# beta_vae+classifier

April 26, 2019

### 0.1 Beta -Variational AutoEnoder:

If each variable in the inferred latent representation z is only sensitive to one single generative factor and relatively invariant to other factors, we will say this representation is disentangled or factorized. One benefit that often comes with disentangled representation is good interpretability and easy generalization to a variety of tasks.

For example, a model trained on photos of human faces might capture the gentle, skin color, hair color, hair length, emotion, whether wearing a pair of glasses and many other relatively independent factors in separate dimensions. Such a disentangled representation is very beneficial to facial image generation.

-VAE (Higgins et al., 2017) is a modification of Variational Autoencoder with a special emphasis to discover disentangled latent factors. Following the same incentive in VAE, we want to maximize the probability of generating real data, while keeping the distance between the real and estimated posterior distributions small (say, under a small constant ):

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

```
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


%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]: ## 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 0x7fa38ae965f0>
```

In [0]:
```
"""
Traverse Latents
"""
def traverse_latents(model, datapoint, nb_latents, file):
  model.eval()
  datapoint = datapoint.to(device)
  if isinstance(model,ConvVAE):
    datapoint = datapoint.unsqueeze(0)
    mu, _ = model.encode(datapoint)
  else:
    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.decode(muc).cpu() if isinstance(model,ConvVAE)
                else model.decoder(muc).cpu()
      recons[i, zi] = recon.view(28, 28)

  torchvision.utils.save_image(recons.view(-1, 1, 28, 28), file,
                               nrow=nb_latents, pad_value=1)
```

## 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 [ ]: path = "./vae-classifier/"
        transformation = transforms.Compose([transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))])
        train_dataset = datasets.MNIST(os.path.join(path,"MNIST/data"),
                        train=True,transform=transformation,download=True)
        test_dataset = datasets.MNIST(os.path.join(path,"MNIST/data"),
                        train=False,transform=transformation,download=True)
        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(train_dataset,
                      batch_size=len(train_dataset),shuffle=True)
        test_data, test_labels = next(iter(testing_tsne))[:10000]

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

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


In [0]: %ls

results/  sample_data/  vae-classifier/
```

### 0.4 Model

```
In [0]: class ConvVAE(nn.Module):
            def __init__(self, nb_latents):
                super(ConvVAE, self).__init__()
                self.conv1 = nn.Sequential(
                    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=6):
          """
          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("beta-results")
        %ls

        save_dir = "beta-results"

beta-results/  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=save_dir):
          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) + '.png')
        save_image(recons.view(-1, 1, 28, 28),
                   filename, nrow=nb_latents, pad_value=1)


    ### Plot TSNE for latent space
    # Show dataset images with T-sne projection of latent space encoding
    import seaborn as sns
    palette = np.array(sns.color_palette("hls", 10))
    def visualize_tsne_of_input(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()
        im = plt.scatter(X_tsne[:,0], X_tsne[:,1],
                         c=[palette[i] for i in labels])
        fig.colorbar(im, ticks=range(10));
        fig.savefig(path, dpi=fig.dpi)
```

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

```
Requirement already satisfied: opencv-python in /home/chivukula_manju/yes/lib/python3.6/site-pa
Requirement already satisfied: numpy>=1.11.3 in /home/chivukula_manju/yes/lib/python3.6/site-pa
You are using pip version 9.0.1, however version 19.0.3 is available.You should consider upgra
```

### 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)

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

            return test_loss()
```

In [0]:
```
train_losses = []
test_losses = []
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.epochs, train_loss)
  print(message)
  test_loss = test_epoch(epoch)
  message = 'Epoch: {}/{}. Test set:' +
     ' Average loss: {:.4f}'.format(epoch + 1,
                    Params.epochs, test_loss)
  test_losses.append(test_loss)
  print(message)
```

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


Epoch: 1/100. Train set: Average loss: -10349.4857
Epoch: 1/100. Test set: Average loss: -10544.8887
Epoch: 2/100. Train set: Average loss: -10452.9163
Epoch: 2/100. Test set: Average loss: -10587.9207
Epoch: 3/100. Train set: Average loss: -10512.6402
Epoch: 3/100. Test set: Average loss: -10684.3192
Epoch: 4/100. Train set: Average loss: -10562.4123
Epoch: 4/100. Test set: Average loss: -10696.9424
Epoch: 5/100. Train set: Average loss: -10611.1751
Epoch: 5/100. Test set: Average loss: -10748.8865
Epoch: 6/100. Train set: Average loss: -10661.2261
Epoch: 6/100. Test set: Average loss: -10813.1165
Epoch: 7/100. Train set: Average loss: -10708.5552
Epoch: 7/100. Test set: Average loss: -10825.0708
```

```
Epoch: 8/100. Train set: Average loss: -10744.1072
Epoch: 8/100. Test set: Average loss: -10855.0397
Epoch: 9/100. Train set: Average loss: -10763.4951
Epoch: 9/100. Test set: Average loss: -10909.9545
Epoch: 10/100. Train set: Average loss: -10783.4877
Epoch: 10/100. Test set: Average loss: -10902.9798
Epoch: 11/100. Train set: Average loss: -10799.0980
Epoch: 11/100. Test set: Average loss: -10885.9479
Epoch: 12/100. Train set: Average loss: -10813.8132
Epoch: 12/100. Test set: Average loss: -10896.8893
Epoch: 13/100. Train set: Average loss: -10830.2422
Epoch: 13/100. Test set: Average loss: -10931.7071
Epoch: 14/100. Train set: Average loss: -10835.9709
Epoch: 14/100. Test set: Average loss: -10937.8951
Epoch: 15/100. Train set: Average loss: -10852.6770
Epoch: 15/100. Test set: Average loss: -10942.0391
Epoch: 16/100. Train set: Average loss: -10860.5345
Epoch: 16/100. Test set: Average loss: -10955.7746
Epoch: 17/100. Train set: Average loss: -10869.8607
Epoch: 17/100. Test set: Average loss: -10967.9799
Epoch: 18/100. Train set: Average loss: -10874.4623
Epoch: 18/100. Test set: Average loss: -10985.3355
Epoch: 19/100. Train set: Average loss: -10890.4998
Epoch: 19/100. Test set: Average loss: -10977.6311
Epoch: 20/100. Train set: Average loss: -10899.1664
Epoch: 20/100. Test set: Average loss: -10981.4035
Epoch: 21/100. Train set: Average loss: -10903.4473
Epoch: 21/100. Test set: Average loss: -10978.4739
Epoch: 22/100. Train set: Average loss: -10912.3935
Epoch: 22/100. Test set: Average loss: -10983.2029
Epoch: 23/100. Train set: Average loss: -10915.2949
Epoch: 23/100. Test set: Average loss: -10967.9590
Epoch: 24/100. Train set: Average loss: -10918.3851
Epoch: 24/100. Test set: Average loss: -10977.4329
Epoch: 25/100. Train set: Average loss: -10925.6430
Epoch: 25/100. Test set: Average loss: -10960.0561
Epoch: 26/100. Train set: Average loss: -10930.3654
Epoch: 26/100. Test set: Average loss: -10987.0091
Epoch: 27/100. Train set: Average loss: -10930.2887
Epoch: 27/100. Test set: Average loss: -11004.4141
Epoch: 28/100. Train set: Average loss: -10935.9621
Epoch: 28/100. Test set: Average loss: -11006.9101
Epoch: 29/100. Train set: Average loss: -10939.9603
Epoch: 29/100. Test set: Average loss: -10992.1838
Epoch: 30/100. Train set: Average loss: -10946.9011
Epoch: 30/100. Test set: Average loss: -10975.8259
Epoch: 31/100. Train set: Average loss: -10949.9610
Epoch: 31/100. Test set: Average loss: -11004.9979
```

```
Epoch: 32/100. Train set: Average loss: -10952.7151
Epoch: 32/100. Test set: Average loss: -11015.1051
Epoch: 33/100. Train set: Average loss: -10959.3073
Epoch: 33/100. Test set: Average loss: -10982.9964
Epoch: 34/100. Train set: Average loss: -10961.4371
Epoch: 34/100. Test set: Average loss: -11008.3863
Epoch: 35/100. Train set: Average loss: -10963.2451
Epoch: 35/100. Test set: Average loss: -11008.2428
Epoch: 36/100. Train set: Average loss: -10966.7791
Epoch: 36/100. Test set: Average loss: -11011.4464
Epoch: 37/100. Train set: Average loss: -10969.2139
Epoch: 37/100. Test set: Average loss: -11000.2670
Epoch: 38/100. Train set: Average loss: -10973.4729
Epoch: 38/100. Test set: Average loss: -11013.9935
Epoch: 39/100. Train set: Average loss: -10974.0112
Epoch: 39/100. Test set: Average loss: -11007.8988
Epoch: 40/100. Train set: Average loss: -10975.1199
Epoch: 40/100. Test set: Average loss: -11001.5884
Epoch: 41/100. Train set: Average loss: -10978.4597
Epoch: 41/100. Test set: Average loss: -11031.9370
Epoch: 42/100. Train set: Average loss: -10984.2374
Epoch: 42/100. Test set: Average loss: -10990.9801
Epoch: 43/100. Train set: Average loss: -10986.2278
Epoch: 43/100. Test set: Average loss: -11009.4645
Epoch: 44/100. Train set: Average loss: -10986.6053
Epoch: 44/100. Test set: Average loss: -11016.3920
Epoch: 45/100. Train set: Average loss: -10988.7983
Epoch: 45/100. Test set: Average loss: -11015.9668
Epoch: 46/100. Train set: Average loss: -10992.2750
Epoch: 46/100. Test set: Average loss: -10995.0556
Epoch: 47/100. Train set: Average loss: -10992.0277
Epoch: 47/100. Test set: Average loss: -10983.6734
Epoch: 48/100. Train set: Average loss: -10994.8473
Epoch: 48/100. Test set: Average loss: -11002.2136
Epoch: 49/100. Train set: Average loss: -10995.0274
Epoch: 49/100. Test set: Average loss: -11023.6951
Epoch: 50/100. Train set: Average loss: -10997.7335
Epoch: 50/100. Test set: Average loss: -11019.6370
Epoch: 51/100. Train set: Average loss: -11002.3454
Epoch: 51/100. Test set: Average loss: -11006.3942
Epoch: 52/100. Train set: Average loss: -11001.4557
Epoch: 52/100. Test set: Average loss: -11022.6608
Epoch: 53/100. Train set: Average loss: -11003.6703
Epoch: 53/100. Test set: Average loss: -11003.6906
Epoch: 54/100. Train set: Average loss: -11005.1104
Epoch: 54/100. Test set: Average loss: -10993.1252
Epoch: 55/100. Train set: Average loss: -11006.1172
Epoch: 55/100. Test set: Average loss: -10999.5482
```

```
Epoch: 56/100. Train set: Average loss: -11008.0077
Epoch: 56/100. Test set: Average loss: -11021.4441
Epoch: 57/100. Train set: Average loss: -11008.7850
Epoch: 57/100. Test set: Average loss: -11020.0875
Epoch: 58/100. Train set: Average loss: -11011.2379
Epoch: 58/100. Test set: Average loss: -11015.5271
Epoch: 59/100. Train set: Average loss: -11013.7164
Epoch: 59/100. Test set: Average loss: -11026.9246
Epoch: 60/100. Train set: Average loss: -11012.7646
Epoch: 60/100. Test set: Average loss: -11012.6244
Epoch: 61/100. Train set: Average loss: -11013.7034
Epoch: 61/100. Test set: Average loss: -11010.1708
Epoch: 62/100. Train set: Average loss: -11016.8086
Epoch: 62/100. Test set: Average loss: -11031.8402
Epoch: 63/100. Train set: Average loss: -11017.4972
Epoch: 63/100. Test set: Average loss: -11021.7985
Epoch: 64/100. Train set: Average loss: -11016.6438
Epoch: 64/100. Test set: Average loss: -10994.2679
Epoch: 65/100. Train set: Average loss: -11019.7879
Epoch: 65/100. Test set: Average loss: -11008.0334
Epoch: 66/100. Train set: Average loss: -11018.8163
Epoch: 66/100. Test set: Average loss: -11001.5700
Epoch: 67/100. Train set: Average loss: -11021.1522
Epoch: 67/100. Test set: Average loss: -11028.0907
Epoch: 68/100. Train set: Average loss: -11022.9235
Epoch: 68/100. Test set: Average loss: -11006.5722
Epoch: 69/100. Train set: Average loss: -11023.1485
Epoch: 69/100. Test set: Average loss: -11008.3074
Epoch: 70/100. Train set: Average loss: -11024.1373
Epoch: 70/100. Test set: Average loss: -11010.4841
Epoch: 71/100. Train set: Average loss: -11026.2412
Epoch: 71/100. Test set: Average loss: -11011.2571
Epoch: 72/100. Train set: Average loss: -11027.5816
Epoch: 72/100. Test set: Average loss: -11031.9309
Epoch: 73/100. Train set: Average loss: -11029.1204
Epoch: 73/100. Test set: Average loss: -10991.2853
Epoch: 74/100. Train set: Average loss: -11026.8116
Epoch: 74/100. Test set: Average loss: -11024.3664
Epoch: 75/100. Train set: Average loss: -11029.0669
Epoch: 75/100. Test set: Average loss: -10992.8561
Epoch: 76/100. Train set: Average loss: -11031.9145
Epoch: 76/100. Test set: Average loss: -11003.8241
Epoch: 77/100. Train set: Average loss: -11032.3702
Epoch: 77/100. Test set: Average loss: -11022.3059
Epoch: 78/100. Train set: Average loss: -11032.2286
Epoch: 78/100. Test set: Average loss: -11029.7318
Epoch: 79/100. Train set: Average loss: -11034.1605
Epoch: 79/100. Test set: Average loss: -11016.2777
```

```
Epoch: 80/100. Train set: Average loss: -11036.1964
Epoch: 80/100. Test set: Average loss: -11026.5891
Epoch: 81/100. Train set: Average loss: -11036.9252
Epoch: 81/100. Test set: Average loss: -11008.2321
Epoch: 82/100. Train set: Average loss: -11037.5574
Epoch: 82/100. Test set: Average loss: -10998.7115
Epoch: 83/100. Train set: Average loss: -11036.8492
Epoch: 83/100. Test set: Average loss: -11009.0513
Epoch: 84/100. Train set: Average loss: -11039.0210
Epoch: 84/100. Test set: Average loss: -11020.4130
Epoch: 85/100. Train set: Average loss: -11038.9508
Epoch: 85/100. Test set: Average loss: -11016.6073
Epoch: 86/100. Train set: Average loss: -11039.4227
Epoch: 86/100. Test set: Average loss: -11011.6423
Epoch: 87/100. Train set: Average loss: -11040.1777
Epoch: 87/100. Test set: Average loss: -11015.3848
Epoch: 88/100. Train set: Average loss: -11038.0958
Epoch: 88/100. Test set: Average loss: -11004.8606
Epoch: 89/100. Train set: Average loss: -11040.6147
Epoch: 89/100. Test set: Average loss: -11020.3086
Epoch: 90/100. Train set: Average loss: -11041.2143
Epoch: 90/100. Test set: Average loss: -11002.2994
Epoch: 91/100. Train set: Average loss: -11043.8785
Epoch: 91/100. Test set: Average loss: -11019.4219
Epoch: 92/100. Train set: Average loss: -11045.6124
Epoch: 92/100. Test set: Average loss: -10999.1384
Epoch: 93/100. Train set: Average loss: -11043.5832
Epoch: 93/100. Test set: Average loss: -11004.1811
Epoch: 94/100. Train set: Average loss: -11043.8821
Epoch: 94/100. Test set: Average loss: -10998.8833
Epoch: 95/100. Train set: Average loss: -11046.3663
Epoch: 95/100. Test set: Average loss: -11007.7211
Epoch: 96/100. Train set: Average loss: -11046.6947
Epoch: 96/100. Test set: Average loss: -11008.7210
Epoch: 97/100. Train set: Average loss: -11047.7590
Epoch: 97/100. Test set: Average loss: -10996.8445
Epoch: 98/100. Train set: Average loss: -11046.2223
Epoch: 98/100. Test set: Average loss: -11001.2824
Epoch: 99/100. Train set: Average loss: -11047.3682
Epoch: 99/100. Test set: Average loss: -11009.4789
Epoch: 100/100. Train set: Average loss: -11047.8037
Epoch: 100/100. Test set: Average loss: -11005.8705
```

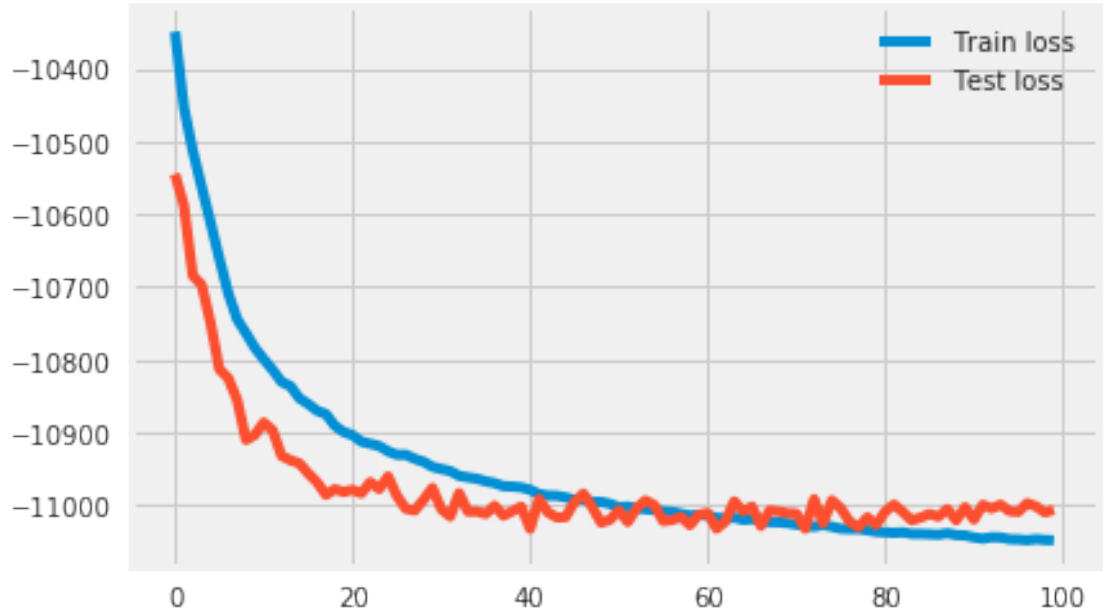
---

```
TypeError                                 Traceback (most recent call last)
```

```
<ipython-input-49-1f4fefb2aaed> in <module>()
 12
 13
---> 14 traverse_latents(model, Params.nb_latents, epoch,save_dir)
 15 plt.plot(range(Params.epochs),train_losses)
 16 plt.plot(range(Params.epochs),test_losses)


TypeError: traverse_latents() missing 1 required positional argument: 'batch_idx'
```

In [0]:
```
# Plot Losses
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/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```




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

path = save_dir+ "/model_state_" + str(epoch) + ".pth"
save_model(model,path)
```

## 0.8 Reconstructions

In [0]:
```
epoch = 99
path = save_dir+ "/model_state_" + str(epoch) + ".pth"

model.load_state_dict(torch.load(path))
```

In [0]:
```
def reconstruction(data,epoch, is_train = True):
    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), epoch, True)
reconstruction(test_batch.to(device), epoch, False)
```

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

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

### 0.8.1 Train Reconstruction

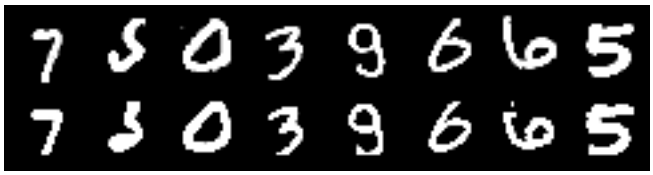
In [0]:
```
from IPython.display import Image
Image(filename=save_dir+'/train_reconstruction_99.png')
```

Out[0]:

### 0.8.2 Test Reconstruction

```
In [0]: Image(filename=save_dir+'/test_reconstruction_99.png')
```

Out[0]:

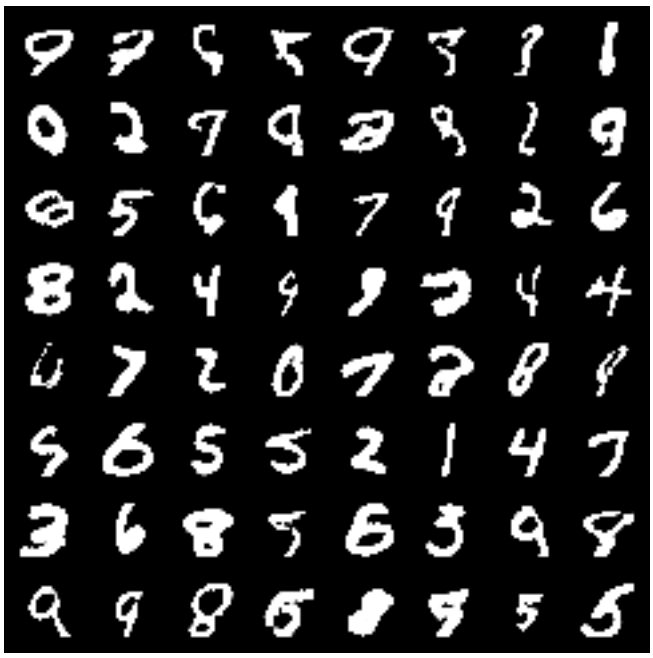


## 0.9 Generated Samples

```
In [0]: # Generate samples of reconstruction
        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),
                 save_dir + '/sample_' + str(epoch) + '.png')
```

```
In [0]: from IPython.display import Image
        Image(filename=save_dir+'/sample_99.png')
```

Out[0]:

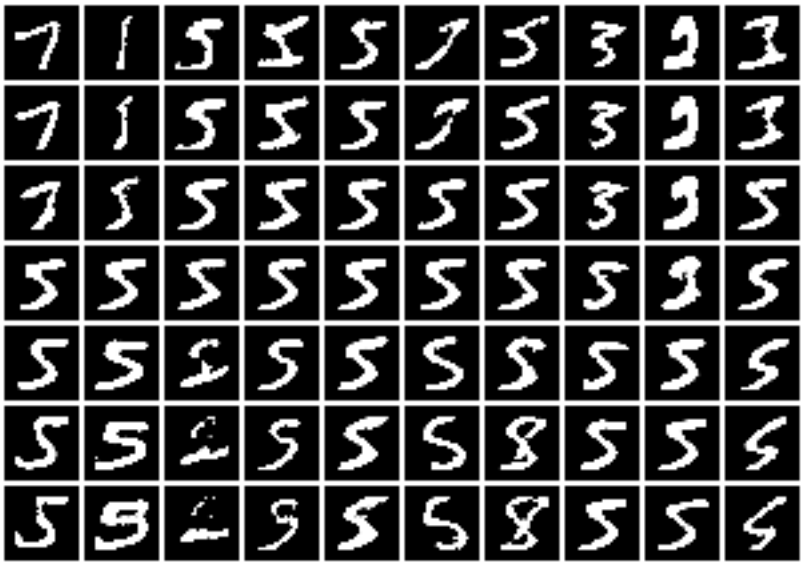


### 0.10 Latent Traversals

```
In [0]: testpoint = torch.Tensor(train_loader.dataset[0][0]).to(device)
        traverse_latents(model, testpoint,
            Params.nb_latents, epoch,1,save_dir)
```

```
In [0]: Image(filename=save_dir+'/traversal_99_1.png')
```

Out[0]:



```
In [0]:
```

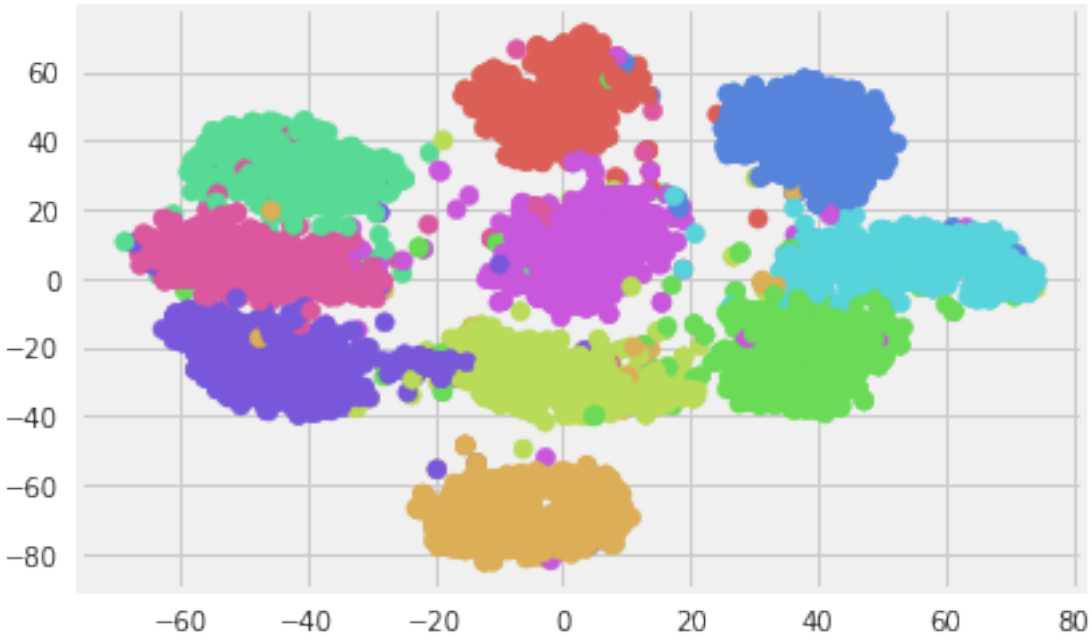
### 0.11 TSNE Plot

```
In [0]: 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.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()
            im = plt.scatter(X_tsne[:,0], X_tsne[:,1],c=[palette[i]
```

```
                        for i in labels])

            fig.savefig(path, dpi=fig.dpi)

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



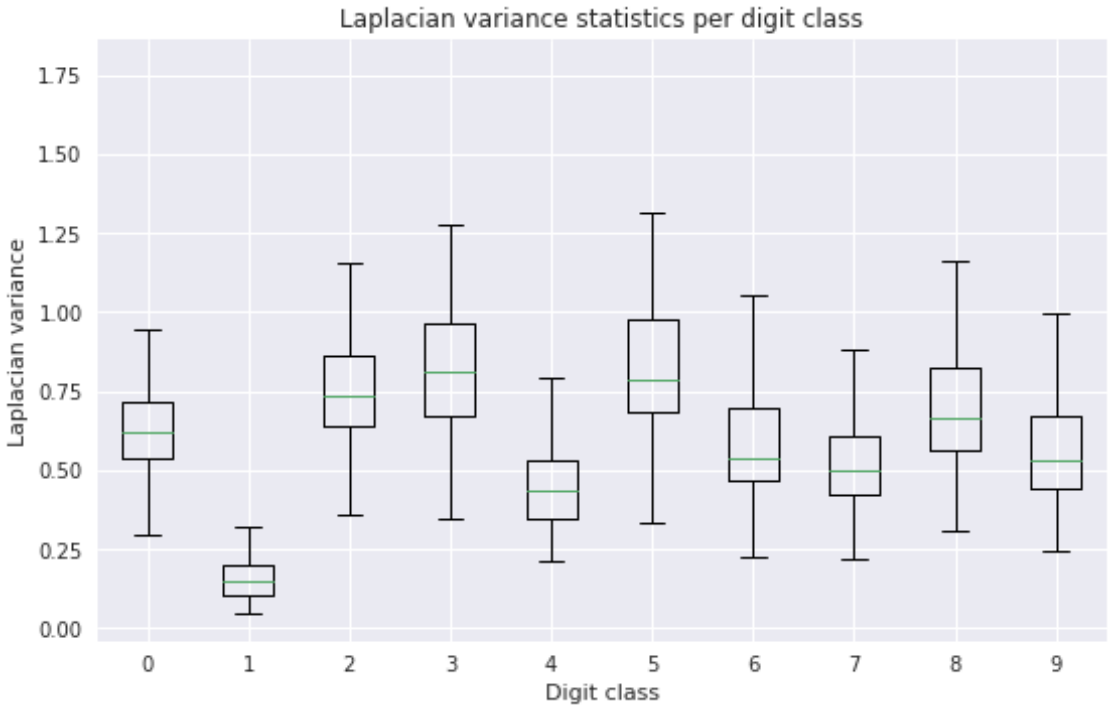

## 0.12 Load model

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]))
```

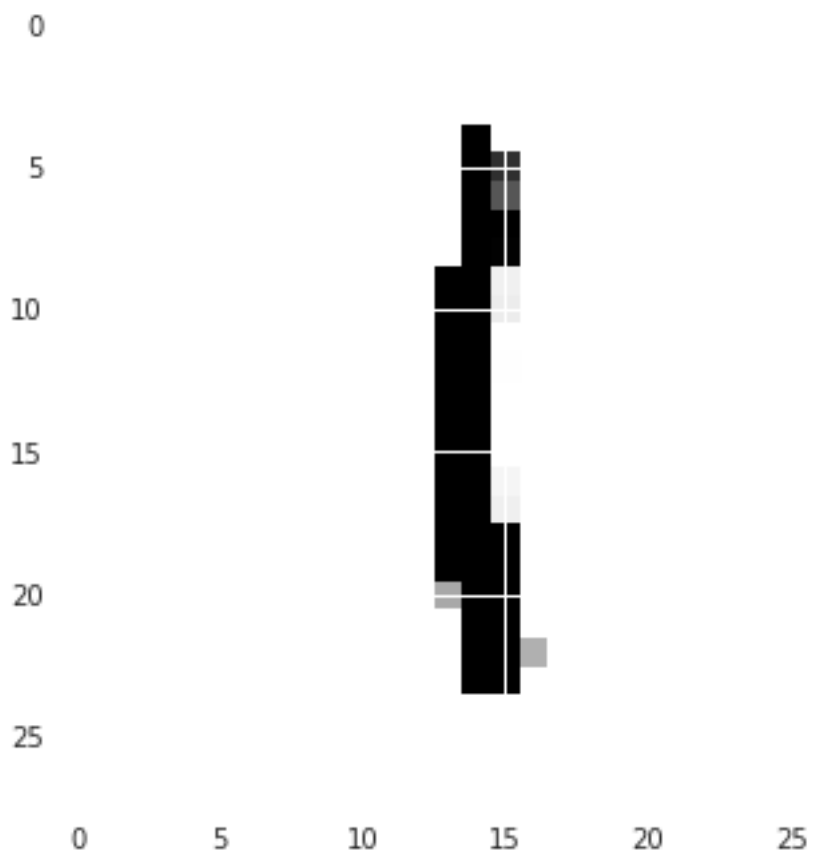



```
In [0]: with torch.no_grad():
            reconstructions = np.empty(shape=(227,1,28,28))
            indx = 0
            for i, (x,y) in enumerate(zip(x_test,y_test)):
                if y == 1:
                    recon_batch, mu, logvar = model(x.unsqueeze(1).to(device))
                    reconstructions[indx]=
                    (recon_batch.squeeze(0).detach().cpu())
                    indx+=1
```

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

```
Out[0]: <matplotlib.image.AxesImage at 0x7fdebee7b470>
```

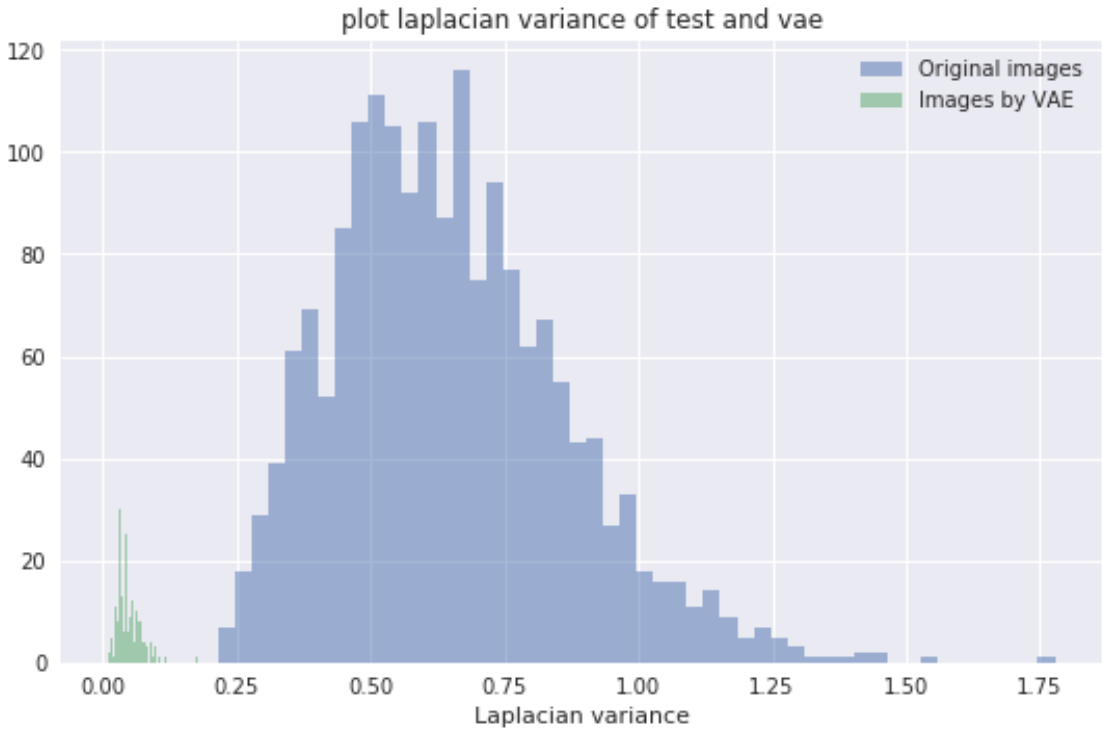
```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (prop.get_family(), self.defaultFamily[fontext]))
```




```
In [0]: laplacian_variances_recons = [laplacian_variance_numpy
                            (np.array(reconstructions[y_test == i], dtype=np.uint8))
                                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();

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

```
/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWar
  (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:

### 0.13 Model 1

- Train VAE
- Test Classifier

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

Out[0]: <torch._C.Generator at 0x7f535c0d2c10>

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):
  classifier.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: 0.0134        Average Accuracy:        Accuracy: 52.31
Epoch: 2/50. Train set: Average loss: 0.0134        Average Accuracy:        Accuracy: 52.31
Epoch: 2/50. Test set: Average loss: 0.0099        Average Accuracy:        Accuracy: 85.35
Epoch: 3/50. Train set: Average loss: 0.0071        Average Accuracy:        Accuracy: 88.91
Epoch: 3/50. Train set: Average loss: 0.0071        Average Accuracy:        Accuracy: 88.91
Epoch: 3/50. Test set: Average loss: 0.0063        Average Accuracy:        Accuracy: 91.46
Epoch: 4/50. Train set: Average loss: 0.0048        Average Accuracy:        Accuracy: 91.64
Epoch: 4/50. Train set: Average loss: 0.0048        Average Accuracy:        Accuracy: 91.64
Epoch: 4/50. Test set: Average loss: 0.0045        Average Accuracy:        Accuracy: 92.62
Epoch: 5/50. Train set: Average loss: 0.0037        Average Accuracy:        Accuracy: 92.61
Epoch: 5/50. Train set: Average loss: 0.0037        Average Accuracy:        Accuracy: 92.61
Epoch: 5/50. Test set: Average loss: 0.0036        Average Accuracy:        Accuracy: 93.16
Epoch: 6/50. Train set: Average loss: 0.0031        Average Accuracy:        Accuracy: 93.11
Epoch: 6/50. Train set: Average loss: 0.0031        Average Accuracy:        Accuracy: 93.11
Epoch: 6/50. Test set: Average loss: 0.0032        Average Accuracy:        Accuracy: 93.48
Epoch: 7/50. Train set: Average loss: 0.0027        Average Accuracy:        Accuracy: 93.40
Epoch: 7/50. Train set: Average loss: 0.0027        Average Accuracy:        Accuracy: 93.40
Epoch: 7/50. Test set: Average loss: 0.0028        Average Accuracy:        Accuracy: 93.76
Epoch: 8/50. Train set: Average loss: 0.0024        Average Accuracy:        Accuracy: 93.62
```

```
Epoch: 8/50. Train set: Average loss: 0.0024        Average Accuracy:        Accuracy: 93.625
Epoch: 8/50. Test set: Average loss: 0.0024        Average Accuracy:        Accuracy: 93.85
Epoch: 9/50. Train set: Average loss: 0.0022        Average Accuracy:        Accuracy: 93.815
Epoch: 9/50. Train set: Average loss: 0.0022        Average Accuracy:        Accuracy: 93.815
Epoch: 9/50. Test set: Average loss: 0.0022        Average Accuracy:        Accuracy: 93.94
Epoch: 10/50. Train set: Average loss: 0.0021        Average Accuracy:        Accuracy: 93.93
Epoch: 10/50. Train set: Average loss: 0.0021        Average Accuracy:        Accuracy: 93.93
Epoch: 10/50. Test set: Average loss: 0.0021        Average Accuracy:        Accuracy: 94.01
Epoch: 11/50. Train set: Average loss: 0.0020        Average Accuracy:        Accuracy: 94.04
Epoch: 11/50. Train set: Average loss: 0.0020        Average Accuracy:        Accuracy: 94.04
Epoch: 11/50. Test set: Average loss: 0.0020        Average Accuracy:        Accuracy: 94.12
Epoch: 12/50. Train set: Average loss: 0.0019        Average Accuracy:        Accuracy: 94.11
Epoch: 12/50. Train set: Average loss: 0.0019        Average Accuracy:        Accuracy: 94.11
Epoch: 12/50. Test set: Average loss: 0.0020        Average Accuracy:        Accuracy: 94.17
Epoch: 13/50. Train set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.22
Epoch: 13/50. Train set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.22
Epoch: 13/50. Test set: Average loss: 0.0020        Average Accuracy:        Accuracy: 94.27
Epoch: 14/50. Train set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.27
Epoch: 14/50. Train set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.27
Epoch: 14/50. Test set: Average loss: 0.0024        Average Accuracy:        Accuracy: 94.25
Epoch: 15/50. Train set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.33
Epoch: 15/50. Train set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.33
Epoch: 15/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.32
Epoch: 16/50. Train set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.36
Epoch: 16/50. Train set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.36
Epoch: 16/50. Test set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.33
Epoch: 17/50. Train set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.41
Epoch: 17/50. Train set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.41
Epoch: 17/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.38
Epoch: 18/50. Train set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.44
Epoch: 18/50. Train set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.44
Epoch: 18/50. Test set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.42
Epoch: 19/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.48
Epoch: 19/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.48
Epoch: 19/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.52
Epoch: 20/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.48
Epoch: 20/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.48
Epoch: 20/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.51
Epoch: 21/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.50
Epoch: 21/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.50
Epoch: 21/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.53
Epoch: 22/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.53
Epoch: 22/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.53
Epoch: 22/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.59
Epoch: 23/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.55
Epoch: 23/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.55
Epoch: 23/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.57
Epoch: 24/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.56
```

```
Epoch: 24/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.56
Epoch: 24/50. Test set: Average loss: 0.0019        Average Accuracy:        Accuracy: 94.63
Epoch: 25/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.59
Epoch: 25/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.59
Epoch: 25/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.6
Epoch: 26/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.62
Epoch: 26/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.62
Epoch: 26/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.62
Epoch: 27/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.63
Epoch: 27/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.63
Epoch: 27/50. Test set: Average loss: 0.0020        Average Accuracy:        Accuracy: 94.66
Epoch: 28/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.61
Epoch: 28/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.61
Epoch: 28/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.66
Epoch: 29/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.64
Epoch: 29/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.64
Epoch: 29/50. Test set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.63
Epoch: 30/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.65
Epoch: 30/50. Train set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.65
Epoch: 30/50. Test set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.65
Epoch: 31/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.66
Epoch: 31/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.66
Epoch: 31/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.66
Epoch: 32/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.68
Epoch: 32/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.68
Epoch: 32/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.67
Epoch: 33/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.69
Epoch: 33/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.69
Epoch: 33/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.68
Epoch: 34/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.69
Epoch: 34/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.69
Epoch: 34/50. Test set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.7
Epoch: 35/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.69
Epoch: 35/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.69
Epoch: 35/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.69
Epoch: 36/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.70
Epoch: 36/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.70
Epoch: 36/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.66
Epoch: 37/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.71
Epoch: 37/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.71
Epoch: 37/50. Test set: Average loss: 0.0018        Average Accuracy:        Accuracy: 94.7
Epoch: 38/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.75
Epoch: 38/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.75
Epoch: 38/50. Test set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.69
Epoch: 39/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.71
Epoch: 39/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.71
Epoch: 39/50. Test set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.66
Epoch: 40/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.74
```

```
Epoch: 40/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.74
Epoch: 40/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.7
Epoch: 41/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.73
Epoch: 41/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.73
Epoch: 41/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.7
Epoch: 42/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.73
Epoch: 42/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.73
Epoch: 42/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.71
Epoch: 43/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.75
Epoch: 43/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.75
Epoch: 43/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.69
Epoch: 44/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.72
Epoch: 44/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.72
Epoch: 44/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.71
Epoch: 45/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.74
Epoch: 45/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.74
Epoch: 45/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.71
Epoch: 46/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.75
Epoch: 46/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.75
Epoch: 46/50. Test set: Average loss: 0.0016        Average Accuracy:        Accuracy: 94.73
Epoch: 47/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.74
Epoch: 47/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.74
Epoch: 47/50. Test set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.73
Epoch: 48/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.75
Epoch: 48/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.75
Epoch: 48/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.72
Epoch: 49/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.74
Epoch: 49/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.74
Epoch: 49/50. Test set: Average loss: 0.0017        Average Accuracy:        Accuracy: 94.7
Epoch: 50/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.76
Epoch: 50/50. Train set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.76
Epoch: 50/50. Test set: Average loss: 0.0015        Average Accuracy:        Accuracy: 94.72
```

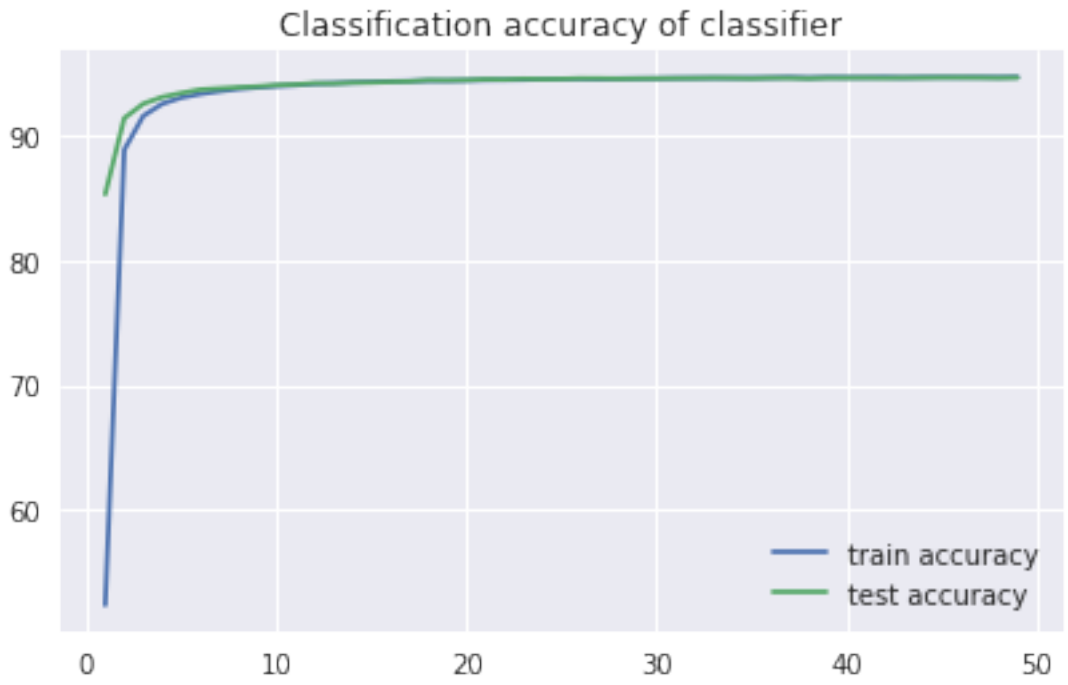
### 0.13.1 Save Classifier

```
In [0]: path =save_dir+"/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)
```

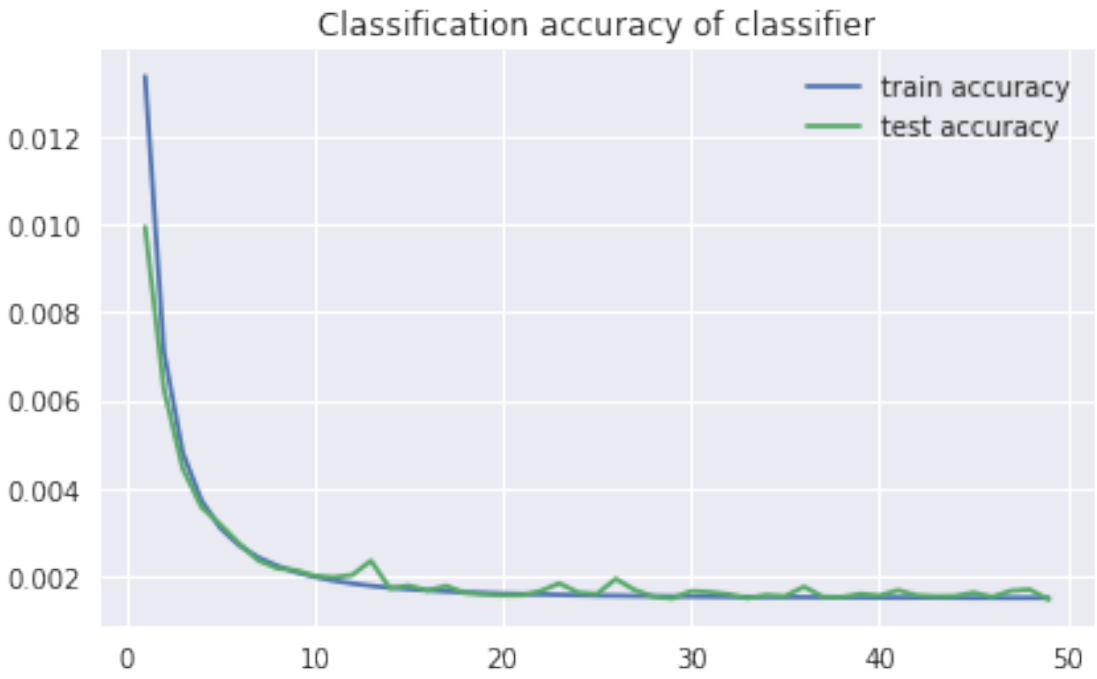
## 0.14 Plots

```
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 0x7fb7497064a8>

/home/chivukula_manju/yes/lib/python3.6/site-packages/matplotlib/font_manager.py:1328: UserWarn
  (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 loss","test loss"])

Out[0]: <matplotlib.legend.Legend at 0x7fb748f34898>

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




In [0]:

In [0]:

### 0.15 Model 3

```
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()
  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)
```

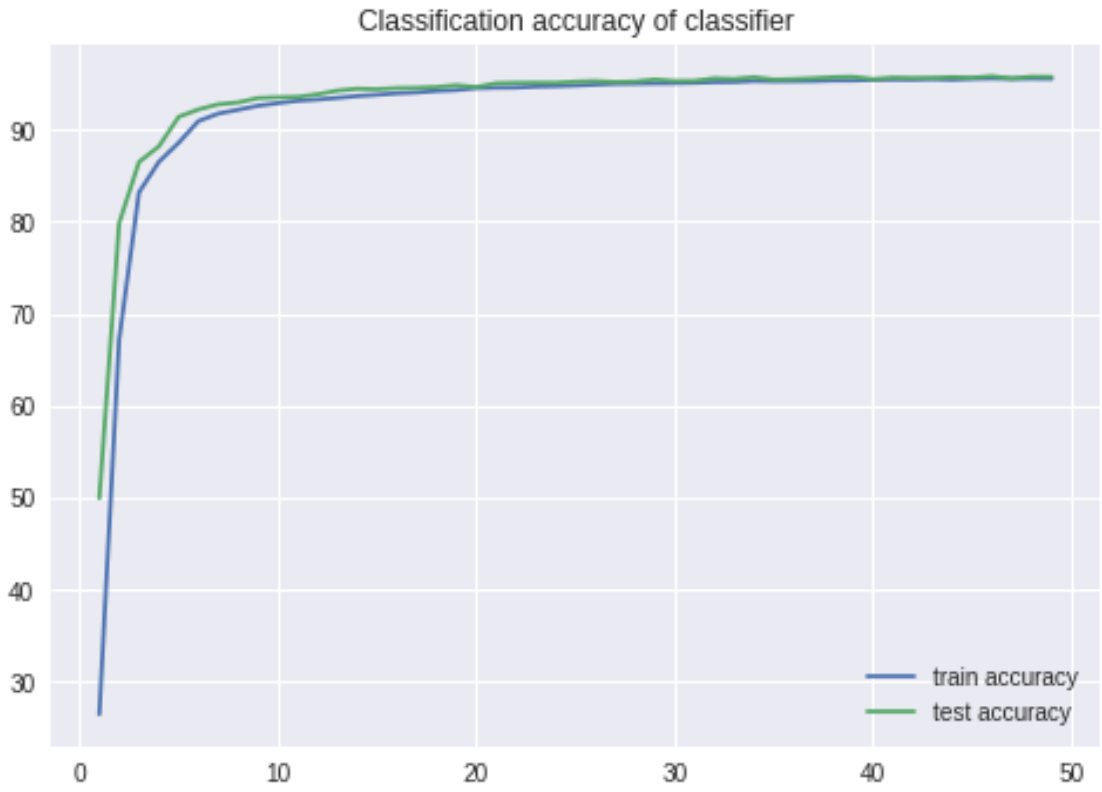
```
Epoch: 2/50. Train set: Average loss: -7492.7246         Average Accuracy:         Accuracy: 2
Epoch: 2/50. Train set: Average loss: -7492.7246         Average Accuracy:         Accuracy: 2
Epoch: 2/50. Test set: Average loss: 0.0000         Average Accuracy:         Accuracy: 52.34
Epoch: 3/50. Train set: Average loss: -8900.7314         Average Accuracy:         Accuracy: 6
Epoch: 3/50. Train set: Average loss: -8900.7314         Average Accuracy:         Accuracy: 6
```

```
Epoch: 3/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 75.07
Epoch: 4/50. Train set: Average loss: -9718.1816       Average Accuracy:        Accuracy: 80
Epoch: 4/50. Train set: Average loss: -9718.1816       Average Accuracy:        Accuracy: 80
Epoch: 4/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 84.7
Epoch: 5/50. Train set: Average loss: -10135.1260      Average Accuracy:        Accuracy: 8
Epoch: 5/50. Train set: Average loss: -10135.1260      Average Accuracy:        Accuracy: 8
Epoch: 5/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 88.17
Epoch: 6/50. Train set: Average loss: -10354.4287      Average Accuracy:        Accuracy: 8
Epoch: 6/50. Train set: Average loss: -10354.4287      Average Accuracy:        Accuracy: 8
Epoch: 6/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 89.86
Epoch: 7/50. Train set: Average loss: -10497.6748      Average Accuracy:        Accuracy: 8
Epoch: 7/50. Train set: Average loss: -10497.6748      Average Accuracy:        Accuracy: 8
Epoch: 7/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 91.05
Epoch: 8/50. Train set: Average loss: -10583.9287      Average Accuracy:        Accuracy: 9
Epoch: 8/50. Train set: Average loss: -10583.9287      Average Accuracy:        Accuracy: 9
Epoch: 8/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 90.33
Epoch: 9/50. Train set: Average loss: -10666.1738      Average Accuracy:        Accuracy: 9
Epoch: 9/50. Train set: Average loss: -10666.1738      Average Accuracy:        Accuracy: 9
Epoch: 9/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 92.25
Epoch: 10/50. Train set: Average loss: -10721.1650      Average Accuracy:        Accuracy:
Epoch: 10/50. Train set: Average loss: -10721.1650      Average Accuracy:        Accuracy:
Epoch: 10/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 92.61
Epoch: 11/50. Train set: Average loss: -10751.0410      Average Accuracy:        Accuracy:
Epoch: 11/50. Train set: Average loss: -10751.0410      Average Accuracy:        Accuracy:
Epoch: 11/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 92.88
Epoch: 12/50. Train set: Average loss: -10840.1602      Average Accuracy:        Accuracy:
Epoch: 12/50. Train set: Average loss: -10840.1602      Average Accuracy:        Accuracy:
Epoch: 12/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 93.58
Epoch: 13/50. Train set: Average loss: -10870.8945      Average Accuracy:        Accuracy:
Epoch: 13/50. Train set: Average loss: -10870.8945      Average Accuracy:        Accuracy:
Epoch: 13/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 93.55
Epoch: 14/50. Train set: Average loss: -10885.9775      Average Accuracy:        Accuracy:
Epoch: 14/50. Train set: Average loss: -10885.9775      Average Accuracy:        Accuracy:
Epoch: 14/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 93.73
Epoch: 15/50. Train set: Average loss: -10899.6182      Average Accuracy:        Accuracy:
Epoch: 15/50. Train set: Average loss: -10899.6182      Average Accuracy:        Accuracy:
Epoch: 15/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 93.86
Epoch: 16/50. Train set: Average loss: -10924.0127      Average Accuracy:        Accuracy:
Epoch: 16/50. Train set: Average loss: -10924.0127      Average Accuracy:        Accuracy:
Epoch: 16/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 93.96
Epoch: 17/50. Train set: Average loss: -10940.0918      Average Accuracy:        Accuracy:
Epoch: 17/50. Train set: Average loss: -10940.0918      Average Accuracy:        Accuracy:
Epoch: 17/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 94.11
Epoch: 18/50. Train set: Average loss: -10950.1260      Average Accuracy:        Accuracy:
Epoch: 18/50. Train set: Average loss: -10950.1260      Average Accuracy:        Accuracy:
Epoch: 18/50. Test set: Average loss: 0.0000       Average Accuracy:        Accuracy: 94.2
Epoch: 19/50. Train set: Average loss: -10966.4785      Average Accuracy:        Accuracy:
Epoch: 19/50. Train set: Average loss: -10966.4785      Average Accuracy:        Accuracy:
```

```
Epoch: 19/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.35
Epoch: 20/50. Train set: Average loss: -10970.7021       Average Accuracy:        Accuracy:
Epoch: 20/50. Train set: Average loss: -10970.7021       Average Accuracy:        Accuracy:
Epoch: 20/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.5
Epoch: 21/50. Train set: Average loss: -10981.4531       Average Accuracy:        Accuracy:
Epoch: 21/50. Train set: Average loss: -10981.4531       Average Accuracy:        Accuracy:
Epoch: 21/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.53
Epoch: 22/50. Train set: Average loss: -10987.8828       Average Accuracy:        Accuracy:
Epoch: 22/50. Train set: Average loss: -10987.8828       Average Accuracy:        Accuracy:
Epoch: 22/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.47
Epoch: 23/50. Train set: Average loss: -10999.9141       Average Accuracy:        Accuracy:
Epoch: 23/50. Train set: Average loss: -10999.9141       Average Accuracy:        Accuracy:
Epoch: 23/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.59
Epoch: 24/50. Train set: Average loss: -11008.8711       Average Accuracy:        Accuracy:
Epoch: 24/50. Train set: Average loss: -11008.8711       Average Accuracy:        Accuracy:
Epoch: 24/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.43
Epoch: 25/50. Train set: Average loss: -11015.1982       Average Accuracy:        Accuracy:
Epoch: 25/50. Train set: Average loss: -11015.1982       Average Accuracy:        Accuracy:
Epoch: 25/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.71
Epoch: 26/50. Train set: Average loss: -11022.5439       Average Accuracy:        Accuracy:
Epoch: 26/50. Train set: Average loss: -11022.5439       Average Accuracy:        Accuracy:
Epoch: 26/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.64
Epoch: 27/50. Train set: Average loss: -11034.2051       Average Accuracy:        Accuracy:
Epoch: 27/50. Train set: Average loss: -11034.2051       Average Accuracy:        Accuracy:
Epoch: 27/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.63
Epoch: 28/50. Train set: Average loss: -11042.3604       Average Accuracy:        Accuracy:
Epoch: 28/50. Train set: Average loss: -11042.3604       Average Accuracy:        Accuracy:
Epoch: 28/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.71
Epoch: 29/50. Train set: Average loss: -11049.2451       Average Accuracy:        Accuracy:
Epoch: 29/50. Train set: Average loss: -11049.2451       Average Accuracy:        Accuracy:
Epoch: 29/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.9
Epoch: 30/50. Train set: Average loss: -11056.2949       Average Accuracy:        Accuracy:
Epoch: 30/50. Train set: Average loss: -11056.2949       Average Accuracy:        Accuracy:
Epoch: 30/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.59
Epoch: 31/50. Train set: Average loss: -11061.5225       Average Accuracy:        Accuracy:
Epoch: 31/50. Train set: Average loss: -11061.5225       Average Accuracy:        Accuracy:
Epoch: 31/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.89
Epoch: 32/50. Train set: Average loss: -11066.0996       Average Accuracy:        Accuracy:
Epoch: 32/50. Train set: Average loss: -11066.0996       Average Accuracy:        Accuracy:
Epoch: 32/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.7
Epoch: 33/50. Train set: Average loss: -11070.9443       Average Accuracy:        Accuracy:
Epoch: 33/50. Train set: Average loss: -11070.9443       Average Accuracy:        Accuracy:
Epoch: 33/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.95
Epoch: 34/50. Train set: Average loss: -11070.9668       Average Accuracy:        Accuracy:
Epoch: 34/50. Train set: Average loss: -11070.9668       Average Accuracy:        Accuracy:
Epoch: 34/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.75
Epoch: 35/50. Train set: Average loss: -11076.9082       Average Accuracy:        Accuracy:
Epoch: 35/50. Train set: Average loss: -11076.9082       Average Accuracy:        Accuracy:
```

```
Epoch: 35/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.09
Epoch: 36/50. Train set: Average loss: -11081.7422      Average Accuracy:        Accuracy:
Epoch: 36/50. Train set: Average loss: -11081.7422      Average Accuracy:        Accuracy:
Epoch: 36/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.93
Epoch: 37/50. Train set: Average loss: -11083.2842      Average Accuracy:        Accuracy:
Epoch: 37/50. Train set: Average loss: -11083.2842      Average Accuracy:        Accuracy:
Epoch: 37/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.96
Epoch: 38/50. Train set: Average loss: -11087.9932      Average Accuracy:        Accuracy:
Epoch: 38/50. Train set: Average loss: -11087.9932      Average Accuracy:        Accuracy:
Epoch: 38/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 94.99
Epoch: 39/50. Train set: Average loss: -11091.5830      Average Accuracy:        Accuracy:
Epoch: 39/50. Train set: Average loss: -11091.5830      Average Accuracy:        Accuracy:
Epoch: 39/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.19
Epoch: 40/50. Train set: Average loss: -11091.2021      Average Accuracy:        Accuracy:
Epoch: 40/50. Train set: Average loss: -11091.2021      Average Accuracy:        Accuracy:
Epoch: 40/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.23
Epoch: 41/50. Train set: Average loss: -11095.1436      Average Accuracy:        Accuracy:
Epoch: 41/50. Train set: Average loss: -11095.1436      Average Accuracy:        Accuracy:
Epoch: 41/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.02
Epoch: 42/50. Train set: Average loss: -11097.5859      Average Accuracy:        Accuracy:
Epoch: 42/50. Train set: Average loss: -11097.5859      Average Accuracy:        Accuracy:
Epoch: 42/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.05
Epoch: 43/50. Train set: Average loss: -11100.3613      Average Accuracy:        Accuracy:
Epoch: 43/50. Train set: Average loss: -11100.3613      Average Accuracy:        Accuracy:
Epoch: 43/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.02
Epoch: 44/50. Train set: Average loss: -11101.3418      Average Accuracy:        Accuracy:
Epoch: 44/50. Train set: Average loss: -11101.3418      Average Accuracy:        Accuracy:
Epoch: 44/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.06
Epoch: 45/50. Train set: Average loss: -11102.9395      Average Accuracy:        Accuracy:
Epoch: 45/50. Train set: Average loss: -11102.9395      Average Accuracy:        Accuracy:
Epoch: 45/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.09
Epoch: 46/50. Train set: Average loss: -11106.3555      Average Accuracy:        Accuracy:
Epoch: 46/50. Train set: Average loss: -11106.3555      Average Accuracy:        Accuracy:
Epoch: 46/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.22
Epoch: 47/50. Train set: Average loss: -11109.7627      Average Accuracy:        Accuracy:
Epoch: 47/50. Train set: Average loss: -11109.7627      Average Accuracy:        Accuracy:
Epoch: 47/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.18
Epoch: 48/50. Train set: Average loss: -11109.0449      Average Accuracy:        Accuracy:
Epoch: 48/50. Train set: Average loss: -11109.0449      Average Accuracy:        Accuracy:
Epoch: 48/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.2
Epoch: 49/50. Train set: Average loss: -11112.1162      Average Accuracy:        Accuracy:
Epoch: 49/50. Train set: Average loss: -11112.1162      Average Accuracy:        Accuracy:
Epoch: 49/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.32
Epoch: 50/50. Train set: Average loss: -11115.4043      Average Accuracy:        Accuracy:
Epoch: 50/50. Train set: Average loss: -11115.4043      Average Accuracy:        Accuracy:
Epoch: 50/50. Test set: Average loss: 0.0000        Average Accuracy:        Accuracy: 95.06
```

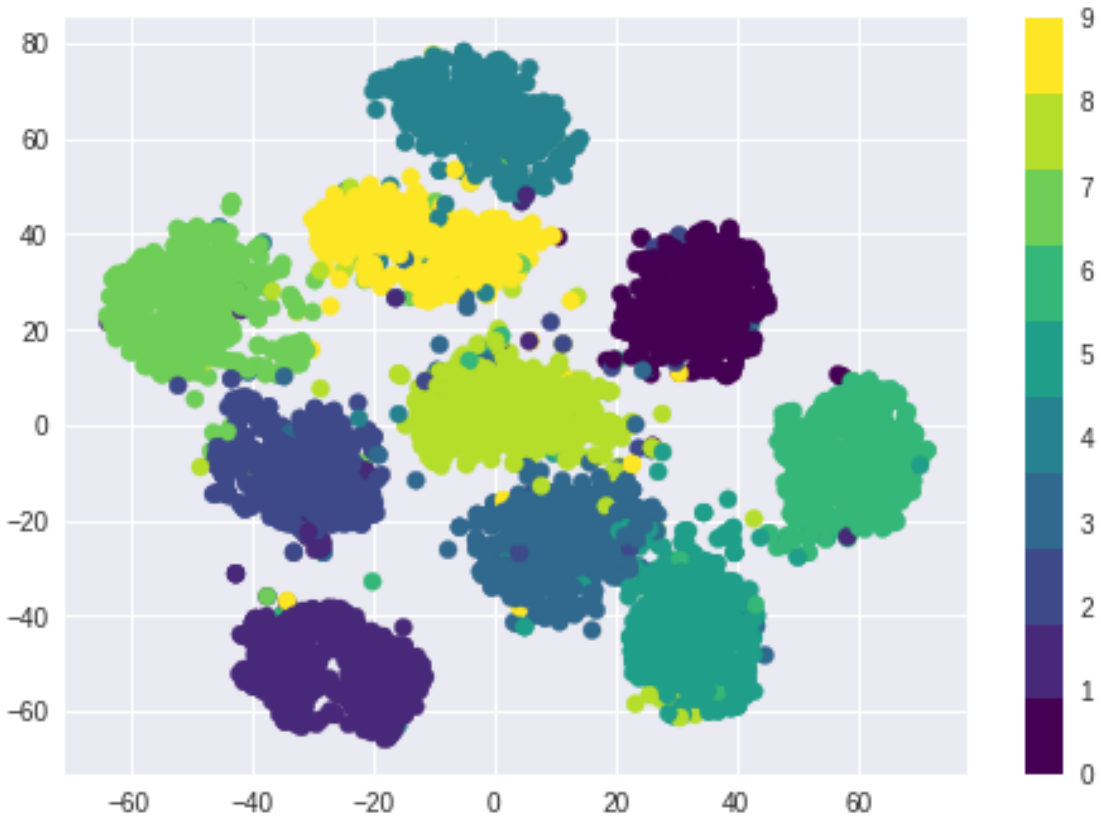
```
In [0]: plt.style.use("seaborn")
        plt.plot(range(1,n_epochs),train_accuracy)
        plt.plot(range(1,n_epochs),test_accuracy)
        plt.title("Classification accuracy of classifier")
        plt.legend(["train accuracy","test accuracy"])

        print(max(train_accuracy),max(test_accuracy))

Out[0]: <matplotlib.legend.Legend at 0x7f996178d4a8>
```




```
In [0]: path = "tsne_bvae+classifier+vae+train.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)
```

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




In [0]:

In [0]: *## latent traversals*

```
        testpoint1, testpointlabel1 = train_loader.dataset[0]
        testpoint2, testpoint2label2 = train_loader.dataset[1]
        testpoint3, testpoint3label3 = train_loader.dataset[2]
        filename = 'beta-results/vae_traversal_testpoint_5.png'
```

```
        traverse_latents(model, testpoint1, Params.nb_latents, filename)

        filename = 'beta-results/vae_traversal_testpoint_0.png'
        traverse_latents(model, testpoint2, Params.nb_latents, filename)

        filename = 'beta-results/vae_traversal_testpoint_4.png'
        traverse_latents(model, testpoint3, Params.nb_latents, filename)
```

In [0]: