Handwritten Recognition via K-Means Clustering

June 8, 2019

Jian Sun
DUID: 873397832

1 Content

- Introduction
- Model
- Result
  - Euclidean Distance
    - 7 Nodes
    - 10 Nodes
    - 12 Nodes
  - Manhattan Distance
    - 7 Nodes
    - 10 Nodes
    - 12 Nodes
- Discussion
- Appendix

2 Introduction

In this project, we implement K-Means Clustering to recognize MNIST dataset.

K Means Clustering is a very common and popular clustering methods. Here, we plan to exam its power from different angles. The Euclidean Distance and Manhattan Distance are separately selected to calculate the distance between clustered mean with chozen points. Then we will try to cluster those training data to 7, 10 and 12 nodes to test which one is best. The evaluation standard are entropy and purity. We guess that more nodes will contribute to better clustering results.

3 Model

3.1 Data Processing

The whole MNIST is splitted as the following 2 parts.

The training set has 60000 28X28 images;
the testing set has 10000 28X28 images.

And only training set is selected to process here.

3.2 Structure

K Means Clustering will initially set some points as nodes. Their values are initially means. Then the other point will be allocated to nearest node. The nearest here is evaluated by the distance between this point and different nodes. Usually there are several methods to calculate distance. Here we choose Euclidean Distance and Manhattan Distance.

For each Distance calculation method, we try to compare the results between different nodes. This project we set the nodes number as 7, 10 and 12.

Next, we will get entropy and purity for each node and total entropy and total purity. Finally, we return the variance and plot each k means out.

3.3 Code

The code structure get great improved on this project, which is very different from the former handwritten recognition projects. We introduce some UI idea here and let user can play with this program. For example, firstly, you can set nodes number as you want, then you can set different epoach times, then choose different distance (currently only 2 distances, but can extend to more in the future) and set threshold value(given that we don't have to do it here, so we commented this part.)

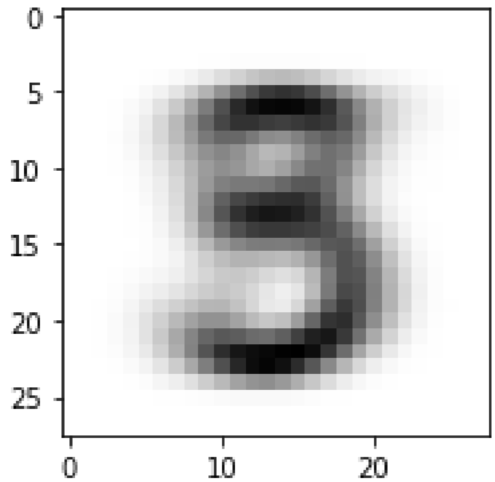
4 Result

4.1 Euclidean Distance + 7 Nodes

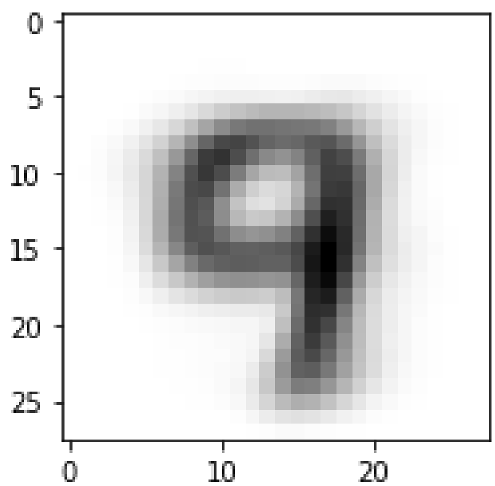
Clustering Nodes Number is: 7
Iteration steps: 160
Total entropy is 1.322
Total purity is 0.5277
The 7 means are plotted in the following.

The rest results are presented in discussion part.

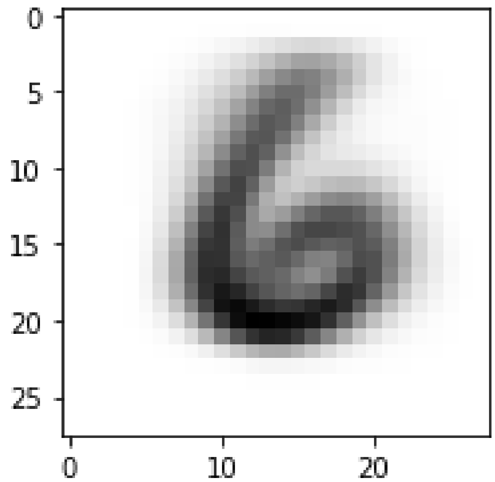
```
In [26]: for i in range(int(K_num)):
             tupian = np.reshape(np.round(KM[i]*255.),(28,28))
             plt.figure(figsize=(10,10))
             plt.subplot(3,3,i+1)
             plt.grid(False)
             plt.imshow(tupian, cmap=plt.cm.binary)
             plt.xlabel("The %dth node " %(i))
             plt.show()
```



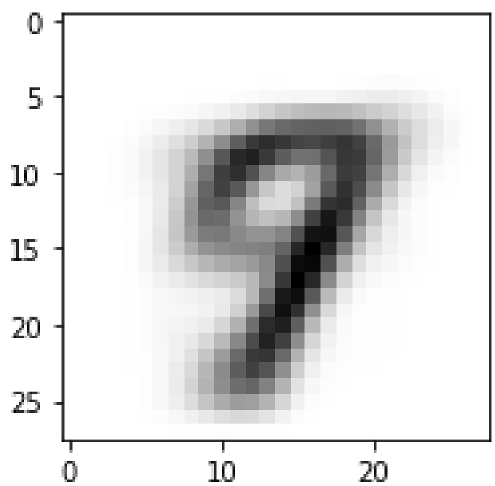
The 0th node



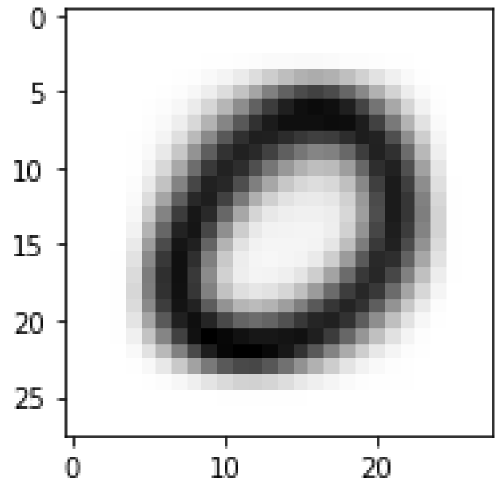
The 1th node



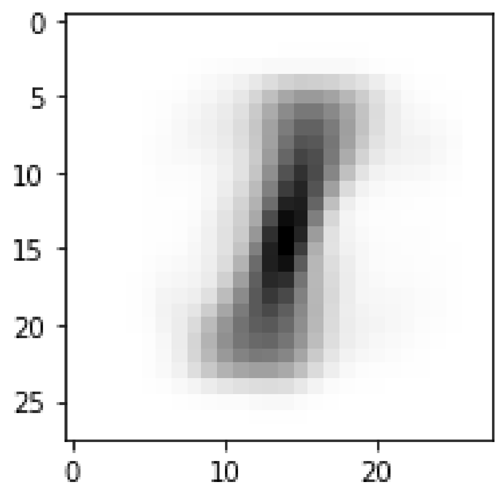
The 2th node



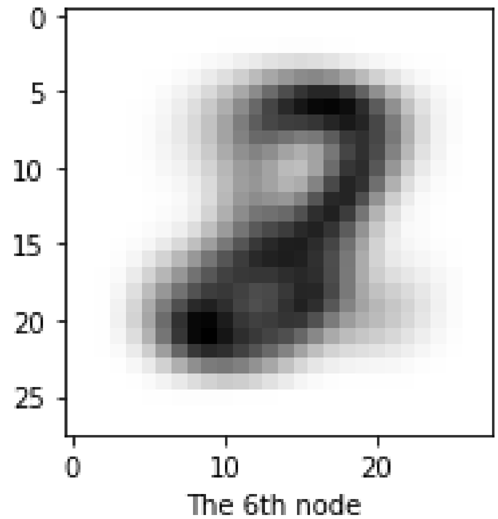
The 3th node



The 4th node



The 5th node

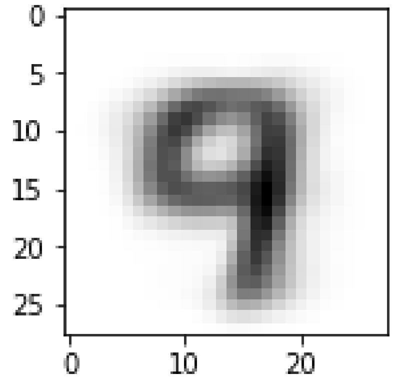


4.2 Euclidean Distance + 10 Nodes

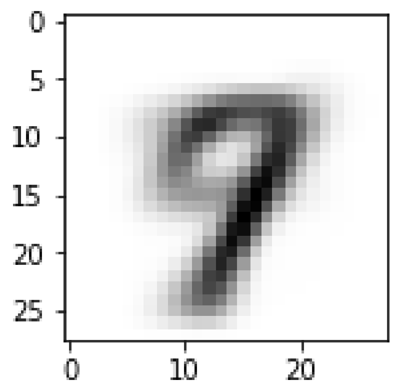
Clustering Nodes Number is: 10
Iteration steps: 160
Total entropy is 1.1333
Total purity is 0.5972
The variance is The 10 means are plotted in the following.

The rest results are presented in discussion part.

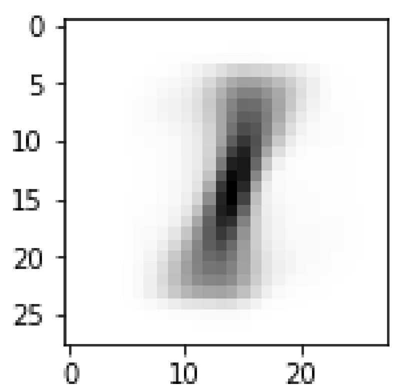
```
In [28]: for i in range(int(K_num)):
             tupian = np.reshape(np.round(KM[i]*255.),(28,28))
             plt.figure(figsize=(10,10))
             plt.subplot(3,4,i+1)
             plt.grid(False)
             plt.imshow(tupian, cmap=plt.cm.binary)
             plt.xlabel("The %dth node " %(i))
             plt.show()
```



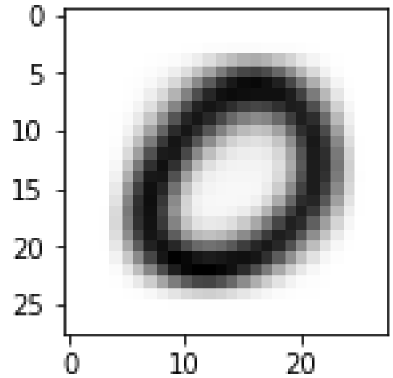
The 0th node



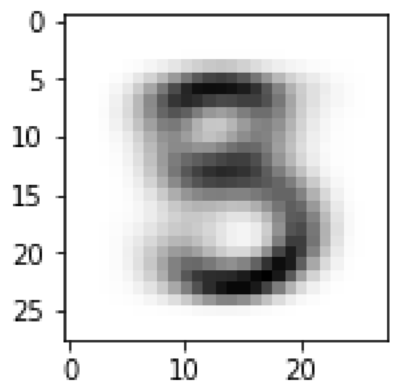
The 1th node



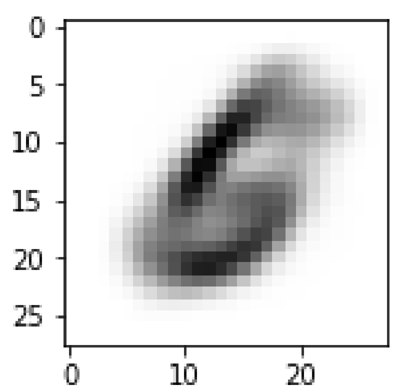
The 2th node



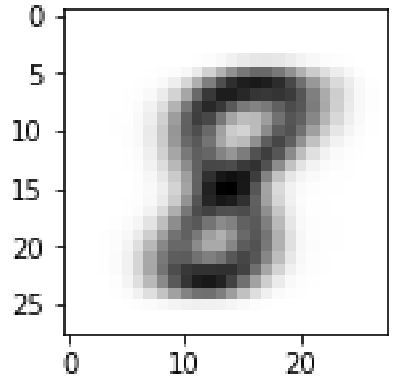
The 3th node



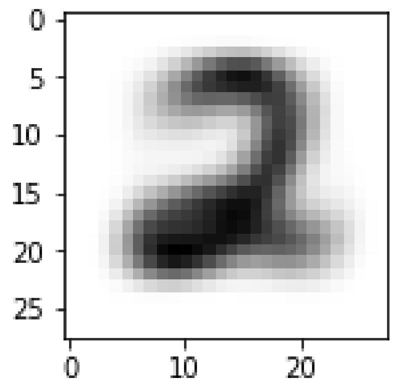
The 4th node



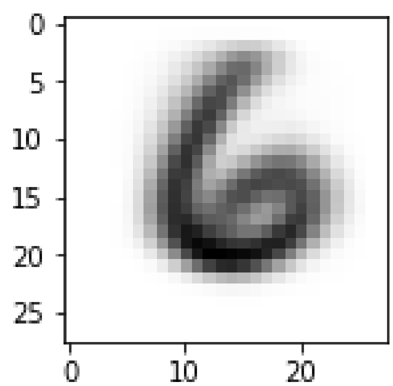
The 5th node



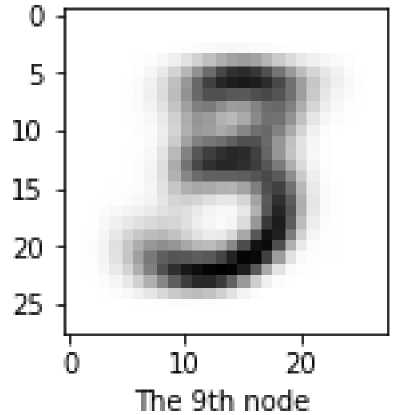
The 6th node



The 7th node



The 8th node



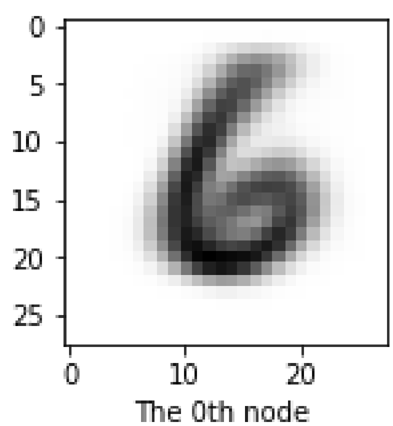
4.3 Euclidean Distance + 12 Nodes

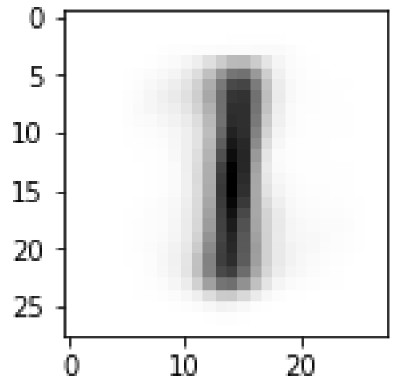
Clustering Nodes Number is: 12
Iteration steps: 160
Total entropy is 1.1117
Total purity is 0.6233

The variance is The 12 means are plotted in the following.

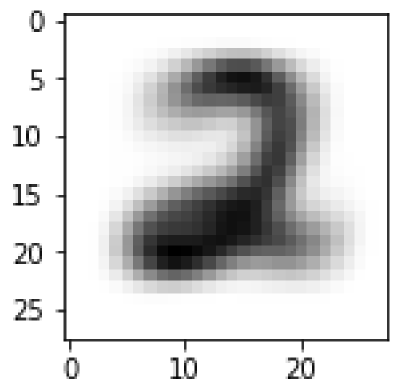
The rest results are presented in discussion part.

```
In [30]: for i in range(int(K_num)):
             tupian = np.reshape(np.round(KM[i]*255.),(28,28))
             plt.figure(figsize=(10,10))
             plt.subplot(3,4,i+1)
             plt.grid(False)
             plt.imshow(tupian, cmap=plt.cm.binary)
             plt.xlabel("The %dth node " %(i))
             plt.show()
```

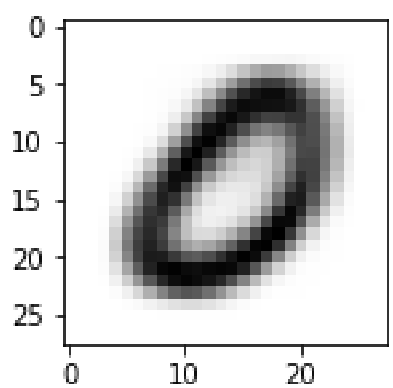




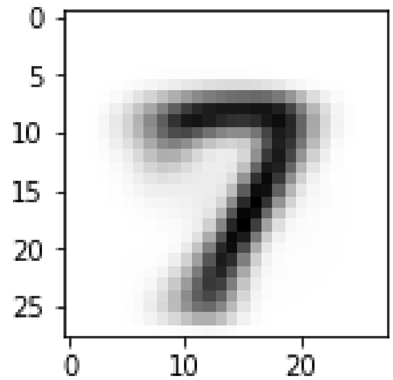
The 1th node



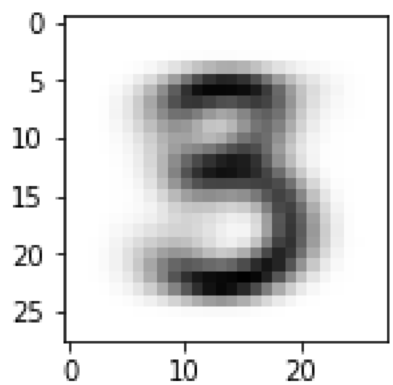
The 2th node



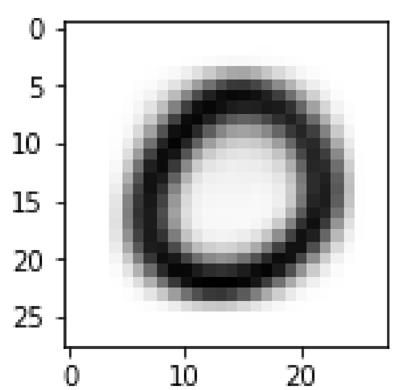
The 3th node



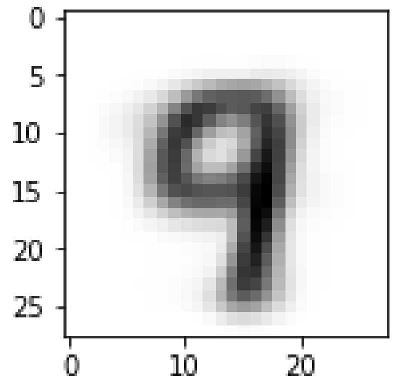
The 4th node



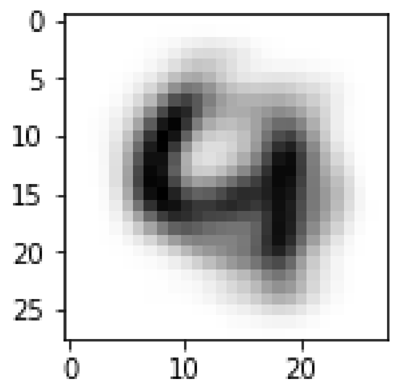
The 5th node



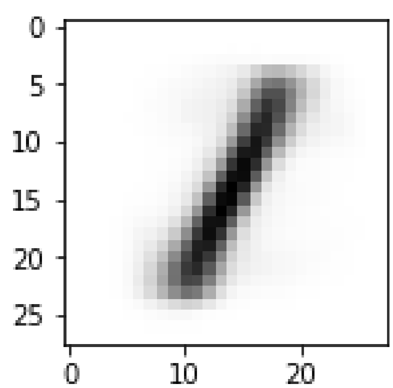
The 6th node



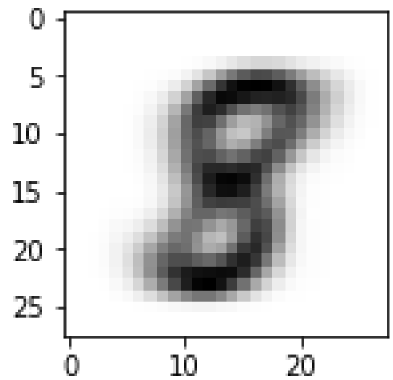
The 7th node



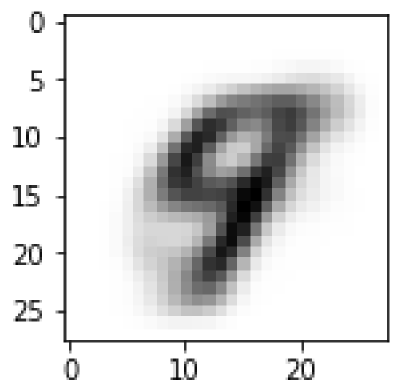
The 8th node



The 9th node



The 10th node



The 11th node

4.4 Manhattan Distance + 7 Nodes

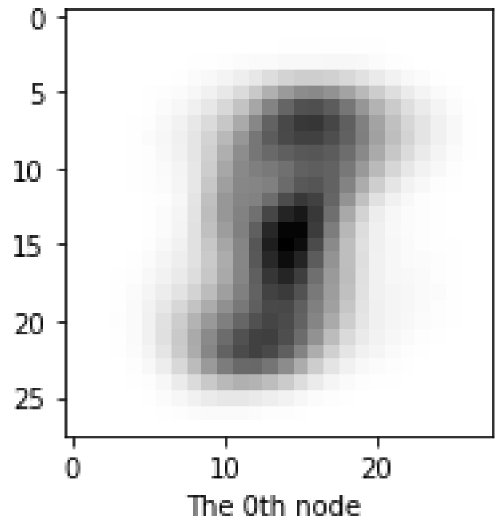
Clustering Nodes Number is: 7
Iteration steps: 160
Total entropy is 1.4971 Total purity is 0.4457

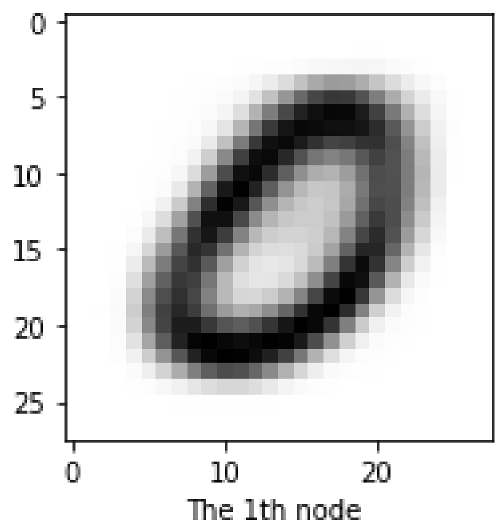
The variance is The 12 means are plotted in the following.

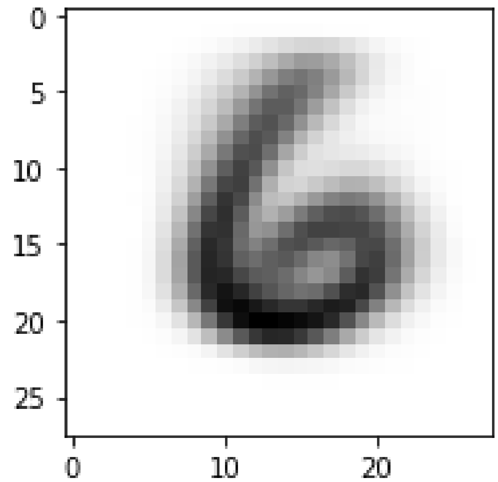
The rest results are presented in discussion part.

```
In [32]: for i in range(int(K_num)):
             tupian = np.reshape(np.round(KM[i]*255.),(28,28))
             plt.figure(figsize=(10,10))
             plt.subplot(3,3,i+1)
             plt.grid(False)
```

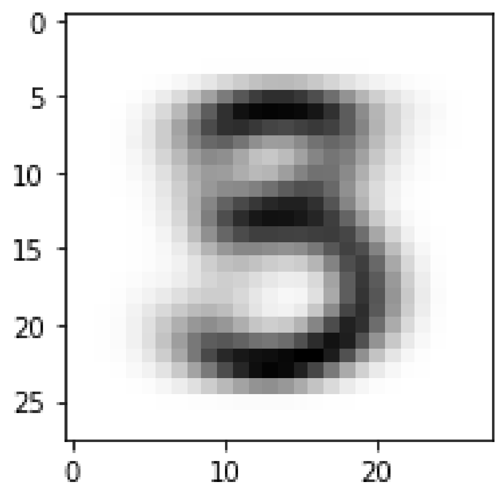
```
plt.imshow(tupian, cmap=plt.cm.binary)
plt.xlabel("The %dth node " %(i))
plt.show()
```



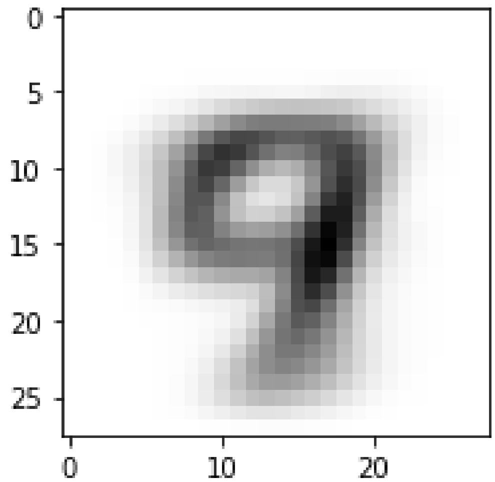




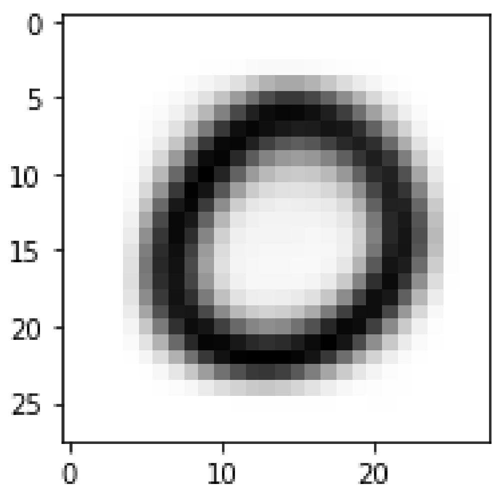
The 2th node



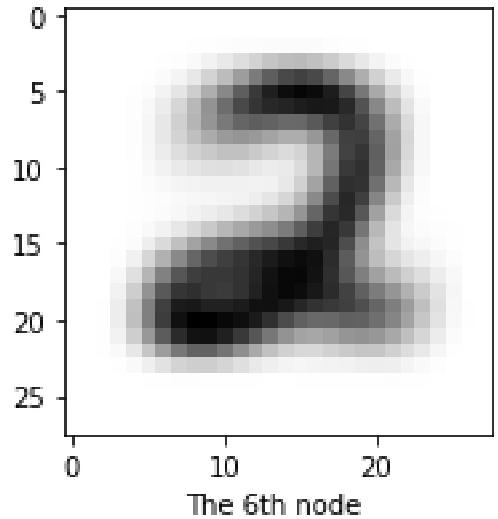
The 3th node



The 4th node



The 5th node



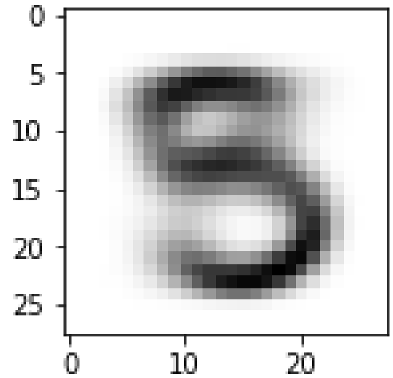
4.5 Manhattan Distance + 10 Nodes

Clustering Nodes Number is: 10
Iteration steps: 160
Total entropy is 1.395
Total purity is 0.4971

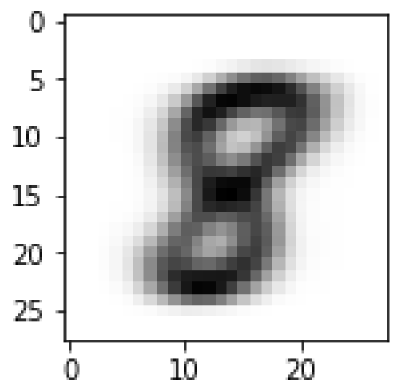
The variance is The 10 means are plotted in the following.

The rest results are presented in discussion part.

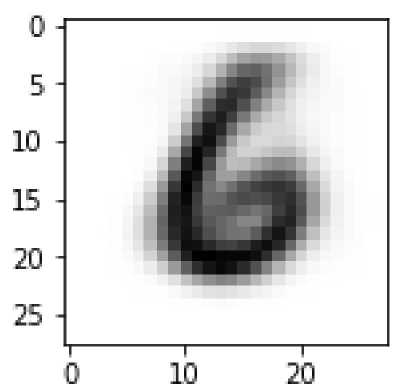
```
In [34]: for i in range(int(K_num)):
             tupian = np.reshape(np.round(KM[i]*255.),(28,28))
             plt.figure(figsize=(10,10))
             plt.subplot(3,4,i+1)
             plt.grid(False)
             plt.imshow(tupian, cmap=plt.cm.binary)
             plt.xlabel("The %dth node " %(i))
             plt.show()
```



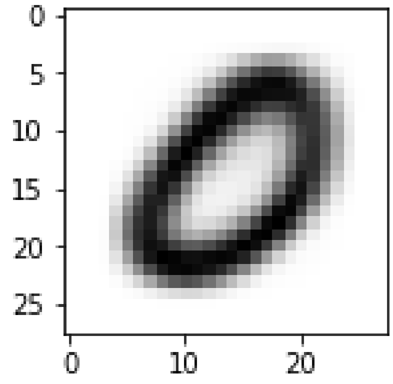
The 0th node



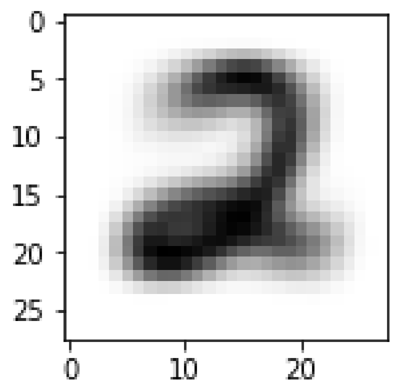
The 1th node



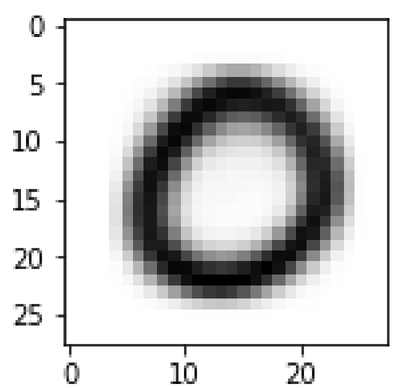
The 2th node



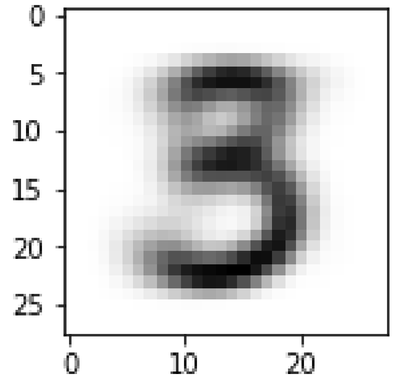
The 3th node



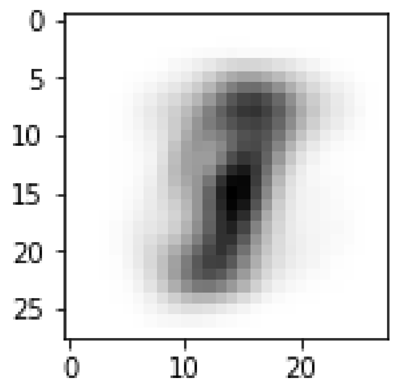
The 4th node



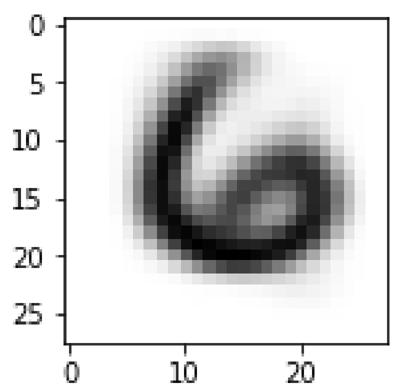
The 5th node



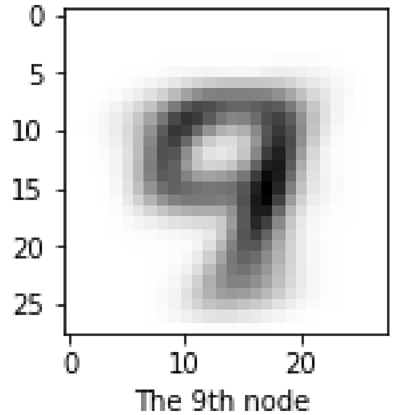
The 6th node



The 7th node



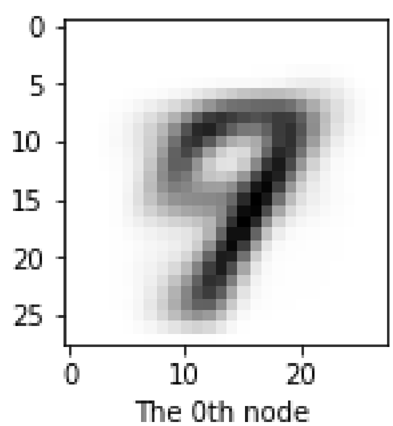
The 8th node

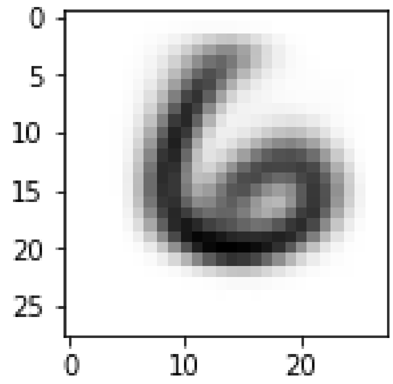


4.6 Manhattan Distance + 12 Nodes

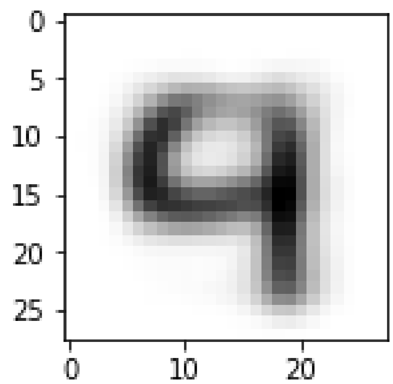
Clustering Nodes Number is: 12
Iteration steps: 160
Total entropy is 1.309
Total purity is 0.5301
The variance is The 12 means are plotted in the following.
The rest results are presented in discussion part.

```
In [36]: for i in range(int(K_num)):
             tupian = np.reshape(np.round(KM[i]*255.),(28,28))
             plt.figure(figsize=(10,10))
             plt.subplot(3,4,i+1)
             plt.grid(False)
             plt.imshow(tupian, cmap=plt.cm.binary)
             plt.xlabel("The %dth node " %(i))
             plt.show()
```

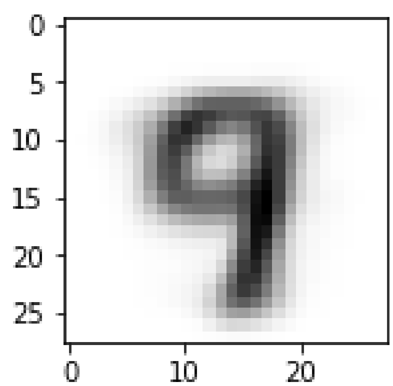




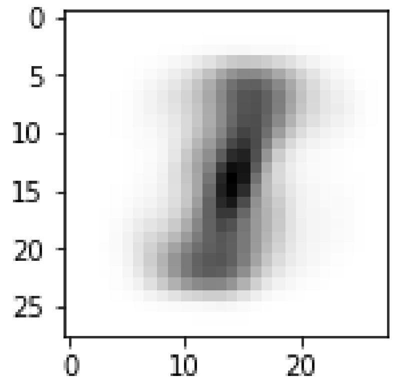
The 1th node



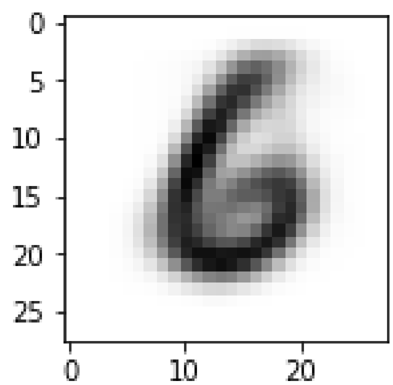
The 2th node



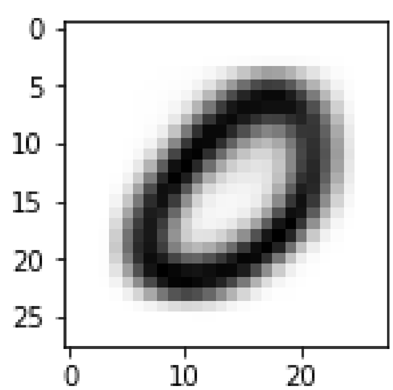
The 3th node



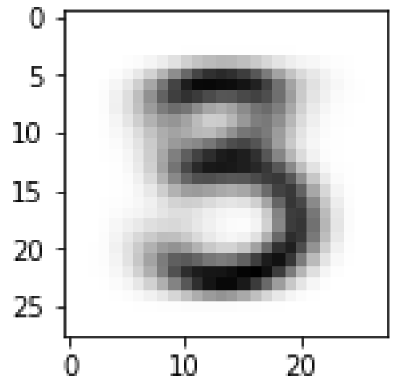
The 4th node



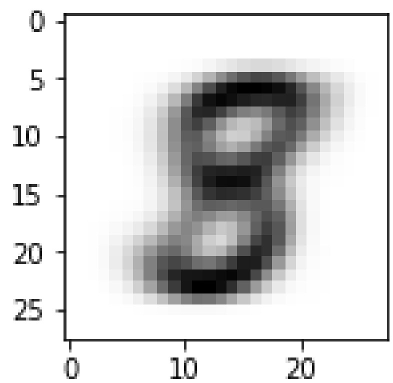
The 5th node



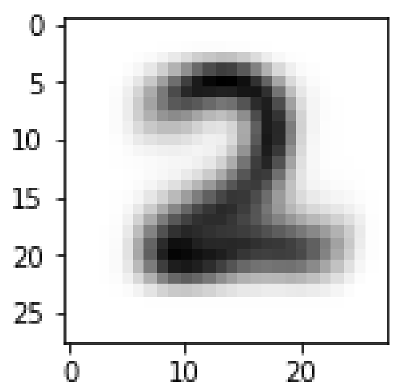
The 6th node



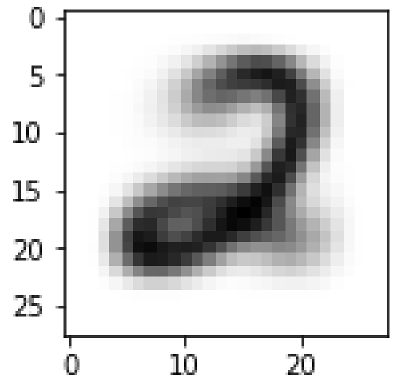
The 7th node



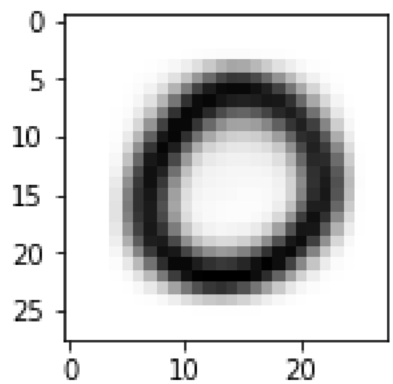
The 8th node



The 9th node



The 10th node



The 11th node

5 Discussion

The all specific results are shown in the following image.

As we can see, the higher entropy will lead to lower purity, the more nodes will contribute to better purity and the clustered results based on Euclidean Distance performs better than that on Manhattan Distance.

Euclidean Distance + 12 Nodes get the best results with Total entropy, 1.1117 and Total purity, 0.6233;

Manhattan Distance + 7 Nodes gets the worst results with Total entropy, 1.4971 and Total purity, 0.4457.

We also find that if set a low threshold, then iteration will stop when the threshold is bigger than the difference between new means and old means. On this condition, we will get better results, but we didn't do this due to the processing time. Meanwhile, I code this part and mute it. We can activate it when we need.

```
In [4]: import matplotlib.pyplot as plt
        def BGR2RGB(Input):
            output = np.zeros(Input.shape);
            output[:,:,0] = Input[:,:,2]
            output[:,:,1] = Input[:,:,1]
            output[:,:,2] = Input[:,:,0]
            output = output.astype('uint8')
            return output
        pic = cv2.imread('./tr.png')

        # plot the image out
        plt.figure(figsize=(50,32))
        plt.subplot(311),plt.imshow(BGR2RGB(pic))
        plt.title('Specific Results'), plt.xticks([]), plt.yticks([])
```

```
Out[4]: (Text(0.5, 1.0, 'Specific Results'),
         ([], <a list of 0 Text xticklabel objects>),
         ([], <a list of 0 Text yticklabel objects>))
```

Specific Results

	Euclidean distance						Manhattan distance					
	7 Nodes		10 Nodes		12 Nodes		7 Nodes		10 Nodes		12 Nodes	
	Entropy	Purity	Entropy	Purity	Entropy	Purity	Entropy	Purity	Entropy	Purity	Entropy	Purity
Node 0	1.3702	0.4486	1.4685	0.3578	0.6457	0.8616	2.1026	0.2601	1.28	0.3822	1.4702	0.3968
Node 1	1.5172	0.3502	1.2552	0.4522	1.3489	0.6506	1.021	0.7357	1.1754	0.6444	0.4732	0.9027
Node 2	0.8019	0.8107	1.2081	0.6999	0.4129	0.9141	0.7424	0.832	1.0576	0.7221	1.1854	0.5347
Node 3	1.5334	0.3692	0.3199	0.9451	0.7944	0.8136	1.1971	0.5288	0.6298	0.8661	1.5614	0.3212
Node 4	0.4193	0.9233	1.4395	0.415	0.5452	0.8596	1.4154	0.3084	0.1809	0.9683	1.8775	0.3873
Node 5	1.5078	0.5871	1.7153	0.3265	1.2428	0.5345	0.383	0.9294	0.2694	0.9549	1.0809	0.7
Node 6	1.5001	0.4714	1.0253	0.7218	0.3505	0.9346	0.2972	0.9441	1.1531	0.5926	0.489	0.9012
Node 7			0.4277	0.92	1.444	0.3947			2.0475	0.2997	1.0594	0.5749
Node 8			0.5888	0.8764	1.682	0.3577			0.7744	0.796	1.1956	0.4945
Node 9			1.1947	0.5897	1.1521	0.7085			1.3339	0.3286	0.2661	0.9479
Node 10					1.2276	0.5673					0.3961	0.9203
Node 11					1.6324	0.3263					0.2554	0.9566
Total Entrop	1.322		1.1333		1.1117		1.4971		1.395		1.309	
Total Purity	0.5277		0.5972		0.6233		0.4457		0.4971		0.5301	
Variance	[6.750157]		[6.35681407]		[6.11714175]		[6.261553]		[6.84667345]		[6.02618212]	
	[6.40938702]		[5.84998509]		[4.61812933]		[6.61231547]		[6.50315863]		[6.49494757]	
	[6.58244662]		[5.02246551]		[6.95197926]		[6.55231755]		[6.25571656]		[6.61311954]	
	[6.04124185]		[6.81180988]		[6.4792368]		[6.72718911]		[6.41985491]		[6.13118276]	
	[6.93782202]		[6.81197983]		[5.57726198]		[6.51882734]		[6.84328915]		[5.88463165]	
	[5.26387373]		[6.43930811]		[6.66615314]		[6.64883158]		[6.54161479]		[6.16642802]	
	[7.06257577]		[6.37253468]		[6.62557659]		[6.89420084]		[6.42878442]		[6.36014801]	
			[6.97400923]		[5.87242778]				[6.0672367]		[6.60459742]	
			[6.44675348]		[7.06665126]				[6.72630396]		[6.51407407]	
			[6.24937239]		[4.61855795]				[6.44480961]		[6.66391477]	
					[6.44568983]						[6.63682428]	
					[6.05418777]						[6.53229741]	

6 Appendix

In [ ]:
```
# import dataset and seperate them as train set and test set
# index x represents image, index y represents label
import os
import cv2
import random
import sklearn
import numpy as np
import sklearn.metrics
import tensorflow as tf
from numpy.linalg import *
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
```

In [2]:
```
# download MNIST dataset from keras
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# convert data type to float 32
x_train=np.float32(x_train)/255.0
x_test=np.float32(x_test)/255.0
x_train = x_train.reshape(np.shape(x_train)[0], 28*28)
x_test = x_test.reshape(np.shape(x_test)[0], 28*28)
# make sure the 10 classes
label_dict = {0: '0', 1: '1', 2: '2', 3: '3', 4: '4',
              5: '5', 6: '6', 7: '7', 8: '8', 9: '9'}
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

In [0]:
```
class SJKMeans(object):
  def _init_(self, x_tr, y_tr, K_num):
    self.x = x_tr
    self.y = y_tr
    self.K_num = int(K_num)

  def pick_init_mean(self):
    rownum = np.shape(self.x)[0]
    KM=np.zeros((self.K_num,784))
    for i in range(self.K_num):
      h = random.randint(0,rownum)
      KM[i,:] = self.x[h]
    return KM

  def calculate_dist(self, index, KM, method):

    if method == 'E':

      kdist = np.sqrt(np.sum((self.x[index]-KM)**2,axis=1))
```

```python
      km_ind = np.where(kdist == min(kdist))[0][0]

    elif method == 'M':

      kdist = np.sum(np.abs(self.x[index]-KM),axis=1)
      km_ind = np.where(kdist == min(kdist))[0][0]

    else: print("Error input. Please try between Euclidean Distance and Manhattan Dista
    return kdist, km_ind

  def assign_pnt(self, pnt_dict, km_ind, xtr_ind):
    pnt_dict[km_ind].append(self.y[xtr_ind])

    return pnt_dict

  def cal_mean_diff(self, old_kmean, K_metrix, count, threshold):
    kill=0
    new_kmean=np.zeros((self.K_num,784))
    new_var=np.zeros((self.K_num,1))
    for node in range(self.K_num):
      new_kmean[node] = np.mean(K_metrix[node,0:int(count[node]),:], axis=0)
      new_var[node] = np.mean(np.sqrt(np.sum((K_metrix[node,0:int(count[node]),:]
                                              - new_kmean[node])**2,axis=1)))
    '''
    diff = np.sqrt(np.sum((new_kmean-old_kmean)**2,axis=1))
    ready_node=[]
    for s in range(self.K_num):
      if (diff[s]<threshold):
        kill+=1
        ready_node.append(s)
        #print("The node %d is ready." %(s))
    old_kmean = new_kmean
    return old_kmean, kill, ready_node, new_var
    '''
    old_kmean = new_kmean
    return old_kmean, new_var

  def entropy_purity(self, pick_one, node_ind, ttl_entro, ttl_purty):
    from collections import Counter

    print("This include the following categories:", Counter(pick_one).keys())
    cate_distri = Counter(pick_one).values()
    print("Categories distribution:", cate_distri)
    cate_dis = np.multiply(list(cate_distri),1/len(pick_one))
    ln_cat_dis = np.log(cate_dis)
    entro = -np.sum(np.multiply(cate_dis, ln_cat_dis))
    ttl_entro.append(entro*len(pick_one))
    print("The entropy of node %d is %4f" %(node_ind, entro))
```

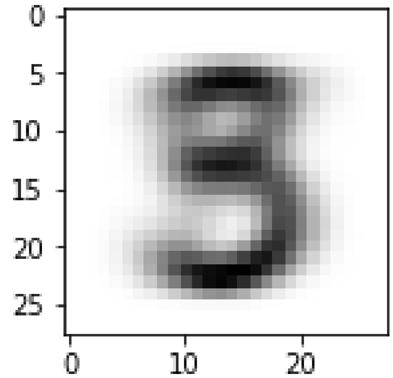
```
        purity = np.max(cate_dis)
        ttl_purty.append(purity*len(pick_one))

        dict_keys=list(Counter(pick_one).keys())
        dict_values=list(Counter(pick_one).values())

        cate_name = dict_keys[np.where(dict_values==np.max(dict_values))[0][0]]

        print("The purity of node %d is category %d with possibility %4f"
              %(node_ind, cate_name, purity))
        return ttl_entro, ttl_purty
```

```
In [25]: K_num = input("the cluster number is:")
         SJ = SJKMeans()
         SJ._init_(x_train, y_train, K_num)
         KM=SJ.pick_init_mean()
         agn_num = np.zeros(int(K_num))
         iter_num = input("select iteration steps:")
         threshold = input("write down the expected threshold:")
         method = input("write E for Euclidean Distance, M for Manhattan Distance:")
         for it in range(int(iter_num)):
           count = np.zeros(int(K_num))
           K_metrix = np.zeros([int(K_num),30010,784])
           pnt_dict = {}
           for kk in range(int(K_num)):
             pnt_dict[kk]=[]
           for j in range(np.shape(x_train)[0]):
             kdist, km_ind = SJ.calculate_dist(j, KM, method)
             K_metrix[km_ind,int(count[km_ind]),:] = x_train[j,]
             count[km_ind]+=1
             pnt_dict = SJ.assign_pnt(pnt_dict, km_ind, j)
           #KM, kill, ready_node, new_var = SJ.cal_mean_diff(KM,
           #K_metrix, count, float(threshold))
           KM, new_var = SJ.cal_mean_diff(KM, K_metrix, count, float(threshold))
           '''
           KM, kill, ready_node, new_var = SJ.cal_mean_diff(KM, K_metrix,
           count, float(threshold))
           if (it%20==0):
             print("The point distribution of each node")
             print(count)
             print("Need more iteration")
             print("The following nodes are ready.", ready_node)
           if kill == int(K_num):
             break
           '''
           if (it%20==0):
             print("The point distribution of each node")
```

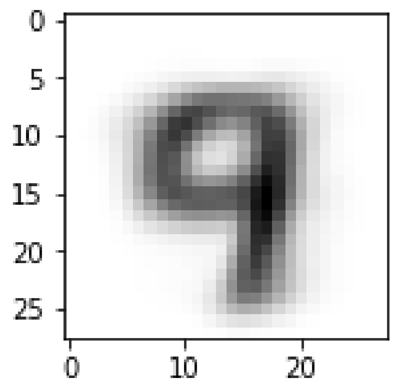
```python
                print(count)
                node_ls=count-agn_num
                lj_node = np.where(node_ls==0)[0]
                print("These nodes are stable,",lj_node)
                if (len(lj_node) == int(K_num)):
                  print("All nodes are ready.")
                  break
                else:
                  print("Need more iteration")
              agn_num = count
            ttl_entro = []
            ttl_purty = []
            for i in range(int(K_num)):
              ttl_entro, ttl_purty = SJ.entropy_purity(pnt_dict[i], i, ttl_entro, ttl_purty)
            Total_Entropy = np.sum(np.multiply(ttl_entro, (1/np.shape(x_train)[0])))
            Total_Purity = np.sum(np.multiply(ttl_purty, (1/np.shape(x_train)[0])))
            print("The final total entropy is: %4f" %(Total_Entropy))
            print("The final total purity is: %4f" %(Total_Purity))
            print("the final variance is ", new_var)
            print("the final k-mean is ", KM)
            for i in range(int(K_num)):
              tupian = np.reshape(np.round(KM[i]*255.),(28,28))
              plt.figure(figsize=(10,10))
              plt.subplot(3,4,i+1)
              #plt.grid(False)
              plt.imshow(tupian, cmap=plt.cm.binary)
              plt.xlabel("The %dth node " %(i))
              plt.show()
```

```
the cluster number is:7
select iteration steps:160
write down the expected threshold:0.1
write E for Euclidean Distance, M for Manhattan Distance:E
The point distribution of each node
[10329. 14398.  7738.  3741.  6746. 14362.  2686.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[ 9885.  9118.  6222. 10111.  5216. 11151.  8297.]
These nodes are stable, [4]
Need more iteration
The point distribution of each node
[ 9895.  9187.  6224. 10124.  5216. 11096.  8258.]
These nodes are stable, [0 1 2 3 4 5 6]
All nodes are ready.
This include the following categories: dict_keys([5, 3, 8, 9, 0, 2, 6, 7, 1, 4])
Categories distribution: dict_values([2579, 4439, 1960, 117, 322, 397, 56, 6, 17, 2])
The entropy of node 0 is 1.370165
```

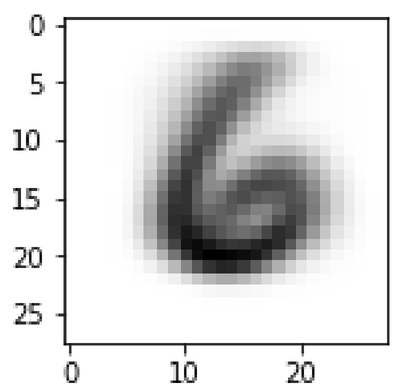
```
The purity of node 0 is category 3 with possibility 0.448610
This include the following categories: dict_keys([4, 7, 3, 5, 9, 1, 2, 6, 0, 8])
Categories distribution: dict_values([3217, 1941, 183, 392, 2910, 14, 188, 80, 41, 221])
The entropy of node 1 is 1.517234
The purity of node 1 is category 4 with possibility 0.350169
This include the following categories: dict_keys([6, 4, 2, 0, 5, 8, 3, 7, 9, 1])
Categories distribution: dict_values([5046, 173, 451, 249, 147, 54, 74, 7, 11, 12])
The entropy of node 2 is 0.801894
The purity of node 2 is category 6 with possibility 0.810733
This include the following categories: dict_keys([9, 4, 7, 2, 5, 8, 3, 1, 0, 6])
Categories distribution: dict_values([2596, 2075, 3738, 72, 922, 569, 96, 14, 41, 1])
The entropy of node 3 is 1.533383
The purity of node 3 is category 7 with possibility 0.369222
This include the following categories: dict_keys([0, 5, 6, 3, 9, 2, 4, 7, 8])
Categories distribution: dict_values([4816, 82, 85, 39, 52, 71, 13, 21, 37])
The entropy of node 4 is 0.419324
The purity of node 4 is category 0 with possibility 0.923313
This include the following categories: dict_keys([1, 5, 7, 4, 8, 2, 6, 0, 9, 3])
Categories distribution: dict_values([6515, 915, 457, 289, 808, 886, 496, 34, 235, 461])
The entropy of node 5 is 1.507849
The purity of node 5 is category 1 with possibility 0.587149
This include the following categories: dict_keys([2, 3, 8, 0, 5, 7, 1, 6, 4, 9])
Categories distribution: dict_values([3893, 839, 2202, 420, 384, 95, 170, 154, 73, 28])
The entropy of node 6 is 1.500117
The purity of node 6 is category 2 with possibility 0.471422
The final total entropy is: 1.321964
The final total purity is: 0.527733
the final variance is  [[6.750157  ]
 [6.40938702]
 [6.58244662]
 [6.04124185]
 [6.93782202]
 [5.26387373]
 [7.06257577]]
the final k-mean is  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```



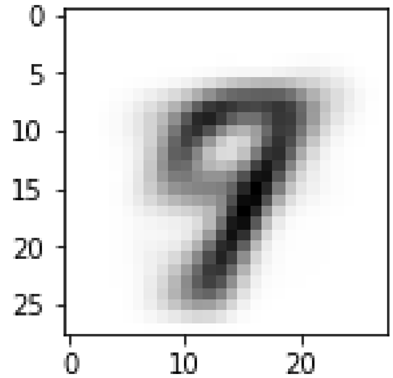
The 0th node



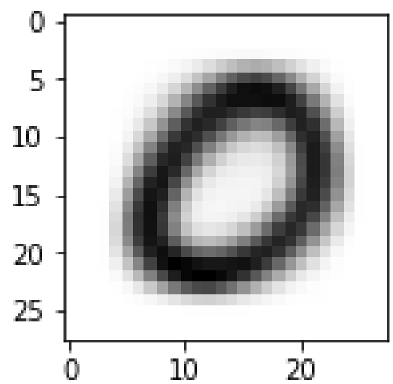
The 1th node



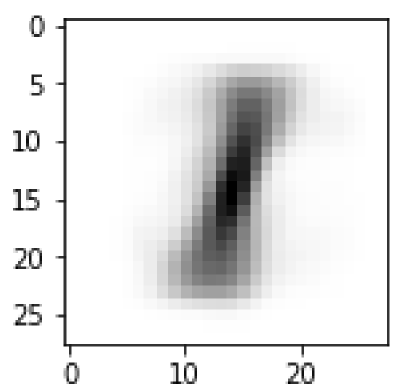
The 2th node



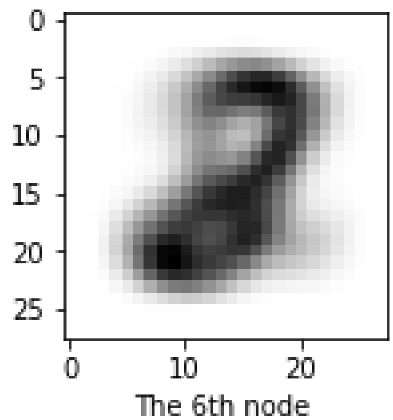
The 3th node



The 4th node



The 5th node



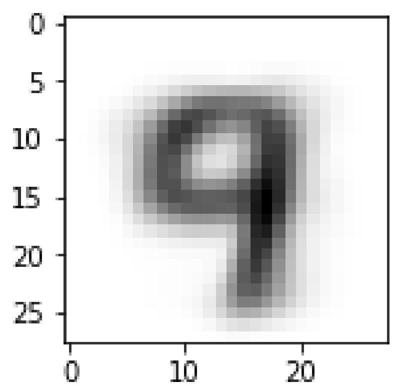
In [27]:
```
K_num = input("the cluster number is:")
SJ = SJKMeans()
SJ._init_(x_train, y_train, K_num)
KM=SJ.pick_init_mean()
agn_num = np.zeros(int(K_num))
iter_num = input("select iteration steps:")
threshold = input("write down the expected threshold:")
method = input("write E for Euclidean Distance, M for Manhattan Distance:")
for it in range(int(iter_num)):
  count = np.zeros(int(K_num))
  K_metrix = np.zeros([int(K_num),30010,784])
  pnt_dict = {}
  for kk in range(int(K_num)):
    pnt_dict[kk]=[]
  for j in range(np.shape(x_train)[0]):
    kdist, km_ind = SJ.calculate_dist(j, KM, method)
    K_metrix[km_ind,int(count[km_ind]),:] = x_train[j,]
    count[km_ind]+=1
    pnt_dict = SJ.assign_pnt(pnt_dict, km_ind, j)
  #KM, kill, ready_node, new_var = SJ.cal_mean_diff(
  #KM, K_metrix, count, float(threshold))
  KM, new_var = SJ.cal_mean_diff(KM, K_metrix, count, float(threshold))
  '''
  KM, kill, ready_node, new_var = SJ.cal_mean_diff(
  KM, K_metrix, count, float(threshold))
  if (it%20==0):
    print("The point distribution of each node")
    print(count)
    print("Need more iteration")
    print("The following nodes are ready.", ready_node)
  if kill == int(K_num):
    break
```

```
            '''
            if (it%20==0):
              print("The point distribution of each node")
              print(count)
              node_ls=count-agn_num
              lj_node = np.where(node_ls==0)[0]
              print("These nodes are stable,",lj_node)
              if (len(lj_node) == int(K_num)):
                print("All nodes are ready.")
                break
              else:
                print("Need more iteration")
            agn_num = count
          ttl_entro = []
          ttl_purty = []
          for i in range(int(K_num)):
            ttl_entro, ttl_purty = SJ.entropy_purity(pnt_dict[i], i, ttl_entro, ttl_purty)
          Total_Entropy = np.sum(np.multiply(ttl_entro, (1/np.shape(x_train)[0])))
          Total_Purity = np.sum(np.multiply(ttl_purty, (1/np.shape(x_train)[0])))
          print("The final total entropy is: %4f" %(Total_Entropy))
          print("The final total purity is: %4f" %(Total_Purity))
          print("the final variance is ", new_var)
          print("the final k-mean is ", KM)
          for i in range(int(K_num)):
            tupian = np.reshape(np.round(KM[i]*255.),(28,28))
            plt.figure(figsize=(10,10))
            plt.subplot(3,4,i+1)
            #plt.grid(False)
            plt.imshow(tupian, cmap=plt.cm.binary)
            plt.xlabel("The %dth node " %(i))
            plt.show()
```

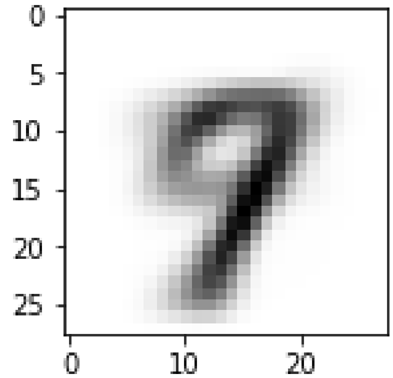
```
the cluster number is:10
select iteration steps:160
write down the expected threshold:0.1
write E for Euclidean Distance, M for Manhattan Distance:E
The point distribution of each node
[13518.  9595. 10651.  3400.  3947.  1667.  4662.  5981.  4057.  2522.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[8976. 8478. 9611. 4479. 4866. 5155. 5001. 4508. 4580. 4346.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[8894. 8483. 9435. 4539. 5032. 5357. 4713. 4475. 4324. 4748.]
These nodes are stable, [3]
Need more iteration
```

```
The point distribution of each node
[8896. 8493. 9433. 4539. 5027. 5345. 4693. 4467. 4295. 4812.]
These nodes are stable, [1 3 6 7 8]
Need more iteration
The point distribution of each node
[8900. 8483. 9423. 4536. 5051. 5347. 4670. 4463. 4290. 4837.]
These nodes are stable, [1 2 3 5 8]
Need more iteration
The point distribution of each node
[8901. 8483. 9422. 4536. 5053. 5347. 4670. 4463. 4287. 4838.]
These nodes are stable, [0 1 2 3 4 5 6 7 8 9]
All nodes are ready.
This include the following categories: dict_keys([4, 7, 3, 9, 1, 2, 5, 8, 6, 0])
Categories distribution: dict_values([3185, 1835, 171, 2938, 16, 167, 356, 149, 53, 31])
The entropy of node 0 is 1.468471
The purity of node 0 is category 4 with possibility 0.357825
This include the following categories: dict_keys([9, 7, 4, 3, 2, 8, 5, 1, 6, 0])
Categories distribution: dict_values([2480, 3836, 1775, 38, 64, 126, 144, 10, 3, 7])
The entropy of node 1 is 1.255153
The purity of node 1 is category 7 with possibility 0.452199
This include the following categories: dict_keys([1, 5, 7, 4, 2, 8, 6, 0, 3, 9])
Categories distribution: dict_values([6594, 284, 442, 221, 675, 508, 173, 5, 300, 220])
The entropy of node 2 is 1.208056
The purity of node 2 is category 1 with possibility 0.699851
This include the following categories: dict_keys([0, 5, 6, 3, 9, 4, 2, 7, 8])
Categories distribution: dict_values([4287, 43, 55, 22, 38, 8, 44, 19, 20])
The entropy of node 3 is 0.319930
The purity of node 3 is category 0 with possibility 0.945106
This include the following categories: dict_keys([3, 2, 9, 0, 8, 5, 6, 7, 4, 1])
Categories distribution: dict_values([2097, 244, 64, 234, 1054, 1313, 29, 8, 6, 4])
The entropy of node 4 is 1.439542
The purity of node 4 is category 3 with possibility 0.415001
This include the following categories: dict_keys([4, 6, 1, 0, 5, 2, 8, 9, 3, 7])
Categories distribution: dict_values([450, 1746, 10, 899, 1480, 252, 261, 71, 156, 22])
The entropy of node 5 is 1.715306
The purity of node 5 is category 6 with possibility 0.326538
This include the following categories: dict_keys([8, 3, 1, 5, 2, 7, 9, 4, 0, 6])
Categories distribution: dict_values([3371, 340, 54, 568, 172, 52, 53, 27, 24, 9])
The entropy of node 6 is 1.025278
The purity of node 6 is category 8 with possibility 0.721842
This include the following categories: dict_keys([2, 7, 3, 4, 6, 9, 8, 1, 0, 5])
Categories distribution: dict_values([4106, 46, 127, 33, 56, 18, 33, 33, 6, 5])
The entropy of node 7 is 0.427666
The purity of node 7 is category 2 with possibility 0.920009
This include the following categories: dict_keys([6, 4, 2, 0, 8, 5, 3, 7, 9, 1])
Categories distribution: dict_values([3757, 136, 116, 150, 26, 55, 27, 4, 9, 7])
The entropy of node 8 is 0.588802
The purity of node 8 is category 6 with possibility 0.876370
```

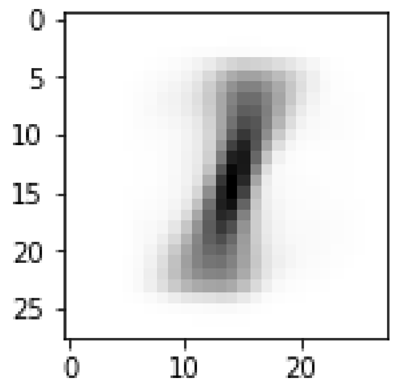
```
This include the following categories: dict_keys([5, 3, 0, 8, 2, 9, 4, 6, 1, 7])
Categories distribution: dict_values([1173, 2853, 280, 303, 118, 58, 1, 37, 14, 1])
The entropy of node 9 is 1.194717
The purity of node 9 is category 3 with possibility 0.589706
The final total entropy is: 1.133309
The final total purity is: 0.597200
the final variance is  [[6.35681407]
 [5.84998509]
 [5.02246551]
 [6.81180988]
 [6.81197983]
 [6.43930811]
 [6.37253468]
 [6.97400923]
 [6.44675348]
 [6.24937239]]
the final k-mean is  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```



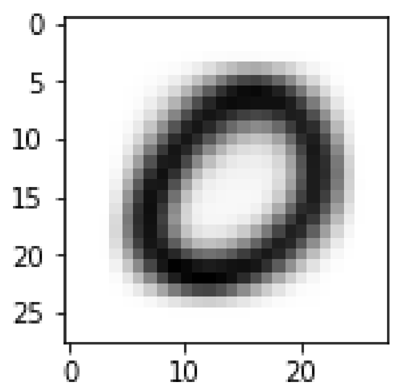
The 0th node



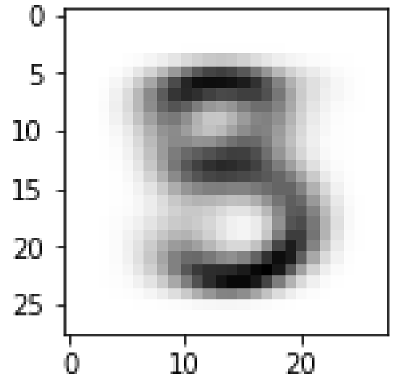
The 1th node



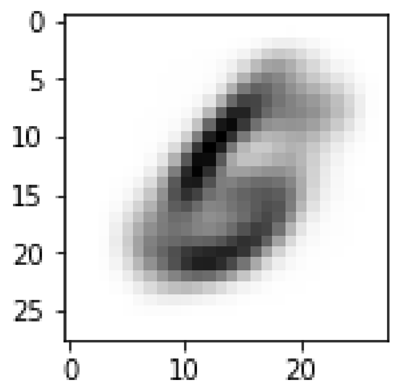
The 2th node



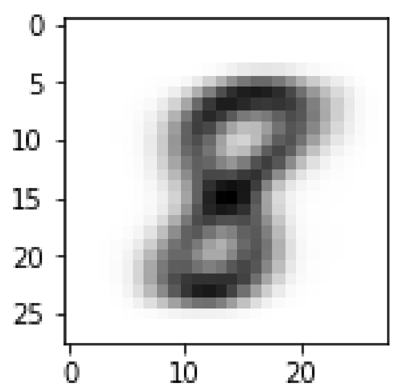
The 3th node



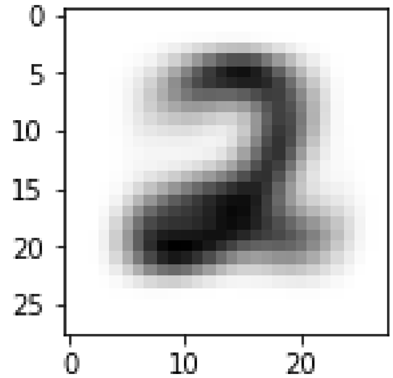
The 4th node



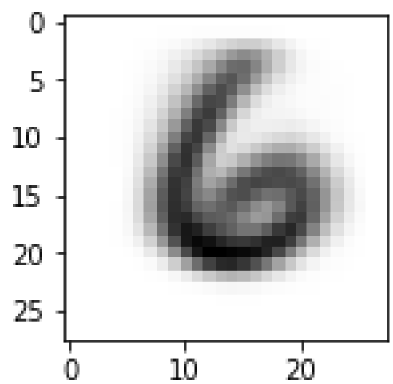
The 5th node



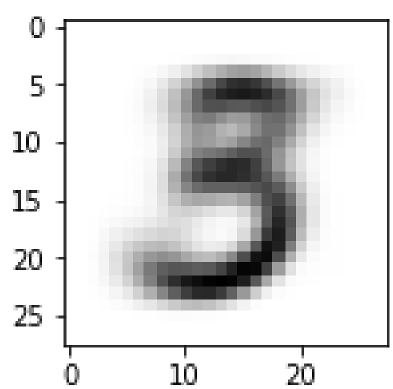
The 6th node



The 7th node



The 8th node



The 9th node

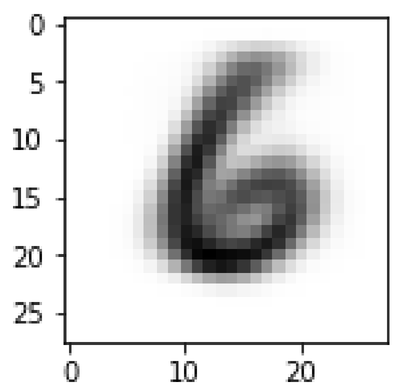
```
In [29]: K_num = input("the cluster number is:")
         SJ = SJKMeans()
         SJ._init_(x_train, y_train, K_num)
         KM=SJ.pick_init_mean()
         agn_num = np.zeros(int(K_num))
         iter_num = input("select iteration steps:")
         threshold = input("write down the expected threshold:")
         method = input("write E for Euclidean Distance, M for Manhattan Distance:")
         for it in range(int(iter_num)):
           count = np.zeros(int(K_num))
           K_metrix = np.zeros([int(K_num),30010,784])
           pnt_dict = {}
           for kk in range(int(K_num)):
             pnt_dict[kk]=[]
           for j in range(np.shape(x_train)[0]):
             kdist, km_ind = SJ.calculate_dist(j, KM, method)
             K_metrix[km_ind,int(count[km_ind]),:] = x_train[j,]
             count[km_ind]+=1
             pnt_dict = SJ.assign_pnt(pnt_dict, km_ind, j)
           #KM, kill, ready_node, new_var = SJ.cal_mean_diff(
           #KM, K_metrix, count, float(threshold))
           KM, new_var = SJ.cal_mean_diff(KM, K_metrix, count, float(threshold))
           '''
           KM, kill, ready_node, new_var = SJ.cal_mean_diff(
           KM, K_metrix, count, float(threshold))
           if (it%20==0):
             print("The point distribution of each node")
             print(count)
             print("Need more iteration")
             print("The following nodes are ready.", ready_node)
           if kill == int(K_num):
             break
           '''
           if (it%20==0):
             print("The point distribution of each node")
             print(count)
             node_ls=count-agn_num
             lj_node = np.where(node_ls==0)[0]
             print("These nodes are stable,",lj_node)
             if (len(lj_node) == int(K_num)):
               print("All nodes are ready.")
               break
             else:
               print("Need more iteration")
           agn_num = count
         ttl_entro = []
         ttl_purty = []
         for i in range(int(K_num)):
```

```
      ttl_entro, ttl_purty = SJ.entropy_purity(pnt_dict[i], i, ttl_entro, ttl_purty)
    Total_Entropy = np.sum(np.multiply(ttl_entro, (1/np.shape(x_train)[0])))
    Total_Purity = np.sum(np.multiply(ttl_purty, (1/np.shape(x_train)[0])))
    print("The final total entropy is:", Total_Entropy)
    print("The final total purity is:", Total_Purity)
    print("the final variance is ", new_var)
    print("the final k-mean is ", KM)
    for i in range(int(K_num)):
      tupian = np.reshape(np.round(KM[i]*255.),(28,28))
      plt.figure(figsize=(10,10))
      plt.subplot(3,4,i+1)
      plt.grid(False)
      plt.imshow(tupian, cmap=plt.cm.binary)
      plt.xlabel("The %dth node " %(i))
      plt.show()
```

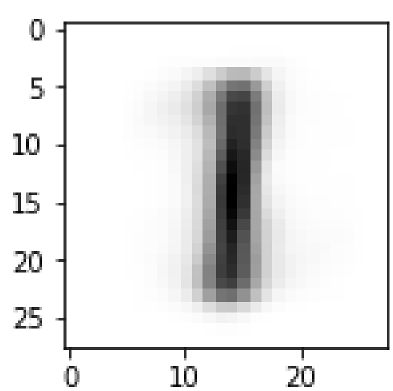
```
the cluster number is:12
select iteration steps:160
write down the expected threshold:0.1
write E for Euclidean Distance, M for Manhattan Distance:E
The point distribution of each node
[ 3866. 17528.  3101.  4847.  2027.  1242.  2704. 11386.  3351.  1511.
  5700.  2737.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[5115. 5648. 4535. 3115. 3992. 6894. 2967. 6015. 4271. 5551. 6606. 5291.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[4962. 5577. 4557. 3080. 4034. 7304. 2949. 6429. 4318. 4587. 6122. 6081.]
These nodes are stable, [ 3  6 10]
Need more iteration
The point distribution of each node
[4884. 5648. 4574. 3070. 4098. 7291. 2934. 6644. 4136. 4429. 6063. 6229.]
These nodes are stable, [3 5 6]
Need more iteration
The point distribution of each node
[4889. 5638. 4567. 3059. 4155. 7290. 2935. 6697. 4144. 4319. 6027. 6280.]
These nodes are stable, [ 0  2  3  6  8 10]
Need more iteration
The point distribution of each node
[4893. 5629. 4574. 3052. 4168. 7286. 2934. 6709. 4143. 4268. 6036. 6308.]
These nodes are stable, [ 0  1  2  3  4  5  6  7  8  9 10 11]
All nodes are ready.
This include the following categories: dict_keys([6, 2, 0, 5, 4, 8, 9, 3, 1, 7])
Categories distribution: dict_values([4216, 126, 212, 135, 89, 35, 8, 60, 11, 1])
The entropy of node 0 is 0.645730
```

```
The purity of node 0 is category 6 with possibility 0.861639
This include the following categories: dict_keys([1, 3, 7, 5, 4, 2, 8, 6, 9, 0])
Categories distribution: dict_values([3662, 401, 267, 164, 98, 352, 288, 221, 174, 2])
The entropy of node 1 is 1.348855
The purity of node 1 is category 1 with possibility 0.650560
This include the following categories: dict_keys([2, 3, 5, 8, 4, 7, 1, 6, 0, 9])
Categories distribution: dict_values([4181, 237, 16, 46, 6, 37, 11, 22, 15, 3])
The entropy of node 2 is 0.412945
The purity of node 2 is category 2 with possibility 0.914080
This include the following categories: dict_keys([0, 2, 5, 3, 9, 7, 6, 8, 4])
Categories distribution: dict_values([2483, 98, 195, 116, 16, 15, 97, 26, 6])
The entropy of node 3 is 0.794413
The purity of node 3 is category 0 with possibility 0.813565
This include the following categories: dict_keys([7, 2, 9, 3, 4, 8, 1, 0, 5])
Categories distribution: dict_values([3583, 86, 419, 29, 17, 22, 5, 1, 6])
The entropy of node 4 is 0.545203
The purity of node 4 is category 7 with possibility 0.859645
This include the following categories: dict_keys([3, 9, 2, 1, 5, 8, 0, 6, 4, 7])
Categories distribution: dict_values([3894, 82, 262, 6, 1796, 1063, 152, 27, 2, 2])
The entropy of node 5 is 1.242773
The purity of node 5 is category 3 with possibility 0.534450
This include the following categories: dict_keys([0, 5, 6, 3, 9, 8, 2, 7, 4])
Categories distribution: dict_values([2742, 52, 66, 15, 14, 30, 10, 4, 1])
The entropy of node 6 is 0.350501
The purity of node 6 is category 0 with possibility 0.934560
This include the following categories: dict_keys([9, 4, 7, 3, 5, 8, 2, 0, 1, 6])
Categories distribution: dict_values([2648, 2018, 1282, 182, 328, 188, 37, 15, 7, 4])
The entropy of node 7 is 1.444031
The purity of node 7 is category 9 with possibility 0.394694
This include the following categories: dict_keys([4, 2, 7, 6, 3, 0, 9, 5, 8])
Categories distribution: dict_values([1482, 217, 316, 1064, 27, 100, 727, 105, 105])
The entropy of node 8 is 1.681964
The purity of node 8 is category 4 with possibility 0.357712
This include the following categories: dict_keys([1, 5, 4, 8, 6, 2, 7, 9, 3, 0])
Categories distribution: dict_values([3024, 207, 59, 229, 114, 327, 217, 42, 45, 4])
The entropy of node 9 is 1.152099
The purity of node 9 is category 1 with possibility 0.708529
This include the following categories: dict_keys([5, 3, 8, 2, 1, 9, 0, 4, 6, 7])
Categories distribution: dict_values([1148, 1049, 3424, 163, 9, 49, 125, 6, 51, 12])
The entropy of node 10 is 1.227564
The purity of node 10 is category 8 with possibility 0.567263
This include the following categories: dict_keys([4, 9, 7, 5, 8, 2, 3, 6, 0, 1])
Categories distribution: dict_values([2058, 1767, 529, 1269, 395, 99, 76, 36, 72, 7])
The entropy of node 11 is 1.632376
The purity of node 11 is category 4 with possibility 0.326252
The final total entropy is: 1.11168982061735
The final total purity is: 0.6232833333333334
the final variance is  [[6.11714175]
```

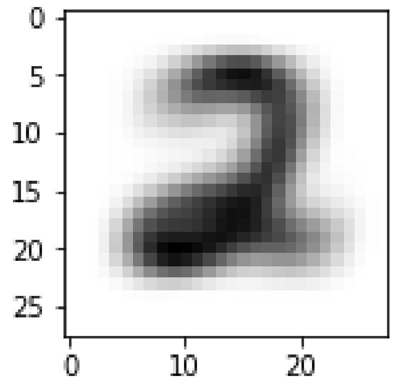
```
 [4.61812933]
 [6.95197926]
 [6.4792368 ]
 [5.57726198]
 [6.66615314]
 [6.62557659]
 [5.87242778]
 [7.06665126]
 [4.61855795]
 [6.44568983]
 [6.05418777]]
the final k-mean is  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```



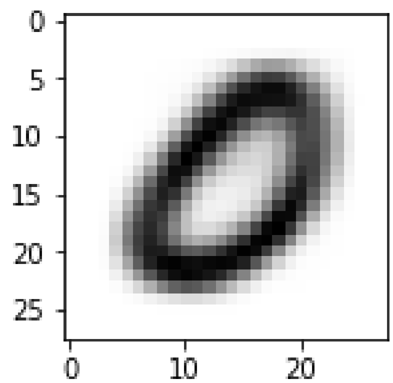
The 0th node



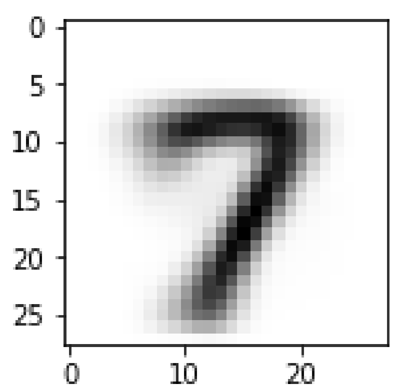
The 1th node



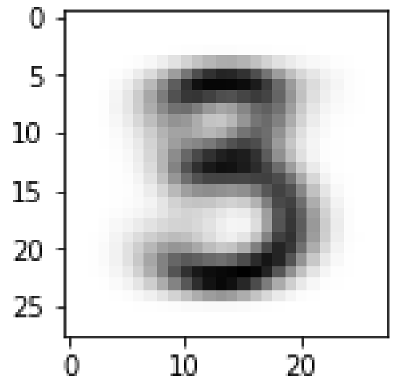
The 2th node



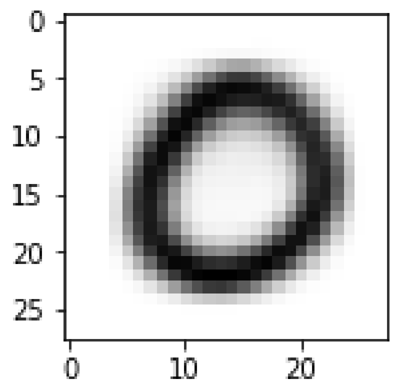
The 3th node



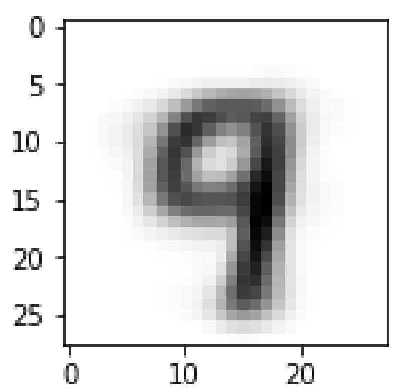
The 4th node



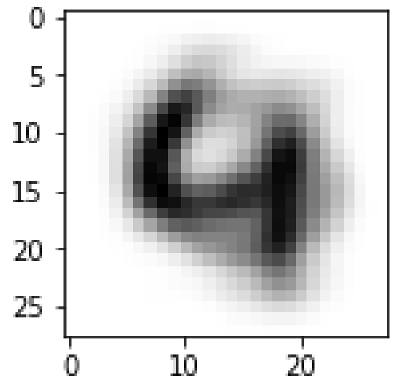
The 5th node



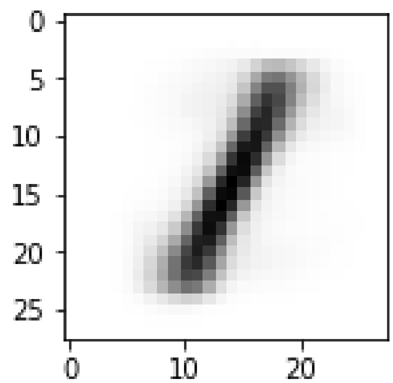
The 6th node



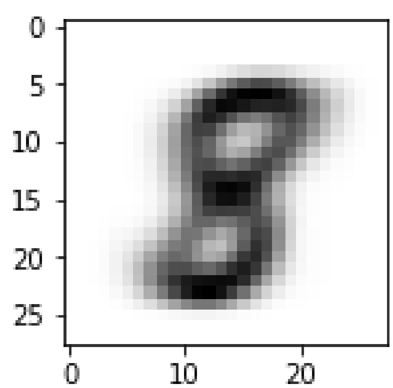
The 7th node



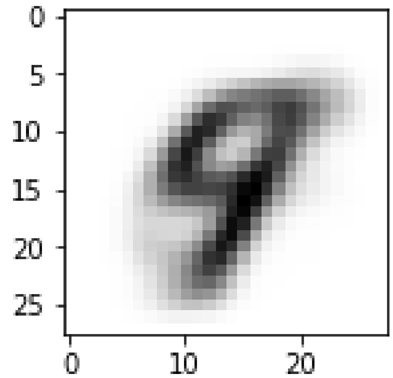
The 8th node



The 9th node



The 10th node



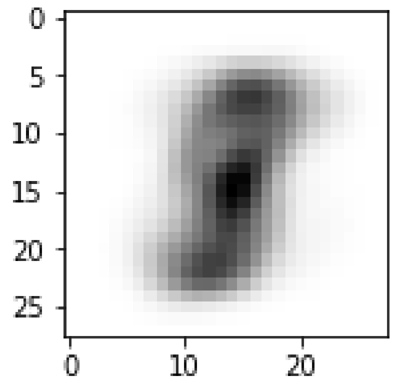

```
In [31]: K_num = input("the cluster number is:")
         SJ = SJKMeans()
         SJ._init_(x_train, y_train, K_num)
         KM=SJ.pick_init_mean()
         agn_num = np.zeros(int(K_num))
         iter_num = input("select iteration steps:")
         threshold = input("write down the expected threshold:")
         method = input("write E for Euclidean Distance, M for Manhattan Distance:")
         for it in range(int(iter_num)):
           count = np.zeros(int(K_num))
           K_metrix = np.zeros([int(K_num),30010,784])
           pnt_dict = {}
           for kk in range(int(K_num)):
             pnt_dict[kk]=[]
           for j in range(np.shape(x_train)[0]):
             kdist, km_ind = SJ.calculate_dist(j, KM, method)
             K_metrix[km_ind,int(count[km_ind]),:] = x_train[j,]
             count[km_ind]+=1
             pnt_dict = SJ.assign_pnt(pnt_dict, km_ind, j)
           #KM, kill, ready_node, new_var = SJ.cal_mean_diff(
           #KM, K_metrix, count, float(threshold))
           KM, new_var = SJ.cal_mean_diff(KM, K_metrix, count, float(threshold))
           '''
           KM, kill, ready_node, new_var = SJ.cal_mean_diff(
           KM, K_metrix, count, float(threshold))
           if (it%20==0):
             print("The point distribution of each node")
             print(count)
             print("Need more iteration")
```

```
                print("The following nodes are ready.", ready_node)
            if kill == int(K_num):
                break
            '''
            if (it%20==0):
                print("The point distribution of each node")
                print(count)
                node_ls=count-agn_num
                lj_node = np.where(node_ls==0)[0]
                print("These nodes are stable,",lj_node)
                if (len(lj_node) == int(K_num)):
                    print("All nodes are ready.")
                    break
                else:
                    print("Need more iteration")
            agn_num = count
        ttl_entro = []
        ttl_purty = []
        for i in range(int(K_num)):
            ttl_entro, ttl_purty = SJ.entropy_purity(pnt_dict[i], i, ttl_entro, ttl_purty)
        Total_Entropy = np.sum(np.multiply(ttl_entro, (1/np.shape(x_train)[0])))
        Total_Purity = np.sum(np.multiply(ttl_purty, (1/np.shape(x_train)[0])))
        print("The final total entropy is:", Total_Entropy)
        print("The final total purity is:", Total_Purity)
        print("the final variance is ", new_var)
        print("the final k-mean is ", KM)
        for i in range(int(K_num)):
            tupian = np.reshape(np.round(KM[i]*255.),(28,28))
            plt.figure(figsize=(10,10))
            plt.subplot(3,4,i+1)
            plt.grid(False)
            plt.imshow(tupian, cmap=plt.cm.binary)
            plt.xlabel("The %dth node " %(i))
            plt.show()
```

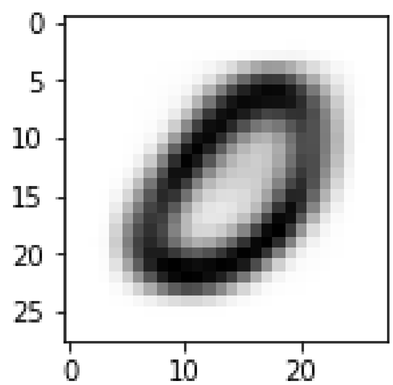
```
the cluster number is:7
select iteration steps:160
write down the expected threshold:0.1
write E for Euclidean Distance, M for Manhattan Distance:M
The point distribution of each node
[14798.  8750.  4188.  7412. 19049.  3702.  2101.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[24804.  3474.  4819.  6102. 13912.  2696.  4193.]
These nodes are stable, []
Need more iteration
The point distribution of each node
```

```
[25739.  3413.  4900.  6359. 13517.  2904.  3168.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[25883.  3450.  5308.  6338. 12922.  2916.  3183.]
These nodes are stable, [1 2 3 5 6]
Need more iteration
The point distribution of each node
[25883.  3450.  5308.  6343. 12917.  2916.  3183.]
These nodes are stable, [0 1 2 3 4 5 6]
All nodes are ready.
This include the following categories: dict_keys([5, 1, 4, 3, 8, 6, 9, 7, 2, 0])
Categories distribution: dict_values([2897, 6733, 1657, 2273, 4278, 1270, 1879, 2300, 2295, 30
The entropy of node 0 is 2.102555
The purity of node 0 is category 1 with possibility 0.260132
This include the following categories: dict_keys([0, 2, 5, 3, 6, 9, 7, 8, 4])
Categories distribution: dict_values([2538, 128, 319, 197, 121, 18, 12, 107, 10])
The entropy of node 1 is 1.020965
The purity of node 1 is category 0 with possibility 0.735652
This include the following categories: dict_keys([6, 4, 2, 0, 5, 8, 3, 9, 1, 7])
Categories distribution: dict_values([4416, 174, 252, 236, 104, 50, 55, 11, 6, 4])
The entropy of node 2 is 0.742439
The purity of node 2 is category 6 with possibility 0.831952
This include the following categories: dict_keys([3, 9, 5, 0, 8, 2, 6, 7, 4])
Categories distribution: dict_values([3354, 70, 1647, 91, 1009, 157, 13, 1, 1])
The entropy of node 3 is 1.197115
The purity of node 3 is category 3 with possibility 0.528772
This include the following categories: dict_keys([4, 9, 7, 2, 3, 5, 8, 0, 6, 1])
Categories distribution: dict_values([3984, 3933, 3927, 112, 143, 405, 341, 42, 29, 1])
The entropy of node 4 is 1.415434
The purity of node 4 is category 4 with possibility 0.308431
This include the following categories: dict_keys([0, 5, 6, 3, 4, 7, 8, 9, 2])
Categories distribution: dict_values([2710, 44, 65, 13, 5, 11, 28, 31, 9])
The entropy of node 5 is 0.383014
The purity of node 5 is category 0 with possibility 0.929355
This include the following categories: dict_keys([2, 3, 8, 4, 9, 0, 5, 7, 1, 6])
Categories distribution: dict_values([3005, 96, 38, 11, 7, 5, 5, 10, 2, 4])
The entropy of node 6 is 0.297249
The purity of node 6 is category 2 with possibility 0.944078
The final total entropy is: 1.497051642495773
The final total purity is: 0.4456666666666667
the final variance is  [[6.261553  ]
 [6.61231547]
 [6.55231755]
 [6.72718911]
 [6.51882734]
 [6.64883158]
 [6.89420084]]
```

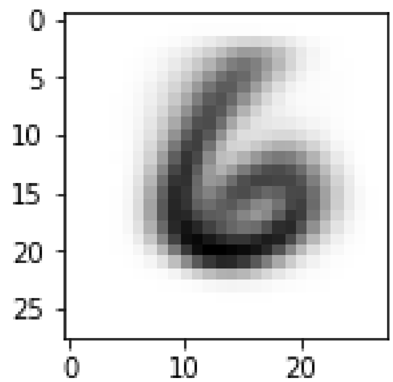
```
the final k-mean is  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```



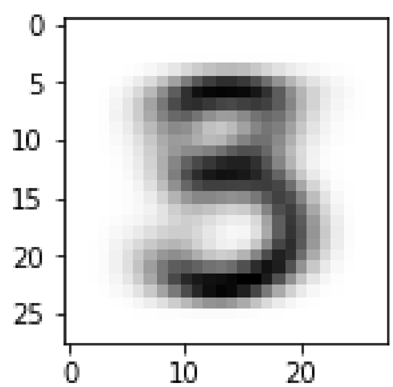
The 0th node



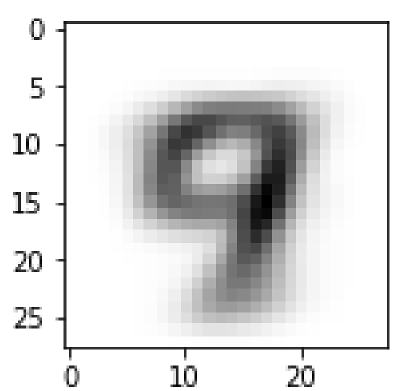
The 1th node



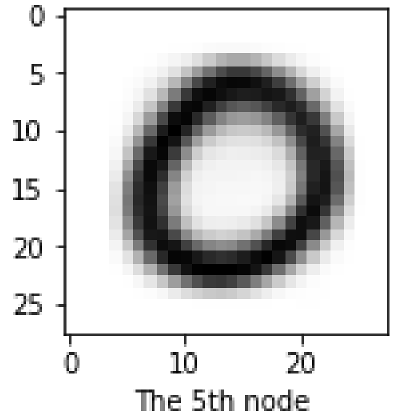
The 2th node



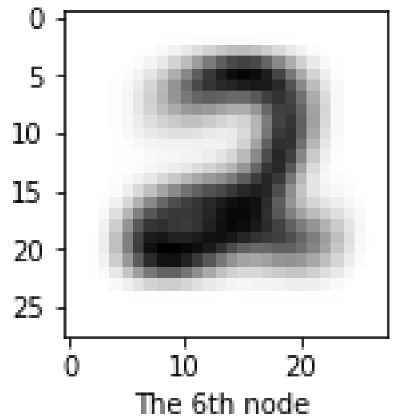
The 3th node



The 4th node



The 5th node



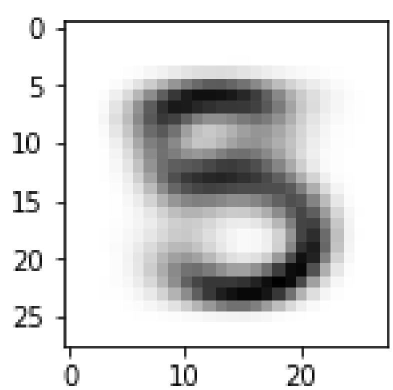
The 6th node

```
In [33]: K_num = input("the cluster number is:")
         SJ = SJKMeans()
         SJ._init_(x_train, y_train, K_num)
         KM=SJ.pick_init_mean()
         agn_num = np.zeros(int(K_num))
         iter_num = input("select iteration steps:")
         threshold = input("write down the expected threshold:")
         method = input("write E for Euclidean Distance, M for Manhattan Distance:")
         for it in range(int(iter_num)):
           count = np.zeros(int(K_num))
           K_metrix = np.zeros([int(K_num),30010,784])
           pnt_dict = {}
           for kk in range(int(K_num)):
             pnt_dict[kk]=[]
           for j in range(np.shape(x_train)[0]):
```

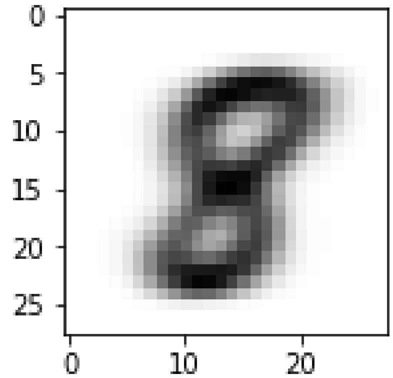
```
      kdist, km_ind = SJ.calculate_dist(j, KM, method)
      K_metrix[km_ind,int(count[km_ind]),:] = x_train[j,]
      count[km_ind]+=1
      pnt_dict = SJ.assign_pnt(pnt_dict, km_ind, j)
    #KM, kill, ready_node, new_var = SJ.cal_mean_diff(
    #KM, K_metrix, count, float(threshold))
    KM, new_var = SJ.cal_mean_diff(KM, K_metrix, count, float(threshold))
    '''
    KM, kill, ready_node, new_var = SJ.cal_mean_diff(
    KM, K_metrix, count, float(threshold))
    if (it%20==0):
      print("The point distribution of each node")
      print(count)
      print("Need more iteration")
      print("The following nodes are ready.", ready_node)
    if kill == int(K_num):
      break
    '''
    if (it%20==0):
      print("The point distribution of each node")
      print(count)
      node_ls=count-agn_num
      lj_node = np.where(node_ls==0)[0]
      print("These nodes are stable,",lj_node)
      if (len(lj_node) == int(K_num)):
        print("All nodes are ready.")
        break
      else:
        print("Need more iteration")
    agn_num = count
  ttl_entro = []
  ttl_purty = []
  for i in range(int(K_num)):
    ttl_entro, ttl_purty = SJ.entropy_purity(pnt_dict[i], i, ttl_entro, ttl_purty)
  Total_Entropy = np.sum(np.multiply(ttl_entro, (1/np.shape(x_train)[0])))
  Total_Purity = np.sum(np.multiply(ttl_purty, (1/np.shape(x_train)[0])))
  print("The final total entropy is:", Total_Entropy)
  print("The final total purity is:", Total_Purity)
  print("the final variance is ", new_var)
  print("the final k-mean is ", KM)
  for i in range(int(K_num)):
    tupian = np.reshape(np.round(KM[i]*255.),(28,28))
    plt.figure(figsize=(10,10))
    plt.subplot(3,4,i+1)
    plt.grid(False)
    plt.imshow(tupian, cmap=plt.cm.binary)
    plt.xlabel("The %dth node " %(i))
    plt.show()
```

```
the cluster number is:10
select iteration steps:160
write down the expected threshold:0.1
write E for Euclidean Distance, M for Manhattan Distance:M
The point distribution of each node
[ 3054.  4371.  3704.  2311.  2440.   739.  5512. 16861.  5297. 15711.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[ 3304.  3488.  5279.  2813.  2913.  2482.  5011. 22277.  1711. 10722.]
These nodes are stable, [1]
Need more iteration
The point distribution of each node
[ 1797.  3776.  5325.  2823.  2936.  2614.  5760. 22427.  1735. 10807.]
These nodes are stable, [2 3 5 8]
Need more iteration
The point distribution of each node
[ 1745.  3799.  5325.  2824.  2937.  2619.  5759. 22436.  1735. 10821.]
These nodes are stable, [0 1 2 3 4 5 6 7 8 9]
All nodes are ready.
This include the following categories: dict_keys([3, 0, 5, 8, 2, 6, 9, 4, 7])
Categories distribution: dict_values([667, 26, 606, 390, 23, 6, 24, 2, 1])
The entropy of node 0 is 1.280003
The purity of node 0 is category 3 with possibility 0.382235
This include the following categories: dict_keys([3, 8, 0, 2, 5, 9, 4, 6, 1, 7])
Categories distribution: dict_values([408, 2448, 47, 152, 580, 98, 28, 7, 1, 30])
The entropy of node 1 is 1.175414
The purity of node 1 is category 8 with possibility 0.644380
This include the following categories: dict_keys([6, 4, 2, 0, 5, 8, 9, 3, 1, 7])
Categories distribution: dict_values([3845, 268, 320, 499, 222, 62, 17, 78, 12, 2])
The entropy of node 2 is 1.057638
The purity of node 2 is category 6 with possibility 0.722066
This include the following categories: dict_keys([0, 2, 5, 9, 3, 7, 6, 8, 4])
Categories distribution: dict_values([2446, 101, 103, 17, 65, 13, 52, 21, 6])
The entropy of node 3 is 0.629797
The purity of node 3 is category 0 with possibility 0.866147
This include the following categories: dict_keys([2, 3, 4, 9, 8, 0, 5, 1, 7])
Categories distribution: dict_values([2844, 61, 5, 7, 8, 4, 2, 2, 4])
The entropy of node 4 is 0.180865
The purity of node 4 is category 2 with possibility 0.968335
This include the following categories: dict_keys([0, 5, 6, 3, 7, 8, 9, 2, 4])
Categories distribution: dict_values([2501, 25, 22, 10, 10, 18, 23, 9, 1])
The entropy of node 5 is 0.269409
The purity of node 5 is category 0 with possibility 0.954945
This include the following categories: dict_keys([3, 2, 0, 9, 5, 8, 7, 6, 1])
Categories distribution: dict_values([3413, 250, 129, 60, 1345, 555, 1, 5, 1])
The entropy of node 6 is 1.153130
The purity of node 6 is category 3 with possibility 0.592638
```

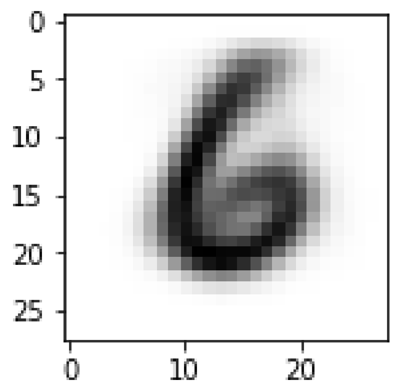
```
This include the following categories: dict_keys([5, 1, 4, 8, 9, 7, 3, 2, 6, 0])
Categories distribution: dict_values([2235, 6724, 1900, 2176, 2112, 2994, 1346, 2174, 593, 182]
The entropy of node 7 is 2.047503
The purity of node 7 is category 1 with possibility 0.299697
This include the following categories: dict_keys([4, 6, 3, 0, 2, 9, 8, 5, 1])
Categories distribution: dict_values([201, 1381, 3, 71, 19, 35, 11, 13, 1])
The entropy of node 8 is 0.774370
The purity of node 8 is category 6 with possibility 0.795965
This include the following categories: dict_keys([4, 9, 7, 2, 3, 5, 8, 0, 6, 1])
Categories distribution: dict_values([3431, 3556, 3210, 66, 80, 290, 162, 18, 7, 1])
The entropy of node 9 is 1.333928
The purity of node 9 is category 9 with possibility 0.328620
The final total entropy is: 1.3950479498307395
The final total purity is: 0.4970833333333333
the final variance is  [[6.84667345]
 [6.50315863]
 [6.25571656]
 [6.41985491]
 [6.84328915]
 [6.54161479]
 [6.42878442]
 [6.0672367 ]
 [6.72630396]
 [6.44480961]]
the final k-mean is  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```



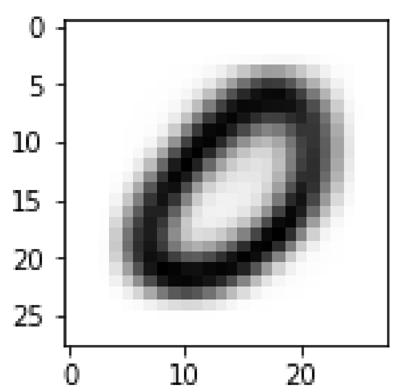
The 0th node



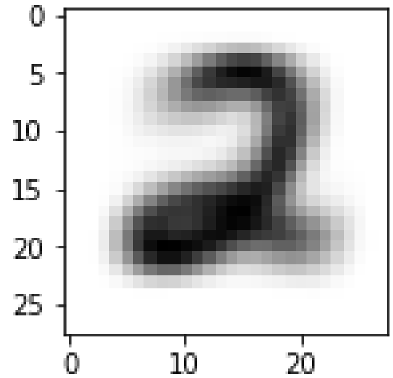
The 1th node



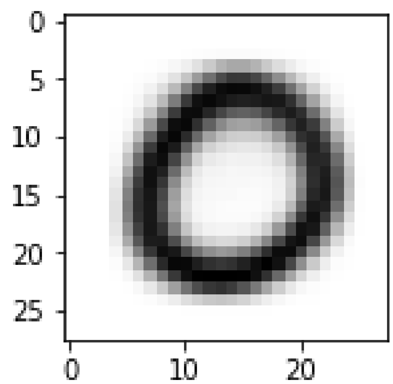
The 2th node



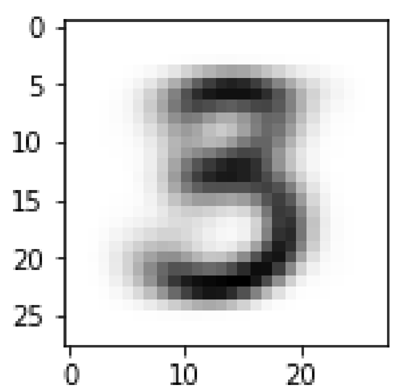
The 3th node



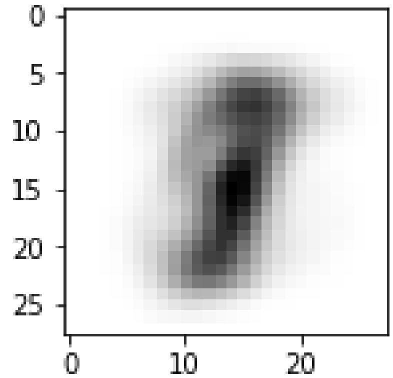
The 4th node



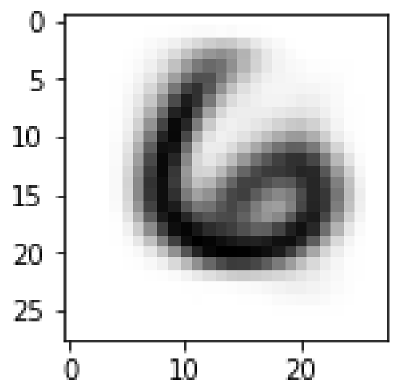
The 5th node



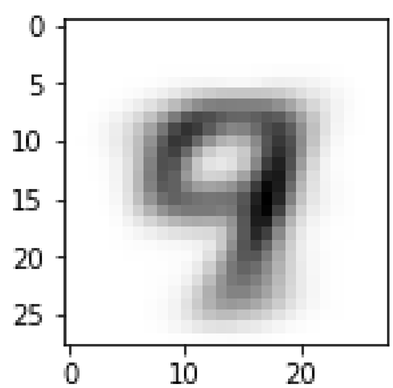
The 6th node



The 7th node



The 8th node



The 9th node

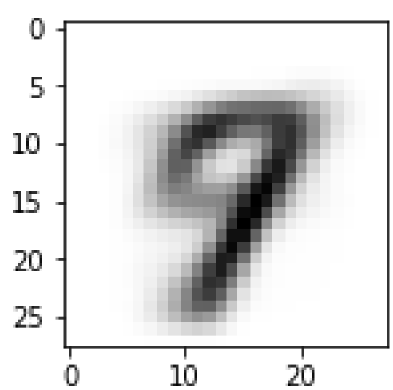
```
In [35]: K_num = input("the cluster number is:")
         SJ = SJKMeans()
         SJ._init_(x_train, y_train, K_num)
         KM=SJ.pick_init_mean()
         agn_num = np.zeros(int(K_num))
         iter_num = input("select iteration steps:")
         threshold = input("write down the expected threshold:")
         method = input("write E for Euclidean Distance, M for Manhattan Distance:")
         for it in range(int(iter_num)):
           count = np.zeros(int(K_num))
           K_metrix = np.zeros([int(K_num),30010,784])
           pnt_dict = {}
           for kk in range(int(K_num)):
             pnt_dict[kk]=[]
           for j in range(np.shape(x_train)[0]):
             kdist, km_ind = SJ.calculate_dist(j, KM, method)
             K_metrix[km_ind,int(count[km_ind]),:] = x_train[j,]
             count[km_ind]+=1
             pnt_dict = SJ.assign_pnt(pnt_dict, km_ind, j)
           #KM, kill, ready_node, new_var = SJ.cal_mean_diff(
           #KM, K_metrix, count, float(threshold))
           KM, new_var = SJ.cal_mean_diff(KM, K_metrix, count, float(threshold))
           '''
           KM, kill, ready_node, new_var = SJ.cal_mean_diff(
           KM, K_metrix, count, float(threshold))
           if (it%20==0):
             print("The point distribution of each node")
             print(count)
             print("Need more iteration")
             print("The following nodes are ready.", ready_node)
           if kill == int(K_num):
             break
           '''
           if (it%20==0):
             print("The point distribution of each node")
             print(count)
             node_ls=count-agn_num
             lj_node = np.where(node_ls==0)[0]
             print("These nodes are stable,",lj_node)
             if (len(lj_node) == int(K_num)):
               print("All nodes are ready.")
               break
             else:
               print("Need more iteration")
           agn_num = count
         ttl_entro = []
         ttl_purty = []
         for i in range(int(K_num)):
```

```
        ttl_entro, ttl_purty = SJ.entropy_purity(pnt_dict[i], i, ttl_entro, ttl_purty)
      Total_Entropy = np.sum(np.multiply(ttl_entro, (1/np.shape(x_train)[0])))
      Total_Purity = np.sum(np.multiply(ttl_purty, (1/np.shape(x_train)[0])))
      print("The final total entropy is:", Total_Entropy)
      print("The final total purity is:", Total_Purity)
      print("the final variance is ", new_var)
      print("the final k-mean is ", KM)
      for i in range(int(K_num)):
        tupian = np.reshape(np.round(KM[i]*255.),(28,28))
        plt.figure(figsize=(10,10))
        plt.subplot(3,4,i+1)
        plt.grid(False)
        plt.imshow(tupian, cmap=plt.cm.binary)
        plt.xlabel("The %dth node " %(i))
        plt.show()
```

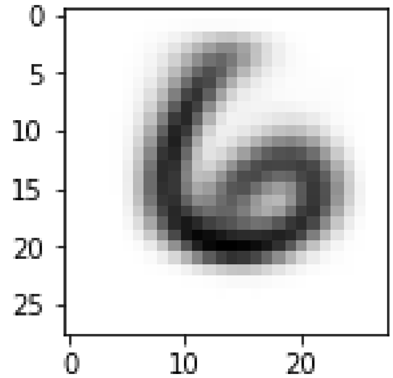
```
the cluster number is:12
select iteration steps:160
write down the expected threshold:0.1
write E for Euclidean Distance, M for Manhattan Distance:M
The point distribution of each node
[ 6218.  2660.  8272.  7825. 13419.  6227.  1211.  3713.  3051.  3029.
  2165.  2210.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[ 8338.  2443.  1767.  8056. 17242.  4703.  2750.  4680.  3720.  2018.
  1948.  2335.]
These nodes are stable, []
Need more iteration
The point distribution of each node
[ 8248.  2048.  1705.  8288. 17316.  4831.  2744.  4423.  3946.  2012.
  1959.  2480.]
These nodes are stable, [ 0 10]
Need more iteration
The point distribution of each node
[ 8225.  2014.  1732.  8292. 17339.  4825.  2742.  4356.  4013.  2032.
  1944.  2486.]
These nodes are stable, [ 5  6  9 10 11]
Need more iteration
The point distribution of each node
[ 8210.  1973.  1743.  8310. 17336.  4843.  2743.  4342.  4036.  2033.
  1945.  2486.]
These nodes are stable, [ 0  1  2  5  6  7  8 11]
Need more iteration
The point distribution of each node
[ 8210.  1973.  1743.  8310. 17336.  4843.  2743.  4342.  4036.  2034.
  1944.  2486.]
```

```
These nodes are stable, [ 0  1  2  3  4  5  6  7  8  9 10 11]
All nodes are ready.
This include the following categories: dict_keys([4, 7, 2, 9, 8, 3, 1, 0, 5, 6])
Categories distribution: dict_values([1824, 3258, 98, 2116, 454, 67, 6, 42, 342, 3])
The entropy of node 0 is 1.470165
The purity of node 0 is category 7 with possibility 0.396833
This include the following categories: dict_keys([6, 3, 4, 0, 5, 2, 8, 9, 1, 7])
Categories distribution: dict_values([1781, 3, 35, 84, 24, 22, 20, 2, 1, 1])
The entropy of node 1 is 0.473190
The purity of node 1 is category 6 with possibility 0.902686
This include the following categories: dict_keys([4, 7, 9, 5, 6, 8, 0, 3, 2])
Categories distribution: dict_values([932, 131, 561, 36, 10, 38, 8, 7, 20])
The entropy of node 2 is 1.185422
The purity of node 2 is category 4 with possibility 0.534710
This include the following categories: dict_keys([9, 4, 7, 1, 3, 2, 5, 8, 0, 6])
Categories distribution: dict_values([2669, 2314, 2105, 8, 251, 94, 514, 295, 49, 11])
The entropy of node 3 is 1.561395
The purity of node 3 is category 9 with possibility 0.321179
This include the following categories: dict_keys([1, 3, 5, 8, 9, 7, 4, 2, 6, 0])
Categories distribution: dict_values([6715, 1958, 1963, 2370, 498, 748, 588, 1685, 655, 156])
The entropy of node 4 is 1.877478
The purity of node 4 is category 1 with possibility 0.387344
This include the following categories: dict_keys([6, 0, 5, 4, 8, 9, 2, 3, 1, 7])
Categories distribution: dict_values([3390, 625, 343, 134, 79, 10, 147, 109, 5, 1])
The entropy of node 5 is 1.080920
The purity of node 5 is category 6 with possibility 0.699979
This include the following categories: dict_keys([0, 2, 5, 9, 3, 6, 8, 4, 7])
Categories distribution: dict_values([2472, 51, 93, 11, 59, 33, 17, 3, 4])
The entropy of node 6 is 0.488956
The purity of node 6 is category 0 with possibility 0.901203
This include the following categories: dict_keys([3, 9, 5, 0, 8, 2, 6, 7])
Categories distribution: dict_values([2496, 39, 1216, 42, 512, 30, 6, 1])
The entropy of node 7 is 1.059386
The purity of node 7 is category 3 with possibility 0.574850
This include the following categories: dict_keys([5, 3, 8, 0, 2, 9, 1, 6])
Categories distribution: dict_values([851, 1029, 1996, 40, 89, 27, 1, 3])
The entropy of node 8 is 1.195619
The purity of node 8 is category 8 with possibility 0.494549
This include the following categories: dict_keys([2, 3, 5, 7, 1, 8, 0, 6])
Categories distribution: dict_values([1928, 68, 4, 4, 6, 19, 2, 3])
The entropy of node 9 is 0.266115
The purity of node 9 is category 2 with possibility 0.947886
This include the following categories: dict_keys([2, 5, 0, 4, 3, 8, 9, 7, 6])
Categories distribution: dict_values([1789, 2, 25, 11, 78, 26, 4, 6, 3])
The entropy of node 10 is 0.396105
The purity of node 10 is category 2 with possibility 0.920267
This include the following categories: dict_keys([0, 5, 6, 3, 8, 9, 7, 2, 4])
Categories distribution: dict_values([2378, 33, 20, 6, 25, 12, 6, 5, 1])
```

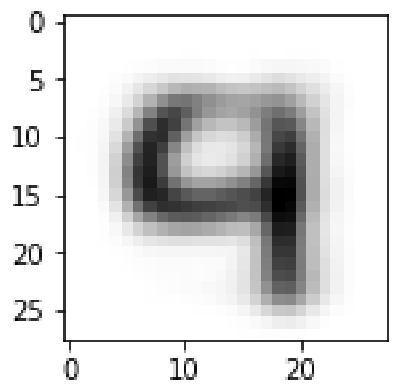
```
The entropy of node 11 is 0.255379
The purity of node 11 is category 0 with possibility 0.956557
The final total entropy is: 1.3090108881244826
The final total purity is: 0.5300666666666667
the final variance is  [[6.02618212]
 [6.49494757]
 [6.61311954]
 [6.13118276]
 [5.88463165]
 [6.16642802]
 [6.36014801]
 [6.60459742]
 [6.51407407]
 [6.66391477]
 [6.63682428]
 [6.53229741]]
the final k-mean is  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```



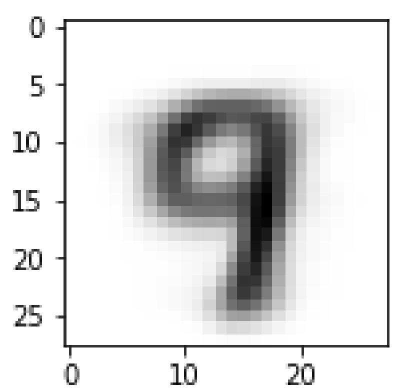
The 0th node



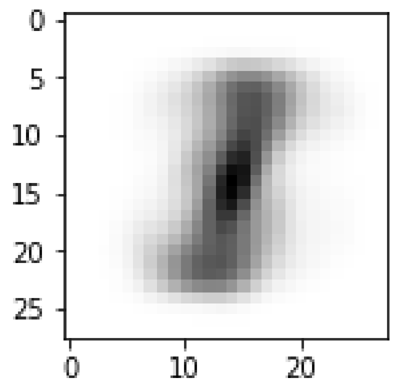
The 1th node



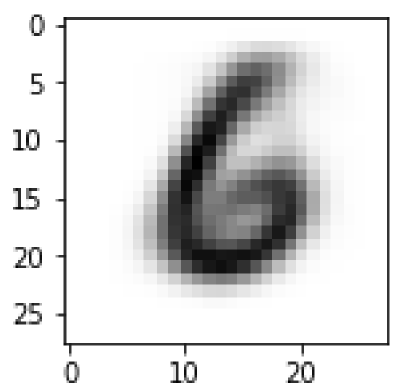
The 2th node



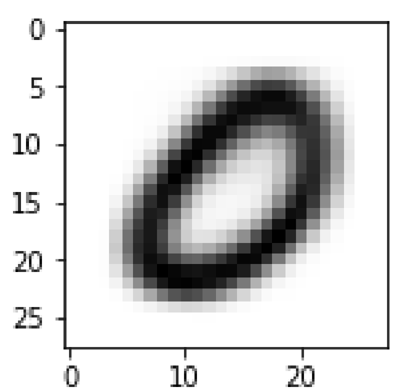
The 3th node



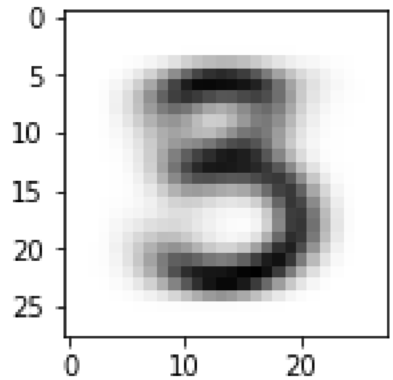
The 4th node



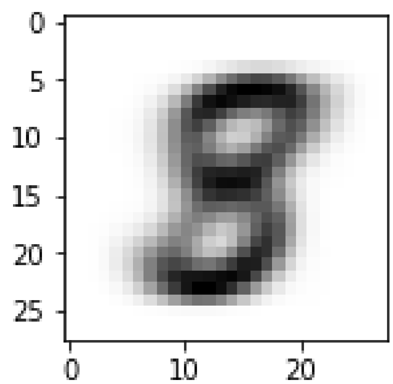
The 5th node



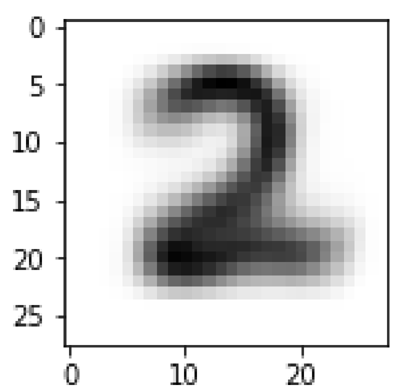
The 6th node



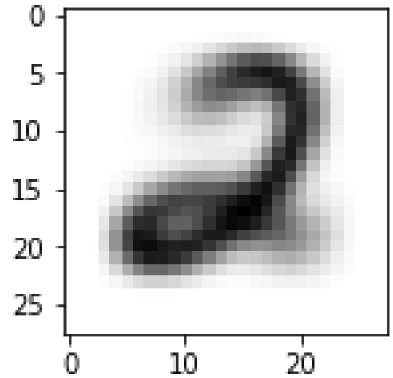
The 7th node



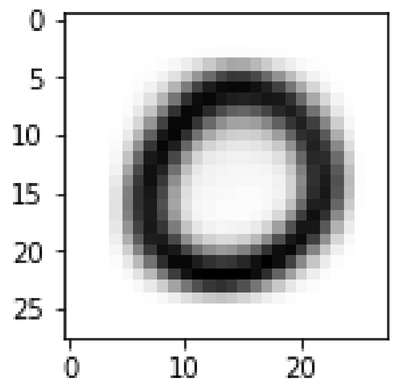
The 8th node



The 9th node



The 10th node



The 11th node