CS498
Applied Machine Learning
Assignment #5

Students:
nidiaib2, Nidia Bucarelli
sunnyk2, Sunny Katiyar
wangx2, Wang Xiang

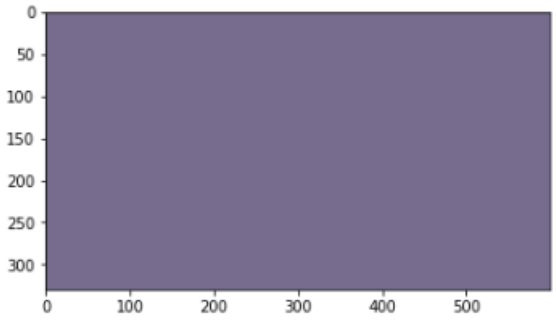
April 28, 2020
Spring 2020

PROBLEM 9.3

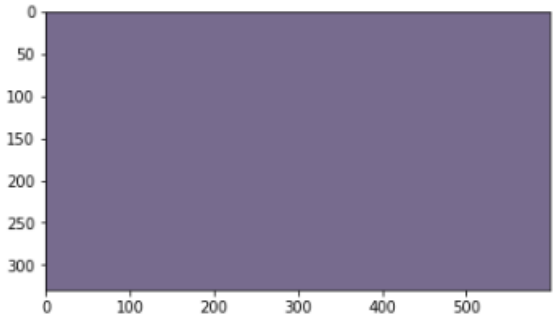
Part A.

Images:

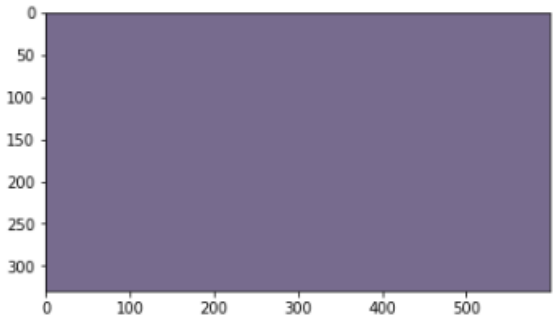
10 Clusters



20 Clusters



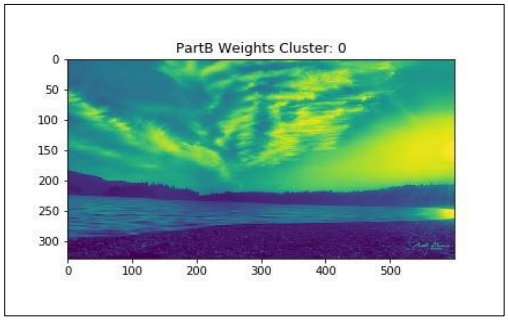
50 Clusters

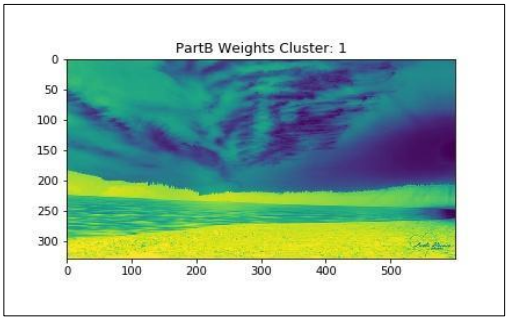


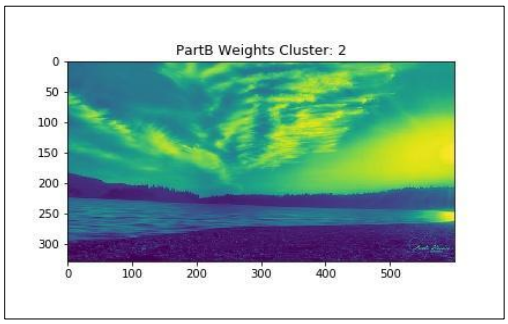
Observation:

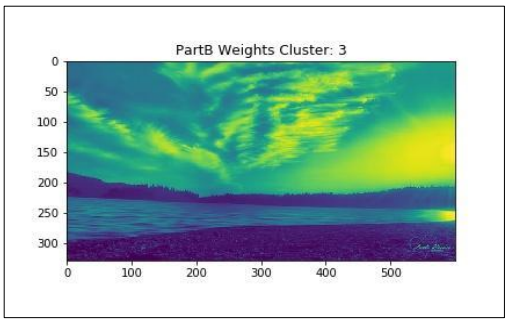
When the covariance matrix is equal to the Identity matrix, we observe that the cluster centers drift together. This happens because, at this scale, all the pixels are essentially one cluster as they are small and too close to each other.

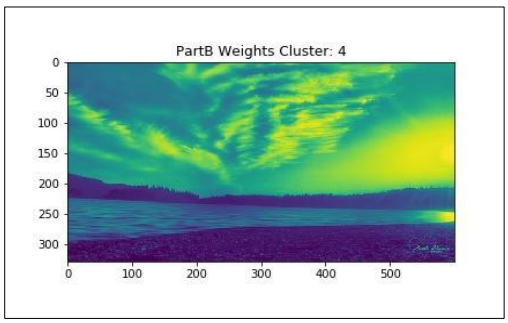
Part B.

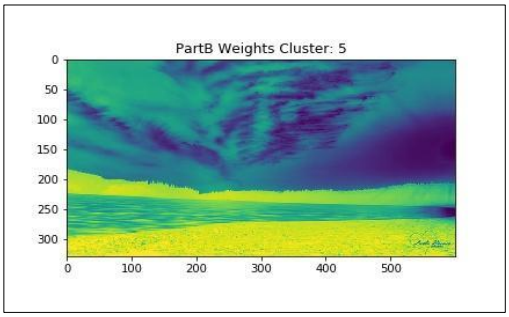

PartB Weights Cluster: 0

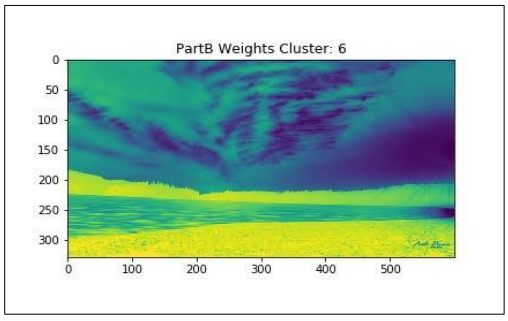

PartB Weights Cluster: 1

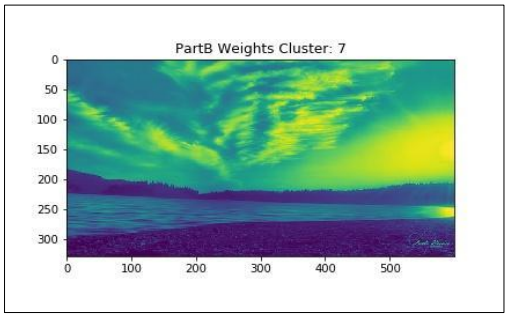

PartB Weights Cluster: 2

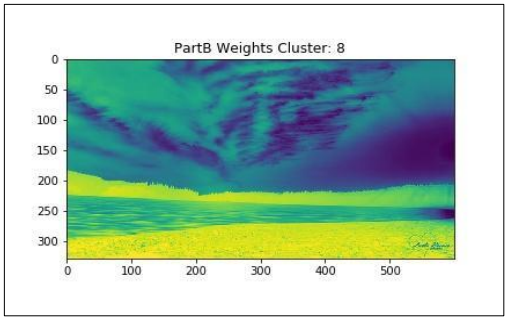

PartB Weights Cluster: 3

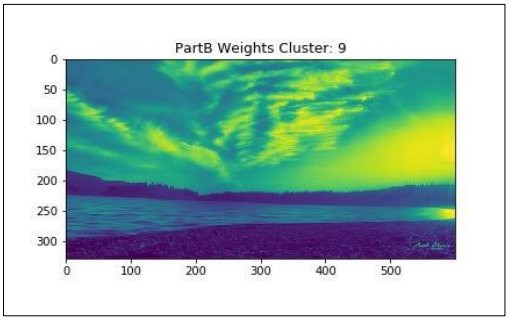

PartB Weights Cluster: 4


PartB Weights Cluster: 5


PartB Weights Cluster: 6


PartB Weights Cluster: 7





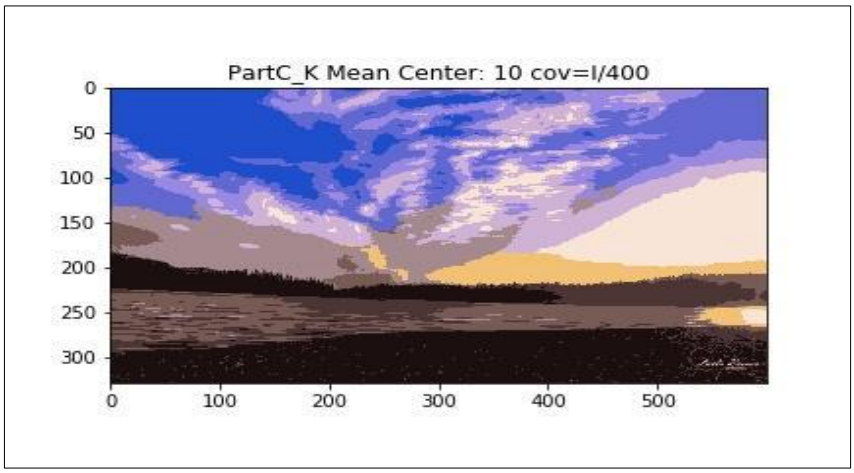
Observation:

We can see that all maps are pretty much the same, validating that the weights linking a given pixel to each cluster center do not vary very much. This is again because pixel values are small and the distance between the pixels is tiny, making them all fall into one cluster. Also, the pixel values being close to each other and with Identity matrix as covariance matrix, the resulting likelihood of each pixel belonging to different clusters is almost same. This results in the similar (posterior probability) weights linking a given pixel to each cluster.

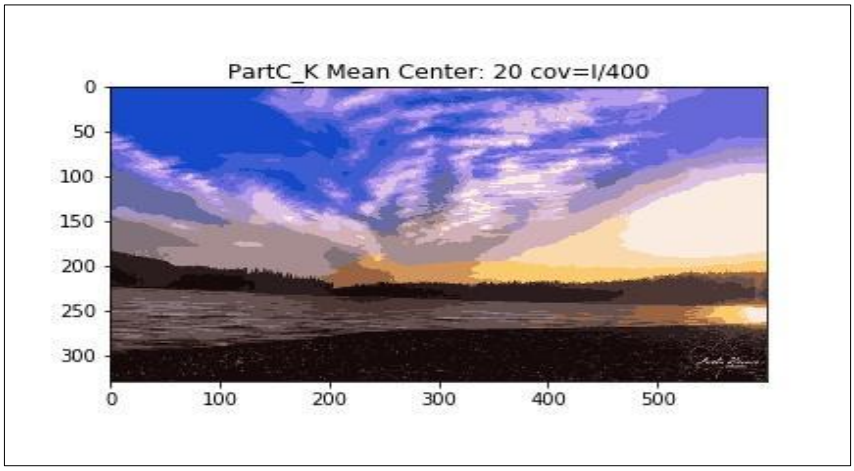
Part C.

Clustering pixels into 10, 20, and 50 clusters using (1/400) * Identity Matrix as the covariance matrix.

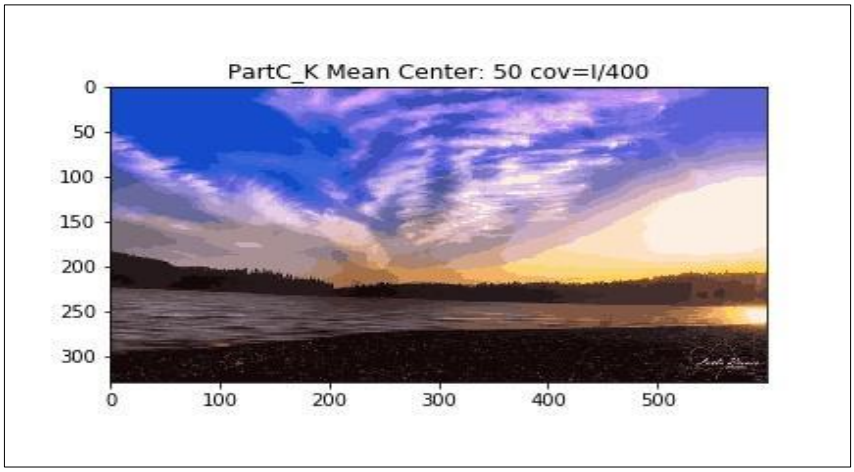
10 Clusters



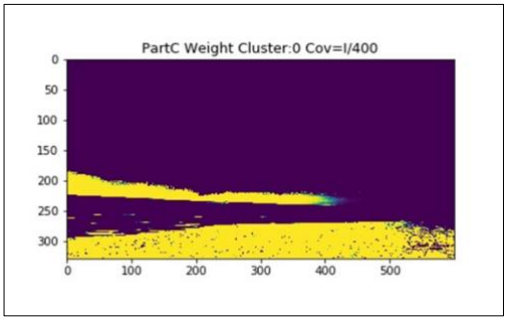
20 Clusters

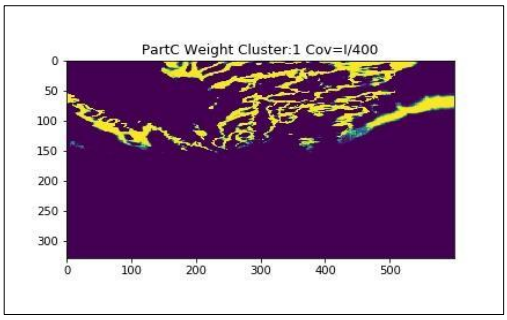


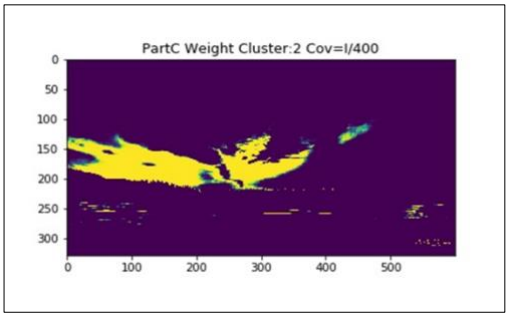
50 Clusters

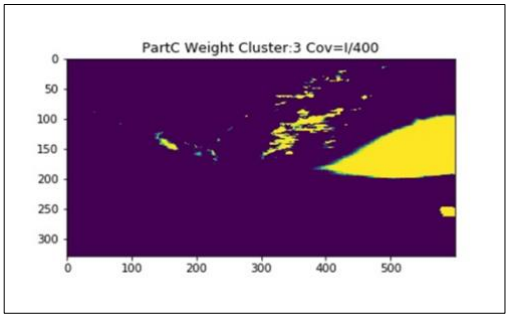


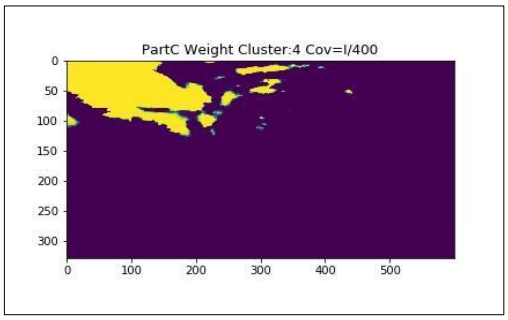
New set of weight maps using (1/400) * Identity Matrix as the covariance matrix:

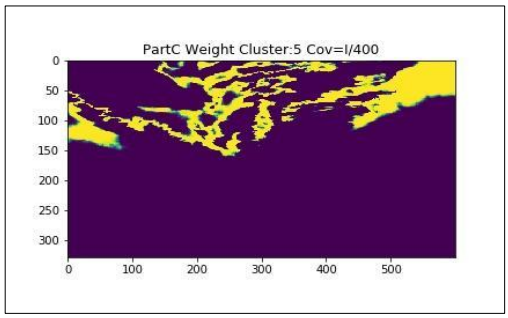


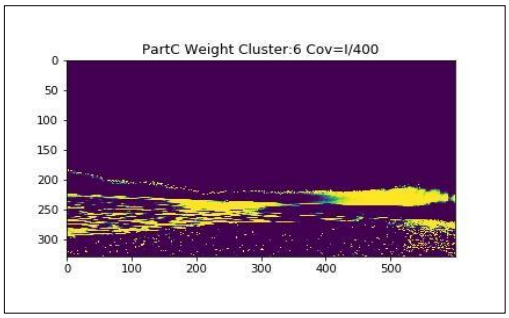


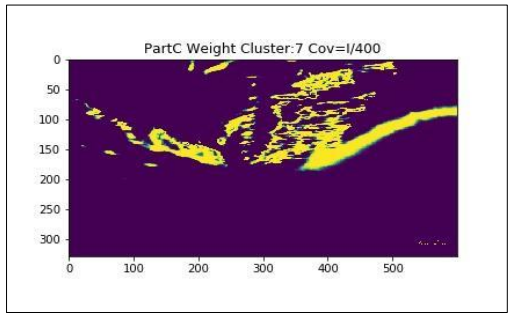


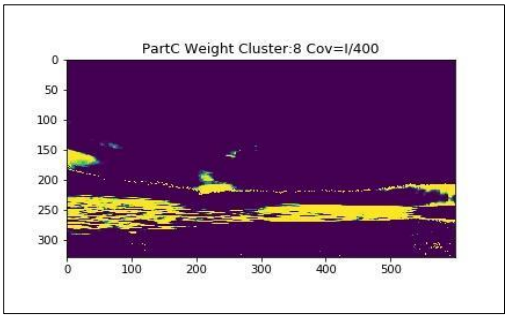


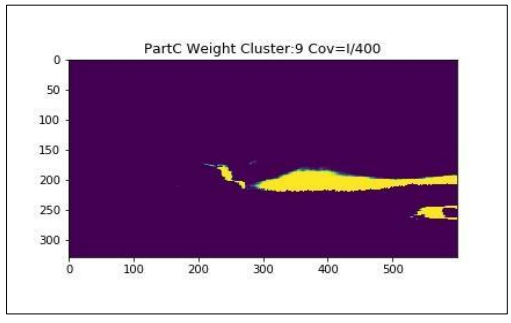












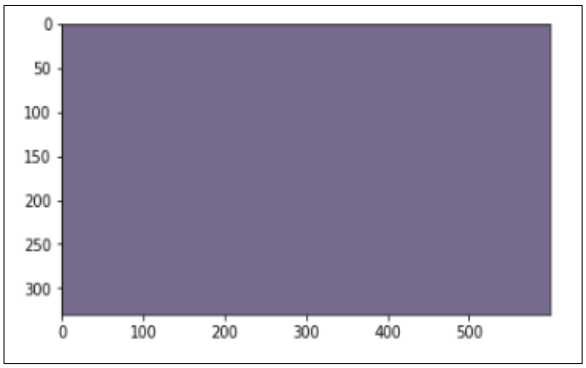
Observation:

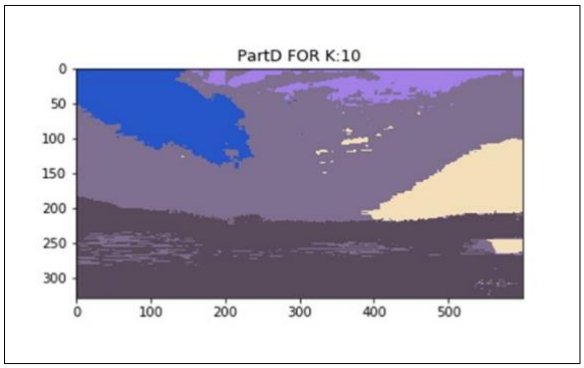
When the covariance matrix is equal to (1/400) * Identity matrix, we observe that the cluster centers don't drift together. Also, the maps of weights linking a given pixel to each cluster center are very different. With the covariance matrix equal to (1/400) * Identity matrix, we are essentially reducing the variance which is making the distributions much slimmer. With narrower distributions, it is easier for the points to shuffle from one Gaussian to another, and hence, they don't all drift together to one cluster.

Part D.

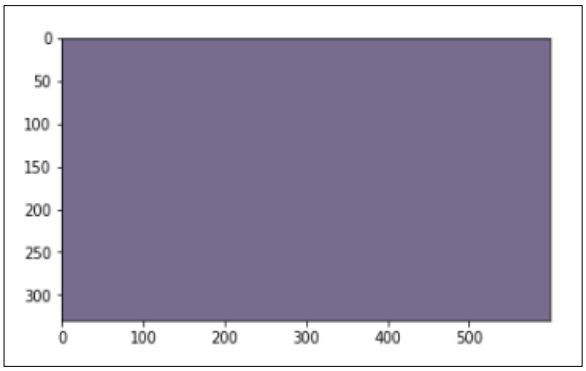
Identity Covariance Matrix | **Data Covariance as Covariance Matrix**

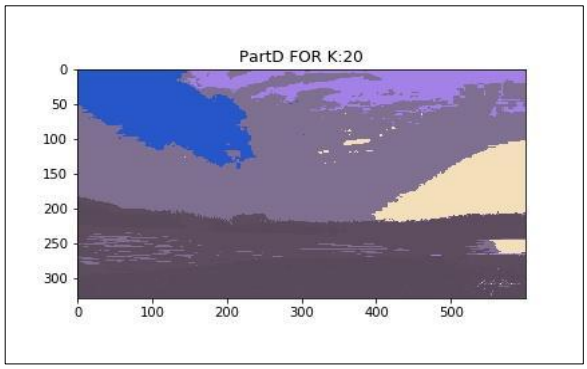
10 Clusters



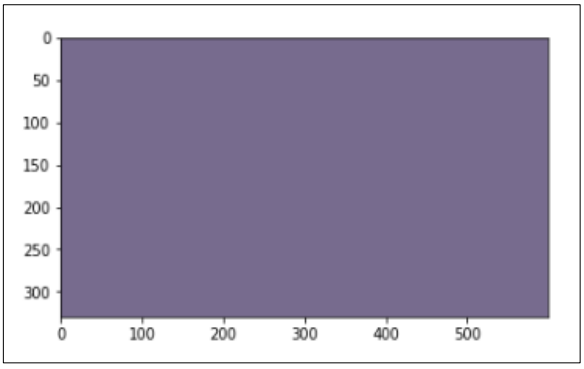


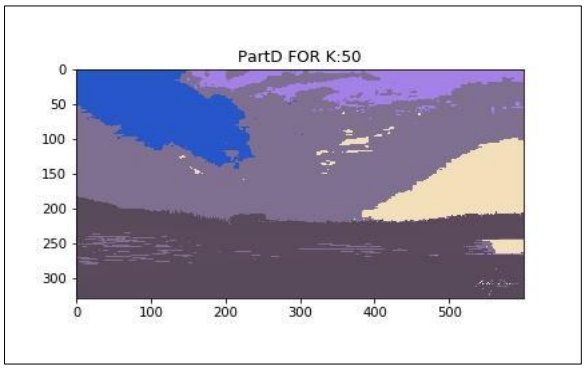
20 Clusters





30 Clusters





Observation:

Unlike what we observed in part-A, we observed that the cluster centers don't drift together when we used the covariance of the data as the covariance matrix. This is because with actual covariance of the data we get Gaussians that are the actual representation of the data. Hence, the pixel values are clustered much better to the Gaussians that are close to their actual representation.

HW5_CS498_SubmissionReport

April 28, 2020

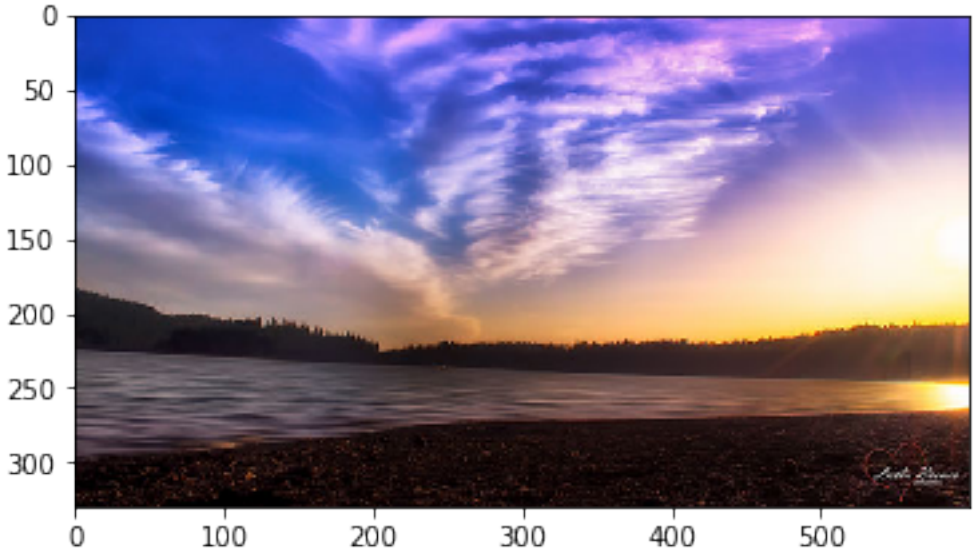
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
# from PIL import Image
import matplotlib.image as mpimg
from sklearn.cluster import KMeans
from sklearn import mixture
from sklearn.mixture import GaussianMixture
from matplotlib import *
from scipy.stats import multivariate_normal
from scipy.spatial import distance
import pickle

import scipy
```

In [2]:
```
img= mpimg.imread('smallsunset.jpg')
plt.figure(figsize=(100,50))
plt.imshow(img)
```

Out[2]: <matplotlib.image.AxesImage at 0x243975b3f08>

In [6]:
```
maxi= np.max(img)
img_n= img/maxi
plt.imshow(img_n)
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x243976f8c88>
```



0.1 Part a.

In [7]:
```
img_r = img_n.reshape(img_n.shape[0]*img_n.shape[1], img_n.shape[2])
img_r.shape
```

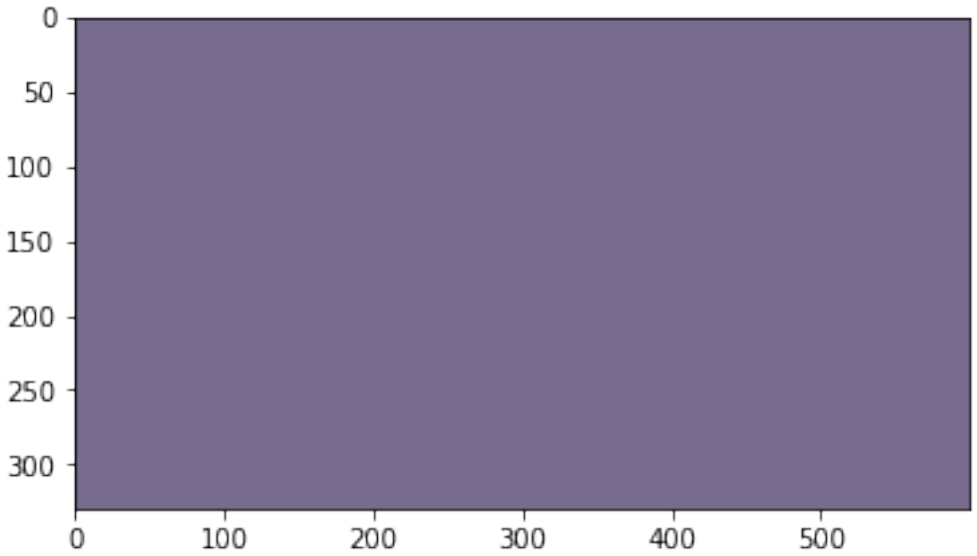
Out[7]:
```
(198000, 3)
```

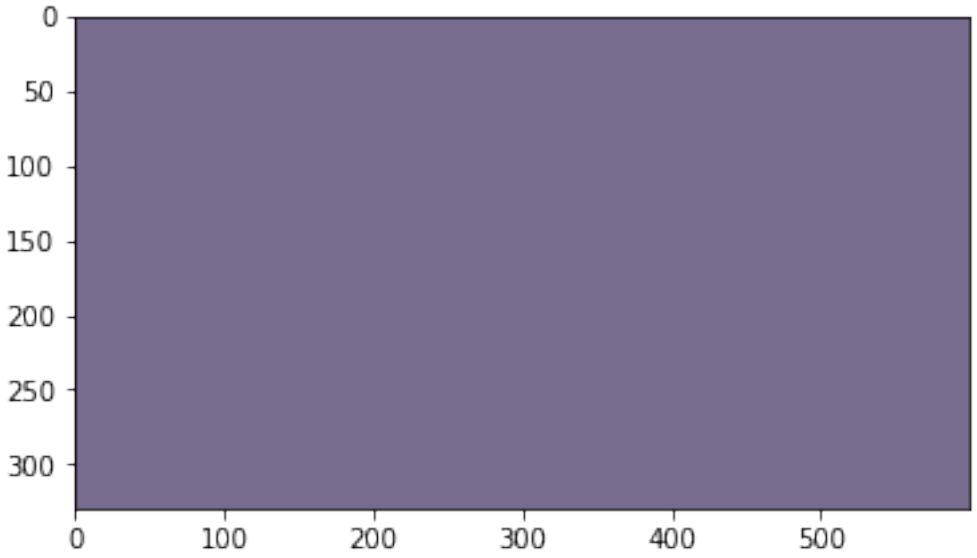
In [ ]:
```
clusters= [10,20,50]
n_epochs= 100
N= len(img_r)
n_f= 3
clusters_result= {}
for clusk in range (0, len(clusters)):
    ##INITIALIZATION
    n_clus= clusters[clusk]
    kmeans= KMeans(n_clusters=n_clus).fit(img_r)
    Centers= kmeans.cluster_centers_
    r= np.zeros([N,n_clus])
    pi= np.ones(n_clus)/n_clus#clusters
    cov_=[]
```

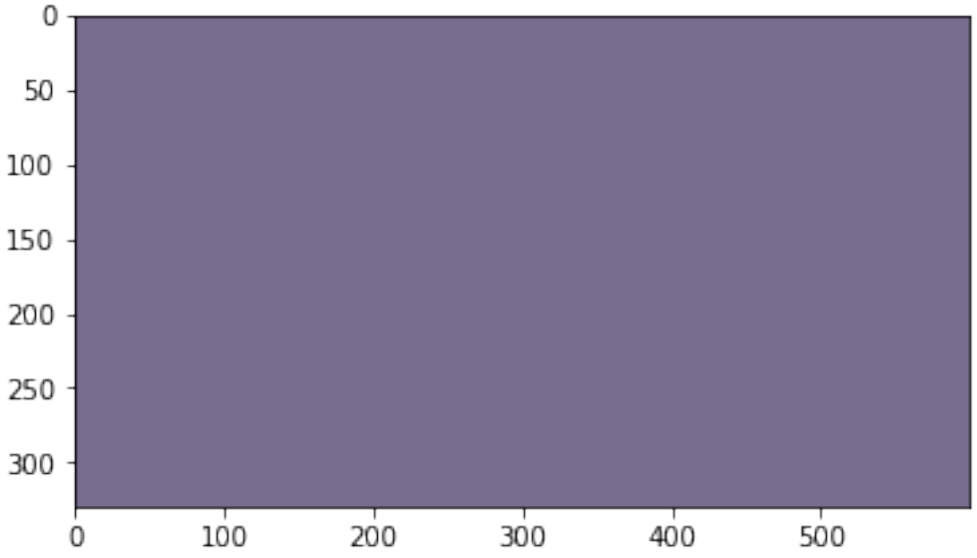
```
for i in range (0, n_clus):
    cov_.append(np.identity(n_f)) ##clustes*features*features

print ('Clustering: ', n_clus)
print ("Weight Dimension: ", r.shape)
print ("pi Dimension: ", pi.shape)
print ("Covariance Dimension: ", np.asarray(cov_).shape)

for epoch in range (0, n_epochs):
    print ('Epoch: ', epoch)
    c_old= Centers.copy()

    ################## E-step:
    mul_gc= {}
    for x in range (0, N):
        mul_g= []
        for k in range (0, n_clus):
              mul_g.append(pi[k]*multivariate_normal.pdf(img_r[x],mean=Centers[k],
        mul_gc[x]= mul_g

    print ('Estimating Weights...')
    ###Weight Estimate
    for x in range (0, N):
        for k in range (0,n_clus):
            r[x,k] =mul_gc[x][k]/sum(mul_gc[x])

    max_pro_index = np.argmax(r, axis=1)
    M= dict([(key, 0) for key in list(range(0, n_clus))])


    print ('Assignments...')
    for sample in range (0, N):
        count= M[max_pro_index[sample]]
        count= count + 1
        M[max_pro_index[sample]]= count

    print (M)

    print ('M-step..')
    ########################### M-step
    for k in range (0, n_clus):
        mc= sum(r[:,k])
        pi[k]= mc/N
        Centers[k] = sum(np.multiply(np.asmatrix(r[:,k]).T,img_r))/mc

    update= distance.euclidean(Centers.flatten(), c_old.flatten())
    print ('Updates', update)
```

```
            if update <= 1e-3:
                break
        ##Saving Parameters
            param= {}
            param['centers']= Centers
            param['r']= r
            param['pi']=pi
            param['updates']= update

            clusters_result[n_clus]= param
            pickleFile = open('clust' + str(n_clus) + '.pickle', 'wb')
            pickle.dump(param,pickleFile,pickle.HIGHEST_PROTOCOL)
            pickleFile.close()
```

```
In [107]: for i in range (0,3):
              dat= clusters_result[clusters[i]]
              r= dat['r']
              Centers= dat['centers']
              max_pro_index = np.argmax(r, axis=1)
              img_mean= np.zeros([N,3])
              for sampl in range (0, N):
                  img_mean[sampl]= Centers[max_pro_index[sampl]]

              img_a= img_mean.reshape(img_n.shape[0], img_n.shape[1], img_n.shape[2])
              # imgplot=plt.imshow(img_mean)
              plt.imshow(img_a)
              plt.savefig('PartA_K' + str(i) + '.jpg')
              plt.show()
```



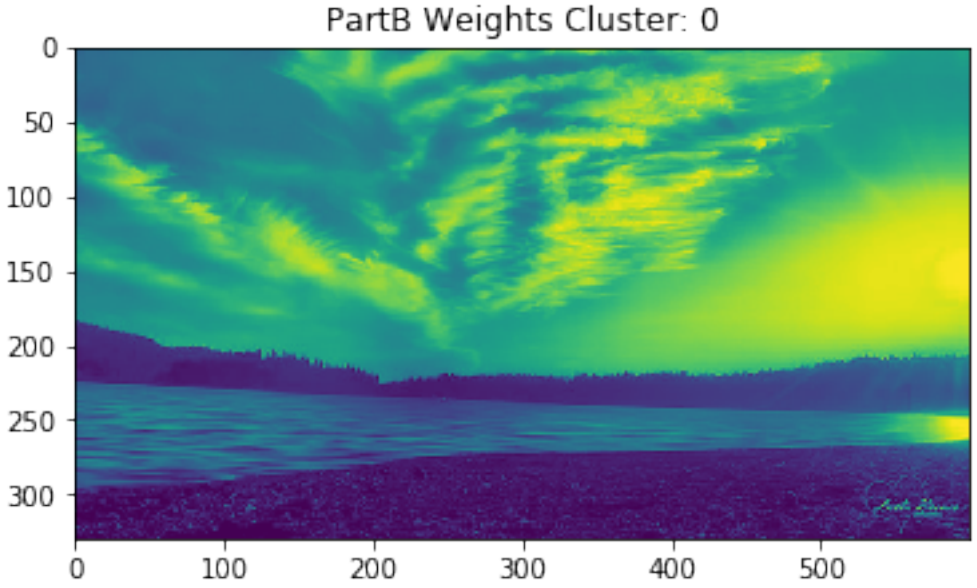


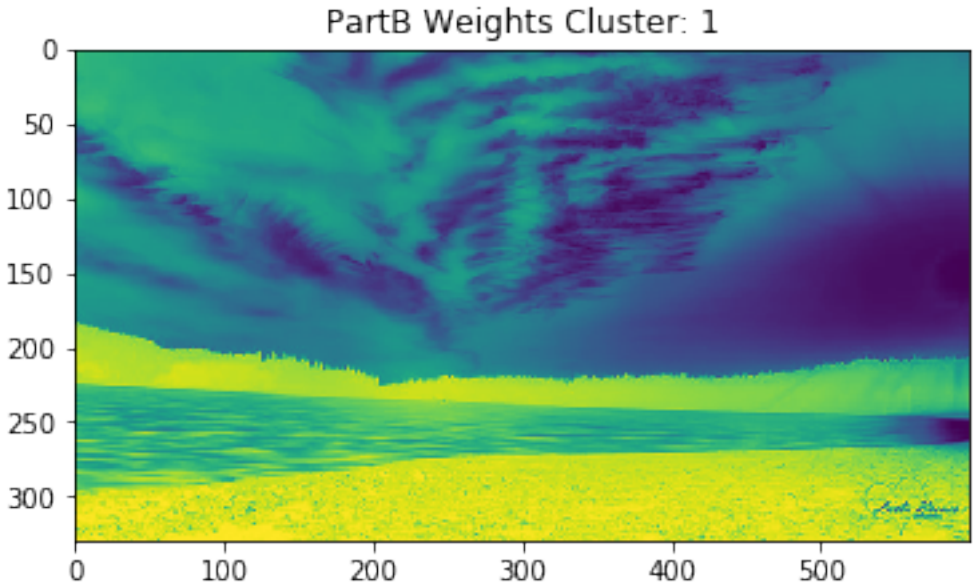


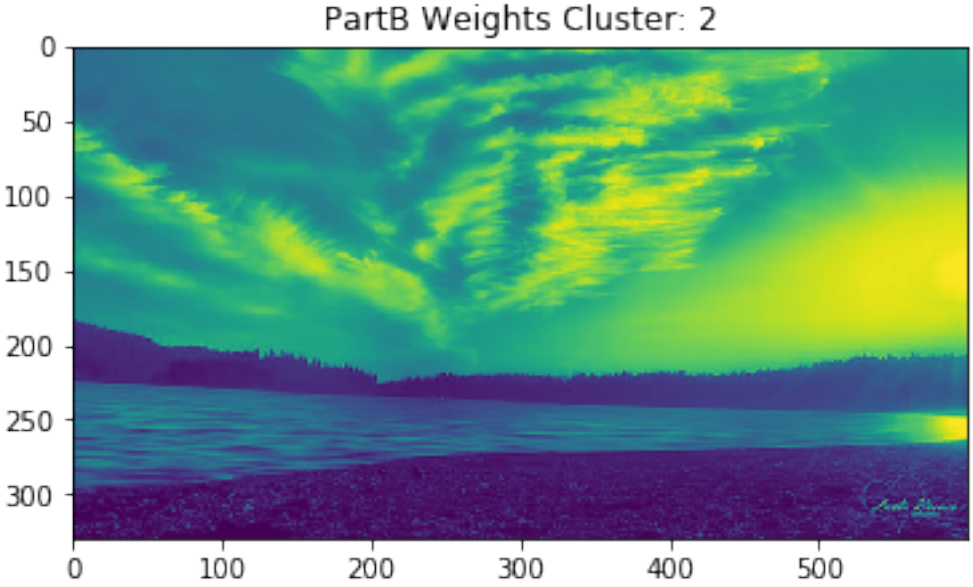
0.2 PART b.

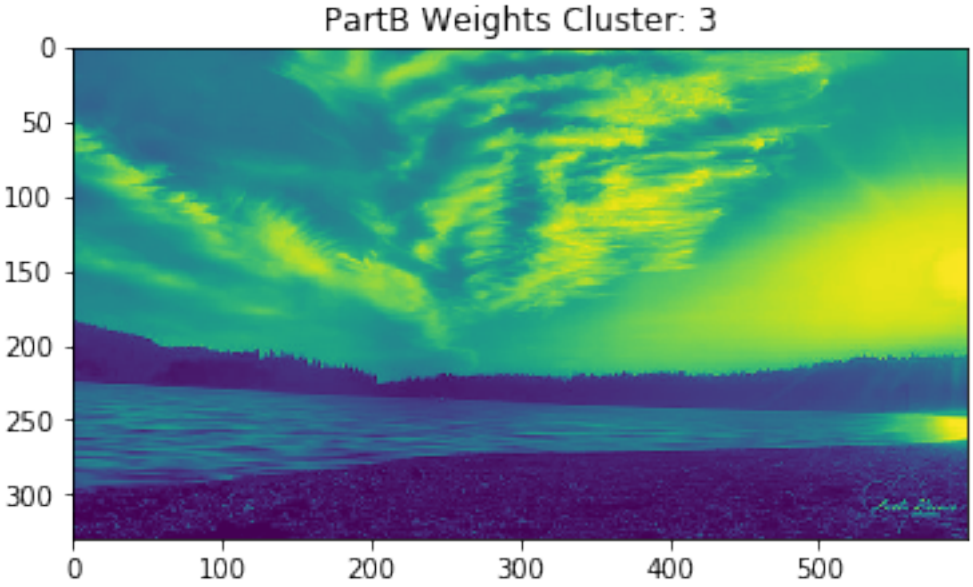
```
In [16]: pickleFile = open('Clust10.pickle', 'rb')
         cluster10 = pickle.load(pickleFile)
         pickleFile.close()
```

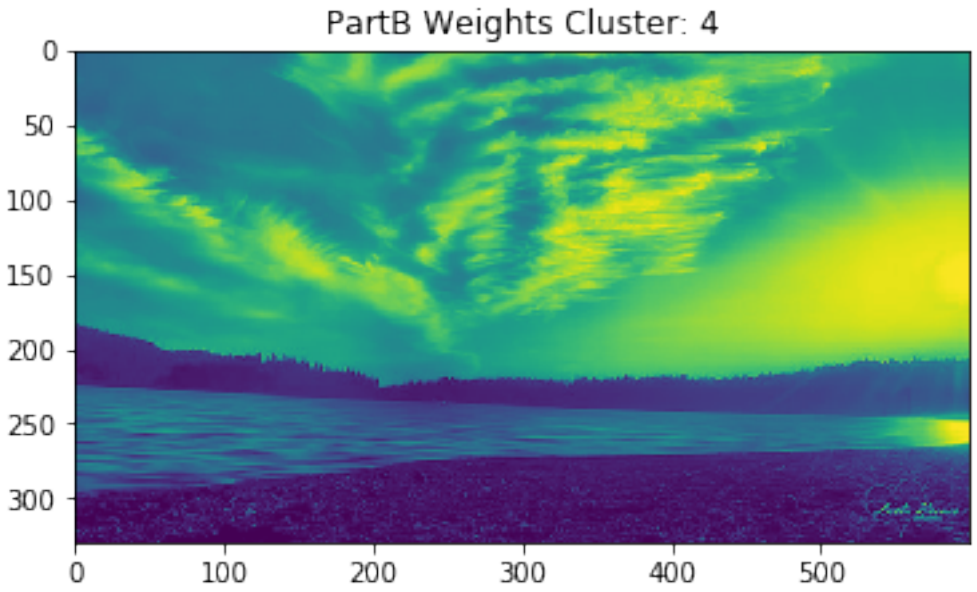
```
In [18]: weights= cluster10['r']

In [19]: for i in range (0,10):
             img_weight= weights[:,i]
             img_plot= img_weight.reshape(img.shape[0], img.shape[1])
             plt.imshow(img_plot)
             plt.title('PartB Weights Cluster: '+str(i))

             plt.savefig('Partb_K' + str(i) + '.jpg')
             plt.show()
```

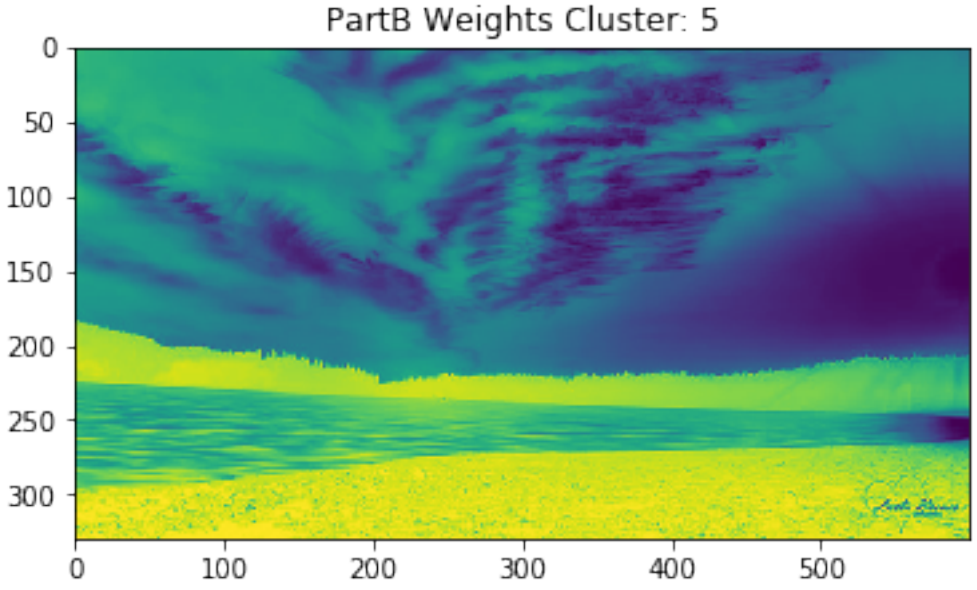


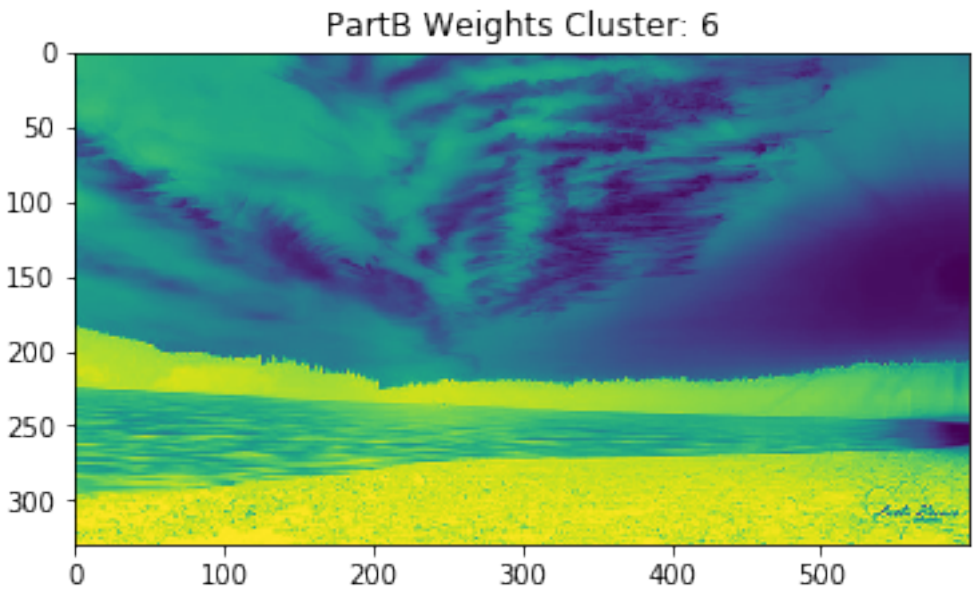


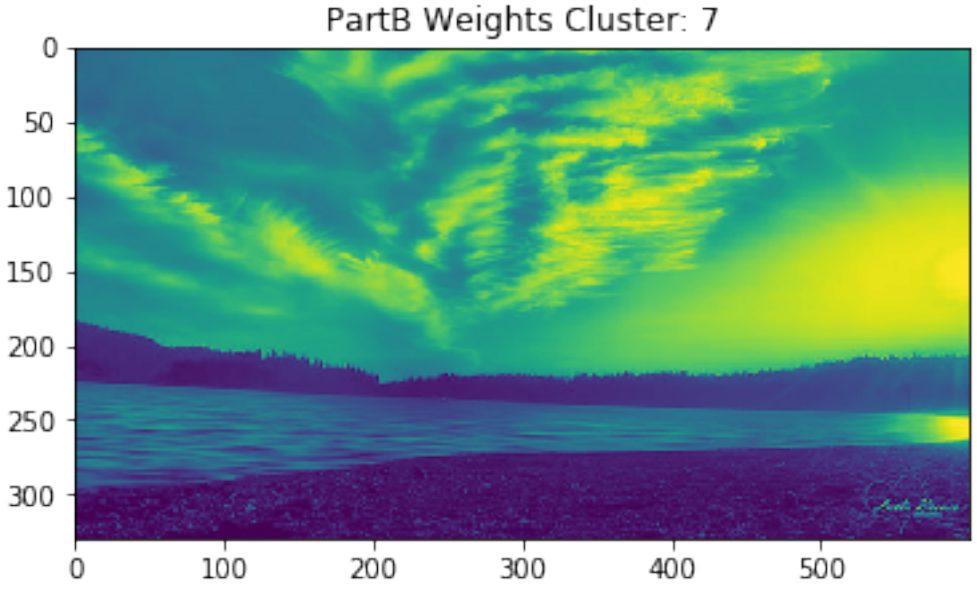

PartB Weights Cluster: 2
0
50
100
150
200
250
300
0
100
200
300
400
500

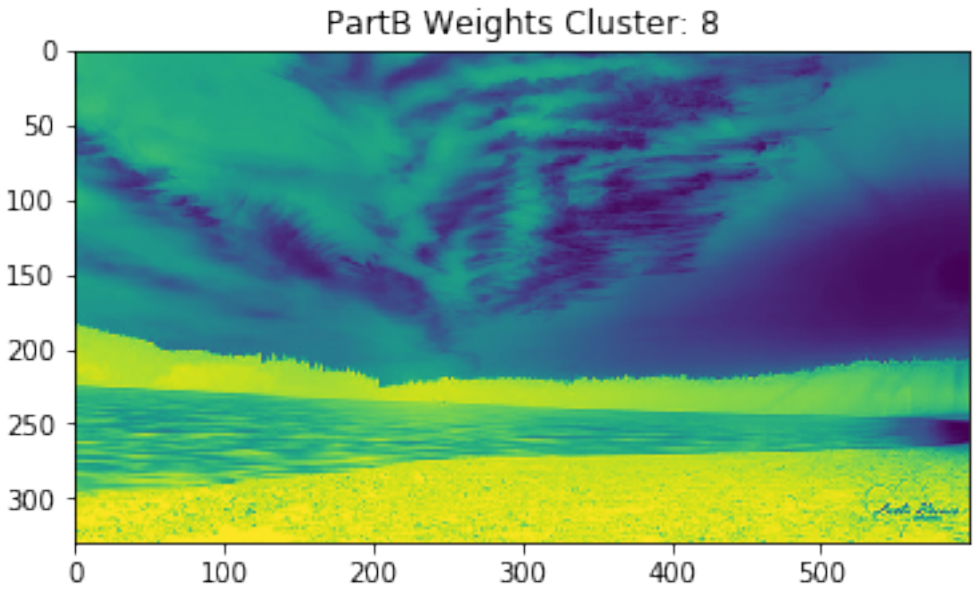

PartB Weights Cluster: 3
0
50
100
150
200
250
300
0
100
200
300
400
500

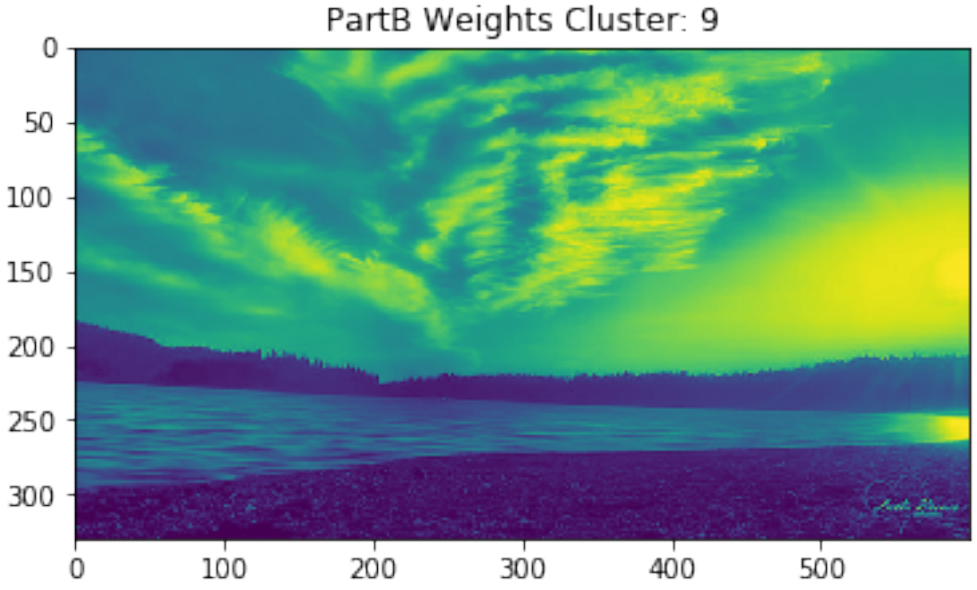

PartB Weights Cluster: 4
0
50
100
150
200
250
300
0
100
200
300
400
500


PartB Weights Cluster: 5
0
50
100
150
200
250
300
0
100
200
300
400
500


PartB Weights Cluster: 6
0
50
100
150
200
250
300
0
100
200
300
400
500


PartB Weights Cluster: 7
0
50
100
150
200
250
300
0
100
200
300
400
500





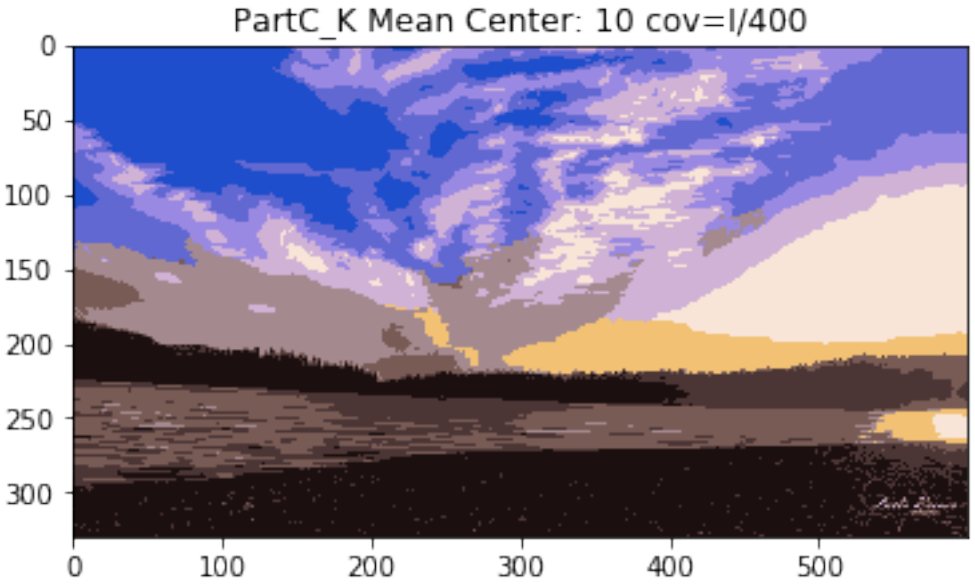
0.3 PART c.

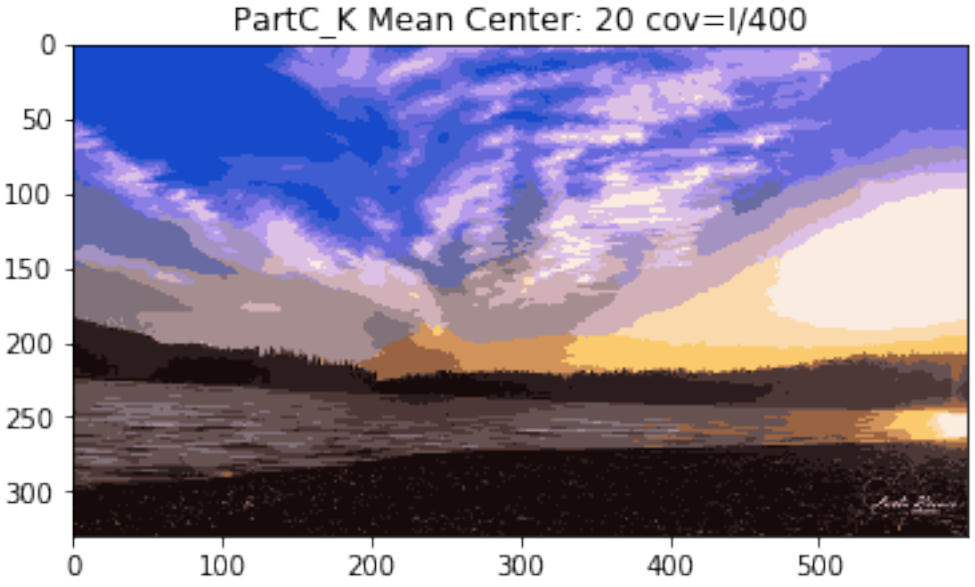
```
In [1]: clusters= [10,20,50]
        n_epochs= 80
        N= len(img_r)
        n_f= 3
```

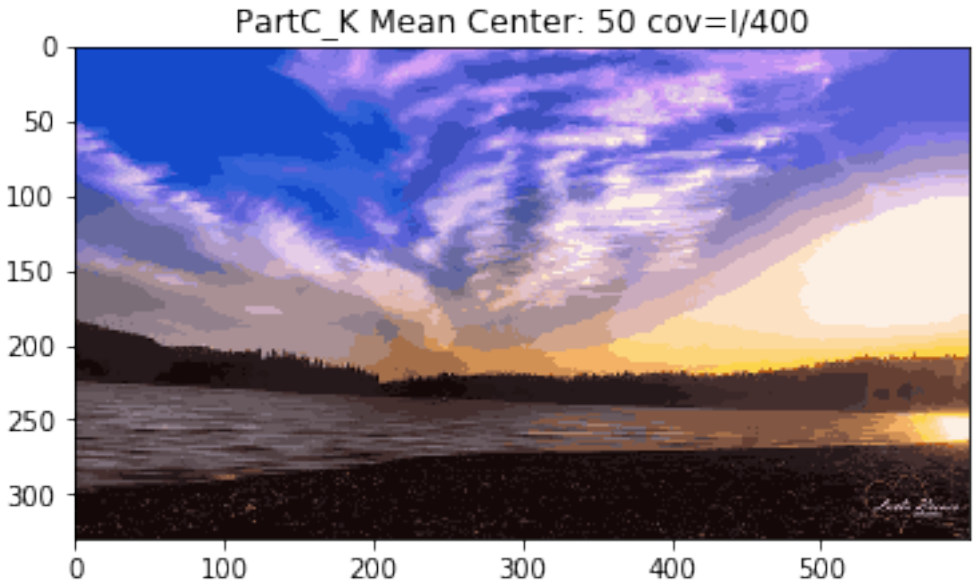
```
clusters_result_c= {}
for clusk in range (0, len(clusters)):
    ##INITIALIZATION
    n_clus= clusters[clusk]
    kmeans= KMeans(n_clusters=n_clus).fit(img_r)
    Centers= kmeans.cluster_centers_
    r= np.zeros([N,n_clus])
    pi= np.ones(n_clus)/n_clus#clusters
    cov_=[]
    for i in range (0, n_clus):
        cov_.append((1/400)*np.identity(n_f)) ##clustes*features*features

    print ('Clustering: ', n_clus)
    print ("Weight Dimension: ", r.shape)
    print ("pi Dimension: ", pi.shape)
    print ("Covariance Dimension: ", np.asarray(cov_).shape)

    for epoch in range (0, n_epochs):
        print ('Epoch: ', epoch)
        c_old= Centers.copy()

        ################## E-step:
        mul_gc= {}
        for x in range (0, N):
            mul_g= []
            for k in range (0, n_clus):
                  mul_g.append(pi[k]*multivariate_normal.pdf(img_r[x],mean=Centers[k],
            mul_gc[x]= mul_g

        print ('Estimating Weights...')
        ###Weight Estimate
        for x in range (0, N):
            for k in range (0,n_clus):
                r[x,k] =mul_gc[x][k]/sum(mul_gc[x])

        max_pro_index = np.argmax(r, axis=1)
        M= dict([(key, 0) for key in list(range(0, n_clus))])
        print ('Assignments...')
        for sample in range (0, N):
            count= M[max_pro_index[sample]]
            count= count + 1
            M[max_pro_index[sample]]= count

        print (M)

        print ('M-step..')
        ########################## M-step
        for k in range (0, n_clus):
```

```
                mc= sum(r[:,k])
                pi[k]= mc/N
                Centers[k] = sum(np.multiply(np.asmatrix(r[:,k]).T,img_r))/mc

            update= distance.euclidean(Centers.flatten(), c_old.flatten())
            print ('Updates', update)
#             print (r[:10])

            if update <= 1e-3:
                break

        ##Saving Parameters
            param= {}
            param['centers']= Centers
            param['r']= r
            param['pi']=pi
            param['updates']= update
            param['iteration']= epoch

        #     clusters_result_idfix_c[n_clus]= param
            pickleFile = open('PartC_Identfixed_clust' + str(n_clus) + '.pickle', 'wb')
            pickle.dump(param,pickleFile,pickle.HIGHEST_PROTOCOL)
            pickleFile.close()


In [8]: results= []
        pickleFile = open('PartC_Identfixed_clust10.pickle', 'rb')
        clus10_c = pickle.load(pickleFile)
        pickleFile.close()
        results.append(clus10_c)

In [9]: pickleFile = open('PartC_Identfixed_clust20.pickle', 'rb')
        clus20_c = pickle.load(pickleFile)
        pickleFile.close()
        results.append(clus20_c)

In [10]: pickleFile = open('PartC_Identfixed_clust50.pickle', 'rb')
         clus50_c = pickle.load(pickleFile)
         pickleFile.close()
         results.append(clus50_c)

In [11]: #PRINTING MEAN
         N= len(img_r)
         for i in range (0,3):
             dictio= results[i]
             r= dictio['r']
             Centers= dictio['centers']
             max_pro_index = np.argmax(r, axis=1)
             img_mean= np.zeros([N,3])
```

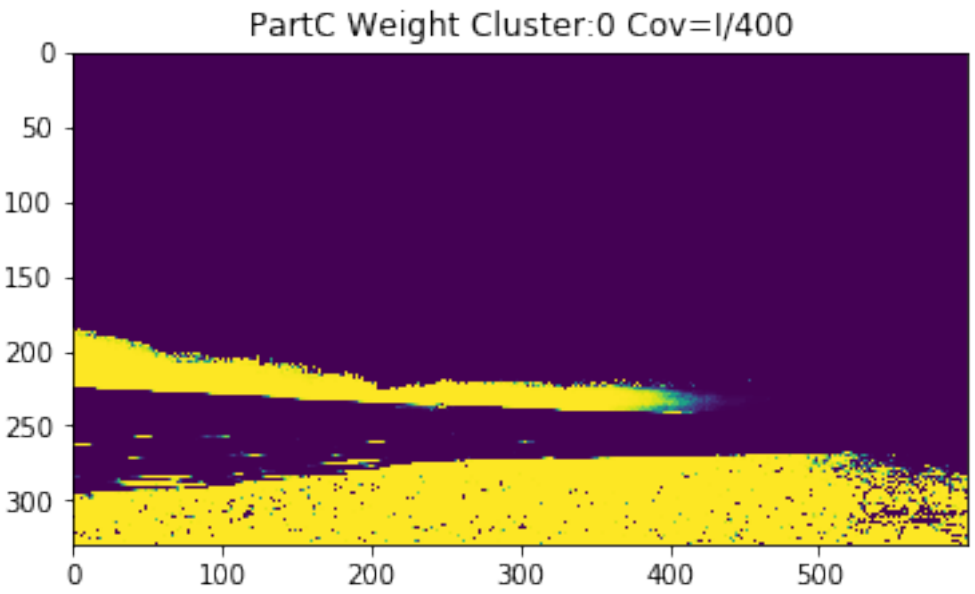
```
for sampl in range (0, N):
    img_mean[sampl]= Centers[max_pro_index[sampl]]

img_a= img_mean.reshape(img_n.shape[0], img_n.shape[1], img_n.shape[2])
# imgplot=plt.imshow(img_mean)
plt.imshow(img_a)
plt.title('PartC_K Mean Center: '+str(len(Centers)) + ' cov=I/400')
plt.savefig('PartC_K' + str(len(Centers)) + 'idfixed.jpg')
plt.show()
```

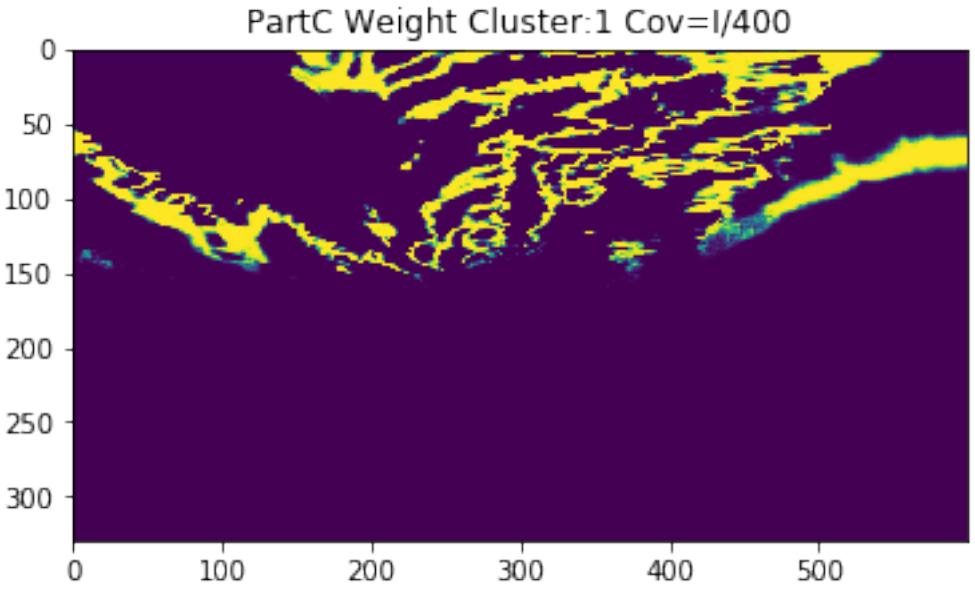


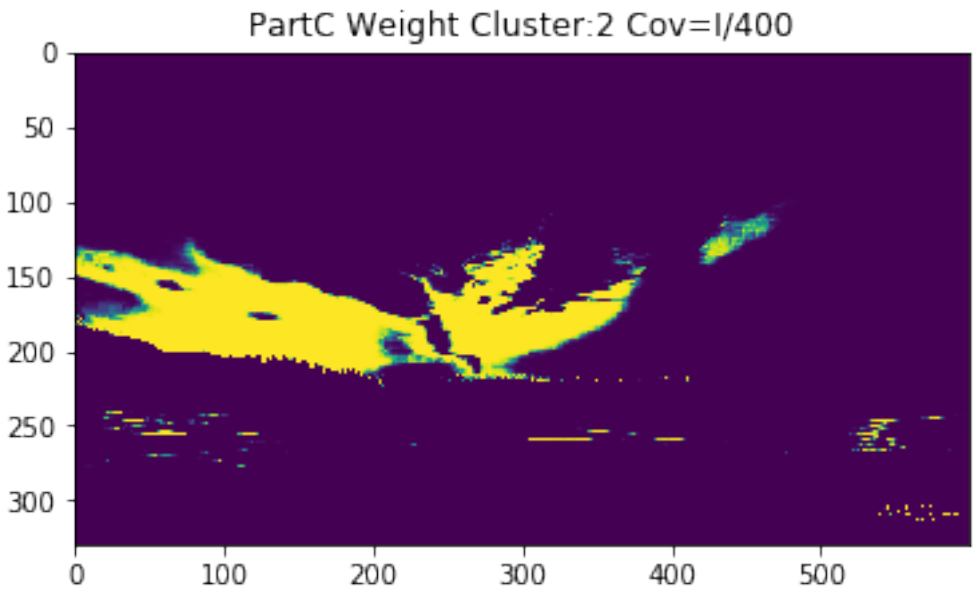


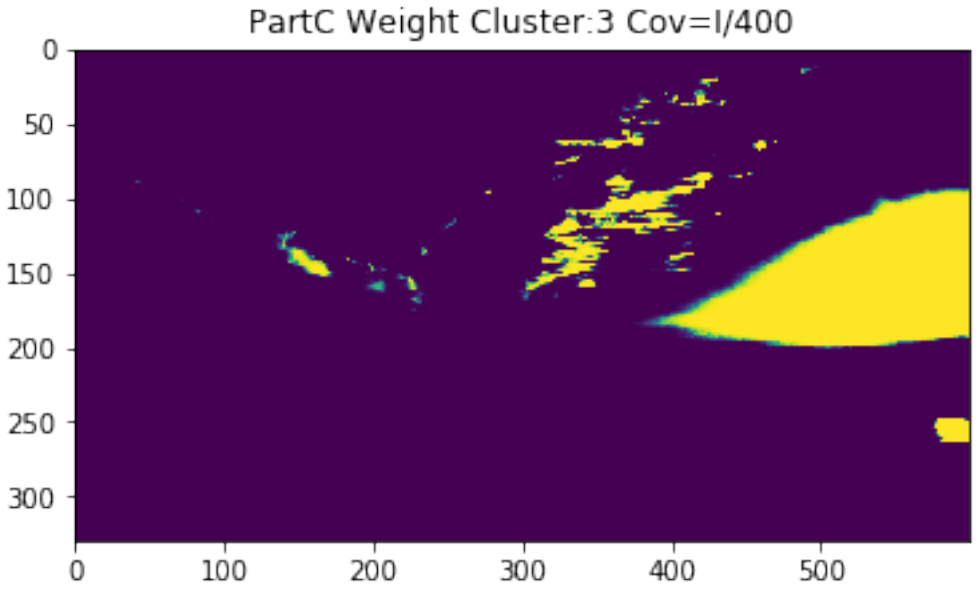


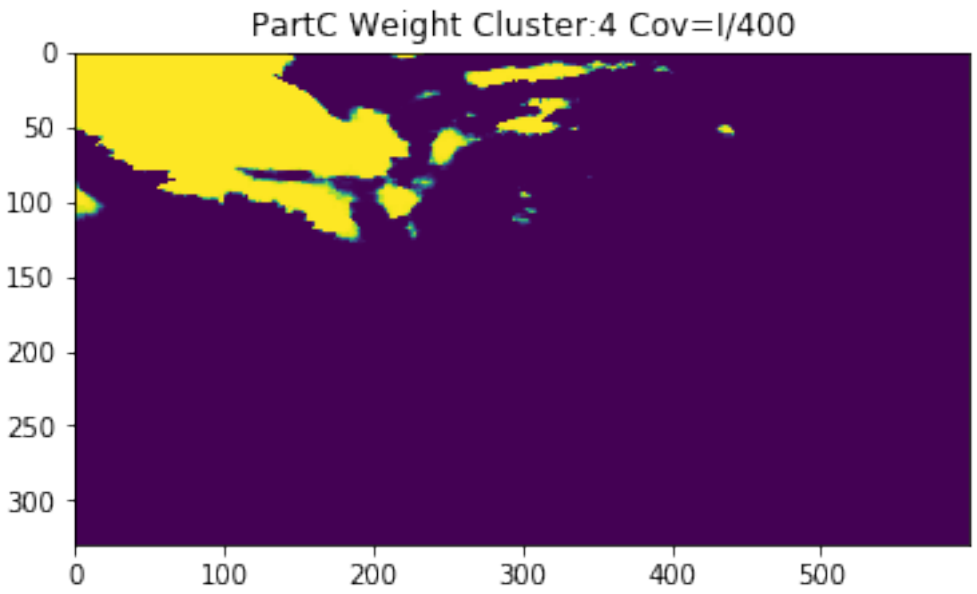

```
In [13]: weights= clus10_c['r']
         for i in range (0,10):
             img_weight= weights[:,i]
             img_plot= img_weight.reshape(img.shape[0], img.shape[1])
             plt.imshow(img_plot)
             plt.title('PartC Weight Cluster:' + str(i) + ' Cov=I/400')
             plt.savefig('Partc_K' + str(i) + '.jpg')
             plt.show()
```

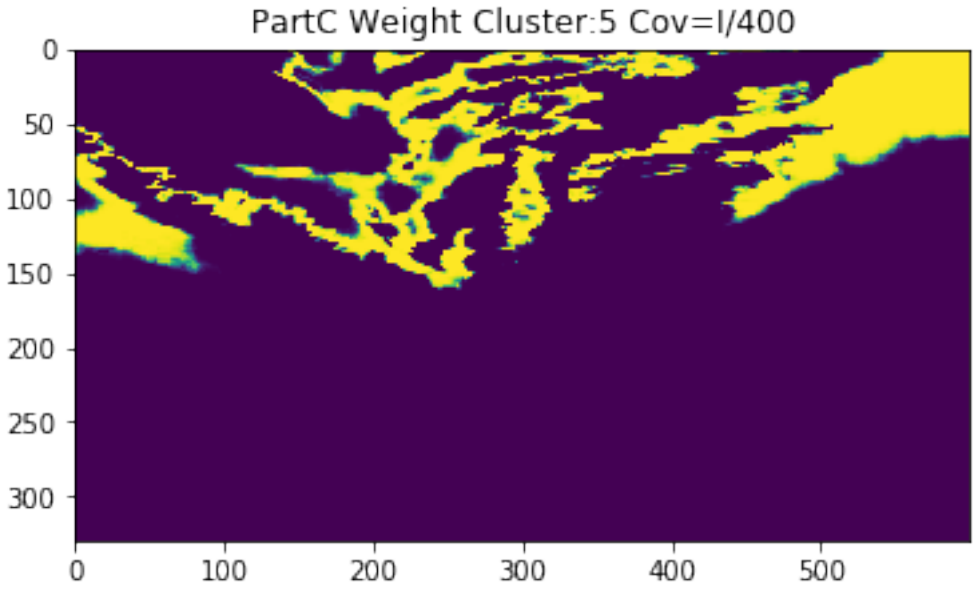

PartC Weight Cluster:0 Cov=I/400
0
50
100
150
200
250
300
0
100
200
300
400
500

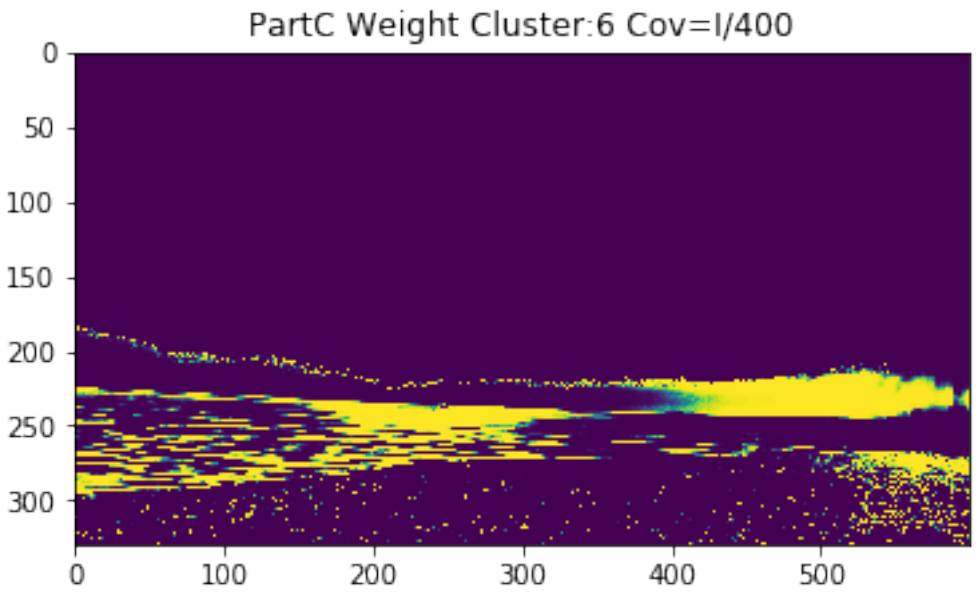

PartC Weight Cluster:1 Cov=I/400
0
50
100
150
200
250
300
0
100
200
300
400
500

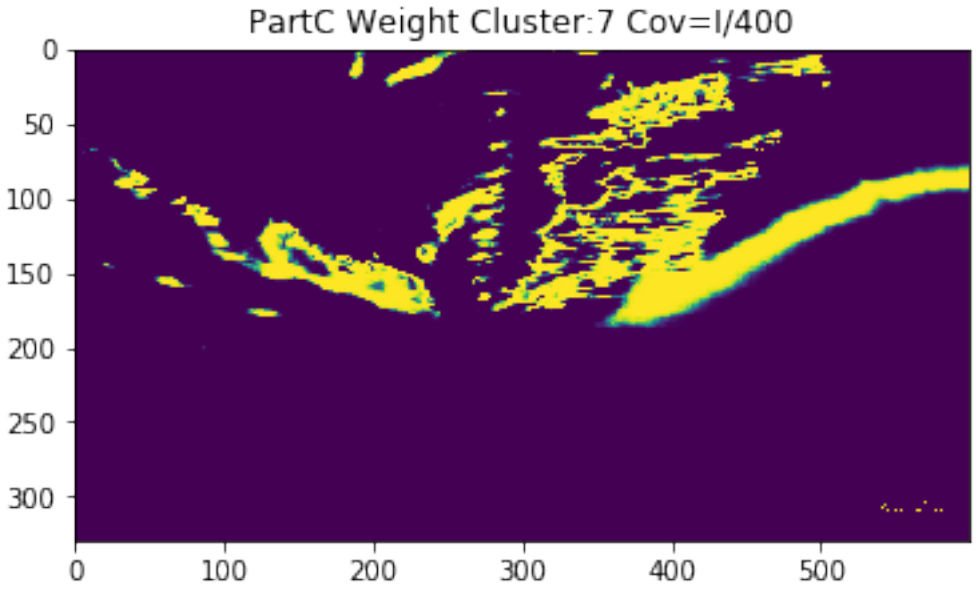

PartC Weight Cluster:2 Cov=I/400
0
50
100
150
200
250
300
0
100
200
300
400
500

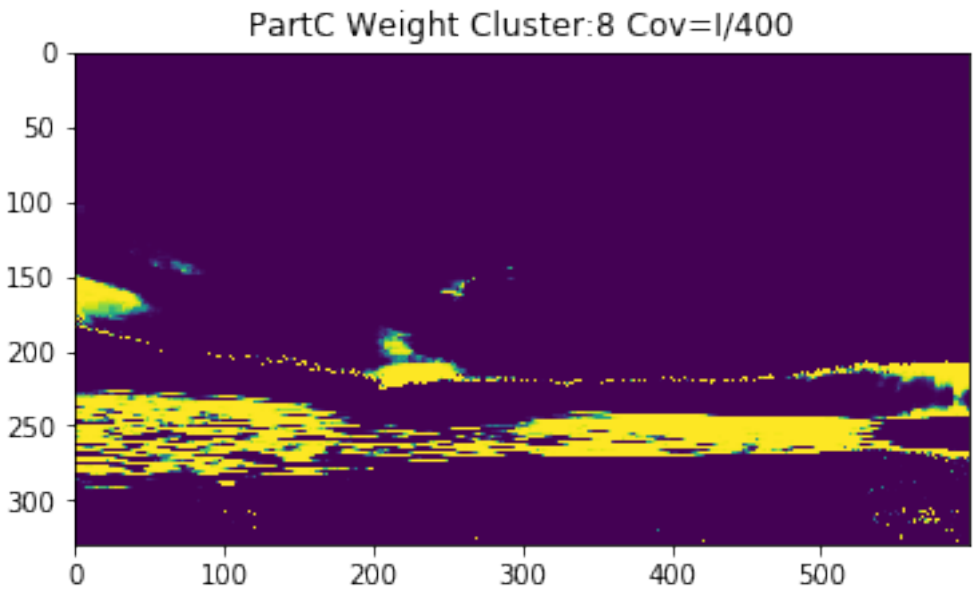

PartC Weight Cluster:3 Cov=I/400
0
50
100
150
200
250
300
0
100
200
300
400
500

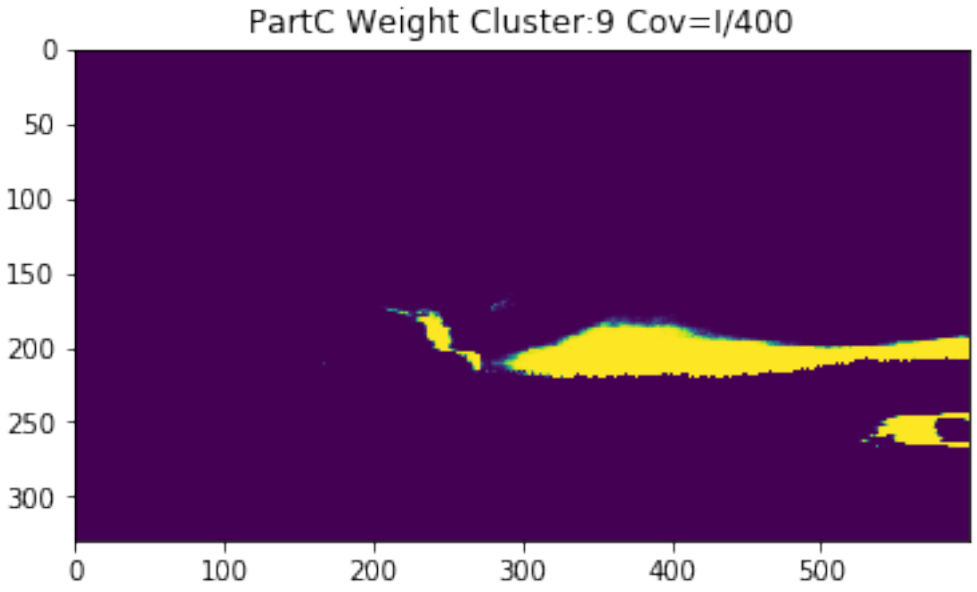

PartC Weight Cluster:4 Cov=I/400
0
50
100
150
200
250
300
0
100
200
300
400
500


PartC Weight Cluster:5 Cov=I/400
0
50
100
150
200
250
300
0
100
200
300
400
500


PartC Weight Cluster:6 Cov=I/400
0
50
100
150
200
250
300
0
100
200
300
400
500


PartC Weight Cluster:7 Cov=I/400
0
50
100
150
200
250
300
0
100
200
300
400
500



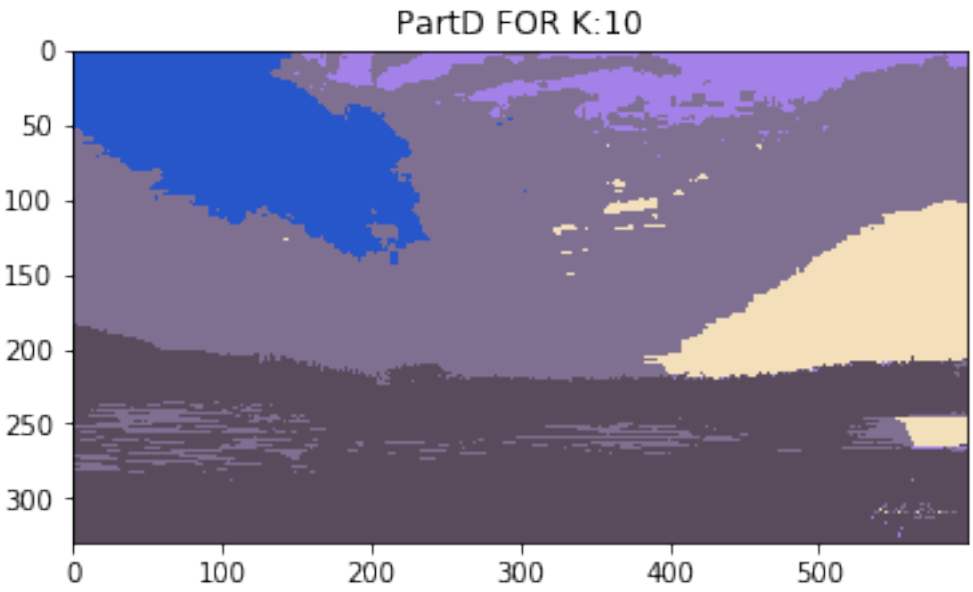


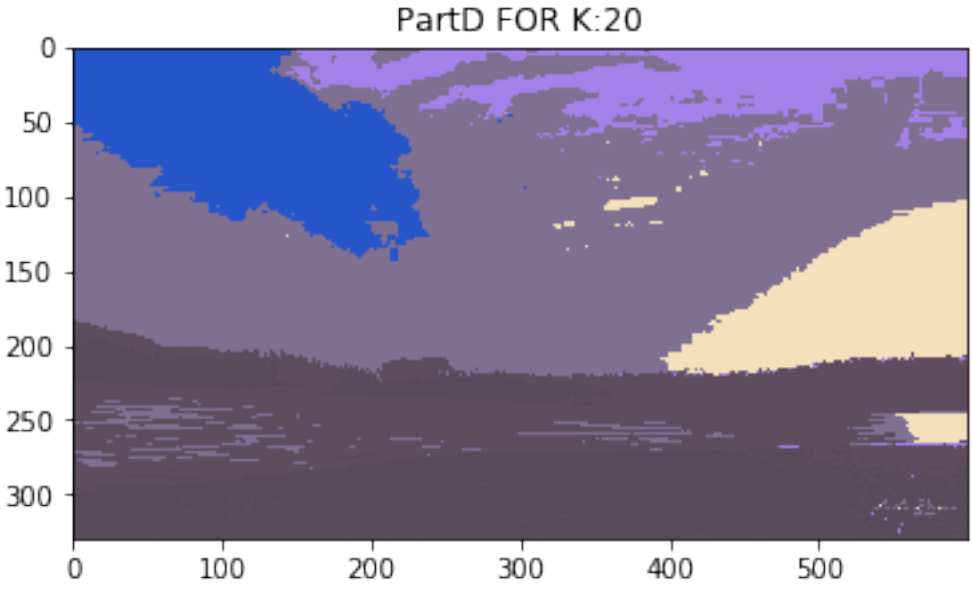
0.4 PART d.

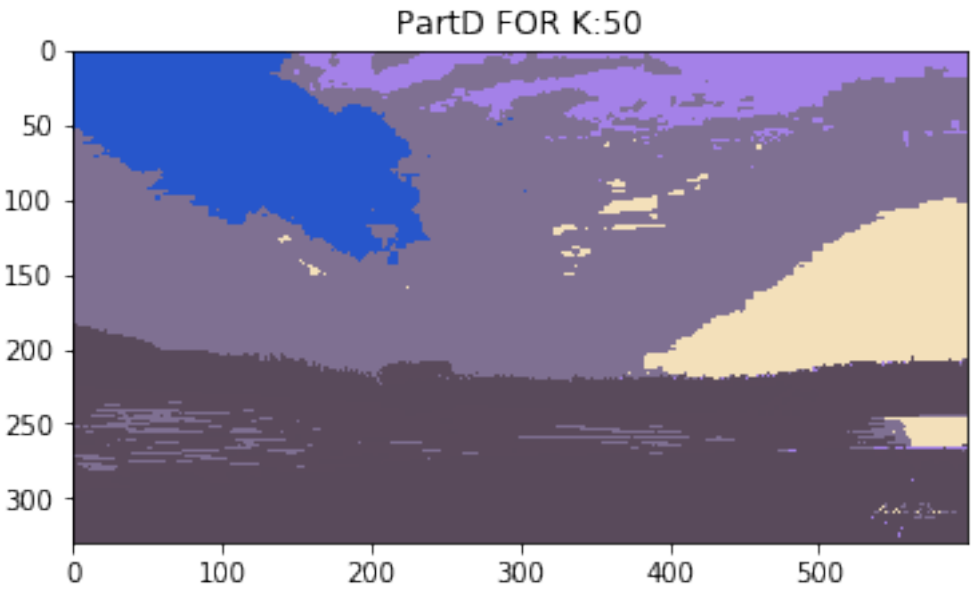
```
In [23]: #covariance
         x= img_r[:,0]
         y= img_r[:,1]
         z= img_r[:,2]
```

```
        data = np.array([x,y,z])
        cov= np.cov(data)

In [2]: clusters= [10,20,50]
        n_epochs= 100
        N= len(img_r)
        n_f= 3
        clusters_result_d= {}
        for clusk in range (0, len(clusters)):
            ##INITIALIZATION
            n_clus= clusters[clusk]
            kmeans= KMeans(n_clusters=n_clus).fit(img_r)
            Centers= kmeans.cluster_centers_
            r= np.zeros([N,n_clus])
            pi= np.ones(n_clus)/n_clus#clusters
            cov_=[]
            for i in range (0, n_clus):
                cov_.append(cov) ##clustes*features*features

            print ('Clustering: ', n_clus)
            print ("Weight Dimension: ", r.shape)
            print ("pi Dimension: ", pi.shape)
            print ("Covariance Dimension: ", np.asarray(cov_).shape)

            for epoch in range (0, n_epochs):
                print ('Epoch: ', epoch)
                c_old= Centers.copy()

                ################## E-step:
                mul_gc= {}
                for x in range (0, N):
                    mul_g= []
                    for k in range (0, n_clus):
                          mul_g.append(pi[k]*multivariate_normal.pdf(img_r[x],mean=Centers[k],
                    mul_gc[x]= mul_g

                print ('Estimating Weights...')
                ###Weight Estimate
                for x in range (0, N):
                    for k in range (0,n_clus):
                        r[x,k] =mul_gc[x][k]/sum(mul_gc[x])

                max_pro_index = np.argmax(r, axis=1)
                M= dict([(key, 0) for key in list(range(0, n_clus))])
                print ('Assignments...')
                for sample in range (0, N):
                    count= M[max_pro_index[sample]]
                    count= count + 1
```

```
                M[max_pro_index[sample]]= count

            print (M)

            print ('M-step..')
            ########################## M-step
            for k in range (0, n_clus):
                mc= sum(r[:,k])
                pi[k]= mc/N
                Centers[k] = sum(np.multiply(np.asmatrix(r[:,k]).T,img_r))/mc

            update= distance.euclidean(Centers.flatten(), c_old.flatten())
            print ('Updates', update)

            if update <= 1e-3:
                break

        ##Saving Parameters
            param= {}
            param['centers']= Centers
            param['r']= r
            param['pi']=pi
            param['updates']= update
            param['iteration']= epoch

            clusters_result_d[n_clus]= param
            pickleFile = open('PartD_clust' + str(n_clus) + '.pickle', 'wb')
            pickle.dump(param,pickleFile,pickle.HIGHEST_PROTOCOL)
            pickleFile.close()


In [63]: for i in range (0,3):
             dat= clusters_result_d[clusters[i]]
             r= dat['r']
             Centers= dat['centers']
             max_pro_index = np.argmax(r, axis=1)
             img_mean= np.zeros([N,3])
             for sampl in range (0, N):
                 img_mean[sampl]= Centers[max_pro_index[sampl]]

             img_a= img_mean.reshape(img_n.shape[0], img_n.shape[1], img_n.shape[2])
             # imgplot=plt.imshow(img_mean)
             plt.imshow(img_a)
             plt.title('PartD FOR K:' + str(clusters[i]))

             plt.savefig('Partd_K' + str(i) + '.jpg')
             plt.show()
```


PartD FOR K:10
0
50
100
150
200
250
300
0
100
200
300
400
500


PartD FOR K:20
0
50
100
150
200
250
300
0
100
200
300
400
500


PartD FOR K:50
0
50
100
150
200
250
300
0
100
200
300
400
500