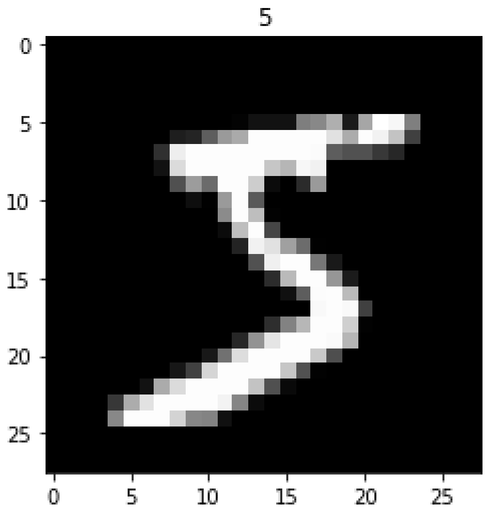
In [4]:

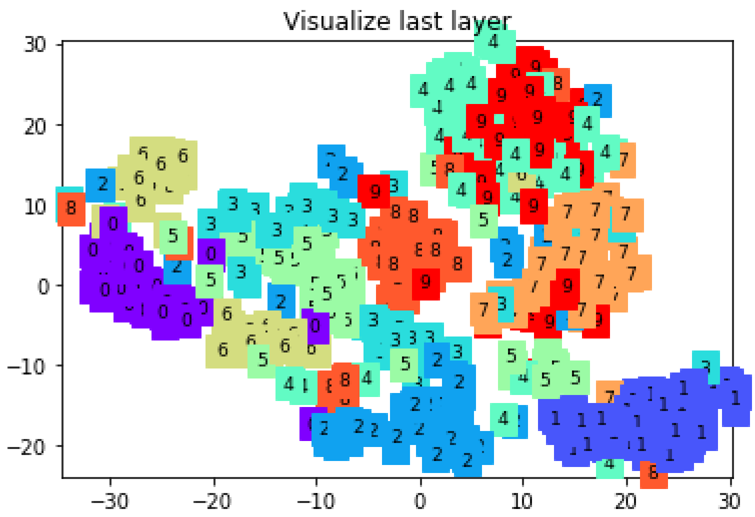
```
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.utils.data as Data
import torchvision
import matplotlib.pyplot as plt
#matplotlib inline

torch.manual_seed(1)    # reproducible
# Hyper Parameters
EPOCH = 1               # train the training data n times, to save time, we just tra
BATCH_SIZE = 50
LR = 0.001              # learning rate
DOWNLOAD_MNIST = False   # set to False if you have downloaded

# Mnist digits dataset
train_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=True,                                     # this is training data
    transform=torchvision.transforms.ToTensor(),    # Converts a PIL.Image or numpy.
                                                    # torch.FloatTensor of shape (C
    download=False,                         # download it if you don't have it
)
# plot one example
print(train_data.train_data.size())                 # (60000, 28, 28)
print(train_data.train_labels.size())               # (60000)
plt.imshow(train_data.train_data[0].numpy(), cmap='gray') # 행과 열을 가진 행렬 형태의 2차
plt.title('%i' % train_data.train_labels[0])
plt.show()

# Data Loader for easy mini-batch return in training, the image batch shape will be
train_loader = Data.DataLoader(dataset=train_data, batch_size=BATCH_SIZE, shuffle=Tr

# convert test data into Variable, pick 2000 samples to speed up testing
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)
print(test_data.test_data[0].size())
test_x = Variable(torch.unsqueeze(test_data.test_data, dim=1)).type(torch.FloatTenso
print(test_x[0].size())
# shape from (2000, 28, 28) to (2000, 1, 28, 28), value in range(0,1)
test_y = test_data.test_labels[:2000]
print(test_y[0])

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(         # input shape (1, 28, 28)
            nn.Conv2d(
                in_channels=1,              # input height
                out_channels=16,            # n_filters
                kernel_size=5,              # filter size
                stride=1,                   # filter movement/step
                padding=2,                  # if want same width and length of this
            ),                              # output shape (16, 28, 28)
            nn.ReLU(),                      # activation
            nn.MaxPool2d(kernel_size=2),    # choose max value in 2x2 area, output s
        )
        self.conv2 = nn.Sequential(         # input shape (1, 28, 28)
            nn.Conv2d(16, 32, 5, 1, 2),     # output shape (32, 14, 14)
            nn.ReLU(),                      # activation
```

```
            nn.MaxPool2d(2),                # output shape (32, 7, 7)
        )
        self.out = nn.Linear(32 * 7 * 7, 10)   # fully connected layer, output 10 cl

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten the output of conv2 to (batch_
        output = self.out(x)
        return output, x    # return x for visualization

cnn = CNN()
print(cnn)  # net architecture

optimizer = torch.optim.Adam(cnn.parameters(), lr=LR)   # optimize all cnn parameter
loss_func = nn.CrossEntropyLoss()                       # the target label is not on

# following function (plot_with_labels) is for visualization, can be ignored if not
from matplotlib import cm
try: from sklearn.manifold import TSNE; HAS_SK = True
except: HAS_SK = False; print('Please install sklearn for layer visualization')
def plot_with_labels(lowDWeights, labels):
    plt.cla()
    X, Y = lowDWeights[:, 0], lowDWeights[:, 1]
    for x, y, s in zip(X, Y, labels):
        c = cm.rainbow(int(255 * s / 9)); plt.text(x, y, s, backgroundcolor=c, fonts
    plt.xlim(X.min(), X.max()); plt.ylim(Y.min(), Y.max()); plt.title('Visualize las

plt.ion()
# training and testing
for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader):   # gives batch data, normalize x w
        b_x = Variable(x)   # batch x
        b_y = Variable(y)   # batch y

        output = cnn(b_x)[0]               # cnn output
        loss = loss_func(output, b_y)   # cross entropy loss
        optimizer.zero_grad()           # clear gradients for this training step
        loss.backward()                 # backpropagation, compute gradients
        optimizer.step()                # apply gradients

        if step % 100 == 0:
            test_output, last_layer = cnn(test_x)
            pred_y = torch.max(test_output, 1)[1].data.squeeze()
            accuracy = (pred_y == test_y).sum().item() / float(test_y.size(0))
            print('Epoch: ', epoch, '| train loss: %.4f' % loss.data, '| test accura
            if HAS_SK:
                # Visualization of trained flatten layer (T-SNE)
                tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=5000)
                plot_only = 500
                low_dim_embs = tsne.fit_transform(last_layer.data.numpy()[:plot_only
                labels = test_y.numpy()[:plot_only]
                plot_with_labels(low_dim_embs, labels)
plt.ioff()
```

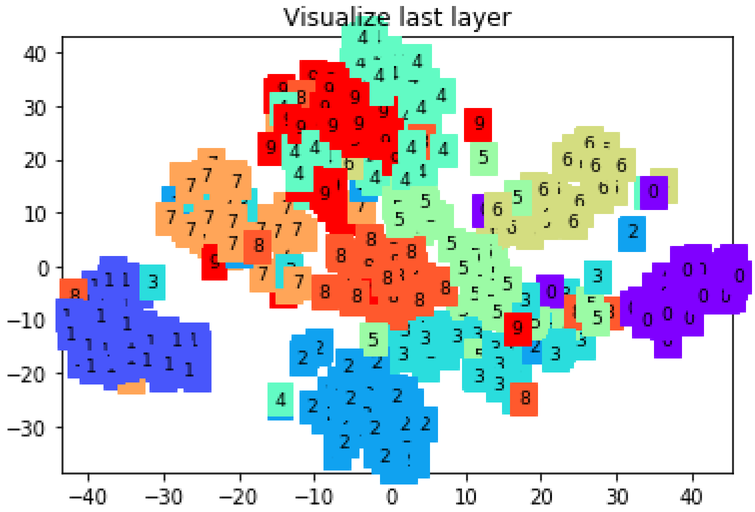
```
torch.Size([60000, 28, 28])
torch.Size([60000])
```



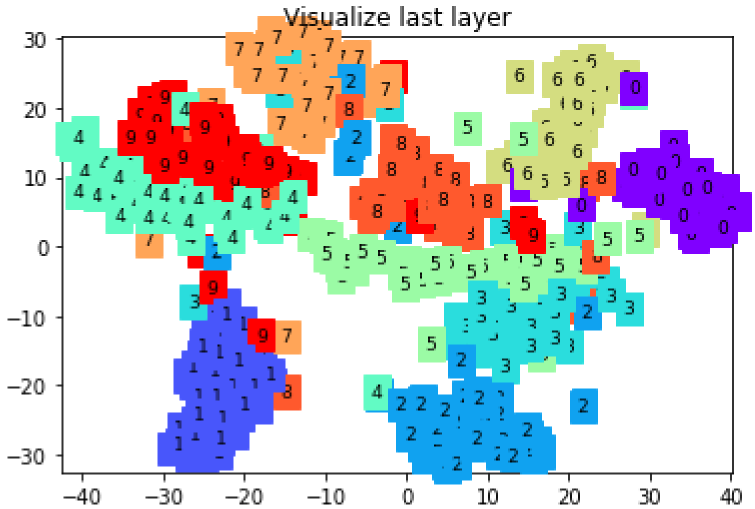

```
torch.Size([28, 28])
torch.Size([1, 28, 28])
tensor(7)
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (out): Linear(in_features=1568, out_features=10, bias=True)
)
Epoch:  0 | train loss: 2.3105 | test accuracy: 0.06
```



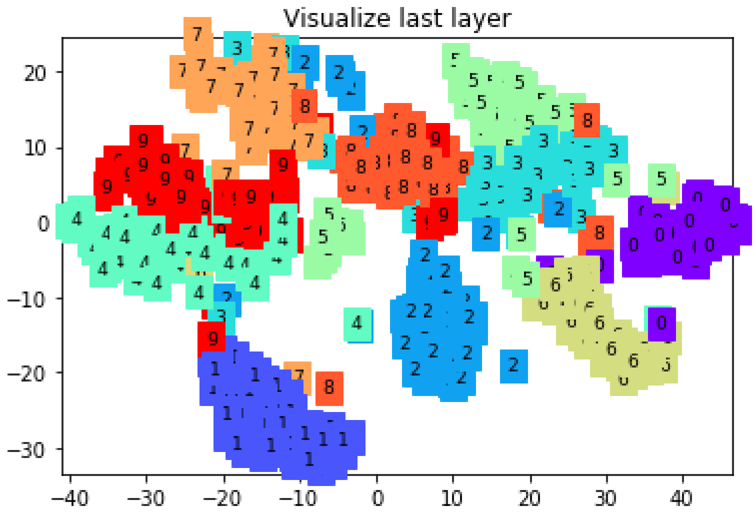

```
Epoch:  0 | train loss: 0.1289 | test accuracy: 0.87
```



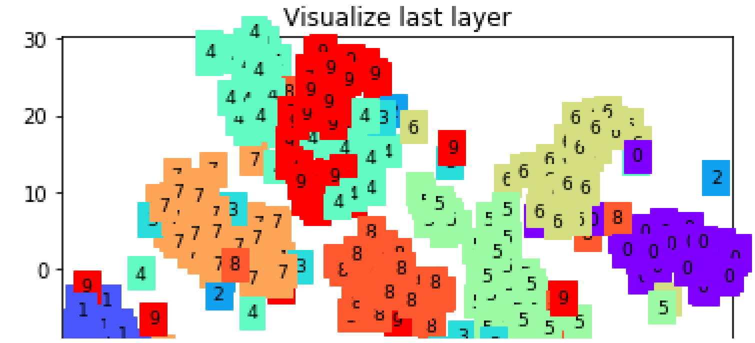
Epoch: 0 | train loss: 0.4050 | test accuracy: 0.93



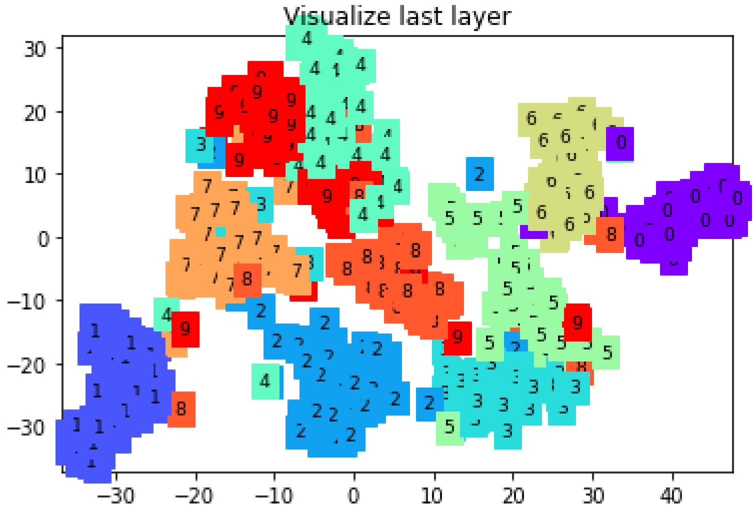
Epoch: 0 | train loss: 0.1943 | test accuracy: 0.94



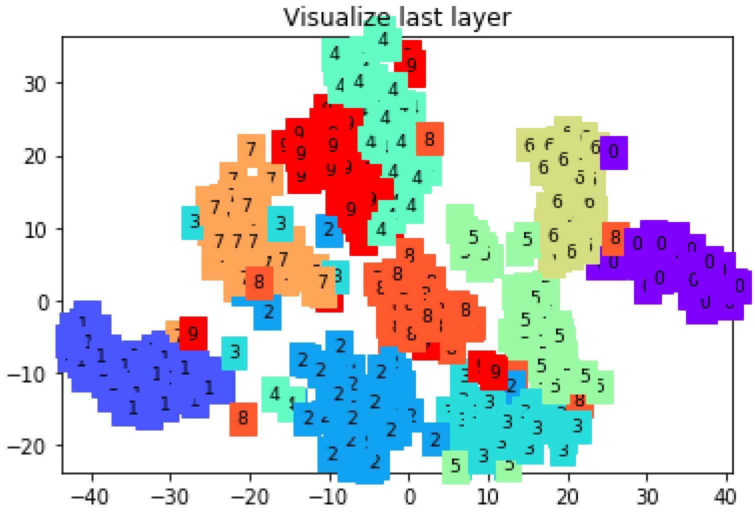
Epoch: 0 | train loss: 0.1280 | test accuracy: 0.96



Epoch: 0 | train loss: 0.2177 | test accuracy: 0.96

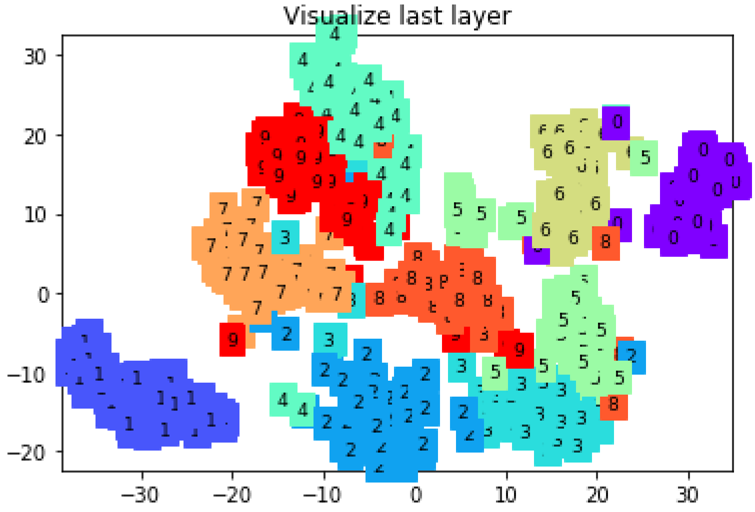


Epoch: 0 | train loss: 0.0232 | test accuracy: 0.97

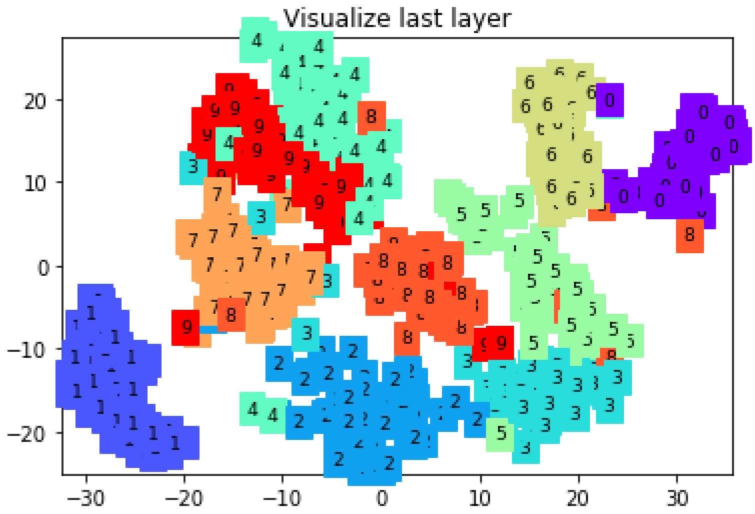


Epoch: 0 | train loss: 0.2171 | test accuracy: 0.97

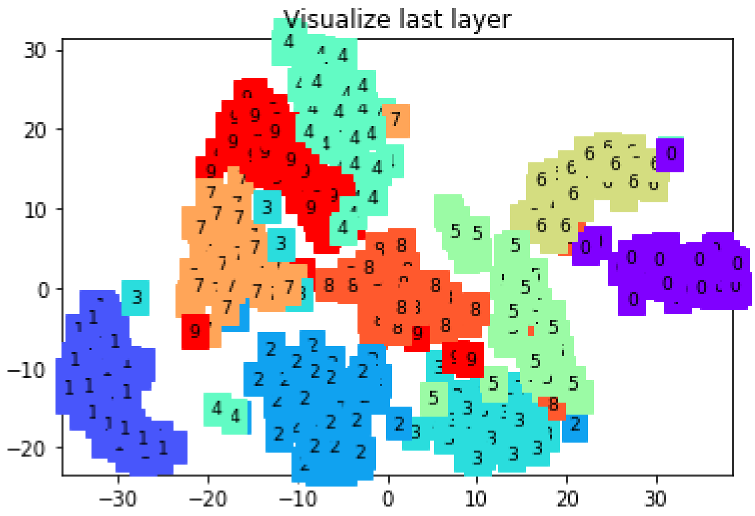
Epoch: 0 | train loss: 0.0445 | test accuracy: 0.97



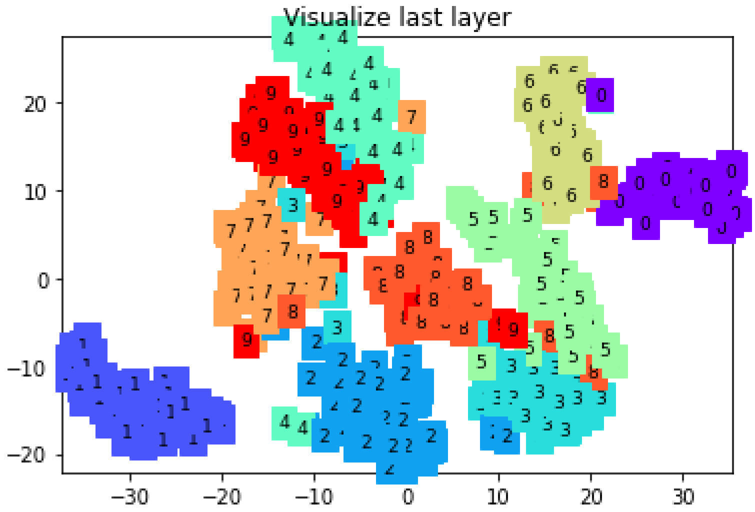
Epoch: 0 | train loss: 0.0572 | test accuracy: 0.98



Epoch: 0 | train loss: 0.0325 | test accuracy: 0.98



Epoch: 0 | train loss: 0.0257 | test accuracy: 0.98


Visualize last layer
20
10
0
−10
−20
−30
−20
−10
0
10
20
30

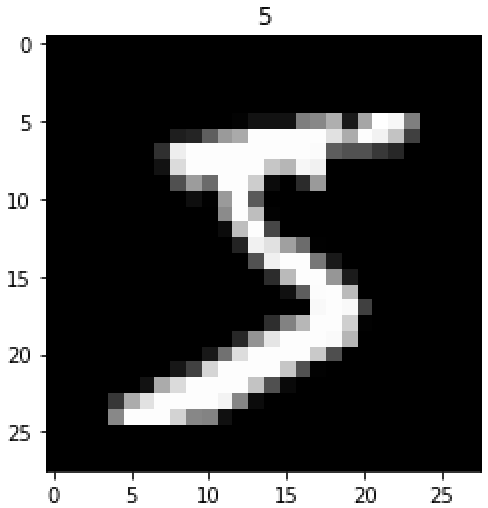
In [5]:

```
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.utils.data as Data
import torchvision
import matplotlib.pyplot as plt
#matplotlib inline

torch.manual_seed(1)    # reproducible
# Hyper Parameters
EPOCH = 1               # train the training data n times, to save time, we just tra
BATCH_SIZE = 50
LR = 0.001              # learning rate
DOWNLOAD_MNIST = False   # set to False if you have downloaded

# Mnist digits dataset
train_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=True,                                     # this is training data
    transform=torchvision.transforms.ToTensor(),    # Converts a PIL.Image or numpy.
                                                    # torch.FloatTensor of shape (C
    download=False,                        # download it if you don't have it
)
# plot one example
print(train_data.train_data.size())                 # (60000, 28, 28)
print(train_data.train_labels.size())               # (60000)
plt.imshow(train_data.train_data[0].numpy(), cmap='gray') # 행과 열을 가진 행렬 형태의 2차
plt.title('%i' % train_data.train_labels[0])
plt.show()

# Data Loader for easy mini-batch return in training, the image batch shape will be
train_loader = Data.DataLoader(dataset=train_data, batch_size=BATCH_SIZE, shuffle=Tr

# convert test data into Variable, pick 2000 samples to speed up testing
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)
print(test_data.test_data[0].size())
test_x = Variable(torch.unsqueeze(test_data.test_data, dim=1)).type(torch.FloatTenso
print(test_x[0].size())
# shape from (2000, 28, 28) to (2000, 1, 28, 28), value in range(0,1)
test_y = test_data.test_labels[:2000]
print(test_y[0])

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(         # input shape (1, 28, 28)
            nn.Conv2d(
                in_channels=1,              # input height
                out_channels=16,            # n_filters
                kernel_size=4,              # filter size
                stride=1,                   # filter movement/step
                padding=2,                  # if want same width and length of this
            ),                              # output shape (16, 28, 28)
            nn.ReLU(),                      # activation
            nn.MaxPool2d(kernel_size=2),    # choose max value in 2x2 area, output s
        )
        self.conv2 = nn.Sequential(         # input shape (1, 28, 28)
            nn.Conv2d(16, 32, 5, 1, 2),     # output shape (32, 14, 14)
            nn.ReLU(),                      # activation
```

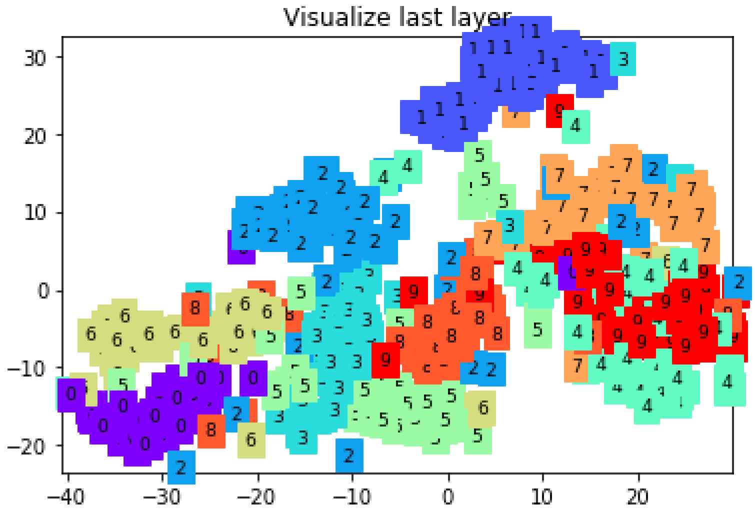
```
            nn.MaxPool2d(2),                # output shape (32, 7, 7)
        )
        self.out = nn.Linear(32 * 7 * 7, 10)   # fully connected layer, output 10 cl

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten the output of conv2 to (batch_
        output = self.out(x)
        return output, x    # return x for visualization

cnn = CNN()
print(cnn)  # net architecture

optimizer = torch.optim.Adam(cnn.parameters(), lr=LR)   # optimize all cnn paramete
loss_func = nn.CrossEntropyLoss()                       # the target label is not o

# following function (plot_with_labels) is for visualization, can be ignored if not
from matplotlib import cm
try: from sklearn.manifold import TSNE; HAS_SK = True
except: HAS_SK = False; print('Please install sklearn for layer visualization')
def plot_with_labels(lowDWeights, labels):
    plt.cla()
    X, Y = lowDWeights[:, 0], lowDWeights[:, 1]
    for x, y, s in zip(X, Y, labels):
        c = cm.rainbow(int(255 * s / 9)); plt.text(x, y, s, backgroundcolor=c, fonts
    plt.xlim(X.min(), X.max()); plt.ylim(Y.min(), Y.max()); plt.title('Visualize las

plt.ion()
# training and testing
for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader):   # gives batch data, normalize x w
        b_x = Variable(x)   # batch x
        b_y = Variable(y)   # batch y

        output = cnn(b_x)[0]               # cnn output
        loss = loss_func(output, b_y)   # cross entropy loss
        optimizer.zero_grad()           # clear gradients for this training step
        loss.backward()                 # backpropagation, compute gradients
        optimizer.step()                # apply gradients

        if step % 100 == 0:
            test_output, last_layer = cnn(test_x)
            pred_y = torch.max(test_output, 1)[1].data.squeeze()
            accuracy = (pred_y == test_y).sum().item() / float(test_y.size(0))
            print('Epoch: ', epoch, '| train loss: %.4f' % loss.data, '| test accura
            if HAS_SK:
                # Visualization of trained flatten layer (T-SNE)
                tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=5000)
                plot_only = 500
                low_dim_embs = tsne.fit_transform(last_layer.data.numpy()[:plot_only
                labels = test_y.numpy()[:plot_only]
                plot_with_labels(low_dim_embs, labels)
plt.ioff()
```

```
torch.Size([60000, 28, 28])
torch.Size([60000])

/Users/soojinlee/opt/anaconda3/lib/python3.7/site-packages/torchvisio
n/datasets/mnist.py:55: UserWarning: train_data has been renamed data
```

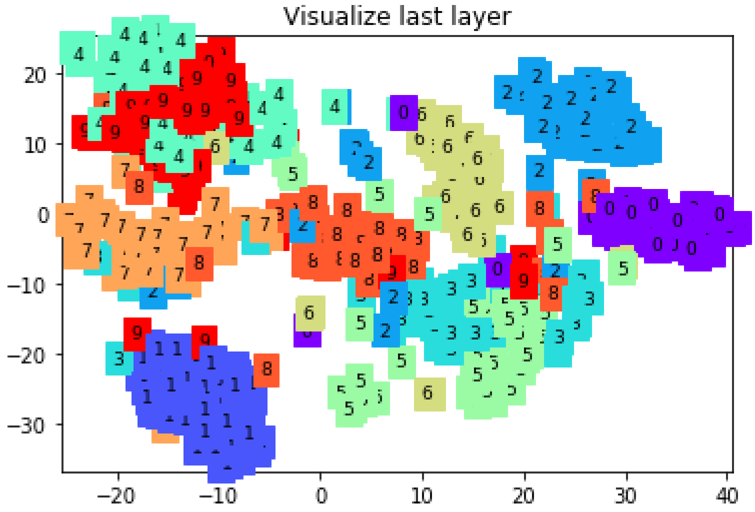
```
  warnings.warn("train_data has been renamed data")
/Users/soojinlee/opt/anaconda3/lib/python3.7/site-packages/torchvisio
n/datasets/mnist.py:45: UserWarning: train_labels has been renamed tar
gets
  warnings.warn("train_labels has been renamed targets")
```



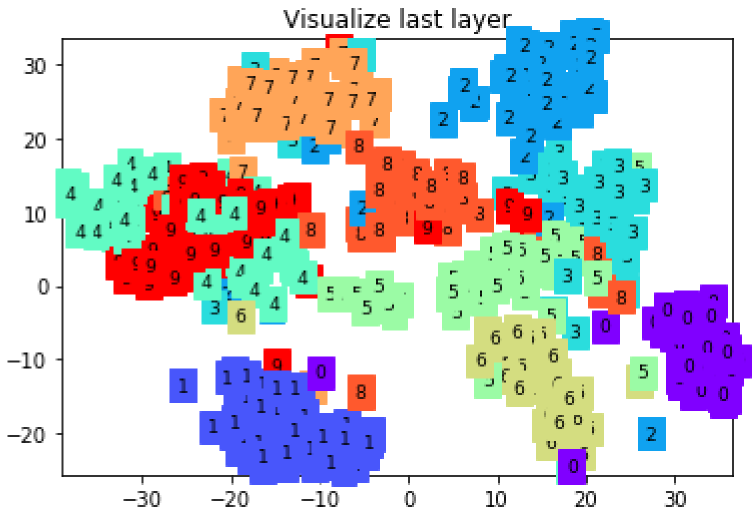
```
/Users/soojinlee/opt/anaconda3/lib/python3.7/site-packages/torchvisio
n/datasets/mnist.py:60: UserWarning: test_data has been renamed data
  warnings.warn("test_data has been renamed data")
/Users/soojinlee/opt/anaconda3/lib/python3.7/site-packages/torchvisio
n/datasets/mnist.py:50: UserWarning: test_labels has been renamed targ
ets
  warnings.warn("test_labels has been renamed targets")

torch.Size([28, 28])
torch.Size([1, 28, 28])
tensor(7)
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(4, 4), stride=(1, 1), padding=(2,
2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (out): Linear(in_features=1568, out_features=10, bias=True)
)
Epoch:  0 | train loss: 2.2922 | test accuracy: 0.10
```



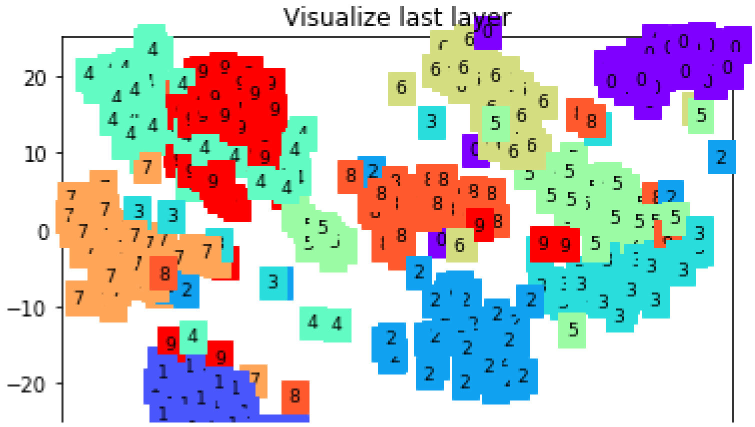

```
Epoch:  0 | train loss: 0.3844 | test accuracy: 0.89
```



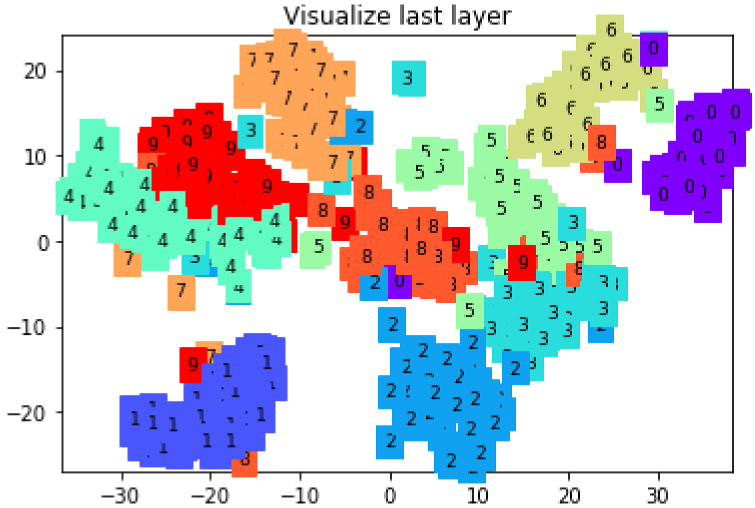

```
Epoch:  0 | train loss: 0.1905 | test accuracy: 0.93
```



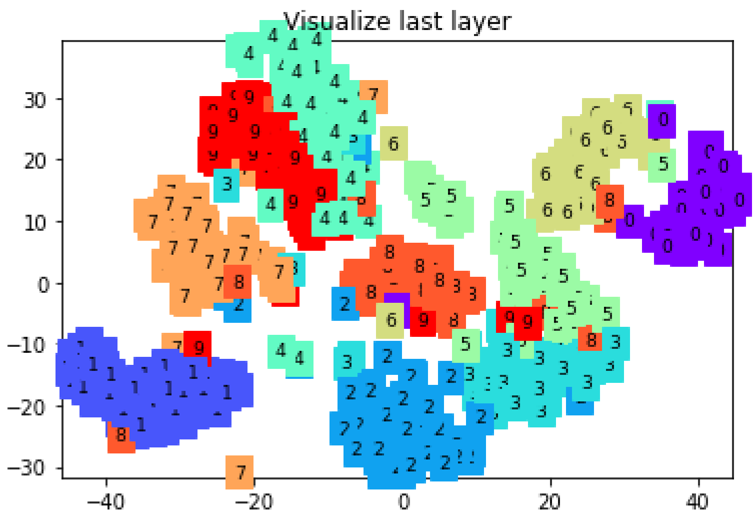

```
Epoch:  0 | train loss: 0.0514 | test accuracy: 0.95
```



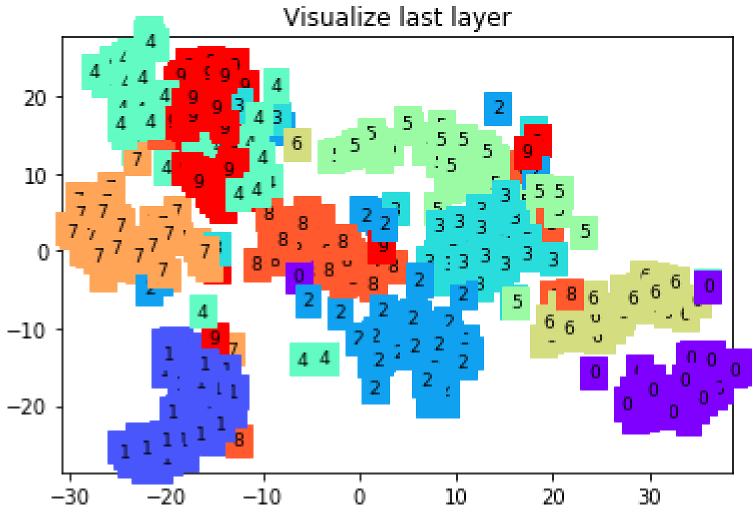
Epoch: 0 | train loss: 0.1706 | test accuracy: 0.96



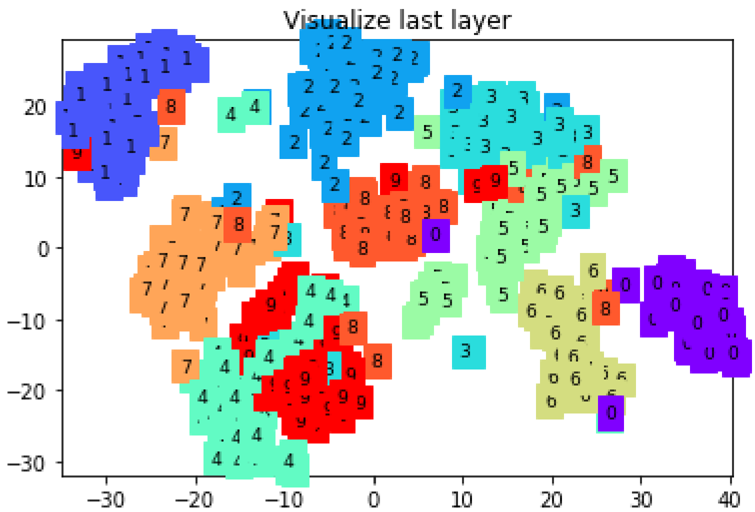
Epoch: 0 | train loss: 0.1461 | test accuracy: 0.96



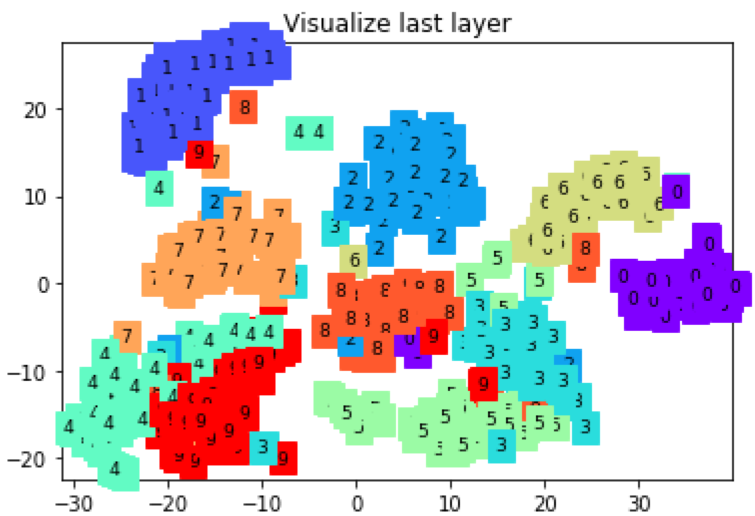
Epoch: 0 | train loss: 0.0355 | test accuracy: 0.97



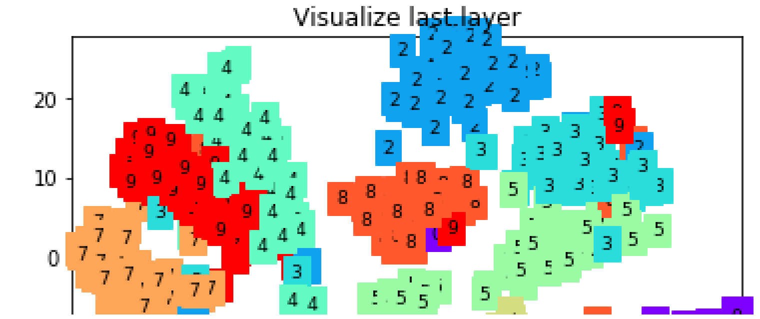
Epoch: 0 | train loss: 0.1302 | test accuracy: 0.96



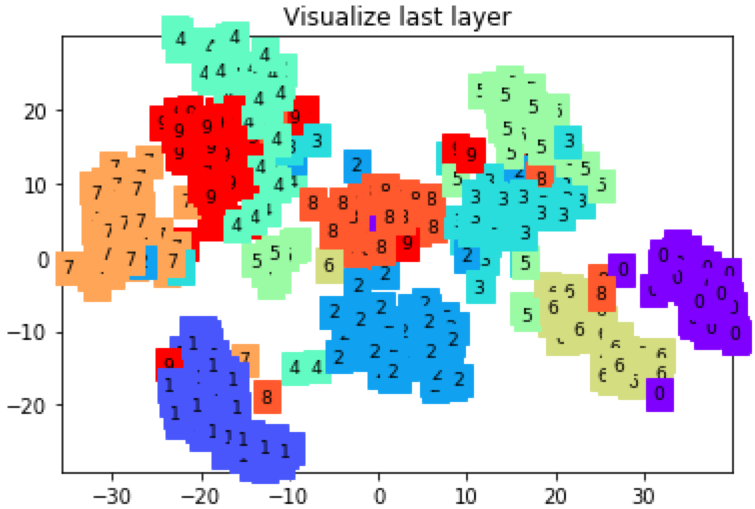
Epoch: 0 | train loss: 0.1552 | test accuracy: 0.97



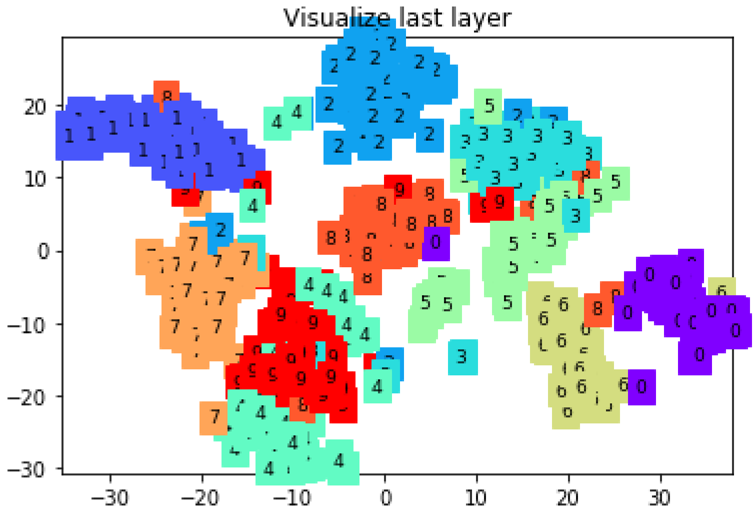
Epoch: 0 | train loss: 0.0949 | test accuracy: 0.97



Epoch: 0 | train loss: 0.0501 | test accuracy: 0.97



Epoch: 0 | train loss: 0.0327 | test accuracy: 0.98

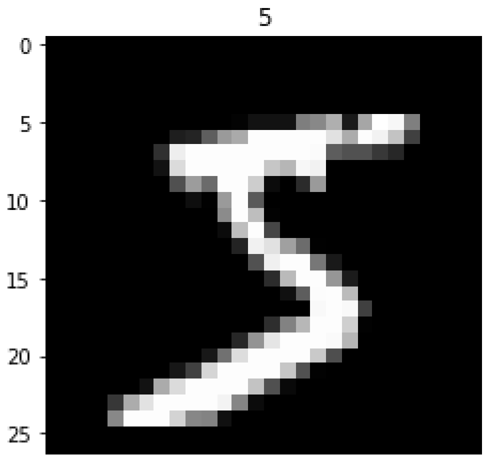


In [8]:

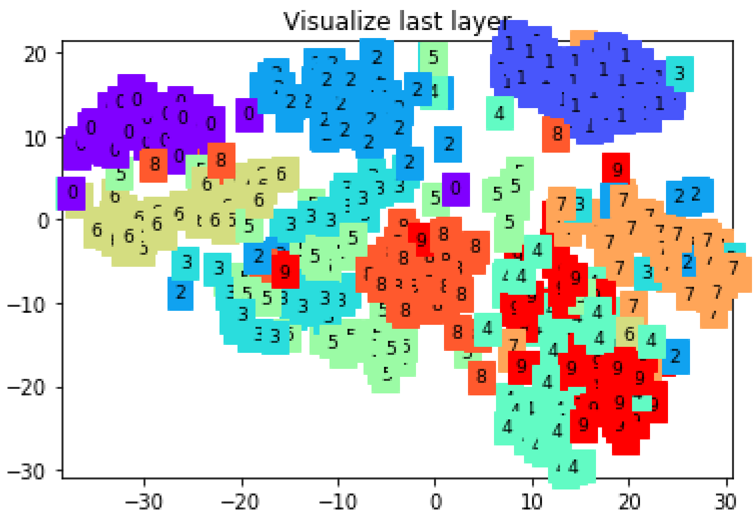
```
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.utils.data as Data
import torchvision
import matplotlib.pyplot as plt
#matplotlib inline

torch.manual_seed(1)    # reproducible
# Hyper Parameters
EPOCH = 1               # train the training data n times, to save time, we just tra
BATCH_SIZE = 50
LR = 0.001              # learning rate
DOWNLOAD_MNIST = False   # set to False if you have downloaded

# Mnist digits dataset
train_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=True,                                     # this is training data
    transform=torchvision.transforms.ToTensor(),    # Converts a PIL.Image or numpy.
                                                    # torch.FloatTensor of shape (C
    download=False,                        # download it if you don't have it
)
# plot one example
print(train_data.train_data.size())                 # (60000, 28, 28)
print(train_data.train_labels.size())               # (60000)
plt.imshow(train_data.train_data[0].numpy(), cmap='gray') # 행과 열을 가진 행렬 형태의 2차
plt.title('%i' % train_data.train_labels[0])
plt.show()

# Data Loader for easy mini-batch return in training, the image batch shape will be
train_loader = Data.DataLoader(dataset=train_data, batch_size=BATCH_SIZE, shuffle=Tr

# convert test data into Variable, pick 2000 samples to speed up testing
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)
print(test_data.test_data[0].size())
test_x = Variable(torch.unsqueeze(test_data.test_data, dim=1)).type(torch.FloatTenso
print(test_x[0].size())
# shape from (2000, 28, 28) to (2000, 1, 28, 28), value in range(0,1)
test_y = test_data.test_labels[:2000]
print(test_y[0])

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(         # input shape (1, 28, 28)
            nn.Conv2d(
                in_channels=1,              # input height
                out_channels=16,            # n_filters
                kernel_size=3,              # filter size
                stride=1,                   # filter movement/step
                padding=2,                  # if want same width and length of this
            ),                              # output shape (16, 28, 28)
            nn.ReLU(),                      # activation
            nn.MaxPool2d(kernel_size=2),    # choose max value in 2x2 area, output s
        )
        self.conv2 = nn.Sequential(         # input shape (1, 28, 28)
            nn.Conv2d(16, 32, 5, 1, 2),     # output shape (32, 14, 14)
            nn.ReLU(),                      # activation
```

```
            nn.MaxPool2d(2),                # output shape (32, 7, 7)
        )
        self.out = nn.Linear(32 * 7 * 7, 10)   # fully connected layer, output 10 cl

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten the output of conv2 to (batch_
        output = self.out(x)
        return output, x    # return x for visualization

cnn = CNN()
print(cnn)  # net architecture

optimizer = torch.optim.Adam(cnn.parameters(), lr=LR)   # optimize all cnn paramete
loss_func = nn.CrossEntropyLoss()                       # the target label is not or

# following function (plot_with_labels) is for visualization, can be ignored if not
from matplotlib import cm
try: from sklearn.manifold import TSNE; HAS_SK = True
except: HAS_SK = False; print('Please install sklearn for layer visualization')
def plot_with_labels(lowDWeights, labels):
    plt.cla()
    X, Y = lowDWeights[:, 0], lowDWeights[:, 1]
    for x, y, s in zip(X, Y, labels):
        c = cm.rainbow(int(255 * s / 9)); plt.text(x, y, s, backgroundcolor=c, fonts
    plt.xlim(X.min(), X.max()); plt.ylim(Y.min(), Y.max()); plt.title('Visualize las

plt.ion()
# training and testing
for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader):   # gives batch data, normalize x w
        b_x = Variable(x)   # batch x
        b_y = Variable(y)   # batch y

        output = cnn(b_x)[0]               # cnn output
        loss = loss_func(output, b_y)   # cross entropy loss
        optimizer.zero_grad()           # clear gradients for this training step
        loss.backward()                 # backpropagation, compute gradients
        optimizer.step()                # apply gradients

        if step % 100 == 0:
            test_output, last_layer = cnn(test_x)
            pred_y = torch.max(test_output, 1)[1].data.squeeze()
            accuracy = (pred_y == test_y).sum().item() / float(test_y.size(0))
            print('Epoch: ', epoch, '| train loss: %.4f' % loss.data, '| test accura
            if HAS_SK:
                # Visualization of trained flatten layer (T-SNE)
                tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=5000)
                plot_only = 500
                low_dim_embs = tsne.fit_transform(last_layer.data.numpy()[:plot_only
                labels = test_y.numpy()[:plot_only]
                plot_with_labels(low_dim_embs, labels)
plt.ioff()
```

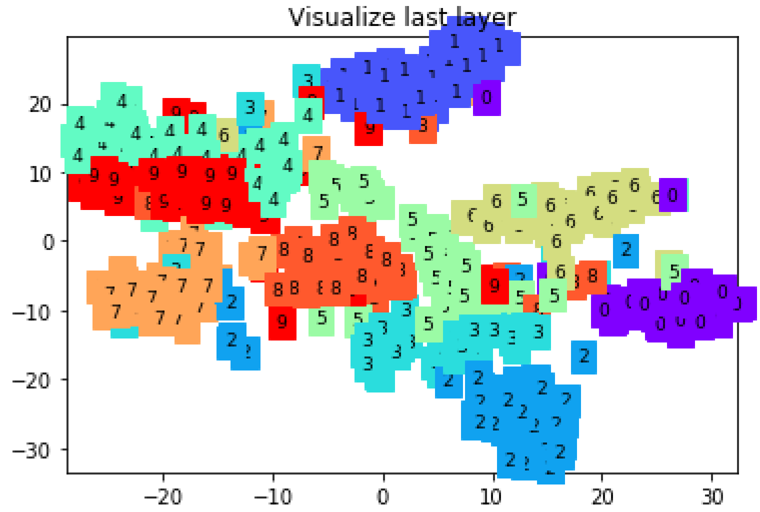
```
torch.Size([60000, 28, 28])
torch.Size([60000])
```



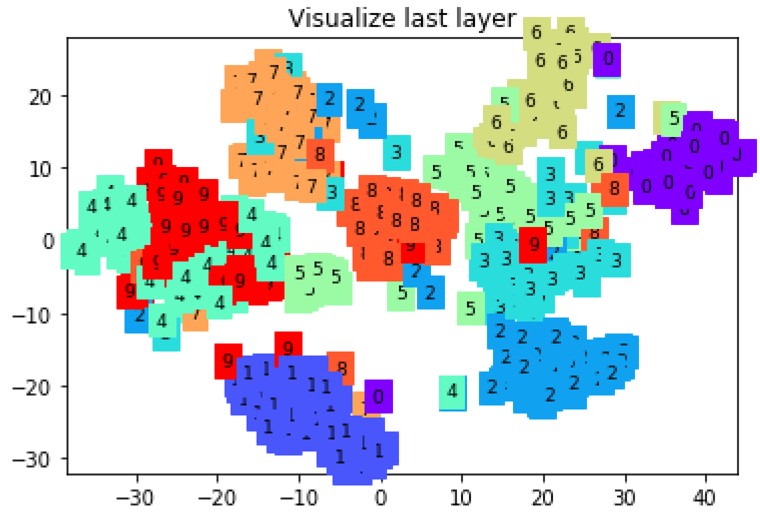

```
torch.Size([28, 28])
torch.Size([1, 28, 28])
tensor(7)
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(2,
2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (out): Linear(in_features=1568, out_features=10, bias=True)
)
Epoch:  0 | train loss: 2.3062 | test accuracy: 0.11
```



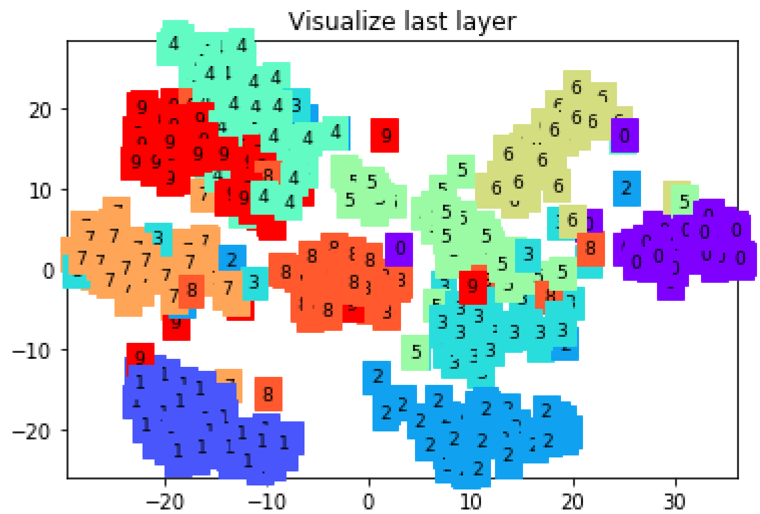

```
Epoch:  0 | train loss: 0.4921 | test accuracy: 0.89
```



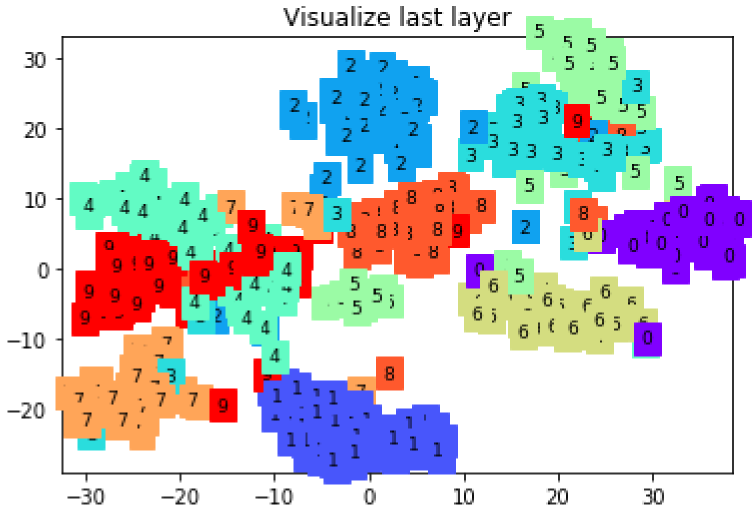
Epoch: 0 | train loss: 0.2306 | test accuracy: 0.93



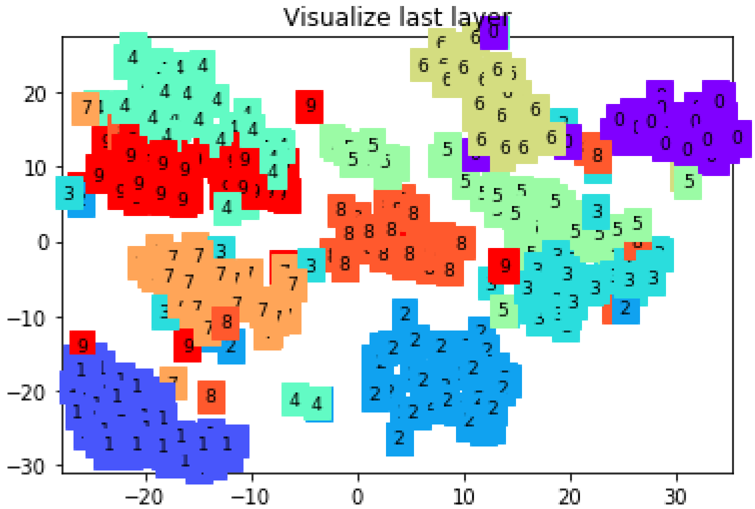
Epoch: 0 | train loss: 0.3424 | test accuracy: 0.94



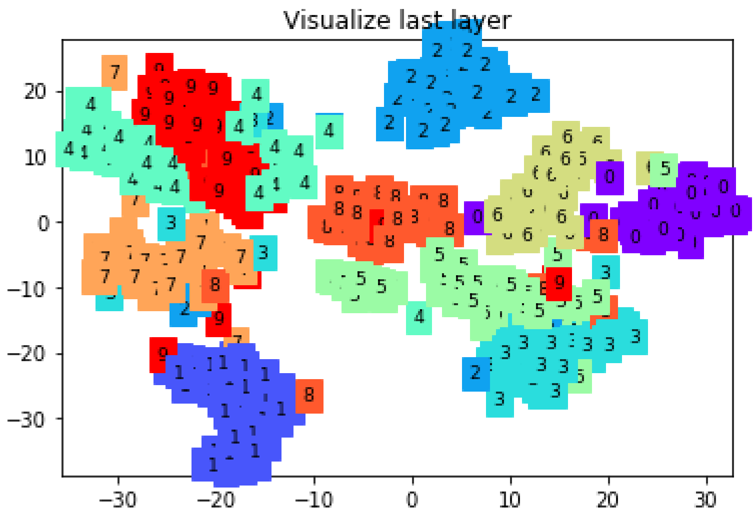
Epoch: 0 | train loss: 0.0909 | test accuracy: 0.96



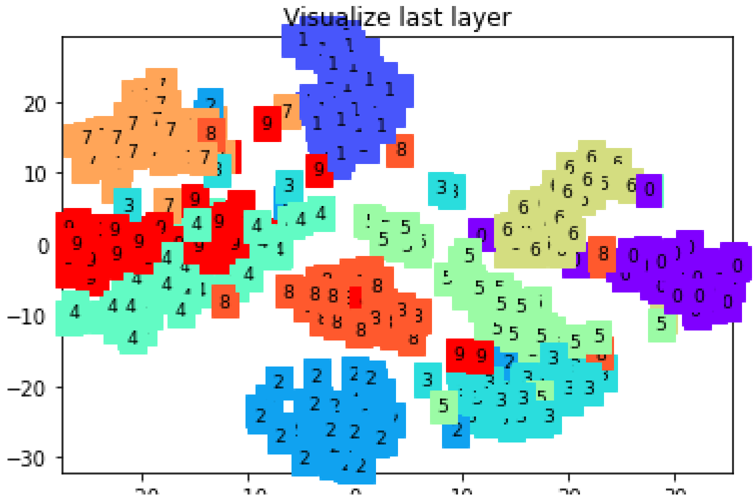
Epoch: 0 | train loss: 0.1735 | test accuracy: 0.95



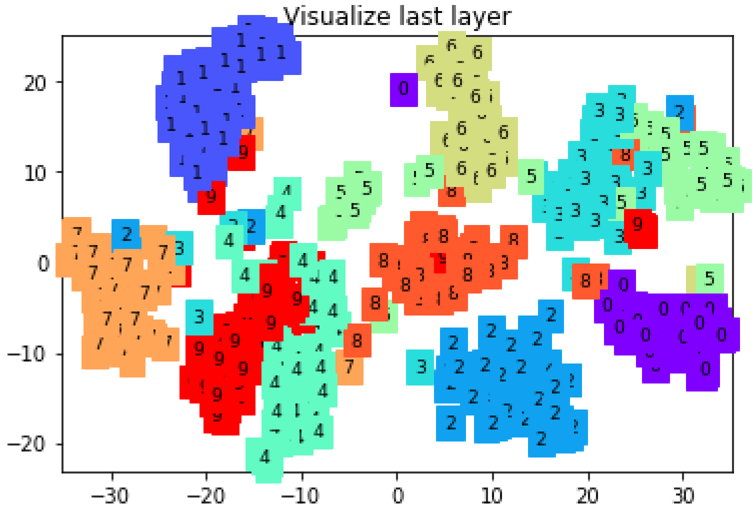
Epoch: 0 | train loss: 0.0708 | test accuracy: 0.96



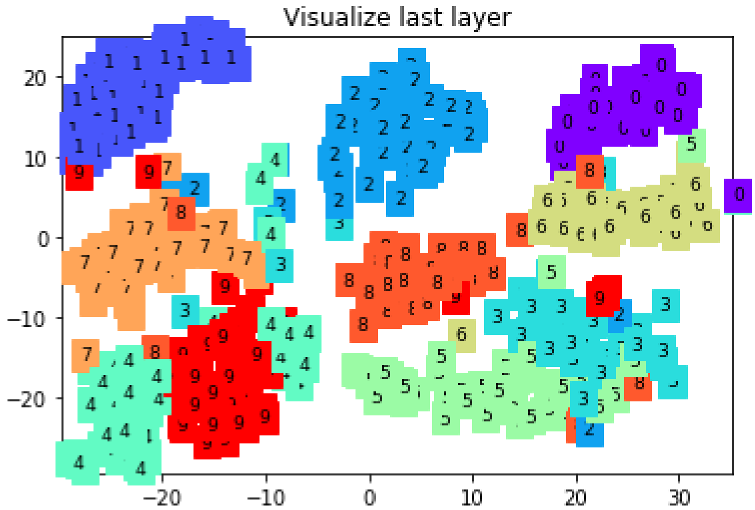
Epoch: 0 | train loss: 0.0871 | test accuracy: 0.96



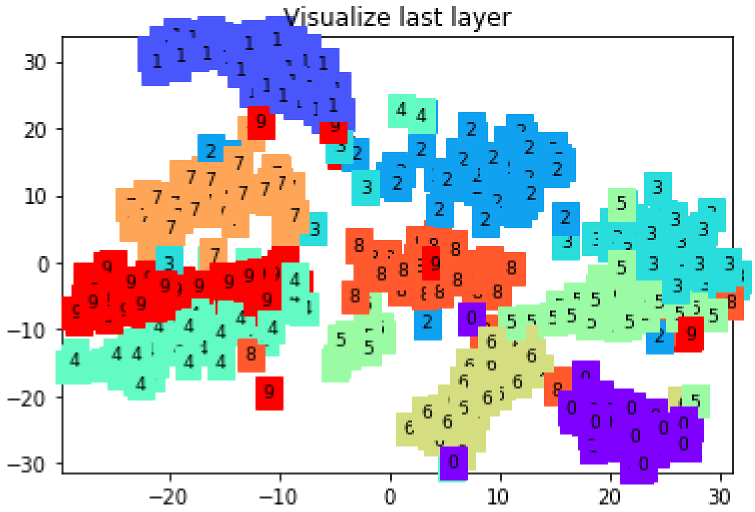
Epoch: 0 | train loss: 0.0682 | test accuracy: 0.97



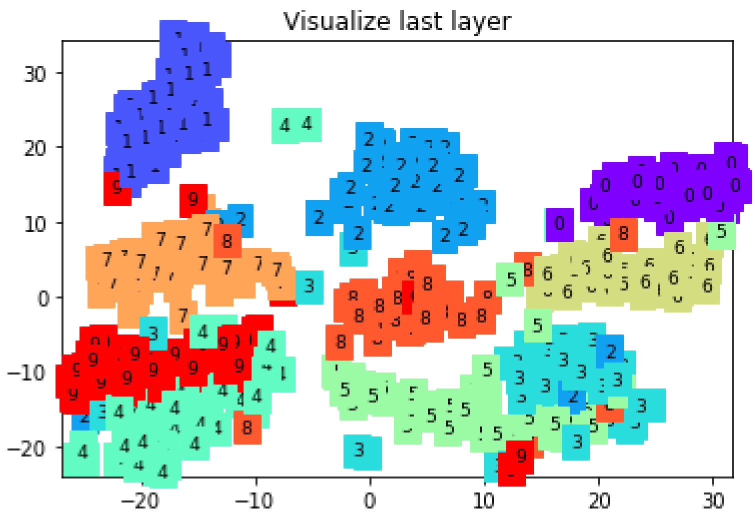
Epoch: 0 | train loss: 0.1203 | test accuracy: 0.97



Epoch: 0 | train loss: 0.0773 | test accuracy: 0.97




```
Epoch:  0 | train loss: 0.0642 | test accuracy: 0.97
```



In [ ]: