VAE+GAN

April 26, 2019

0.1 Autoencoding beyond pixels using a learned similarity metric

paper

```
In [0]: from google.colab import drive
        drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/co
```

```
In [0]: %cd /content/drive/My\ Drive/Masters-DS/CSCI-B659/project/vae+gan
```

```
/content/drive/My Drive/Masters-DS/CSCI-B659/project/vae+gan
```

```
In [0]: import os
        #os.makedirs("vae+gan")
        os.makedirs("results")
```

```
In [0]: %ls
```

```
results/
```

0.2 Import Libraries

```
In [0]: from __future__ import print_function
        import argparse
        import h5py
        import numpy as np
        import os
        import time
        import torch
        import torch.utils.data
        import torch.nn as nn
        import torch.optim as optim
        from torch.utils.data.dataset import Dataset
        from torch.autograd import Variable
        from torchvision import datasets, transforms
        from torchvision.utils import make_grid , save_image
        import torchvision.utils as vutils
```

In [0]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def show_and_save(file_name,img,show = False):
    npimg = np.transpose(img.numpy(),(1,2,0))
    f = "./%s.png" % file_name
    fig = plt.figure(dpi=300)
    fig.suptitle(file_name, fontsize=14, fontweight='bold')
    #plt.imshow(npimg)
    if show:
      plt.axis("off")
      plt.imshow(npimg)
    else:
      plt.imsave(f,npimg)

def save_model(epoch, encoder, decoder, D):
    torch.save(decoder.cpu().state_dict(), './VAE_GAN_decoder_%d.pth' % epoch)
    torch.save(encoder.cpu().state_dict(),'./VAE_GAN_encoder_%d.pth' % epoch)
    torch.save(D.cpu().state_dict(), 'VAE_GAN_D_%d.pth' % epoch)
    decoder.cuda()
    encoder.cuda()
    D.cuda()

def load_model(epoch, encoder, decoder, D):
    #  restore models
    decoder.load_state_dict(torch.load('./VAE_GAN_decoder_%d.pth' % epoch))
    decoder.cuda()
    encoder.load_state_dict(torch.load('./VAE_GAN_encoder_%d.pth' % epoch))
    encoder.cuda()
    D.load_state_dict(torch.load('VAE_GAN_D_%d.pth' % epoch))
    D.cuda()
```

In [0]:
```
## Load DataSet
class Params:
  batch_size = 128
  data_dir="../MNIST/data"
  save_dir = "results/"
  nb_latents = 10
```

In [0]:
```
## Data loader
batch_size =  Params.batch_size

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST(Params.data_dir, train=True, download=True,
                   transform=transforms.ToTensor()),
    batch_size=batch_size, shuffle=True)
```

```
        test_loader = torch.utils.data.DataLoader(
            datasets.MNIST(Params.data_dir, train=False, transform=transforms.ToTensor()),
            batch_size=batch_size, shuffle=True)
```

```
0it [00:00, ?it/s]

Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../MNIST/data/MNIST/


9920512it [00:01, 8784119.11it/s]


Extracting ../MNIST/data/MNIST/raw/train-images-idx3-ubyte.gz


  0%|          | 0/28881 [00:00<?, ?it/s]

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../MNIST/data/MNIST/


32768it [00:00, 134750.55it/s]
  0%|          | 0/1648877 [00:00<?, ?it/s]

Extracting ../MNIST/data/MNIST/raw/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../MNIST/data/MNIST/r


1654784it [00:00, 2196044.64it/s]
0it [00:00, ?it/s]

Extracting ../MNIST/data/MNIST/raw/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../MNIST/data/MNIST/r


8192it [00:00, 51421.34it/s]


Extracting ../MNIST/data/MNIST/raw/t10k-labels-idx1-ubyte.gz
Processing...
Done!
```

```
In [0]: data, labels = next(iter(train_loader))
        print(len(data))
        show_and_save("inputs",make_grid(data.cpu(),8),True)
```

```
128
```

inputs

0.3 Networks

0.3.1 Encoder Network

```
In [0]: class Encoder(nn.Module):
            def __init__(self, input_channels, output_channels, representation_size = 32):
                super(Encoder, self).__init__()
                # input parameters
                self.input_channels = input_channels
                self.output_channels = output_channels #Params.nb_latents

                self.features = nn.Sequential(
                    # nc x 32 x 32
                    nn.Conv2d(self.input_channels, representation_size, 5,
                              stride=2, padding=2),
                    nn.BatchNorm2d(representation_size),
                    nn.ReLU(),
                    # hidden_size x 16 x 16
                    nn.Conv2d(representation_size, representation_size*2, 5,
                              stride=2, padding=2),
                    nn.BatchNorm2d(representation_size * 2),
                    nn.ReLU(),
                    # hidden_size*2 x 8 x 8
                    nn.Conv2d(representation_size*2, representation_size*4, 5,
```

```
                    stride=2, padding=2),
                nn.BatchNorm2d(representation_size * 4),
                nn.ReLU())
                # hidden_size*4 x 4 x 4

            self.mean = nn.Sequential(
                nn.Linear(representation_size*4*4*4, 1024),

                nn.ReLU(),
                nn.Linear(1024, output_channels))

            self.logvar = nn.Sequential(
                nn.Linear(representation_size*4*4*4, 1024),

                nn.ReLU(),
                nn.Linear(1024, output_channels))

        def forward(self, x):
            batch_size = x.size()[0]

            hidden_representation = self.features(x)
            hidden_representation = hidden_representation.view(-1,
                                     self.num_flat_features(hidden_representation))

            mean = self.mean(hidden_representation)
            logvar = self.logvar(hidden_representation)

            return mean, logvar

        def hidden_layer(self, x):
            batch_size = x.size()[0]
            output = self.features(x)
            return output


        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
```

0.3.2 Decoder Network

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

```
        self.input_size = input_size ##

        self.fc1 = nn.Linear(self.input_size, 256)
        self.fc2 = nn.Linear(256, 1024)
        self.fc3 = 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)

        # 1 x 28 x 28
        self.activation = nn.Sigmoid()
        self.relu = nn.Tanh()


    def forward(self, z):
        bs = z.size()[0]

        x = self.relu(self.fc1(z))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.deconv1(x.view(-1, 64, 7, 7)))
        x = self.deconv2(x)

        return self.activation(x)
```

0.3.3 VAE - GAN Network

```
In [0]: class VAE_GAN_Generator(nn.Module):
            def __init__(self, input_channels, hidden_size):
                super(VAE_GAN_Generator, self).__init__()
                self.input_channels = input_channels
                self.hidden_size = hidden_size

                self.encoder = Encoder(input_channels, hidden_size)
                self.decoder = Decoder(hidden_size)

            def forward(self, x):
                batch_size = x.size()[0]
                mean, logvar = self.encoder(x)
                std = logvar.mul(0.5).exp_()

                reparametrized_noise = Variable(torch.randn((batch_size,
                                                    self.hidden_size))).cuda()

                reparametrized_noise = mean + std * reparametrized_noise

                rec_images = self.decoder(reparametrized_noise)
```

```
        return mean, logvar, rec_images
```

0.3.4 Discriminator Network

```
In [0]: class Discriminator(nn.Module):
            def __init__(self,input_channels, representation_size = 32):
                super(Discriminator, self).__init__()
                self.input_channels = input_channels
                dim = 128 * 4 * 4
                self.main = nn.Sequential(
                    # nc x 32 x 32
                    nn.Conv2d(self.input_channels, representation_size, 5,
                              stride=2, padding=2),
                    nn.BatchNorm2d(representation_size),
                    nn.LeakyReLU(0.2),
                    # hidden_size x 16 x 16
                    nn.Conv2d(representation_size, representation_size*2, 5,
                              stride=2, padding=2),
                    nn.BatchNorm2d(representation_size * 2),
                    nn.LeakyReLU(0.2),
                    # hidden_size*2 x 8 x 8
                    nn.Conv2d(representation_size*2, representation_size*4, 5,
                              stride=2, padding=2),
                    nn.BatchNorm2d(representation_size * 4),
                    nn.LeakyReLU(0.2))

                self.lth_features = nn.Sequential(
                    nn.Linear(dim, 2048),
                    nn.LeakyReLU(0.2))

                self.sigmoid_output = nn.Sequential(nn.Linear(2048,1),nn.Sigmoid())


            def forward(self, x):
                batch_size = x.size()[0]
                features = self.main(x)
                lth_rep = self.lth_features(features.view(batch_size, -1))
                output = self.sigmoid_output(lth_rep)
                return output

            def similarity(self, x):
                batch_size = x.size()[0]
                features = self.main(x)
                lth_rep = self.lth_features(features.view(batch_size, -1))
                return lth_rep
```

0.3.5 Initialize the Parameters

In [0]:
```python
# define constant
input_channels = 1
hidden_size = 10
max_epochs = 250
lr = 3e-4

beta = 5
alpha = 0.1
gamma = 15
```

In [0]:
```python
G = VAE_GAN_Generator(input_channels, hidden_size).cuda()
D = Discriminator(input_channels).cuda()

criterion = nn.BCELoss()
criterion.cuda()

opt_enc = optim.RMSprop(G.encoder.parameters(), lr=lr)
opt_dec = optim.RMSprop(G.decoder.parameters(), lr=lr)
opt_dis = optim.RMSprop(D.parameters(), lr=lr * alpha)
```

0.3.6 Utils - Metric

In [0]:
```python
fixed_noise = Variable(torch.randn(batch_size, hidden_size)).cuda()
data, _ = next(iter(train_loader))
fixed_batch = Variable(data).cuda()
```

In [0]:
```python
## 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 )
```

0.3.7 Training

In [0]:

```
for epoch in range(max_epochs):
    D_real_list, D_rec_enc_list, D_rec_noise_list, D_list =RunningAverage(),
    RunningAverage(), RunningAverage(),RunningAverage()
    g_loss_list, rec_loss_list, prior_loss_list = RunningAverage(),
    RunningAverage(),RunningAverage()
    for data, _ in train_loader:
        batch_size = data.size()[0]
        ones_label = Variable(torch.ones(batch_size)).cuda()
        zeros_label = Variable(torch.zeros(batch_size)).cuda()
        ## Run encoder - network + decoder
        datav = Variable(data).cuda()
        mean, logvar, rec_enc = G(datav)


        ## noise vector - same size as encoder
        noisev = Variable(torch.randn(batch_size, hidden_size)).cuda()
        rec_noise = G.decoder(noisev) # decode noise

        # train discriminator
        output = D(datav)
        errD_real = criterion(output.squeeze(1), ones_label) ## real inputs
        D_real_list.update(output.data.mean())

        ## Fake inputs, 1. reconstructed output - x
        output = D(rec_enc)
        errD_rec_enc = criterion(output.squeeze(1), zeros_label)
        D_rec_enc_list.update(output.data.mean())


        ## Noise output
        output = D(rec_noise)
        errD_rec_noise = criterion(output.squeeze(1), zeros_label)
        D_rec_noise_list.update(output.data.mean())

        ## Total discriminator los
        dis_img_loss = errD_real + errD_rec_enc + errD_rec_noise

        D_list.update(dis_img_loss.data.mean())

        ## Discriminator
        opt_dis.zero_grad()
        dis_img_loss.backward(retain_graph=True)
```

```
        opt_dis.step()

        # train decoder
        output = D(datav)

        errD_real = criterion(output.squeeze(1), ones_label)

        output = D(rec_enc)
        errD_rec_enc = criterion(output.squeeze(1), zeros_label)

        output = D(rec_noise)
        errD_rec_noise = criterion(output.squeeze(1), zeros_label)

        similarity_rec_enc = D.similarity(rec_enc)
        similarity_data = D.similarity(datav)

        dis_img_loss = errD_real + errD_rec_enc + errD_rec_noise
        gen_img_loss = - dis_img_loss

        g_loss_list.update(gen_img_loss.data.mean())
        rec_loss = ((similarity_rec_enc - similarity_data) ** 2).mean()
        rec_loss_list.update(rec_loss.data.mean())
        err_dec = gamma * rec_loss + gen_img_loss ## Decoder Loss

        opt_dec.zero_grad()
        err_dec.backward(retain_graph=True)
        opt_dec.step()

        # train encoder
        prior_loss = 1 + logvar - mean.pow(2) - logvar.exp()
        prior_loss = (-0.5 * torch.sum(prior_loss))/torch.numel(mean.data)

        prior_loss_list.update(prior_loss.data.mean())
        err_enc = prior_loss + beta * rec_loss ## Encoder Loss

        opt_enc.zero_grad()
        err_enc.backward()
        opt_enc.step()

    #_, _, rec_imgs = G(fixed_batch)
    #show_and_save('rec_epoch_%d.png' % epoch ,make_grid((rec_imgs.data*0.5+0.5).cpu()

    #samples = G.decoder(fixed_noise)
    #vutils.save_image(samples.data,
         #  'sample_epoch_%d.png' % epoch,
     #       normalize=True)
    #localtime = time.asctime( time.localtime(time.time()) )
    #print (localtime)
```

```
            print ('[%d/%d]: D_real:%.4f, D_enc:%.4f, D_noise:%.4f, Loss_D:%.4f,'
                   'Loss_G:%.4f,'+
                   'rec_loss:%.4f, prior_loss:%.4f'
                   % (epoch,
                      max_epochs,
                      (D_real_list()),
                      (D_rec_enc_list()),
                      (D_rec_noise_list()),
                      (D_list()),
                      (g_loss_list()),
                      (rec_loss_list()),
                      (prior_loss_list())))
```

```
[0/250]: D_real:0.4018, D_enc:0.3001, D_noise:0.3000, Loss_D:1.7033, Loss_G:-1.6198, rec_loss:(
[1/250]: D_real:0.3857, D_enc:0.3074, D_noise:0.3074, Loss_D:1.7642, Loss_G:-1.6994, rec_loss:(
[2/250]: D_real:0.3830, D_enc:0.3090, D_noise:0.3085, Loss_D:1.7704, Loss_G:-1.7108, rec_loss:(
[3/250]: D_real:0.3897, D_enc:0.3049, D_noise:0.3049, Loss_D:1.7475, Loss_G:-1.6932, rec_loss:(
[4/250]: D_real:0.3961, D_enc:0.3017, D_noise:0.3030, Loss_D:1.7314, Loss_G:-1.6795, rec_loss:(
[5/250]: D_real:0.4059, D_enc:0.2969, D_noise:0.2978, Loss_D:1.7046, Loss_G:-1.6526, rec_loss:(
[6/250]: D_real:0.4082, D_enc:0.2959, D_noise:0.2964, Loss_D:1.7013, Loss_G:-1.6498, rec_loss:(
[7/250]: D_real:0.4094, D_enc:0.2945, D_noise:0.2952, Loss_D:1.6987, Loss_G:-1.6467, rec_loss:(
[8/250]: D_real:0.4185, D_enc:0.2916, D_noise:0.2919, Loss_D:1.6754, Loss_G:-1.6254, rec_loss:(
[9/250]: D_real:0.4210, D_enc:0.2900, D_noise:0.2898, Loss_D:1.6743, Loss_G:-1.6237, rec_loss:(
[10/250]: D_real:0.4159, D_enc:0.2920, D_noise:0.2920, Loss_D:1.6857, Loss_G:-1.6377, rec_loss
[11/250]: D_real:0.4144, D_enc:0.2933, D_noise:0.2921, Loss_D:1.6867, Loss_G:-1.6367, rec_loss
[12/250]: D_real:0.4189, D_enc:0.2898, D_noise:0.2915, Loss_D:1.6721, Loss_G:-1.6219, rec_loss
[13/250]: D_real:0.4195, D_enc:0.2903, D_noise:0.2911, Loss_D:1.6731, Loss_G:-1.6193, rec_loss
[14/250]: D_real:0.4219, D_enc:0.2895, D_noise:0.2895, Loss_D:1.6674, Loss_G:-1.6124, rec_loss
[15/250]: D_real:0.4230, D_enc:0.2883, D_noise:0.2894, Loss_D:1.6606, Loss_G:-1.6068, rec_loss
[16/250]: D_real:0.4252, D_enc:0.2874, D_noise:0.2882, Loss_D:1.6558, Loss_G:-1.5967, rec_loss
[17/250]: D_real:0.4223, D_enc:0.2891, D_noise:0.2888, Loss_D:1.6678, Loss_G:-1.6124, rec_loss
[18/250]: D_real:0.4216, D_enc:0.2893, D_noise:0.2901, Loss_D:1.6717, Loss_G:-1.6132, rec_loss
[19/250]: D_real:0.4250, D_enc:0.2865, D_noise:0.2884, Loss_D:1.6572, Loss_G:-1.5970, rec_loss
[20/250]: D_real:0.4261, D_enc:0.2870, D_noise:0.2864, Loss_D:1.6568, Loss_G:-1.5943, rec_loss
[21/250]: D_real:0.4240, D_enc:0.2881, D_noise:0.2881, Loss_D:1.6651, Loss_G:-1.6028, rec_loss
[22/250]: D_real:0.4251, D_enc:0.2883, D_noise:0.2879, Loss_D:1.6562, Loss_G:-1.5913, rec_loss
[23/250]: D_real:0.4253, D_enc:0.2874, D_noise:0.2869, Loss_D:1.6595, Loss_G:-1.5947, rec_loss
[24/250]: D_real:0.4253, D_enc:0.2867, D_noise:0.2877, Loss_D:1.6599, Loss_G:-1.5934, rec_loss
[25/250]: D_real:0.4262, D_enc:0.2856, D_noise:0.2876, Loss_D:1.6548, Loss_G:-1.5865, rec_loss
[26/250]: D_real:0.4277, D_enc:0.2855, D_noise:0.2874, Loss_D:1.6514, Loss_G:-1.5838, rec_loss
[27/250]: D_real:0.4315, D_enc:0.2835, D_noise:0.2842, Loss_D:1.6358, Loss_G:-1.5655, rec_loss
[28/250]: D_real:0.4330, D_enc:0.2833, D_noise:0.2837, Loss_D:1.6377, Loss_G:-1.5697, rec_loss
[29/250]: D_real:0.4324, D_enc:0.2841, D_noise:0.2841, Loss_D:1.6362, Loss_G:-1.5621, rec_loss
[30/250]: D_real:0.4319, D_enc:0.2838, D_noise:0.2849, Loss_D:1.6395, Loss_G:-1.5648, rec_loss
[31/250]: D_real:0.4356, D_enc:0.2821, D_noise:0.2827, Loss_D:1.6281, Loss_G:-1.5511, rec_loss
```

```
[32/250]: D_real:0.4336, D_enc:0.2834, D_noise:0.2825, Loss_D:1.6355, Loss_G:-1.5573, rec_loss
[33/250]: D_real:0.4372, D_enc:0.2807, D_noise:0.2816, Loss_D:1.6226, Loss_G:-1.5435, rec_loss
[34/250]: D_real:0.4395, D_enc:0.2791, D_noise:0.2810, Loss_D:1.6179, Loss_G:-1.5413, rec_loss
[35/250]: D_real:0.4388, D_enc:0.2798, D_noise:0.2815, Loss_D:1.6234, Loss_G:-1.5441, rec_loss
[36/250]: D_real:0.4435, D_enc:0.2770, D_noise:0.2798, Loss_D:1.6036, Loss_G:-1.5226, rec_loss
[37/250]: D_real:0.4446, D_enc:0.2750, D_noise:0.2796, Loss_D:1.6025, Loss_G:-1.5219, rec_loss
[38/250]: D_real:0.4452, D_enc:0.2776, D_noise:0.2779, Loss_D:1.6022, Loss_G:-1.5203, rec_loss
[39/250]: D_real:0.4435, D_enc:0.2769, D_noise:0.2800, Loss_D:1.6069, Loss_G:-1.5241, rec_loss
[40/250]: D_real:0.4449, D_enc:0.2772, D_noise:0.2781, Loss_D:1.6032, Loss_G:-1.5204, rec_loss
[41/250]: D_real:0.4453, D_enc:0.2764, D_noise:0.2784, Loss_D:1.6001, Loss_G:-1.5179, rec_loss
[42/250]: D_real:0.4437, D_enc:0.2763, D_noise:0.2793, Loss_D:1.6075, Loss_G:-1.5202, rec_loss
[43/250]: D_real:0.4494, D_enc:0.2737, D_noise:0.2766, Loss_D:1.5856, Loss_G:-1.5000, rec_loss
[44/250]: D_real:0.4507, D_enc:0.2735, D_noise:0.2757, Loss_D:1.5859, Loss_G:-1.4987, rec_loss
[45/250]: D_real:0.4498, D_enc:0.2744, D_noise:0.2757, Loss_D:1.5882, Loss_G:-1.5015, rec_loss
[46/250]: D_real:0.4549, D_enc:0.2713, D_noise:0.2738, Loss_D:1.5732, Loss_G:-1.4855, rec_loss
[47/250]: D_real:0.4581, D_enc:0.2702, D_noise:0.2717, Loss_D:1.5612, Loss_G:-1.4724, rec_loss
[48/250]: D_real:0.4624, D_enc:0.2694, D_noise:0.2683, Loss_D:1.5502, Loss_G:-1.4608, rec_loss
[49/250]: D_real:0.4657, D_enc:0.2686, D_noise:0.2653, Loss_D:1.5427, Loss_G:-1.4514, rec_loss
[50/250]: D_real:0.4653, D_enc:0.2676, D_noise:0.2667, Loss_D:1.5426, Loss_G:-1.4526, rec_loss
[51/250]: D_real:0.4689, D_enc:0.2644, D_noise:0.2670, Loss_D:1.5331, Loss_G:-1.4403, rec_loss
[52/250]: D_real:0.4676, D_enc:0.2642, D_noise:0.2669, Loss_D:1.5386, Loss_G:-1.4474, rec_loss
[53/250]: D_real:0.4696, D_enc:0.2642, D_noise:0.2666, Loss_D:1.5355, Loss_G:-1.4419, rec_loss
[54/250]: D_real:0.4732, D_enc:0.2630, D_noise:0.2641, Loss_D:1.5235, Loss_G:-1.4299, rec_loss
[55/250]: D_real:0.4748, D_enc:0.2627, D_noise:0.2618, Loss_D:1.5147, Loss_G:-1.4244, rec_loss
[56/250]: D_real:0.4742, D_enc:0.2620, D_noise:0.2641, Loss_D:1.5211, Loss_G:-1.4297, rec_loss
[57/250]: D_real:0.4755, D_enc:0.2610, D_noise:0.2627, Loss_D:1.5161, Loss_G:-1.4178, rec_loss
[58/250]: D_real:0.4800, D_enc:0.2585, D_noise:0.2618, Loss_D:1.5040, Loss_G:-1.4094, rec_loss
[59/250]: D_real:0.4818, D_enc:0.2570, D_noise:0.2605, Loss_D:1.4949, Loss_G:-1.4008, rec_loss
[60/250]: D_real:0.4823, D_enc:0.2567, D_noise:0.2614, Loss_D:1.4950, Loss_G:-1.4003, rec_loss
[61/250]: D_real:0.4829, D_enc:0.2566, D_noise:0.2593, Loss_D:1.4965, Loss_G:-1.3965, rec_loss
[62/250]: D_real:0.4846, D_enc:0.2565, D_noise:0.2585, Loss_D:1.4880, Loss_G:-1.3896, rec_loss
[63/250]: D_real:0.4897, D_enc:0.2549, D_noise:0.2560, Loss_D:1.4762, Loss_G:-1.3771, rec_loss
[64/250]: D_real:0.4895, D_enc:0.2549, D_noise:0.2547, Loss_D:1.4820, Loss_G:-1.3818, rec_loss
[65/250]: D_real:0.4936, D_enc:0.2521, D_noise:0.2542, Loss_D:1.4621, Loss_G:-1.3620, rec_loss
[66/250]: D_real:0.4968, D_enc:0.2491, D_noise:0.2531, Loss_D:1.4532, Loss_G:-1.3543, rec_loss
[67/250]: D_real:0.4986, D_enc:0.2509, D_noise:0.2511, Loss_D:1.4485, Loss_G:-1.3480, rec_loss
[68/250]: D_real:0.5010, D_enc:0.2473, D_noise:0.2517, Loss_D:1.4415, Loss_G:-1.3487, rec_loss
[69/250]: D_real:0.5038, D_enc:0.2469, D_noise:0.2493, Loss_D:1.4349, Loss_G:-1.3332, rec_loss
[70/250]: D_real:0.5081, D_enc:0.2431, D_noise:0.2479, Loss_D:1.4182, Loss_G:-1.3197, rec_loss
[71/250]: D_real:0.5125, D_enc:0.2412, D_noise:0.2464, Loss_D:1.4123, Loss_G:-1.3156, rec_loss
[72/250]: D_real:0.5122, D_enc:0.2404, D_noise:0.2470, Loss_D:1.4098, Loss_G:-1.3121, rec_loss
[73/250]: D_real:0.5107, D_enc:0.2423, D_noise:0.2479, Loss_D:1.4251, Loss_G:-1.3184, rec_loss
[74/250]: D_real:0.5114, D_enc:0.2429, D_noise:0.2442, Loss_D:1.4129, Loss_G:-1.3111, rec_loss
[75/250]: D_real:0.5148, D_enc:0.2407, D_noise:0.2455, Loss_D:1.4043, Loss_G:-1.2982, rec_loss
[76/250]: D_real:0.5197, D_enc:0.2381, D_noise:0.2415, Loss_D:1.3869, Loss_G:-1.2862, rec_loss
[77/250]: D_real:0.5251, D_enc:0.2365, D_noise:0.2389, Loss_D:1.3745, Loss_G:-1.2734, rec_loss
[78/250]: D_real:0.5210, D_enc:0.2363, D_noise:0.2421, Loss_D:1.3880, Loss_G:-1.2802, rec_loss
[79/250]: D_real:0.5263, D_enc:0.2340, D_noise:0.2387, Loss_D:1.3701, Loss_G:-1.2684, rec_loss
```

```
[80/250]: D_real:0.5298, D_enc:0.2330, D_noise:0.2365, Loss_D:1.3634, Loss_G:-1.2564, rec_loss
[81/250]: D_real:0.5347, D_enc:0.2316, D_noise:0.2346, Loss_D:1.3445, Loss_G:-1.2448, rec_loss
[82/250]: D_real:0.5370, D_enc:0.2292, D_noise:0.2329, Loss_D:1.3419, Loss_G:-1.2304, rec_loss
[83/250]: D_real:0.5417, D_enc:0.2265, D_noise:0.2314, Loss_D:1.3287, Loss_G:-1.2259, rec_loss
[84/250]: D_real:0.5395, D_enc:0.2267, D_noise:0.2342, Loss_D:1.3354, Loss_G:-1.2244, rec_loss
[85/250]: D_real:0.5456, D_enc:0.2226, D_noise:0.2309, Loss_D:1.3147, Loss_G:-1.2052, rec_loss
[86/250]: D_real:0.5466, D_enc:0.2231, D_noise:0.2298, Loss_D:1.3123, Loss_G:-1.2005, rec_loss
[87/250]: D_real:0.5522, D_enc:0.2206, D_noise:0.2274, Loss_D:1.2967, Loss_G:-1.1939, rec_loss
[88/250]: D_real:0.5531, D_enc:0.2198, D_noise:0.2265, Loss_D:1.2962, Loss_G:-1.1841, rec_loss
[89/250]: D_real:0.5541, D_enc:0.2196, D_noise:0.2255, Loss_D:1.2948, Loss_G:-1.1816, rec_loss
[90/250]: D_real:0.5583, D_enc:0.2197, D_noise:0.2232, Loss_D:1.2786, Loss_G:-1.1789, rec_loss
[91/250]: D_real:0.5626, D_enc:0.2151, D_noise:0.2221, Loss_D:1.2650, Loss_G:-1.1641, rec_loss
[92/250]: D_real:0.5657, D_enc:0.2137, D_noise:0.2195, Loss_D:1.2620, Loss_G:-1.1504, rec_loss
[93/250]: D_real:0.5709, D_enc:0.2118, D_noise:0.2165, Loss_D:1.2436, Loss_G:-1.1329, rec_loss
[94/250]: D_real:0.5759, D_enc:0.2106, D_noise:0.2146, Loss_D:1.2309, Loss_G:-1.1181, rec_loss
[95/250]: D_real:0.5791, D_enc:0.2063, D_noise:0.2138, Loss_D:1.2255, Loss_G:-1.1241, rec_loss
[96/250]: D_real:0.5825, D_enc:0.2065, D_noise:0.2107, Loss_D:1.2137, Loss_G:-1.1100, rec_loss
[97/250]: D_real:0.5901, D_enc:0.2028, D_noise:0.2063, Loss_D:1.1864, Loss_G:-1.0931, rec_loss
[98/250]: D_real:0.5896, D_enc:0.2016, D_noise:0.2091, Loss_D:1.1960, Loss_G:-1.0958, rec_loss
[99/250]: D_real:0.5929, D_enc:0.2015, D_noise:0.2042, Loss_D:1.1856, Loss_G:-1.0835, rec_loss
[100/250]: D_real:0.6010, D_enc:0.1968, D_noise:0.2028, Loss_D:1.1590, Loss_G:-1.0656, rec_los
[101/250]: D_real:0.6122, D_enc:0.1901, D_noise:0.1956, Loss_D:1.1202, Loss_G:-1.0317, rec_los
[102/250]: D_real:0.6133, D_enc:0.1892, D_noise:0.1971, Loss_D:1.1307, Loss_G:-1.0368, rec_los
[103/250]: D_real:0.6145, D_enc:0.1894, D_noise:0.1958, Loss_D:1.1230, Loss_G:-1.0302, rec_los
[104/250]: D_real:0.6223, D_enc:0.1861, D_noise:0.1915, Loss_D:1.0965, Loss_G:-1.0140, rec_los
[105/250]: D_real:0.6304, D_enc:0.1823, D_noise:0.1865, Loss_D:1.0732, Loss_G:-0.9874, rec_los
[106/250]: D_real:0.6401, D_enc:0.1783, D_noise:0.1802, Loss_D:1.0470, Loss_G:-0.9675, rec_los
[107/250]: D_real:0.6494, D_enc:0.1737, D_noise:0.1776, Loss_D:1.0175, Loss_G:-0.9378, rec_los
[108/250]: D_real:0.6615, D_enc:0.1669, D_noise:0.1703, Loss_D:0.9899, Loss_G:-0.9205, rec_los
[109/250]: D_real:0.6651, D_enc:0.1648, D_noise:0.1697, Loss_D:0.9724, Loss_G:-0.8965, rec_los
[110/250]: D_real:0.6729, D_enc:0.1635, D_noise:0.1637, Loss_D:0.9617, Loss_G:-0.9010, rec_los
[111/250]: D_real:0.6870, D_enc:0.1538, D_noise:0.1571, Loss_D:0.9197, Loss_G:-0.8547, rec_los
[112/250]: D_real:0.6823, D_enc:0.1580, D_noise:0.1589, Loss_D:0.9343, Loss_G:-0.8675, rec_los
[113/250]: D_real:0.6865, D_enc:0.1532, D_noise:0.1599, Loss_D:0.9269, Loss_G:-0.8457, rec_los
[114/250]: D_real:0.6917, D_enc:0.1521, D_noise:0.1546, Loss_D:0.9145, Loss_G:-0.8415, rec_los
[115/250]: D_real:0.7144, D_enc:0.1405, D_noise:0.1451, Loss_D:0.8465, Loss_G:-0.7975, rec_los
[116/250]: D_real:0.7384, D_enc:0.1272, D_noise:0.1337, Loss_D:0.7767, Loss_G:-0.7383, rec_los
[117/250]: D_real:0.7411, D_enc:0.1258, D_noise:0.1310, Loss_D:0.7812, Loss_G:-0.7549, rec_los
[118/250]: D_real:0.7586, D_enc:0.1164, D_noise:0.1242, Loss_D:0.7344, Loss_G:-0.7096, rec_los
[119/250]: D_real:0.7564, D_enc:0.1169, D_noise:0.1264, Loss_D:0.7302, Loss_G:-0.6970, rec_los
[120/250]: D_real:0.7564, D_enc:0.1199, D_noise:0.1224, Loss_D:0.7355, Loss_G:-0.7011, rec_los
[121/250]: D_real:0.7868, D_enc:0.1039, D_noise:0.1083, Loss_D:0.6464, Loss_G:-0.6240, rec_los
[122/250]: D_real:0.8205, D_enc:0.0898, D_noise:0.0877, Loss_D:0.5528, Loss_G:-0.5609, rec_los
[123/250]: D_real:0.7817, D_enc:0.1088, D_noise:0.1088, Loss_D:0.6759, Loss_G:-0.6533, rec_los
[124/250]: D_real:0.7823, D_enc:0.1050, D_noise:0.1121, Loss_D:0.6692, Loss_G:-0.6382, rec_los
[125/250]: D_real:0.7915, D_enc:0.1009, D_noise:0.1058, Loss_D:0.6431, Loss_G:-0.6018, rec_los
[126/250]: D_real:0.8330, D_enc:0.0818, D_noise:0.0830, Loss_D:0.5234, Loss_G:-0.5152, rec_los
[127/250]: D_real:0.7760, D_enc:0.1080, D_noise:0.1148, Loss_D:0.6853, Loss_G:-0.6388, rec_los
```

```
[128/250]: D_real:0.8789, D_enc:0.0599, D_noise:0.0602, Loss_D:0.3693, Loss_G:-0.3902, rec_loss
[129/250]: D_real:0.8359, D_enc:0.0786, D_noise:0.0837, Loss_D:0.5427, Loss_G:-0.5354, rec_loss
[130/250]: D_real:0.8447, D_enc:0.0751, D_noise:0.0789, Loss_D:0.4729, Loss_G:-0.4678, rec_loss
[131/250]: D_real:0.8161, D_enc:0.0896, D_noise:0.0937, Loss_D:0.5957, Loss_G:-0.5576, rec_loss
[132/250]: D_real:0.8903, D_enc:0.0536, D_noise:0.0549, Loss_D:0.3593, Loss_G:-0.3968, rec_loss
[133/250]: D_real:0.8838, D_enc:0.0547, D_noise:0.0585, Loss_D:0.3857, Loss_G:-0.4112, rec_loss
[134/250]: D_real:0.9012, D_enc:0.0471, D_noise:0.0502, Loss_D:0.3206, Loss_G:-0.3441, rec_loss
[135/250]: D_real:0.8924, D_enc:0.0512, D_noise:0.0552, Loss_D:0.3730, Loss_G:-0.3713, rec_loss
[136/250]: D_real:0.8572, D_enc:0.0681, D_noise:0.0735, Loss_D:0.4485, Loss_G:-0.4527, rec_loss
[137/250]: D_real:0.9075, D_enc:0.0453, D_noise:0.0467, Loss_D:0.3098, Loss_G:-0.3422, rec_loss
[138/250]: D_real:0.8628, D_enc:0.0622, D_noise:0.0720, Loss_D:0.4419, Loss_G:-0.4342, rec_loss
[139/250]: D_real:0.9445, D_enc:0.0260, D_noise:0.0290, Loss_D:0.1939, Loss_G:-0.2345, rec_loss
[140/250]: D_real:0.9384, D_enc:0.0304, D_noise:0.0302, Loss_D:0.2239, Loss_G:-0.2735, rec_loss
[141/250]: D_real:0.8732, D_enc:0.0600, D_noise:0.0658, Loss_D:0.4047, Loss_G:-0.3927, rec_loss
[142/250]: D_real:0.9297, D_enc:0.0317, D_noise:0.0360, Loss_D:0.2494, Loss_G:-0.2832, rec_loss
[143/250]: D_real:0.9144, D_enc:0.0396, D_noise:0.0451, Loss_D:0.2663, Loss_G:-0.2806, rec_loss
[144/250]: D_real:0.9207, D_enc:0.0378, D_noise:0.0403, Loss_D:0.2859, Loss_G:-0.3238, rec_loss
[145/250]: D_real:0.9130, D_enc:0.0416, D_noise:0.0435, Loss_D:0.2970, Loss_G:-0.3272, rec_loss
[146/250]: D_real:0.9276, D_enc:0.0344, D_noise:0.0363, Loss_D:0.2458, Loss_G:-0.2769, rec_loss
[147/250]: D_real:0.9361, D_enc:0.0301, D_noise:0.0329, Loss_D:0.2288, Loss_G:-0.2820, rec_loss
[148/250]: D_real:0.9316, D_enc:0.0320, D_noise:0.0352, Loss_D:0.2565, Loss_G:-0.2919, rec_loss
[149/250]: D_real:0.9230, D_enc:0.0357, D_noise:0.0403, Loss_D:0.2615, Loss_G:-0.2835, rec_loss
[150/250]: D_real:0.9130, D_enc:0.0410, D_noise:0.0444, Loss_D:0.2997, Loss_G:-0.3257, rec_loss
[151/250]: D_real:0.9211, D_enc:0.0362, D_noise:0.0426, Loss_D:0.2645, Loss_G:-0.2744, rec_loss
[152/250]: D_real:0.9367, D_enc:0.0290, D_noise:0.0318, Loss_D:0.2364, Loss_G:-0.2694, rec_loss
[153/250]: D_real:0.9393, D_enc:0.0272, D_noise:0.0329, Loss_D:0.2013, Loss_G:-0.2403, rec_loss
[154/250]: D_real:0.9401, D_enc:0.0280, D_noise:0.0298, Loss_D:0.2116, Loss_G:-0.2753, rec_loss
[155/250]: D_real:0.9247, D_enc:0.0342, D_noise:0.0396, Loss_D:0.2493, Loss_G:-0.2795, rec_loss
[156/250]: D_real:0.9553, D_enc:0.0208, D_noise:0.0233, Loss_D:0.1675, Loss_G:-0.2124, rec_loss
[157/250]: D_real:0.8845, D_enc:0.0539, D_noise:0.0603, Loss_D:0.3749, Loss_G:-0.3706, rec_loss
[158/250]: D_real:0.9138, D_enc:0.0395, D_noise:0.0452, Loss_D:0.2798, Loss_G:-0.2888, rec_loss
[159/250]: D_real:0.9197, D_enc:0.0357, D_noise:0.0443, Loss_D:0.2944, Loss_G:-0.2950, rec_loss
[160/250]: D_real:0.9353, D_enc:0.0297, D_noise:0.0338, Loss_D:0.2069, Loss_G:-0.2282, rec_loss
[161/250]: D_real:0.9078, D_enc:0.0417, D_noise:0.0489, Loss_D:0.3269, Loss_G:-0.3239, rec_loss
[162/250]: D_real:0.9068, D_enc:0.0405, D_noise:0.0518, Loss_D:0.3027, Loss_G:-0.2994, rec_loss
[163/250]: D_real:0.9308, D_enc:0.0321, D_noise:0.0364, Loss_D:0.2466, Loss_G:-0.2775, rec_loss
[164/250]: D_real:0.8904, D_enc:0.0500, D_noise:0.0586, Loss_D:0.3580, Loss_G:-0.3379, rec_loss
[165/250]: D_real:0.9430, D_enc:0.0255, D_noise:0.0297, Loss_D:0.2060, Loss_G:-0.2584, rec_loss
[166/250]: D_real:0.9682, D_enc:0.0151, D_noise:0.0152, Loss_D:0.1209, Loss_G:-0.2035, rec_loss
[167/250]: D_real:0.9478, D_enc:0.0247, D_noise:0.0264, Loss_D:0.2146, Loss_G:-0.2897, rec_loss
[168/250]: D_real:0.9093, D_enc:0.0422, D_noise:0.0463, Loss_D:0.3014, Loss_G:-0.2999, rec_loss
[169/250]: D_real:0.8993, D_enc:0.0469, D_noise:0.0530, Loss_D:0.3238, Loss_G:-0.2952, rec_loss
[170/250]: D_real:0.9428, D_enc:0.0263, D_noise:0.0299, Loss_D:0.2135, Loss_G:-0.2553, rec_loss
[171/250]: D_real:0.9363, D_enc:0.0284, D_noise:0.0342, Loss_D:0.2006, Loss_G:-0.2250, rec_loss
[172/250]: D_real:0.9530, D_enc:0.0207, D_noise:0.0249, Loss_D:0.1790, Loss_G:-0.2250, rec_loss
[173/250]: D_real:0.9594, D_enc:0.0198, D_noise:0.0206, Loss_D:0.1597, Loss_G:-0.2101, rec_loss
[174/250]: D_real:0.9364, D_enc:0.0277, D_noise:0.0330, Loss_D:0.2077, Loss_G:-0.2171, rec_loss
[175/250]: D_real:0.9496, D_enc:0.0234, D_noise:0.0262, Loss_D:0.1834, Loss_G:-0.2109, rec_loss
```

```
[176/250]: D_real:0.9431, D_enc:0.0253, D_noise:0.0300, Loss_D:0.2085, Loss_G:-0.2816, rec_loss
[177/250]: D_real:0.9359, D_enc:0.0296, D_noise:0.0340, Loss_D:0.2079, Loss_G:-0.2299, rec_loss
[178/250]: D_real:0.9256, D_enc:0.0344, D_noise:0.0381, Loss_D:0.2687, Loss_G:-0.3121, rec_loss
[179/250]: D_real:0.9562, D_enc:0.0201, D_noise:0.0227, Loss_D:0.1624, Loss_G:-0.2061, rec_loss
[180/250]: D_real:0.9177, D_enc:0.0376, D_noise:0.0437, Loss_D:0.2727, Loss_G:-0.2684, rec_loss
[181/250]: D_real:0.9348, D_enc:0.0307, D_noise:0.0354, Loss_D:0.2407, Loss_G:-0.2799, rec_loss
[182/250]: D_real:0.9382, D_enc:0.0274, D_noise:0.0324, Loss_D:0.2120, Loss_G:-0.2249, rec_loss
[183/250]: D_real:0.9584, D_enc:0.0179, D_noise:0.0219, Loss_D:0.1206, Loss_G:-0.1522, rec_loss
[184/250]: D_real:0.9500, D_enc:0.0238, D_noise:0.0261, Loss_D:0.2086, Loss_G:-0.2408, rec_loss
[185/250]: D_real:0.9205, D_enc:0.0349, D_noise:0.0432, Loss_D:0.2698, Loss_G:-0.2726, rec_loss
[186/250]: D_real:0.9341, D_enc:0.0299, D_noise:0.0354, Loss_D:0.2268, Loss_G:-0.2352, rec_loss
[187/250]: D_real:0.9528, D_enc:0.0212, D_noise:0.0243, Loss_D:0.1705, Loss_G:-0.1934, rec_loss
[188/250]: D_real:0.9588, D_enc:0.0188, D_noise:0.0209, Loss_D:0.1602, Loss_G:-0.1916, rec_loss
[189/250]: D_real:0.9564, D_enc:0.0193, D_noise:0.0237, Loss_D:0.1685, Loss_G:-0.2229, rec_loss
[190/250]: D_real:0.9290, D_enc:0.0312, D_noise:0.0384, Loss_D:0.2454, Loss_G:-0.2675, rec_loss
[191/250]: D_real:0.9426, D_enc:0.0271, D_noise:0.0296, Loss_D:0.2018, Loss_G:-0.2174, rec_loss
[192/250]: D_real:0.9150, D_enc:0.0393, D_noise:0.0446, Loss_D:0.2821, Loss_G:-0.2944, rec_loss
[193/250]: D_real:0.9326, D_enc:0.0309, D_noise:0.0350, Loss_D:0.2112, Loss_G:-0.2120, rec_loss
[194/250]: D_real:0.9151, D_enc:0.0378, D_noise:0.0456, Loss_D:0.2966, Loss_G:-0.2938, rec_loss
[195/250]: D_real:0.9307, D_enc:0.0326, D_noise:0.0367, Loss_D:0.2372, Loss_G:-0.2452, rec_loss
[196/250]: D_real:0.9659, D_enc:0.0143, D_noise:0.0187, Loss_D:0.1165, Loss_G:-0.1721, rec_loss
[197/250]: D_real:0.9521, D_enc:0.0221, D_noise:0.0246, Loss_D:0.1718, Loss_G:-0.2240, rec_loss
[198/250]: D_real:0.9849, D_enc:0.0075, D_noise:0.0078, Loss_D:0.0658, Loss_G:-0.1538, rec_loss
[199/250]: D_real:0.9372, D_enc:0.0277, D_noise:0.0341, Loss_D:0.2178, Loss_G:-0.2391, rec_loss
[200/250]: D_real:0.9435, D_enc:0.0246, D_noise:0.0309, Loss_D:0.1886, Loss_G:-0.2193, rec_loss
[201/250]: D_real:0.9731, D_enc:0.0114, D_noise:0.0140, Loss_D:0.0929, Loss_G:-0.1528, rec_loss
[202/250]: D_real:0.9771, D_enc:0.0110, D_noise:0.0108, Loss_D:0.1205, Loss_G:-0.2293, rec_loss
[203/250]: D_real:0.9546, D_enc:0.0210, D_noise:0.0234, Loss_D:0.1444, Loss_G:-0.1669, rec_loss
[204/250]: D_real:0.9386, D_enc:0.0274, D_noise:0.0329, Loss_D:0.1988, Loss_G:-0.2500, rec_loss
[205/250]: D_real:0.9453, D_enc:0.0249, D_noise:0.0286, Loss_D:0.1955, Loss_G:-0.2014, rec_loss
[206/250]: D_real:0.9555, D_enc:0.0188, D_noise:0.0253, Loss_D:0.1670, Loss_G:-0.2070, rec_loss
[207/250]: D_real:0.9649, D_enc:0.0150, D_noise:0.0188, Loss_D:0.1177, Loss_G:-0.1417, rec_loss
[208/250]: D_real:0.9634, D_enc:0.0156, D_noise:0.0194, Loss_D:0.1305, Loss_G:-0.1848, rec_loss
[209/250]: D_real:0.9623, D_enc:0.0162, D_noise:0.0200, Loss_D:0.1417, Loss_G:-0.1952, rec_loss
[210/250]: D_real:0.9489, D_enc:0.0218, D_noise:0.0279, Loss_D:0.1740, Loss_G:-0.2074, rec_loss
[211/250]: D_real:0.9563, D_enc:0.0188, D_noise:0.0236, Loss_D:0.1559, Loss_G:-0.1776, rec_loss
[212/250]: D_real:0.9308, D_enc:0.0295, D_noise:0.0383, Loss_D:0.2430, Loss_G:-0.2340, rec_loss
[213/250]: D_real:0.9598, D_enc:0.0188, D_noise:0.0208, Loss_D:0.1504, Loss_G:-0.1697, rec_loss
[214/250]: D_real:0.9203, D_enc:0.0346, D_noise:0.0446, Loss_D:0.2772, Loss_G:-0.2722, rec_loss
[215/250]: D_real:0.9450, D_enc:0.0231, D_noise:0.0297, Loss_D:0.1796, Loss_G:-0.1821, rec_loss
[216/250]: D_real:0.9512, D_enc:0.0201, D_noise:0.0277, Loss_D:0.1900, Loss_G:-0.2297, rec_loss
[217/250]: D_real:0.9442, D_enc:0.0250, D_noise:0.0293, Loss_D:0.1938, Loss_G:-0.2192, rec_loss
[218/250]: D_real:0.9545, D_enc:0.0211, D_noise:0.0241, Loss_D:0.1658, Loss_G:-0.1667, rec_loss
[219/250]: D_real:0.9668, D_enc:0.0149, D_noise:0.0167, Loss_D:0.1165, Loss_G:-0.1738, rec_loss
[220/250]: D_real:0.9583, D_enc:0.0176, D_noise:0.0236, Loss_D:0.1442, Loss_G:-0.1663, rec_loss
[221/250]: D_real:0.9813, D_enc:0.0083, D_noise:0.0093, Loss_D:0.0790, Loss_G:-0.2047, rec_loss
[222/250]: D_real:0.9457, D_enc:0.0231, D_noise:0.0297, Loss_D:0.1881, Loss_G:-0.2084, rec_loss
[223/250]: D_real:0.9381, D_enc:0.0275, D_noise:0.0335, Loss_D:0.2244, Loss_G:-0.2232, rec_loss
```

```
[224/250]: D_real:0.9627, D_enc:0.0172, D_noise:0.0191, Loss_D:0.1373, Loss_G:-0.1811, rec_loss
[225/250]: D_real:0.9291, D_enc:0.0306, D_noise:0.0395, Loss_D:0.2425, Loss_G:-0.2576, rec_loss
[226/250]: D_real:0.9332, D_enc:0.0296, D_noise:0.0361, Loss_D:0.2227, Loss_G:-0.2196, rec_loss
[227/250]: D_real:0.9363, D_enc:0.0285, D_noise:0.0338, Loss_D:0.2132, Loss_G:-0.2355, rec_loss
[228/250]: D_real:0.9578, D_enc:0.0176, D_noise:0.0237, Loss_D:0.1445, Loss_G:-0.1855, rec_loss
[229/250]: D_real:0.9709, D_enc:0.0136, D_noise:0.0148, Loss_D:0.1045, Loss_G:-0.1432, rec_loss
[230/250]: D_real:0.9753, D_enc:0.0103, D_noise:0.0133, Loss_D:0.0944, Loss_G:-0.1456, rec_loss
[231/250]: D_real:0.9458, D_enc:0.0243, D_noise:0.0300, Loss_D:0.2024, Loss_G:-0.1997, rec_loss
[232/250]: D_real:0.9292, D_enc:0.0308, D_noise:0.0392, Loss_D:0.2142, Loss_G:-0.1965, rec_loss
[233/250]: D_real:0.9523, D_enc:0.0209, D_noise:0.0265, Loss_D:0.1941, Loss_G:-0.2233, rec_loss
[234/250]: D_real:0.9662, D_enc:0.0151, D_noise:0.0177, Loss_D:0.1213, Loss_G:-0.1384, rec_loss
[235/250]: D_real:0.9793, D_enc:0.0090, D_noise:0.0103, Loss_D:0.0772, Loss_G:-0.1348, rec_loss
[236/250]: D_real:0.9755, D_enc:0.0109, D_noise:0.0122, Loss_D:0.0903, Loss_G:-0.1524, rec_loss
[237/250]: D_real:0.9503, D_enc:0.0207, D_noise:0.0283, Loss_D:0.1809, Loss_G:-0.1932, rec_loss
[238/250]: D_real:0.9522, D_enc:0.0207, D_noise:0.0263, Loss_D:0.1855, Loss_G:-0.2112, rec_loss
[239/250]: D_real:0.9460, D_enc:0.0239, D_noise:0.0290, Loss_D:0.1951, Loss_G:-0.1843, rec_loss
[240/250]: D_real:0.9679, D_enc:0.0151, D_noise:0.0165, Loss_D:0.1114, Loss_G:-0.1864, rec_loss
[241/250]: D_real:0.9733, D_enc:0.0122, D_noise:0.0131, Loss_D:0.0816, Loss_G:-0.1601, rec_loss
[242/250]: D_real:0.9637, D_enc:0.0165, D_noise:0.0193, Loss_D:0.1335, Loss_G:-0.1658, rec_loss
[243/250]: D_real:0.9671, D_enc:0.0145, D_noise:0.0180, Loss_D:0.1319, Loss_G:-0.1620, rec_loss
[244/250]: D_real:0.9871, D_enc:0.0061, D_noise:0.0061, Loss_D:0.0494, Loss_G:-0.1589, rec_loss
[245/250]: D_real:0.9678, D_enc:0.0139, D_noise:0.0175, Loss_D:0.1330, Loss_G:-0.1779, rec_loss
[246/250]: D_real:0.9511, D_enc:0.0205, D_noise:0.0274, Loss_D:0.1850, Loss_G:-0.2086, rec_loss
[247/250]: D_real:0.9531, D_enc:0.0203, D_noise:0.0253, Loss_D:0.1546, Loss_G:-0.1744, rec_loss
[248/250]: D_real:0.9744, D_enc:0.0108, D_noise:0.0142, Loss_D:0.1118, Loss_G:-0.1515, rec_loss
[249/250]: D_real:0.9436, D_enc:0.0248, D_noise:0.0309, Loss_D:0.1800, Loss_G:-0.2238, rec_loss
```

In [0]:
```
save_model(epoch, G.encoder, G.decoder, D) # Save encoder, decoder and D
```

0.4 Model 1 - Test Classifier

Test classifier with VAE - gradients set to false

In [0]:

In [0]:
```
## Classifier
### 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'
```

In [0]:
```
save_dir = ""

### Plot TSNE for latent space
# Show dataset images with T-sne projection of latent space encoding
```

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

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.6/dist-packages (3.4.5.
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packages (from o
```

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]
```

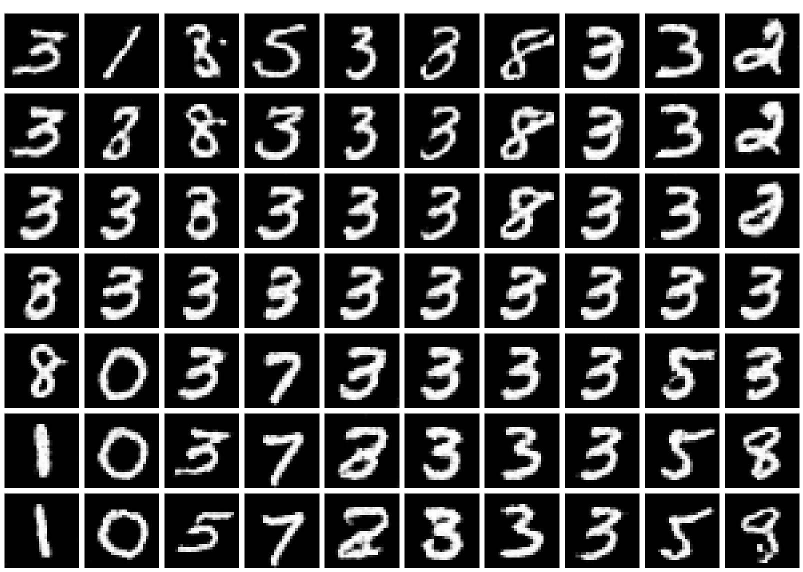
```
def laplacian_variance_numpy(images):
    return [cv2.Laplacian(image, cv2.CV_32F).var() for image in images]
```

In [0]:
```
testpoint = torch.Tensor(train_loader.dataset[0][0]).cuda()
```

In [0]:
```
def traverse_latents(model, datapoint, nb_latents, epoch, batch_idx, dirpath=save_dir)
  model.eval()
  datapoint = datapoint.cuda()

  datapoint = datapoint.unsqueeze(0)
  mu, _ = model.encoder(datapoint)

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

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

traverse_latents(G, testpoint,Params.nb_latents, epoch,1,save_dir)
```

In [0]:
```
from IPython.display import Image
Image(filename=save_dir+f'traversal_210_1.png')
```

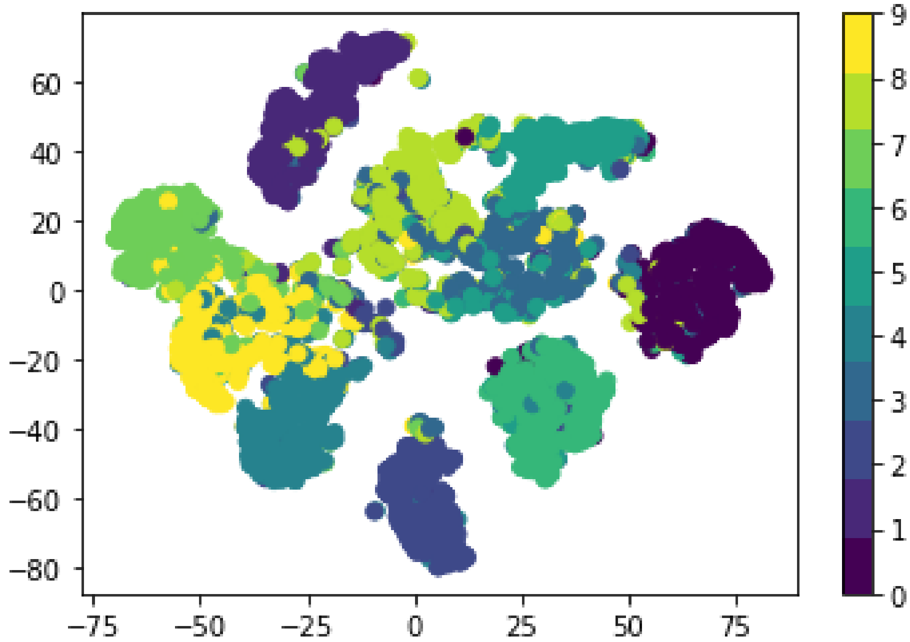
Out[0]:



In [0]:
```
train_dataset = datasets.MNIST(Params.data_dir,train=True,
                               transform=transforms.ToTensor(),download=True)
testing_tsne = torch.utils.data.DataLoader(train_dataset,
                    batch_size=len(train_dataset),shuffle=True)
test_data, test_labels = next(iter(testing_tsne))[:10000]
```

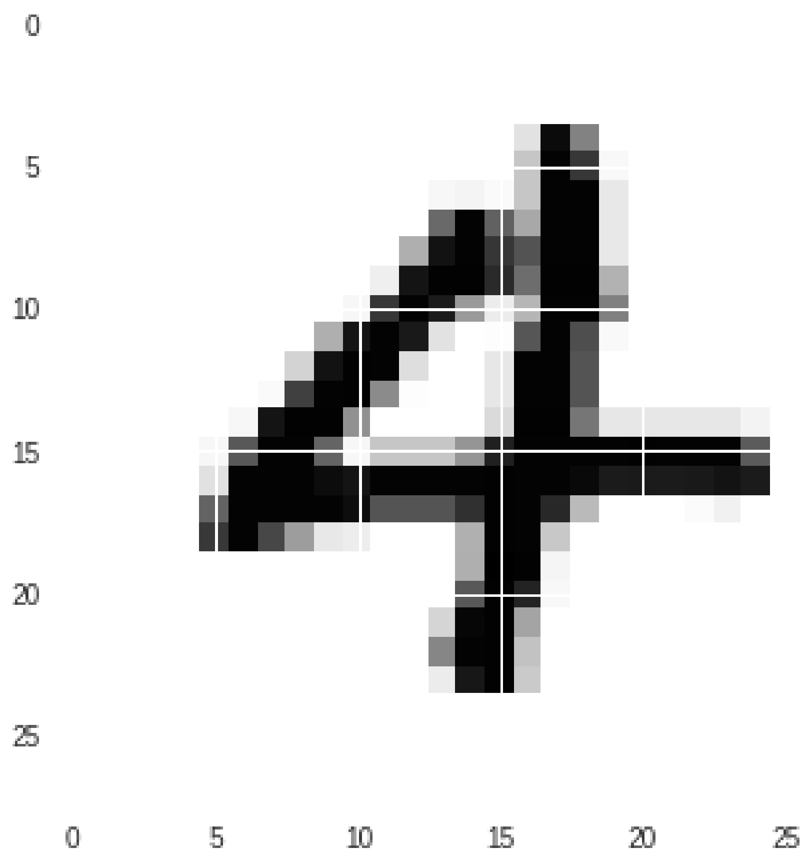
In [0]:
```
from scipy.stats import norm
from sklearn import manifold
path = save_dir+'latent_space.png'
def visualize_tsne(X, labels, model, path):
    # Compute latent space representation
    print("Computing latent space projection...")

    X_encoded, _ = model.encoder(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].cuda(),test_labels[:5000],G,path)
```

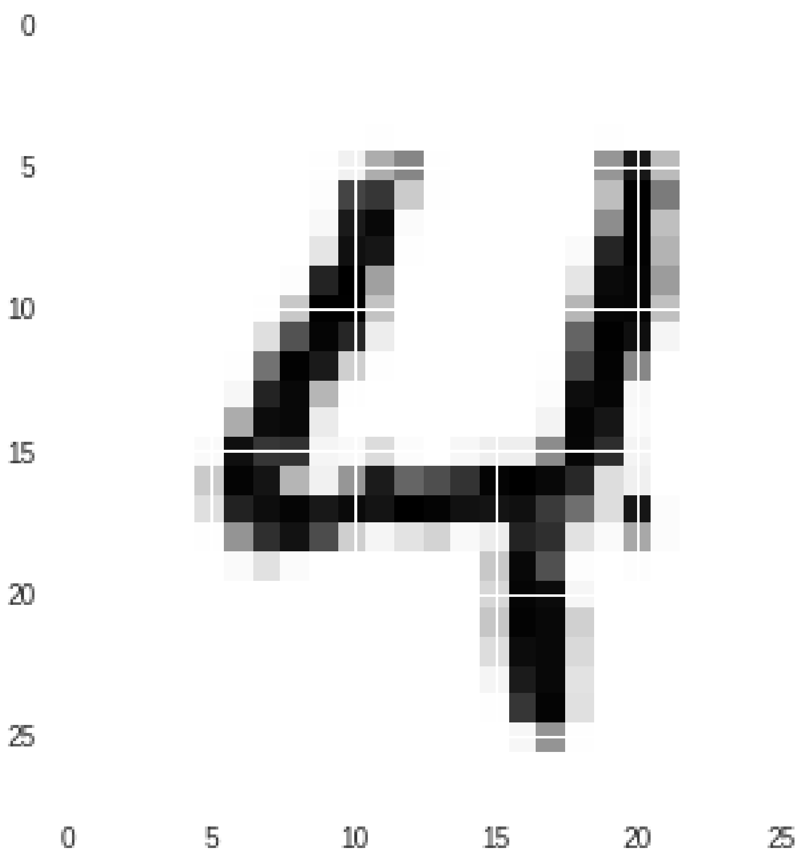
```
Computing latent space projection...
```



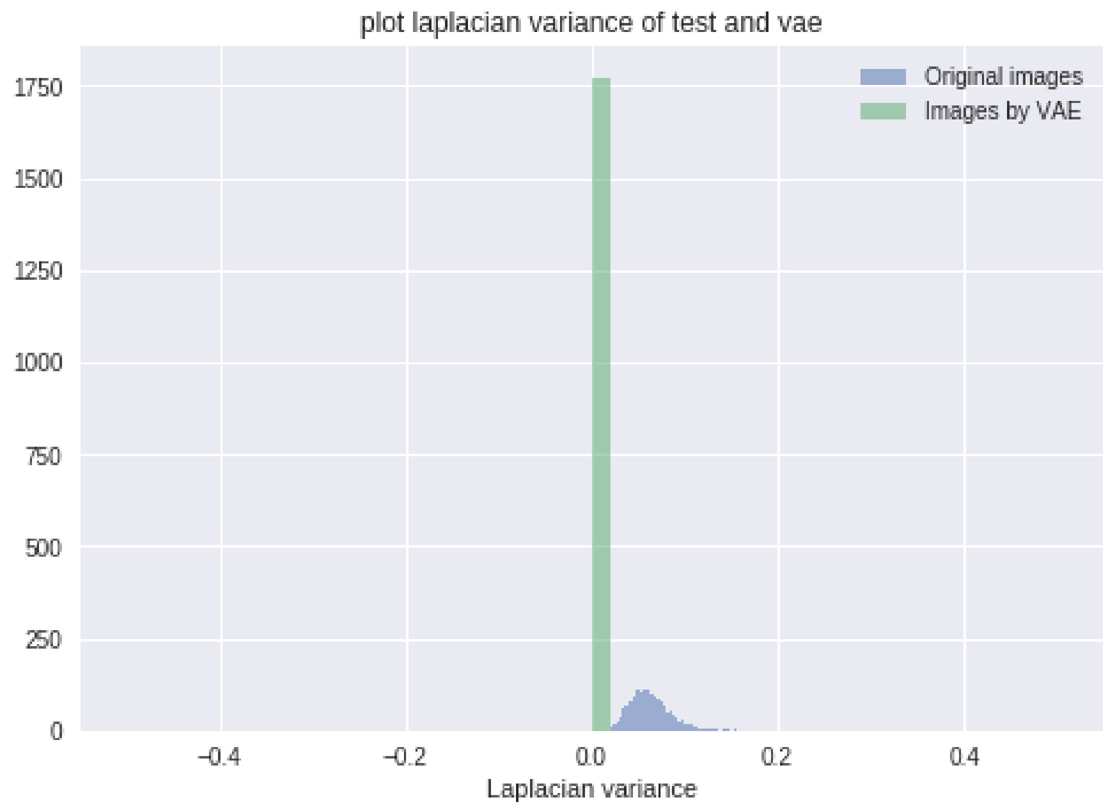
In [0]:
```
x_test, y_test = test_data[:2000], test_labels[:2000]
not_ones = y_test != 1
x_test_not_ones, y_test_not_ones = x_test[not_ones], y_test[not_ones]
with torch.no_grad():
    reconstructions = np.empty(shape=(len(x_test_not_ones),1,28,28))

    indx = 0
    for i, (x,y) in enumerate(zip(x_test_not_ones,y_test_not_ones)):
      mean, logvar, rec_enc = G(x.unsqueeze(0).cuda())
      reconstructions[indx]=(rec_enc.squeeze(0).detach().cpu())
      indx+=1
```

In [0]:
```
x_test_not_ones, y_test_not_ones = x_test[not_ones], y_test[not_ones]
len(y_test_not_ones)


plt.imshow(x_test_not_ones[5].squeeze(0))
plt.show()
plt.imshow(reconstructions[5].squeeze(0))
plt.show()
```


0
5
10
15
20
25
0
5
10
15
20
25



In [0]:
```
len(x_test[y_test==1])

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();

plot_laplacian_variances(lvs_1, lvs_2, "plot laplacian variance of test and vae")
```




```
In [0]: import torch.nn.functional as F
        import torch.optim as optim
        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()
classifier = classifier.cuda()

# Loss and optimizer
learning_rate = 0.001
momemtum = 0.9
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(classifier.parameters(), lr=learning_rate)
```

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

## Train Classifier with pretrained vae
vae_parameters = list(G.decoder.named_parameters())
for name, param in vae_parameters:
    param.requires_grad = False
```

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


    mu, logvar,recon_batch = G(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

  ## 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.cuda()
      labels = labels.cuda()


      mu, logvar,recon_batch = G(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: 0.0148         Average Accuracy:          Accuracy: 48.78
Epoch: 2/50. Train set: Average loss: 0.0148         Average Accuracy:          Accuracy: 48.78
Epoch: 2/50. Test set: Average loss: 0.0128         Average Accuracy:          Accuracy: 71.89
Epoch: 3/50. Train set: Average loss: 0.0100         Average Accuracy:          Accuracy: 76.78
Epoch: 3/50. Train set: Average loss: 0.0100         Average Accuracy:          Accuracy: 76.78
Epoch: 3/50. Test set: Average loss: 0.0090         Average Accuracy:          Accuracy: 81.3
Epoch: 4/50. Train set: Average loss: 0.0076         Average Accuracy:          Accuracy: 82.58
Epoch: 4/50. Train set: Average loss: 0.0076         Average Accuracy:          Accuracy: 82.58
Epoch: 4/50. Test set: Average loss: 0.0074         Average Accuracy:          Accuracy: 84.75
Epoch: 5/50. Train set: Average loss: 0.0062         Average Accuracy:          Accuracy: 84.51
Epoch: 5/50. Train set: Average loss: 0.0062         Average Accuracy:          Accuracy: 84.51
Epoch: 5/50. Test set: Average loss: 0.0061         Average Accuracy:          Accuracy: 85.82
Epoch: 6/50. Train set: Average loss: 0.0054         Average Accuracy:          Accuracy: 85.35
Epoch: 6/50. Train set: Average loss: 0.0054         Average Accuracy:          Accuracy: 85.35
Epoch: 6/50. Test set: Average loss: 0.0055         Average Accuracy:          Accuracy: 86.57
Epoch: 7/50. Train set: Average loss: 0.0049         Average Accuracy:          Accuracy: 85.89
Epoch: 7/50. Train set: Average loss: 0.0049         Average Accuracy:          Accuracy: 85.89
Epoch: 7/50. Test set: Average loss: 0.0048         Average Accuracy:          Accuracy: 87.09
Epoch: 8/50. Train set: Average loss: 0.0045         Average Accuracy:          Accuracy: 86.36
Epoch: 8/50. Train set: Average loss: 0.0045         Average Accuracy:          Accuracy: 86.36
Epoch: 8/50. Test set: Average loss: 0.0046         Average Accuracy:          Accuracy: 87.33
Epoch: 9/50. Train set: Average loss: 0.0042         Average Accuracy:          Accuracy: 86.70
Epoch: 9/50. Train set: Average loss: 0.0042         Average Accuracy:          Accuracy: 86.70
Epoch: 9/50. Test set: Average loss: 0.0042         Average Accuracy:          Accuracy: 87.69
Epoch: 10/50. Train set: Average loss: 0.0039         Average Accuracy:          Accuracy: 87.0
Epoch: 10/50. Train set: Average loss: 0.0039         Average Accuracy:          Accuracy: 87.0
Epoch: 10/50. Test set: Average loss: 0.0039         Average Accuracy:          Accuracy: 87.88
Epoch: 11/50. Train set: Average loss: 0.0037         Average Accuracy:          Accuracy: 87.3
Epoch: 11/50. Train set: Average loss: 0.0037         Average Accuracy:          Accuracy: 87.3
Epoch: 11/50. Test set: Average loss: 0.0041         Average Accuracy:          Accuracy: 88.17
Epoch: 12/50. Train set: Average loss: 0.0036         Average Accuracy:          Accuracy: 87.5
Epoch: 12/50. Train set: Average loss: 0.0036         Average Accuracy:          Accuracy: 87.5
Epoch: 12/50. Test set: Average loss: 0.0037         Average Accuracy:          Accuracy: 88.3
Epoch: 13/50. Train set: Average loss: 0.0035         Average Accuracy:          Accuracy: 87.7
```

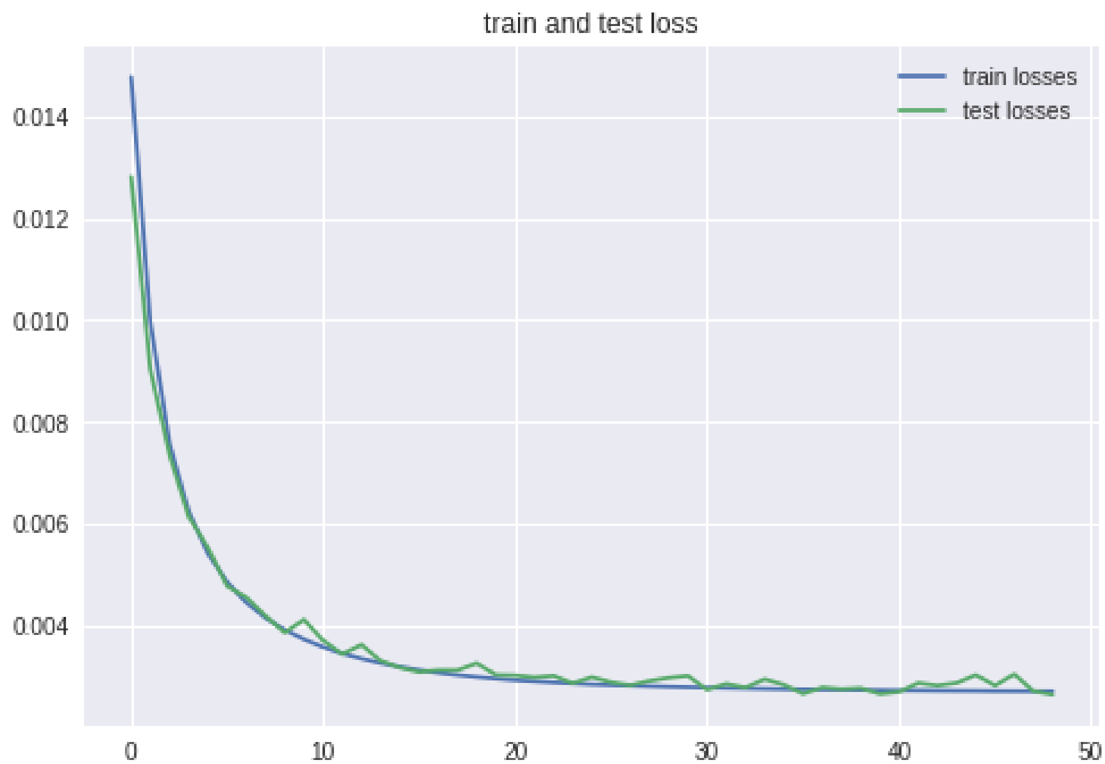
```
Epoch: 13/50. Train set: Average loss: 0.0035     Average Accuracy:     Accuracy: 87.75
Epoch: 13/50. Test set: Average loss: 0.0034     Average Accuracy:     Accuracy: 88.45
Epoch: 14/50. Train set: Average loss: 0.0034     Average Accuracy:     Accuracy: 87.93
Epoch: 14/50. Train set: Average loss: 0.0034     Average Accuracy:     Accuracy: 87.93
Epoch: 14/50. Test set: Average loss: 0.0036     Average Accuracy:     Accuracy: 88.64
Epoch: 15/50. Train set: Average loss: 0.0033     Average Accuracy:     Accuracy: 88.14
Epoch: 15/50. Train set: Average loss: 0.0033     Average Accuracy:     Accuracy: 88.14
Epoch: 15/50. Test set: Average loss: 0.0033     Average Accuracy:     Accuracy: 88.75
Epoch: 16/50. Train set: Average loss: 0.0032     Average Accuracy:     Accuracy: 88.24
Epoch: 16/50. Train set: Average loss: 0.0032     Average Accuracy:     Accuracy: 88.24
Epoch: 16/50. Test set: Average loss: 0.0032     Average Accuracy:     Accuracy: 89.05
Epoch: 17/50. Train set: Average loss: 0.0031     Average Accuracy:     Accuracy: 88.37
Epoch: 17/50. Train set: Average loss: 0.0031     Average Accuracy:     Accuracy: 88.37
Epoch: 17/50. Test set: Average loss: 0.0031     Average Accuracy:     Accuracy: 89.14
Epoch: 18/50. Train set: Average loss: 0.0031     Average Accuracy:     Accuracy: 88.47
Epoch: 18/50. Train set: Average loss: 0.0031     Average Accuracy:     Accuracy: 88.47
Epoch: 18/50. Test set: Average loss: 0.0031     Average Accuracy:     Accuracy: 89.27
Epoch: 19/50. Train set: Average loss: 0.0030     Average Accuracy:     Accuracy: 88.59
Epoch: 19/50. Train set: Average loss: 0.0030     Average Accuracy:     Accuracy: 88.59
Epoch: 19/50. Test set: Average loss: 0.0031     Average Accuracy:     Accuracy: 89.28
Epoch: 20/50. Train set: Average loss: 0.0030     Average Accuracy:     Accuracy: 88.62
Epoch: 20/50. Train set: Average loss: 0.0030     Average Accuracy:     Accuracy: 88.62
Epoch: 20/50. Test set: Average loss: 0.0033     Average Accuracy:     Accuracy: 89.41
Epoch: 21/50. Train set: Average loss: 0.0030     Average Accuracy:     Accuracy: 88.72
Epoch: 21/50. Train set: Average loss: 0.0030     Average Accuracy:     Accuracy: 88.72
Epoch: 21/50. Test set: Average loss: 0.0030     Average Accuracy:     Accuracy: 89.49
Epoch: 22/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.80
Epoch: 22/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.80
Epoch: 22/50. Test set: Average loss: 0.0030     Average Accuracy:     Accuracy: 89.61
Epoch: 23/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.86
Epoch: 23/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.86
Epoch: 23/50. Test set: Average loss: 0.0030     Average Accuracy:     Accuracy: 89.6
Epoch: 24/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.92
Epoch: 24/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.92
Epoch: 24/50. Test set: Average loss: 0.0030     Average Accuracy:     Accuracy: 89.67
Epoch: 25/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.97
Epoch: 25/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.97
Epoch: 25/50. Test set: Average loss: 0.0029     Average Accuracy:     Accuracy: 89.72
Epoch: 26/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.98
Epoch: 26/50. Train set: Average loss: 0.0029     Average Accuracy:     Accuracy: 88.98
Epoch: 26/50. Test set: Average loss: 0.0030     Average Accuracy:     Accuracy: 89.75
Epoch: 27/50. Train set: Average loss: 0.0028     Average Accuracy:     Accuracy: 89.03
Epoch: 27/50. Train set: Average loss: 0.0028     Average Accuracy:     Accuracy: 89.03
Epoch: 27/50. Test set: Average loss: 0.0029     Average Accuracy:     Accuracy: 89.8
Epoch: 28/50. Train set: Average loss: 0.0028     Average Accuracy:     Accuracy: 89.06
Epoch: 28/50. Train set: Average loss: 0.0028     Average Accuracy:     Accuracy: 89.06
Epoch: 28/50. Test set: Average loss: 0.0028     Average Accuracy:     Accuracy: 89.76
Epoch: 29/50. Train set: Average loss: 0.0028     Average Accuracy:     Accuracy: 89.10
```

```
Epoch: 29/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.10
Epoch: 29/50. Test set: Average loss: 0.0029    Average Accuracy:    Accuracy: 89.82
Epoch: 30/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.09
Epoch: 30/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.09
Epoch: 30/50. Test set: Average loss: 0.0030    Average Accuracy:    Accuracy: 89.79
Epoch: 31/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.12
Epoch: 31/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.12
Epoch: 31/50. Test set: Average loss: 0.0030    Average Accuracy:    Accuracy: 89.82
Epoch: 32/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.17
Epoch: 32/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.17
Epoch: 32/50. Test set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.84
Epoch: 33/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.17
Epoch: 33/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.17
Epoch: 33/50. Test set: Average loss: 0.0029    Average Accuracy:    Accuracy: 89.81
Epoch: 34/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.19
Epoch: 34/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.19
Epoch: 34/50. Test set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.83
Epoch: 35/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.23
Epoch: 35/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.23
Epoch: 35/50. Test set: Average loss: 0.0030    Average Accuracy:    Accuracy: 89.83
Epoch: 36/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.26
Epoch: 36/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.26
Epoch: 36/50. Test set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.83
Epoch: 37/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.26
Epoch: 37/50. Train set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.26
Epoch: 37/50. Test set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.88
Epoch: 38/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.28
Epoch: 38/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.28
Epoch: 38/50. Test set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.89
Epoch: 39/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.31
Epoch: 39/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.31
Epoch: 39/50. Test set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.96
Epoch: 40/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.31
Epoch: 40/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.31
Epoch: 40/50. Test set: Average loss: 0.0028    Average Accuracy:    Accuracy: 89.98
Epoch: 41/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.33
Epoch: 41/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.33
Epoch: 41/50. Test set: Average loss: 0.0027    Average Accuracy:    Accuracy: 90.0
Epoch: 42/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.34
Epoch: 42/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.34
Epoch: 42/50. Test set: Average loss: 0.0027    Average Accuracy:    Accuracy: 90.02
Epoch: 43/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.35
Epoch: 43/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.35
Epoch: 43/50. Test set: Average loss: 0.0029    Average Accuracy:    Accuracy: 90.01
Epoch: 44/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.36
Epoch: 44/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.36
Epoch: 44/50. Test set: Average loss: 0.0028    Average Accuracy:    Accuracy: 90.07
Epoch: 45/50. Train set: Average loss: 0.0027    Average Accuracy:    Accuracy: 89.39
```

```
Epoch: 45/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.3
Epoch: 45/50. Test set: Average loss: 0.0029         Average Accuracy:         Accuracy: 90.07
Epoch: 46/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.3
Epoch: 46/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.3
Epoch: 46/50. Test set: Average loss: 0.0030         Average Accuracy:         Accuracy: 90.05
Epoch: 47/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.4
Epoch: 47/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.4
Epoch: 47/50. Test set: Average loss: 0.0028         Average Accuracy:         Accuracy: 90.06
Epoch: 48/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.4
Epoch: 48/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.4
Epoch: 48/50. Test set: Average loss: 0.0031         Average Accuracy:         Accuracy: 90.08
Epoch: 49/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.3
Epoch: 49/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.3
Epoch: 49/50. Test set: Average loss: 0.0027         Average Accuracy:         Accuracy: 90.1
Epoch: 50/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.4
Epoch: 50/50. Train set: Average loss: 0.0027         Average Accuracy:         Accuracy: 89.4
Epoch: 50/50. Test set: Average loss: 0.0026         Average Accuracy:         Accuracy: 90.07
```

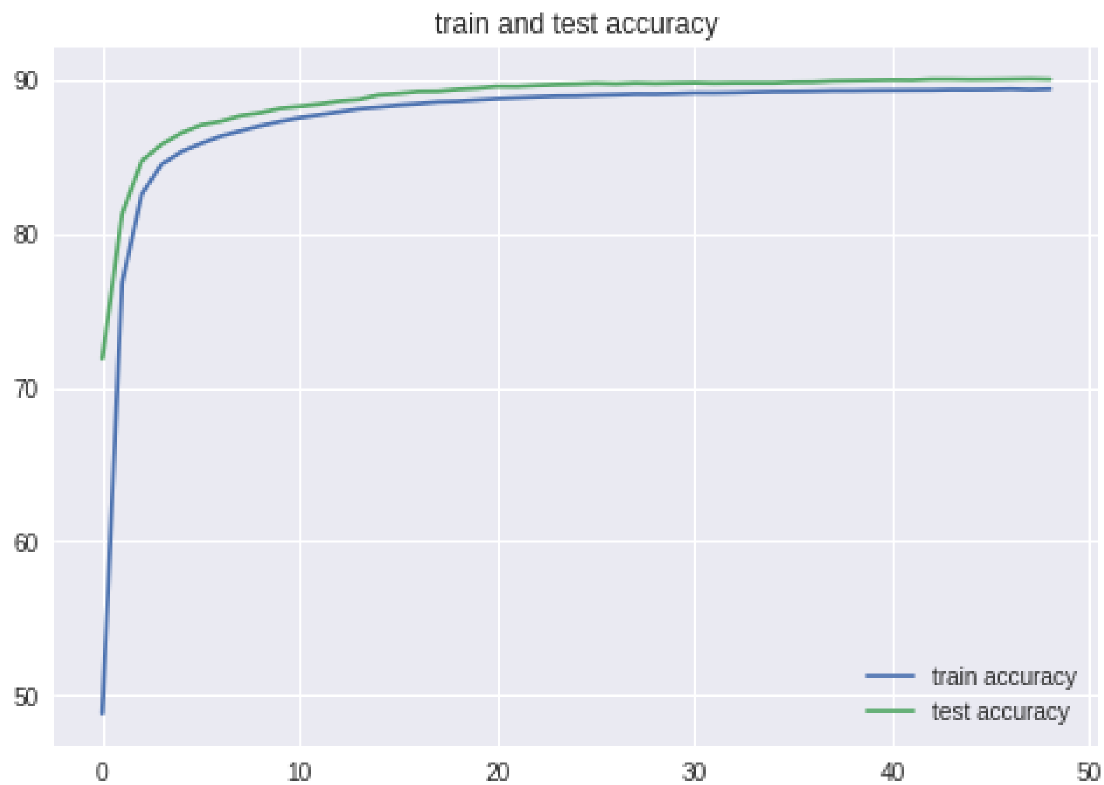
In [0]:
```python
plt.plot(range(n_epochs-1),train_losses)
plt.plot(range(n_epochs-1),test_losses)
plt.legend(["train losses","test losses"])
plt.title("train and test loss")
```

Out[0]:
```
Text(0.5, 1.0, 'train and test loss')
```



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

Out[0]:
```
Text(0.5, 1.0, 'train and test accuracy')
```



In [0]:

0.5 Model 2 - Test Classifier

GAN + Classifier trained

In [0]:
```
def loss_function(recon_x, x, mu, logvar):
    """
    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]: classifier = Classifier()


        classifier = classifier.cuda()

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

        """
        Train VAE + classifier
        """
        def train_classifier_all(epoch):

          G.train()
          classifier.train()
          metric = AccumulatedAccuracyMetric()
          losses = RunningAverage()
          for idx, (data, labels) in enumerate(train_loader):

            data = data.cuda()
            labels = labels.cuda()

            mu, logvar,recon_batch = G(data)

            outputs = classifier(mu)
            classifier_loss = criterion(outputs, labels)
            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()
    G.eval()
    metric = AccumulatedAccuracyMetric()
    #losses = RunningAverage()
    for idx, (data, labels) in enumerate(test_loader):

      data, labels = data.cuda(), labels.cuda()

      mu, logvar,recon_batch = G(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: 224.1224         Average Accuracy:          Accuracy: 43.5
Epoch: 2/50. Train set: Average loss: 224.1224         Average Accuracy:          Accuracy: 43.5
Epoch: 2/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 67.7
Epoch: 3/50. Train set: Average loss: 223.7884         Average Accuracy:          Accuracy: 74.6
Epoch: 3/50. Train set: Average loss: 223.7884         Average Accuracy:          Accuracy: 74.6
Epoch: 3/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 81.46
Epoch: 4/50. Train set: Average loss: 223.5140         Average Accuracy:          Accuracy: 82.9
Epoch: 4/50. Train set: Average loss: 223.5140         Average Accuracy:          Accuracy: 82.9
Epoch: 4/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 85.65
Epoch: 5/50. Train set: Average loss: 223.0415         Average Accuracy:          Accuracy: 85.8
Epoch: 5/50. Train set: Average loss: 223.0415         Average Accuracy:          Accuracy: 85.8
Epoch: 5/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 88.57
Epoch: 6/50. Train set: Average loss: 222.7645         Average Accuracy:          Accuracy: 87.4
Epoch: 6/50. Train set: Average loss: 222.7645         Average Accuracy:          Accuracy: 87.4
Epoch: 6/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 89.23
Epoch: 7/50. Train set: Average loss: 222.4207         Average Accuracy:          Accuracy: 88.1
Epoch: 7/50. Train set: Average loss: 222.4207         Average Accuracy:          Accuracy: 88.1
Epoch: 7/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 90.44
Epoch: 8/50. Train set: Average loss: 222.1505         Average Accuracy:          Accuracy: 88.7
Epoch: 8/50. Train set: Average loss: 222.1505         Average Accuracy:          Accuracy: 88.7
Epoch: 8/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 89.95
Epoch: 9/50. Train set: Average loss: 221.9600         Average Accuracy:          Accuracy: 89.0
Epoch: 9/50. Train set: Average loss: 221.9600         Average Accuracy:          Accuracy: 89.0
Epoch: 9/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 90.7
Epoch: 10/50. Train set: Average loss: 221.7382         Average Accuracy:          Accuracy: 89
Epoch: 10/50. Train set: Average loss: 221.7382         Average Accuracy:          Accuracy: 89
Epoch: 10/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 89.83
Epoch: 11/50. Train set: Average loss: 221.5612         Average Accuracy:          Accuracy: 89
Epoch: 11/50. Train set: Average loss: 221.5612         Average Accuracy:          Accuracy: 89
Epoch: 11/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 90.61
Epoch: 12/50. Train set: Average loss: 221.2601         Average Accuracy:          Accuracy: 89
Epoch: 12/50. Train set: Average loss: 221.2601         Average Accuracy:          Accuracy: 89
Epoch: 12/50. Test set: Average loss: 0.0000         Average Accuracy:          Accuracy: 90.59
Epoch: 13/50. Train set: Average loss: 220.9629         Average Accuracy:          Accuracy: 89
Epoch: 13/50. Train set: Average loss: 220.9629         Average Accuracy:          Accuracy: 89
```

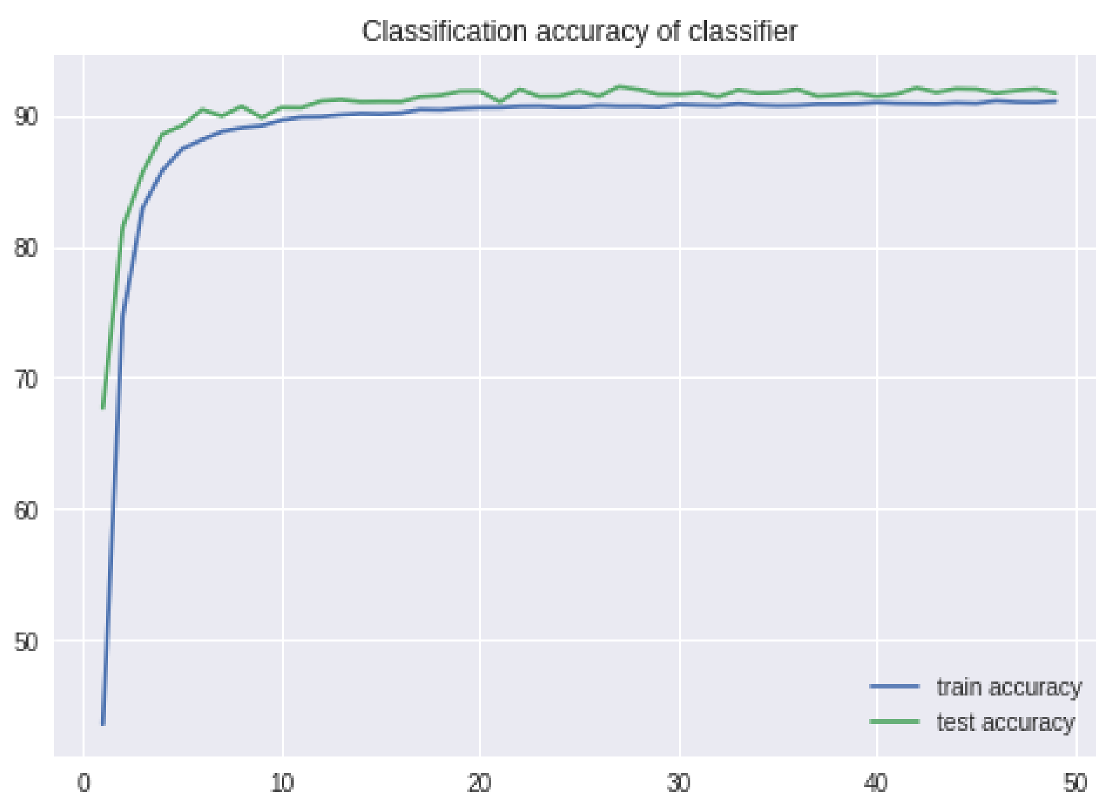
```
Epoch: 13/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.1
Epoch: 14/50. Train set: Average loss: 220.8702        Average Accuracy:        Accuracy: 90
Epoch: 14/50. Train set: Average loss: 220.8702        Average Accuracy:        Accuracy: 90
Epoch: 14/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.22
Epoch: 15/50. Train set: Average loss: 220.5683        Average Accuracy:        Accuracy: 90
Epoch: 15/50. Train set: Average loss: 220.5683        Average Accuracy:        Accuracy: 90
Epoch: 15/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.03
Epoch: 16/50. Train set: Average loss: 220.4476        Average Accuracy:        Accuracy: 90
Epoch: 16/50. Train set: Average loss: 220.4476        Average Accuracy:        Accuracy: 90
Epoch: 16/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.05
Epoch: 17/50. Train set: Average loss: 220.1611        Average Accuracy:        Accuracy: 90
Epoch: 17/50. Train set: Average loss: 220.1611        Average Accuracy:        Accuracy: 90
Epoch: 17/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.04
Epoch: 18/50. Train set: Average loss: 220.0744        Average Accuracy:        Accuracy: 90
Epoch: 18/50. Train set: Average loss: 220.0744        Average Accuracy:        Accuracy: 90
Epoch: 18/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.42
Epoch: 19/50. Train set: Average loss: 219.8210        Average Accuracy:        Accuracy: 90
Epoch: 19/50. Train set: Average loss: 219.8210        Average Accuracy:        Accuracy: 90
Epoch: 19/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.53
Epoch: 20/50. Train set: Average loss: 219.7000        Average Accuracy:        Accuracy: 90
Epoch: 20/50. Train set: Average loss: 219.7000        Average Accuracy:        Accuracy: 90
Epoch: 20/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.84
Epoch: 21/50. Train set: Average loss: 219.4364        Average Accuracy:        Accuracy: 90
Epoch: 21/50. Train set: Average loss: 219.4364        Average Accuracy:        Accuracy: 90
Epoch: 21/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.85
Epoch: 22/50. Train set: Average loss: 219.2481        Average Accuracy:        Accuracy: 90
Epoch: 22/50. Train set: Average loss: 219.2481        Average Accuracy:        Accuracy: 90
Epoch: 22/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.04
Epoch: 23/50. Train set: Average loss: 219.0282        Average Accuracy:        Accuracy: 90
Epoch: 23/50. Train set: Average loss: 219.0282        Average Accuracy:        Accuracy: 90
Epoch: 23/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.99
Epoch: 24/50. Train set: Average loss: 218.8495        Average Accuracy:        Accuracy: 90
Epoch: 24/50. Train set: Average loss: 218.8495        Average Accuracy:        Accuracy: 90
Epoch: 24/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.44
Epoch: 25/50. Train set: Average loss: 218.8220        Average Accuracy:        Accuracy: 90
Epoch: 25/50. Train set: Average loss: 218.8220        Average Accuracy:        Accuracy: 90
Epoch: 25/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.47
Epoch: 26/50. Train set: Average loss: 218.7407        Average Accuracy:        Accuracy: 90
Epoch: 26/50. Train set: Average loss: 218.7407        Average Accuracy:        Accuracy: 90
Epoch: 26/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.86
Epoch: 27/50. Train set: Average loss: 218.6563        Average Accuracy:        Accuracy: 90
Epoch: 27/50. Train set: Average loss: 218.6563        Average Accuracy:        Accuracy: 90
Epoch: 27/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.47
Epoch: 28/50. Train set: Average loss: 218.3506        Average Accuracy:        Accuracy: 90
Epoch: 28/50. Train set: Average loss: 218.3506        Average Accuracy:        Accuracy: 90
Epoch: 28/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 92.2
Epoch: 29/50. Train set: Average loss: 218.4039        Average Accuracy:        Accuracy: 90
Epoch: 29/50. Train set: Average loss: 218.4039        Average Accuracy:        Accuracy: 90
```

```
Epoch: 29/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.97
Epoch: 30/50. Train set: Average loss: 218.1558        Average Accuracy:        Accuracy: 90
Epoch: 30/50. Train set: Average loss: 218.1558        Average Accuracy:        Accuracy: 90
Epoch: 30/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.62
Epoch: 31/50. Train set: Average loss: 218.1680        Average Accuracy:        Accuracy: 90
Epoch: 31/50. Train set: Average loss: 218.1680        Average Accuracy:        Accuracy: 90
Epoch: 31/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.59
Epoch: 32/50. Train set: Average loss: 217.9056        Average Accuracy:        Accuracy: 90
Epoch: 32/50. Train set: Average loss: 217.9056        Average Accuracy:        Accuracy: 90
Epoch: 32/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.74
Epoch: 33/50. Train set: Average loss: 217.6832        Average Accuracy:        Accuracy: 90
Epoch: 33/50. Train set: Average loss: 217.6832        Average Accuracy:        Accuracy: 90
Epoch: 33/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.43
Epoch: 34/50. Train set: Average loss: 217.9556        Average Accuracy:        Accuracy: 90
Epoch: 34/50. Train set: Average loss: 217.9556        Average Accuracy:        Accuracy: 90
Epoch: 34/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.93
Epoch: 35/50. Train set: Average loss: 217.5710        Average Accuracy:        Accuracy: 90
Epoch: 35/50. Train set: Average loss: 217.5710        Average Accuracy:        Accuracy: 90
Epoch: 35/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.7
Epoch: 36/50. Train set: Average loss: 217.4137        Average Accuracy:        Accuracy: 90
Epoch: 36/50. Train set: Average loss: 217.4137        Average Accuracy:        Accuracy: 90
Epoch: 36/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.75
Epoch: 37/50. Train set: Average loss: 217.4305        Average Accuracy:        Accuracy: 90
Epoch: 37/50. Train set: Average loss: 217.4305        Average Accuracy:        Accuracy: 90
Epoch: 37/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.97
Epoch: 38/50. Train set: Average loss: 217.3812        Average Accuracy:        Accuracy: 90
Epoch: 38/50. Train set: Average loss: 217.3812        Average Accuracy:        Accuracy: 90
Epoch: 38/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.47
Epoch: 39/50. Train set: Average loss: 217.3492        Average Accuracy:        Accuracy: 90
Epoch: 39/50. Train set: Average loss: 217.3492        Average Accuracy:        Accuracy: 90
Epoch: 39/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.56
Epoch: 40/50. Train set: Average loss: 217.1224        Average Accuracy:        Accuracy: 90
Epoch: 40/50. Train set: Average loss: 217.1224        Average Accuracy:        Accuracy: 90
Epoch: 40/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.69
Epoch: 41/50. Train set: Average loss: 217.0350        Average Accuracy:        Accuracy: 90
Epoch: 41/50. Train set: Average loss: 217.0350        Average Accuracy:        Accuracy: 90
Epoch: 41/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.45
Epoch: 42/50. Train set: Average loss: 217.0286        Average Accuracy:        Accuracy: 90
Epoch: 42/50. Train set: Average loss: 217.0286        Average Accuracy:        Accuracy: 90
Epoch: 42/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.63
Epoch: 43/50. Train set: Average loss: 216.7855        Average Accuracy:        Accuracy: 90
Epoch: 43/50. Train set: Average loss: 216.7855        Average Accuracy:        Accuracy: 90
Epoch: 43/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 92.11
Epoch: 44/50. Train set: Average loss: 216.8624        Average Accuracy:        Accuracy: 90
Epoch: 44/50. Train set: Average loss: 216.8624        Average Accuracy:        Accuracy: 90
Epoch: 44/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.75
Epoch: 45/50. Train set: Average loss: 216.6823        Average Accuracy:        Accuracy: 90
Epoch: 45/50. Train set: Average loss: 216.6823        Average Accuracy:        Accuracy: 90
```

```
Epoch: 45/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 92.04
Epoch: 46/50. Train set: Average loss: 216.6726        Average Accuracy:        Accuracy: 90
Epoch: 46/50. Train set: Average loss: 216.6726        Average Accuracy:        Accuracy: 90
Epoch: 46/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 92.0
Epoch: 47/50. Train set: Average loss: 216.6122        Average Accuracy:        Accuracy: 91
Epoch: 47/50. Train set: Average loss: 216.6122        Average Accuracy:        Accuracy: 91
Epoch: 47/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.7
Epoch: 48/50. Train set: Average loss: 216.3396        Average Accuracy:        Accuracy: 91
Epoch: 48/50. Train set: Average loss: 216.3396        Average Accuracy:        Accuracy: 91
Epoch: 48/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.89
Epoch: 49/50. Train set: Average loss: 216.3508        Average Accuracy:        Accuracy: 90
Epoch: 49/50. Train set: Average loss: 216.3508        Average Accuracy:        Accuracy: 90
Epoch: 49/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 92.01
Epoch: 50/50. Train set: Average loss: 216.3538        Average Accuracy:        Accuracy: 91
Epoch: 50/50. Train set: Average loss: 216.3538        Average Accuracy:        Accuracy: 91
Epoch: 50/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.71
```

In [0]:
```python
plt.style.use("seaborn")
plt.plot(range(1,50),train_accuracy)
plt.plot(range(1,50),test_accuracy)
plt.title("Classification accuracy of classifier")
plt.legend(["train accuracy","test accuracy"])
```

Out[0]: <matplotlib.legend.Legend at 0x7fde9a039c18>

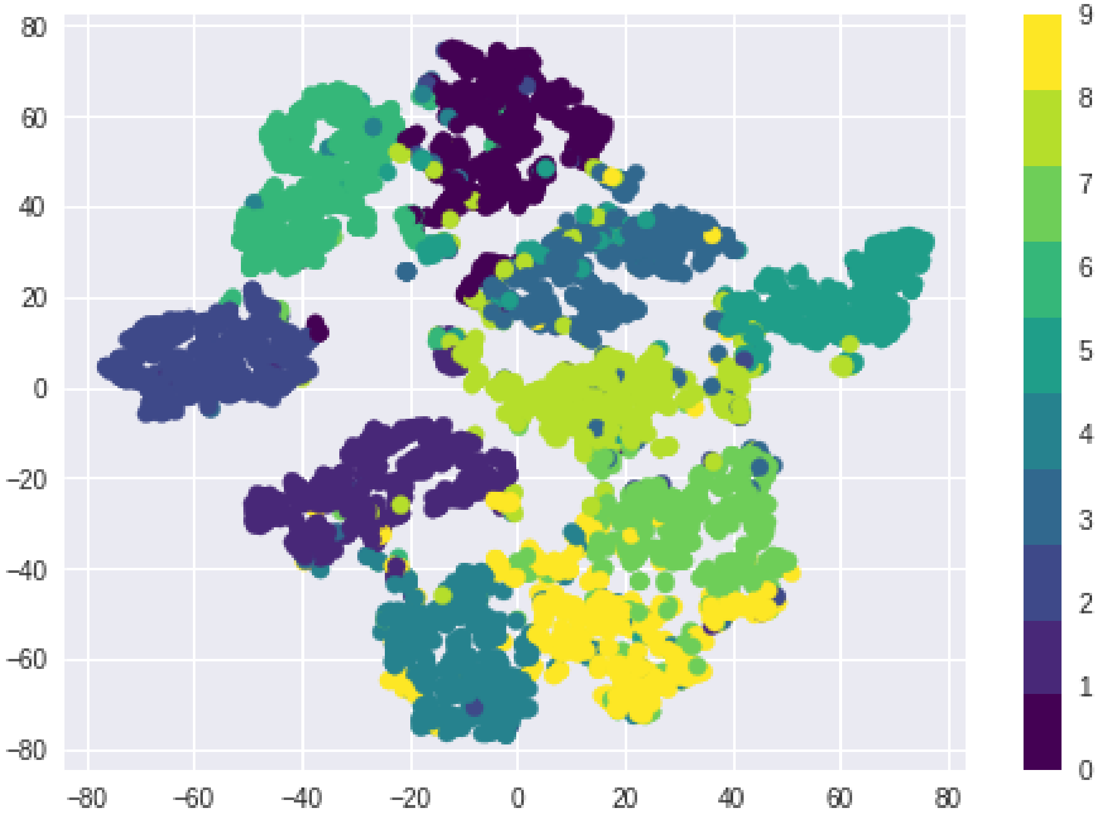


```
In [0]: print(max(train_accuracy),max(test_accuracy))
```

```
91.115 92.2
```

```
In [0]: path = "tsne_vae_gan+classifier+vae+train.png"
        def visualize_tsne(X, labels, model, path):
            # Compute latent space representation
            print("Computing latent space projection...")

            X_encoded, _ = model.encoder(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].cuda(),test_labels[:5000],G,path)
```

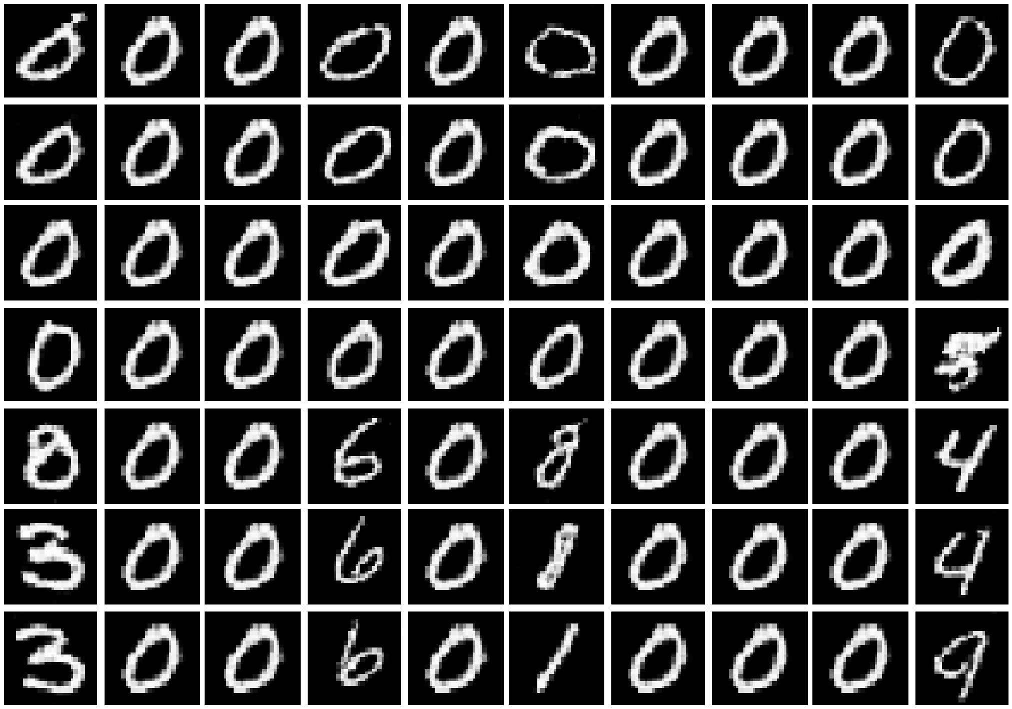
```
Computing latent space projection...
```

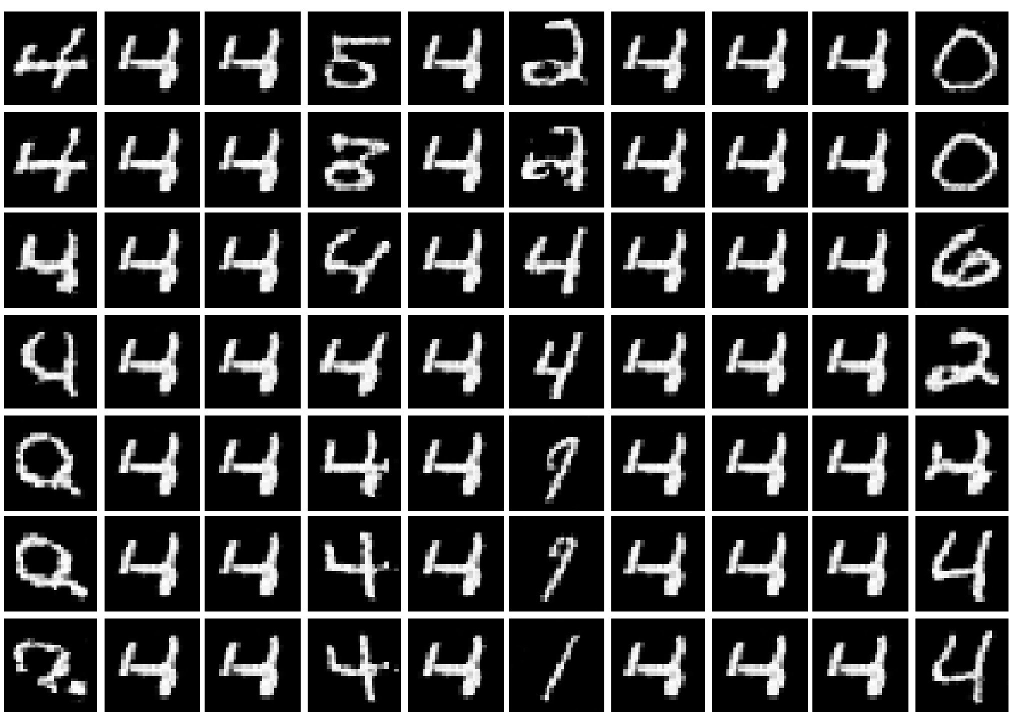


In [0]:
```
## latent traversals
#os.makedirs("vae-gan")
testpoint1, testpointlabel1 = train_loader.dataset[0]
testpoint2, testpoint2label2 = train_loader.dataset[1]
testpoint3, testpoint3label3 = train_loader.dataset[2]
traverse_latents(G, testpoint1,Params.nb_latents, epoch,1,"vae-gan")
traverse_latents(G, testpoint2,Params.nb_latents, epoch,2,"vae-gan")
traverse_latents(G, testpoint3,Params.nb_latents, epoch,3,"vae-gan")
```

In [0]:
```
from IPython.display import Image
import matplotlib.image as mpimg

img=mpimg.imread("vae-gan/traversal_49_1.png")
plt.imshow(img)
plt.axis("off")
plt.show()

img=mpimg.imread("vae-gan/traversal_49_2.png")
plt.imshow(img)
plt.axis("off")
plt.show()
```

```
img=mpimg.imread("vae-gan/traversal_49_3.png")
plt.imshow(img)
plt.axis("off")
plt.show()
```





```
In [0]:
```

```
In [0]:
```