

# CEE 498 Applied Machine Learning - HW4

Rini Jasmine Gladstone (rjg7) and Maksymillian Podraza(podraza3)

March 30, 2020

## 1 Problem 1

### 1.1 Part a

We followed the following steps for this part.

- (a) Built a function for creating 10 x 10 patches and extracting these patches from a 4x4 grid of an image

```
1 #Creating 4x4 grid of 10x10 patches
2 def patch_creation(image):
3     patches = []
4     x=[0,6,12,18]
5     y=[0,6,12,18]
6     for i in x:
7         for j in y:
8             patch = image[i:i+10,j:j+10].reshape(-1,100)
9             patches.append(patch)
10    return(patches)
```

- (b) Created patches for all 60,000 training images.

```
1 #reshaping each 784 1-d array into 28x28 2-d array and
   creating patches for each image
2 training_patches=[]
3 for i in range(60000):
4     image = train_set[0][i].reshape(28,28)
5     training_patches.append(patch_creation(image))
```

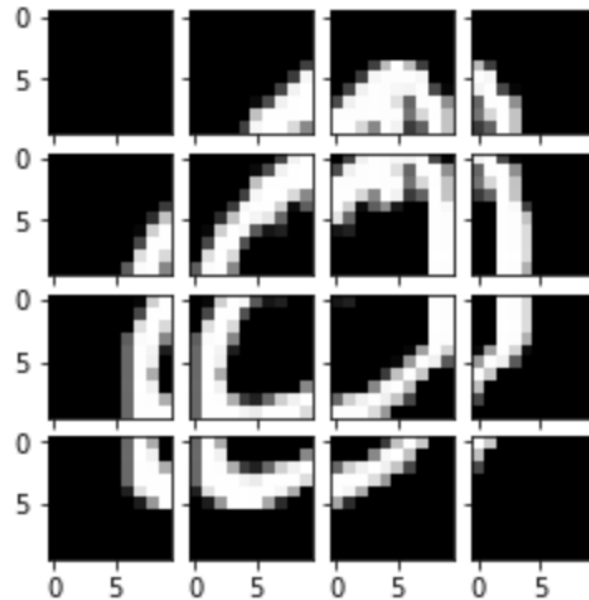
```
print("Amount of training images: ", len(training_patches))
('Amount of training images: ', 60000)
```

```
print("Amount of matrices (patches) in each training image: ", len(training_patches[0]))
('Amount of matrices (patches) in each training image: ', 16)
```

An example of the train image with the patches is as shown below.

```
test_image()
```

```
('Image shape: ', (28, 28))  
('Num patches: ', 16)
```



- (c) For each training image, one of these patches is chosen uniformly and at random. This dataset of 60,000 patches is sub-sampled uniformly and at random to produce a 6000 element dataset

```
1 #Choosing 1 patch from the 16 previously created for each  
   image at random and appending to training list  
2 train_cluster=[]  
3 for i in range(60000):  
4     n = random.randint(0, 15)  
5     train_cluster.append(training_patches[i][n])  
6 #creates a list of 6000 numbers where each is between 0 -  
   59999  
7 indices = random.sample(range(60000), 6000)  
8 #appending randomly sampled patches using the previously  
   created list of indices  
9 train_cluster_sample=[]  
10 for ind in indices:  
11     train_cluster_sample.append(train_cluster[ind])  
  
print("Length of our randomly sampled patches: ", len(train_cluster_sample))  
('Length of our randomly sampled patches: ', 6000)
```

(d) The above dataset is clustered to 50 centres.

```
1 #creating a 6000x100 matrix of zeros that where each row will
   be one of our samples
2 training_matrix = np.zeros((6000,100))
3 for i in range(6000):
4     #making eacch row a random patch
5     training_matrix[i,:] = train_cluster_sample[i][0]
6 # create kmeans object
7 kmeans = KMeans(n_clusters=50)
8 # fit kmeans object to data
9 kmeans.fit(training_matrix)
10 # save new clusters for chart
11 y_km = kmeans.predict(training_matrix)
```

(e) Assigned the closest cluster to all the patches of the 60,000 training images.

```
1 training_matrix_60k = np.zeros((60000,100))
2 for i in range(60000):
3     training_matrix_60k[i,:] = train_cluster[i][0]
4 y_km_60k = kmeans.predict(training_matrix_60k)
```

(f) Subset datasets for each cluster centre and cluster them again to 50 clusters.

```
1 cluster_dict = {}
2 for i in range(50):
3     key = str(i)
4     subtrain_matrix = training_matrix_60k[y_km_60k==i,:]
5     kmeans_sub = KMeans(n_clusters=50)
6     kmeans_sub.fit(subtrain_matrix)
7     cluster_dict[key] = kmeans_sub
8     y_km_sub = kmeans_sub.predict(subtrain_matrix)
```

## 1.2 Part b

We followed the following steps for this part.

(a) Built a function for padding the images

```
1 def pad_with(vector, pad_width, iaxis, kwargs):
2     pad_value = kwargs.get('padder', 0)
3     vector[:pad_width[0]] = pad_value
4     vector[-pad_width[1]:] = pad_value
```

(b) Built a function for creating 9 sub-patches for each of the 10x10 patch in the 4x4 grid.

```
1 def patch_b(image):
2     patches16 = []
3     x_ = [0, 1, 2]
4     y_ = [0, 1, 2]
5     x=[0,6,12,18]
6     y=[0,6,12,18]
7     for i in x:
```

```

8         for j in y:
9             patches9 = []
10            for k in x_:
11                for l in y_:
12                    patch = image[i+k:i+k+10, j+l:j+l+10].
13                        reshape(-1,100)
14                    patches9.append(patch)
15                patches16.append(patches9)
16            return patches16

```

- (c) The sub patches were created for 10,000 testing images and 6,000 training images.

```

1 testing_patches = []
2 testing_labels = []
3 for i in range(10000):
4     testing_resaped = test_set[0][i].reshape(28,28)
5     testing_labels.append(test_set[1][i])
6     testing_padded = np.pad(testing_resaped, 1, pad_with)
7     testing_patched = patch_b(testing_padded)
8     testing_patches.append(testing_patched)

```

```

print("Shape of padded image: ", testing_padded.shape)
print("Number of original patches: ", len(testing_patched))
print("Number of recentered patches per original patch: ", len(testing_patched[0]))
print("Total number of patches for each image: ", len(testing_patched) * len(testing_patched[0]))

('Shape of padded image: ', (30, 30))
('Number of original patches: ', 16)
('Number of recentered patches per original patch: ', 9)
('Total number of patches for each image: ', 144)

```

```

1 training_patches = []
2 random_indices = random.sample(range(60000), 6000)
3 training_labels = []
4 for i in random_indices:
5     training_resaped = train_set[0][i].reshape(28,28)
6     training_labels.append(train_set[1][i])
7     training_padded = np.pad(training_resaped, 1, pad_with)
8     training_patched = patch_b(training_padded)
9     training_patches.append(training_patched)

```

```

print("Shape of padded image: ", training_padded.shape)
print("Number of original patches: ", len(training_patched))
print("Number of recentered patches per original patch: ", len(training_patched[0]))
print("Total number of patches for each image: ", len(training_patched) * len(training_patched[0]))

('Shape of padded image: ', (30, 30))
('Number of original patches: ', 16)
('Number of recentered patches per original patch: ', 9)
('Total number of patches for each image: ', 144)

```

- (d) Then, we built a function for finding the cluster and sub cluster for each of the sub patches.

```

1 def finding_cluster(data,num_images=10000,num_grids=16,
2                     num_patches=9):
3     data_matrix_long = np.zeros((num_images*num_grids*
4                                   num_patches,100))
5     z=0
6     for i in range(num_images):

```

```

5         for j in range(num_grids):
6             for k in range(num_patches):
7                 data_matrix_long[z,:] = data[i][j][k]
8                 z=z+1
9         #Finding the main 50 clusters
10        data_km_long = kmeans.predict(data_matrix_long)
11        data_50clusters= np.zeros((num_images,num_grids,
12                                   num_patches))
13        z=0
14        for i in range(num_images):
15            for j in range(num_grids):
16                for k in range(num_patches):
17                    data_50clusters[i,j,k] = data_km_long[z]
18                    z=z+1
19        data_2500clusters= [['000_000' for i in range(num_patches)
20                             ] for j in range(num_grids)] for k in range(num_images)]
21        subcluster_data_fit = np.zeros((1,100))
22        z=1
23        #Finding the subclusters of each patch
24        for i in range(num_images):
25            for j in range(num_grids):
26                for k in range(num_patches):
27                    cluster = data_50clusters[i,j,k]
28                    subcluster_data_fit[0,:] = data[i][j][k]
29                    fitted_centre=cluster_dict[str(int(cluster))].
30                    predict(subcluster_data_fit)
31                    text=str(int(cluster))+ '_' +str(fitted_centre
32                           [0])
33                    data_2500clusters[i][j][k]= text
34                    if z%10000==0 and z>=10000:
35                        print(str(z*100/(num_images*num_grids*
36                                       num_patches))+ '% done')
37                    z=z+1
38        return(data_2500clusters)

```

- (e) We built a function for creating the histogram, i.e. we built the dataset of shape 10,000 x 2,500 for testing and 6,000 x 2,500 for training with each column giving the number of patches with a certain cluster\_subcluster.

```

1 def histogram_patch_creation(data,num_images=10000,num_grids
2                               =16,num_patches=9):
3     colnames=['image_num']
4     for i in range(50):
5         for j in range(50):
6             colnames.append(str(i)+'_'+str(j))
7     df=np.zeros((num_images,2501))
8     hist_patches = pd.DataFrame(df,columns = colnames)
9     hist_patches['image_num']=range(num_images)
10    z=1
11    for i in range(num_images):
12        for j in range(16):
13            for k in range(9):
14                hist_patches.loc[i,data[i][j][k]]+=1
15                if z%10000==0 and z>=10000:
16                    print(str(z*100/(num_images*num_grids*
17                                   num_patches))+ '% done')
18                z=z+1

```

```

17     return(hist_patches)

1  #Creating testing data of shape 10000x2500
2  test_hist_patches=histogram_patch_creation(test_2500clusters
      ,10000,16,9)
3  #Creating training data of shape 6000x2500
4  train_2500clusters = finding_cluster(training_patches, 6000,
      16, 9)
5  train_hist_patches=histogram_patch_creation(train_2500clusters
      ,6000,16,9)

```

### 1.3 Part c

We built a random forest classifier using the training data created as above and tested it on the testing data. Number of trees is set to be 100 and there is no depth set.

```

1  from sklearn.ensemble import RandomForestClassifier
2  from sklearn.metrics import accuracy_score
3  from sklearn.metrics import confusion_matrix
4  rf = RandomForestClassifier(n_estimators=100, oob_score=True).fit(
      train_hist_patches,training_labels)
5  test_y_pred = rf.predict(test_hist_patches)
6  train_y_pred = rf.predict(train_hist_patches)

```

```

1  #Accuracy on the training dataset
2  accuracy_score(training_labels,train_y_pred)*100

```

```

: accuracy_score(training_labels,train_y_pred)*100

: 100.0

```

```

1  #Accuracy on the testing dataset
2  accuracy_score(testing_labels,test_y_pred)*100

```

```

accuracy_score(testing_labels,test_y_pred)*100

89.79

```

### 1.4 Part d

### 1.5 Part e