vae+classifier

April 25, 2019

0.1 Variational AutoEnoder:

1. Understanding - disentagled latent spaces
2. Using classifier - to understand its performance, VAE (detach updates) + Classifier => performance
3. Using VAE+ classifier - to understand its performance

In [0]:
```
from glob import glob
import os
import numpy as np
import matplotlib.pyplot as plt
import shutil
from torchvision import transforms
from torchvision import models
import torchvision
import torch
from torch.autograd import Variable
import torch.nn as nn
from torch.optim import lr_scheduler
from torch import optim
from torchvision.datasets import ImageFolder
from torchvision.utils import make_grid
import time
from torchvision import datasets,transforms
import torch.nn.functional as F
import torch.optim as optim


import os
from torchvision.utils import save_image

## Plotting library
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas

from scipy.stats import norm
from sklearn import manifold
```

```python
plt.style.use('fivethirtyeight')
%matplotlib inline

print('Torch', torch.__version__, 'CUDA', torch.version.cuda)
print('Device:', torch.device('cuda:0'))
print(torch.cuda.is_available())

is_cuda = torch.cuda.is_available()
device = torch.device ( "cuda:0" if torch.cuda.is_available () else "cpu" )
```

```
Torch 1.0.1.post2 CUDA 10.0.130
Device: cuda:0
True
```

In [0]:
```python
## Show image
def imshow(img,title=None):
  """Imshow for Tensor."""
  img = img.numpy().transpose((1,2,0))
  mean = np.array([0.485, 0.456, 0.406])
  std = np.array([0.229, 0.224, 0.225])
  img = std * img + mean # normalize
  img = np.clip(img, 0, 1) # clip image
  plt.figure(figsize=(16,4))
  plt.axis('off')
  plt.imshow(img)

  if title is not None:
    plt.title(title)

def plot_grid(inputs):
  # Make a grid from batch
  out = torchvision.utils.make_grid(inputs,10,10)
  imshow(out, title="")

## Visualize some images in the dataset
def visualizeDataset(X):
  for i,image in enumerate(X):
    cv2.imshow(str(i),image)
    cv2.waitKey()
    cv2.destroyAllWindows()

def plot_loss(y, title):
  plt.figure()
  plt.plot(y)
  plt.title(title)
  plt.xlabel('epochs')
  plt.ylabel('Loss')
```

```
def plot_accuracy(y, title):
  plt.figure()
  plt.plot(y)
  plt.title(title)
  plt.xlabel('epochs')
  plt.ylabel('accuracy')


## Scatter Plot
def scatterplot(x, y, ax, imageData, zoom):
  images = []
  imageSize = 28
  for i in range(len(x)):
    x0, y0 = x[i], y[i]

    # Convert to image
    img = imageData[i]*255.
    img = (img).numpy()
    img = img.astype(np.uint8).reshape([imageSize,imageSize])
    img = cv2.cvtColor(img,cv2.COLOR_GRAY2RGB)

    # Note: OpenCV uses BGR and plt uses RGB
    image = OffsetImage(img, zoom=zoom)
    ab = AnnotationBbox(image, (x0, y0), xycoords='data', frameon=False)
    images.append(ax.add_artist(ab))

  ax.update_datalim(np.column_stack([x, y]))
  ax.autoscale()


import seaborn as sns
palette = np.array(sns.color_palette("hls", 10))

def plot_scatter(projection,labels):
    plt.scatter(projection[:,0], projection[:,1],c=[palette[i] for i in labels])
```

In [0]:
```
class Params:
  nb_latents = 10
  batch_size = 128
  epochs = 100
  log_interval = 100
  save_interval = 1000


torch.manual_seed(5)
```

Out[0]: <torch._C.Generator at 0x7f3e918c16f0>

0.2 Metrics

In [0]:
```
### Metrics - Base Class For all Metrics
class Metric:
  def __init__(self):
    pass
  def __call__(self, outputs, target, loss):
    raise NotImplementedError

  def reset(self):
    raise NotImplementedError

  def value(self):
    raise NotImplementedError

  def name(self):
    raise NotImplementedError

## Accuracy Metric
class AccumulatedAccuracyMetric(Metric):
  def __init__(self):
      self.correct = 0
      self.total = 0

  def __call__(self, outputs, target):
      # Track the accuracy
      _, argmax = torch.max(outputs, 1)
      accuracy = (target == argmax.squeeze()).float().sum()
      self.correct += accuracy
      self.total += target.size(0)
      return self.value()

  def reset(self):
      self.correct = 0
      self.total = 0

  def value(self):
      return 100 * float(self.correct) / self.total

  def name(self):
      return 'Accuracy'


## Loss
class RunningAverage ():
    """A simple class that maintains the running average of a quantity
    Example:
    ```
```
```

```
        loss_avg = RunningAverage()
        loss_avg.update(2)
        loss_avg.update(4)
        loss_avg() = 3
        ```
        """

        def __init__(self):
            self.steps = 0
            self.total = 0

        def update(self, val):
            self.total += val
            self.steps += 1

        def __call__(self):
            return self.total / float (self.steps)
```

## 0.3 Dataloader

```
In [0]: path = "./vae-classifier/"
        transformation = transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.1307
        train_dataset = datasets.MNIST(os.path.join(path,"MNIST/data"),train=True,transform=tra
        test_dataset = datasets.MNIST(os.path.join(path,"MNIST/data"),train=False,transform=tra
        train_loader = torch.utils.data.DataLoader(train_dataset,batch_size=128,shuffle=True)
        test_loader = torch.utils.data.DataLoader(test_dataset,batch_size=128,shuffle=True)
        ## Test TSNE plot for reconstrunction on 1000 test samples
        testing_tsne = torch.utils.data.DataLoader(test_dataset,batch_size=len(train_dataset),s
        test_data, test_labels = next(iter(testing_tsne))[:10000]
```

```
In [0]: print(f"Total number of train images: {len(train_dataset)}, total number of test images
```

```
Total number of train images: 60000, total number of test images: 10000, total number of train
```

```
In [0]: %ls
        import os
        os.makedirs("vae-classifier/results")
```

```
beta-results/     results/      vae-classifier/    VAE_GAN_decoder_249.pth
latent_space.png  sample_data/  VAE_GAN_D_249.pth  VAE_GAN_encoder_249.pth
```

0.4 Model

```
In [0]: class ConvVAE(nn.Module):
            def __init__(self, nb_latents):
                super(ConvVAE, self).__init__()
                self.conv1 = nn.Sequential(
```
```

```python
            nn.Conv2d(1, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))


        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))

        self.conv4 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=2, stride=1),
            nn.ReLU())

        self.fc1 = nn.Linear(1024, 256)

        self.fc_mean = nn.Linear(256, nb_latents)
        self.fc_std = nn.Linear(256, nb_latents)

        self.fc2 = nn.Linear(nb_latents, 256)
        self.fc3 = nn.Linear(256, 1024)



        self.fc4 = nn.Linear(1024,7*7*64)

        self.deconv1 = nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2)
        self.deconv2 = nn.ConvTranspose2d(32, 1, kernel_size=2, stride=2)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def encode(self, x):
        x = (self.conv1(x))
        x =  (self.conv2(x))
        x =  (self.conv3(x))
        x =  (self.conv4(x))
        x = x.reshape(x.size(0), -1)
        x = self.relu(self.fc1(x))
        return self.fc_mean(x), self.fc_std(x)

    def reparameterize(self, mu, logvar):
        if self.training:
            std = logvar.mul(0.5).exp_()
```

```
                eps = Variable(std.data.new(std.size()).normal_())
                return eps.mul(std).add_(mu)
            else:
                return mu

        def decode(self, z):
            x = self.relu(self.fc2(z))
            x = self.relu(self.fc3(x))
            x = self.relu(self.fc4(x))
            x = self.relu(self.deconv1(x.view(-1, 64, 7, 7)))
            x = self.deconv2(x)
            return self.sigmoid(x)

        def forward(self, x):
            mu, logvar = self.encode(x)
            z = self.reparameterize(mu, logvar)
            return self.decode(z), mu, logvar
```

In [0]:
```
model = ConvVAE(Params.nb_latents)
if is_cuda:
  model = model.to(device)

print(model)
```

```
ConvVAE(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(128, 256, kernel_size=(2, 2), stride=(1, 1))
    (1): ReLU()
  )
  (fc1): Linear(in_features=1024, out_features=256, bias=True)
  (fc_mean): Linear(in_features=256, out_features=10, bias=True)
  (fc_std): Linear(in_features=256, out_features=10, bias=True)
  (fc2): Linear(in_features=10, out_features=256, bias=True)
```

```
  (fc3): Linear(in_features=256, out_features=1024, bias=True)
  (fc4): Linear(in_features=1024, out_features=3136, bias=True)
  (deconv1): ConvTranspose2d(64, 32, kernel_size=(2, 2), stride=(2, 2))
  (deconv2): ConvTranspose2d(32, 1, kernel_size=(2, 2), stride=(2, 2))
  (relu): ReLU()
  (sigmoid): Sigmoid()
)
```

0.5 Loss Function

```
In [0]: ### Loss Function
        ### Loss Function
        def loss_function(recon_x, x, mu, logvar,beta=1):
          """
          Reconstruction loss  + KL divergence loss over all elements of the batch
          """
          bce = F.binary_cross_entropy(recon_x, x.view(-1, 28*28), size_average=False)

          kld = -0.5* (1+ logvar -mu.pow(2) -  logvar.exp())
          return kld.mean(dim = 0), bce + beta*kld.sum()
```

```
In [0]: import os
        os.makedirs("vae-classifier/results")
        %ls
```

```
results/  sample_data/  vae-classifier/
```

0.6 Latent Space Visualization

```
In [0]: criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=1e-3)

        """
        Traverse Latents
        """
        def traverse_latents(model, datapoint, nb_latents, epoch, batch_idx, dirpath="./results
          model.eval()
          datapoint = datapoint.to(device)
          if isinstance(model,ConvVAE):
            datapoint = datapoint.unsqueeze(0)
            mu, _ = model.encode(datapoint)
          else:
            mu, _ = model.encode(datapoint.view(-1))

          recons = torch.zeros((7, nb_latents, 28, 28))
          for zi in range(nb_latents):
```

```
        muc = mu.squeeze().clone()
        for i, val in enumerate(np.linspace(-3, 3, 7)):
          muc[zi] = val

          recon = model.decode(muc).cpu()
          recons[i, zi] = recon.view(28, 28)

      filename = os.path.join(dirpath, 'traversal_' + str(epoch) + '_' + str(batch_idx) +
      save_image(recons.view(-1, 1, 28, 28), filename, nrow=nb_latents, pad_value=1)
```

In [0]:
```
## Load Model state
## Save model
def save_model(model,path):

    torch.save(model.state_dict(),path)


def load_model(model,path):
  model.load_state_dict(torch.load(path))


load_model(model,"./vae-classifier/results/model_state_99.pth")
```

In [0]:
```
#!pip install tqdm
!pip install opencv-python
```

0.7 Compute Laplacian Variance for the blur - cv2

In [0]:
```
## Import CV
import cv2
import numpy


def laplacian_variance(images):
    return [cv2.Laplacian(image.numpy(), cv2.CV_32F).var() for image in images]


def laplacian_variance_numpy(images):
    return [cv2.Laplacian(image, cv2.CV_32F).var() for image in images]
```

In [0]:
```
log_interval = 400
save_interval = 400
testpoint = torch.Tensor(train_loader.dataset[0][0]).to(device)
save_results = "./results/"
def train_epoch(epoch):
  model.train()
  train_losses = RunningAverage()
```

```
    for batch_idx, (data, labels) in enumerate(train_loader):
      data = data.to(device)
      labels = labels.to(device)
      batch_size = data.size(0)
      recon_batch, mu, logvar = model(data)

      kld, loss = loss_function(recon_batch.squeeze().view(-1,28*28), data, mu, logvar)
      ## parameter update
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()


      loss /=len(data)
      train_losses.update(loss.item())
      if batch_idx % log_interval == 0:
        print("Epoch {}, batch: {}/{}, loss: {:.2f}".format(
                epoch, batch_idx, len(train_loader), loss))

    return train_losses()
```

In [0]:
```
## Test Epoch
"""
Test
"""
def test_epoch(epoch):
    model.eval()
    test_loss = RunningAverage()

    ## Test reconstruction test
    testpoint = torch.Tensor(test_loader.dataset[0][0]).to(device)

    with torch.no_grad():
        for i, (data,labels) in enumerate(test_loader):
            data = data.to(device)

            batch_size = data.size(0)
            recon_batch, mu, logvar = model(data)

            ## Loss of VAE
            kld, loss = loss_function(recon_batch.squeeze().view(-1,28*28), data, mu, l

            test_loss.update(loss.item() / len(data))

    return test_loss()
```

In [0]:
```
train_losses = []
test_losses = []
```

```python
for epoch in range(Params.epochs):
  train_loss = train_epoch(epoch)
  train_losses.append(train_loss)
  message = 'Epoch: {}/{}. Train set: Average loss: {:.4f}'.format(epoch + 1, Params.ep
  test_loss = test_epoch(epoch)
  message = 'Epoch: {}/{}. Test set: Average loss: {:.4f}'.format(epoch + 1, Params.epo
  test_losses.append(test_loss)

plt.plot(range(Params.epochs),train_losses)
plt.plot(range(Params.epochs),test_losses)
plt.legend(["Train loss","Test loss"])
plt.show()
```

```
/home/chivukula_manju/yes/lib/python3.6/site-packages/torch/nn/_reduction.py:49: UserWarning: s
  warnings.warn(warning.format(ret))


Epoch 0, batch: 0/469, loss: -11185.63
Epoch 0, batch: 400/469, loss: -11298.16
Epoch 1, batch: 0/469, loss: -11267.52
Epoch 1, batch: 400/469, loss: -11227.96
Epoch 2, batch: 0/469, loss: -11326.61
Epoch 2, batch: 400/469, loss: -11165.04
Epoch 3, batch: 0/469, loss: -11360.92
Epoch 3, batch: 400/469, loss: -11202.07
Epoch 4, batch: 0/469, loss: -11215.79
Epoch 4, batch: 400/469, loss: -11148.99
Epoch 5, batch: 0/469, loss: -11297.42
Epoch 5, batch: 400/469, loss: -11143.45
Epoch 6, batch: 0/469, loss: -11193.46
Epoch 6, batch: 400/469, loss: -11238.29
Epoch 7, batch: 0/469, loss: -11163.70
Epoch 7, batch: 400/469, loss: -11278.06
Epoch 8, batch: 0/469, loss: -11306.77
Epoch 8, batch: 400/469, loss: -11321.87
Epoch 9, batch: 0/469, loss: -11192.15
Epoch 9, batch: 400/469, loss: -11207.91
Epoch 10, batch: 0/469, loss: -11279.09
Epoch 10, batch: 400/469, loss: -11185.44
Epoch 11, batch: 0/469, loss: -11338.42
Epoch 11, batch: 400/469, loss: -11197.82
Epoch 12, batch: 0/469, loss: -11444.47
Epoch 12, batch: 400/469, loss: -11431.17
Epoch 13, batch: 0/469, loss: -11046.42
Epoch 13, batch: 400/469, loss: -11271.06
Epoch 14, batch: 0/469, loss: -11226.61
Epoch 14, batch: 400/469, loss: -11239.33
Epoch 15, batch: 0/469, loss: -11340.82
```

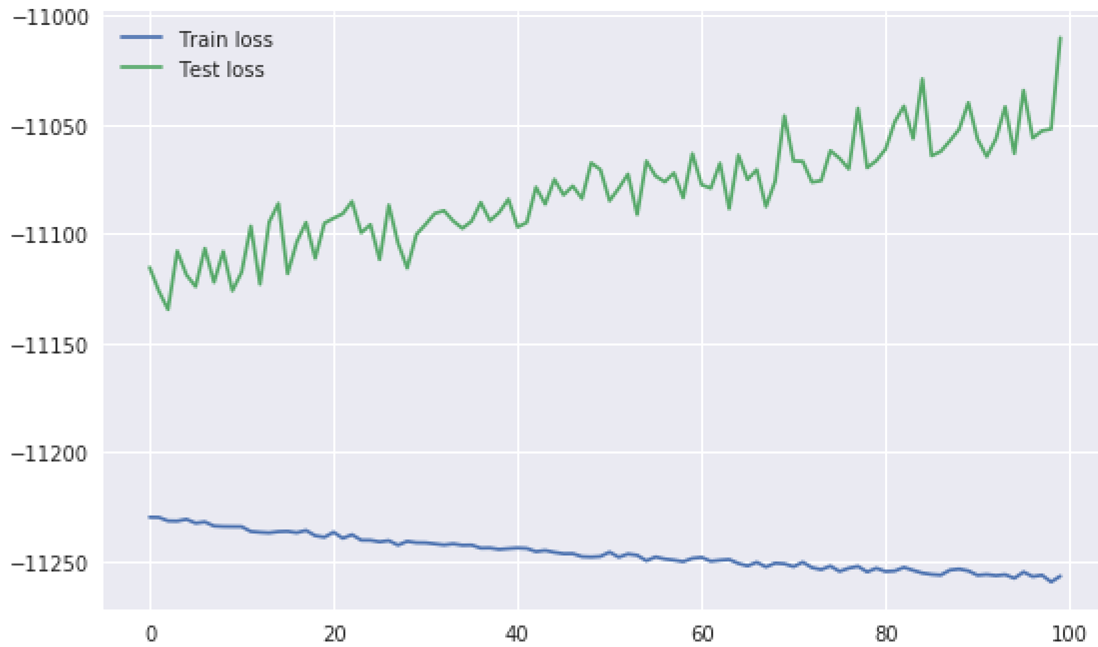
```
Epoch 15, batch: 400/469, loss: -11190.63
Epoch 16, batch: 0/469, loss: -11313.45
Epoch 16, batch: 400/469, loss: -11242.71
Epoch 17, batch: 0/469, loss: -11254.94
Epoch 17, batch: 400/469, loss: -11239.55
Epoch 18, batch: 0/469, loss: -11342.11
Epoch 18, batch: 400/469, loss: -11109.28
Epoch 19, batch: 0/469, loss: -11267.64
Epoch 19, batch: 400/469, loss: -11215.08
Epoch 20, batch: 0/469, loss: -11136.83
Epoch 20, batch: 400/469, loss: -11262.31
Epoch 21, batch: 0/469, loss: -11237.15
Epoch 21, batch: 400/469, loss: -11452.75
Epoch 22, batch: 0/469, loss: -11316.09
Epoch 22, batch: 400/469, loss: -11309.62
Epoch 23, batch: 0/469, loss: -11258.09
Epoch 23, batch: 400/469, loss: -11177.51
Epoch 24, batch: 0/469, loss: -11159.50
Epoch 24, batch: 400/469, loss: -11172.05
Epoch 25, batch: 0/469, loss: -11299.92
Epoch 25, batch: 400/469, loss: -11311.95
Epoch 26, batch: 0/469, loss: -11214.66
Epoch 26, batch: 400/469, loss: -11371.07
Epoch 27, batch: 0/469, loss: -11238.23
Epoch 27, batch: 400/469, loss: -11337.68
Epoch 28, batch: 0/469, loss: -11133.16
Epoch 28, batch: 400/469, loss: -11116.90
Epoch 29, batch: 0/469, loss: -11208.38
Epoch 29, batch: 400/469, loss: -11383.78
Epoch 30, batch: 0/469, loss: -11318.38
Epoch 30, batch: 400/469, loss: -11176.62
Epoch 31, batch: 0/469, loss: -11361.70
Epoch 31, batch: 400/469, loss: -11146.33
Epoch 32, batch: 0/469, loss: -11312.34
Epoch 32, batch: 400/469, loss: -11254.66
Epoch 33, batch: 0/469, loss: -11167.90
Epoch 33, batch: 400/469, loss: -11129.65
Epoch 34, batch: 0/469, loss: -11245.61
Epoch 34, batch: 400/469, loss: -11304.05
Epoch 35, batch: 0/469, loss: -11225.67
Epoch 35, batch: 400/469, loss: -11278.12
Epoch 36, batch: 0/469, loss: -11219.12
Epoch 36, batch: 400/469, loss: -11254.66
Epoch 37, batch: 0/469, loss: -11274.40
Epoch 37, batch: 400/469, loss: -11127.90
Epoch 38, batch: 0/469, loss: -11397.62
Epoch 38, batch: 400/469, loss: -11167.00
Epoch 39, batch: 0/469, loss: -11222.82
```

```
Epoch 39, batch: 400/469, loss: -11233.03
Epoch 40, batch: 0/469, loss: -11230.74
Epoch 40, batch: 400/469, loss: -11221.88
Epoch 41, batch: 0/469, loss: -11192.66
Epoch 41, batch: 400/469, loss: -11409.42
Epoch 42, batch: 0/469, loss: -11315.82
Epoch 42, batch: 400/469, loss: -11241.08
Epoch 43, batch: 0/469, loss: -11283.51
Epoch 43, batch: 400/469, loss: -11255.81
Epoch 44, batch: 0/469, loss: -11337.77
Epoch 44, batch: 400/469, loss: -11398.83
Epoch 45, batch: 0/469, loss: -11241.06
Epoch 45, batch: 400/469, loss: -11165.84
Epoch 46, batch: 0/469, loss: -11187.24
Epoch 46, batch: 400/469, loss: -11428.44
Epoch 47, batch: 0/469, loss: -11288.18
Epoch 47, batch: 400/469, loss: -11089.81
Epoch 48, batch: 0/469, loss: -11167.10
Epoch 48, batch: 400/469, loss: -11120.99
Epoch 49, batch: 0/469, loss: -11163.09
Epoch 49, batch: 400/469, loss: -11015.25
Epoch 50, batch: 0/469, loss: -11234.59
Epoch 50, batch: 400/469, loss: -11310.05
Epoch 51, batch: 0/469, loss: -11319.05
Epoch 51, batch: 400/469, loss: -11188.44
Epoch 52, batch: 0/469, loss: -11366.91
Epoch 52, batch: 400/469, loss: -11375.14
Epoch 53, batch: 0/469, loss: -11237.67
Epoch 53, batch: 400/469, loss: -11245.77
Epoch 54, batch: 0/469, loss: -11323.61
Epoch 54, batch: 400/469, loss: -11415.95
Epoch 55, batch: 0/469, loss: -11275.73
Epoch 55, batch: 400/469, loss: -11351.73
Epoch 56, batch: 0/469, loss: -11306.99
Epoch 56, batch: 400/469, loss: -11206.51
Epoch 57, batch: 0/469, loss: -11523.84
Epoch 57, batch: 400/469, loss: -11063.80
Epoch 58, batch: 0/469, loss: -11176.46
Epoch 58, batch: 400/469, loss: -11262.54
Epoch 59, batch: 0/469, loss: -11485.71
Epoch 59, batch: 400/469, loss: -11450.83
Epoch 60, batch: 0/469, loss: -11154.47
Epoch 60, batch: 400/469, loss: -11235.35
Epoch 61, batch: 0/469, loss: -11256.90
Epoch 61, batch: 400/469, loss: -11280.53
Epoch 62, batch: 0/469, loss: -11368.22
Epoch 62, batch: 400/469, loss: -11356.50
Epoch 63, batch: 0/469, loss: -11183.44
```

```
Epoch 63, batch: 400/469, loss: -11238.67
Epoch 64, batch: 0/469, loss: -11177.21
Epoch 64, batch: 400/469, loss: -11338.27
Epoch 65, batch: 0/469, loss: -11106.60
Epoch 65, batch: 400/469, loss: -11321.24
Epoch 66, batch: 0/469, loss: -11308.42
Epoch 66, batch: 400/469, loss: -11249.77
Epoch 67, batch: 0/469, loss: -11173.26
Epoch 67, batch: 400/469, loss: -11161.37
Epoch 68, batch: 0/469, loss: -11095.59
Epoch 68, batch: 400/469, loss: -11343.83
Epoch 69, batch: 0/469, loss: -11210.80
Epoch 69, batch: 400/469, loss: -11190.92
Epoch 70, batch: 0/469, loss: -11217.82
Epoch 70, batch: 400/469, loss: -11375.70
Epoch 71, batch: 0/469, loss: -11367.38
Epoch 71, batch: 400/469, loss: -11419.90
Epoch 72, batch: 0/469, loss: -11227.61
Epoch 72, batch: 400/469, loss: -11320.68
Epoch 73, batch: 0/469, loss: -11225.68
Epoch 73, batch: 400/469, loss: -11178.75
Epoch 74, batch: 0/469, loss: -11006.33
Epoch 74, batch: 400/469, loss: -11146.27
Epoch 75, batch: 0/469, loss: -11104.97
Epoch 75, batch: 400/469, loss: -11298.81
Epoch 76, batch: 0/469, loss: -11300.87
Epoch 76, batch: 400/469, loss: -11231.55
Epoch 77, batch: 0/469, loss: -11236.34
Epoch 77, batch: 400/469, loss: -11331.01
Epoch 78, batch: 0/469, loss: -11350.61
Epoch 78, batch: 400/469, loss: -11344.79
Epoch 79, batch: 0/469, loss: -11253.50
Epoch 79, batch: 400/469, loss: -11144.26
Epoch 80, batch: 0/469, loss: -11381.91
Epoch 80, batch: 400/469, loss: -11386.23
Epoch 81, batch: 0/469, loss: -11287.51
Epoch 81, batch: 400/469, loss: -11225.97
Epoch 82, batch: 0/469, loss: -11413.30
Epoch 82, batch: 400/469, loss: -11090.85
Epoch 83, batch: 0/469, loss: -11247.80
Epoch 83, batch: 400/469, loss: -11235.80
Epoch 84, batch: 0/469, loss: -11209.45
Epoch 84, batch: 400/469, loss: -11313.77
Epoch 85, batch: 0/469, loss: -11341.40
Epoch 85, batch: 400/469, loss: -11210.36
Epoch 86, batch: 0/469, loss: -11321.11
Epoch 86, batch: 400/469, loss: -11160.73
Epoch 87, batch: 0/469, loss: -11396.13
```

```
Epoch 87, batch: 400/469, loss: -11193.14
Epoch 88, batch: 0/469, loss: -11210.42
Epoch 88, batch: 400/469, loss: -11237.79
Epoch 89, batch: 0/469, loss: -11529.76
Epoch 89, batch: 400/469, loss: -11192.57
Epoch 90, batch: 0/469, loss: -11193.83
Epoch 90, batch: 400/469, loss: -11222.44
Epoch 91, batch: 0/469, loss: -11452.07
Epoch 91, batch: 400/469, loss: -11340.28
Epoch 92, batch: 0/469, loss: -11194.47
Epoch 92, batch: 400/469, loss: -11223.18
Epoch 93, batch: 0/469, loss: -11094.52
Epoch 93, batch: 400/469, loss: -11234.59
Epoch 94, batch: 0/469, loss: -11228.25
Epoch 94, batch: 400/469, loss: -11226.19
Epoch 95, batch: 0/469, loss: -11188.20
Epoch 95, batch: 400/469, loss: -11255.75
Epoch 96, batch: 0/469, loss: -11136.09
Epoch 96, batch: 400/469, loss: -11356.51
Epoch 97, batch: 0/469, loss: -11080.59
Epoch 97, batch: 400/469, loss: -11180.15
Epoch 98, batch: 0/469, loss: -11255.50
Epoch 98, batch: 400/469, loss: -11310.52
Epoch 99, batch: 0/469, loss: -11286.48
Epoch 99, batch: 400/469, loss: -11204.91
```

```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```



0.8 Reconstructions

```
In [0]: def reconstruction(data,epoch, is_train = True):
            save_dir = "vae-classifier/results/"
            n = min(data.size(0), 8)
            recon_batch, mu, logvar = model(data)
            comparison = torch.cat([data[:n],recon_batch[:n]])
            if is_train:
                name = save_dir + "train_reconstruction_" + str(epoch) + '.png'
            else:
                name = save_dir + "test_reconstruction_" + str(epoch) + '.png'
            save_image(comparison.cpu(),name, nrow=n)


        train_batch, train_label = next(iter(train_loader))
        test_batch, test_label = next(iter(test_loader))
        reconstruction(train_batch.to(device), 100, True)
        reconstruction(test_batch.to(device), 100, False)
```

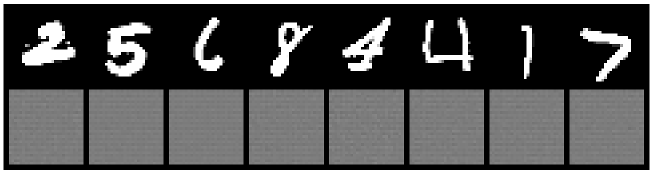
```
In [0]: %pwd
```

```
Out[0]: '/home/chivukula_manju/vae'
```

0.8.1 Train Reconstruction

```
In [0]: from IPython.display import Image
        Image(filename='./vae-classifier/results/train_reconstruction_100.png')
```

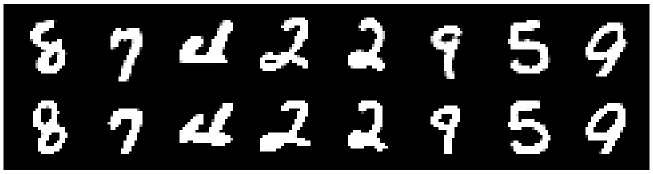
Out[0]:



0.8.2 Test Reconstruction

```
In [0]: Image(filename='./vae-classifier/results/test_reconstruction_100.png')
```

Out[0]:

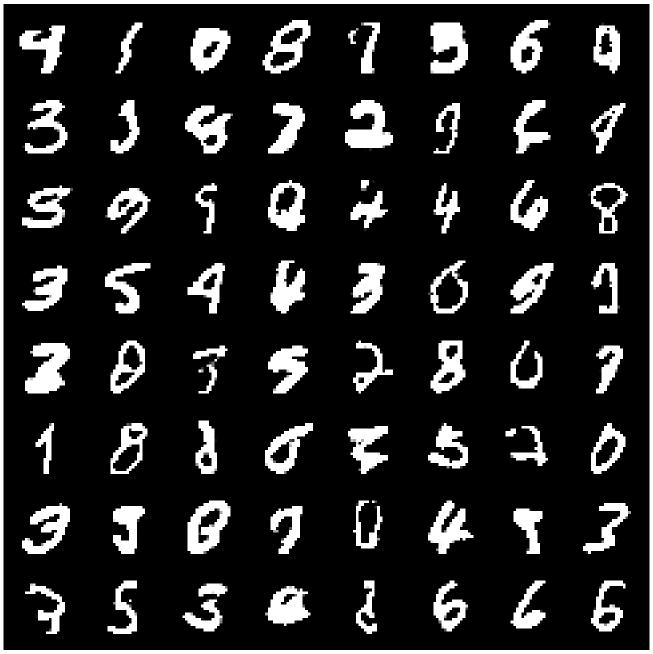


0.9 Generated Samples

```
In [0]: epoch = 100
        with torch.no_grad():
            sample = torch.randn(64, Params.nb_latents).to(device)
            sample = model.decode(sample).cpu()
            save_image(sample.view(64, 1, 28, 28),
                       './vae-classifier/results/sample_' + str(epoch) + '.png')
```

```
In [0]: from IPython.display import Image
        Image(filename='./vae-classifier/results/sample_100.png')
```

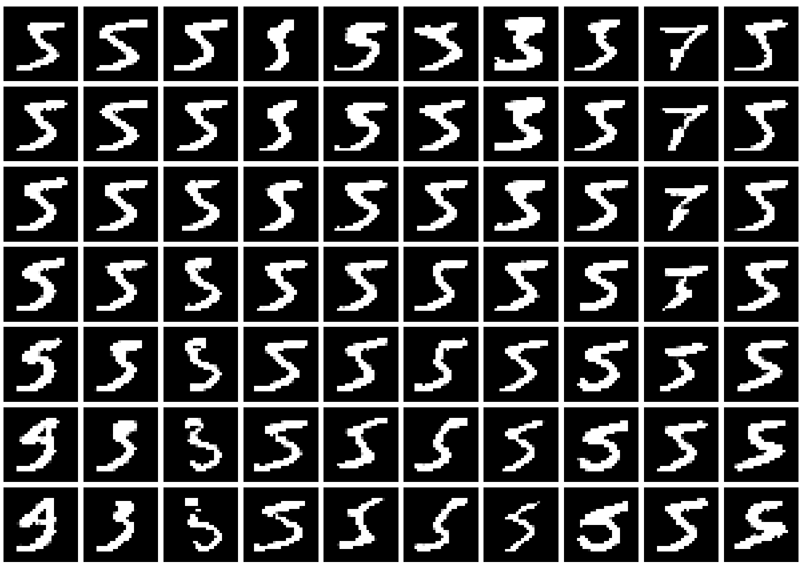
Out[0]:



0.10 Latent Traversals

```
In [0]: Image(filename='./vae-classifier/results/traversal_99_468.png')
```

Out[0]:



```
In [0]:
```

0.11 TSNE Plot

```
In [0]: path = 'latent_space.png'
        def visualize_tsne(X, labels, model, path):
            # Compute latent space representation
            print("Computing latent space projection...")

            X_encoded, _ = model.encode(X)

            # Compute t-SNE embedding of latent space
            tsne = manifold.TSNE(n_components=2, init='pca', random_state=0)
            X_tsne = tsne.fit_transform(X_encoded.data.detach().cpu())

            # Plot images according to t-sne embedding
            fig, ax = plt.subplots()

            plt.scatter(X_tsne[:,0], X_tsne[:,1], c=labels, cmap=plt.cm.get_cmap("viridis", 10)
            plt.colorbar(ticks=range(10))
            fig.savefig(path, dpi=fig.dpi)

        #visualize_tsne(test_data[:5000].to(device),test_labels[:5000],model,path)
```

0.12 Loss Plot

```
In [0]: plt.plot(range(Params.epochs),train_losses)
        plt.plot(range(Params.epochs),test_losses)
        plt.legend(["Train loss","Test loss"])
        plt.show()
        save_model(model,epoch)
```

```
In [0]:
```

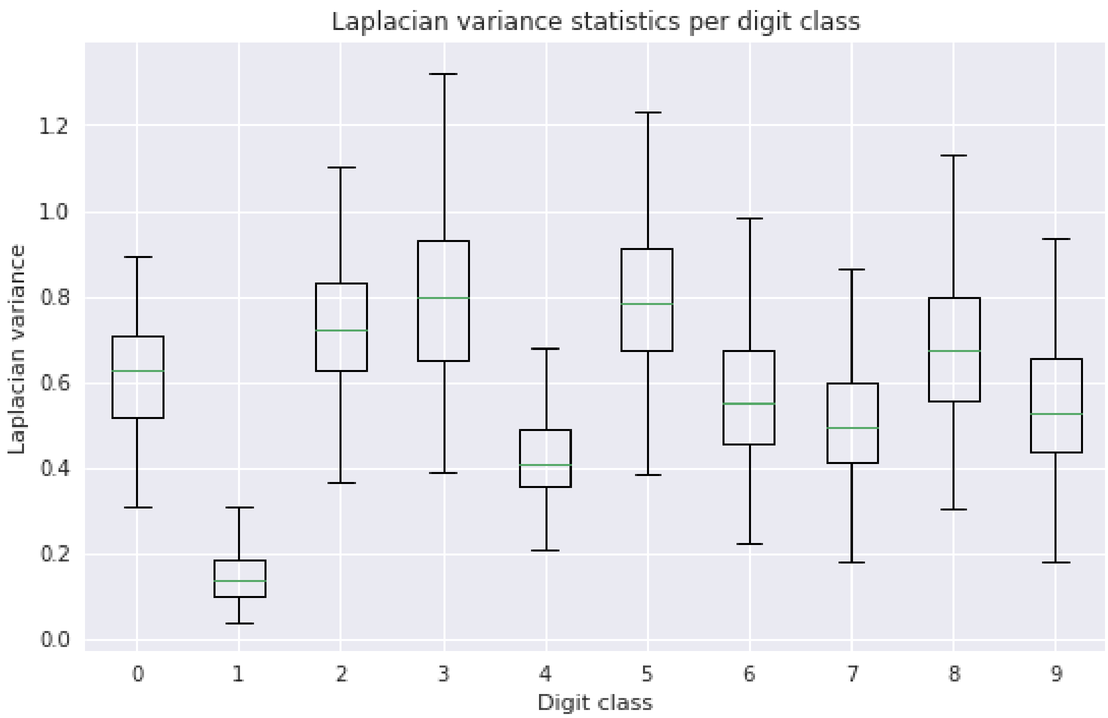
0.13 Load model

```
In [0]: load_model(model,"./vae-classifier/results/model_state_99.pth")
```

```
In [0]: x_test, y_test = test_data[:2000], test_labels[:2000]
        laplacian_variances = [laplacian_variance(x_test[y_test == i]) for i in range(10)]
```

```
In [0]: plt.boxplot(laplacian_variances, labels=range(10));
        plt.xlabel('Digit class')
        plt.ylabel('Laplacian variance')
        plt.title('Laplacian variance statistics per digit class');
```

```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```

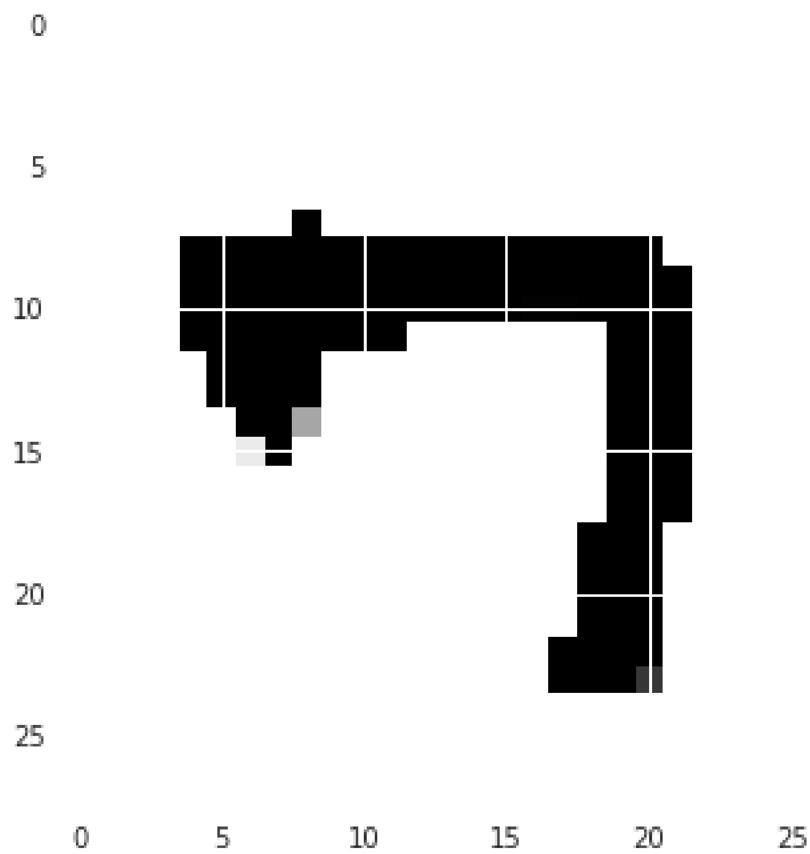



```
In [0]: not_ones = y_test != 1
        x_test_not_ones, y_test_not_ones = x_test[not_ones], y_test[not_ones]
        with torch.no_grad():
            reconstructions = np.empty(shape=(len(x_test_not_ones),1,28,28))
            indx = 0
            for i, (x,y) in enumerate(zip(x_test_not_ones,y_test_not_ones)):
                if y != 1:
                    recon_batch, mu, logvar = model(x.unsqueeze(1).to(device))
                    reconstructions[indx]=(recon_batch.squeeze(0).detach().cpu())
                    indx+=1
```

```
In [0]: plt.imshow(np.array(reconstructions)[1].squeeze())
```

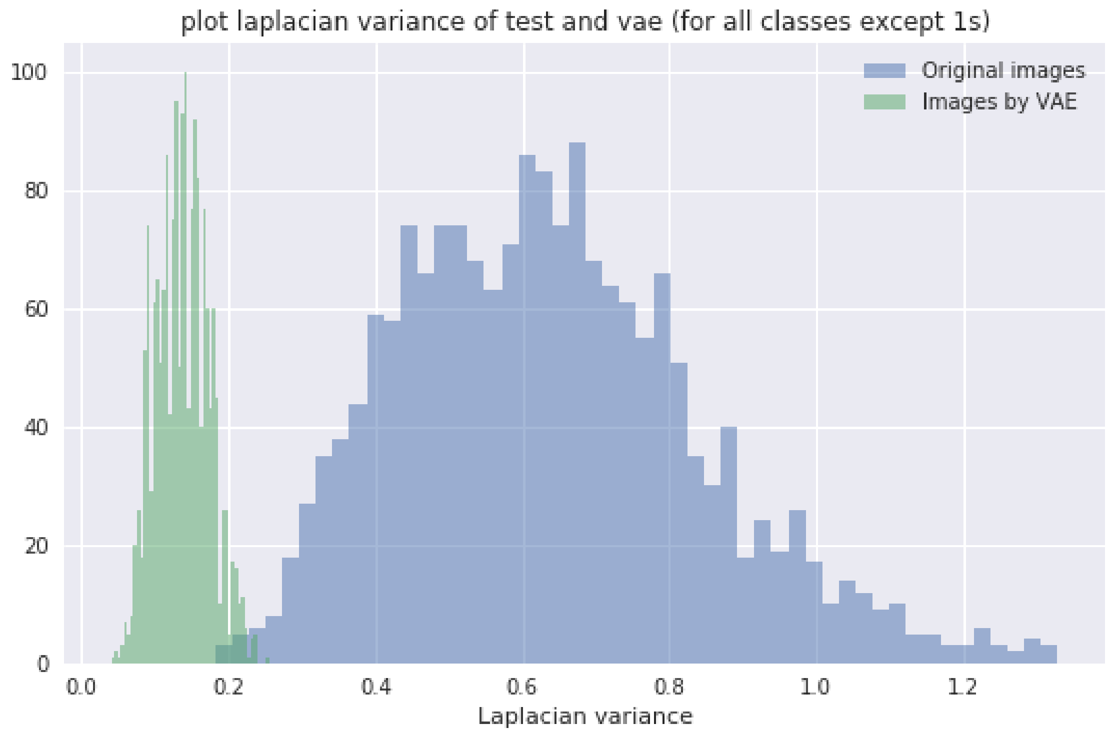
```
Out[0]: <matplotlib.image.AxesImage at 0x7f892f7d9240>
```

```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```



```
In [0]: laplacian_variances_recons = [laplacian_variance_numpy(np.array(reconstructions[y_test
                                     for i in range(10)]

In [0]: not_ones = y_test != 1

        lvs_1 = laplacian_variance(x_test[not_ones])
        lvs_2 = laplacian_variance_numpy(np.array(reconstructions, dtype=np.uint8))
        def plot_laplacian_variances(lvs_1, lvs_2, title):
            plt.hist(lvs_1, bins=50, alpha=0.5 , label='Original images');
            plt.hist(lvs_2, bins=50, alpha= 0.5 , label='Images by VAE');
            plt.xlabel('Laplacian variance')
            plt.title(title)
            plt.legend();
```

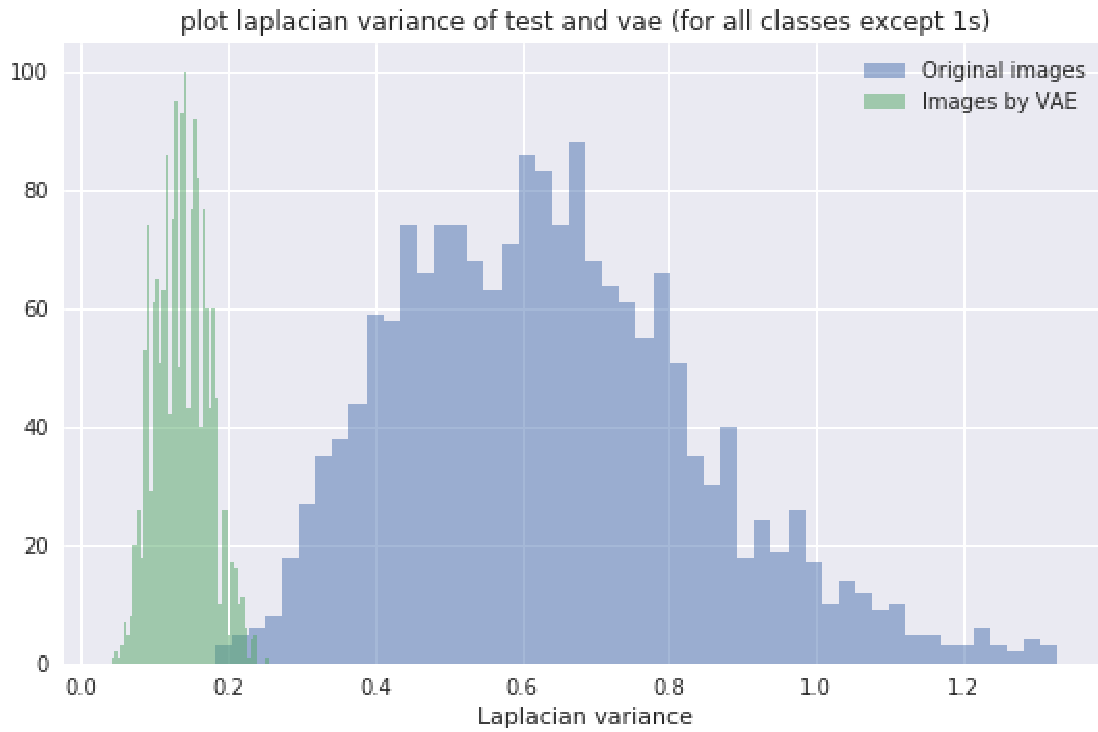
```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```



The Laplacian variance increases with increased focus of an image or decreases with increased blur.Furthermore, images with a smaller amount of edges tend to have a smaller Laplacian variance (the Laplacian kernel is often used for edge detection in images). Therefore we first have to analyze the Laplacian variances for digit classes 0-9 in the MNIST test set before we can compare blur differences in generated images:

```
In [0]: plot_laplacian_variances(lvs_1, lvs_2, "plot laplacian variance of test and vae (for al
```

```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```



0.14 Model 2

- VAE is detached
- Test Classifier

```
In [0]: torch.manual_seed(5)
```

```
Out[0]: <torch._C.Generator at 0x7f89605ce030>
```

```
In [0]: class Classifier(nn.Module):
          def __init__(self):
            super(Classifier,self).__init__()

            ## Define NN
            self.fc1 = nn.Linear(10, 10)


          def forward(self,x):
            ## flat input features
            x = x.view(-1, self.num_flat_features(x))
            x = self.fc1(x)

            return F.log_softmax(x, dim=1)
```

```
    def num_flat_features(self,x):
      size = x.size()[1:] # all dimensions except the batch dimension
      num_features = 1
      for s in size:
        num_features *=s
      return num_features
```

In [0]:
```
## Training
from tqdm import trange

criterion = nn.CrossEntropyLoss()
classifier = Classifier()
if is_cuda:
  classifier = classifier.to(device)

# Loss and optimizer
learning_rate = 0.001
momemtum = 0.9
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(classifier.parameters(), lr=learning_rate)
scheduler = lr_scheduler.StepLR(optimizer,step_size = 7,gamma = 0.1)
print(classifier)
```

```
Classifier(
  (fc1): Linear(in_features=10, out_features=10, bias=True)
)
```

In [0]:
```
## Train Classifier with pretrained vae
#vae_parameters = list(model.named_parameters())
#for name, param in vae_parameters:
 #   param.requires_grad = False
```

In [0]:
```
def train_classifier_epoch(epoch):
  model.train()
  metric = AccumulatedAccuracyMetric()
  losses = RunningAverage()
  for idx, (data, labels) in enumerate(train_loader):
    data= data.to(device)
    labels = labels.to(device)


    recon_batch, mu, logvar = model(data)
    ## classifier, pass latent vector
    outputs = classifier(mu)
    classifier_loss = criterion(outputs, labels)

    optimizer.zero_grad()
    classifier_loss.backward()
```

```
      optimizer.step()

      classifier_loss /= data.size(0)
      losses.update(classifier_loss)

      metric(outputs, labels)


    return losses(), metric
```

In [0]:
```
## Test Epoch
"""
Test, classifier on learnt features
"""
def test_classifier_epoch(epoch):
  classifier.eval()
  metric = AccumulatedAccuracyMetric()
  losses = RunningAverage()
  for idx, (data, labels) in enumerate(test_loader):
    data= data.to(device)
    labels = labels.to(device)


    recon_batch, mu, logvar = model(data)
    ## classifier, pass latent vector
    outputs = classifier(mu)
    classifier_loss = criterion(outputs, labels)

    classifier_loss /= data.size(0)
    losses.update(classifier_loss)

    metric(outputs, labels)


  return losses(), metric
```

In [0]:
```
train_losses = []
train_accuracy = []
test_losses = []
test_accuracy = []
n_epochs = 50
for epoch in range(1, n_epochs):

  # Train stage
  train_loss, metric = train_classifier_epoch(epoch)
  train_losses.append(train_loss)
  train_accuracy.append(metric.value())
```

```
        message = 'Epoch: {}/{}. Train set: Average loss: {:.4f}'
          .format(epoch + 1, n_epochs, train_loss)
        message += '\t Average Accuracy: \t{}: {}'
          .format(metric.name(), metric.value())
        print(message)

        val_loss, metrics = test_classifier_epoch(epoch)
        test_losses.append(val_loss)
        test_accuracy.append(metrics.value())

        message += '\nEpoch: {}/{}. Test set: Average loss: {:.4f}'
          .format(epoch + 1, n_epochs,val_loss)

        message += '\t Average Accuracy: \t{}: {}'
          .format(metrics.name(), metrics.value())


        print(message)
```

```
Epoch: 2/50. Train set: Average loss: 0.0122         Average Accuracy:          Accuracy: 56.773
Epoch: 2/50. Train set: Average loss: 0.0122         Average Accuracy:          Accuracy: 56.773
Epoch: 2/50. Test set: Average loss: 0.0077         Average Accuracy:          Accuracy: 87.38
Epoch: 3/50. Train set: Average loss: 0.0053         Average Accuracy:          Accuracy: 90.18
Epoch: 3/50. Train set: Average loss: 0.0053         Average Accuracy:          Accuracy: 90.18
Epoch: 3/50. Test set: Average loss: 0.0045         Average Accuracy:          Accuracy: 92.66
Epoch: 4/50. Train set: Average loss: 0.0034         Average Accuracy:          Accuracy: 93.075
Epoch: 4/50. Train set: Average loss: 0.0034         Average Accuracy:          Accuracy: 93.075
Epoch: 4/50. Test set: Average loss: 0.0033         Average Accuracy:          Accuracy: 93.87
Epoch: 5/50. Train set: Average loss: 0.0026         Average Accuracy:          Accuracy: 94.095
Epoch: 5/50. Train set: Average loss: 0.0026         Average Accuracy:          Accuracy: 94.095
Epoch: 5/50. Test set: Average loss: 0.0026         Average Accuracy:          Accuracy: 94.5
Epoch: 6/50. Train set: Average loss: 0.0022         Average Accuracy:          Accuracy: 94.696
Epoch: 6/50. Train set: Average loss: 0.0022         Average Accuracy:          Accuracy: 94.696
Epoch: 6/50. Test set: Average loss: 0.0022         Average Accuracy:          Accuracy: 94.83
Epoch: 7/50. Train set: Average loss: 0.0020         Average Accuracy:          Accuracy: 94.986
Epoch: 7/50. Train set: Average loss: 0.0020         Average Accuracy:          Accuracy: 94.986
Epoch: 7/50. Test set: Average loss: 0.0020         Average Accuracy:          Accuracy: 95.06
Epoch: 8/50. Train set: Average loss: 0.0018         Average Accuracy:          Accuracy: 95.215
Epoch: 8/50. Train set: Average loss: 0.0018         Average Accuracy:          Accuracy: 95.215
Epoch: 8/50. Test set: Average loss: 0.0017         Average Accuracy:          Accuracy: 95.21
Epoch: 9/50. Train set: Average loss: 0.0017         Average Accuracy:          Accuracy: 95.39
Epoch: 9/50. Train set: Average loss: 0.0017         Average Accuracy:          Accuracy: 95.39
Epoch: 9/50. Test set: Average loss: 0.0017         Average Accuracy:          Accuracy: 95.3
Epoch: 10/50. Train set: Average loss: 0.0016         Average Accuracy:          Accuracy: 95.48
Epoch: 10/50. Train set: Average loss: 0.0016         Average Accuracy:          Accuracy: 95.48
Epoch: 10/50. Test set: Average loss: 0.0016         Average Accuracy:          Accuracy: 95.41
Epoch: 11/50. Train set: Average loss: 0.0015         Average Accuracy:          Accuracy: 95.56
```

```
Epoch: 11/50. Train set: Average loss: 0.0015          Average Accuracy:          Accuracy: 95.56
Epoch: 11/50. Test set: Average loss: 0.0017          Average Accuracy:          Accuracy: 95.46
Epoch: 12/50. Train set: Average loss: 0.0015          Average Accuracy:          Accuracy: 95.63
Epoch: 12/50. Train set: Average loss: 0.0015          Average Accuracy:          Accuracy: 95.63
Epoch: 12/50. Test set: Average loss: 0.0015          Average Accuracy:          Accuracy: 95.51
Epoch: 13/50. Train set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.70
Epoch: 13/50. Train set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.70
Epoch: 13/50. Test set: Average loss: 0.0015          Average Accuracy:          Accuracy: 95.53
Epoch: 14/50. Train set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.75
Epoch: 14/50. Train set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.75
Epoch: 14/50. Test set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.59
Epoch: 15/50. Train set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.77
Epoch: 15/50. Train set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.77
Epoch: 15/50. Test set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.58
Epoch: 16/50. Train set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.84
Epoch: 16/50. Train set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.84
Epoch: 16/50. Test set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.61
Epoch: 17/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.80
Epoch: 17/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.80
Epoch: 17/50. Test set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.64
Epoch: 18/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.83
Epoch: 18/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.83
Epoch: 18/50. Test set: Average loss: 0.0017          Average Accuracy:          Accuracy: 95.61
Epoch: 19/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.87
Epoch: 19/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.87
Epoch: 19/50. Test set: Average loss: 0.0016          Average Accuracy:          Accuracy: 95.67
Epoch: 20/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.88
Epoch: 20/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.88
Epoch: 20/50. Test set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.67
Epoch: 21/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.9
Epoch: 21/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.9
Epoch: 21/50. Test set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.71
Epoch: 22/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.89
Epoch: 22/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.89
Epoch: 22/50. Test set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.73
Epoch: 23/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
Epoch: 23/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
Epoch: 23/50. Test set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.76
Epoch: 24/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
Epoch: 24/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
Epoch: 24/50. Test set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.73
Epoch: 25/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
Epoch: 25/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
Epoch: 25/50. Test set: Average loss: 0.0014          Average Accuracy:          Accuracy: 95.76
Epoch: 26/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
Epoch: 26/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
Epoch: 26/50. Test set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.82
Epoch: 27/50. Train set: Average loss: 0.0013          Average Accuracy:          Accuracy: 95.90
```

```
Epoch: 27/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.90
Epoch: 27/50. Test set: Average loss: 0.0014         Average Accuracy:         Accuracy: 95.79
Epoch: 28/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 28/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 28/50. Test set: Average loss: 0.0014         Average Accuracy:         Accuracy: 95.81
Epoch: 29/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.90
Epoch: 29/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.90
Epoch: 29/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.84
Epoch: 30/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.90
Epoch: 30/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.90
Epoch: 30/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.82
Epoch: 31/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 31/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 31/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.78
Epoch: 32/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 32/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 32/50. Test set: Average loss: 0.0015         Average Accuracy:         Accuracy: 95.81
Epoch: 33/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.91
Epoch: 33/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.91
Epoch: 33/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.84
Epoch: 34/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.93
Epoch: 34/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.93
Epoch: 34/50. Test set: Average loss: 0.0014         Average Accuracy:         Accuracy: 95.81
Epoch: 35/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 35/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 35/50. Test set: Average loss: 0.0016         Average Accuracy:         Accuracy: 95.82
Epoch: 36/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.94
Epoch: 36/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.94
Epoch: 36/50. Test set: Average loss: 0.0017         Average Accuracy:         Accuracy: 95.83
Epoch: 37/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.93
Epoch: 37/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.93
Epoch: 37/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.83
Epoch: 38/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.94
Epoch: 38/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.94
Epoch: 38/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.84
Epoch: 39/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 39/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 39/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.83
Epoch: 40/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.94
Epoch: 40/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.94
Epoch: 40/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.84
Epoch: 41/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.93
Epoch: 41/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.93
Epoch: 41/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.82
Epoch: 42/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 42/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.92
Epoch: 42/50. Test set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.84
Epoch: 43/50. Train set: Average loss: 0.0013         Average Accuracy:         Accuracy: 95.91
```

```
Epoch: 43/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 43/50. Test set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.85
Epoch: 44/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 44/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 44/50. Test set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.87
Epoch: 45/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 45/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 45/50. Test set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.86
Epoch: 46/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 46/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 46/50. Test set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.85
Epoch: 47/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 47/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 47/50. Test set: Average loss: 0.0014        Average Accuracy:         Accuracy: 95.83
Epoch: 48/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 48/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 48/50. Test set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.84
Epoch: 49/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 49/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 49/50. Test set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.83
Epoch: 50/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 50/50. Train set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.9
Epoch: 50/50. Test set: Average loss: 0.0013        Average Accuracy:         Accuracy: 95.88
```

0.14.1 Save Classifier

```
In [0]: path ="classifier_model_state_"+str(epoch)+".pth"
        torch.save(classifier.state_dict(),path)
```

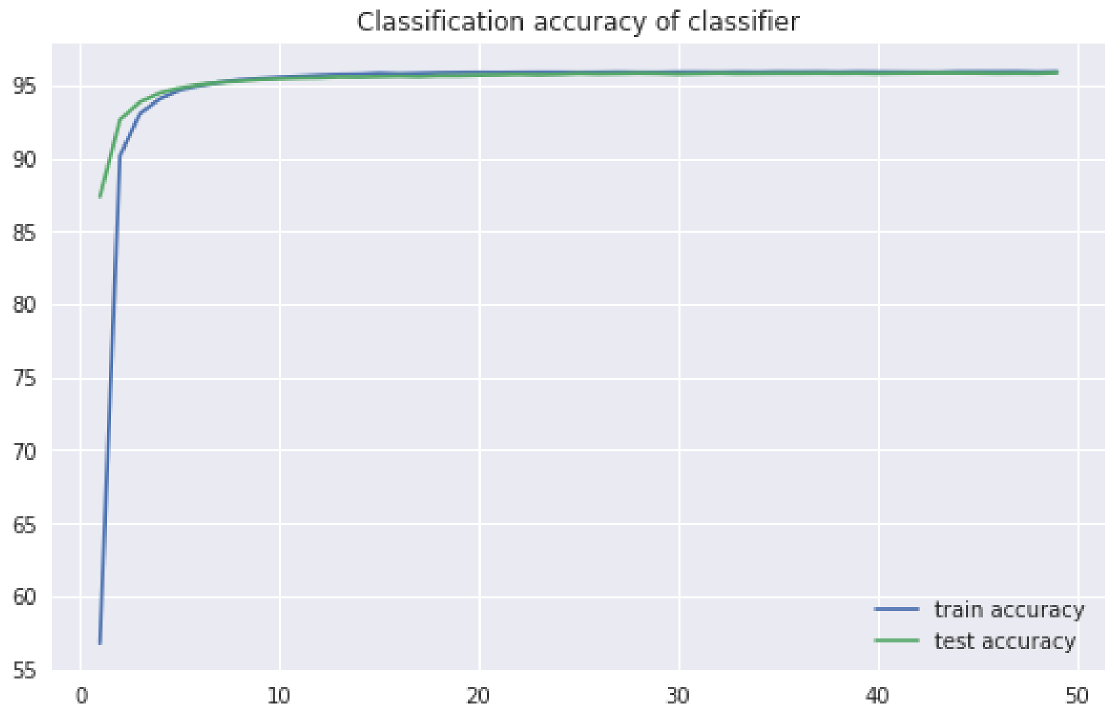
```
In [0]: path ="classifier_model_state_"+str(49)+".pth"
        load_model(classifier, path)
```

0.15 Plots

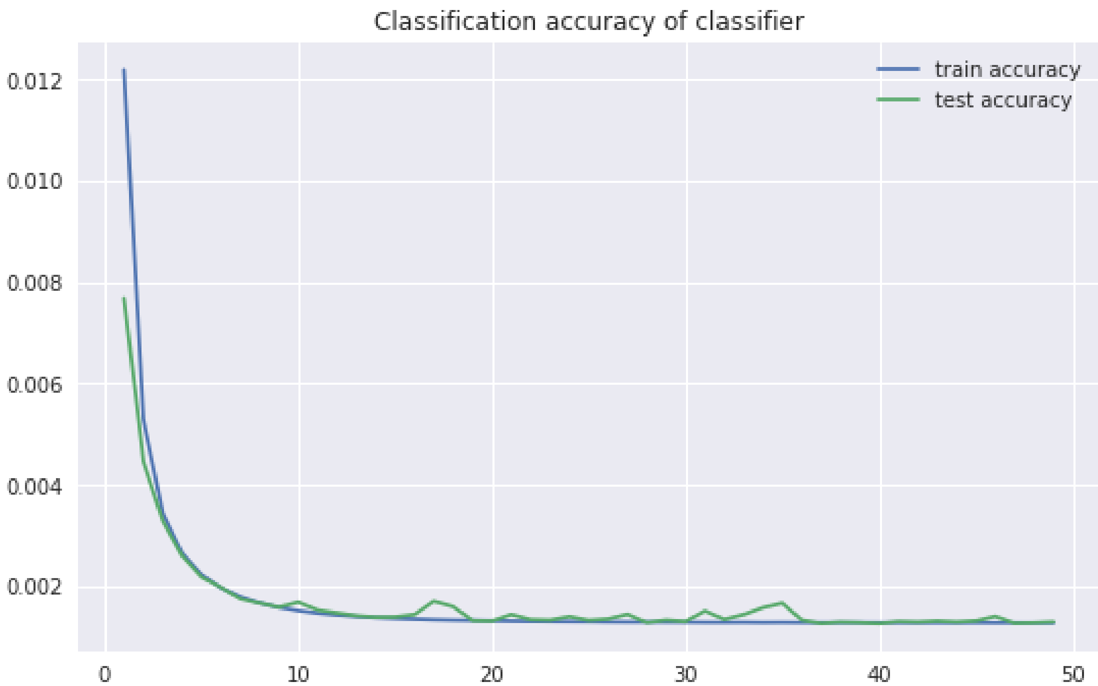
```
In [0]: plt.plot(range(1,n_epochs),train_accuracy)
        plt.plot(range(1,n_epochs),test_accuracy)
        plt.title("Classification accuracy of classifier")
        plt.legend(["train accuracy","test accuracy"])
```

```
Out[0]: <matplotlib.legend.Legend at 0x7f9186880eb8>
```

```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWar
  (prop.get_family(), self.defaultFamily[fontext]))
```



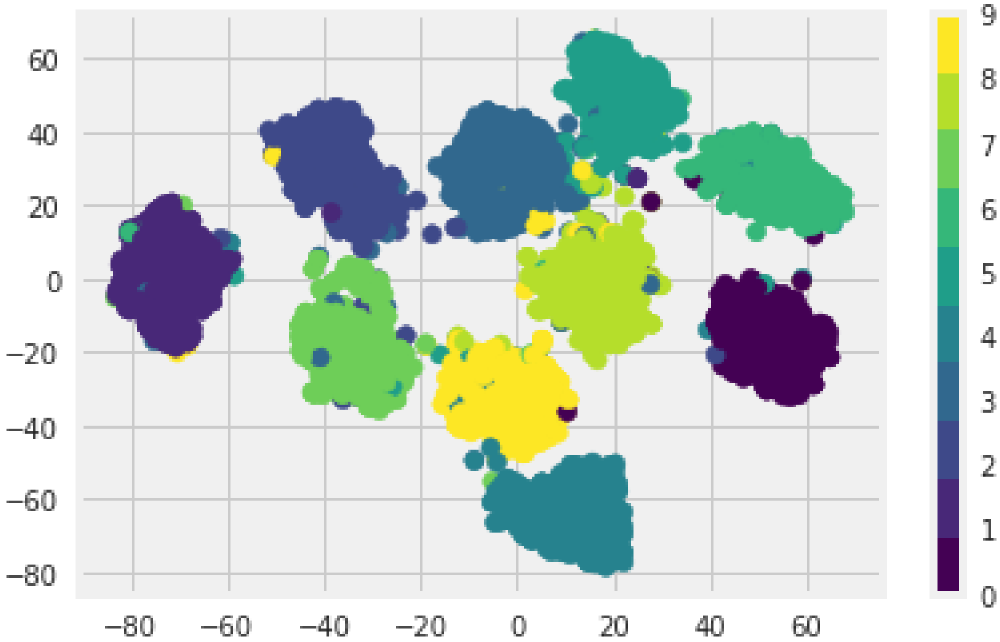

```
In [0]: plt.plot(range(1,n_epochs),train_losses)
        plt.plot(range(1,n_epochs),test_losses)
        plt.title("Classification loss of classifier")
        plt.legend(["train accuracy","test accuracy"])

Out[0]: <matplotlib.legend.Legend at 0x7f91867d94a8>

/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```




```
In [0]: path = "tsne_vae+classifier_trained.png"
        visualize_tsne(test_data[:5000].to(device),test_labels[:5000],model,path)
```

```
Computing latent space projection...
```

```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```



0.16 Model 3

Train with: VAE + Classifier

In [0]:

In [0]:
```
classifier = Classifier()
model = ConvVAE(Params.nb_latents)

if is_cuda:
  classifier = classifier.to(device)
  model   = model.cuda()

## VAE is trained
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(list(model.parameters())
                    + list(classifier.parameters()), lr=1e-3)

"""
Train VAE + but not classifier
"""
def train_classifier_all(epoch):

  model.train()
  classifier.train()
  metric = AccumulatedAccuracyMetric()
```

```python
  losses = RunningAverage()
  for idx, (data, labels) in enumerate(train_loader):

    data = data.to(device)
    labels = labels.to(device)

    recon_batch, mu, logvar = model(data)
    kld, loss = loss_function(recon_batch.squeeze().view(-1,28*28), data, mu, logvar)

    ## classifier, pass latent vector
    outputs = classifier(mu)
    classifier_loss =  criterion(outputs, labels)

    ## Add all losses.
    loss = loss + classifier_loss

    ## parameter update
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    loss /=len(data)

    losses.update(loss)

    metric(outputs, labels)


  return losses(), metric



## Test Epoch
"""
Test, classifier on learnt features
"""
def test_with_all_training(epoch):
  classifier.eval()
  model.eval()
  metric = AccumulatedAccuracyMetric()
  #losses = RunningAverage()
  for idx, (data, labels) in enumerate(test_loader):

    data, labels = data.to(device), labels.to(device)

    recon_batch, mu, logvar = model(data)

    ## classifier, pass latent vector
```

```
        outputs = classifier(mu)

        metric(outputs, labels)


      return 0.0, metric
```

In [0]:
```
import warnings

warnings.filterwarnings("ignore")
train_losses = []
train_accuracy = []
test_losses = []
test_accuracy = []
n_epochs = 50
for epoch in range(1, n_epochs):

  # Train stage
  train_loss, metric = train_classifier_all(epoch)
  train_losses.append(train_loss)
  train_accuracy.append(metric.value())

  message = 'Epoch: {}/{}. Train set: Average loss: {:.4f}'
    .format(epoch + 1, n_epochs, train_loss)
  message += '\t Average Accuracy: \t{}: {}'
    .format(metric.name(), metric.value())
  print(message)

  val_loss, metrics = test_with_all_training(epoch)
  test_losses.append(val_loss)
  test_accuracy.append(metrics.value())

  message += '\nEpoch: {}/{}. Test set: Average loss: {:.4f}'
    .format(epoch + 1, n_epochs, val_loss)

  message += '\t Average Accuracy: \t{}: {}'
    .format(metrics.name(), metrics.value())


  print(message)
```

```
Epoch: 2/50. Train set: Average loss: -7092.0972        Average Accuracy:        Accuracy: 20
Epoch: 2/50. Train set: Average loss: -7092.0972        Average Accuracy:        Accuracy: 20
Epoch: 2/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 46.78
Epoch: 3/50. Train set: Average loss: -8804.5527        Average Accuracy:        Accuracy: 63
Epoch: 3/50. Train set: Average loss: -8804.5527        Average Accuracy:        Accuracy: 63
Epoch: 3/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 73.54
Epoch: 4/50. Train set: Average loss: -9690.9590        Average Accuracy:        Accuracy: 78
```

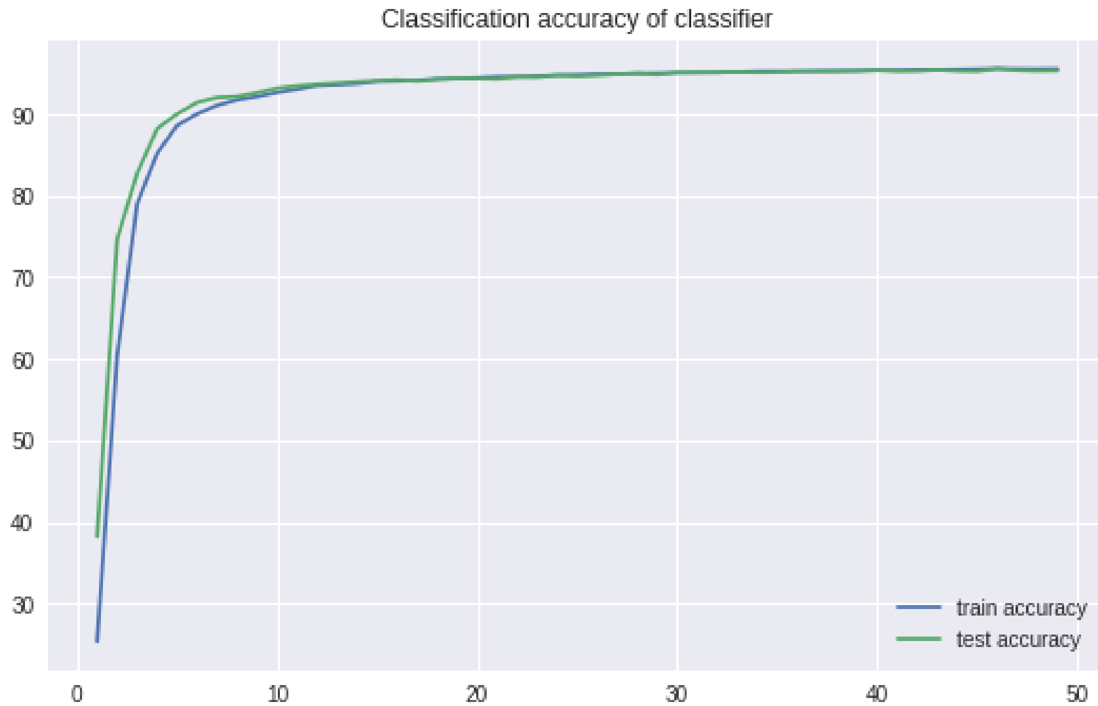
```
Epoch: 4/50. Train set: Average loss: -9690.9590        Average Accuracy:         Accuracy: 78
Epoch: 4/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 83.35
Epoch: 5/50. Train set: Average loss: -10132.9014        Average Accuracy:         Accuracy: 8
Epoch: 5/50. Train set: Average loss: -10132.9014        Average Accuracy:         Accuracy: 8
Epoch: 5/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 86.94
Epoch: 6/50. Train set: Average loss: -10435.4551        Average Accuracy:         Accuracy: 8
Epoch: 6/50. Train set: Average loss: -10435.4551        Average Accuracy:         Accuracy: 8
Epoch: 6/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 90.46
Epoch: 7/50. Train set: Average loss: -10606.8926        Average Accuracy:         Accuracy: 9
Epoch: 7/50. Train set: Average loss: -10606.8926        Average Accuracy:         Accuracy: 9
Epoch: 7/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 91.98
Epoch: 8/50. Train set: Average loss: -10767.1396        Average Accuracy:         Accuracy: 9
Epoch: 8/50. Train set: Average loss: -10767.1396        Average Accuracy:         Accuracy: 9
Epoch: 8/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 92.47
Epoch: 9/50. Train set: Average loss: -10905.2227        Average Accuracy:         Accuracy: 9
Epoch: 9/50. Train set: Average loss: -10905.2227        Average Accuracy:         Accuracy: 9
Epoch: 9/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 92.74
Epoch: 10/50. Train set: Average loss: -10973.8369        Average Accuracy:         Accuracy:
Epoch: 10/50. Train set: Average loss: -10973.8369        Average Accuracy:         Accuracy:
Epoch: 10/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 93.2
Epoch: 11/50. Train set: Average loss: -11020.5469        Average Accuracy:         Accuracy:
Epoch: 11/50. Train set: Average loss: -11020.5469        Average Accuracy:         Accuracy:
Epoch: 11/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 93.64
Epoch: 12/50. Train set: Average loss: -11048.2842        Average Accuracy:         Accuracy:
Epoch: 12/50. Train set: Average loss: -11048.2842        Average Accuracy:         Accuracy:
Epoch: 12/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 94.04
Epoch: 13/50. Train set: Average loss: -11078.0010        Average Accuracy:         Accuracy:
Epoch: 13/50. Train set: Average loss: -11078.0010        Average Accuracy:         Accuracy:
Epoch: 13/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 94.2
Epoch: 14/50. Train set: Average loss: -11096.6172        Average Accuracy:         Accuracy:
Epoch: 14/50. Train set: Average loss: -11096.6172        Average Accuracy:         Accuracy:
Epoch: 14/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 94.43
Epoch: 15/50. Train set: Average loss: -11112.1279        Average Accuracy:         Accuracy:
Epoch: 15/50. Train set: Average loss: -11112.1279        Average Accuracy:         Accuracy:
Epoch: 15/50. Test set: Average loss: 0.0000        Average Accuracy:         Accuracy: 94.64
```

In [0]:

In [0]:
```
plt.plot(range(1,n_epochs),train_losses)
plt.plot(range(1,n_epochs),test_losses)
plt.title("Classification loss of classifier")
plt.legend(["train loss","test loss"])
```

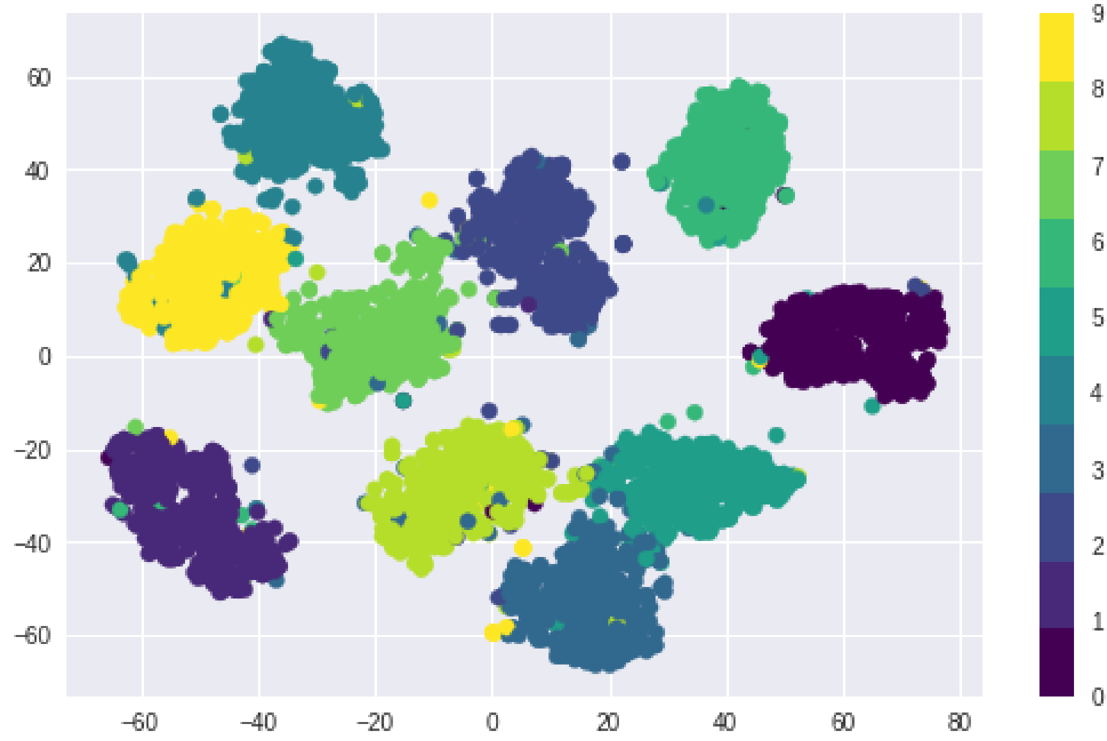
In [0]:
```
plt.style.use("seaborn")
plt.plot(range(1,n_epochs),train_accuracy)
plt.plot(range(1,n_epochs),test_accuracy)
plt.title("Classification accuracy of classifier")
plt.legend(["train accuracy","test accuracy"])
```

Out[0]: <matplotlib.legend.Legend at 0x7f4fcc660198>




```
In [0]: path = "tsne_vae+classifier+vae+train.png"
        visualize_tsne(test_data[:5000].to(device),test_labels[:5000],model,path)
```

```
Computing latent space projection...
```



In [0]: