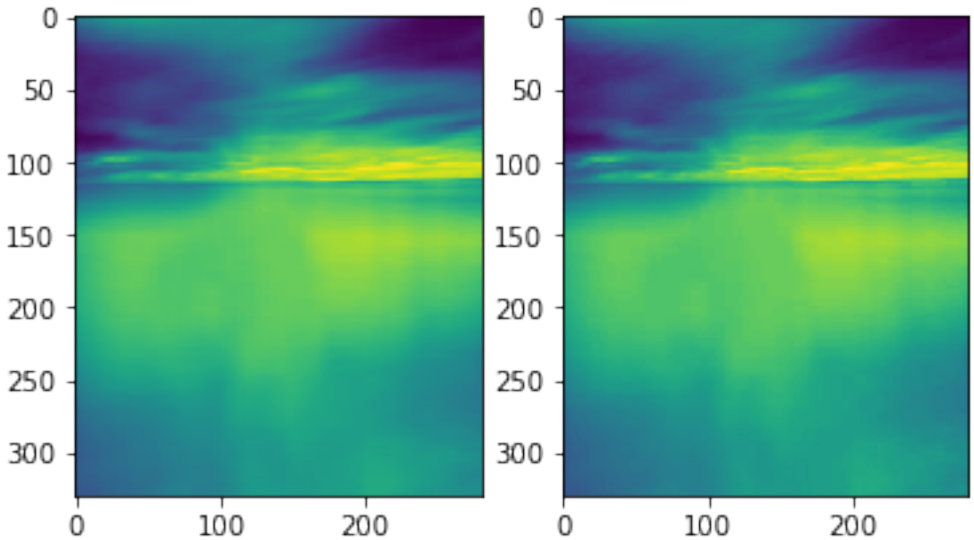
Rank 45



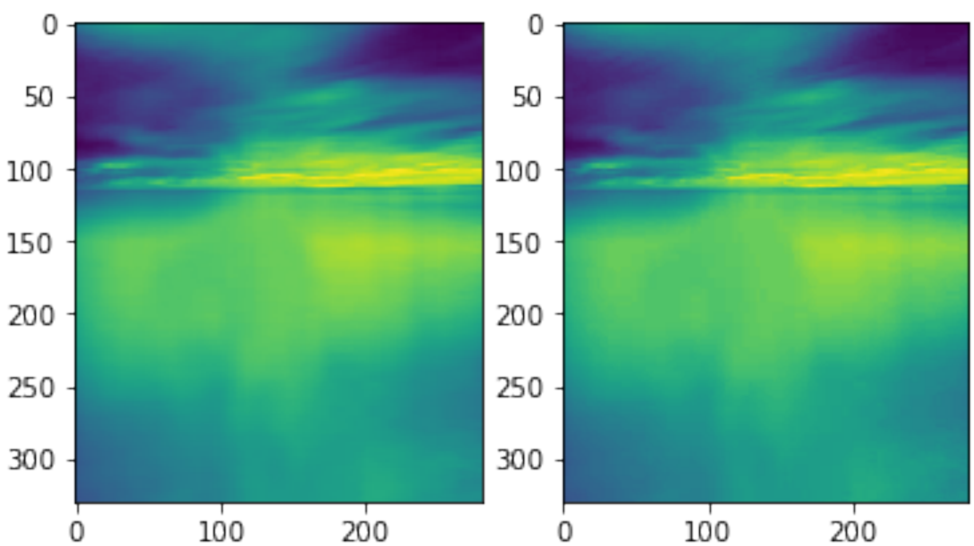

```
Rank 50
```

For the face image, a rank 40 approximation shows significantly degraded resolution (especially in the forehead area), and a rank 45 approximation still seems a bit fuzzy. I find the rank 50 approximation to be sufficiently close for the two images to be considered identical.

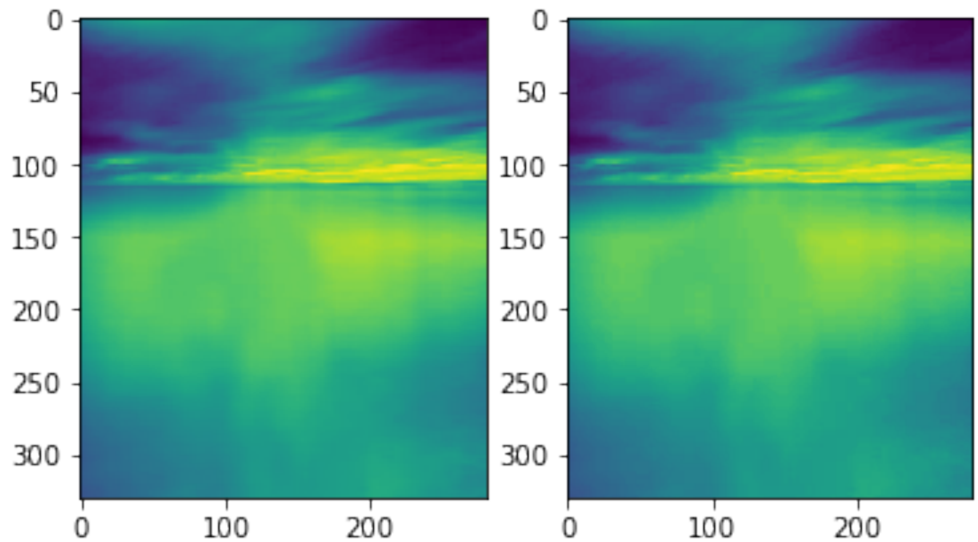
```
In [11]: for rank in [20,25,30]:
             lra = LRA(sky,rank)

             fig = plt.figure()
             fig.add_subplot(1,2,1)
             plt.imshow(lra)
             fig.add_subplot(1,2,2)
             plt.imshow(sky)
             plt.show()
             print('Rank',rank,'\n')
```



Rank 20



Rank 25




```
Rank 30
```

For the sky image, the rank 20 image shows a slight degradation in quality in definition of the colors in the bottom left. This effect is significantly reduced in a rank 25 image, and by a rank 30 approximation, the original and the low-rank image are nearly indistiguishable.

The difference between the sky and face images is likely caused by the level of detail. The sky image has less distinct features, so reducing the rank will likely have less impact on the clarity of each of those features. (Additionally, from a biological perspective, the human brain is incredibly adept at identifying other human faces. Even very slight differences in the face image may trigger the brain to reject the images as identical, whereas it is more apt to overlook those slight differences in the sky image.)

```
In [ ]:
```