BS 6207 ASSIGNMENT 4 - *Tian Siqi, G2101015G*

Github link: https://github.com/SiqiT/Assigmnment

1. Generate results for MNIST dataset for ucc and clustering as in Table 1 in the paper. Plot all your results for analysis and explain them.

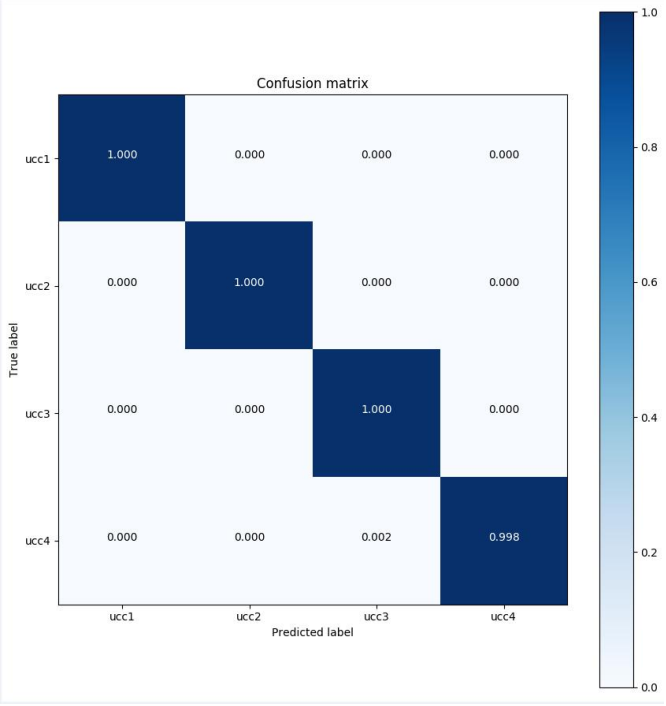
(1) UCC model

This model **has an** autoencoder branch in their aechitecture and they were optimized jointly over both **autoencoder** loss and **ucc** loss. It is trained on bags with labels of **ucc1 to ucc4**.

The model I used here is the h5 model from the paper, so no need to train a new model, this model has epochs=1000001.

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.9995	0.958
all_kmeans++		0.945



(2) UCC^{2+} model

This model **has an** autoencoder branch in their aechitecture and they were optimized jointly over both **autoencoder** loss and **ucc** loss. It is trained on bags with labels of **ucc2 to ucc4**.

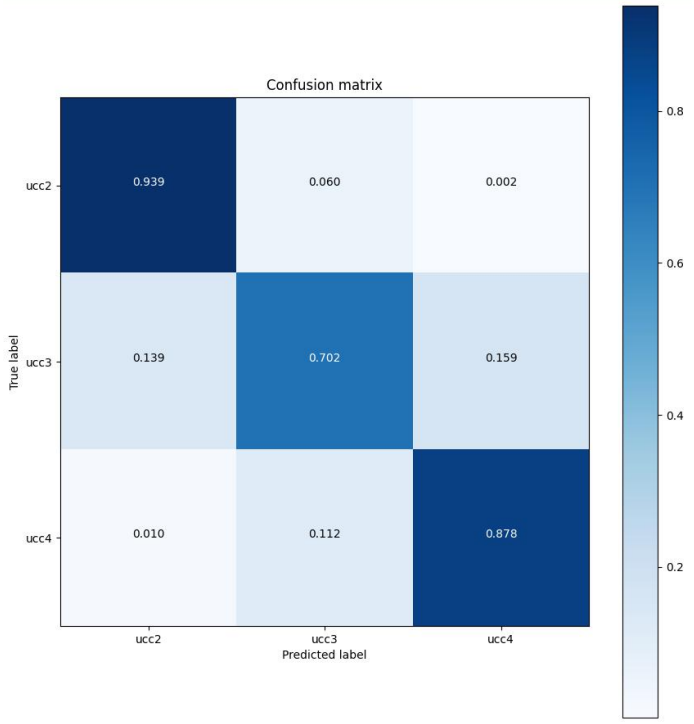
I train this new model with epochs=2001.

```
parser.add_argument('--ucc_start', default='1', type=int, help='ucc start', dest='ucc_start')
parser.add_argument('--ucc_end', default='4', type=int, help='ucc end', dest='ucc_end')
```

I change those 2 parameters in train.py and test.py to make it rained on bages with labels of ucc2 to ucc4.

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.8397	0.984
all_kmeans++		0.977




```
clustering_type:kmeans
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.9773
Clustering accuracy calculation completed!!!
obtain_clustering_labels.py
description:2019_09_05__18_43_15__128000
clustering_type:nn
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.9839
Clustering accuracy calculation completed!!!
```

(3) UCC α =1

This model **doesn't have an** autoencoder branch in their aechitecture and they were optimized jointly over only **ucc** loss. It is trained on bags with labels of **ucc2 to ucc4**.

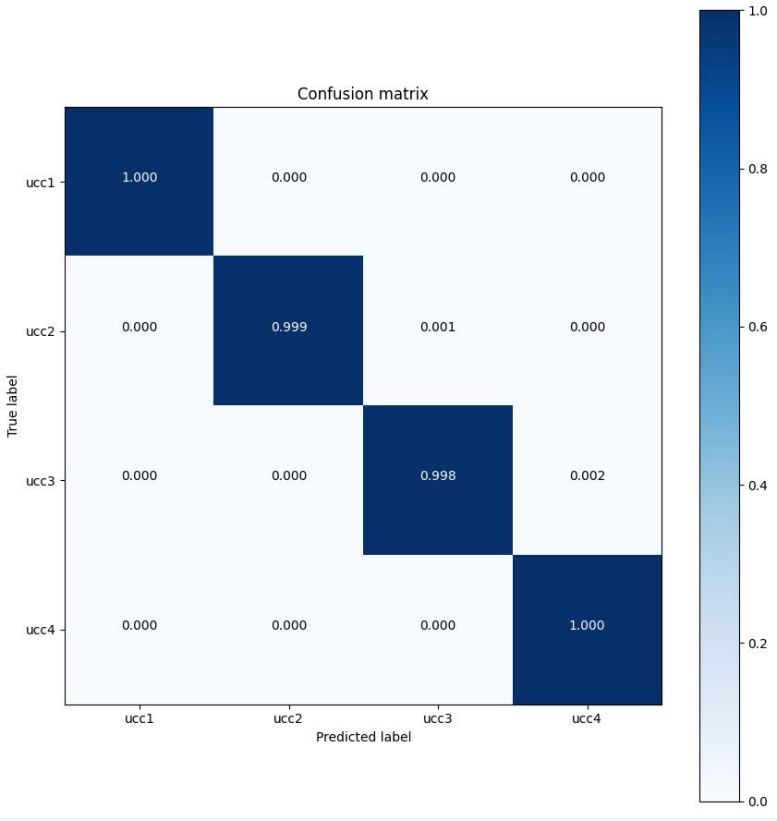
I train this new model with epochs=2001.

```
self._classification_model.compile(optimizer=optimizer, loss=['categorical_crossentropy','mse'], metrics=
['accuracy'], loss_weights=[1, 0])
```

I change the weight for ucc loss and autoencoder loss in model.py to make it only uses the ucc loss. Since we know the ucc loss is 'categorical_crossentropy', ae_loss is 'mse'.

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.9993	0.984
all_kmeans++		0.977




```
clustering_type:kmeans
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.9773
Clustering accuracy calculation completed!!!
obtain_clustering_labels.py
description:2019_09_05__18_43_15__128000
clustering_type:nn
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.9839
Clustering accuracy calculation completed!!!
```

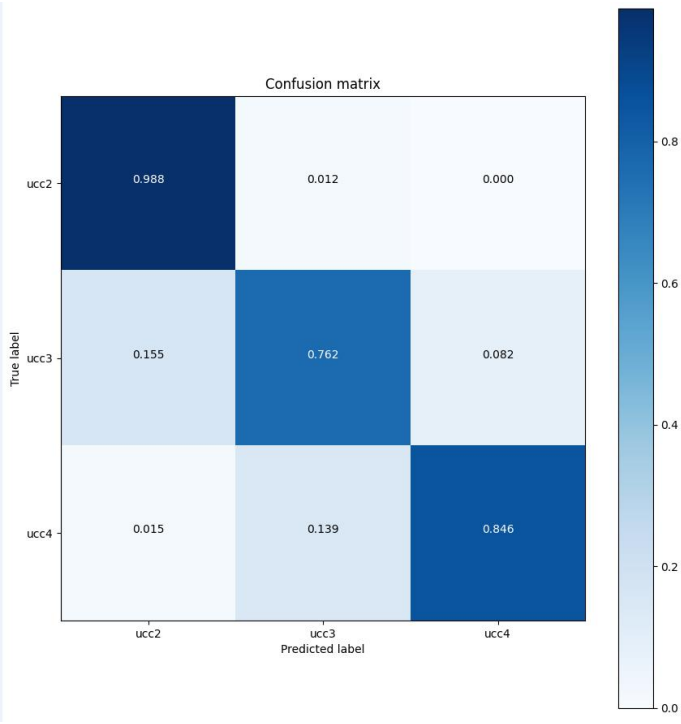
(4)UCC(2+, α =1)

This model **doesn't have** autoencoder branch in their aechitecture and they were optimized jointly over only **ucc** loss. It is trained on bags with labels of **ucc2 to ucc4**.

I train this new model with epochs=2001.

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.8656	0.984
all_kmeans++		0.977




```
clustering_type:kmeans
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.9773
Clustering accuracy calculation completed!!!
obtain_clustering_labels.py
description:2019_09_05__18_43_15__128000
clustering_type:nn
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.9839
Clustering accuracy calculation completed!!!
```

Analysis:

(1)From all 4 results, we can see the ucc accuracy is better than the final clustering accuracy. That’ s reasonable since the result from ucc is based on a self-supervised model, the result from clustering is based on an unsupervised model and the input has already been processed.

(2)Neural network can have a better performance than k-mean in this case.

(3)Using 4 labels for the bags can get a more accurate result than just using 3. Since using more labels can make the model catch more details about the data.

2. Make different feature extractors and distribution regression

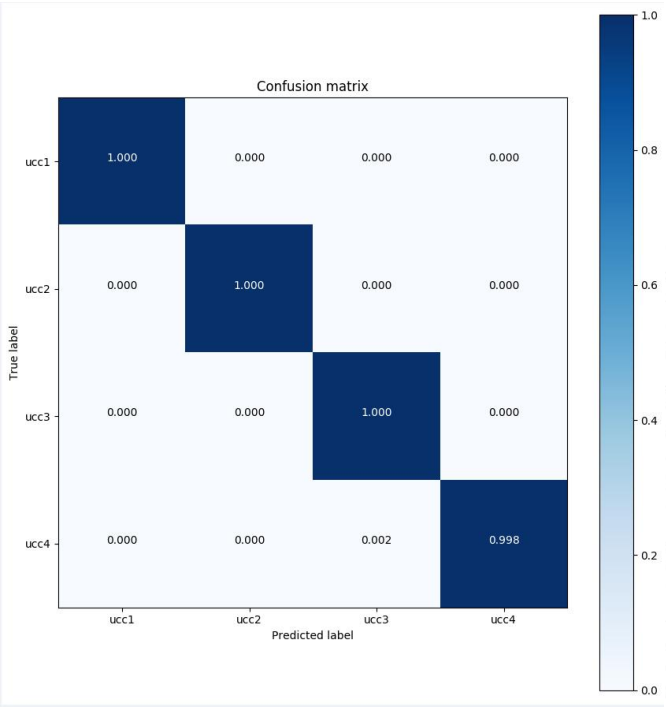
(1) Make different feature extractors

```
class Keras_Model(object):
    def __init__(self, patch_size=28, num_instances=2, num_classes=2, learning_rate=1e-4, num_bins=None, num_features=10,
batch_size=None):
        self._lr_rate=learning_rate
        self._num_features = num_features

        kernel_kwargs = {
            'kernel_initializer': glorot_uniform(),
            'kernel_regularizer': None
        }

        patch_input = Input((28, 28, 1))
        x0 = Conv2D(16, (3, 3), padding='same', bias_initializer=Constant(value=0.1), **kernel_kwargs)(patch_input)
        x0 = self.wide_residual_blocks(x0, 2, 1, 16)
        x0 = self.wide_residual_blocks(x0, 4, 1, 16, True)
        x0 = self.wide_residual_blocks(x0, 8, 1, 16, True)
        x0 = Activation('sigmoid')(x0)
```

The default activation function for Keras_model is ‘relu’, I change it to ‘sigmoid’ to make the feature extractor different.

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.9995	0.984
all_kmeans++		0.977




```
clustering_type:kmeans
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.9773
Clustering accuracy calculation completed!!!
obtain_clustering_labels.py
description:2019_09_05__18_43_15__128000
clustering_type:nn
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.9838
Clustering accuracy calculation completed!!!
```

From this result, we can see change the activation function of the extractor doesn't affect the accuracy of classification.

(2) Make different distribution regression

In this part, I change the **kernel** function in KDE from **Gaussian** kernel to Exponential kernel. This change is done in obtain_distributions.py.

$$h^j_{\sigma_\zeta}(v) = \frac{1}{|\sigma_\zeta|}\sum_{i=1}^{|\sigma_\zeta|}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2\sigma^2}\left(v-f^{j,i}_{\sigma_\zeta}\right)^2}$$

Gaussian:

$$h^j_{\sigma_\zeta}(v) = \frac{1}{|\sigma_\zeta|}\sum_{i=1}^{|\sigma_\zeta|}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2\sigma^2}\left(v-f^{j,i}_{\sigma_\zeta}\right)}$$

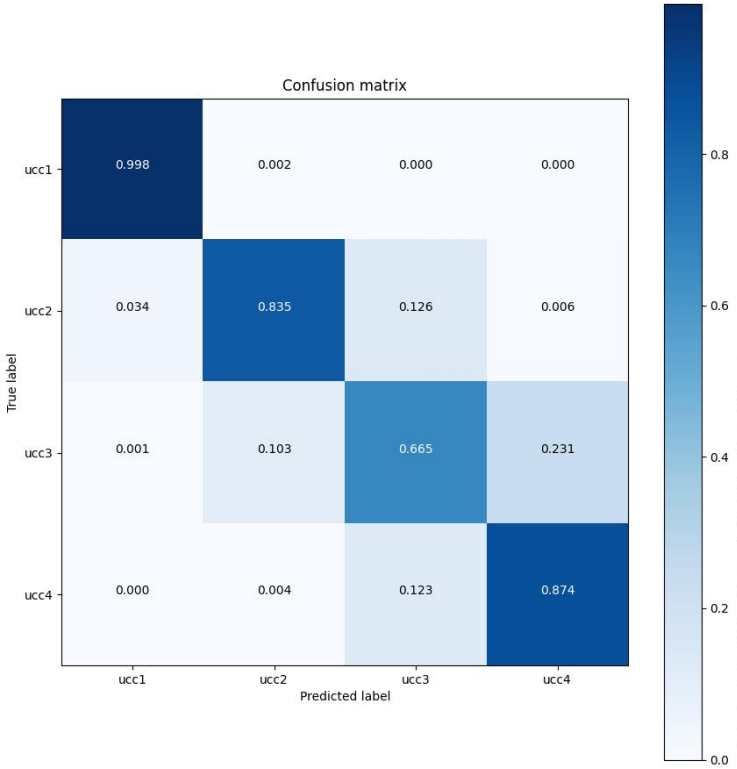
Exponential:

The difference between them is Gaussian is using L2 distance, Exponential kernel is using L1 distance, this will make the result less dependent on the parameters.

```
def kde(data, num_nodes=None, sigma=None, batch_size=None, num_features=None):
    # print('kde data shape:{}'.format(data.shape))
    # print('num_nodes:{}'.format(num_nodes))
    # print('sigma:{}'.format(sigma))

    k_sample_points = np.tile(np.linspace(0,1,num=num_nodes),[batch_size,data.shape[1],1])
    # print('kde k_sample_points shape:{}'.format(k_sample_points.shape))

    k_alfa = 1/np.sqrt(2*np.pi*np.square(sigma))
    k_beta = -1/(2*np.square(sigma))

    out_list = list()
    for i in range(num_features):
        temp_data = np.reshape(data[:,:,i],(-1,data.shape[1],1))
        # print('temp_data shape:{}'.format(temp_data.shape))

        k_diff = k_sample_points - np.tile(temp_data,[1,1,num_nodes])
        #k_diff_2 = np.square(k_diff)
        # print('k_diff_2 shape:{}'.format(k_diff_2.shape))

        k_result = k_alfa * np.exp(k_beta*k_diff)
```

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.8429	0.797
all_kmeans++		0.592




```
clustering_type:kmeans
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.5922
Clustering accuracy calculation completed!!!
obtain_clustering_labels.py
description:2019_09_05__18_43_15__128000
clustering_type:nn
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.7967
Clustering accuracy calculation completed!!!
```

This result shows that the Exponential kernel perform worse than the Gaussian kernel.

3. **Use full MNIST data set 60,000 for training, also use subset of MNIST for training and analyse the overfitting effects.**

In this question, I use different training set to train the **UCC models** to analyse the overfitting effects.

```
    splitted_val_indices_list += list(temp_mnist_train_indices[:splitted_num_val])

    splitted_train_indices_list += list(temp_mnist_train_indices[:])#splitted_num_val:])
```

In this section I change the training data list range in prepare_splitting_indices_files.py to choose different training set range.

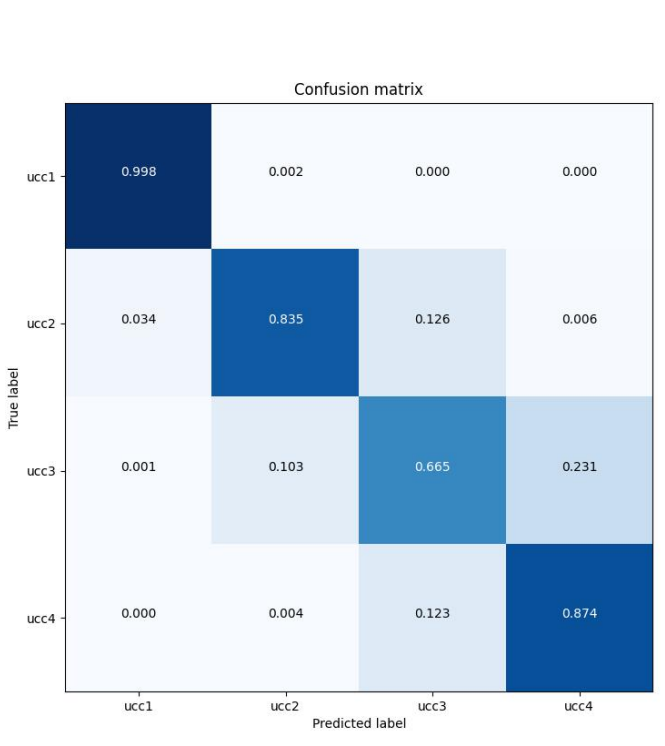
Each new model was trained with epochs=2001.

(1)60000 for training

```
##### Splitted Dataset #####
digit:0,        num_train:5923, num_val:987,    num_test:980,   total:7890
digit:1,        num_train:6742, num_val:1123,   num_test:1135,  total:9000
digit:2,        num_train:5958, num_val:993,    num_test:1032,  total:7983
digit:3,        num_train:6131, num_val:1021,   num_test:1010,  total:8162
digit:4,        num_train:5842, num_val:973,    num_test:982,   total:7797
digit:5,        num_train:5421, num_val:907,    num_test:892,   total:7220
digit:6,        num_train:5918, num_val:986,    num_test:958,   total:7862
digit:7,        num_train:6265, num_val:1044,   num_test:1028,  total:8337
digit:8,        num_train:5851, num_val:975,    num_test:974,   total:7800
digit:9,        num_train:5949, num_val:991,    num_test:1009,  total:7949
TOTAL:, num_train:60000.0,      num_val:10000.0,        num_test:10000.0,       total:80000.0
```

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.8429	0.798
all_kmeans++		0.597




```
clustering_type:kmeans
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.5965
Clustering accuracy calculation completed!!!
obtain_clustering_labels.py
description:2019_09_05__18_43_15__128000
clustering_type:nn
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.7981
Clustering accuracy calculation completed!!!
```

II. 20,000 for training (same number of examples per class)

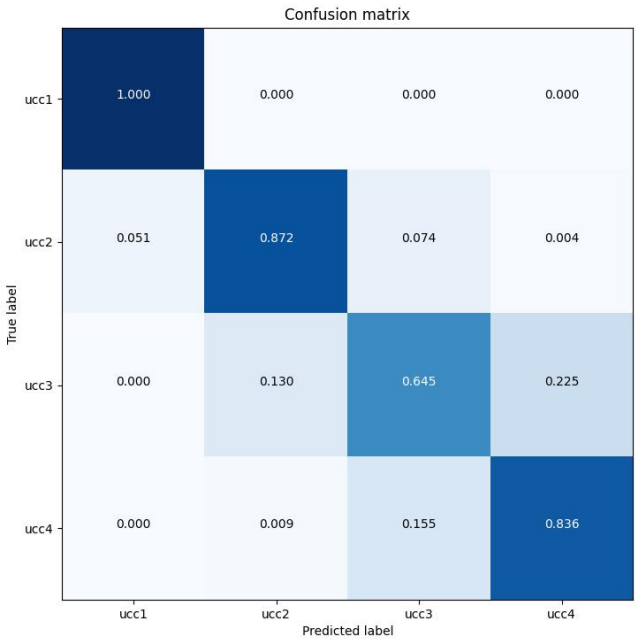
```
for i in range(10):
    digit_key = 'digit' + str(i)

    temp_digit_dict = dict()

    temp_digit_dict['train_indices'] = np.where(y_train == i)[0]
    temp_digit_dict['num_train'] = 2000#len(temp_digit_dict['train_indices'])

    splitted_train_indices_list += list(temp_mnist_train_indices[:2000])#splitted_num_val:])
```

In order to achieve the same number of examples per class, I change the temp_digit_dict['nunm_train'] and temp_digit_dict['num_train'] in prepare_splitting_indices_files.py to achieve this goal.

```
##### Splitted Dataset #####
digit:0,        num_train:2000, num_val:333,    num_test:980,   total:3313
digit:1,        num_train:2000, num_val:333,    num_test:1135,  total:3468
digit:2,        num_train:2000, num_val:333,    num_test:1032,  total:3365
digit:3,        num_train:2000, num_val:333,    num_test:1010,  total:3343
digit:4,        num_train:2000, num_val:333,    num_test:982,   total:3315
digit:5,        num_train:2000, num_val:5421,   num_test:892,   total:8313
digit:6,        num_train:2000, num_val:333,    num_test:958,   total:3291
digit:7,        num_train:2000, num_val:333,    num_test:1028,  total:3361
digit:8,        num_train:2000, num_val:333,    num_test:974,   total:3307
digit:9,        num_train:2000, num_val:333,    num_test:1009,  total:3342
TOTAL:, num_train:20000.0,      num_val:8418.0, num_test:10000.0,       total:38418.0
```

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.8381	0.700
all_kmeans++		0.778




```
clustering_type:kmeans
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.7781
Clustering accuracy calculation completed!!!
obtain_clustering_labels.py
description:2019_09_05__18_43_15__128000
clustering_type:nn
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.7006
Clustering accuracy calculation completed!!!
```

III. 5,000 for training (same number of examples per class)

```
##### Splitted Dataset #####
digit:0,        num_train:500,  num_val:83,     num_test:980,   total:1563
digit:1,        num_train:500,  num_val:83,     num_test:1135,  total:1718
digit:2,        num_train:500,  num_val:83,     num_test:1032,  total:1615
digit:3,        num_train:500,  num_val:83,     num_test:1010,  total:1593
digit:4,        num_train:500,  num_val:83,     num_test:982,   total:1565
digit:5,        num_train:500,  num_val:5421,   num_test:892,   total:6813
digit:6,        num_train:500,  num_val:83,     num_test:958,   total:1541
digit:7,        num_train:500,  num_val:83,     num_test:1028,  total:1611
digit:8,        num_train:500,  num_val:83,     num_test:974,   total:1557
digit:9,        num_train:500,  num_val:83,     num_test:1009,  total:1592
TOTAL:, num_train:5000.0,       num_val:6168.0, num_test:10000.0,       total:21168.0
```

Results:

	ucc_acc	Clustering_acc
all_spectral_nn	0.8569	0.787
all_kmeans++		0.497


```
clustering_type:kmeans
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.4967
Clustering accuracy calculation completed!!!
obtain_clustering_labels.py
description:2019_09_05__18_43_15__128000
clustering_type:nn
__2019_09_05__18_43_15__0123456789__128000
Clustering completed!!!
calculate_clustering_accuracy.py
description:2019_09_05__18_43_15__128000
__2019_09_05__18_43_15__0123456789__128000
Clustering acc:0.7868
Clustering accuracy calculation completed!!!
```

IIII. 5,00 for training (same number of examples per class)

```
~/.conda/envs/my_bash/lib/python3.8/site-packages/numpy/core/fromnumeric.py in _wrapfunc(obj, method, *args, **kwds)
     55
     56     try:
---> 57         return bound(*args, **kwds)
     58     except TypeError:
     59         # A TypeError occurs if the object does have such a method in its

ValueError: cannot reshape array of size 343392 into shape (32,28,28,1)
```

Can't get a training result.

Analysis:

From all these results above, we can see overfitting can get a good ucc_accuracy, but for the clustering accuracy, it is bad, especially in k_mean model. Underfitting would also get a bad clustering result.

Only if we choose the appropriate training data size, we can get the best classification result.
NN model can perform better in all the cases than the k_mean model.

4.Devise ways to improve this algorithm

In the algorithm, now the learning rate for training the CNN model is 0.0001, this is unchangeable. However, maybe we can choose to use an adaptive learning rate.
The performance of the model on the training dataset can be monitored by the learning algorithm and the learning rate can be adjusted in response. This allows large weight changes at the beginning of the learning process and small changes or fine-tuning towards the end of the learning process.
We would first give a bigger learning rate to the model, once the performance of the model plateaus, we will update the learning rate to a smaller value, such as by decreasing the learning rate by a factor of two or an order of magnitude.
In this way, we can not only get an accurate model but also get a more efficient training time.