Tutorial of layer visualization

September 19, 2020

Tutorial of layer visualization for multi-label case

This notebook is provided to show the usage of UMAP for multiple label case. Please see another notebook 'Tutorial of layer visualization' as the first step.

NOTE: The script used 3rd module UMAP for layer plotting.

1. Preparation

In this notebook, few packages are necessary.

Assuming that we are in 'root-of-autoBioSeqpy/notebook', then we need to add the search path and import the modules.

Please install UMAP, matplotlib, tensorFlow and keras before using this notebook.

[1]:
```
import os, sys
sys.path.append('../tool/libs')
sys.path.append('../')
oriPath = os.path.curdir
import paraParser
import moduleRead
import dataProcess
#import analysisPlot
import numpy as np
#from sklearn.metrics import␣
↪accuracy_score,f1_score,roc_auc_score,recall_score,precision_score,confusion_matrix,matthew
↪
import tensorflow as tf
from utils import TextDecorate, evalStrList
from keras.models import Model
from keras.models import Sequential

import umap
import umap.plot
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
```

```
Using TensorFlow backend.
```

Using UMAP is a simple way for model visualization, but still we need the model as well as the data for generating the predicted data for plotting. Thus, here we need few steps to get a built

model and the related data.

1.1 Model building

Like the binary case, using running.py could build a model:

python running.py –dataType protein –dataEncodingType onehot –dataTrainFilePaths examples/Gramnegativebacterialsecretedproteins/data/T1SS_train.txt examples/Gramnegativebacterialsecretedproteins/data/T2SS_train.txt examples/Gramnegativebacterialsecretedproteins/data/T3SS_train.txt examples/Gramnegativebacterialsecretedproteins/data/T4SS_train.txt examples/Gramnegativebacterialsecretedproteins/data/T5SS_train.txt examples/Gramnegativebacterialsecretedproteins/data/T7SS_train.txt –dataTrainLabel 0 1 2 3 4 5 –dataTestFilePaths examples/Gramnegativebacterialsecretedproteins/data/T1SS_test.txt examples/Gramnegativebacterialsecretedproteins/data/T2SS_test.txt examples/Gramnegativebacterialsecretedproteins/data/T3SS_test.txt examples/Gramnegativebacterialsecretedproteins/data/T4SS_test.txt examples/Gramnegativebacterialsecretedproteins/data/T5SS_test.txt examples/Gramnegativebacterialsecretedproteins/data/T7SS_test.txt –dataTestLabel 0 1 2 3 4 5 –modelLoadFile examples/Gramnegativebacterialsecretedproteins/model/CNN.py –verbose 1 –outSaveFolderPath tmpOut –savePrediction 1 –saveFig 1 –batch_size 10 –epochs 30 –spcLen 2000 –shuffleDataTrain 1 –modelSaveName tmpMod.json –weightSaveName tmpWeight.bin –noGPU 0 –paraSaveName parameters.txt –labelToMat 1

The outputs will be saved at ../tmpOut, including the parameters, the path of the data and constructure of the model.

If users would like to use their own model, please don't forget to save the model and the weight by using parameters "–modelSaveName" and "–weightSaveName".

1.2 Parameter pasering and data loading

The parameters are saved in ../tmoOut/parameters.txt, we can get the information easily by using the paraParser module.

[2]:
```
paraFile = '../tmpOut/parameters.txt'
paraDict = paraParser.parseParametersFromFile(paraFile)
#print
for k in paraDict:
    print('%r: %r' %(k,paraDict[k]))
```

```
'dataType': ['protein']
'dataEncodingType': ['onehot']
'spcLen': [2000]
'firstKernelSize': []
'dataTrainFilePaths':
['examples/Gramnegativebacterialsecretedproteins/data/T1SS_train.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T2SS_train.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T3SS_train.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T4SS_train.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T5SS_train.txt',
```

```
'examples/Gramnegativebacterialsecretedproteins/data/T7SS_train.txt']
'dataTrainLabel': [0, 1, 2, 3, 4, 5]
'dataTestFilePaths':
['examples/Gramnegativebacterialsecretedproteins/data/T1SS_test.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T2SS_test.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T3SS_test.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T4SS_test.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T5SS_test.txt',
'examples/Gramnegativebacterialsecretedproteins/data/T7SS_test.txt']
'dataTestLabel': [0, 1, 2, 3, 4, 5]
'dataTrainModelInd': []
'dataTestModelInd': []
'outSaveFolderPath': 'tmpOut'
'showFig': True
'saveFig': True
'figDPI': 300
'savePrediction': True
'dataSplitScale': None
'modelLoadFile': ['examples/Gramnegativebacterialsecretedproteins/model/CNN.py']
'weightLoadFile': []
'shuffleDataTrain': True
'shuffleDataTest': False
'batch_size': 10
'epochs': 30
'useKMer': []
'KMerNum': []
'inputLength': []
'loss': 'binary_crossentropy'
'optimizer': 'optimizers.Adam()'
'metrics': ['acc', 'acc']
'modelSaveName': 'tmpMod.json'
'weightSaveName': 'tmpWeight.bin'
'noGPU': False
'paraFile': None
'paraSaveName': 'parameters.txt'
'seed': 1
'labelToMat': True
'colorText': 'Auto'
'verbose': True
'reshapeSize': []
```

Then using predicting module could help us get the model and the datasets.

[3]:
```
import predicting
```

Here we need some modification to specify the dataset due to the layer plotting needs the training dataset.

[4]:
```
paraDict['dataTestFilePaths'] = paraDict['dataTrainFilePaths']
paraDict['dataTestModelInd'] = paraDict['dataTrainModelInd']
paraDict['dataTestLabel'] = paraDict['dataTrainLabel']
```

[5]:
```
os.chdir('../')#to the root path of autoBioSeqpy
```

[6]:
```
predictedLabel,predicted_Probability,testNameLists,testDataMats,testLabelArr,model
↪= predicting.predict(paraDict)
```

```
Enconding protein data for model 0 …
Checking the number of test files, which should be larger than 1 (e.g. at least
two labels)…
Begin to generate test dataset…
Test datasets generated.
Since labelToMat is set, the labels would be changed to onehot-like matrix
begin to prepare model…
Checking module file for modeling
Loading module and weight file
Building model…
Module loaded, generating the summary of the module
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 2000, 20)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1996, 250)         25250
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 250)               0
_________________________________________________________________
dense_1 (Dense)              (None, 650)               163150
_________________________________________________________________
dropout_1 (Dropout)          (None, 650)               0
_________________________________________________________________
activation_1 (Activation)    (None, 650)               0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 3906
_________________________________________________________________
activation_2 (Activation)    (None, 6)                 0
=================================================================
Total params: 192,306
Trainable params: 192,306
Non-trainable params: 0
_________________________________________________________________
outpath tmpOut is exists, the outputs might be overwirten
```

Please note that the test label will NOT used for predicting, thus we have to generate it manually.

[7]:
```
dataLoaders = []
for i,dataPath in enumerate(paraDict['dataTestFilePaths']):
    featureGenerator =  dataProcess.ProteinFeatureGenerator('protein',␣
 ↪useKMer=False)
    dataLoader = dataProcess.DataLoader(label = paraDict['dataTestLabel'][i],␣
 ↪featureGenerator=featureGenerator)
    dataLoader.readFile(dataPath, spcLen = paraDict['spcLen'][0])
    dataLoaders.append(dataLoader)
testLabelArrs = []
testDataSetCreator = dataProcess.DataSetCreator(dataLoaders)
testDataMat, testLabelArr, nameList = testDataSetCreator.
 ↪getDataSet(toShuffle=False, withNameList=True)
oriLabel = testLabelArr
```

Few printing operations for display the current information.

[8]:
```
testLabelArr.shape
```

[8]:
```
(667,)
```

[9]:
```
print('%d matrix generated:\n' %(len(testDataMats)))
for i in range(len(testDataMats)):
    print('Matrix and label %d:' %i)
    print(testDataMats[i])
    print(testLabelArr)
    print('\n')
```

```
1 matrix generated:

Matrix and label 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]
 [1 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 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 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 0 0 0 0 0
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

```
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5]
```

Here we simply plot the confusion matrix, please note that here the confusion matrix is from training data.

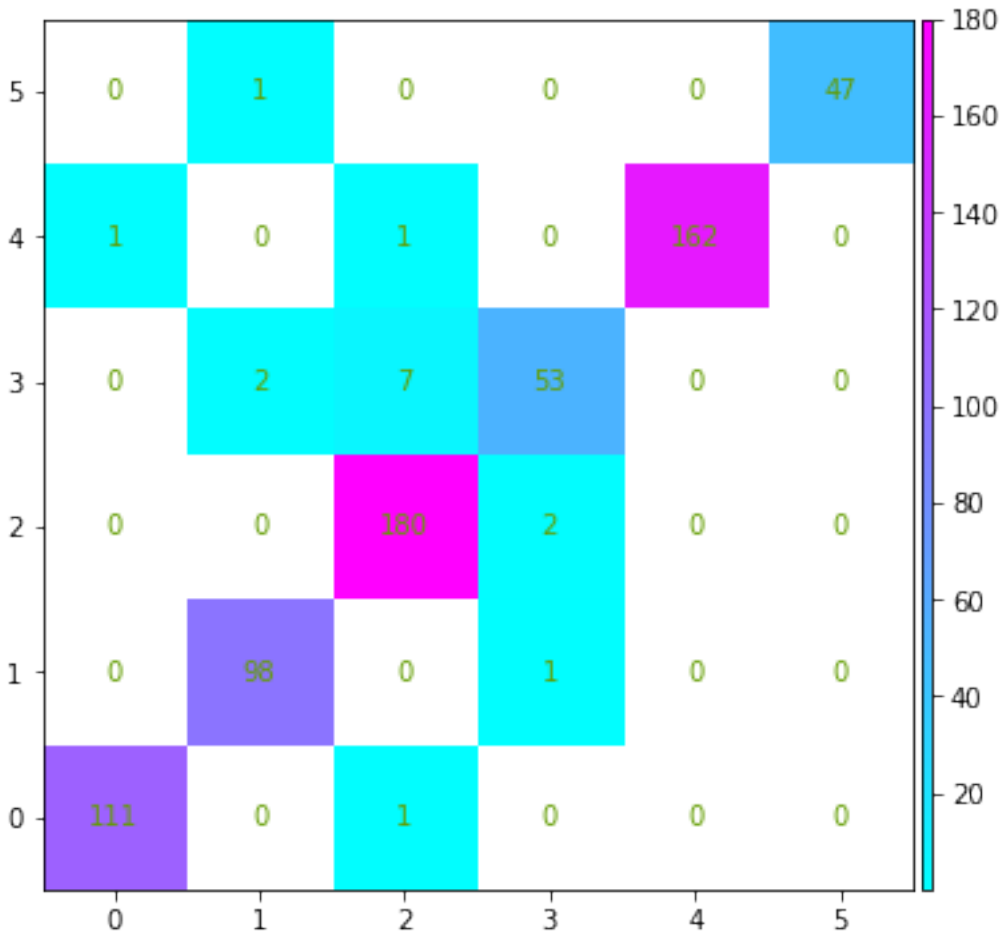
[10]:
```
predictedLabel = model.predict_classes(testDataMat)
```

[11]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
import analysisPlot
```

[12]:
```
cm=confusion_matrix(testLabelArr,predictedLabel)
print(cm)
print('ACC: %.2f%%' %(accuracy_score(oriLabel,predictedLabel)*100))
```

```
[[111   0   1   0   0   0]
 [  0  98   0   1   0   0]
 [  0   0 180   2   0   0]
 [  0   2   7  53   0   0]
 [  1   0   1   0 162   0]
 [  0   1   0   0   0  47]]
ACC: 97.60%
```

[13]:
```
classes = np.arange(6).astype(str)
analysisPlot.
 ↪showMatWithVal(cm,precision='%d',xtickLabels=classes,ytickLabels=classes,toInvert=False,cma
```



2. Using UMAP for layer visualization

In this section, we will compare the output before and after the global_max_pooling layer, both outputs will be generated and plotted with different parameters.

2.1 Plotting for the output after the pooling layer

Having the module, let us see what is in the sequential. A function is used to make this process more simple.

[14]:
```
def unBoundLayers(modelIn,layers = []):
    for layer in modelIn.layers:
        if not 'sequential' in layer.name.lower():
            layers.append(layer)
        else:
            unBoundLayers(layer,layers)
    return layers
```

[15]:
```
layers = unBoundLayers(model)
for i, layer in enumerate(layers):
    print('layer %d: %s' %(i,layer.name))
```

```
layer 0: reshape_1
layer 1: conv1d_1
layer 2: global_max_pooling1d_1
layer 3: dense_1
layer 4: dropout_1
layer 5: activation_1
layer 6: dense_2
layer 7: activation_2
```

Since the data UMAP need is not the finall output from the full connection layer, usually we only need to use few of the layers. For example, if we only use the layers until the 3rd last layers, the code becomes:

[16]:
```
newModel_after_pool = Sequential()
for layer in layers[:4]:
    newModel_after_pool.add(layer)
newModel_after_pool.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 2000, 20)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1996, 250)         25250
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 250)               0
_________________________________________________________________
dense_1 (Dense)              (None, 650)               163150
=================================================================
Total params: 188,400
Trainable params: 188,400
Non-trainable params: 0
_________________________________________________________________
```

Now the model above is available for UMAP.

The shape of intermediate data is 2-d, thus could be used for plotting directly.

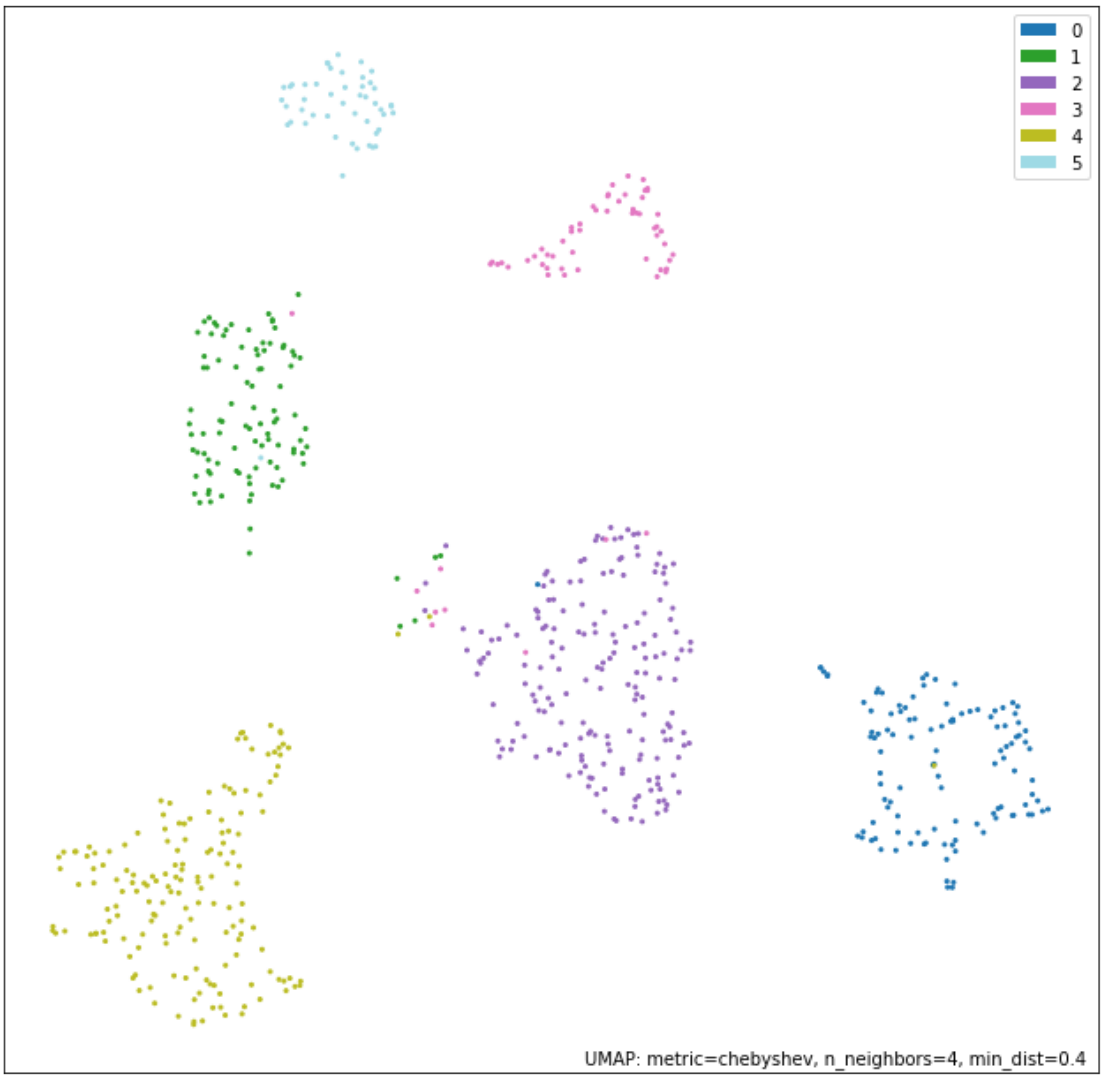
[17]:
```
predicted_Probability_after_pool = newModel_after_pool.predict(testDataMats)
```

[18]:
```
predicted_Probability_after_pool.shape
```

[18]:
```
(667, 650)
```

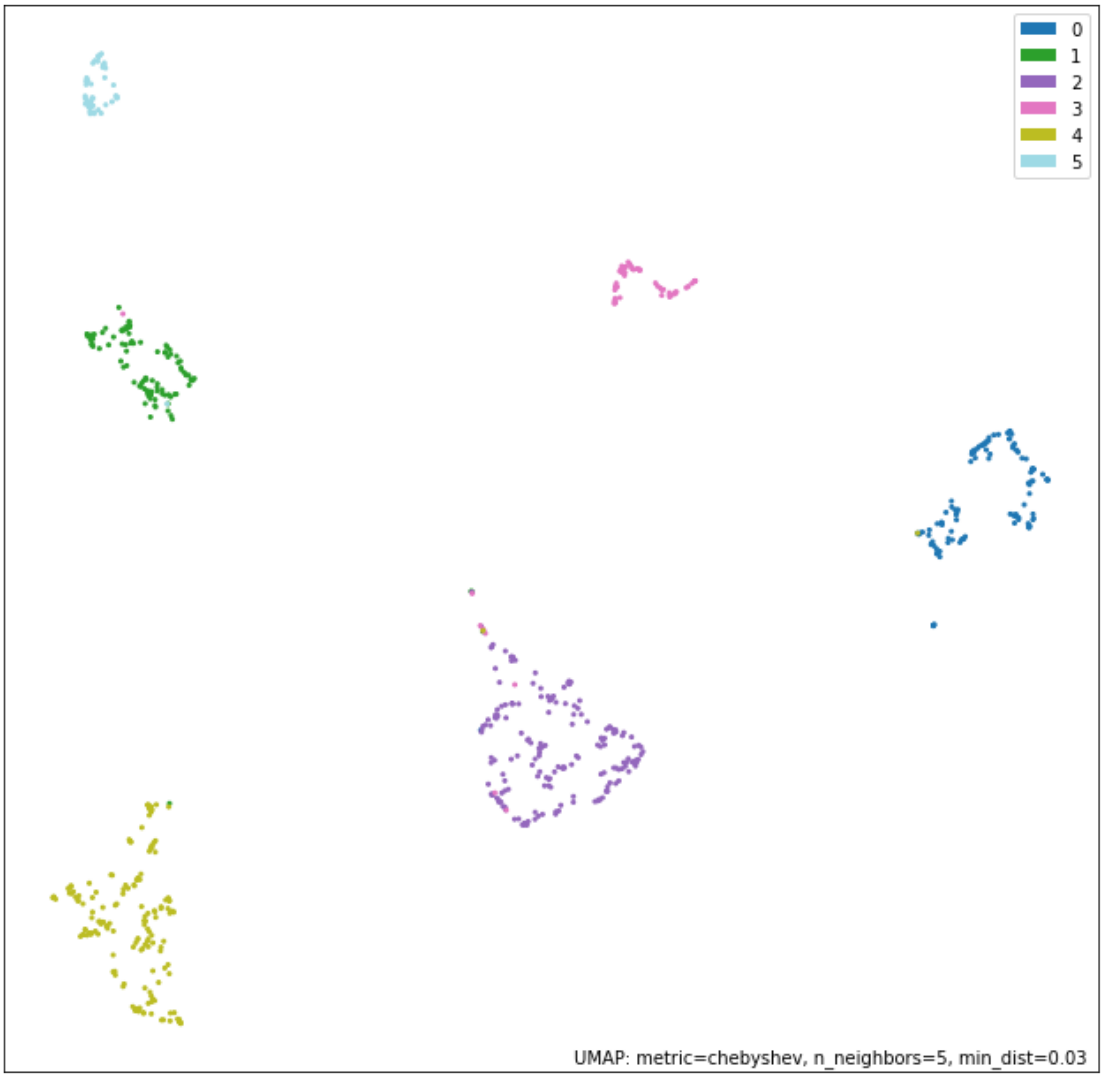
Using the parameter for simple plotting.

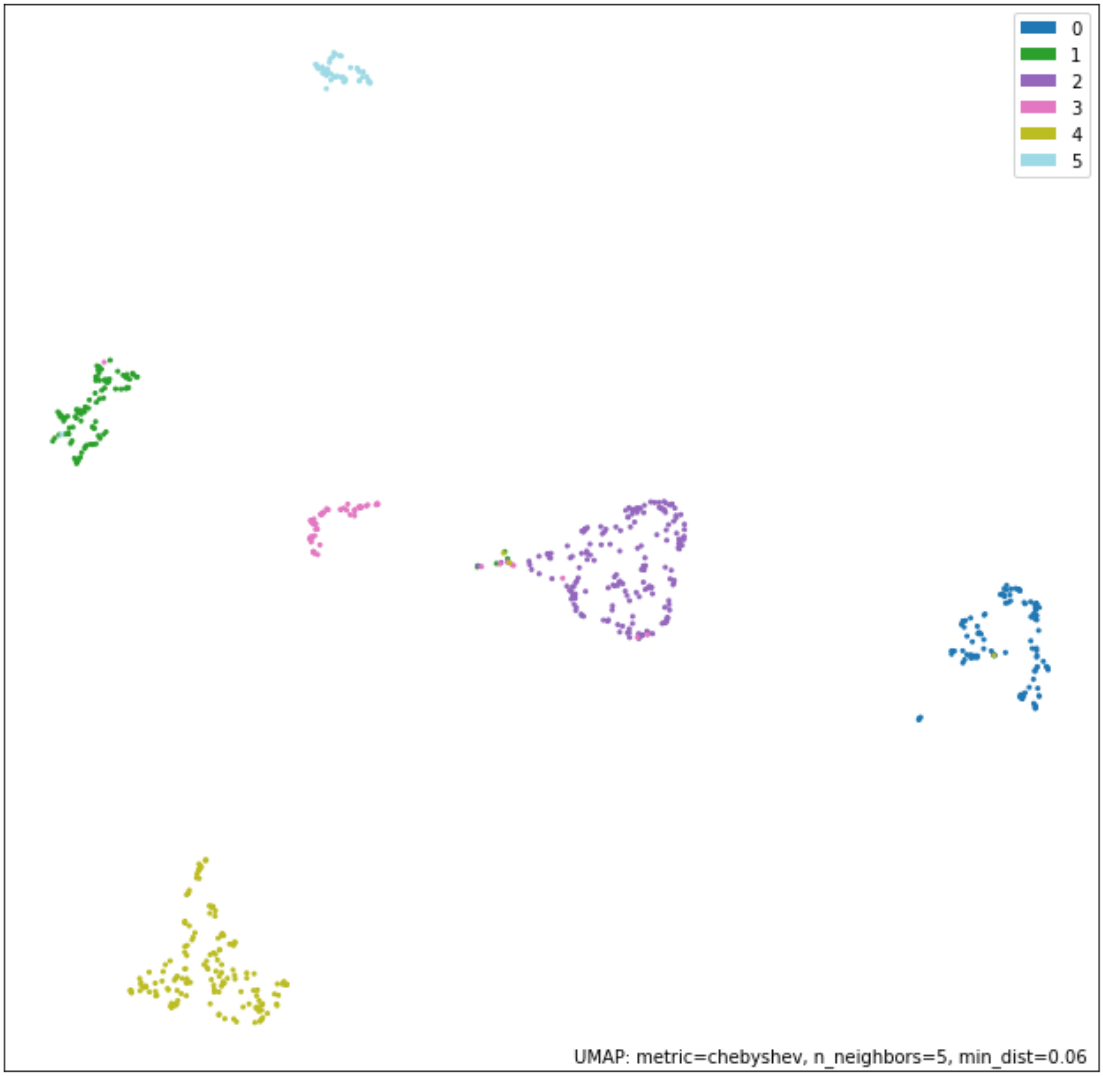
[19]:
```
# the featureDict could contain more parameters, use '?umap.UMAP' for more␣
 ↪details.
featureDict={
        'n_neighbors' : 4,
        'min_dist' : 0.4,
        'metric' : 'chebyshev',
        }

mapper = umap.UMAP(**featureDict).fit(predicted_Probability_after_pool)
plotObj = umap.plot.points(mapper, labels=testLabelArr, theme='blue',␣
 ↪background='white')
plt.show()
```

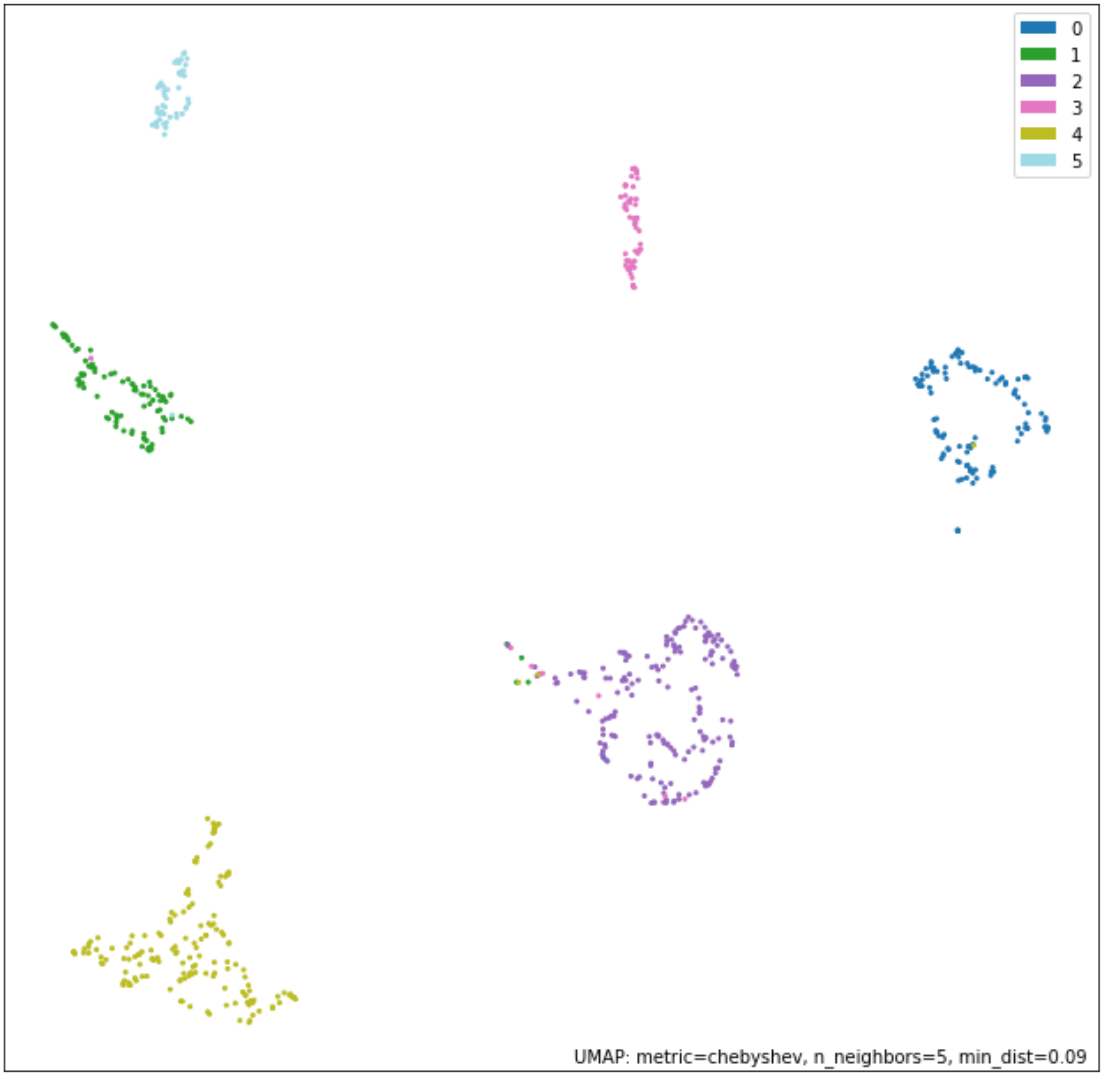


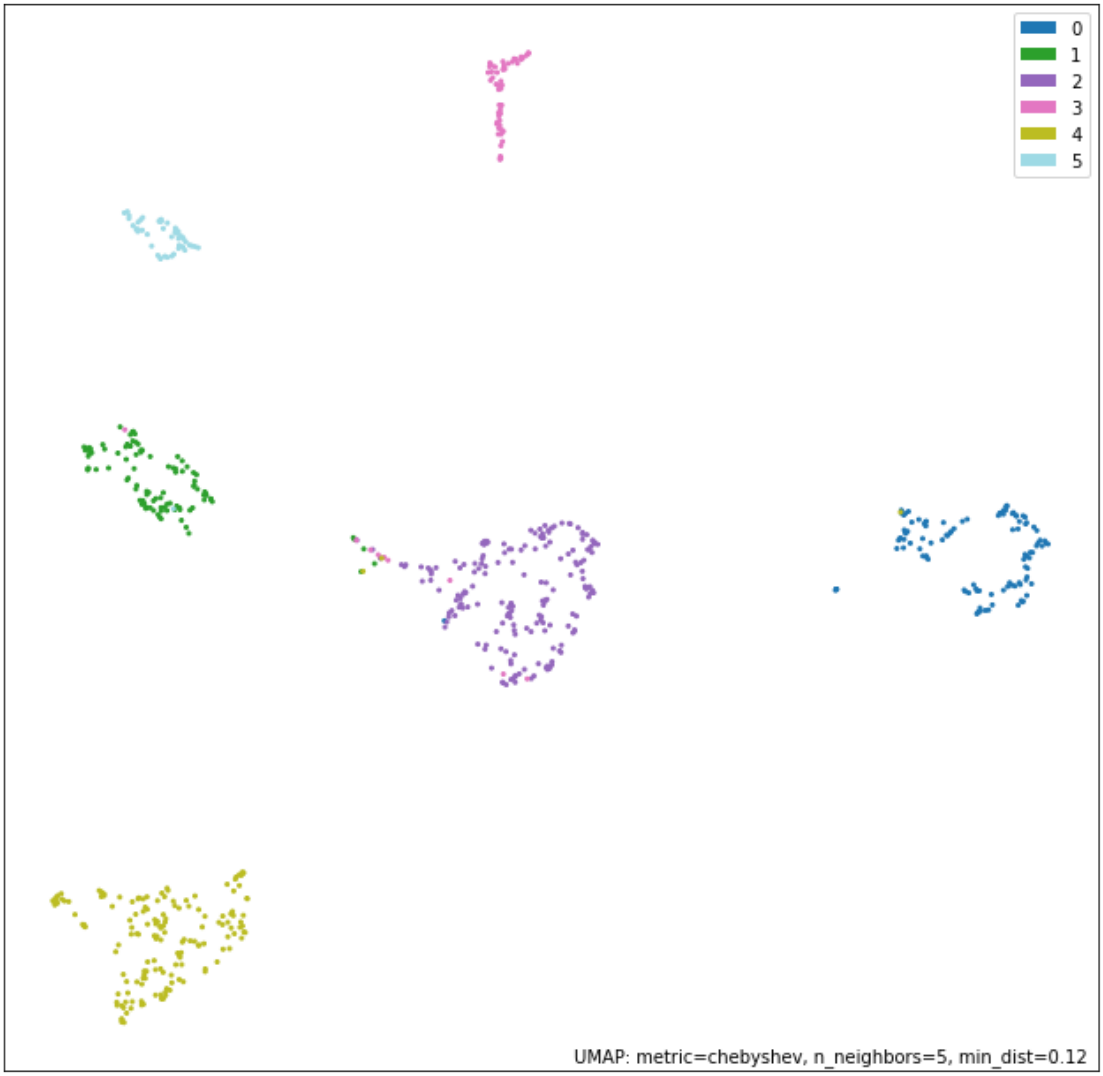
Using a loop for multiple plotting with different parameters.

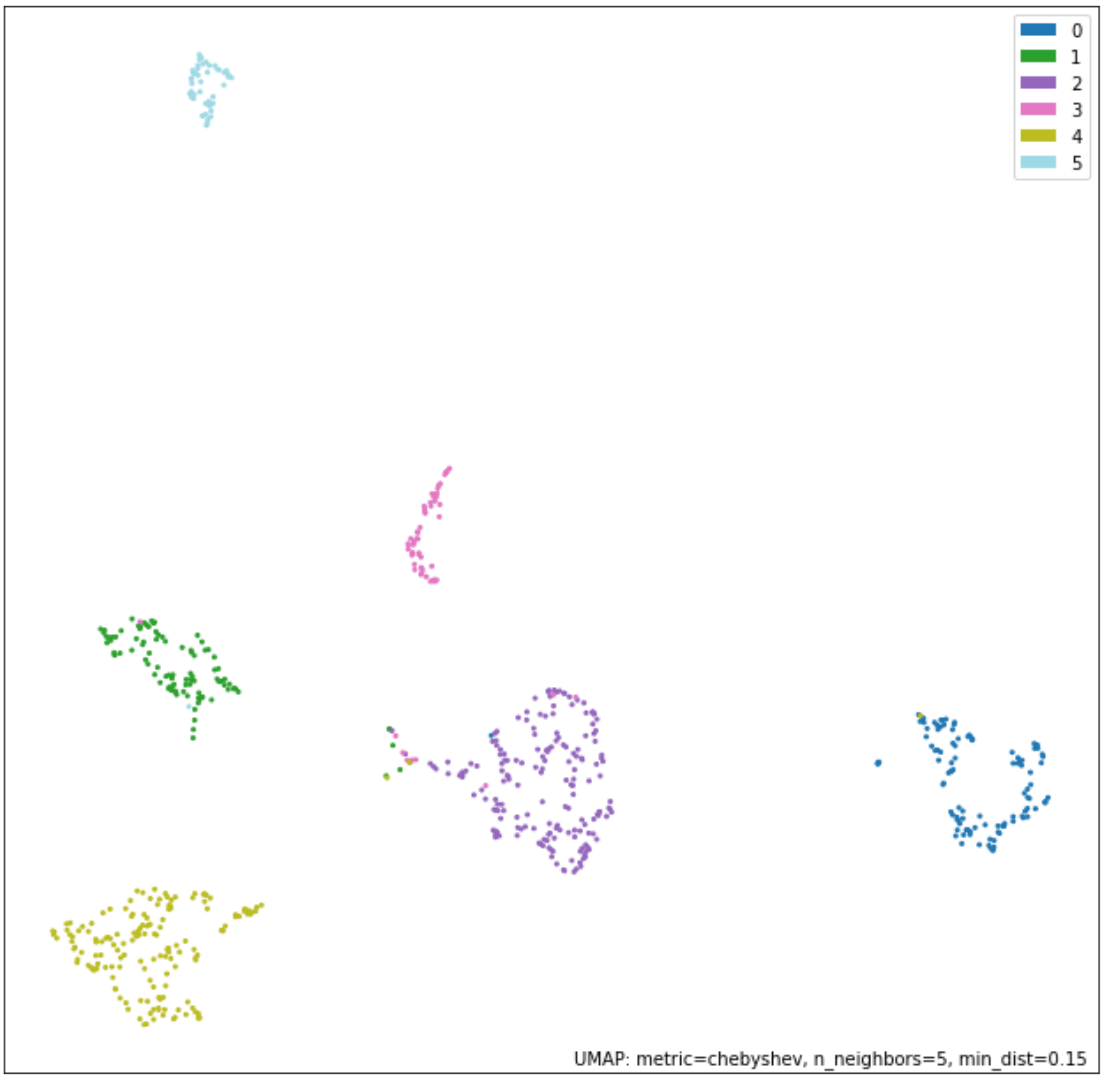
[20]:
```
featureDict={
        'n_neighbors' : 4,
        'min_dist' : 0.4,
        'metric' : 'chebyshev',
        }
for n_neighbors in np.arange(5,30,5).astype(int):
    featureDict['n_neighbors'] = n_neighbors
    for min_dist in np.arange(0.03,0.2,0.03):
        featureDict['min_dist'] = min_dist
        mapper = umap.UMAP(**featureDict).fit(predicted_Probability_after_pool)
        plotObj = umap.plot.points(mapper, labels=testLabelArr, theme='blue')
        plt.show()
        plt.close() #close the plotted figure
```

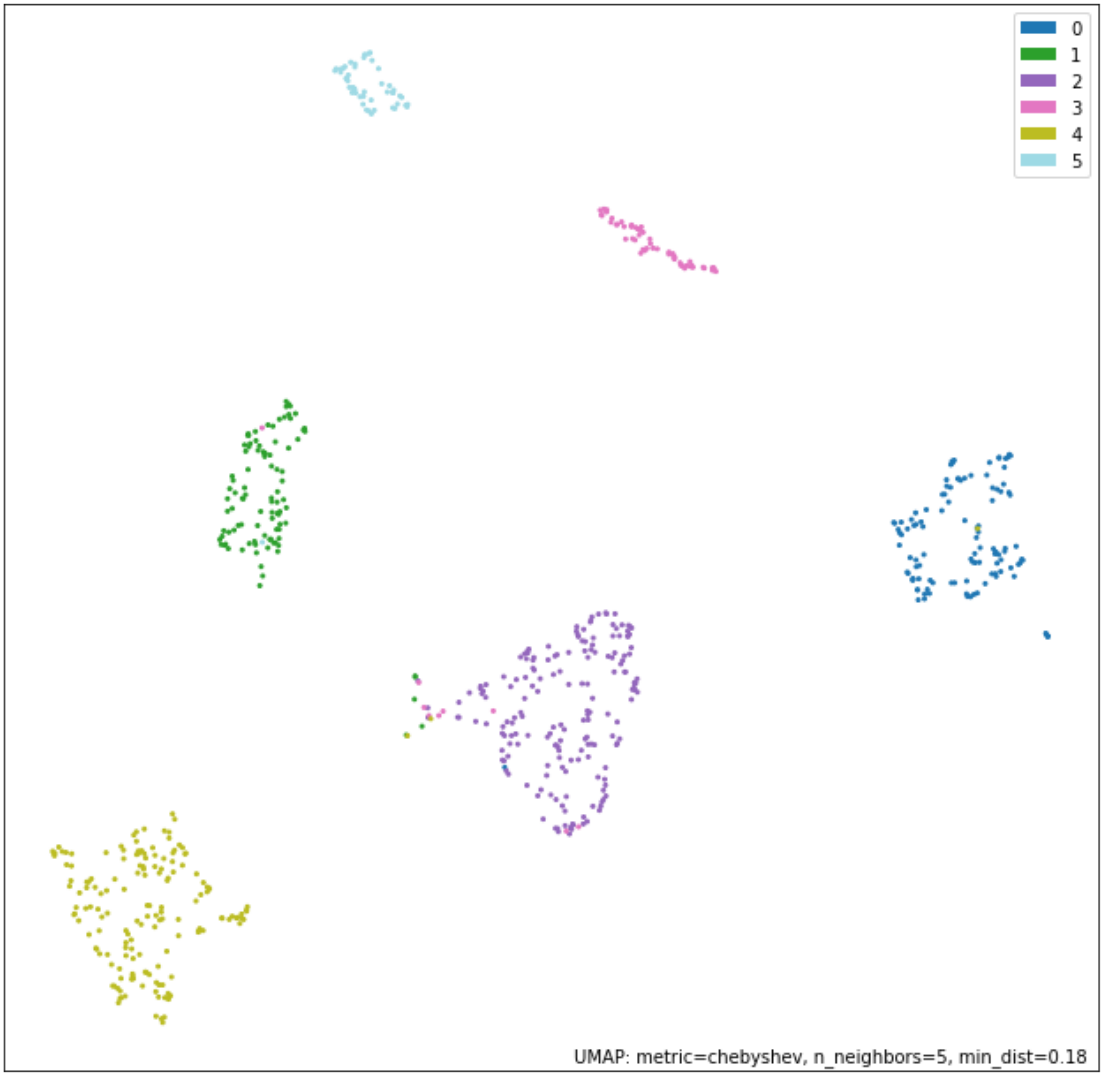


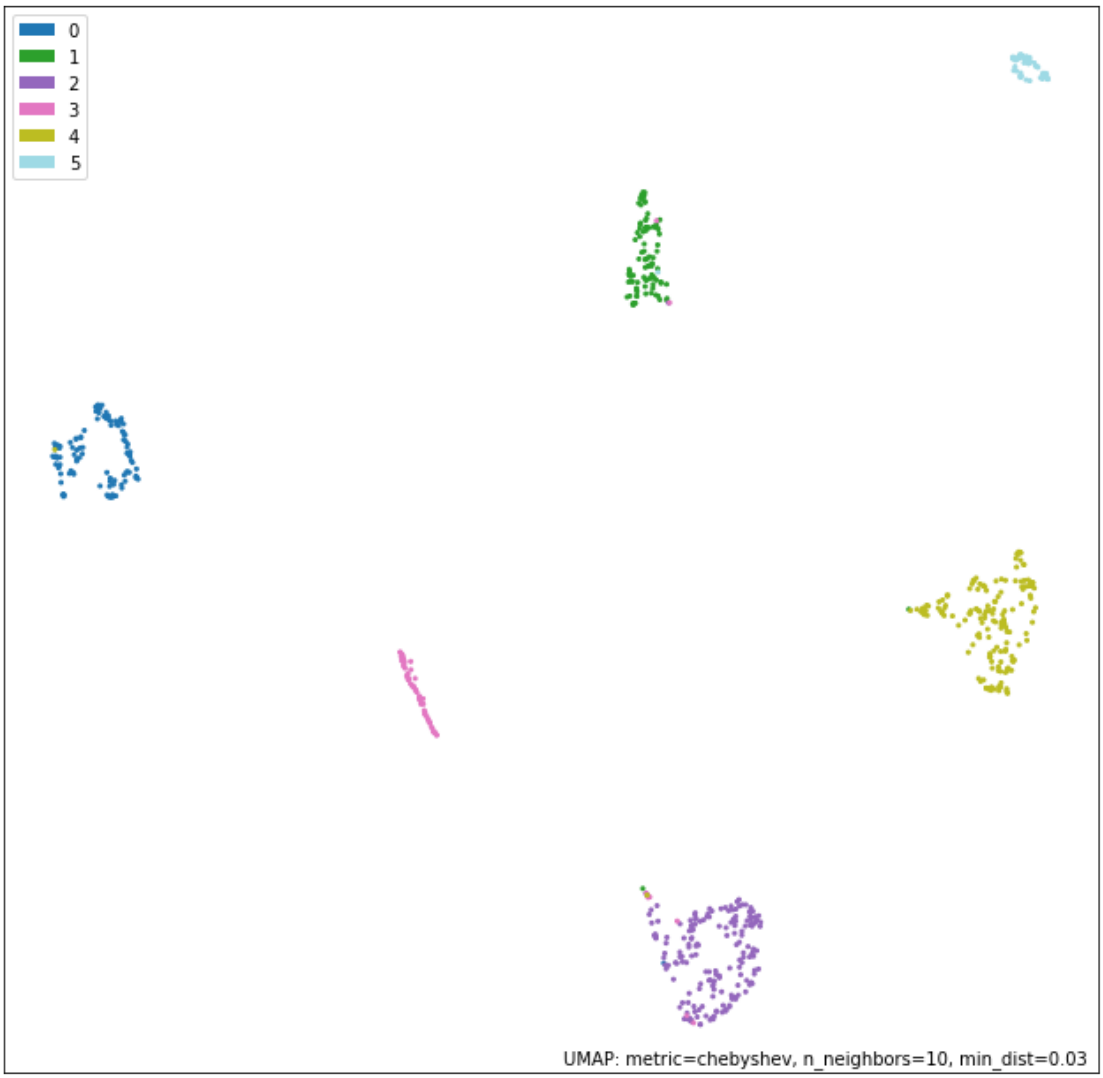

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=5, min_dist=0.06


0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=5, min_dist=0.09

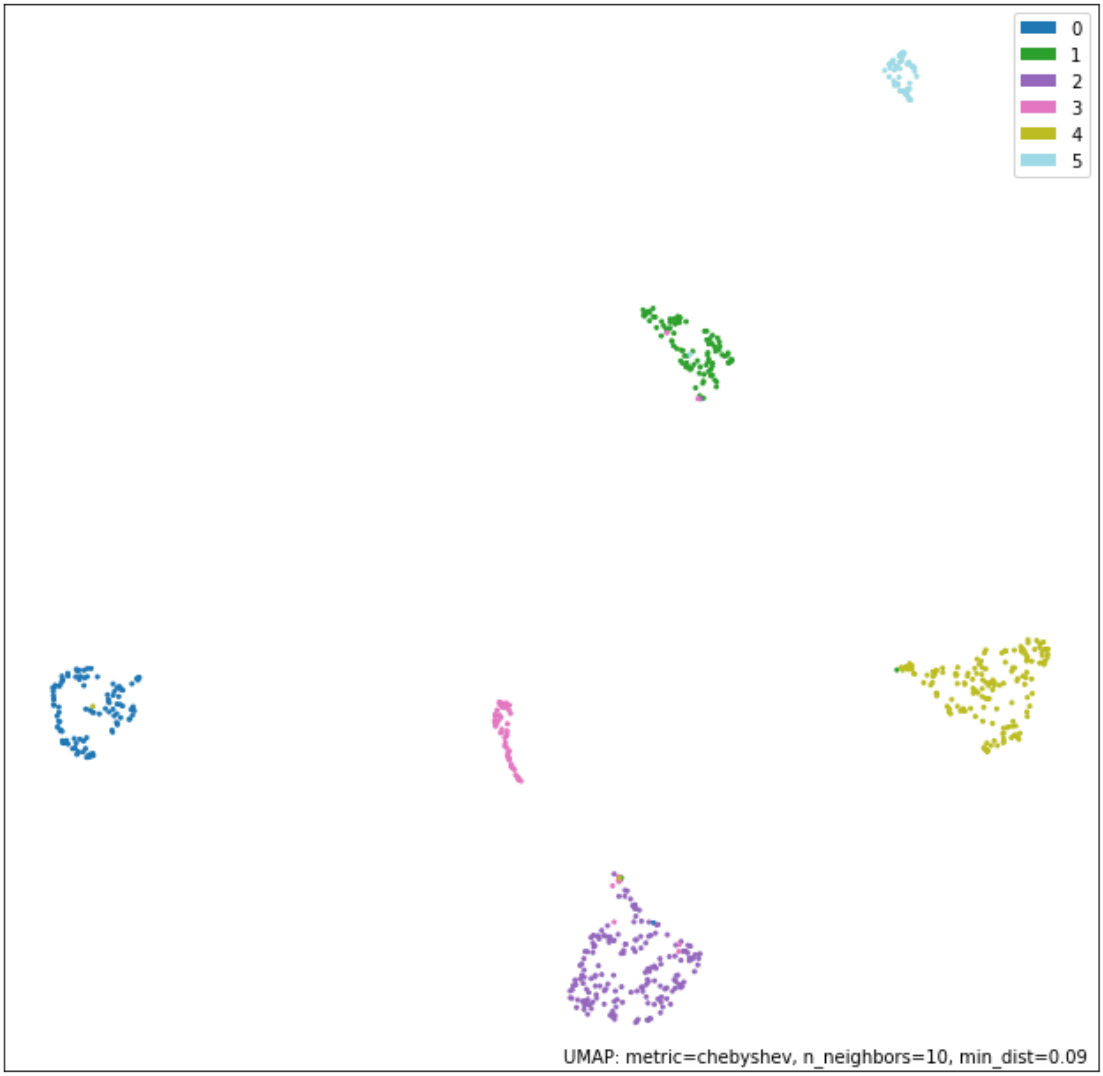

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=5, min_dist=0.12

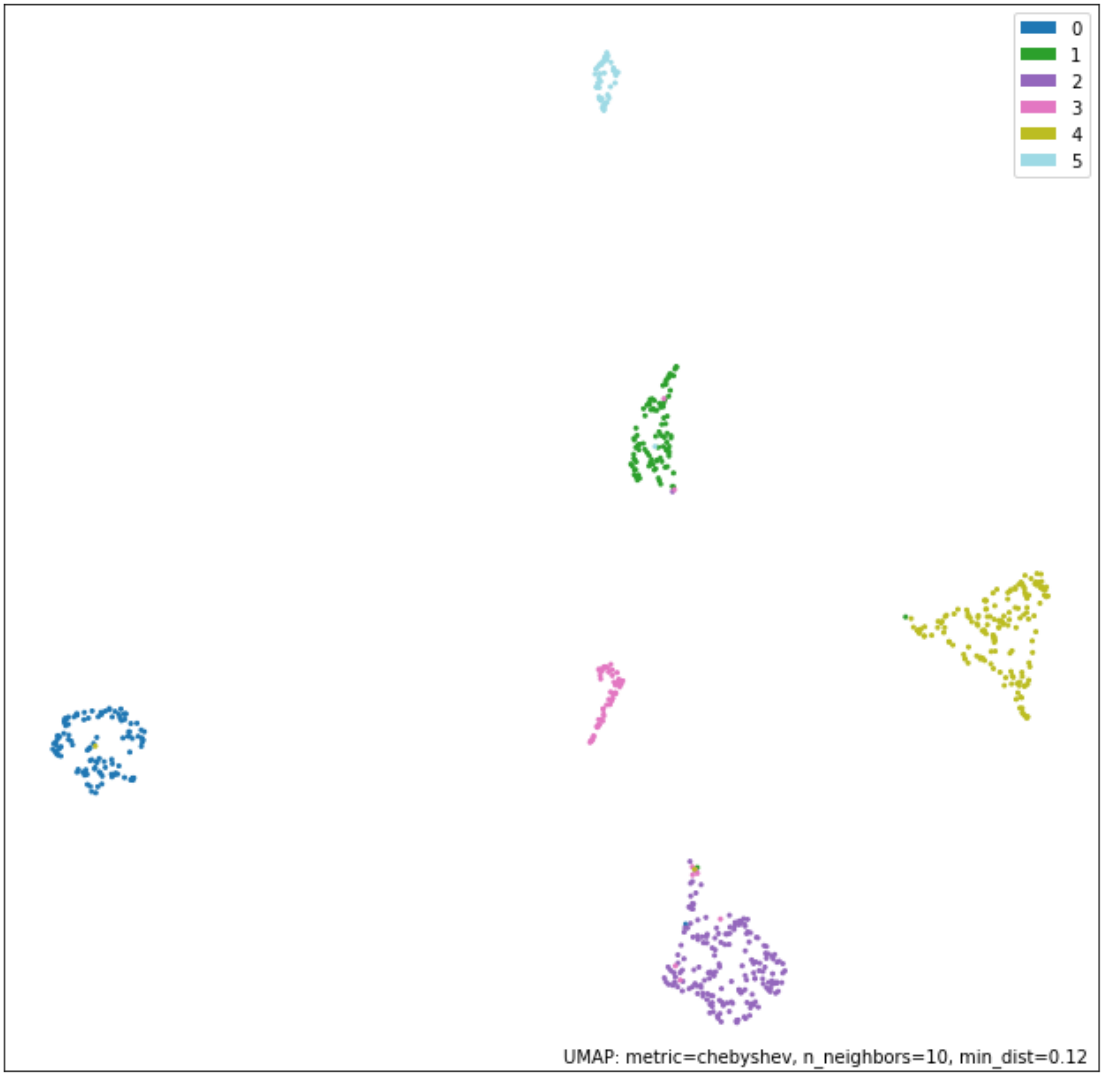

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=5, min_dist=0.15

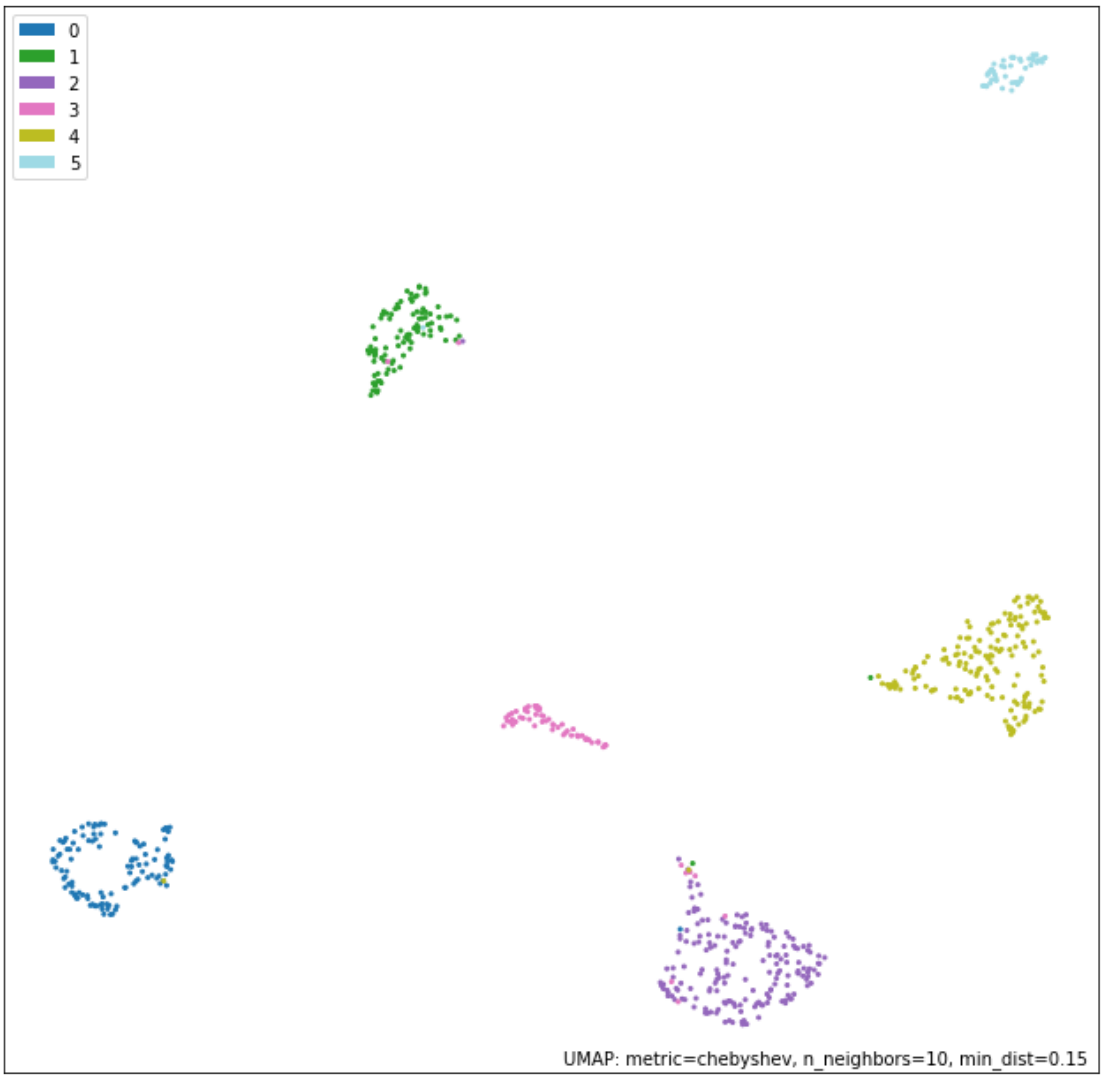

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=5, min_dist=0.18

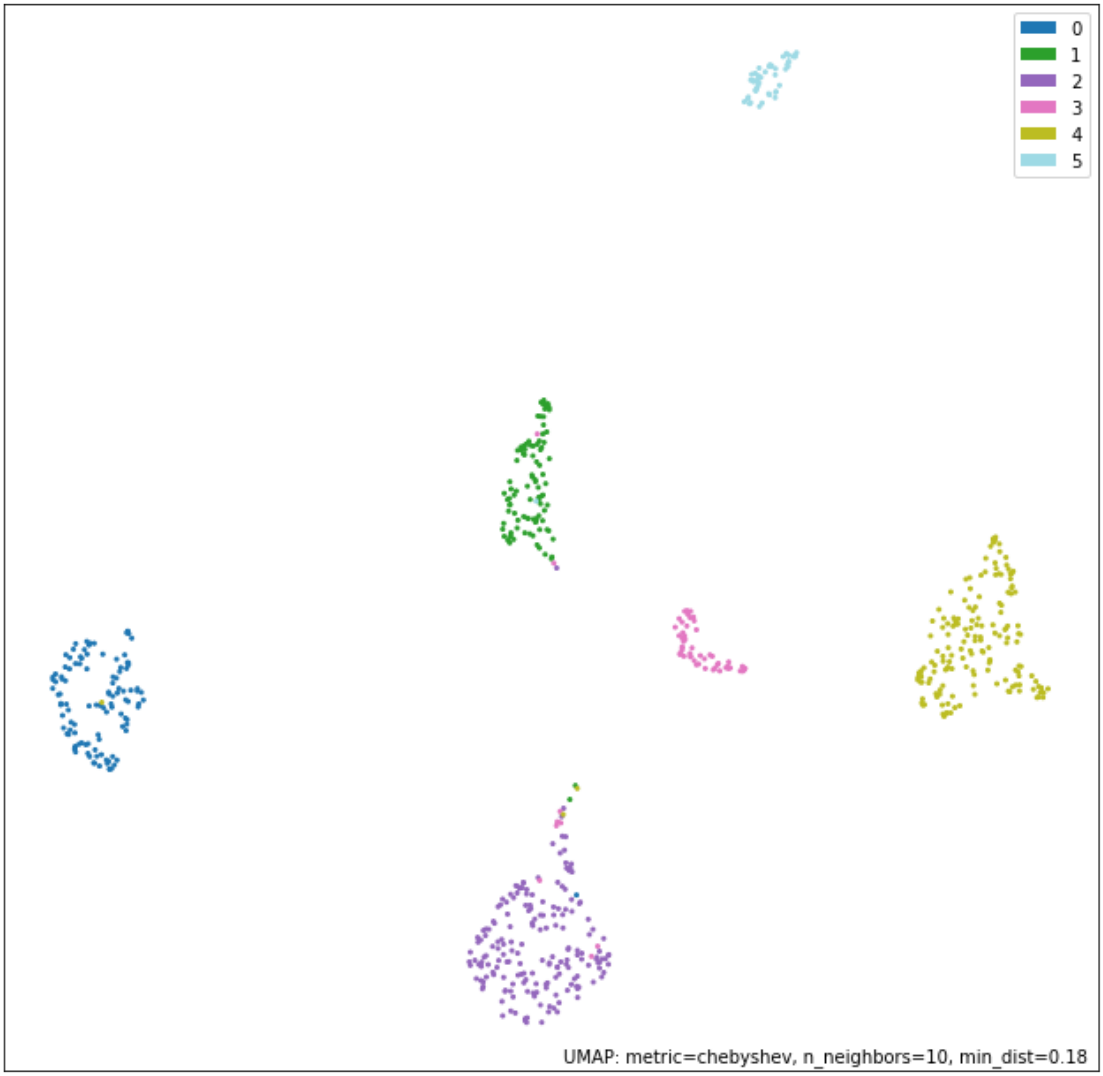

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=10, min_dist=0.03

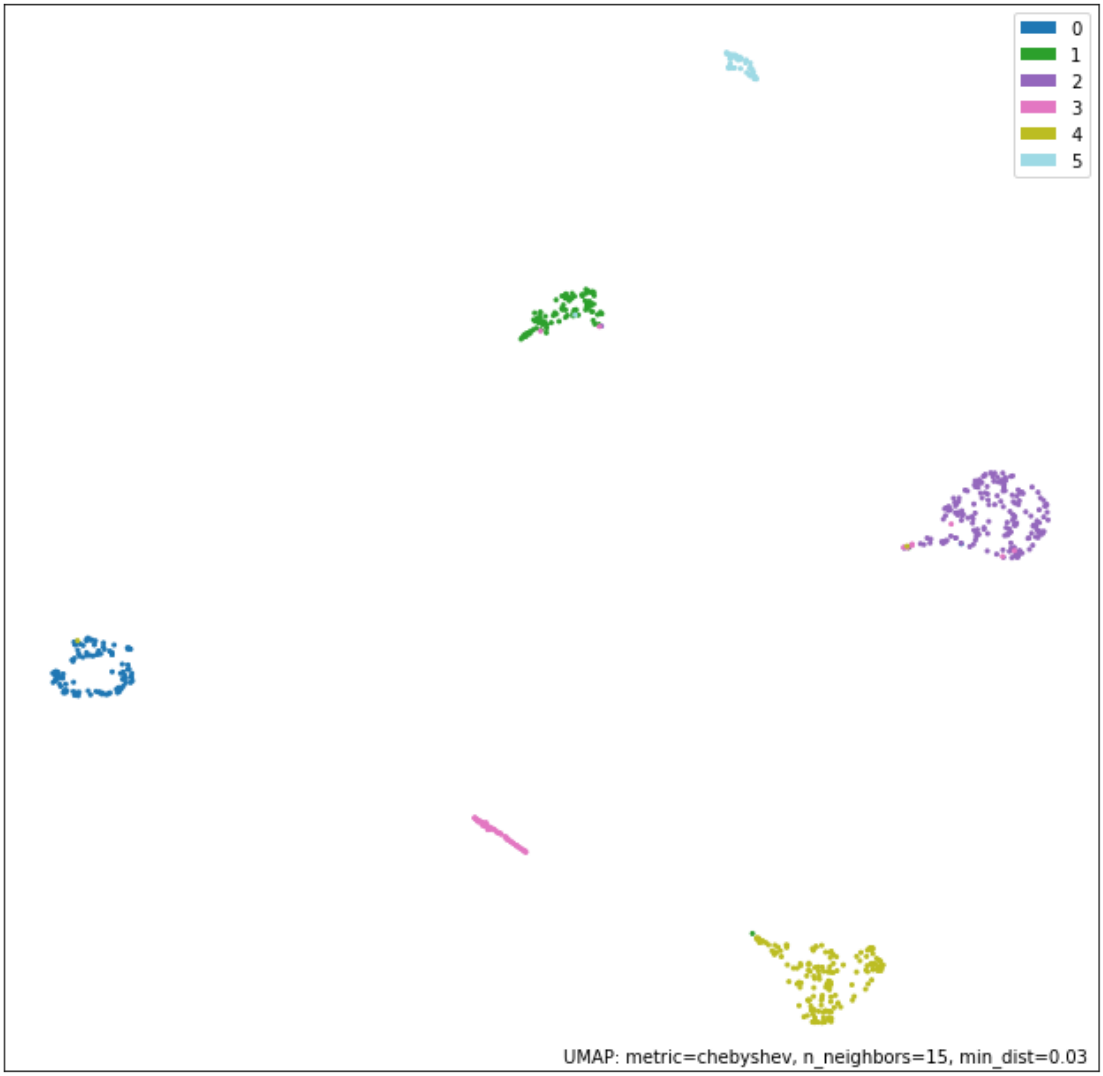

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=10, min_dist=0.06


0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=10, min_dist=0.09

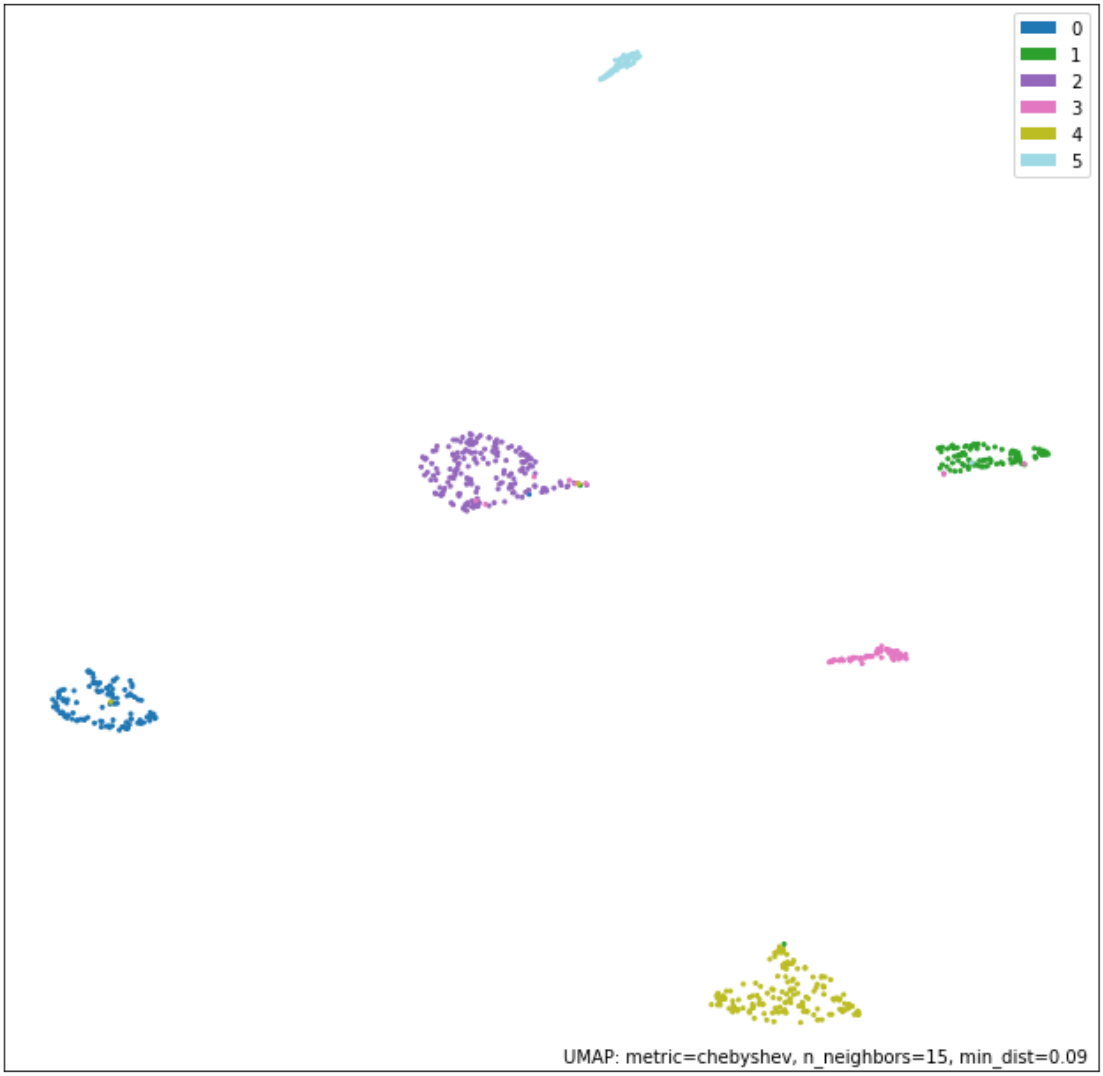

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=10, min_dist=0.12

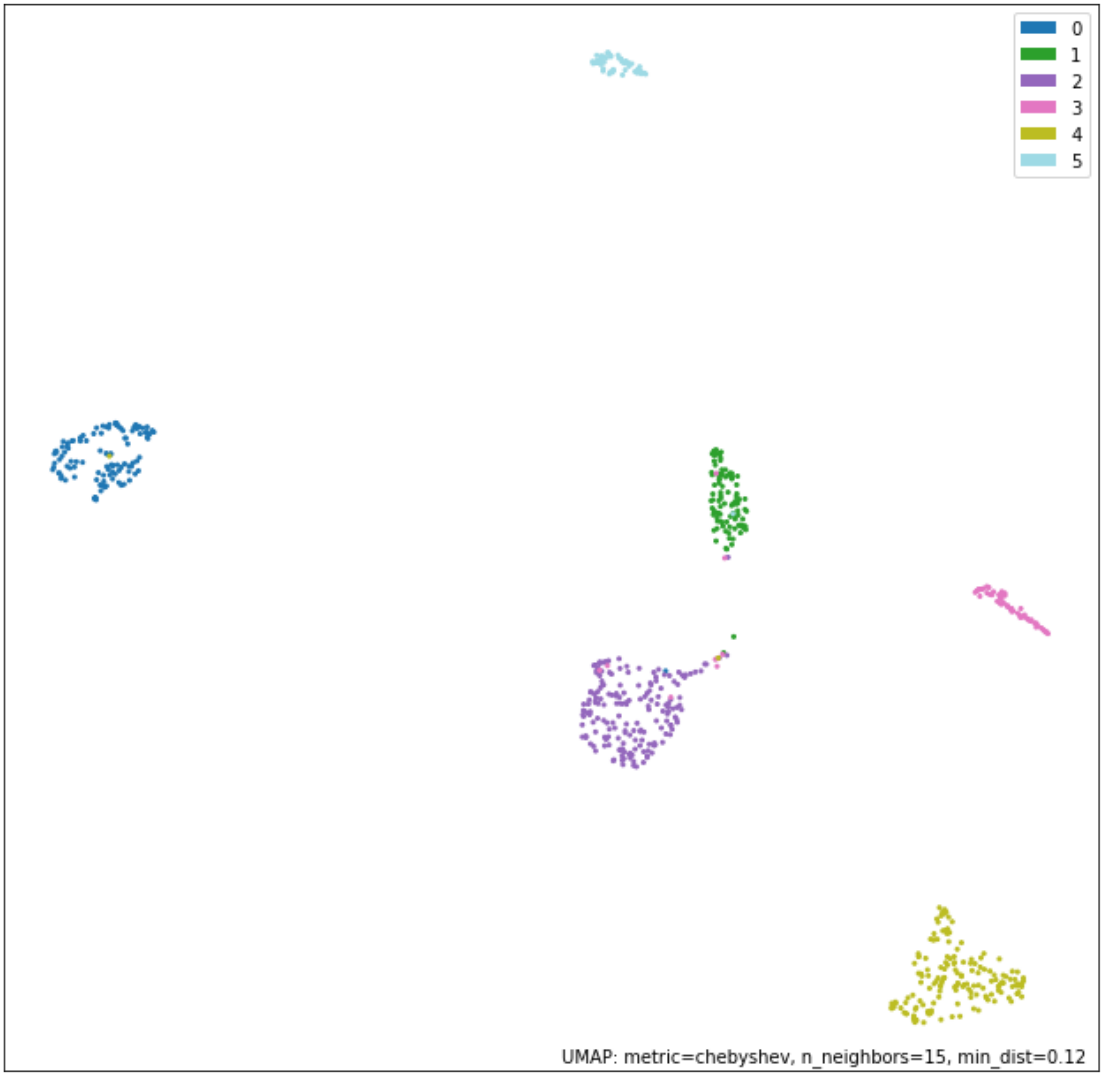

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=10, min_dist=0.15

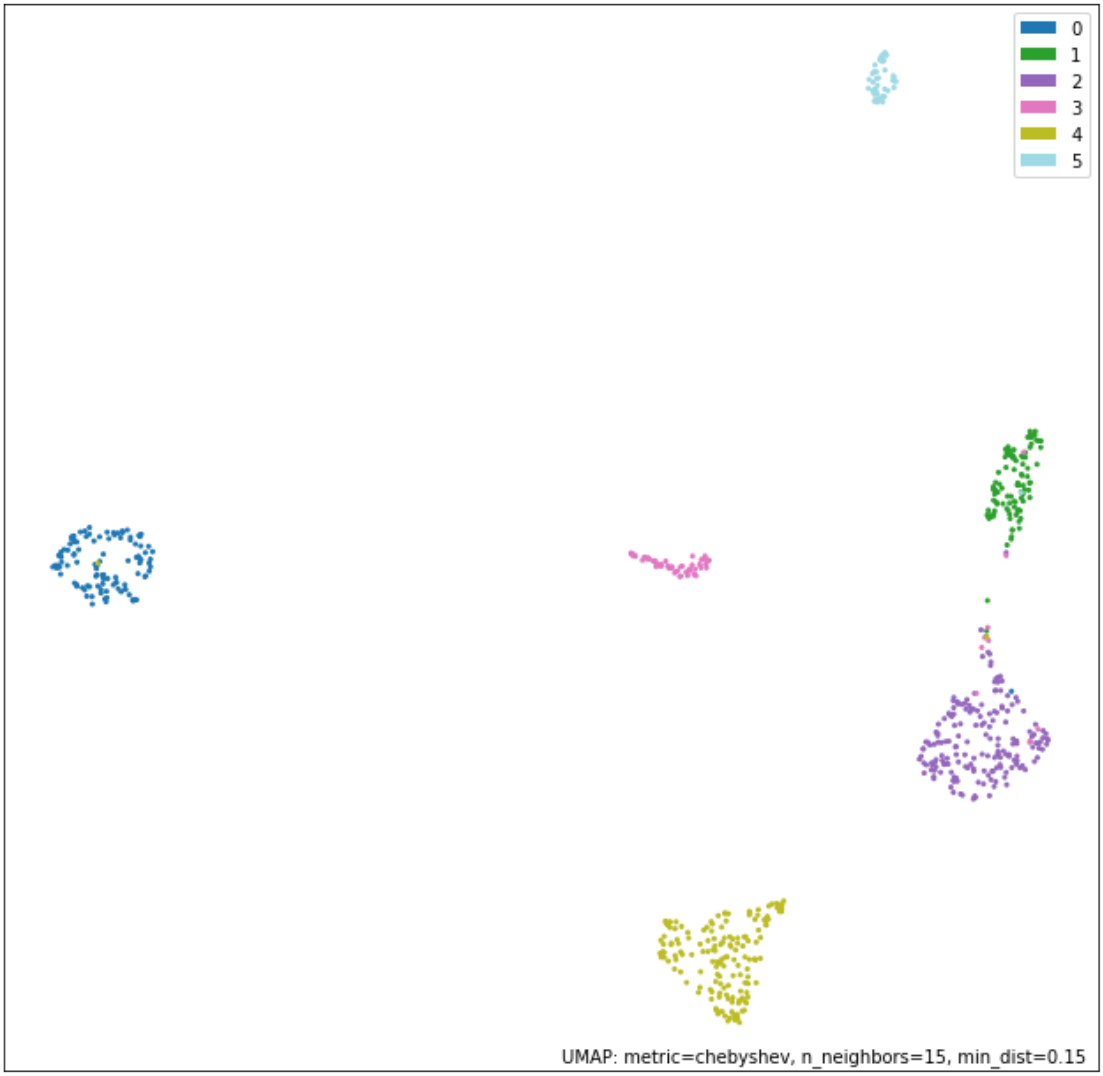

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=10, min_dist=0.18

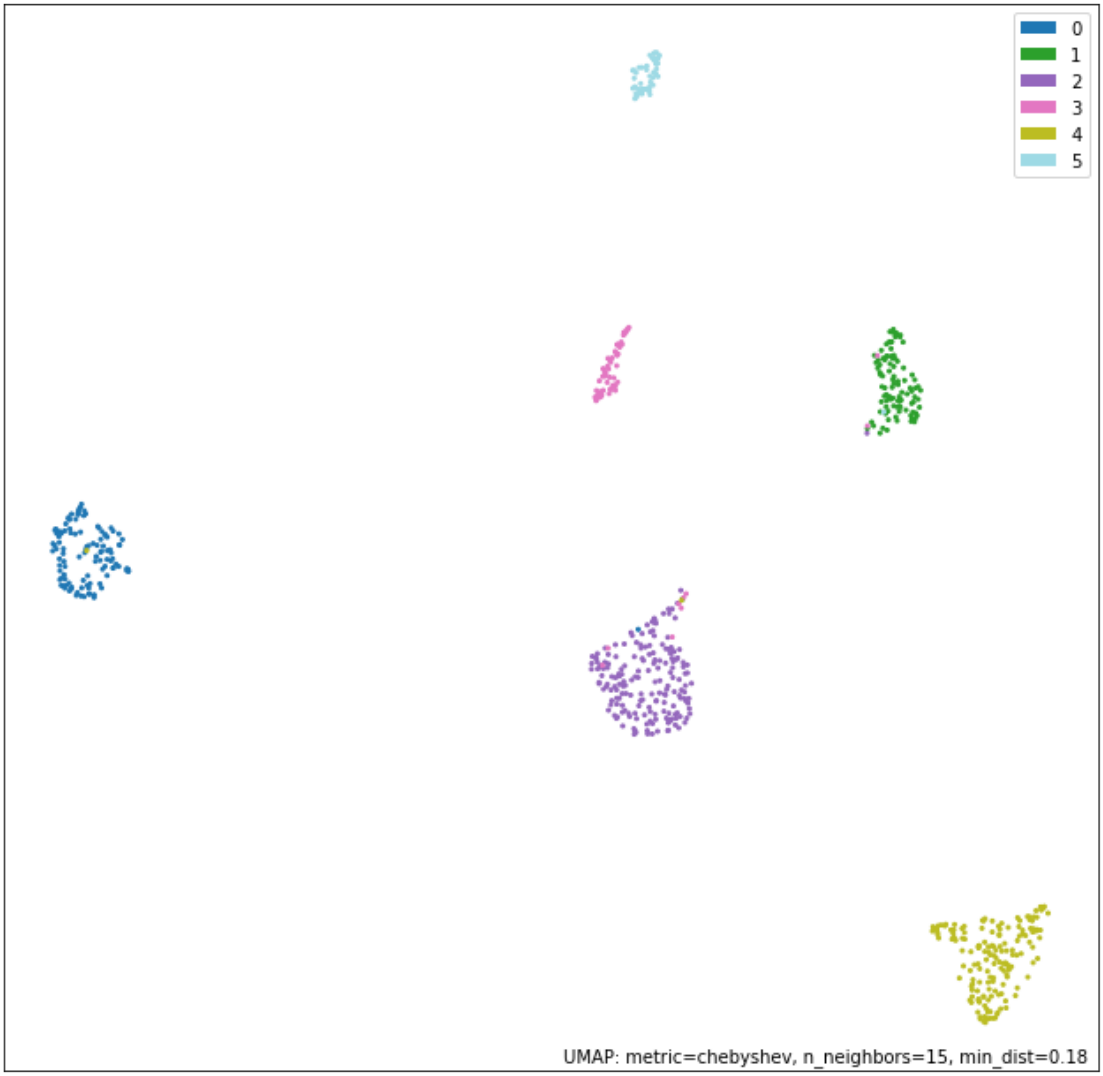

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=15, min_dist=0.03

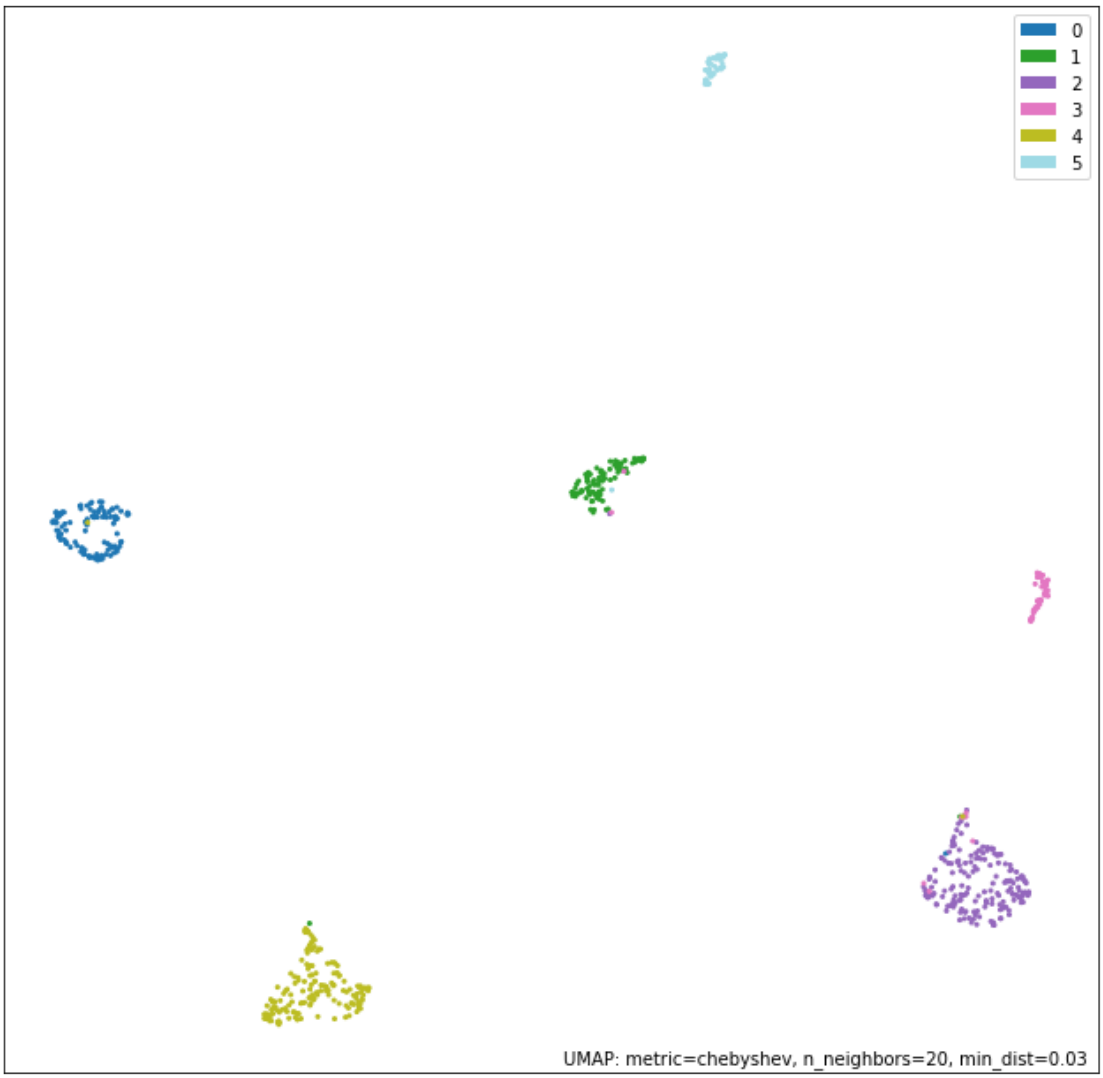

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=15, min_dist=0.06

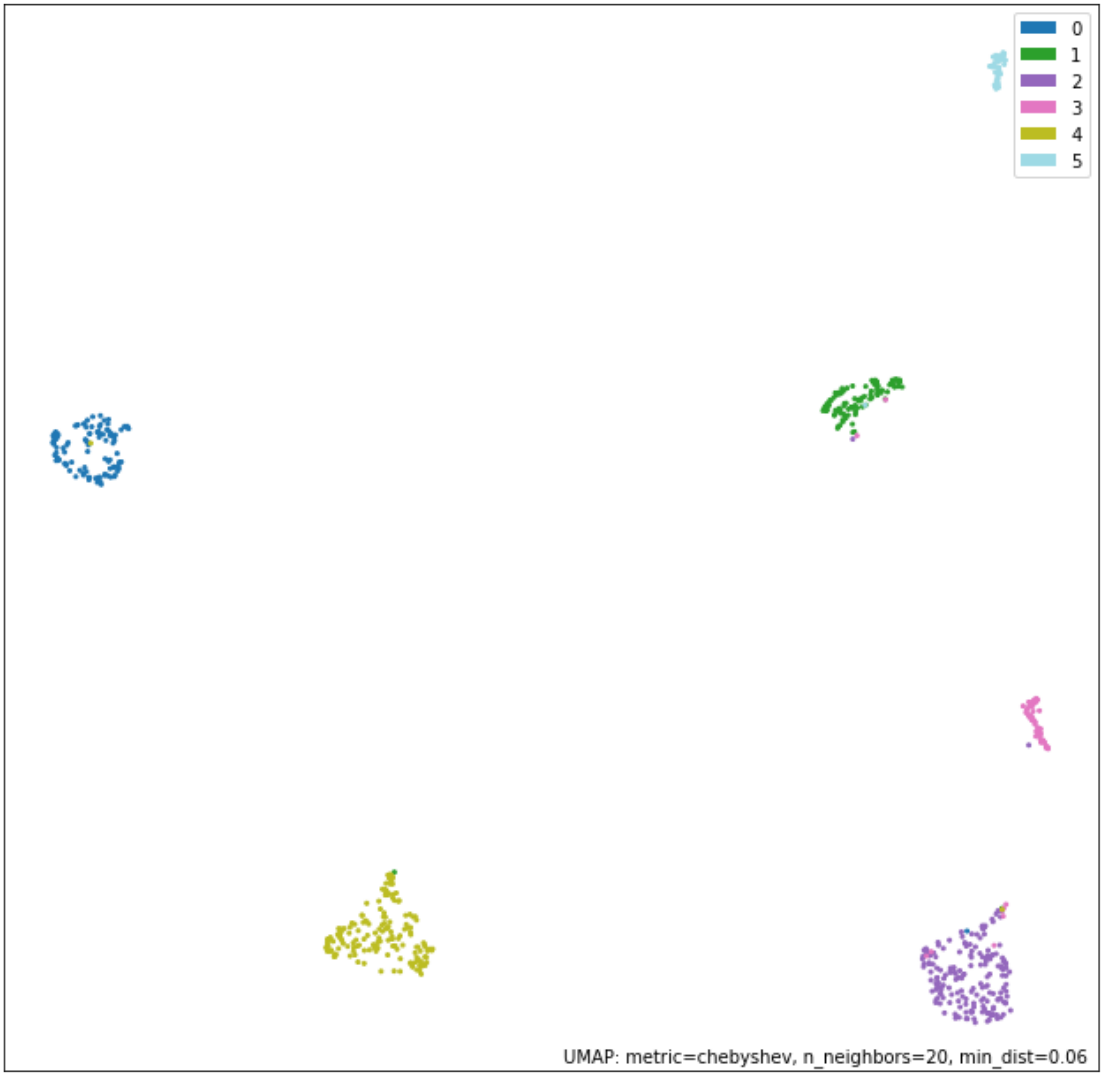

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=15, min_dist=0.09

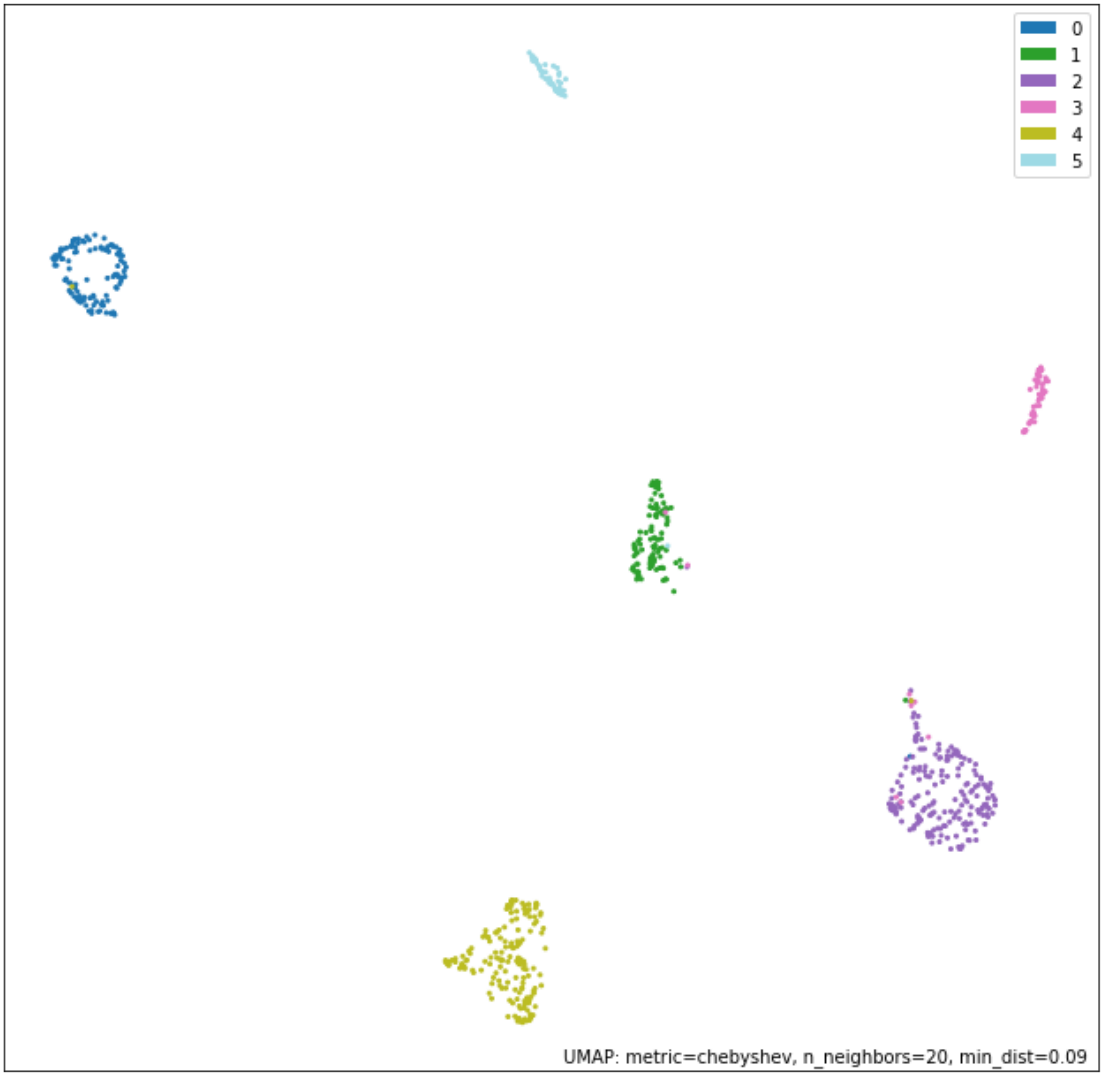

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=15, min_dist=0.12

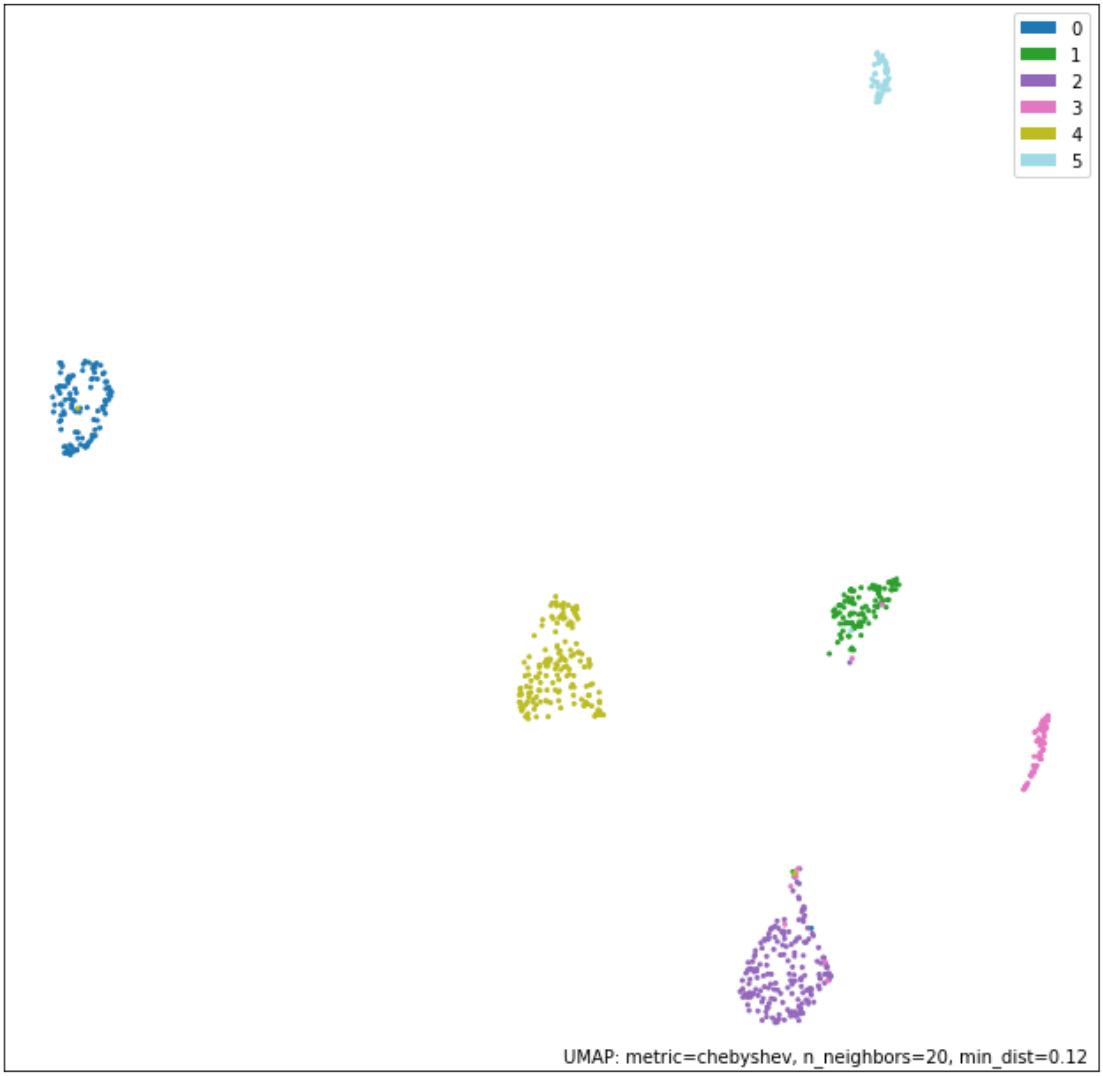

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=15, min_dist=0.15

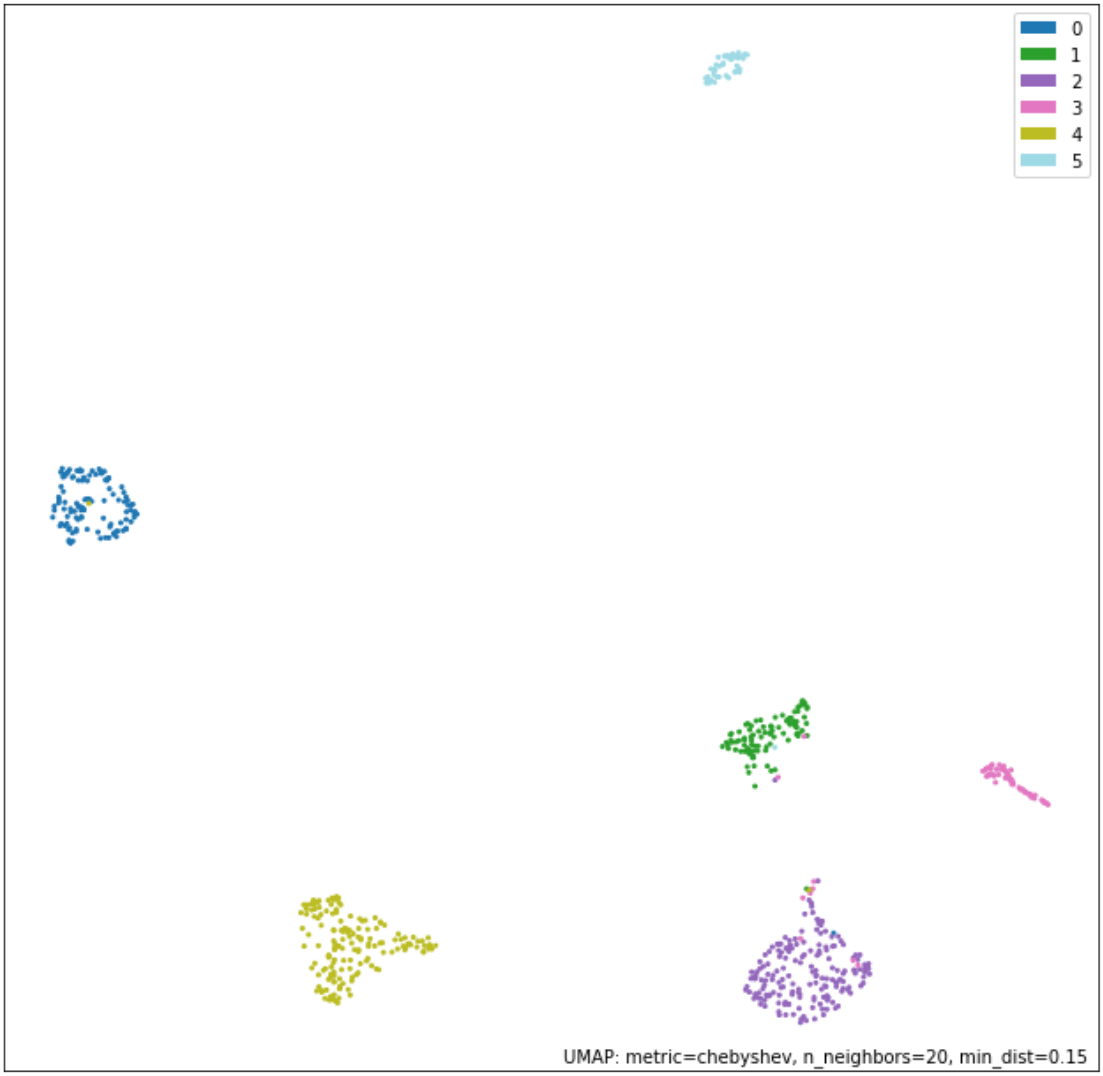

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=15, min_dist=0.18

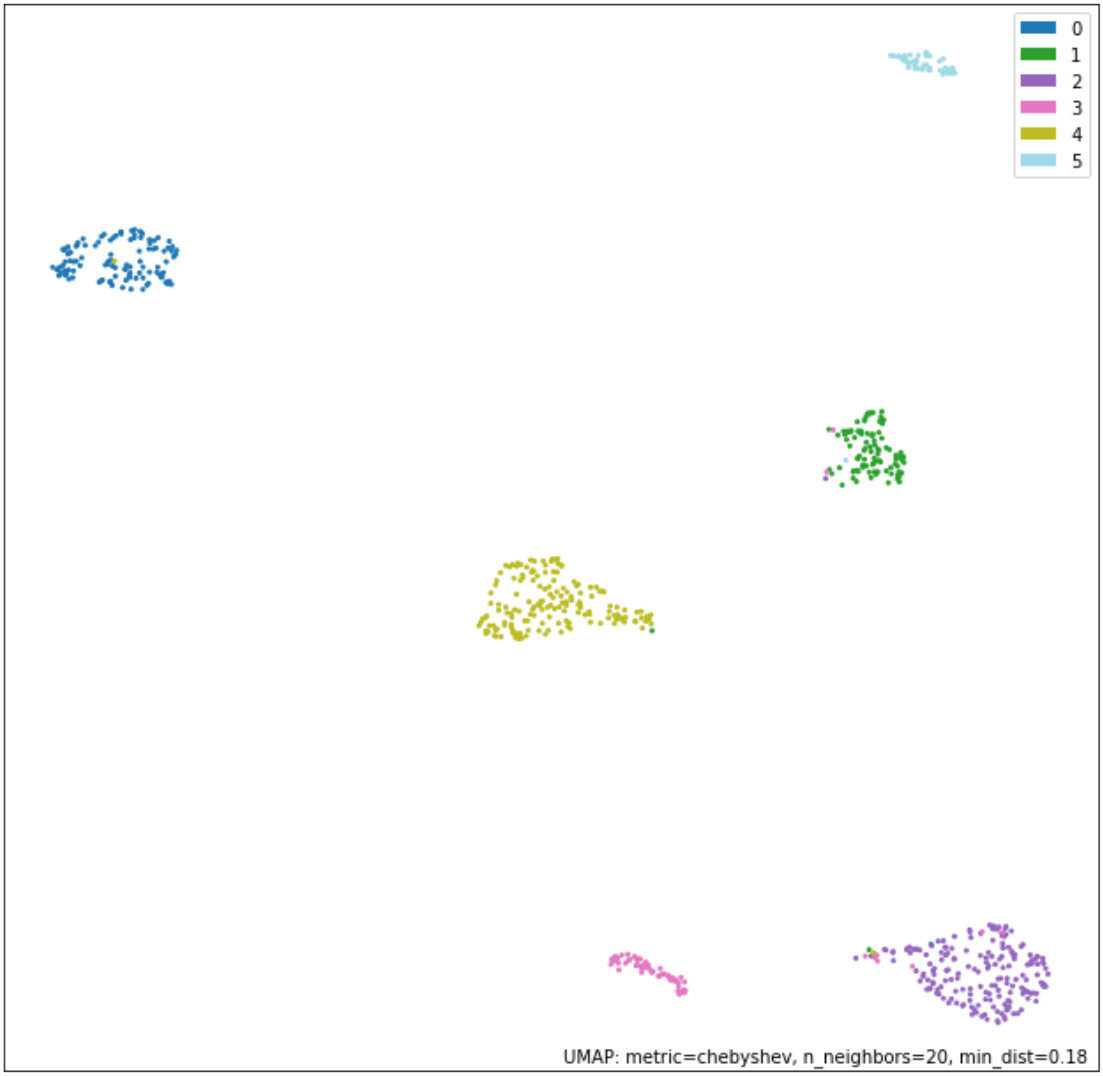

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=20, min_dist=0.03

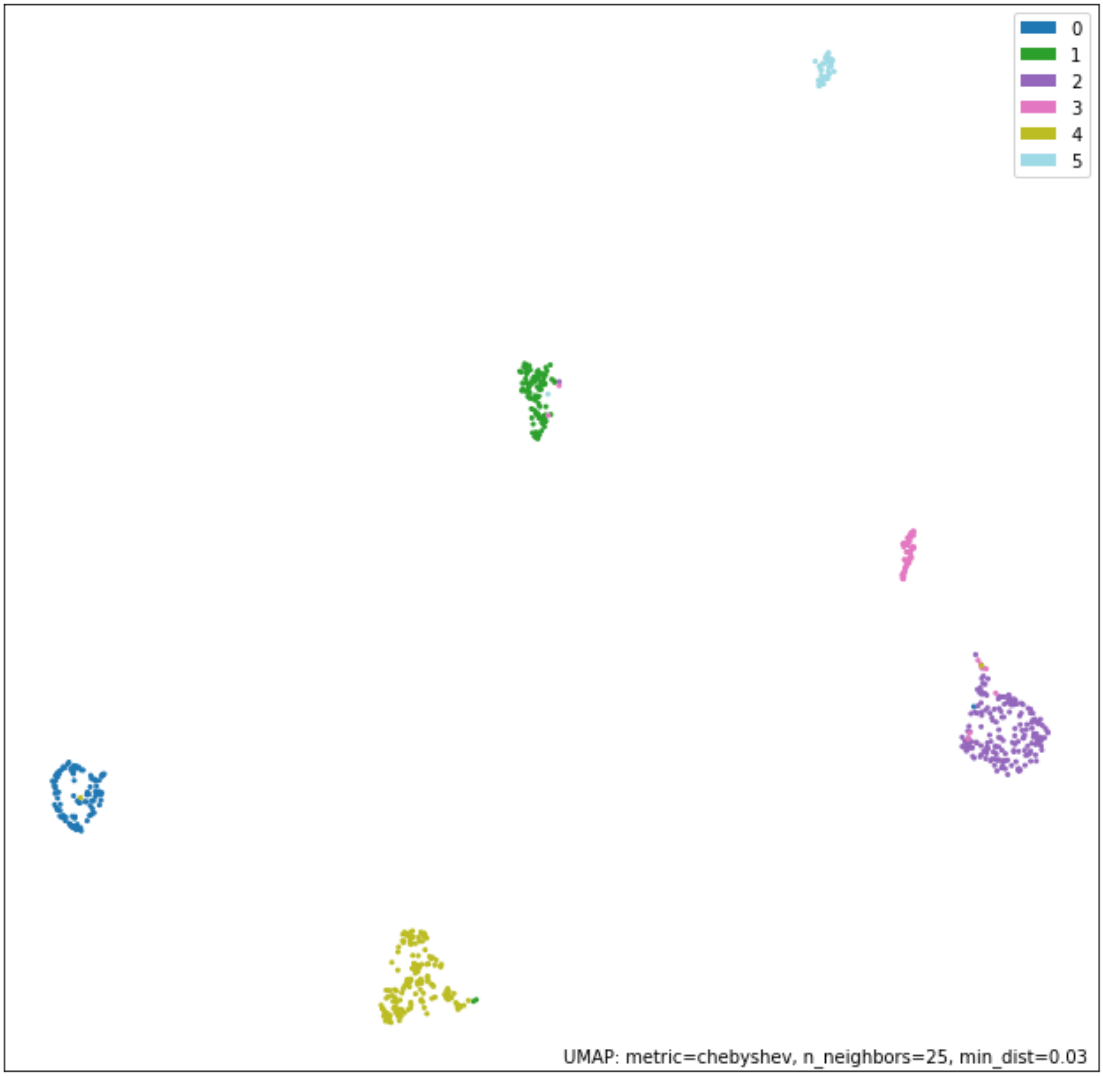

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=20, min_dist=0.06

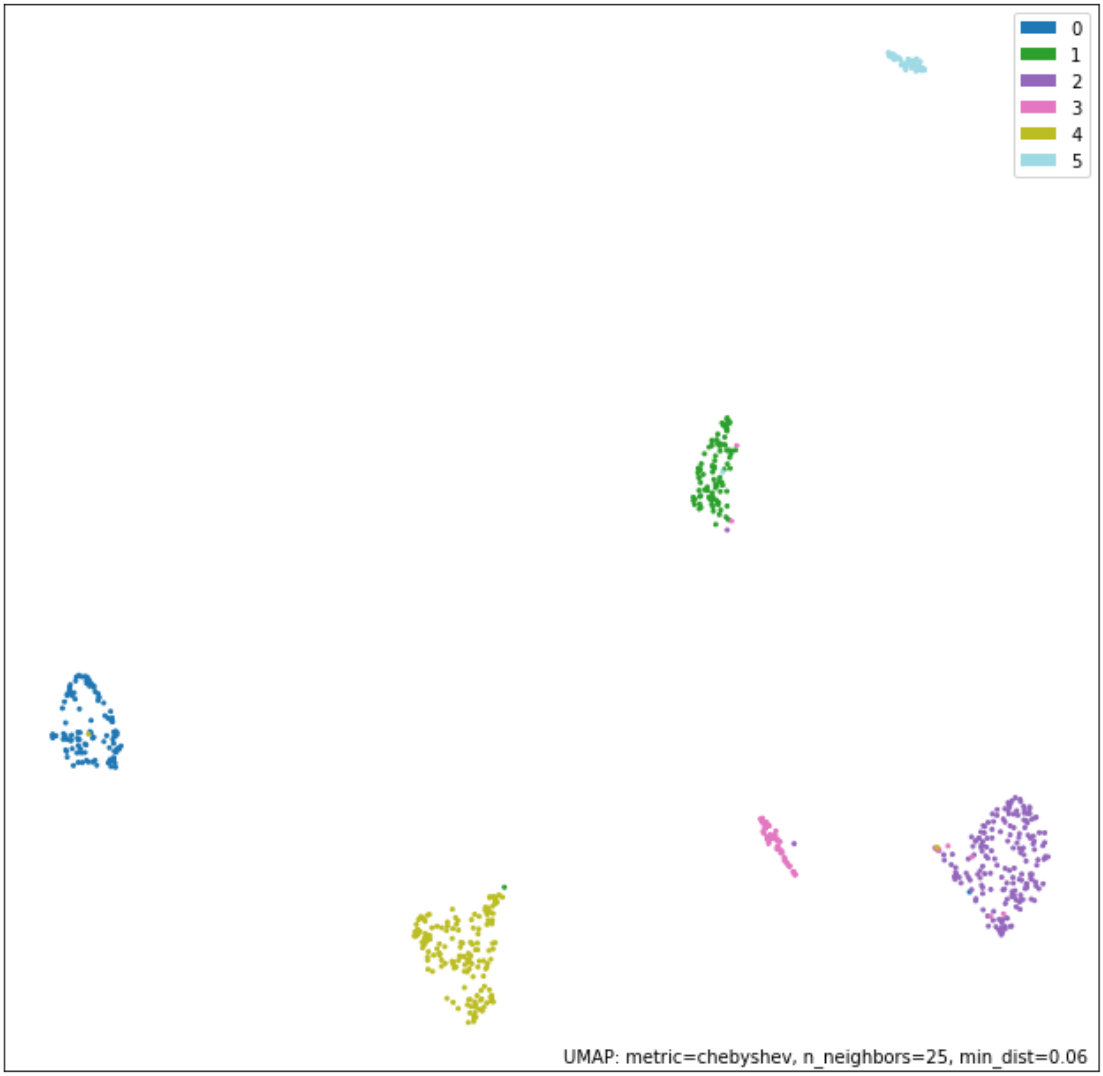

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=20, min_dist=0.09

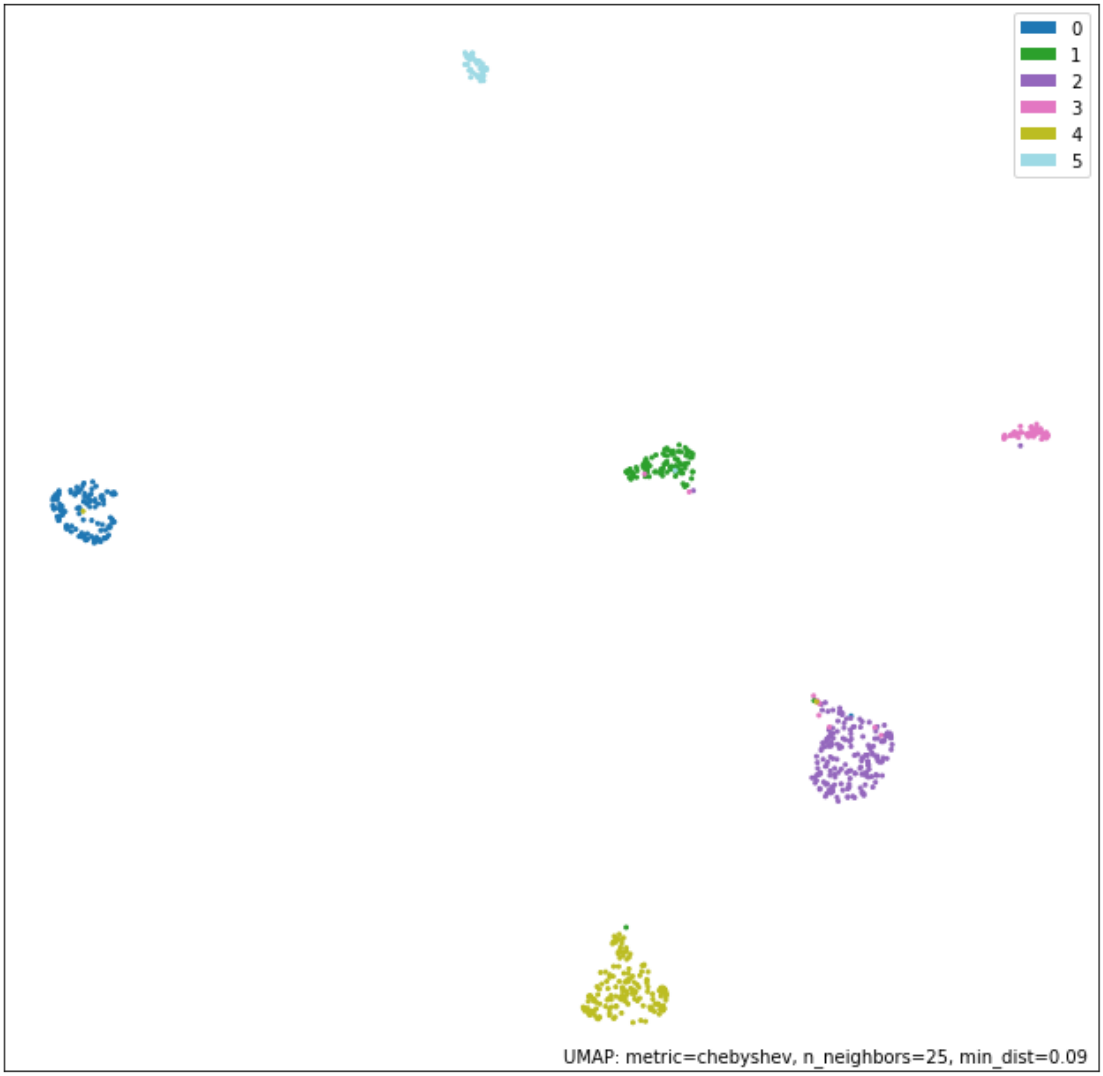

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=20, min_dist=0.12

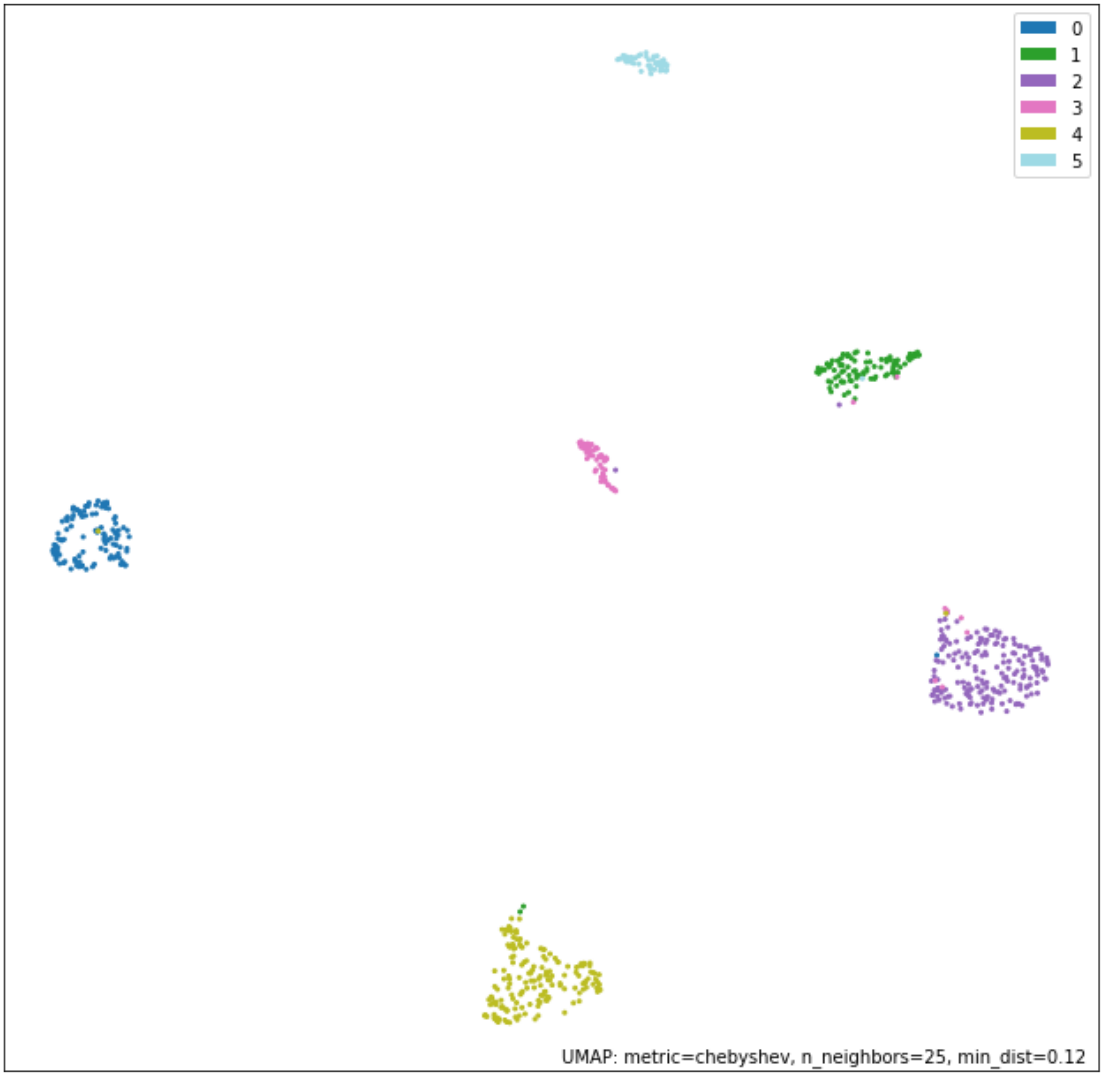

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=20, min_dist=0.15

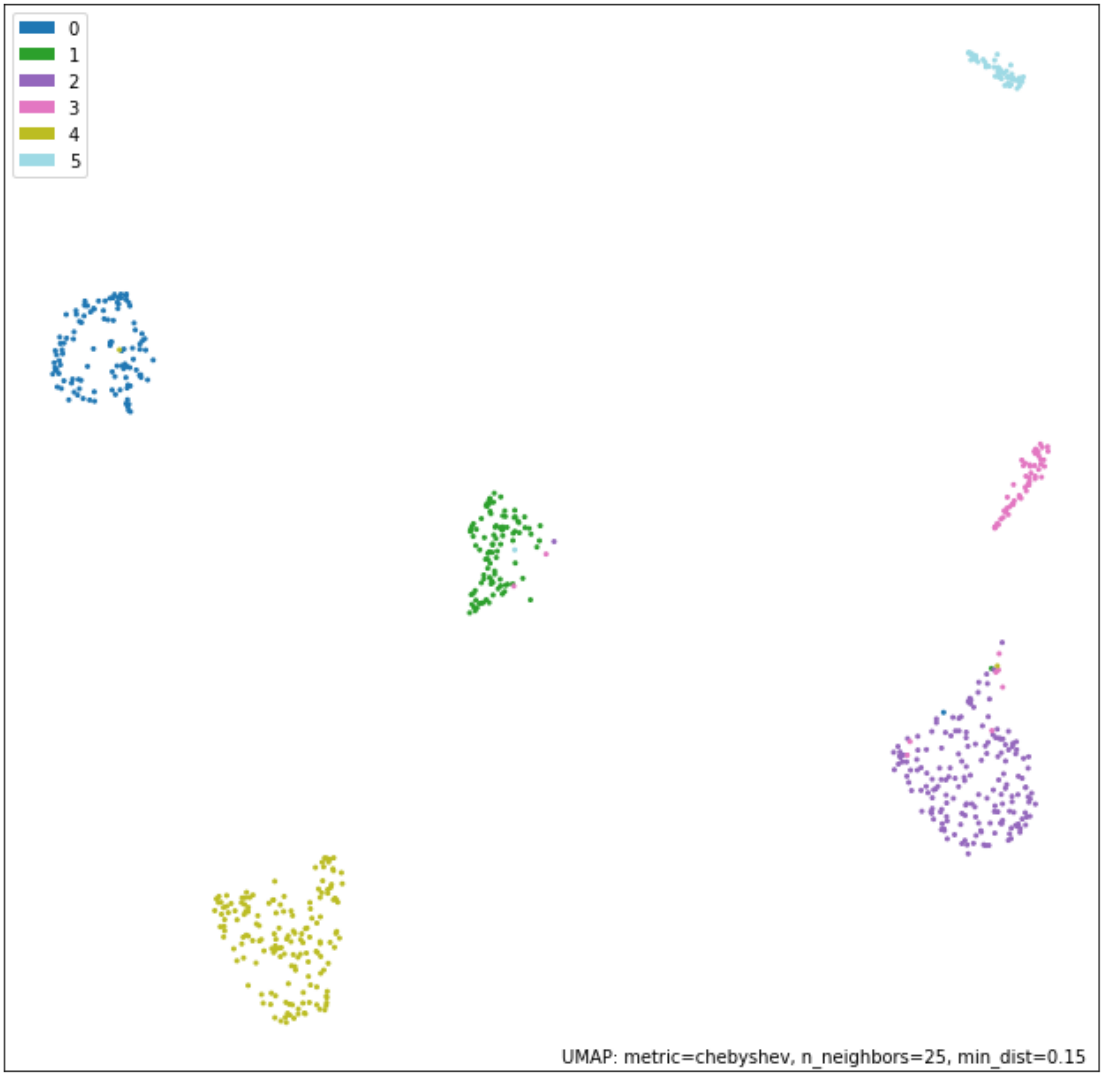

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=20, min_dist=0.18

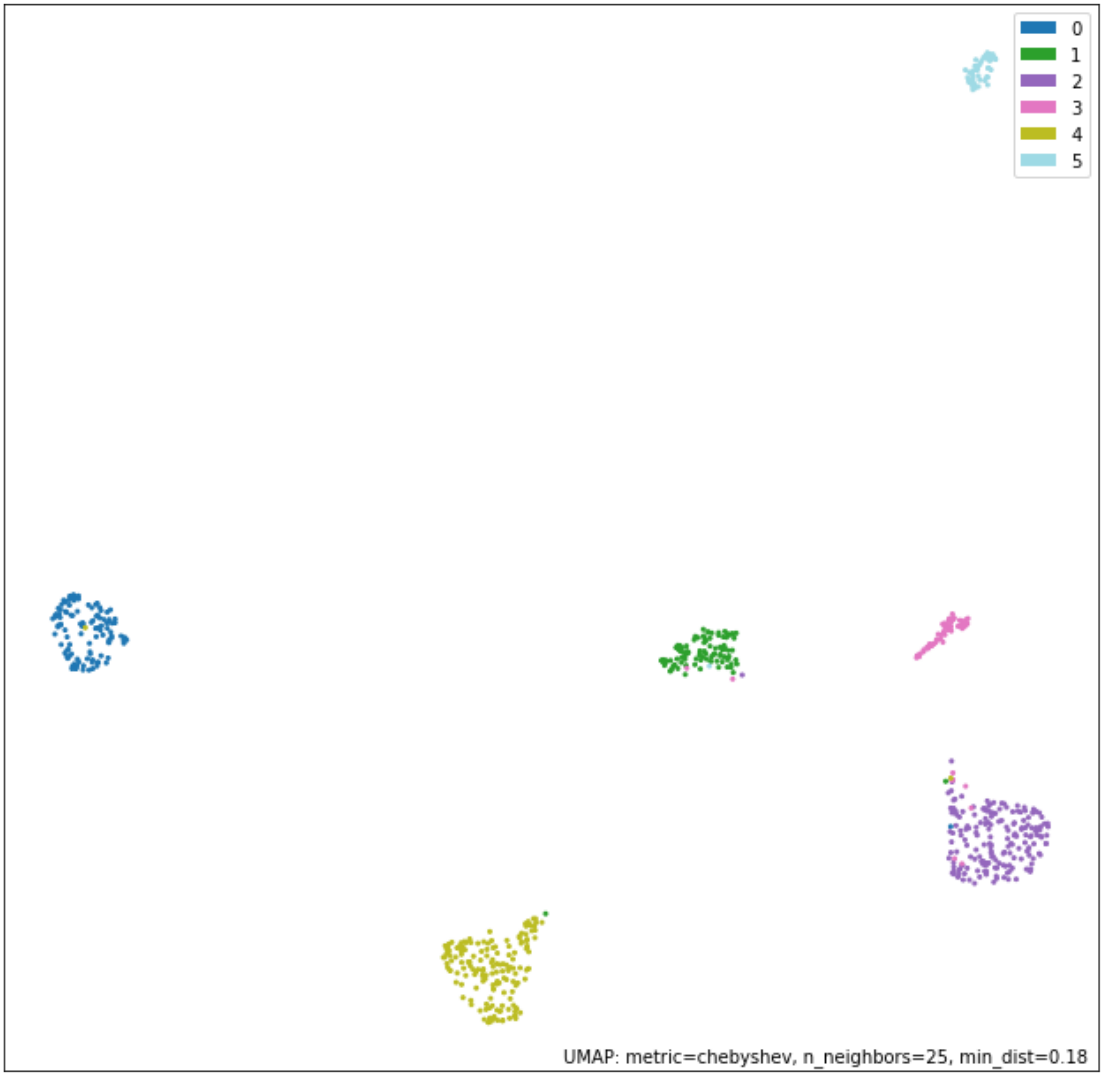

0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=25, min_dist=0.03


0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=25, min_dist=0.06


0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=25, min_dist=0.09


0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=25, min_dist=0.12


0
1
2
3
4
5
UMAP: metric=chebyshev, n_neighbors=25, min_dist=0.15



2.2 Plotting for the output before the pooling layer

Sometimes the output of the layer could be more than 2-D, in this situation, we have to reduce the dimension of the dataset.

[21]:
```
newModel_before_pool = Sequential()
for layer in layers[:2]:
    newModel_before_pool.add(layer)
newModel_before_pool.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 2000, 20)          0
```

```
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1996, 250)         25250
=================================================================
Total params: 25,250
Trainable params: 25,250
Non-trainable params: 0
_________________________________________________________________
```

[22]:
```
predicted_Probability_before_pool = newModel_before_pool.predict(testDataMats)
```

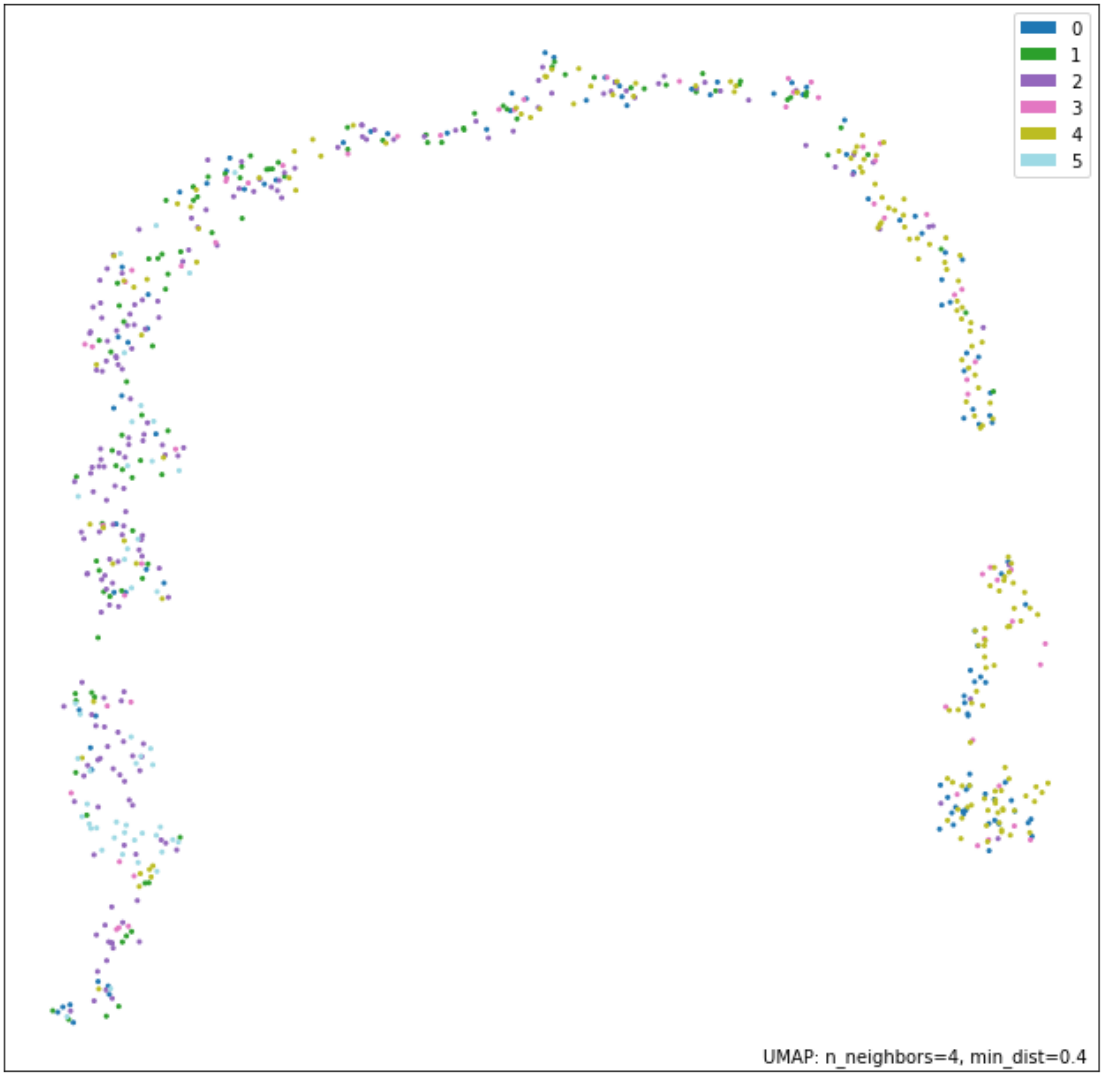
[23]:
```
predicted_Probability_before_pool.shape
```

[23]:
```
(667, 1996, 250)
```

Here we simply use np.mean for dimensional reduction.

You can try np.max, np.mean or flatten as needed.

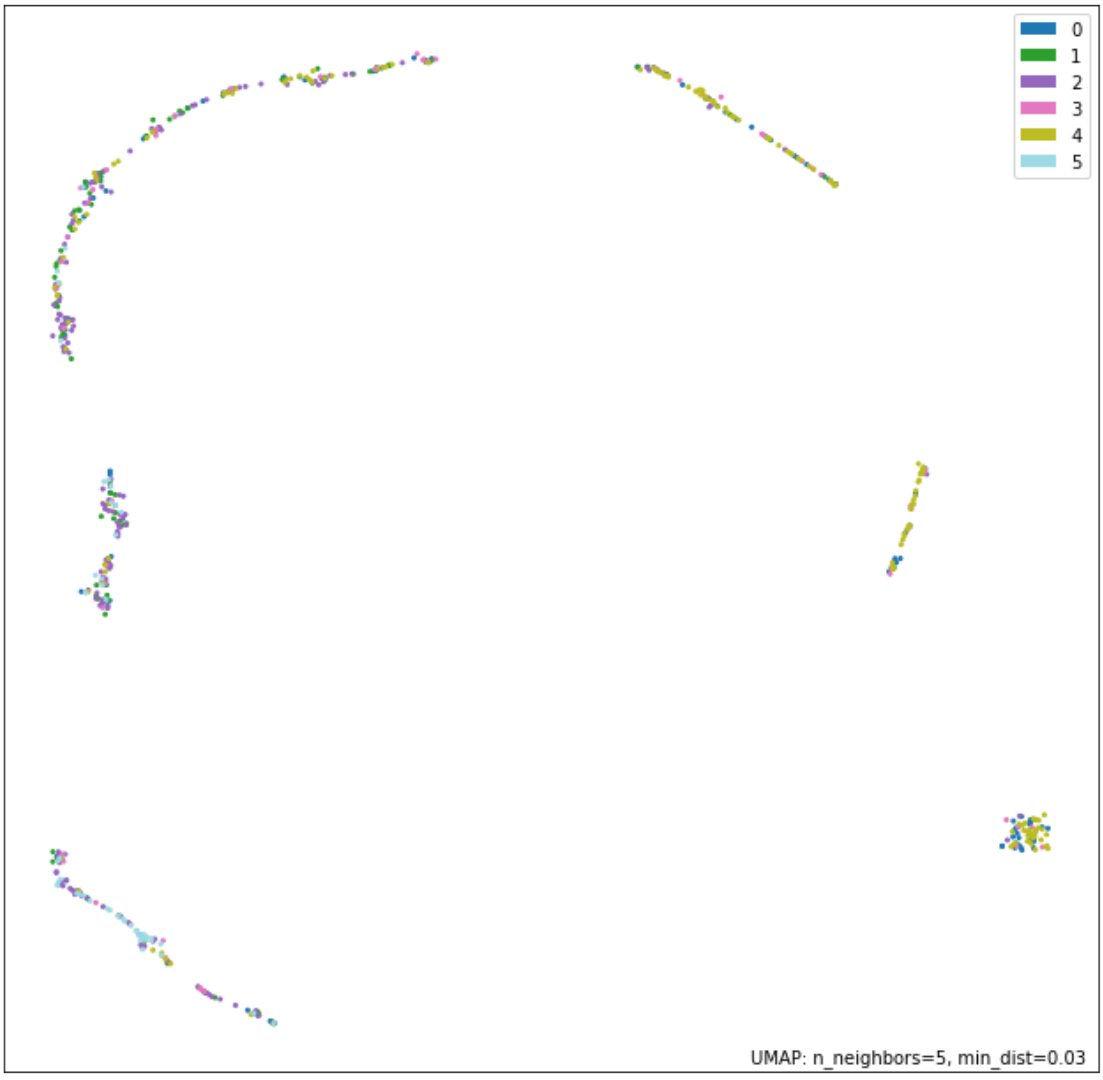
[24]:
```
# the featureDict could contain more parameters, use '?umap.UMAP' for more␣
 ↪details.
featureDict={
        'n_neighbors' : 4,
        'min_dist' : 0.4,
        'metric' : 'euclidean',
        }

mapper = umap.UMAP(**featureDict).fit(np.
 ↪mean(predicted_Probability_before_pool,axis=2))
plotObj = umap.plot.points(mapper, labels=testLabelArr,  theme='blue')
plt.show()
```

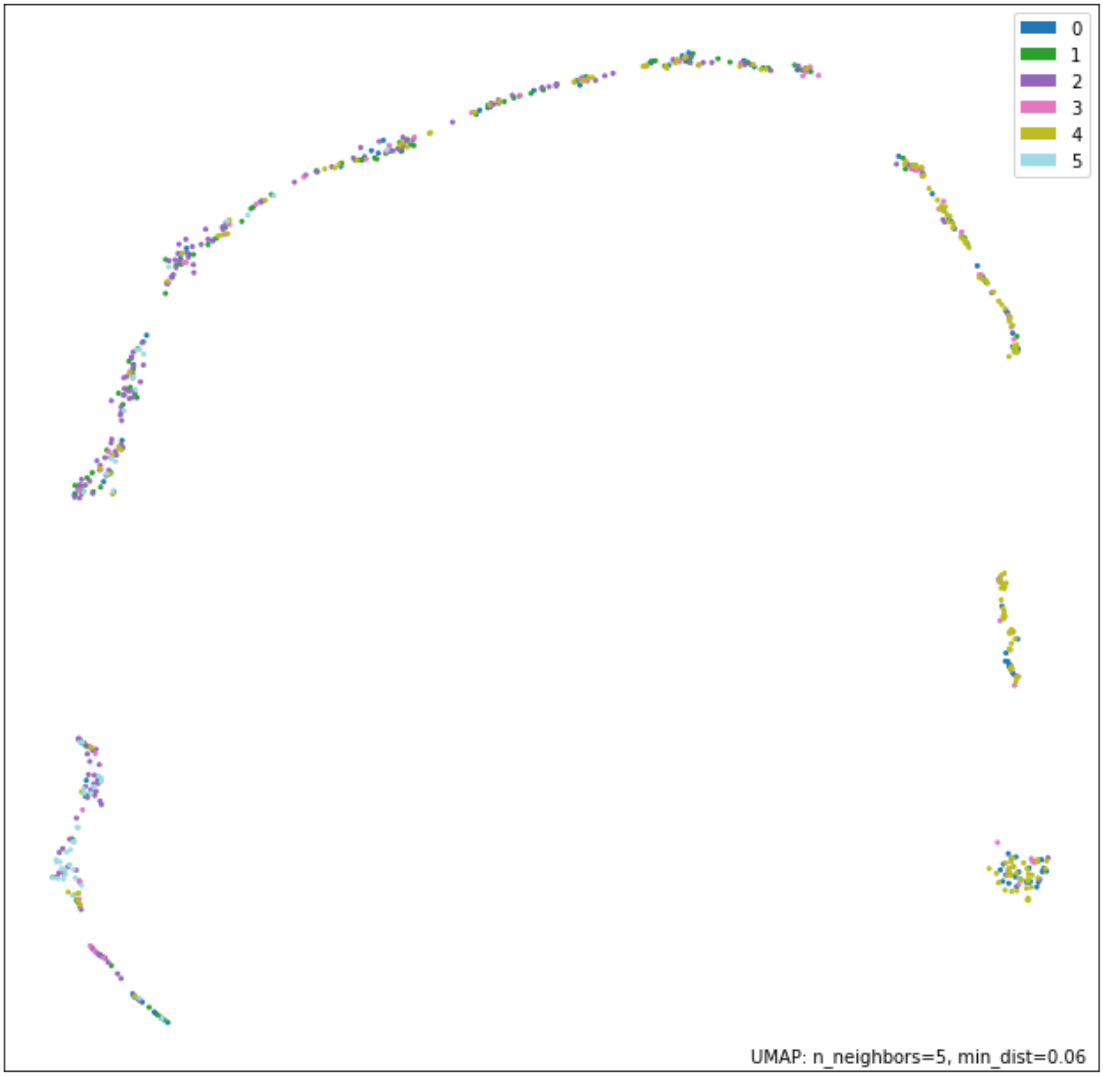


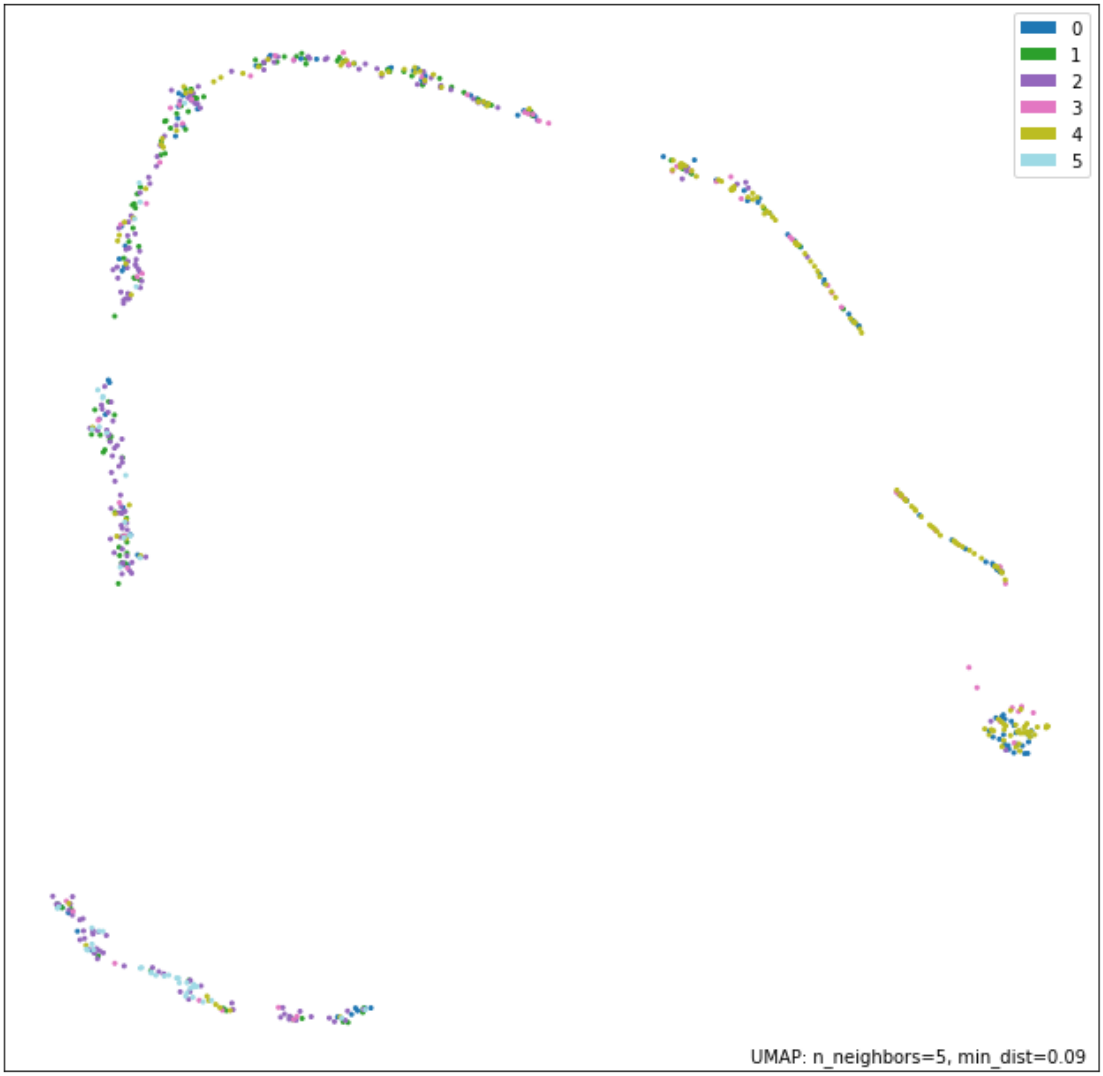
Plotting with different paramenters by changing the ‘n_neighbors’ and ‘min_dist’ .

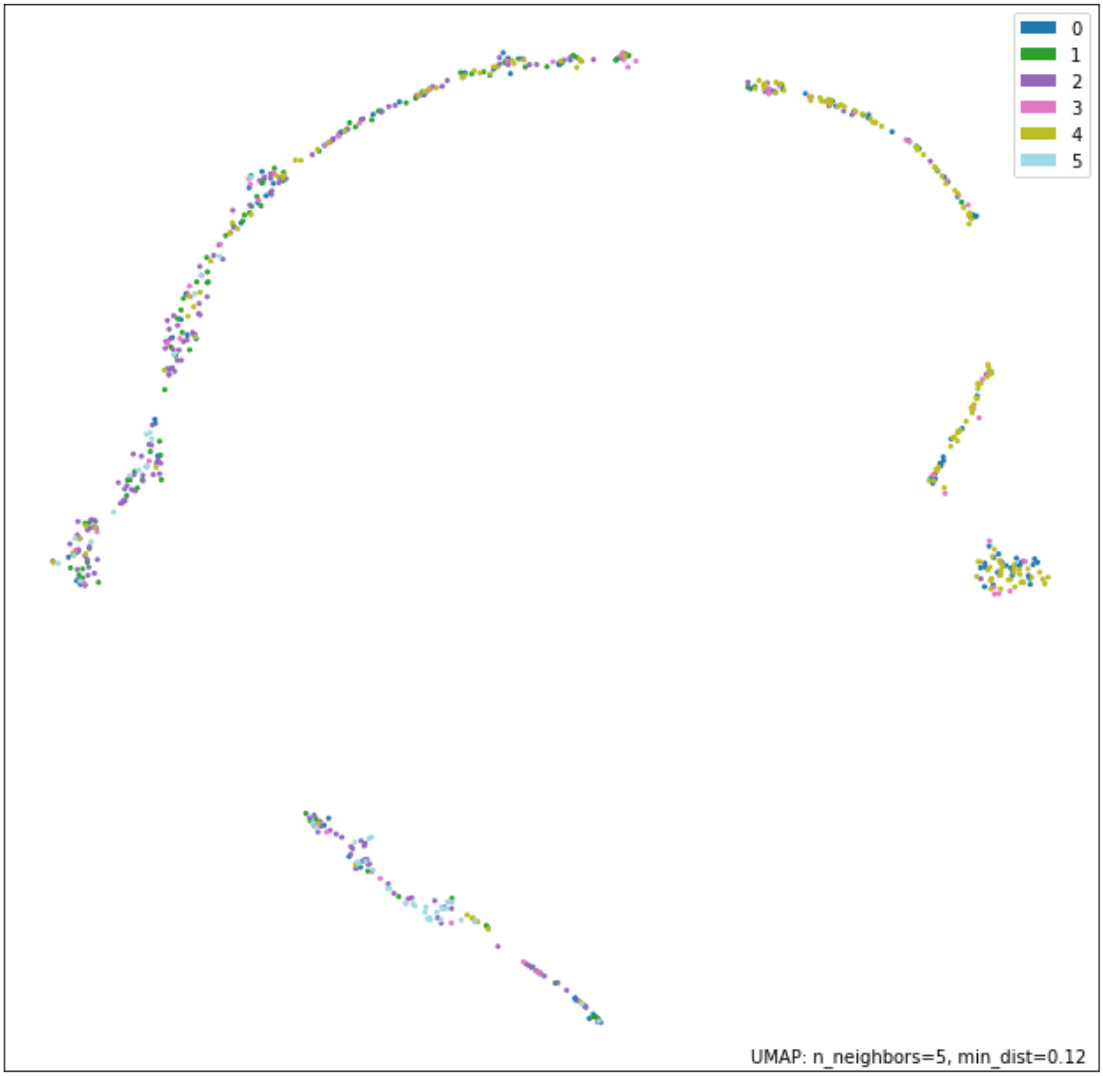
[25]:
```
featureDict={
        'n_neighbors' : 4,
        'min_dist' : 0.4,
        'metric' : 'euclidean',
        }
for n_neighbors in np.arange(5,30,5).astype(int):
    featureDict['n_neighbors'] = n_neighbors
    for min_dist in np.arange(0.03,0.2,0.03):
        featureDict['min_dist'] = min_dist
        mapper = umap.UMAP(**featureDict).fit(np.
 ↪mean(predicted_Probability_before_pool,axis=2))
        plotObj = umap.plot.points(mapper, labels=testLabelArr, theme='blue')
```

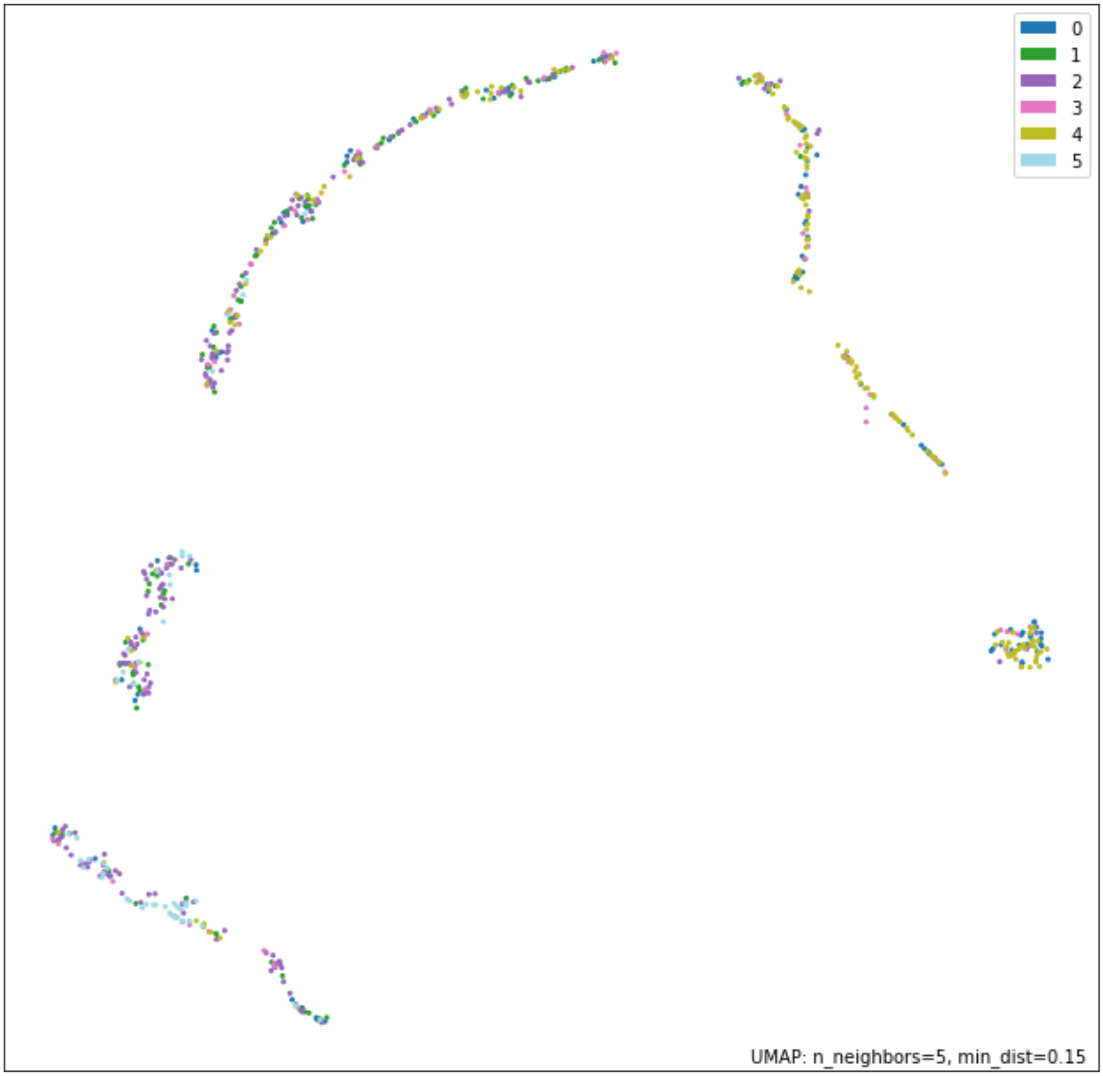
```
        plt.show()
        plt.close() #close the plotted figure
```

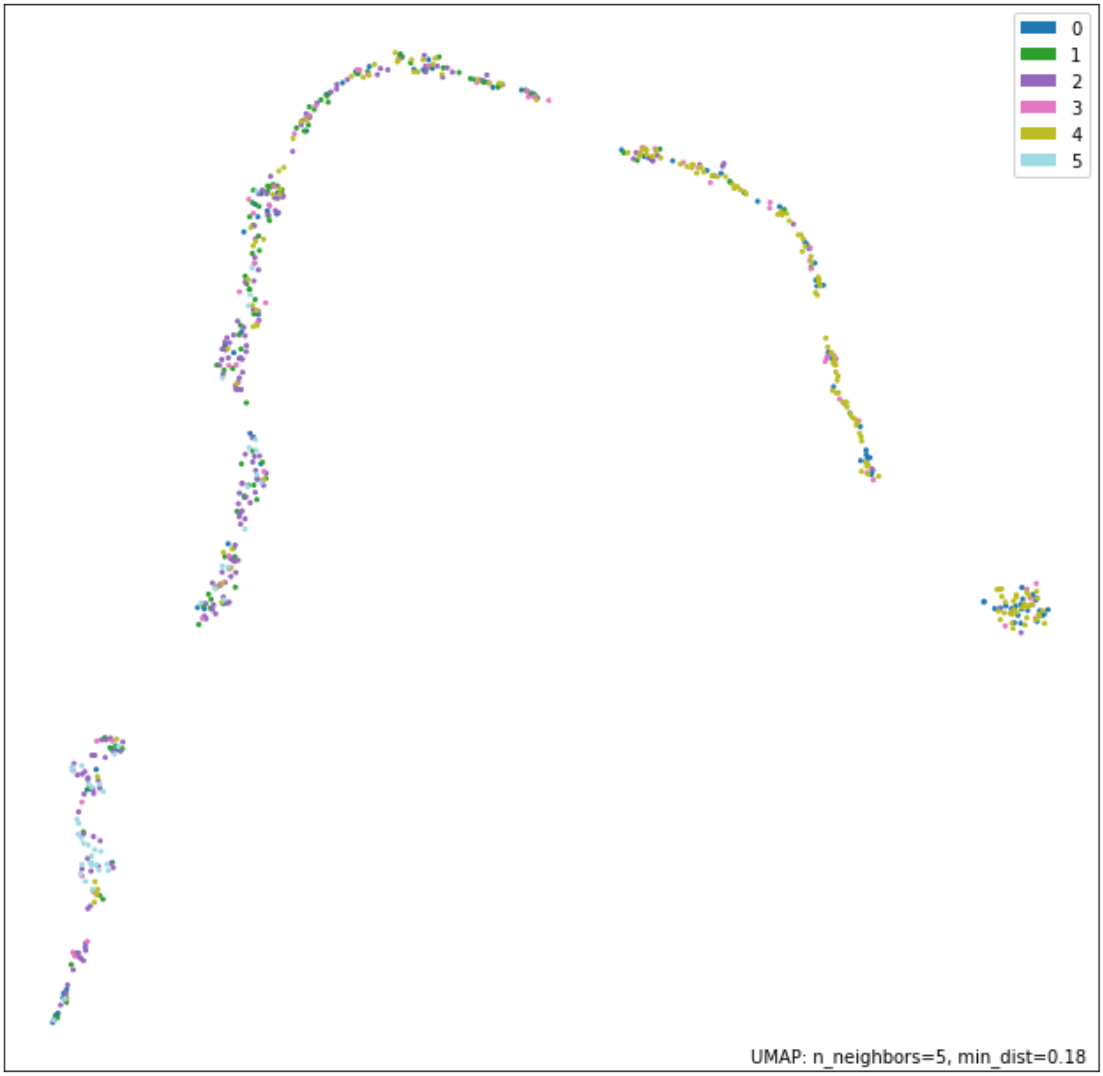


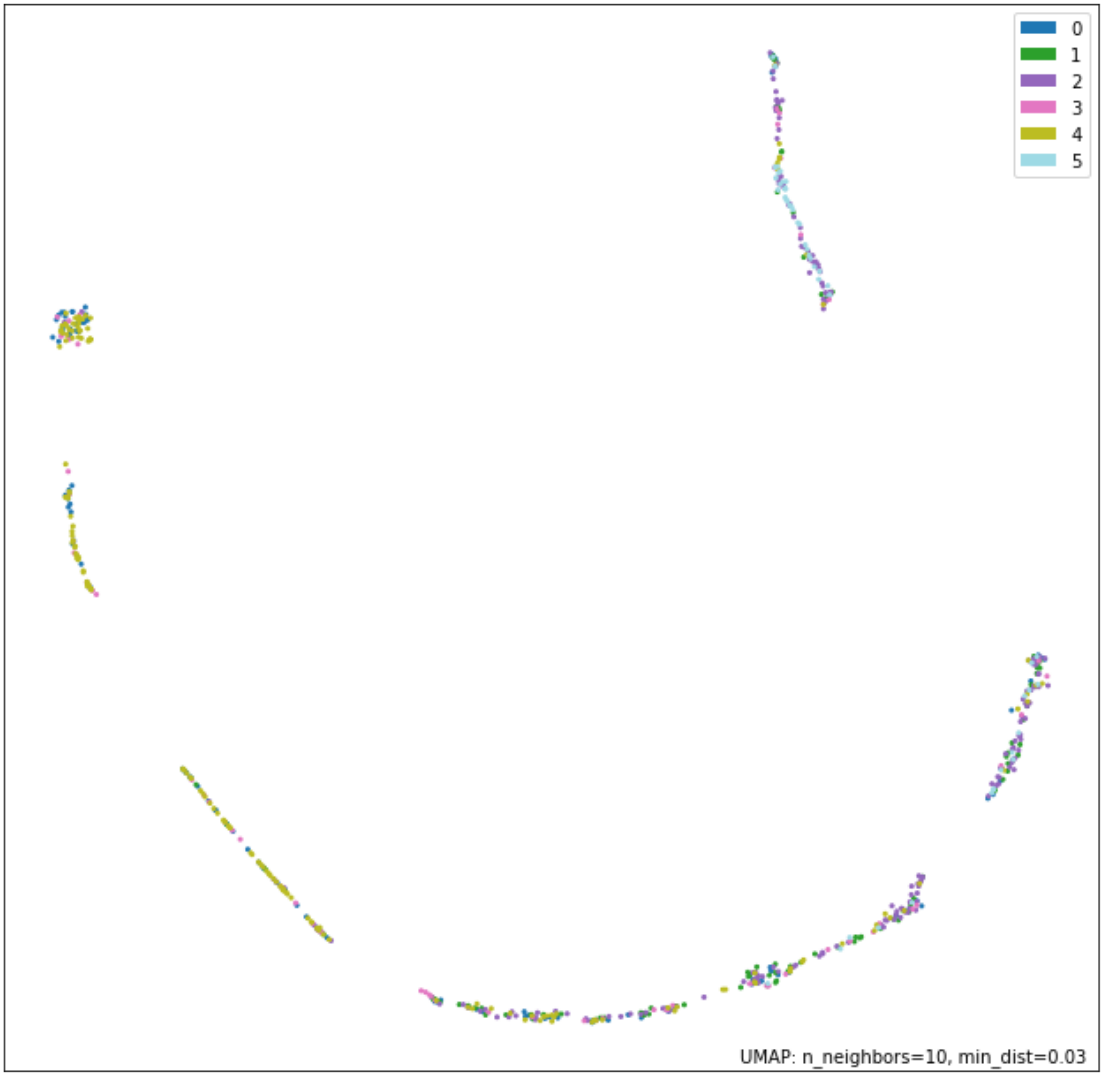

0
1
2
3
4
5
UMAP: n_neighbors=5, min_dist=0.06

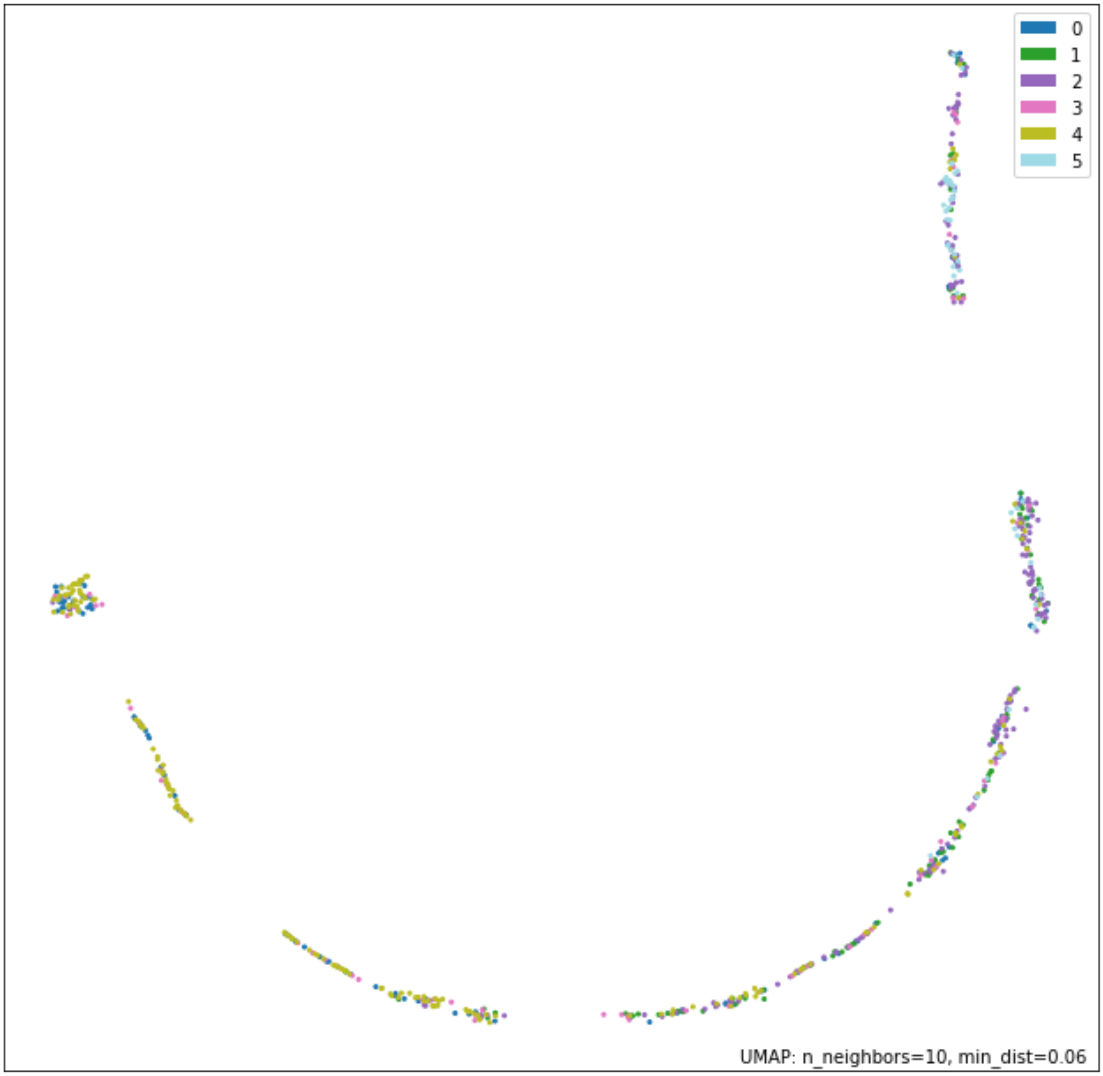

0
1
2
3
4
5
UMAP: n_neighbors=5, min_dist=0.09

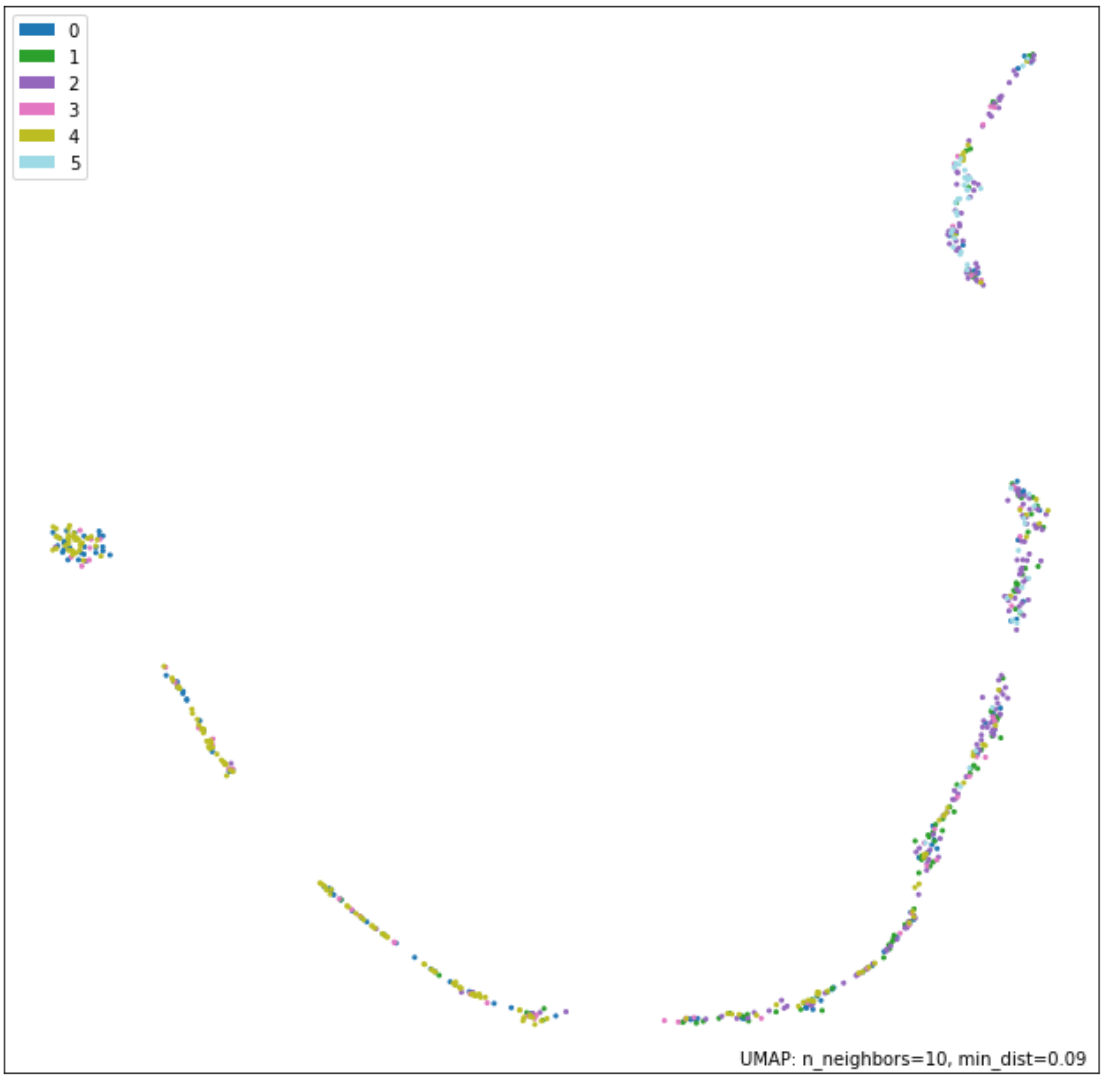

0
1
2
3
4
5
UMAP: n_neighbors=5, min_dist=0.12

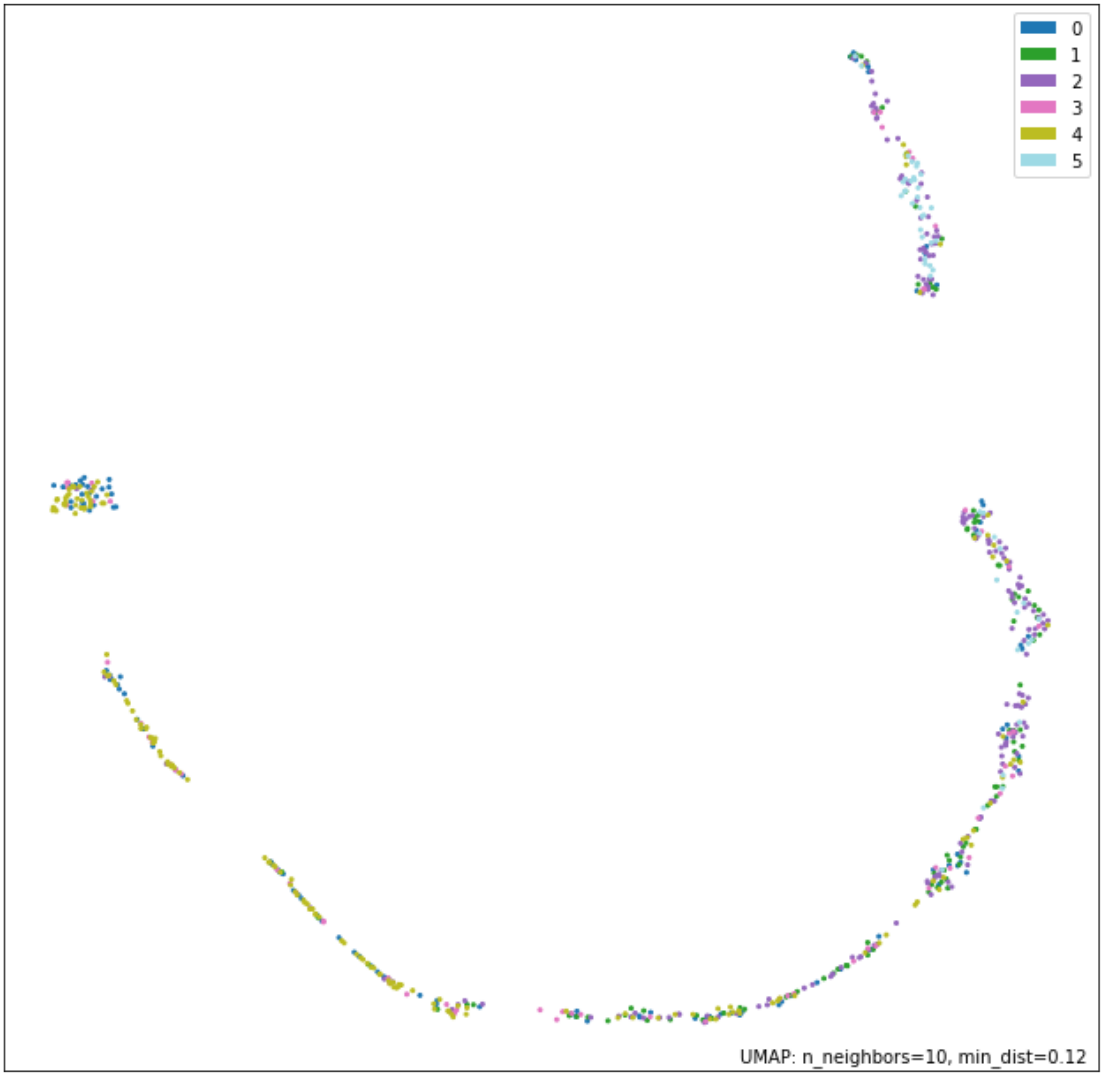

0
1
2
3
4
5
UMAP: n_neighbors=5, min_dist=0.15

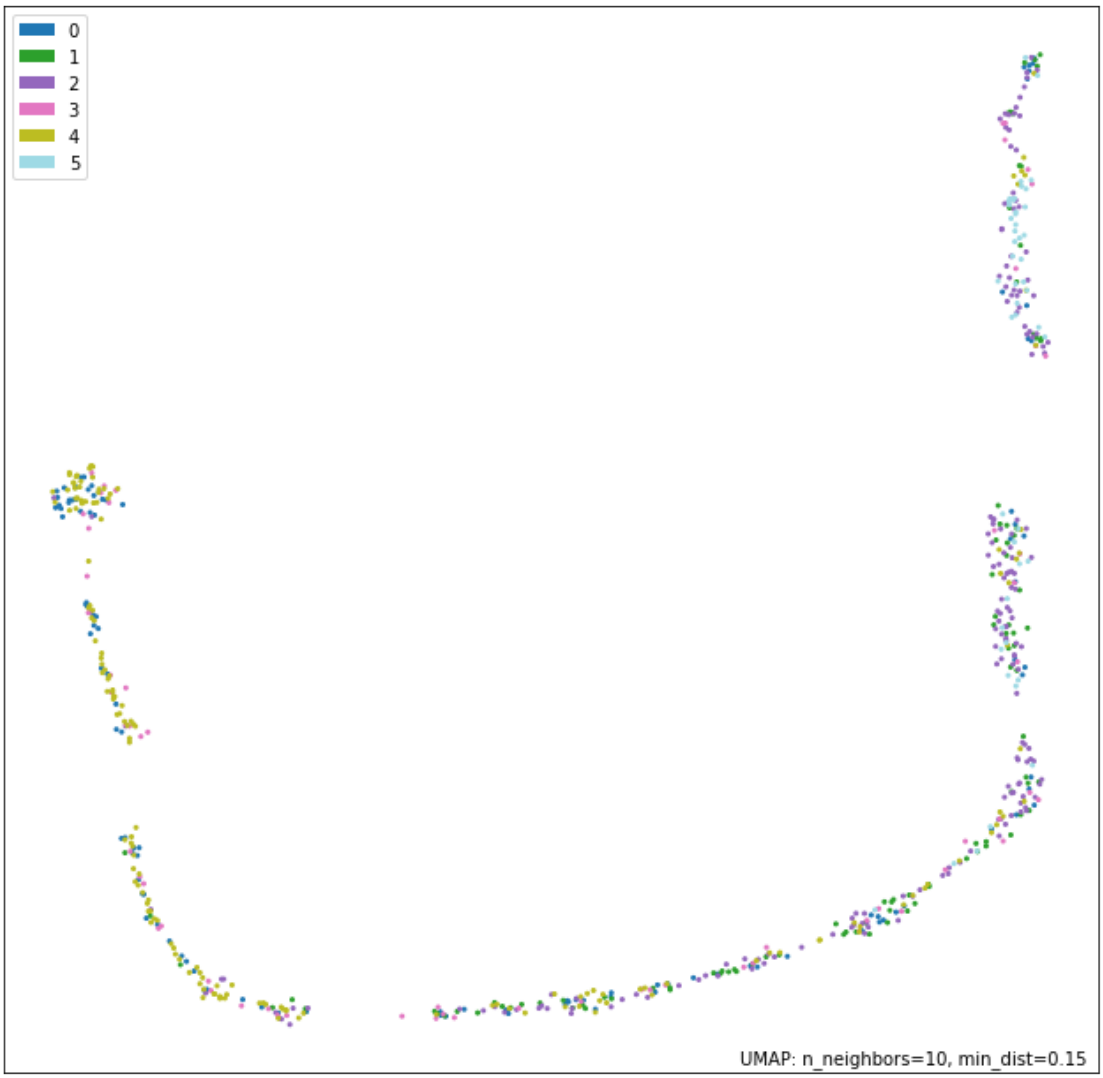

0
1
2
3
4
5
UMAP: n_neighbors=5, min_dist=0.18

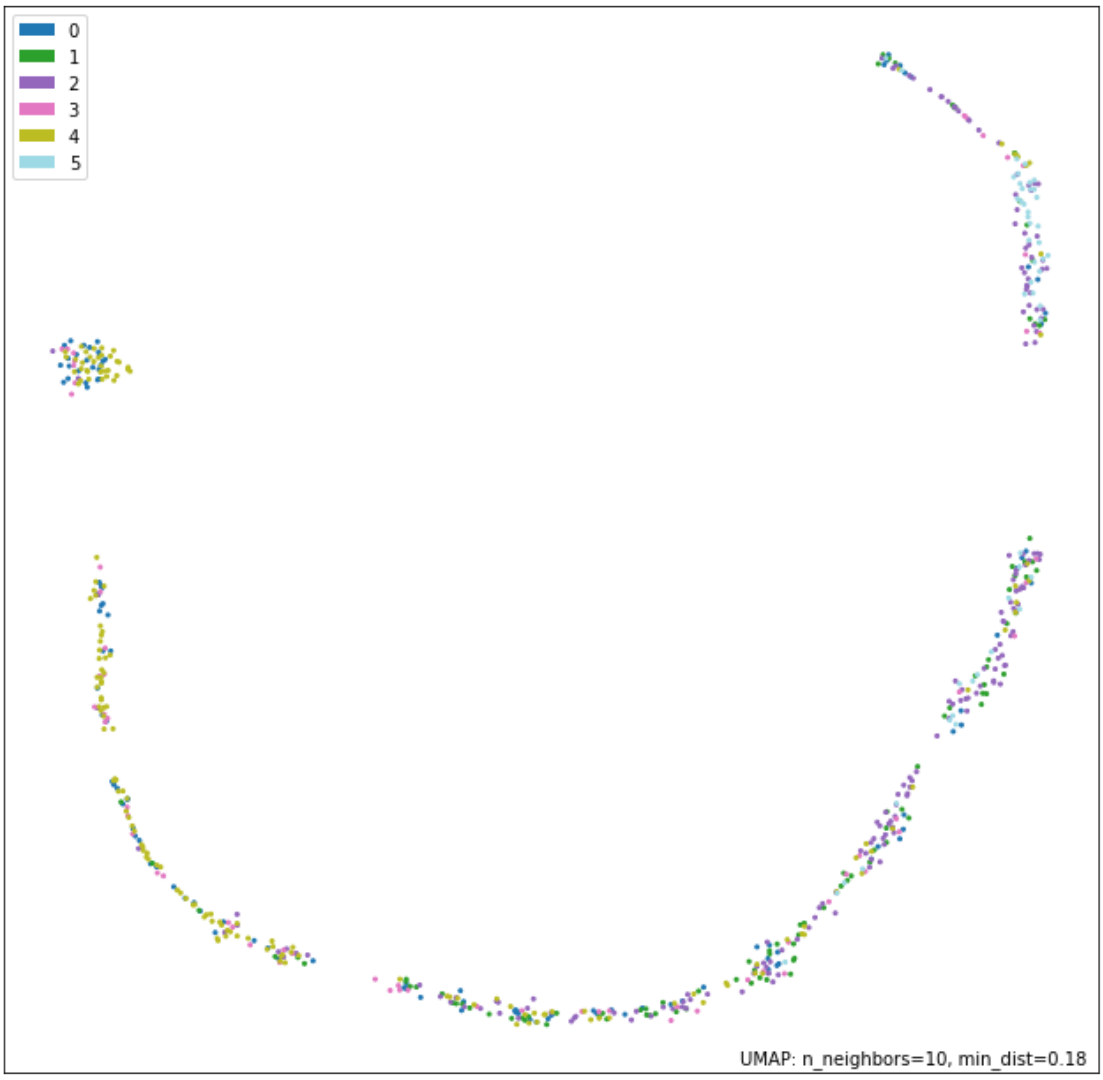

0
1
2
3
4
5
UMAP: n_neighbors=10, min_dist=0.03

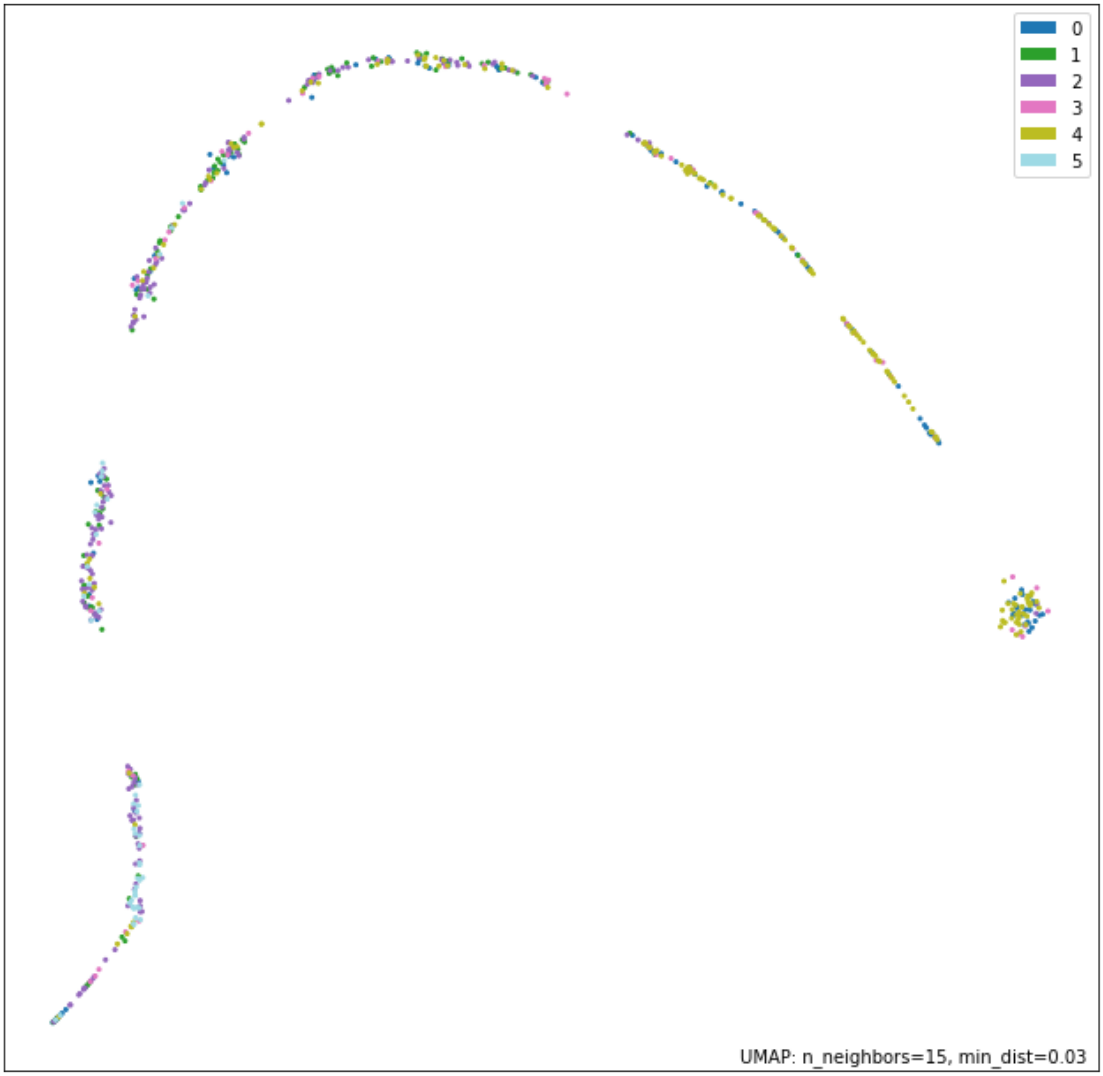

0
1
2
3
4
5
UMAP: n_neighbors=10, min_dist=0.06

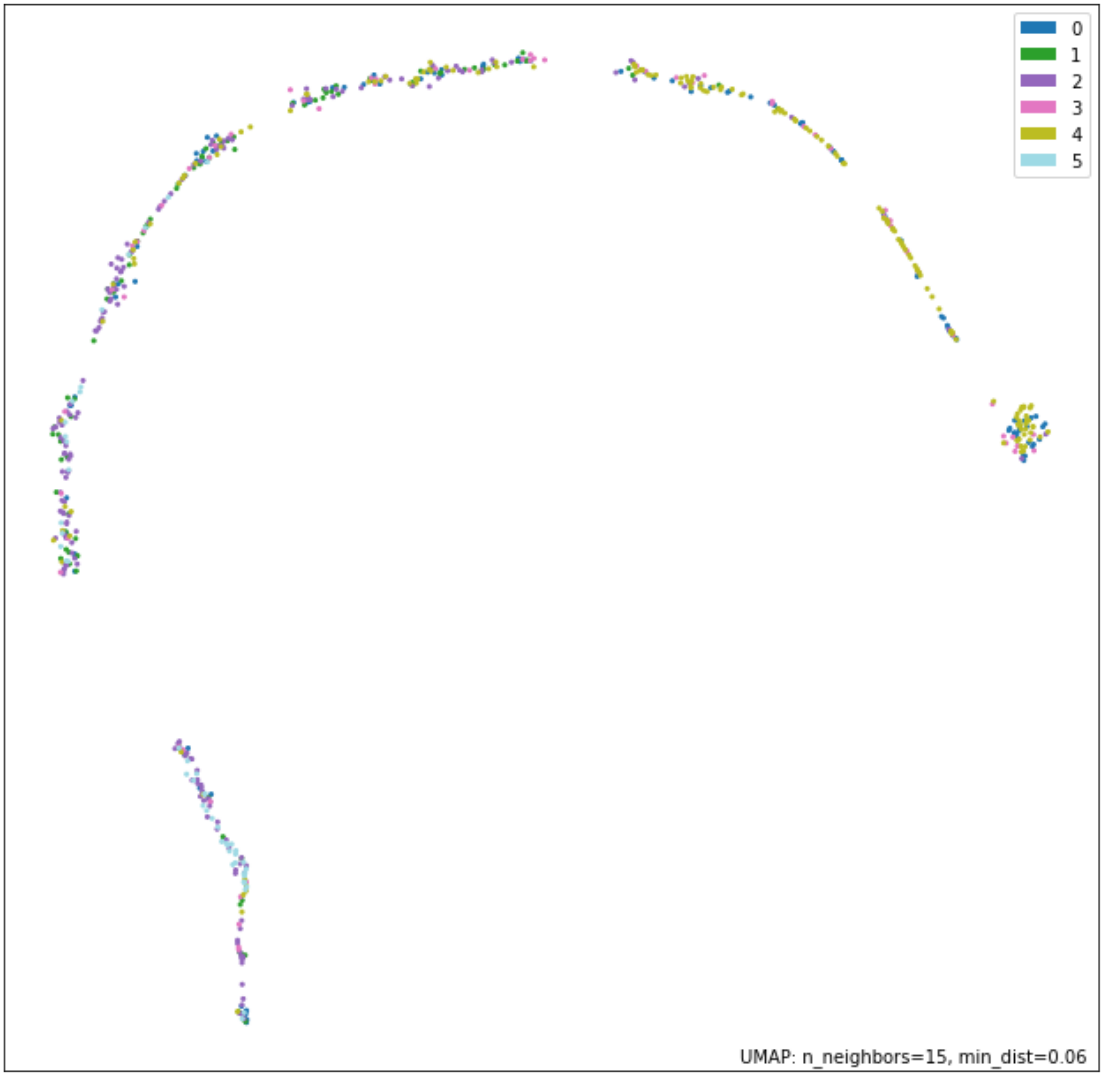

0
1
2
3
4
5
UMAP: n_neighbors=10, min_dist=0.09

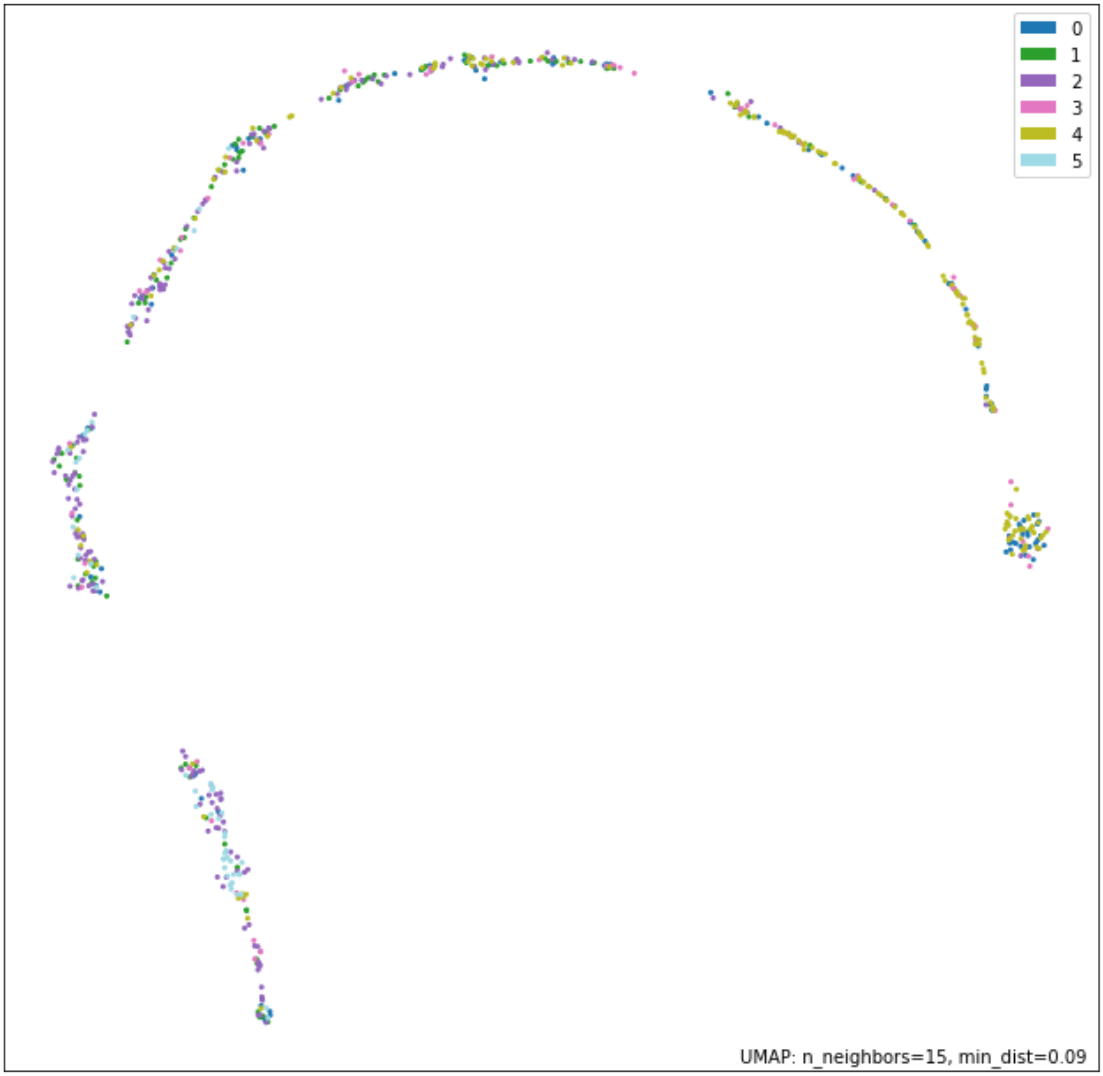

0
1
2
3
4
5
UMAP: n_neighbors=10, min_dist=0.12

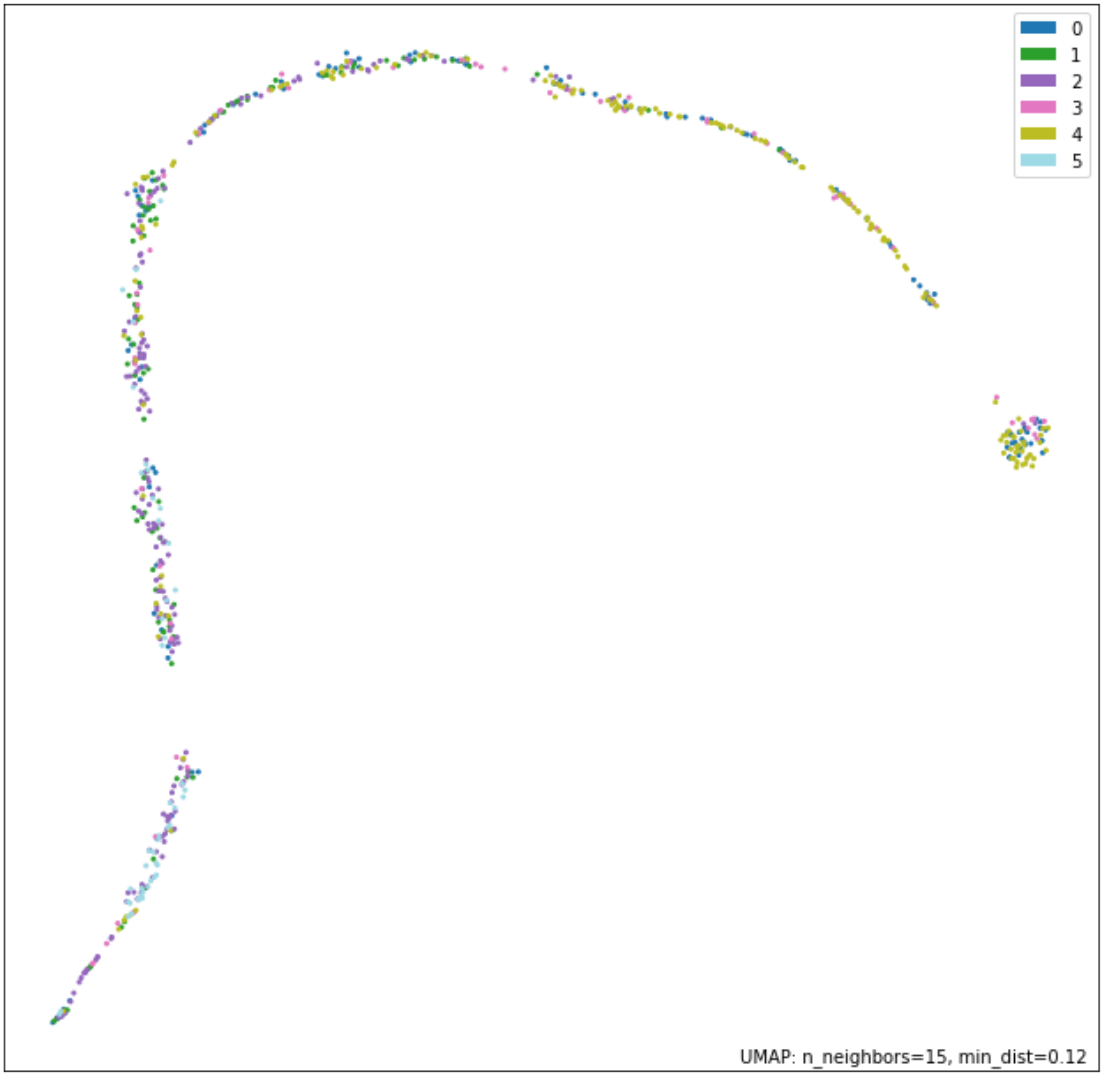

0
1
2
3
4
5
UMAP: n_neighbors=10, min_dist=0.15

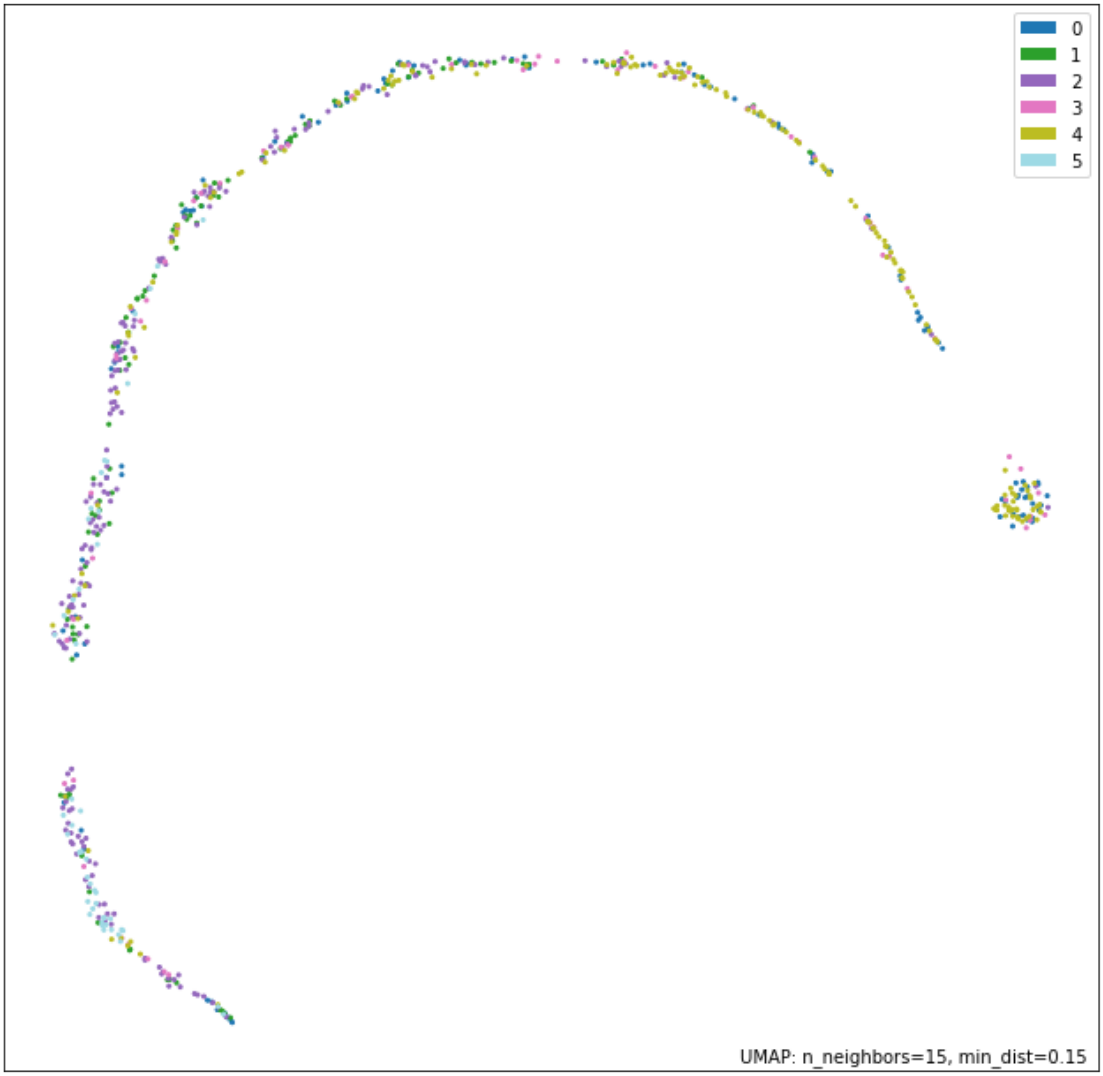

0
1
2
3
4
5
UMAP: n_neighbors=10, min_dist=0.18

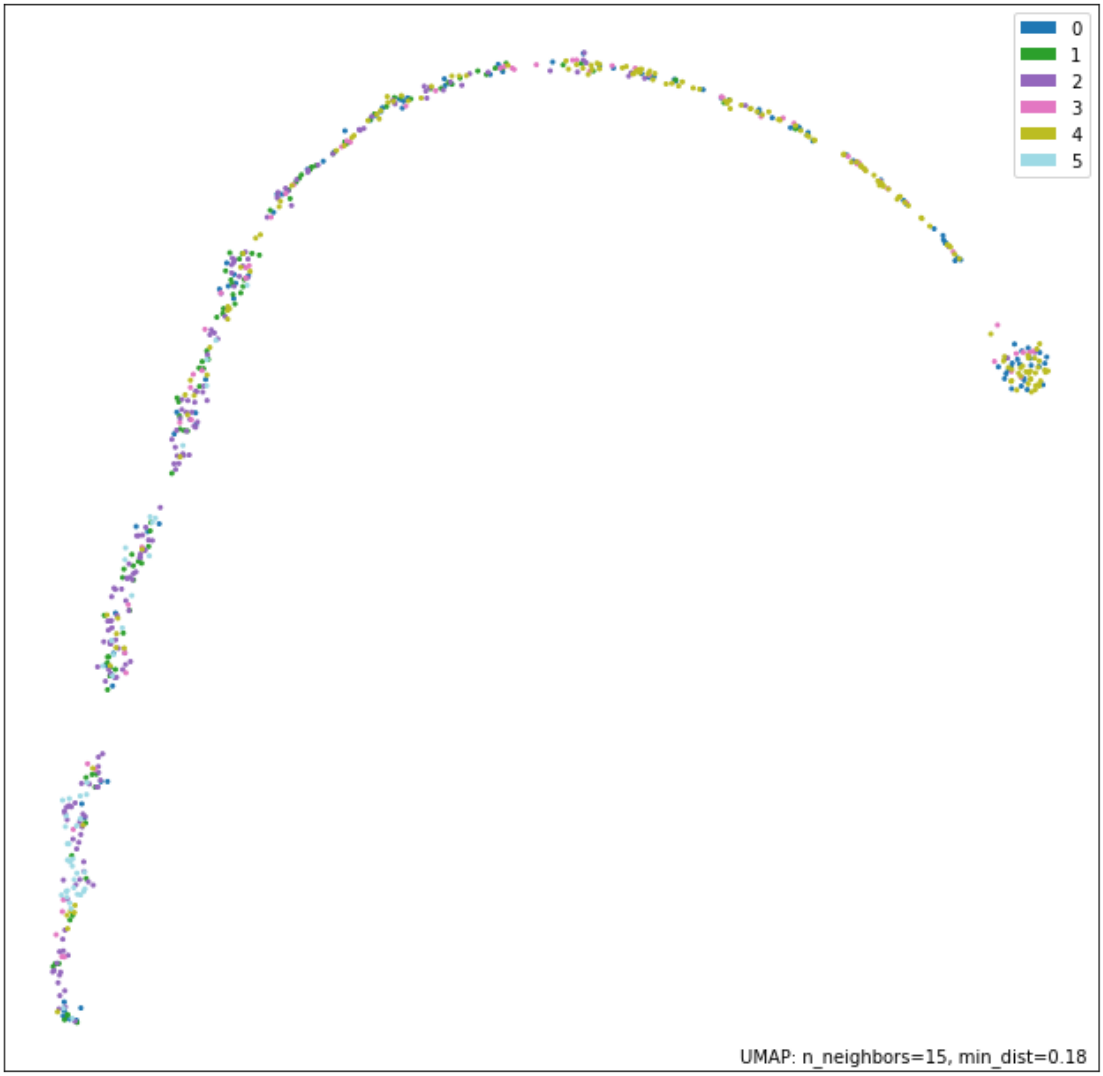

0
1
2
3
4
5
UMAP: n_neighbors=15, min_dist=0.03

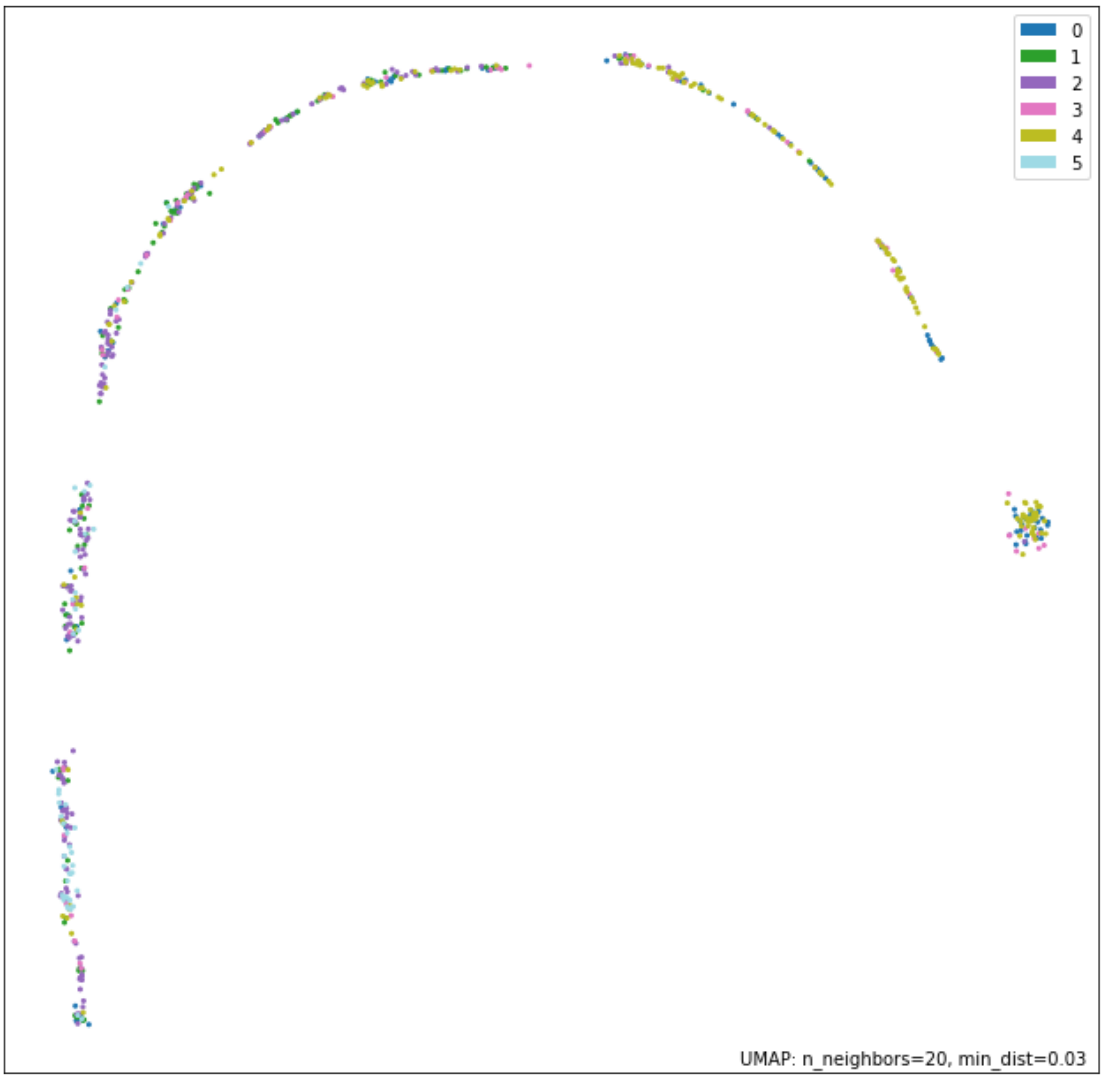

0
1
2
3
4
5
UMAP: n_neighbors=15, min_dist=0.06

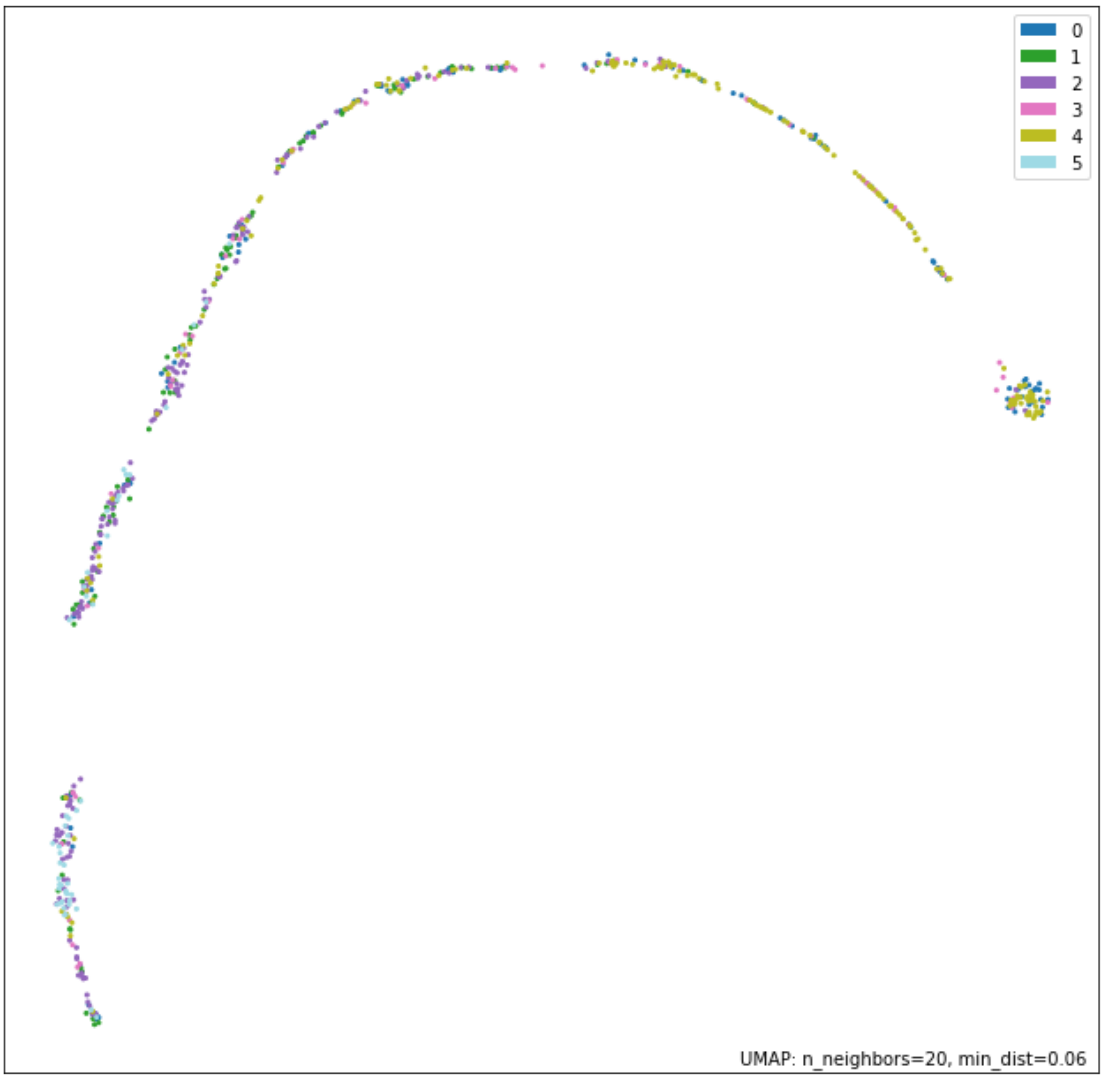

0
1
2
3
4
5
UMAP: n_neighbors=15, min_dist=0.09

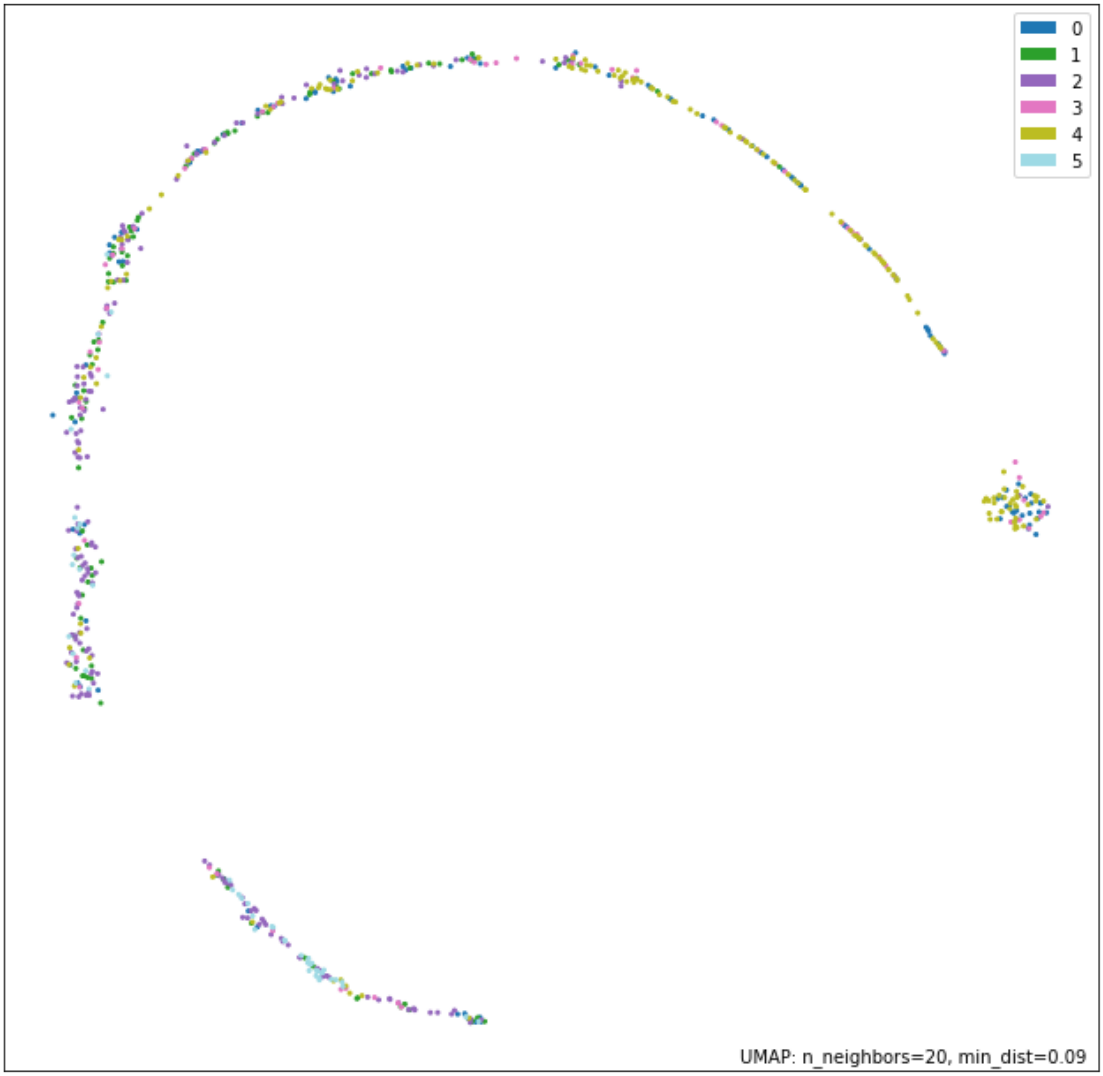

0
1
2
3
4
5
UMAP: n_neighbors=15, min_dist=0.12

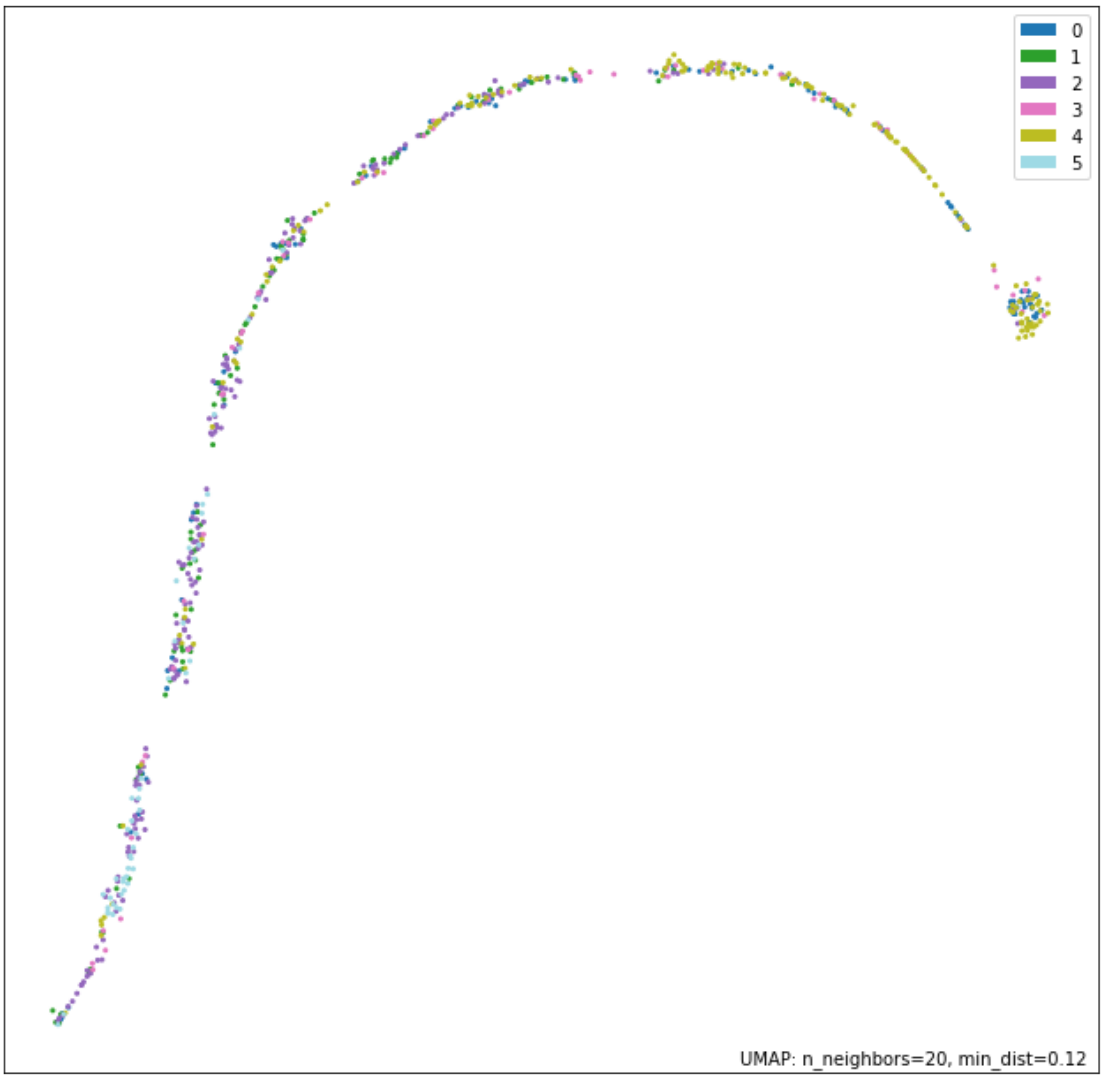

0
1
2
3
4
5
UMAP: n_neighbors=15, min_dist=0.15

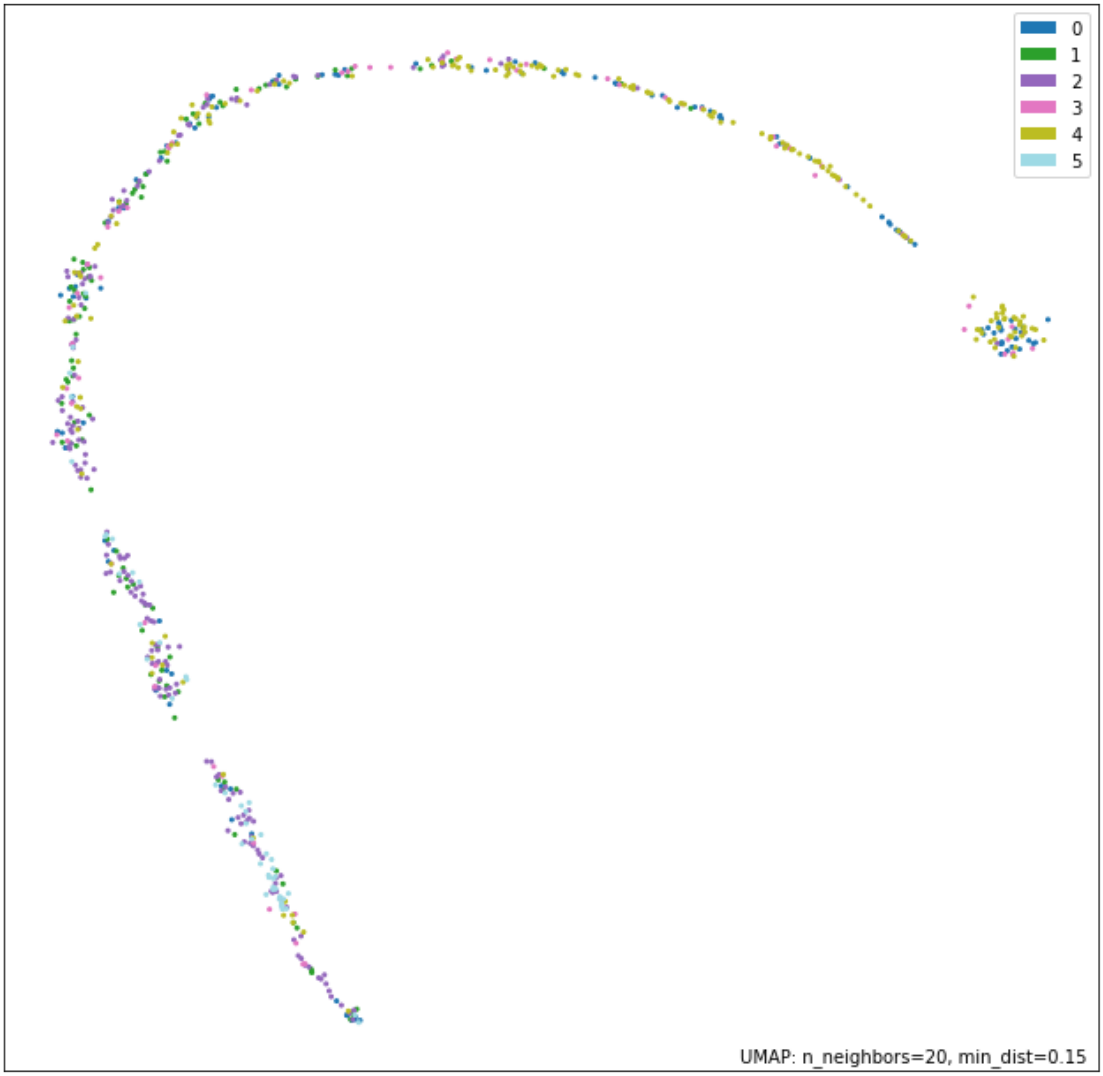

0
1
2
3
4
5
UMAP: n_neighbors=15, min_dist=0.18

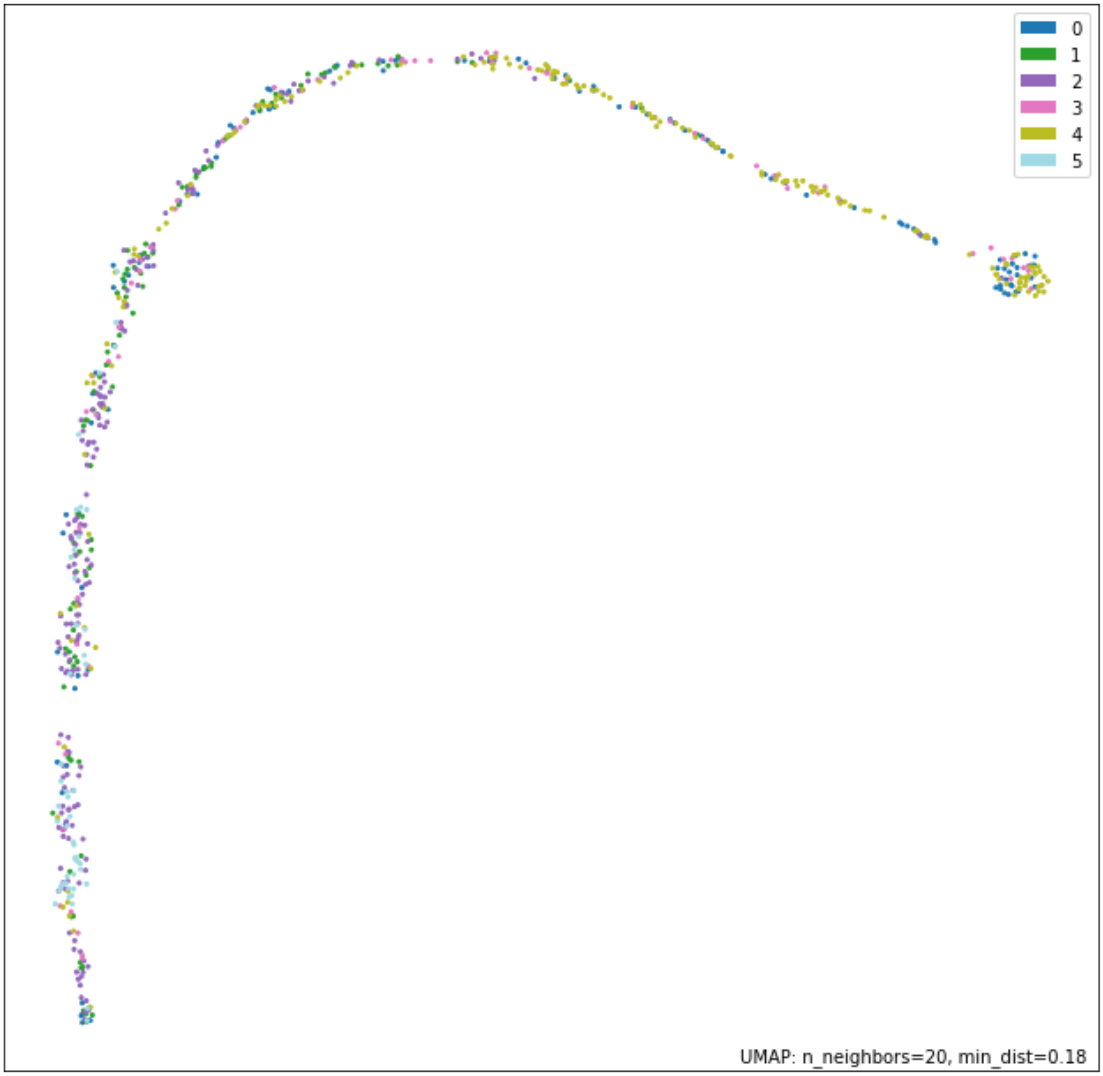

0
1
2
3
4
5
UMAP: n_neighbors=20, min_dist=0.03

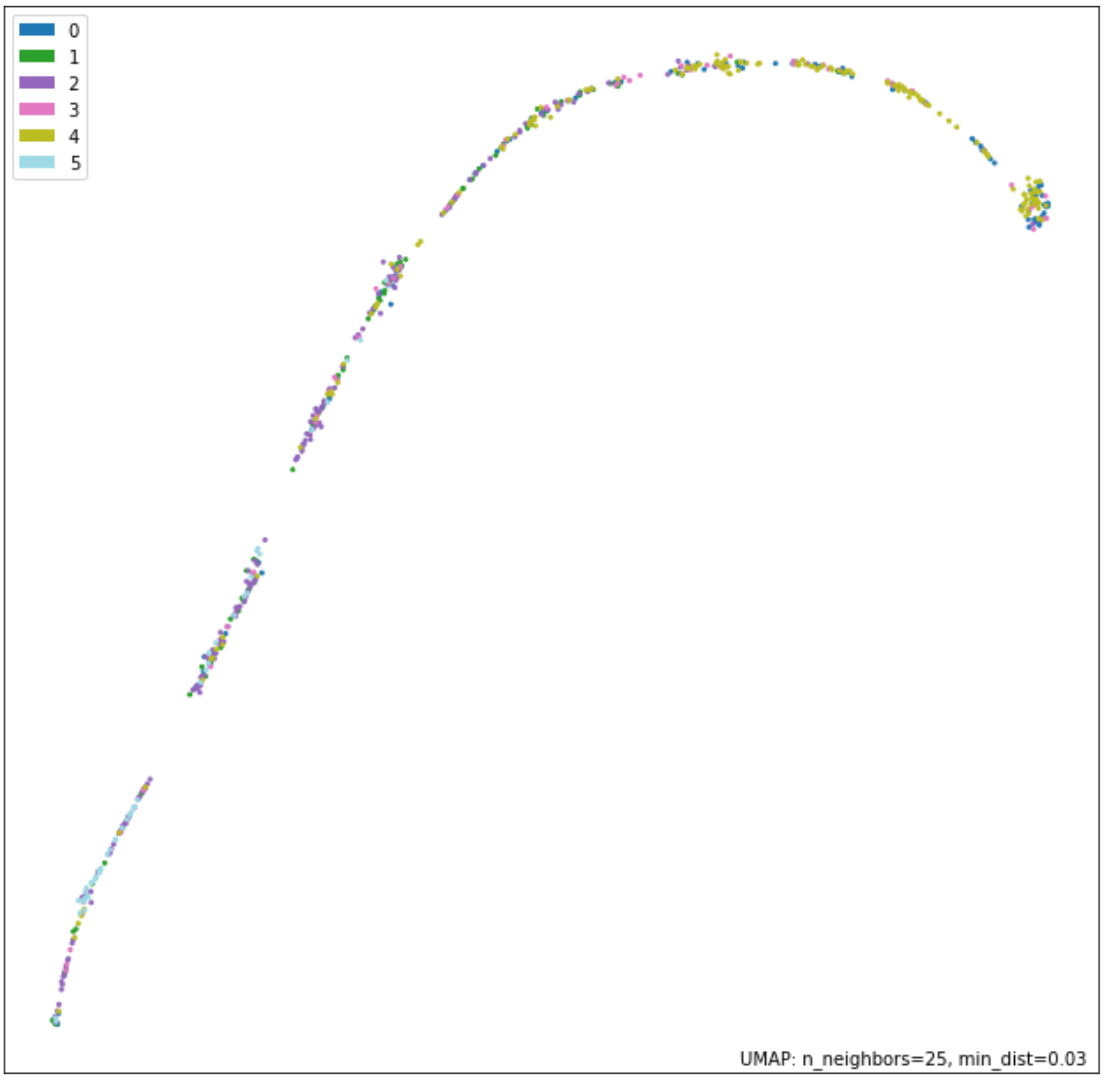

0
1
2
3
4
5
UMAP: n_neighbors=20, min_dist=0.06

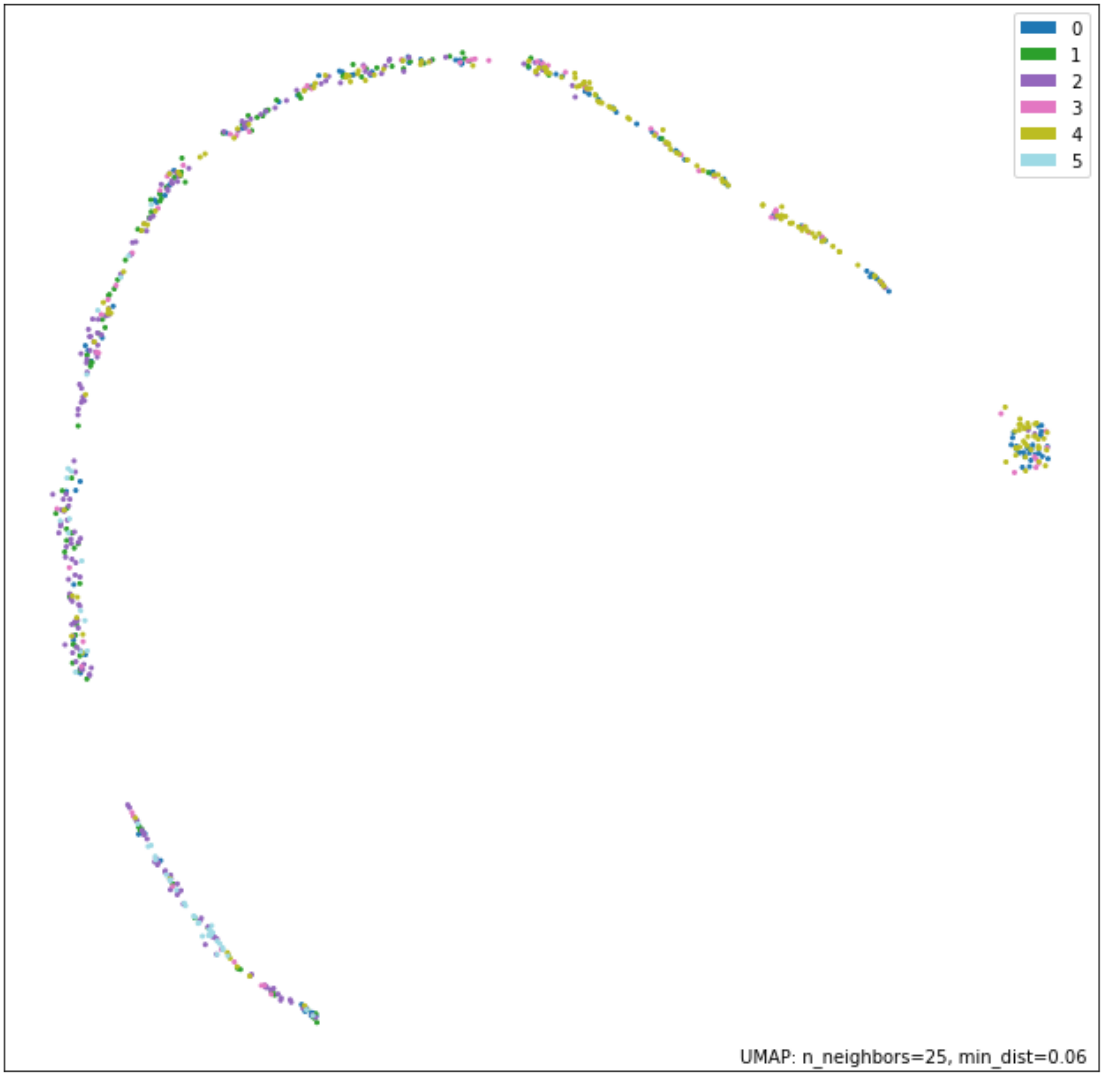

0
1
2
3
4
5
UMAP: n_neighbors=20, min_dist=0.09

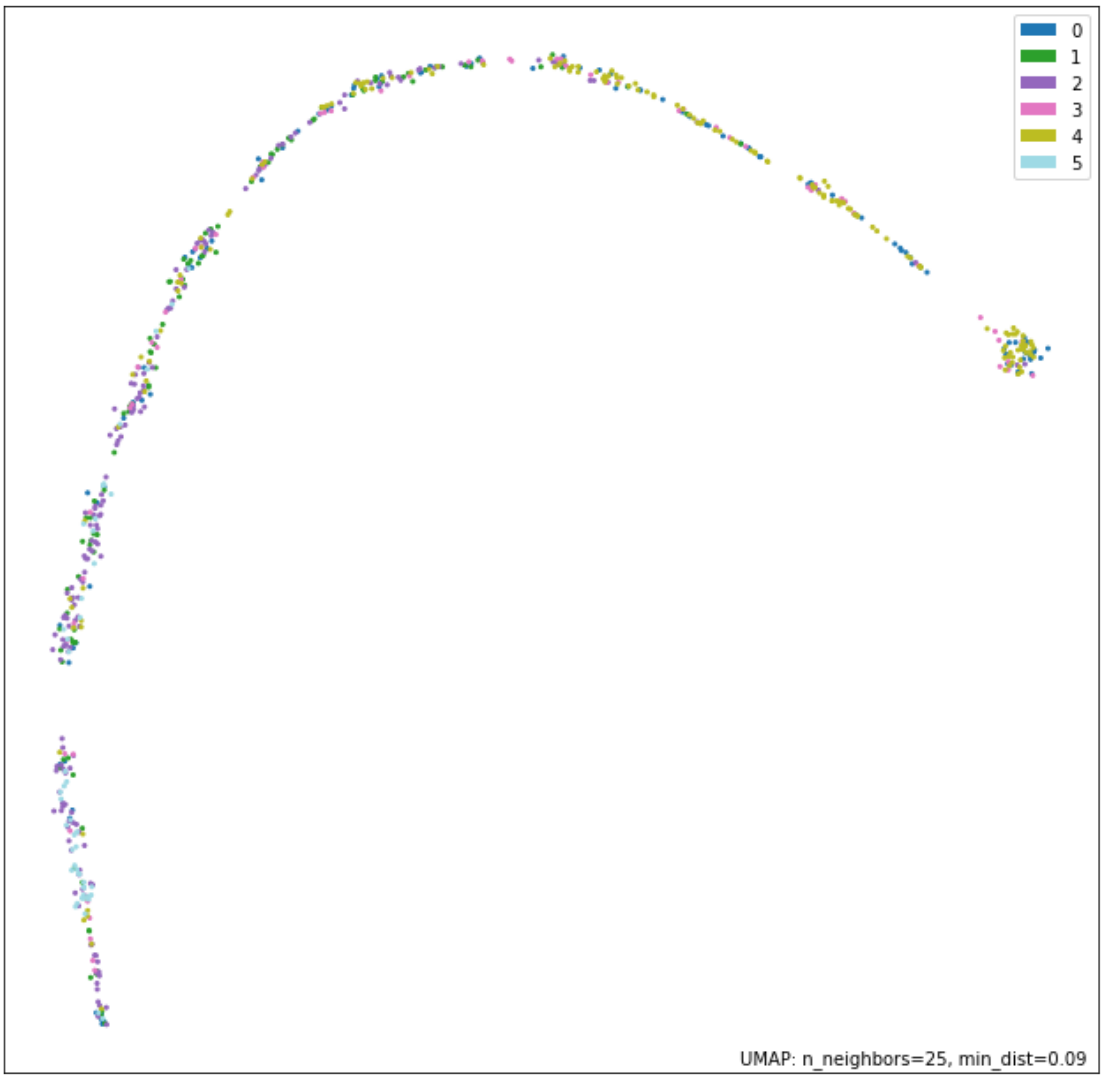

0
1
2
3
4
5
UMAP: n_neighbors=20, min_dist=0.12

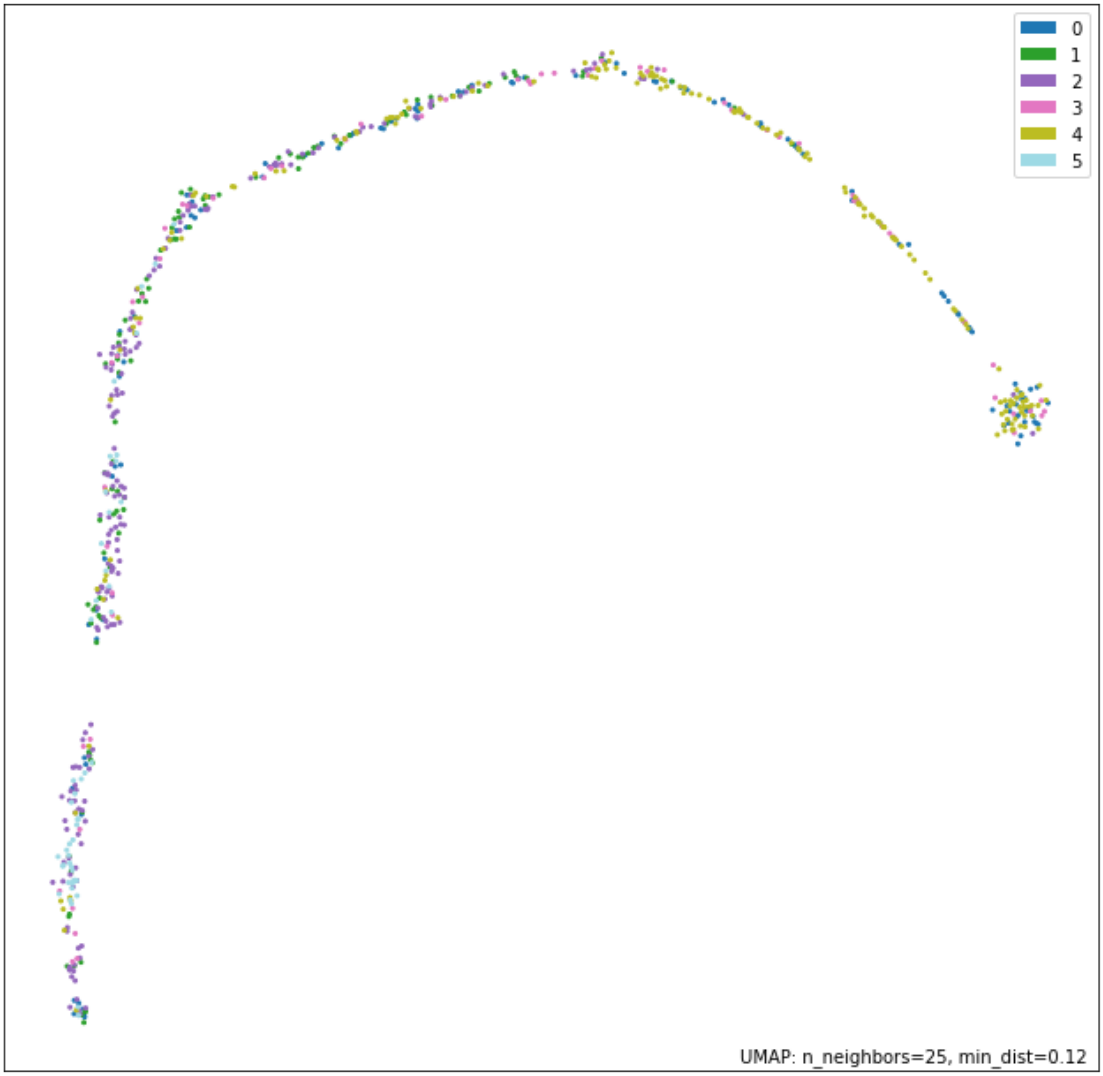

0
1
2
3
4
5
UMAP: n_neighbors=20, min_dist=0.15

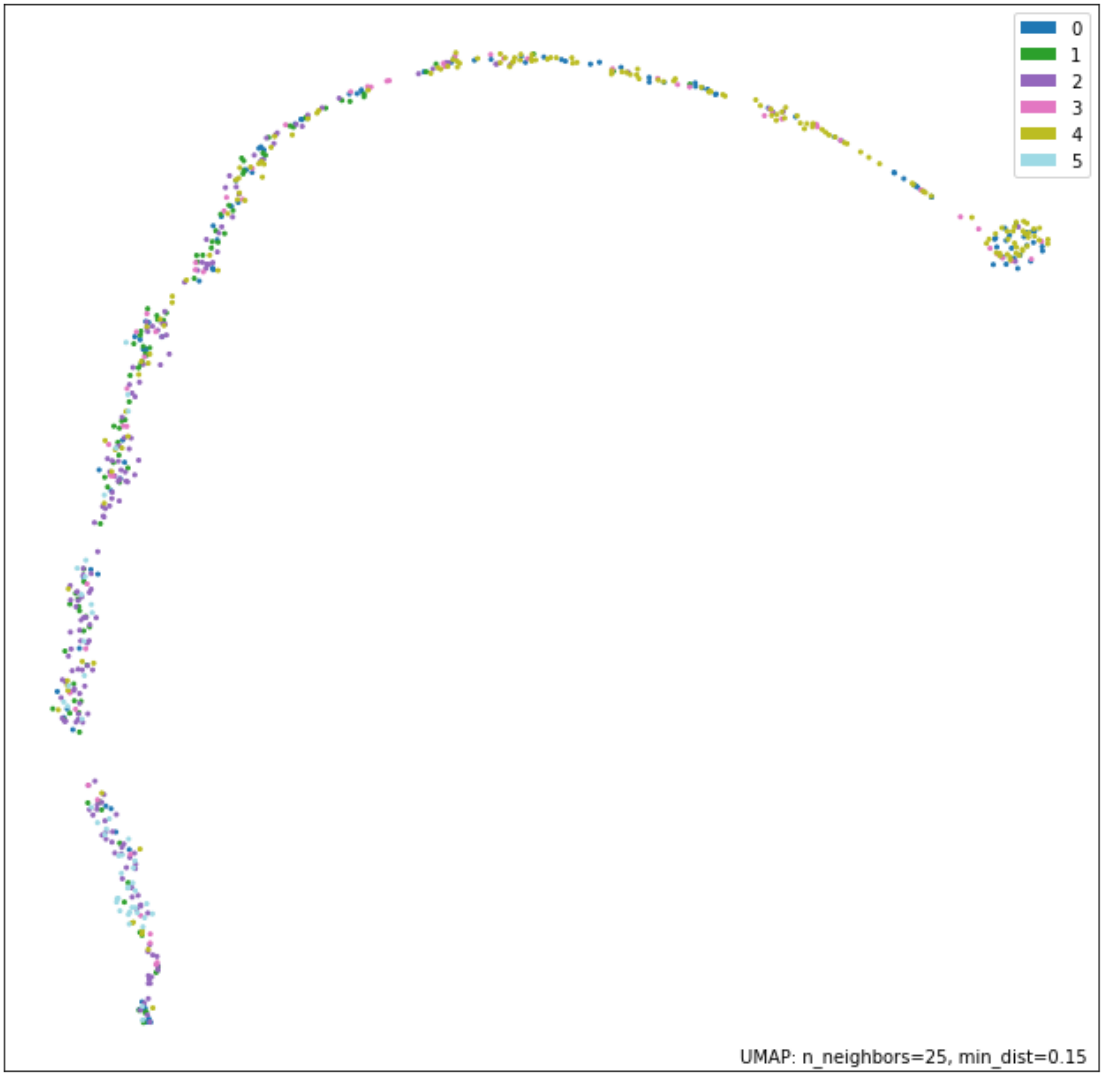

0
1
2
3
4
5
UMAP: n_neighbors=20, min_dist=0.18

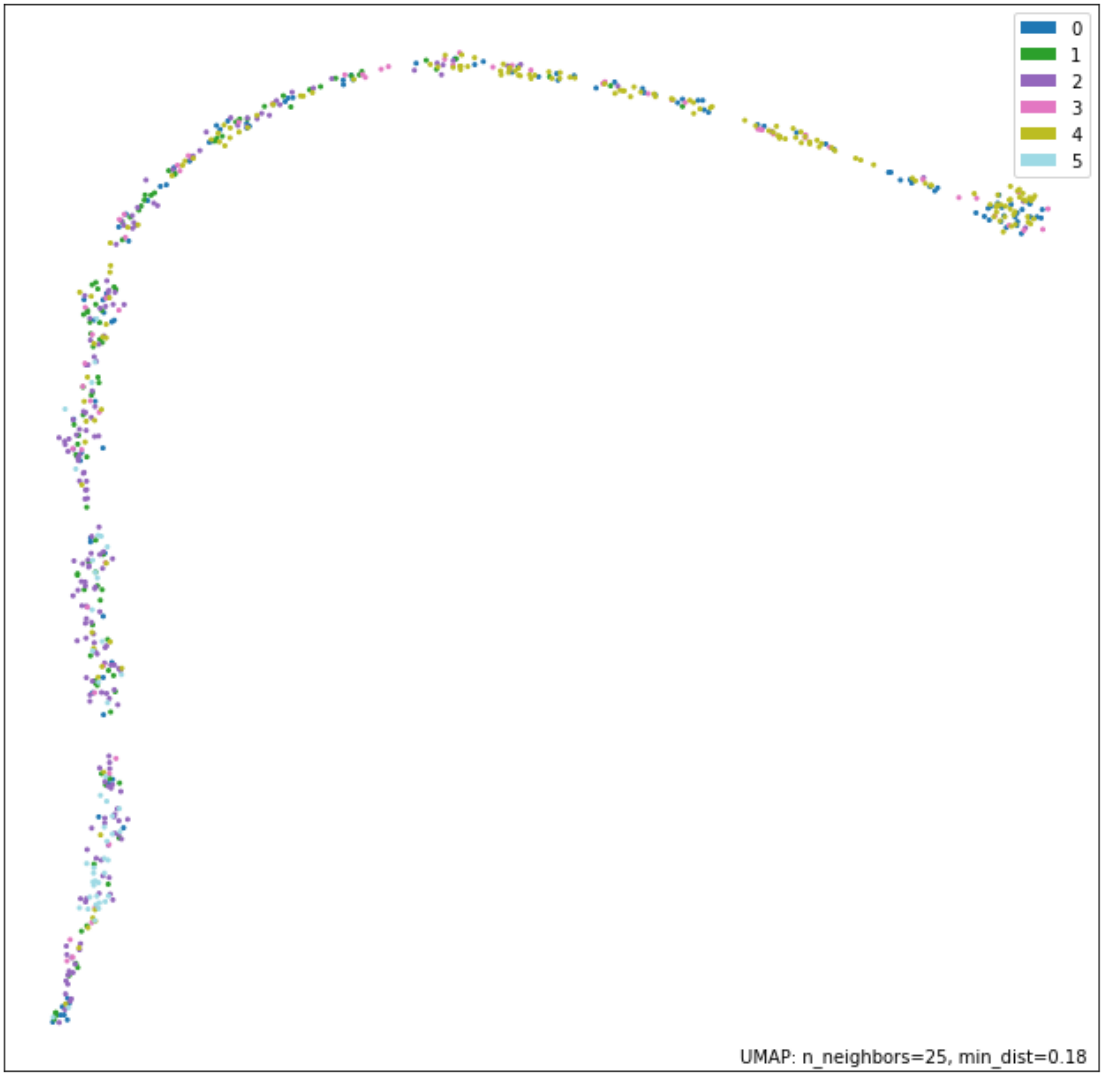

0
1
2
3
4
5
UMAP: n_neighbors=25, min_dist=0.03


0
1
2
3
4
5
UMAP: n_neighbors=25, min_dist=0.06


0
1
2
3
4
5
UMAP: n_neighbors=25, min_dist=0.09


0
1
2
3
4
5
UMAP: n_neighbors=25, min_dist=0.12


0
1
2
3
4
5
UMAP: n_neighbors=25, min_dist=0.15



It's clear that using average is not enough for plotting the high-dimensional tensor, but it is possible for splitting it into slices and plot all of them. And thus you can find out the better filter according to the plots.

[26]:
```
predicted_Probability_before_pool.shape
```

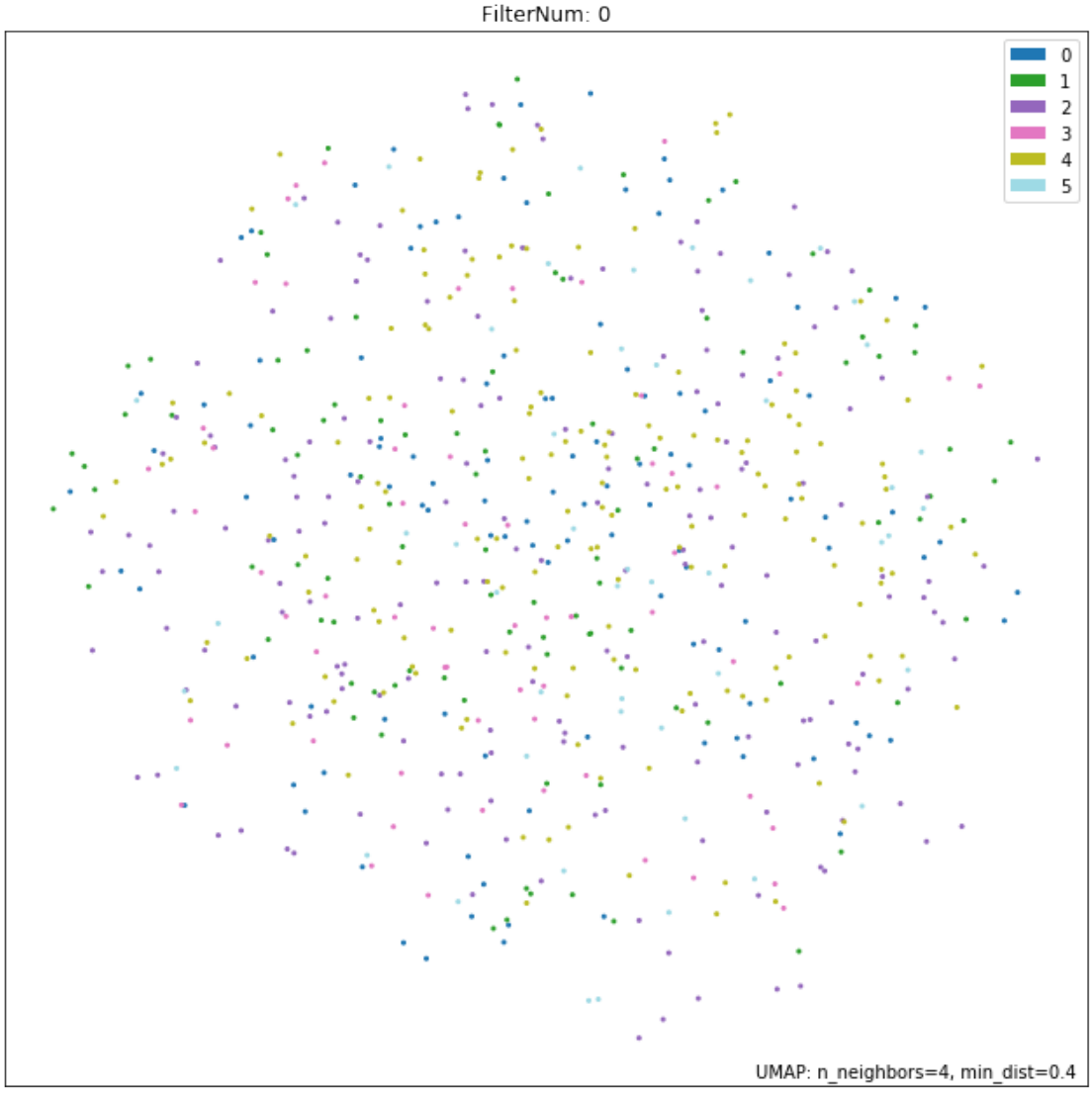
[26]:
```
(667, 1996, 250)
```

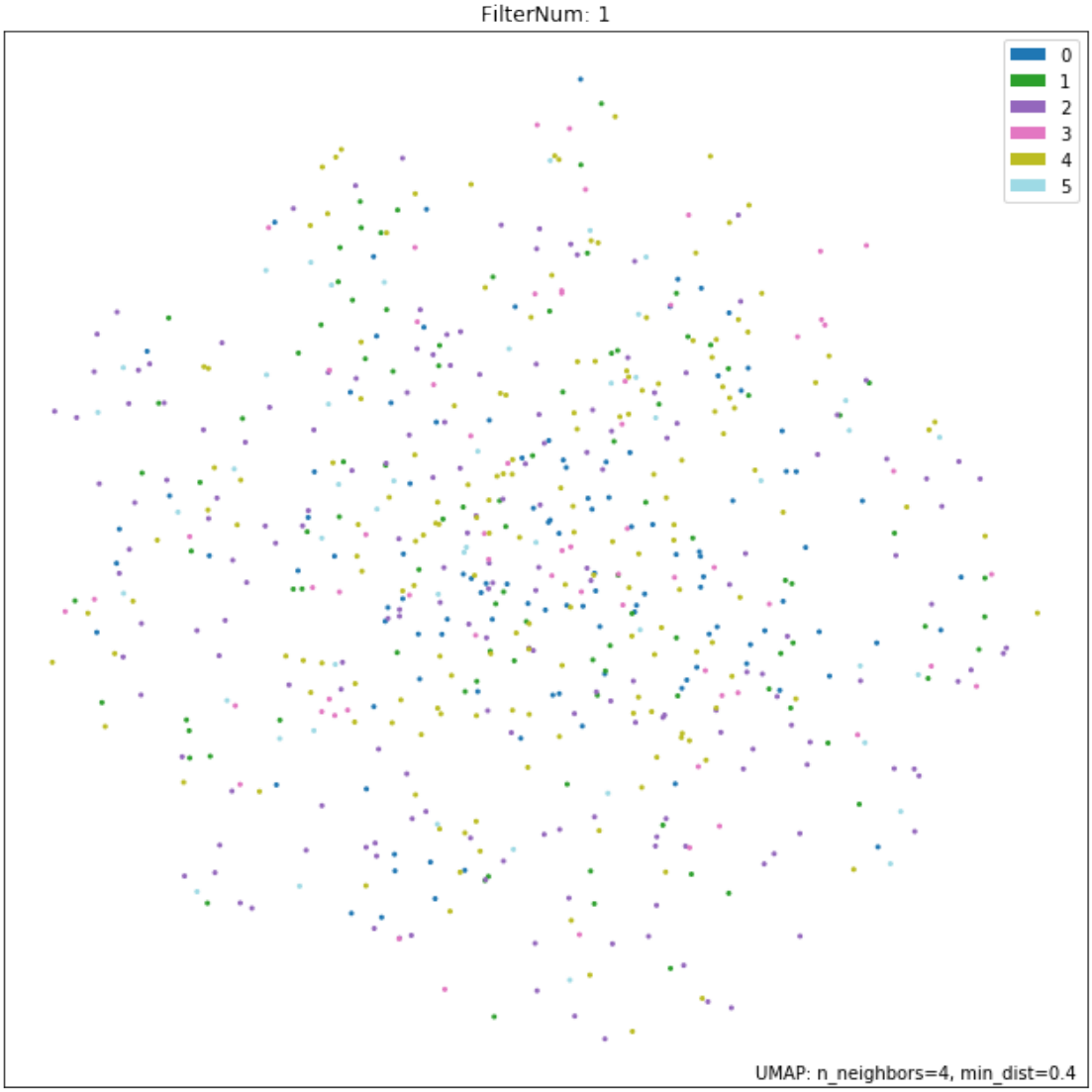
Here the last dim '250' represent the number of filters, we can extract the slices by splitting the filters.

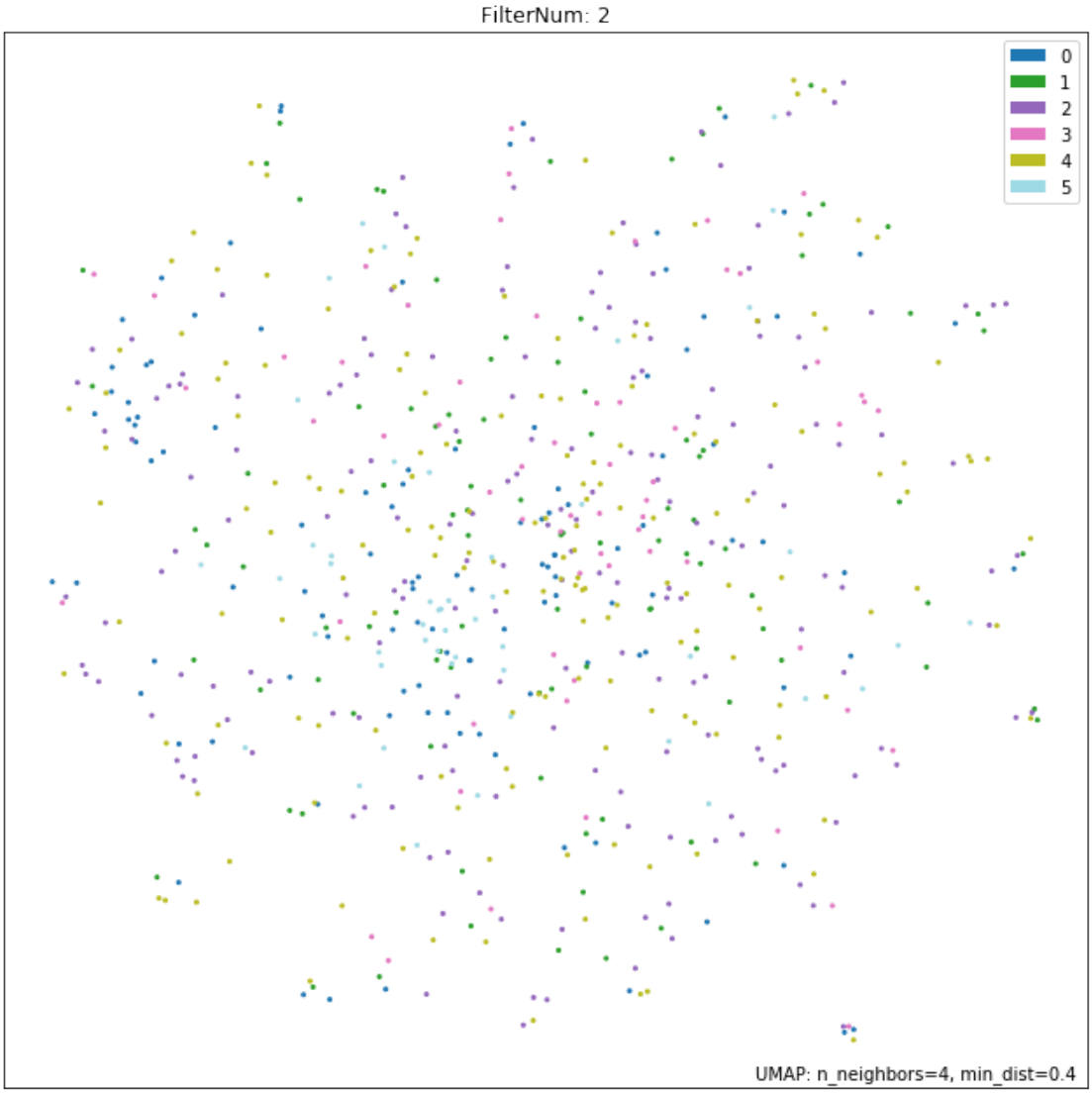
Considering the length of the notebook, we only extract the first 10 plots.

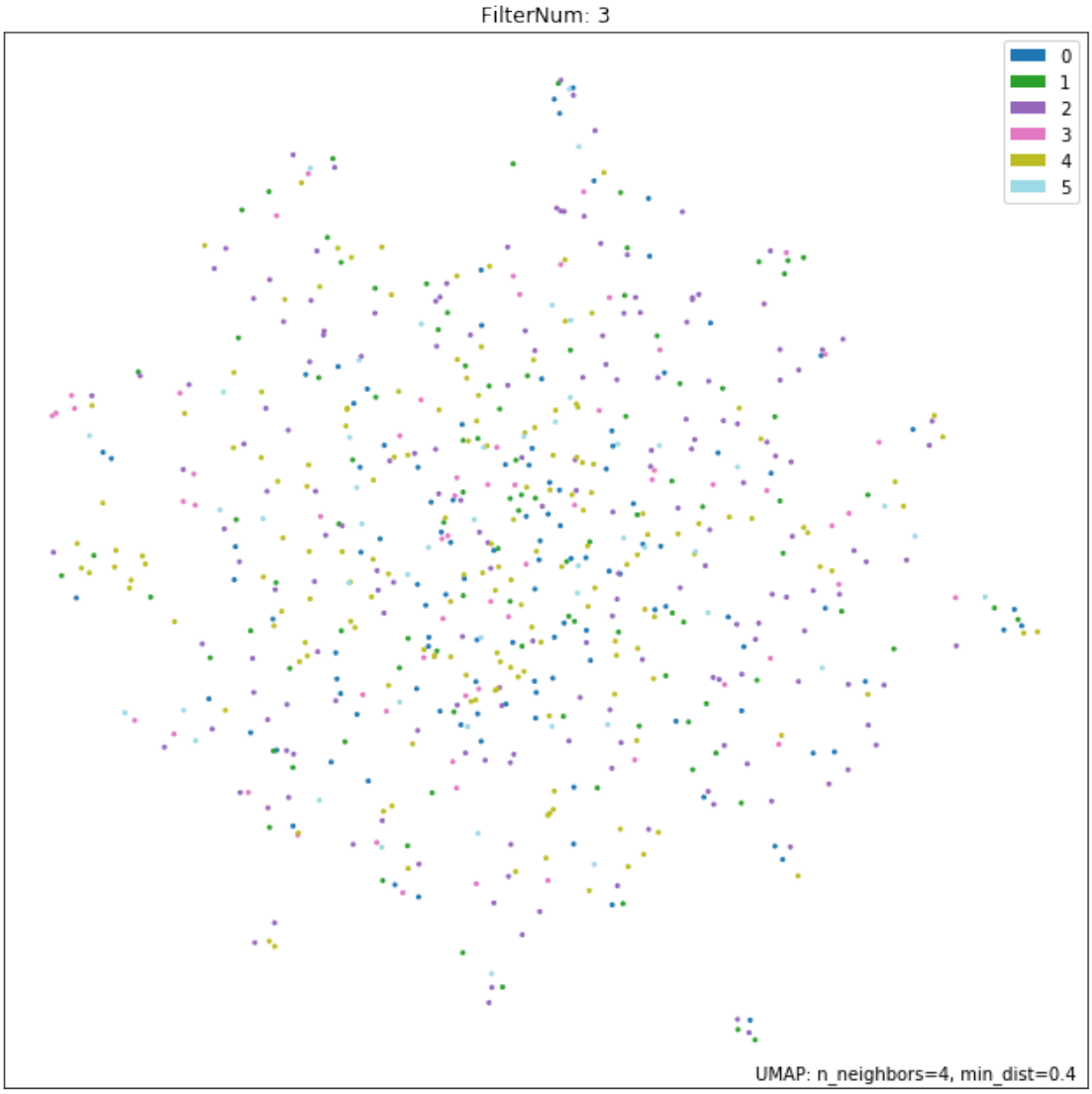
[27]:
```
# the featureDict could contain more parameters, use '?umap.UMAP' for more␣
↪details.
```

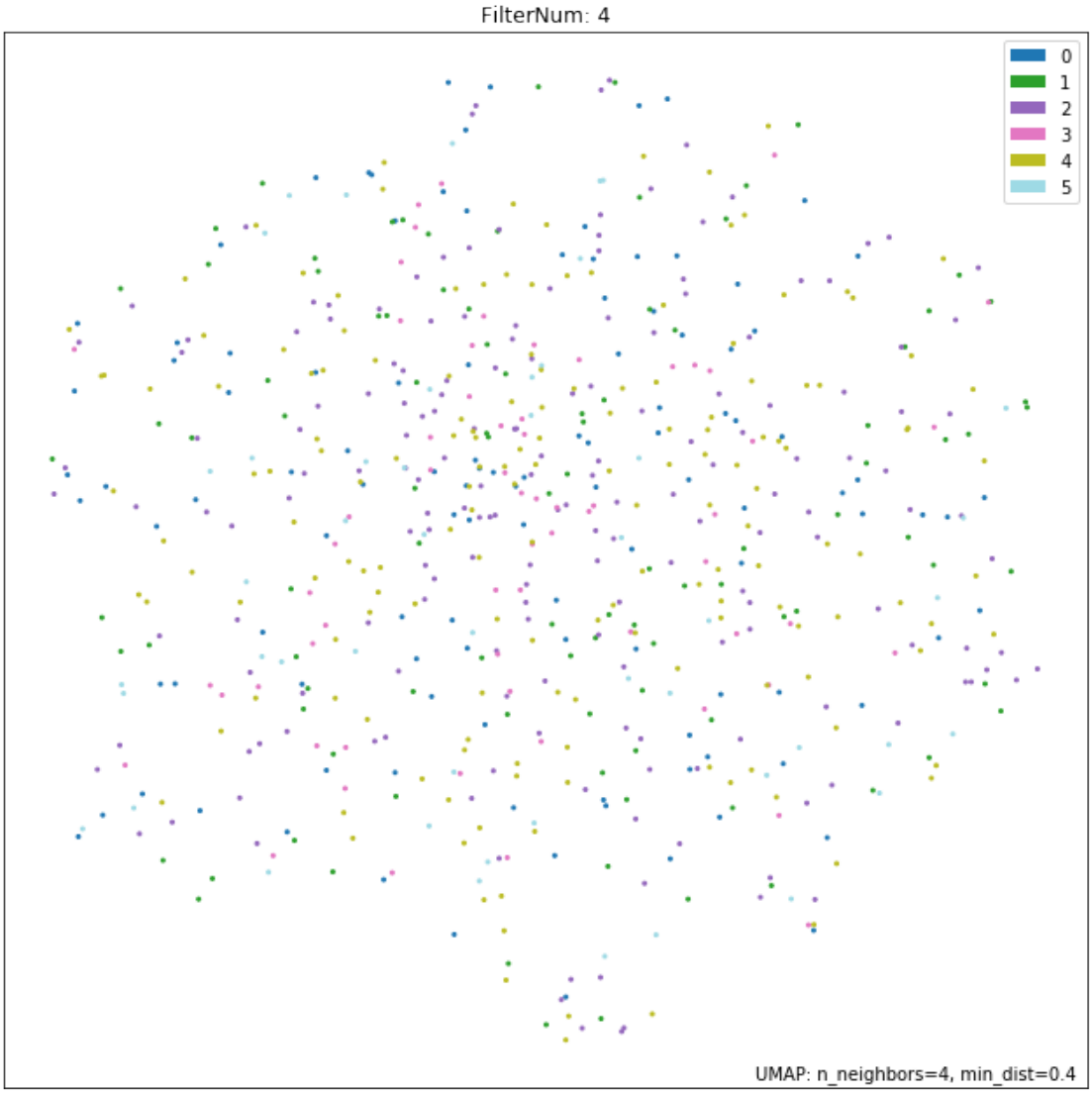
```
featureDict={
        'n_neighbors' : 4,
        'min_dist' : 0.4,
        'metric' : 'euclidean',
        }

for filterDim in range(predicted_Probability_before_pool.shape[-1]):
    subMat = predicted_Probability_before_pool[:,:,filterDim]
    mapper = umap.UMAP(**featureDict).fit(subMat)
    plotObj = umap.plot.points(mapper, labels=testLabelArr,  theme='blue')
    plotObj.set_title('FilterNum: %d' %filterDim)
    plt.show()
    if filterDim >= 10:
        break
```

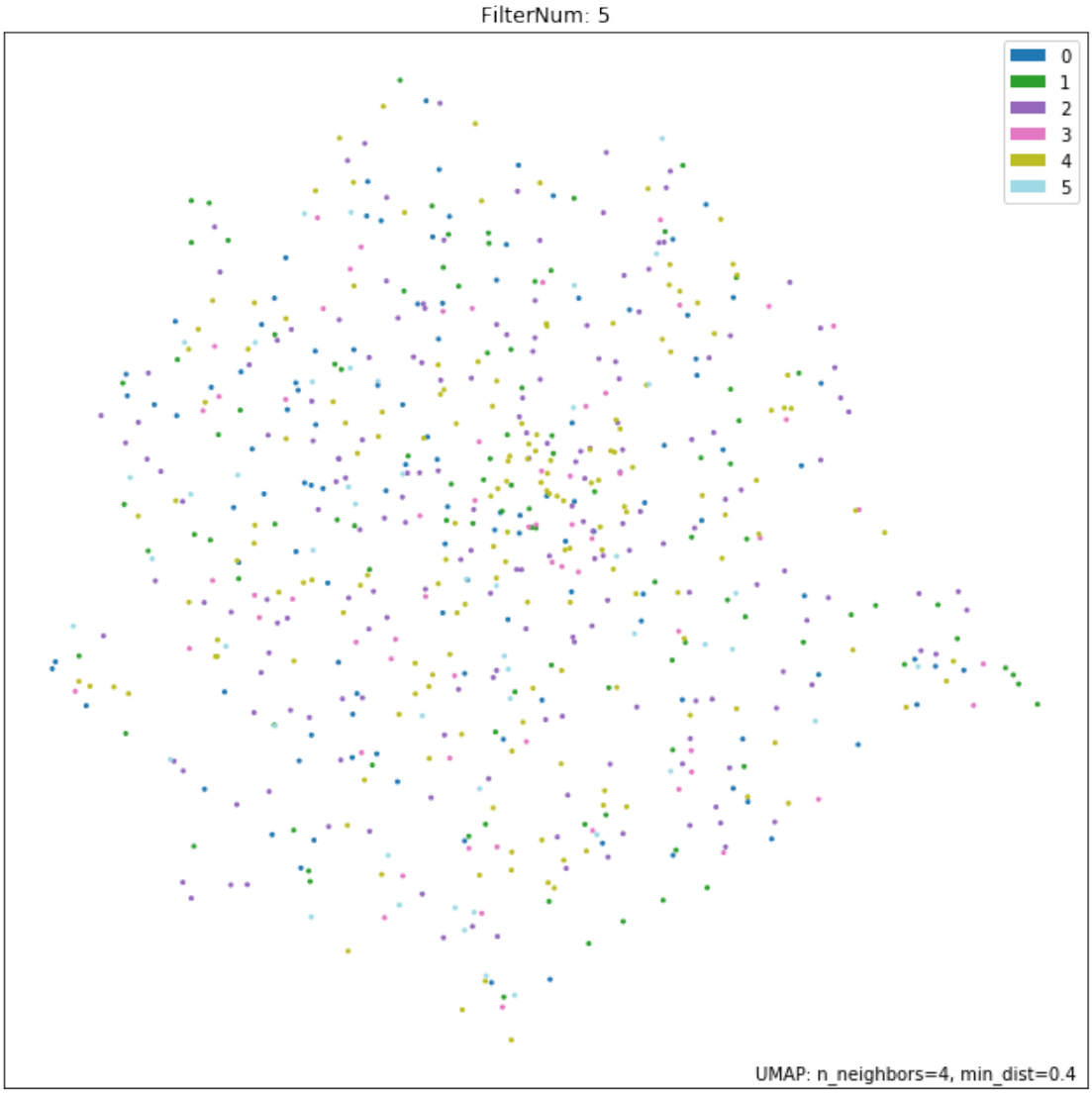


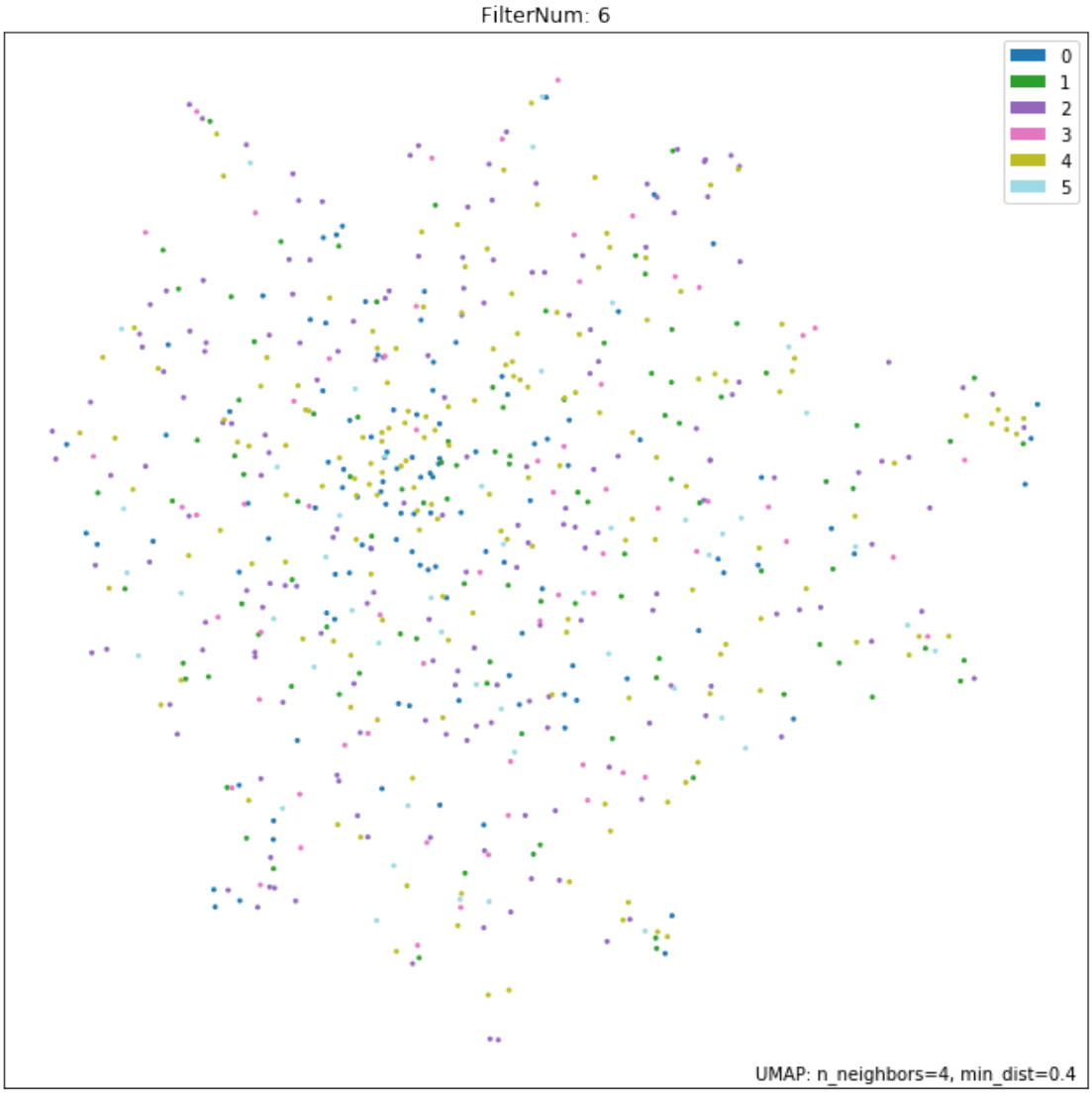

FilterNum: 1
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4

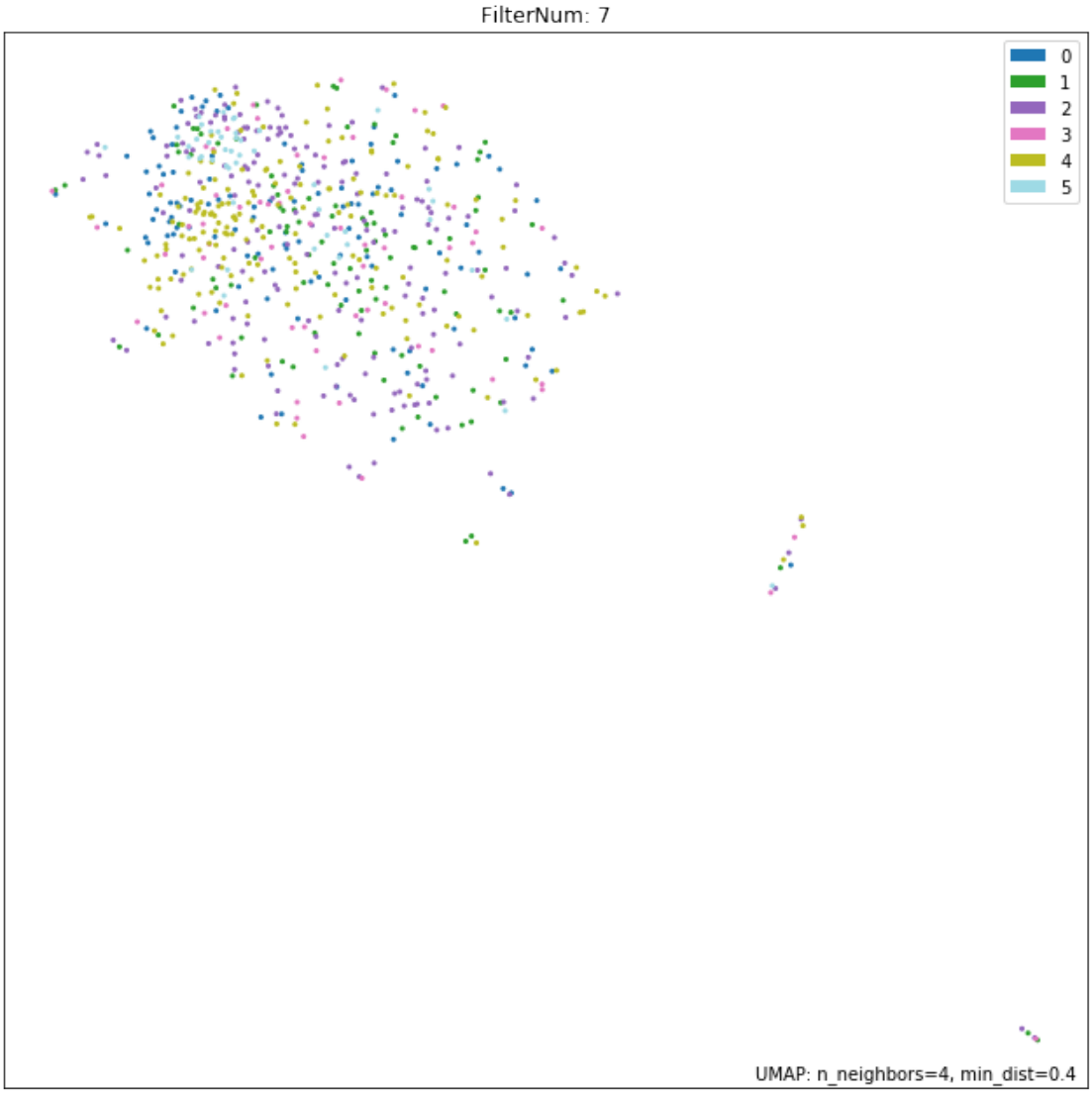

FilterNum: 2
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4

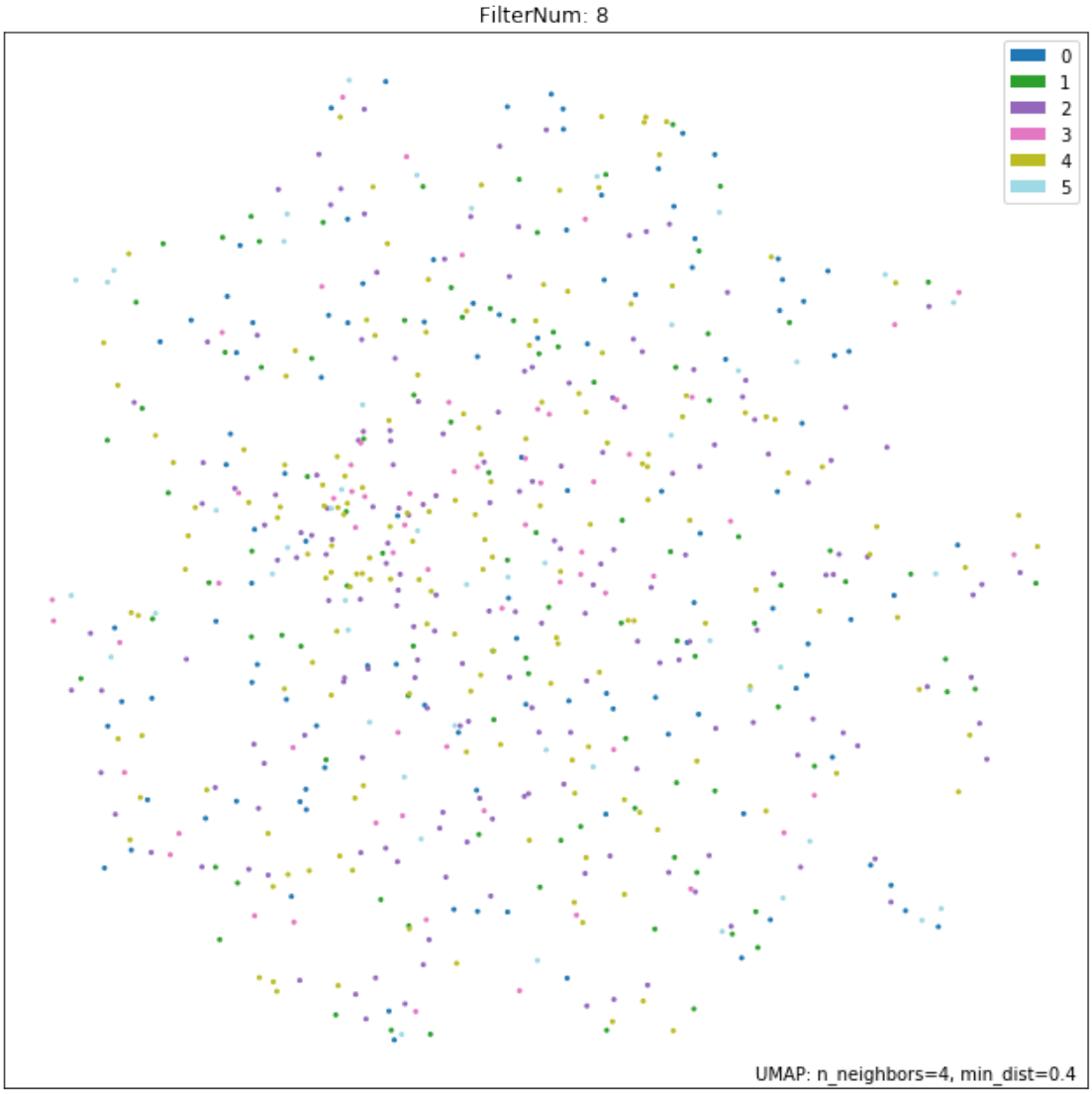

FilterNum: 3
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4

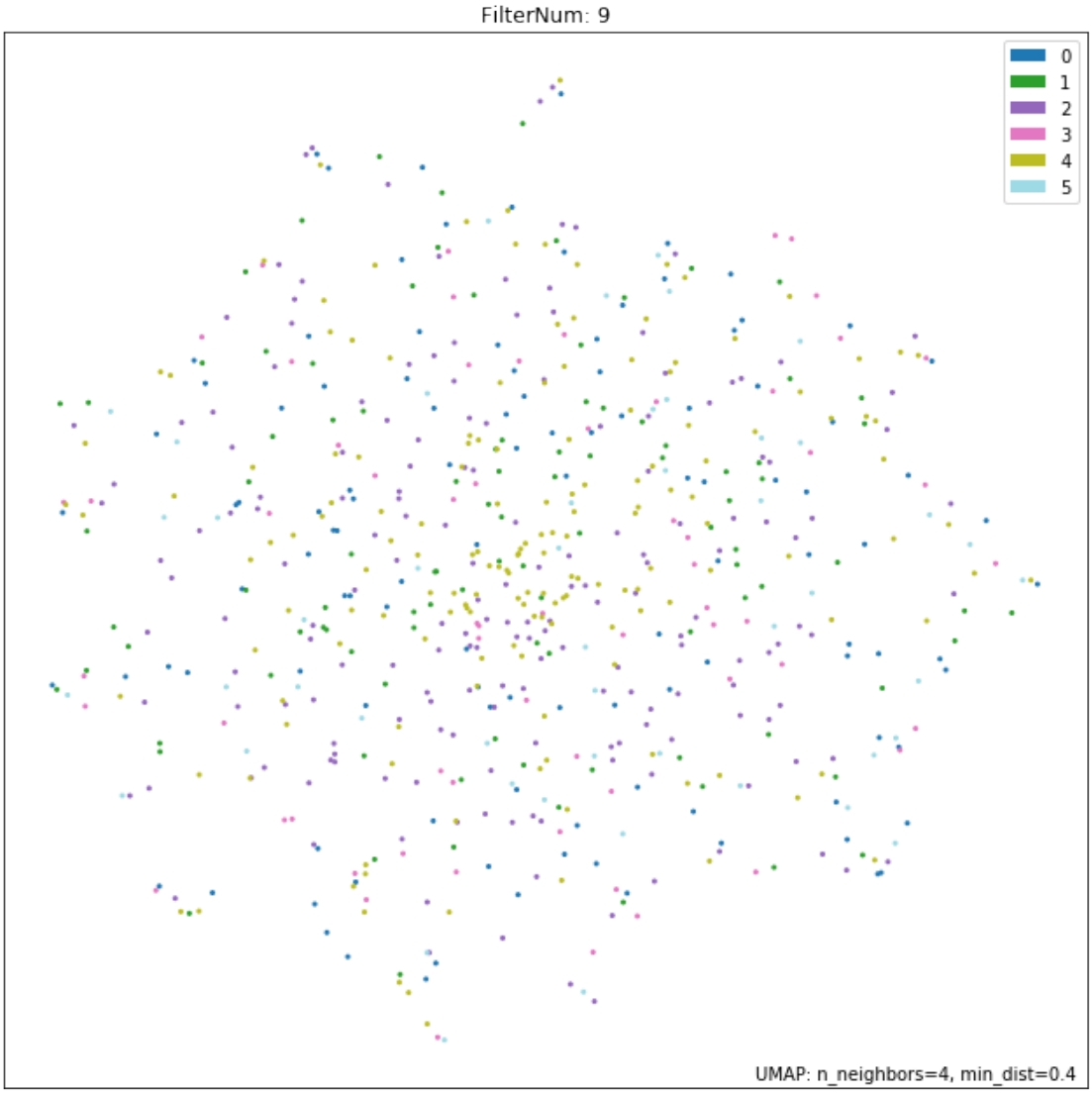

FilterNum: 4
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4

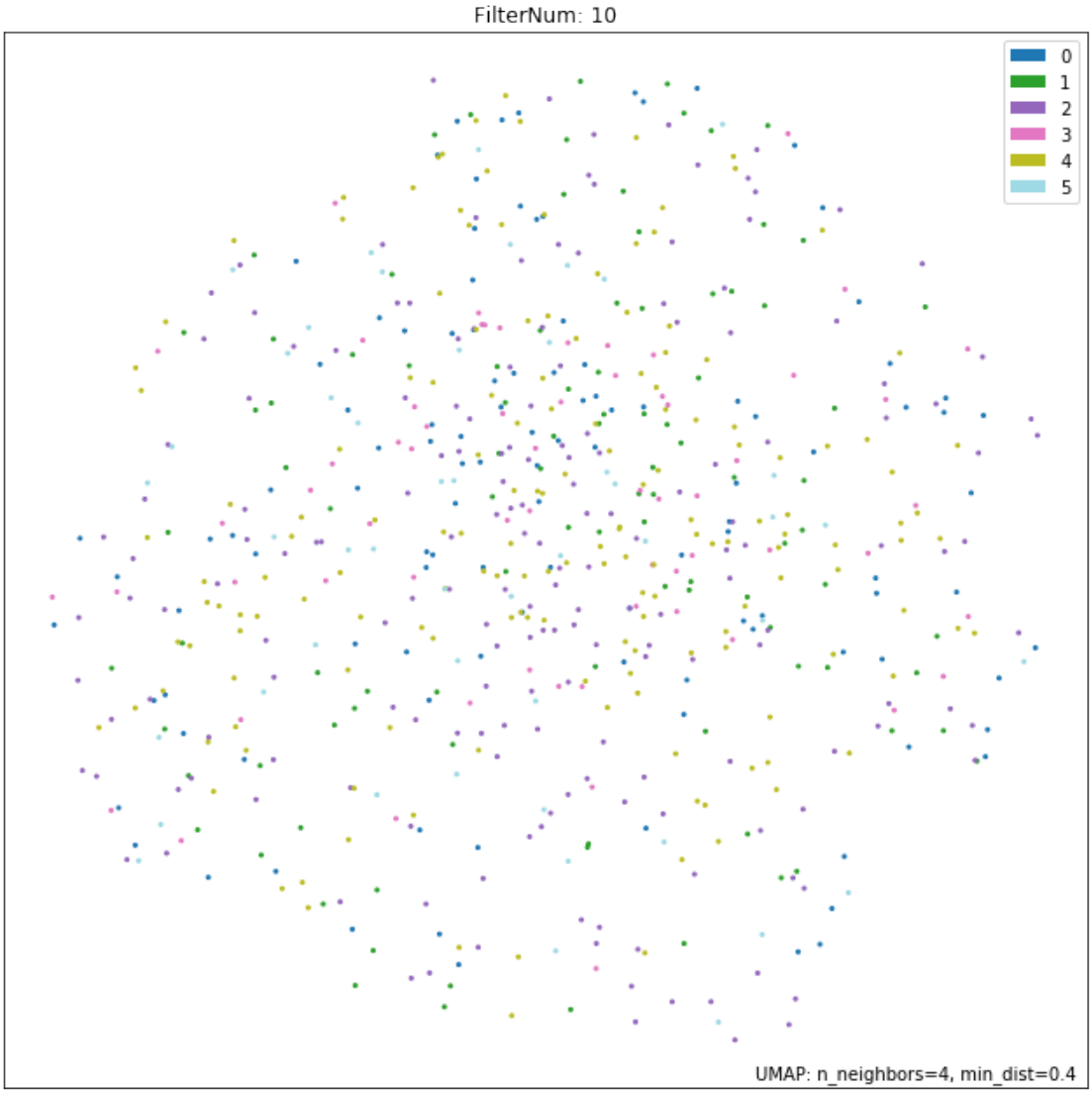

FilterNum: 5
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4


FilterNum: 6
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4


FilterNum: 7
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4


FilterNum: 8
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4


FilterNum: 9
0
1
2
3
4
5
UMAP: n_neighbors=4, min_dist=0.4



That's all the informations for plotting the layer of multi-label. If you still have more issues, please fell free to connect us at ljs@swmu.edu.cn, thanks.