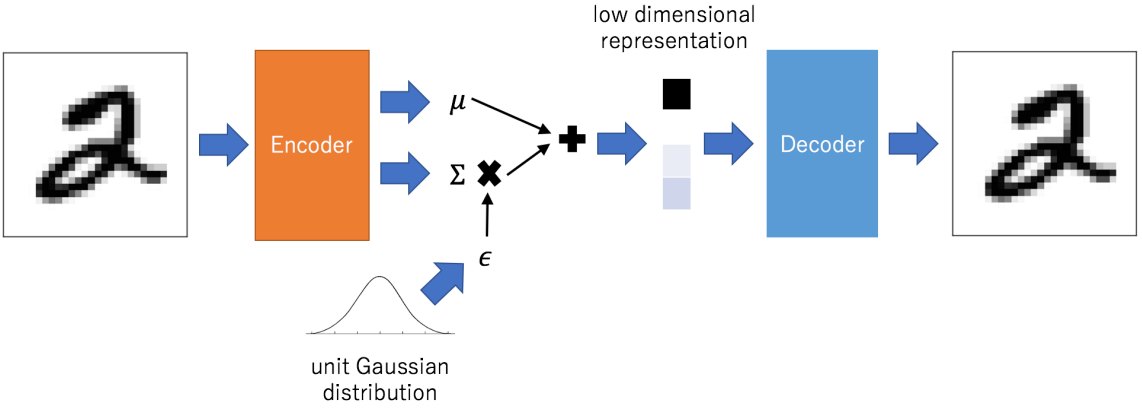
Problem 1 - Variational Auto-Encoder (VAE)

Open in Colab

Variational Auto-Encoders (VAEs) are a widely used class of generative models. They are simple to implement and, in contrast to other generative model classes like Generative Adversarial Networks, they optimize an explicit maximum likelihood objective to train the model. Finally, their architecture makes them well-suited for unsupervised representation learning, i.e. learning low-dimensional representations of high-dimenionsal inputs, like images, with only self-supervised objectives (data reconstruction in the case of VAEs).



(image source: https://mlexplained.com/2017/12/28/an-intuitive-explanation-of-variational-autoencoders-vaes-part-1)

By working on this problem you will learn and practice the following steps:

1. Set up a data loading pipeline in PyTorch.
2. Implement, train and visualize an auto-encoder architecture.
3. Extend your implementation to a variational auto-encoder.
4. Learn how to tune the critical beta parameter of your VAE.
5. Inspect the learned representation of your VAE.

Note: For faster training of the models in this assignment you can use Colab with enabled GPU support. In Colab, navigate to "Runtime" --> "Change Runtime Type" and set the "Hardware Accelerator" to "GPU".

1. MNIST Dataset

We will perform all experiments for this problem using the MNIST dataset, a standard dataset of handwritten digits. The main benefits of this dataset are that it is small and relatively easy to model. It therefore allows for quick experimentation and serves as initial test bed in many papers.

Another benefit is that it is so widely used that PyTorch even provides functionality to

automatically download it.

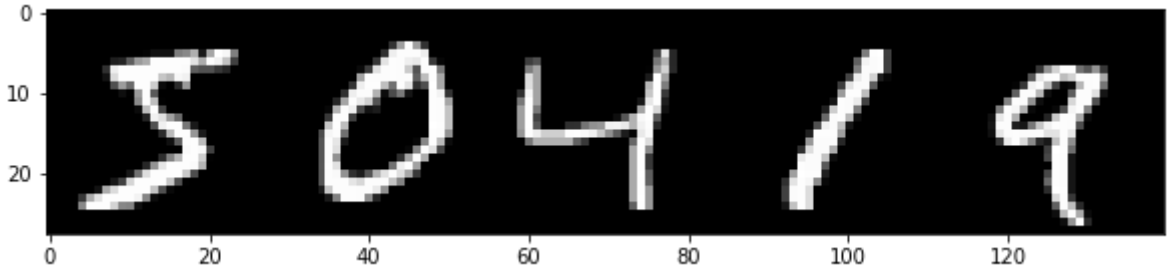
Let's start by downloading the data and visualizing some samples.

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

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-i
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
import torch
import torchvision

# this will automatically download the MNIST training set
mnist_train = torchvision.datasets.MNIST(root='./data',
                                         train=True,
                                         download=True,
                                         transform=torchvision.transforms.T
print("\n Download complete! Downloaded {} training examples!".format(len(m
```

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

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

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

Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

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

Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

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

Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw


 Download complete! Downloaded 60000 training examples!
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# Let's display some of the training samples.
sample_images = []
mnist_it = iter(mnist_train)  # create simple iterator, later we will use p
for _ in range(5):
  sample = next(mnist_it)     # samples a tuple (image, label)
  sample_images.append(sample[0][0].data.cpu().numpy())

fig = plt.figure(figsize = (10, 50))
ax1 = plt.subplot(111)
ax1.imshow(np.concatenate(sample_images, axis=1), cmap='gray')
plt.show()
```



2. Auto-Encoder

Before implementing the full VAE, we will first implement an **auto-encoder architecture**. Auto-encoders feature the same encoder-decoder architecture as VAEs and therefore also learn a low-dimensional representation of the input data without supervision. In contrast to VAEs they are **fully deterministic** models and do not employ variational inference for optimization.

The **architecture** is very simple: we will encode the input image into a low-dimensional representation using a convolutional network with strided convolutions that reduce the image resolution in every layer. This results in a low-dimensional representation of the input image. This representation will get decoded back into the dimensionality of the input image using a convolutional decoder network that mirrors the architecture of the encoder. It employs transposed convolutions to increase the resolution of its input in every layer. The whole model is trained by **minimizing a reconstruction loss** between the input and the decoded image.

Intuitively, the **auto-encoder needs to compress the information contained in the input image** into a much lower dimensional representation (e.g. 28x28=784px vs. 64 embedding dimensions for our MNIST model). This is possible since the information captured in the pixels is *highly redundant*. E.g. encoding an MNIST image requires <4 bits to encode which of the 10 possible digits is displayed and a few additional bits to capture information about shape and orientation. This is much less than the $255^{28\cdot 28}$ bits of information that could be theoretically captured in the input image.

Learning such a **compressed representation can make downstream task learning easier**. For example, learning to add two numbers based on the inferred digits is much easier than performing the task based on two piles of pixel values that depict the digits.

In the following, we will first define the architecture of encoder and decoder and then train the auto-encoder model.

Defining the Auto-Encoder Architecture [6pt]

In [ ]:

```
import torch.nn as nn

# Let's define encoder and decoder networks
###########################################################################
# Encoder Architecture:                                                   #
#   - Conv2d, hidden units: 32, output resolution: 14x14, kernel: 4       #
#   - LeakyReLU                                                           #
#   - Conv2d, hidden units: 64, output resolution: 7x7, kernel: 4         #
#   - BatchNorm2d                                                         #
#   - LeakyReLU                                                           #
#   - Conv2d, hidden units: 128, output resolution: 3x3, kernel: 3        #
#   - BatchNorm2d                                                         #
#   - LeakyReLU                                                           #
#   - Conv2d, hidden units: 256, output resolution: 1x1, kernel: 3        #
#   - BatchNorm2d                                                         #
#   - LeakyReLU                                                           #
#   - Flatten                                                             #
#   - Linear, output units: nz (= representation dimensionality)          #
###########################################################################

class Encoder(nn.Module):
  def __init__(self, nz):
    super().__init__()
    ################################## TODO ##################################
    # Create the network architecture using a nn.Sequential module wrapper.
    # All convolutional layers should also learn a bias.
    # HINT: use the given information to compute stride and padding
    #       for each convolutional layer. Verify the shapes of intermediate
    #       by running partial networks (with the next cell) and visualizin
    #       output shapes.
    ##########################################################################
    self.net = nn.Sequential(
        # add your network layers here
        # ...
        nn.Conv2d(in_channels=1, out_channels=32, kernel_size=4, stride=2,
        nn.LeakyReLU(negative_slope=1e-2),
        nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2,
        nn.BatchNorm2d(64),
        nn.LeakyReLU(negative_slope=1e-2),
        nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=2
        nn.BatchNorm2d(128),
        nn.LeakyReLU(negative_slope=1e-2),
        nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=
        nn.BatchNorm2d(256),
        nn.LeakyReLU(negative_slope=1e-2),
        nn.Flatten(),
        nn.Linear(in_features=256, out_features=nz, bias=True)
    )
    ################################ END TODO ################################

  def forward(self, x):
    return self.net(x)


###########################################################################
# Decoder Architecture (mirrors encoder architecture):                    #
#   - Linear, output units: 256                                           #
#   - Reshape, output shape: (256, 1, 1)                                  #
#   - BatchNorm2d                                                         #
#   - LeakyReLU                                                           #
#   - ConvT2d, hidden units: 128, output resolution: 3x3, kernel: 3       #
#   - BatchNorm2d                                                         #
#   - LeakyReLU                                                           #
```

```
#   - LeakyReLU                                                          #
#   - ConvT2d, hidden units: 64, output resolution: 7x7, kernel: 3       #
#   - ...                                                                #
#   - ...                                                                #
#   - ConvT2d, output units: 1, output resolution: 28x28, kernel: 4      #
#   - Sigmoid (to limit output in range [0...1])                         #
########################################################################

class Decoder(nn.Module):
  def __init__(self, nz):
    super().__init__()
    ################################## TODO ##################################
    # Create the network architecture using a nn.Sequential module wrapper.
    # Again, all (transposed) convolutional layers should also learn a bias
    # We need to separate the intial linear layer into a separate variable
    # nn.Sequential does not support reshaping. Instead the "Reshape" is pe
    # in the forward() function below and does not need to be added to self
    # HINT: use the class nn.ConvTranspose2d for the transposed convolution
    #       Verify the shapes of intermediate layers by running partial net
    #       (using the next cell) and visualizing the output shapes.
    ##########################################################################
    self.map = nn.Linear(in_features=nz, out_features=256, bias=True)   # f
    self.net = nn.Sequential(
        # add your network layers here
        # ...
        nn.BatchNorm2d(256),
        nn.LeakyReLU(),
        nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=3
        nn.BatchNorm2d(128),
        nn.LeakyReLU(),
        nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=3,
        nn.BatchNorm2d(64),
        nn.LeakyReLU(),
        nn.ConvTranspose2d(in_channels=64, out_channels=32, kernel_size=4,
        nn.LeakyReLU(),
        nn.ConvTranspose2d(in_channels=32, out_channels=1, kernel_size=4, s
        nn.Sigmoid()
    )
    ################################ END TODO ################################

  def forward(self, x):
    return self.net(self.map(x).reshape(-1, 256, 1, 1))
```

Testing the Auto-Encoder Forward Pass [1pt]

In [ ]:
```
# To test your encoder/decoder, let's encode/decode some sample images
# first, make a PyTorch DataLoader object to sample data batches
batch_size = 64
nworkers = 4        # number of wrokers used for efficient data loading

######################################## TODO ##############################
# Create a PyTorch DataLoader object for efficiently generating training ba
# Make sure that the data loader automatically shuffles the training datase
# HINT: The DataLoader wraps the MNIST dataset class we created earlier.
#       Use the given batch_size and number of data loading workers when cr
#       the DataLoader.
############################################################################
mnist_data_loader = torch.utils.data.DataLoader(mnist_train, batch_size=bat
#################################### END TODO ##############################

# now we can run a forward pass for encoder and decoder and check the produ
nz = 64          # dimensionality of the learned embedding
encoder = Encoder(nz)
decoder = Decoder(nz)
for sample_img, sample_label in mnist_data_loader:
  enc = encoder(sample_img)
  print("Shape of encoding vector (should be [batch_size, nz]): {}".format(
  dec = decoder(enc)
  print("Shape of decoded image (should be [batch_size, 1, 28, 28]): {}".fo
  break
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481:
UserWarning: This DataLoader will create 4 worker processes in total. Our s
uggested max number of worker in current system is 2, which is smaller than
what this DataLoader is going to create. Please be aware that excessive wor
ker creation might get DataLoader running slow or even freeze, lower the wo
rker number to avoid potential slowness/freeze if necessary.
  cpuset_checked))
Shape of encoding vector (should be [batch_size, nz]): torch.Size([64, 64])
Shape of decoded image (should be [batch_size, 1, 28, 28]): torch.Size([64,
1, 28, 28])
```

Now that we defined encoder and decoder network our architecture is nearly complete. However, before we start training, we can wrap encoder and decoder into an auto-encoder class for easier handling.

In [ ]:
```
class AutoEncoder(nn.Module):
  def __init__(self, nz):
    super().__init__()
    self.encoder = Encoder(nz)
    self.decoder = Decoder(nz)

  def forward(self, x):
    return self.decoder(self.encoder(x))

  def reconstruct(self, x):
    """Only used later for visualization."""
    return self.forward(x)
```

Setting up the Auto-Encoder Training Loop [6pt]

After implementing the network architecture, we can now set up the training loop and run training.

In [ ]:

```
epochs = 10
learning_rate = 1e-3

# build AE model
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')   #
ae_model = AutoEncoder(nz).to(device)    # transfer model to GPU if availab
ae_model = ae_model.train()   # set model in train mode (eg batchnorm param

# build optimizer and loss function
######################################## TODO ##############################
# Create the optimizer and loss classes. For the loss you can use a loss la
# from the torch.nn package.
# HINT: We will use the Adam optimizer (learning rate given above, otherwis
#       default parameters) and MSE loss for the criterion / loss.
# NOTE: We could also use alternative loss functions like cross entropy, de
#       on the assumptions we are making about the output distribution. Her
#       will use MSE loss as it is the most common choice, assuming a Gauss
#       output distribution.
############################################################################
opt = torch.optim.Adam(params=ae_model.parameters(), lr=learning_rate)
criterion = nn.MSELoss()    # create loss layer instance
######################################## END TODO ##########################

train_it = 0
for ep in range(epochs):
  print("Run Epoch {}".format(ep))
  ######################################## TODO ############################
  # Implement the main training loop for the auto-encoder model.
  # HINT: Your training loop should sample batches from the data loader, ru
  #       forward pass of the AE, compute the loss, perform the backward pa
  #       perform one gradient step with the optimizer.
  # HINT: Don't forget to erase old gradients before performing the backwar
  ############################################################################
  for sample_img, sample_label in mnist_data_loader:
    # add training loop commands here
    # ...
    # Move data to GPU
    sample_img = sample_img.to(device)

    opt.zero_grad()
    target = ae_model.forward(sample_img)
    rec_loss = criterion(sample_img, target)
    rec_loss.backward()
    opt.step()
    ######################################## END TODO ########################

    if train_it % 100 == 0:
      print("It {}: Reconstruction Loss: {}".format(train_it, rec_loss))
    train_it += 1

print("Done!")
```

```
Run Epoch 0
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481:
UserWarning: This DataLoader will create 4 worker processes in total. Our s
uggested max number of worker in current system is 2, which is smaller than
what this DataLoader is going to create. Please be aware that excessive wor
ker creation might get DataLoader running slow or even freeze, lower the wo
rker number to avoid potential slowness/freeze if necessary.
  cpuset_checked))
It 0: Reconstruction Loss: 0.18664182722568512
It 100: Reconstruction Loss: 0.0263246800750494
```

```
It 200: Reconstruction Loss: 0.016419680789113045
It 300: Reconstruction Loss: 0.013837716542184353
It 400: Reconstruction Loss: 0.011019397526979446
It 500: Reconstruction Loss: 0.011521183885633945
It 600: Reconstruction Loss: 0.009608311578631401
It 700: Reconstruction Loss: 0.009960976429283619
It 800: Reconstruction Loss: 0.009381072595715523
It 900: Reconstruction Loss: 0.008728671818971634
Run Epoch 1
It 1000: Reconstruction Loss: 0.008005676791071892
It 1100: Reconstruction Loss: 0.007739451713860035
It 1200: Reconstruction Loss: 0.007920853793621063
It 1300: Reconstruction Loss: 0.008873661048710346
It 1400: Reconstruction Loss: 0.006991246249526739
It 1500: Reconstruction Loss: 0.007462984416633844
It 1600: Reconstruction Loss: 0.0065509844571352005
It 1700: Reconstruction Loss: 0.007878832519054413
It 1800: Reconstruction Loss: 0.007533562369644642
Run Epoch 2
It 1900: Reconstruction Loss: 0.006996598560363054
It 2000: Reconstruction Loss: 0.006903706584125757
It 2100: Reconstruction Loss: 0.005974492523819208
It 2200: Reconstruction Loss: 0.006070496514439583
It 2300: Reconstruction Loss: 0.0068871863186359406
It 2400: Reconstruction Loss: 0.007140104193240404
It 2500: Reconstruction Loss: 0.006445891223847866
It 2600: Reconstruction Loss: 0.005407628137618303
It 2700: Reconstruction Loss: 0.005920184776186943
It 2800: Reconstruction Loss: 0.0068471599370241165
Run Epoch 3
It 2900: Reconstruction Loss: 0.005907618906348944
It 3000: Reconstruction Loss: 0.007192608900368214
It 3100: Reconstruction Loss: 0.00592842185869813
It 3200: Reconstruction Loss: 0.006346351932734251
It 3300: Reconstruction Loss: 0.00574900396168232
It 3400: Reconstruction Loss: 0.005313708446919918
It 3500: Reconstruction Loss: 0.005397276021540165
It 3600: Reconstruction Loss: 0.005362116266041994
It 3700: Reconstruction Loss: 0.005842178128659725
Run Epoch 4
It 3800: Reconstruction Loss: 0.005256583448499441
It 3900: Reconstruction Loss: 0.005855184514075518
It 4000: Reconstruction Loss: 0.0055256495252251625
It 4100: Reconstruction Loss: 0.005327051505446434
It 4200: Reconstruction Loss: 0.005004513077437878
It 4300: Reconstruction Loss: 0.0063035134226083755
It 4400: Reconstruction Loss: 0.005292521324008703
It 4500: Reconstruction Loss: 0.005102543160319328
It 4600: Reconstruction Loss: 0.0051817456260323524
Run Epoch 5
It 4700: Reconstruction Loss: 0.004771128762513399
It 4800: Reconstruction Loss: 0.005044563673436642
It 4900: Reconstruction Loss: 0.00499194348230958
It 5000: Reconstruction Loss: 0.004607779439538717
It 5100: Reconstruction Loss: 0.005668171215802431
It 5200: Reconstruction Loss: 0.004626581445336342
It 5300: Reconstruction Loss: 0.004911825060844421
It 5400: Reconstruction Loss: 0.005654565989971161
It 5500: Reconstruction Loss: 0.004666066262871027
It 5600: Reconstruction Loss: 0.004035145044326782
Run Epoch 6
It 5700: Reconstruction Loss: 0.004997056908905506
It 5800: Reconstruction Loss: 0.00633472204208374
```

```
It 5900: Reconstruction Loss: 0.004174941219389439
It 6000: Reconstruction Loss: 0.004814298823475838
It 6100: Reconstruction Loss: 0.005388380028307438
It 6200: Reconstruction Loss: 0.004971726331859827
It 6300: Reconstruction Loss: 0.004994853399693966
It 6400: Reconstruction Loss: 0.004561730194836855
It 6500: Reconstruction Loss: 0.0046912203542888165
Run Epoch 7
It 6600: Reconstruction Loss: 0.004468838218599558
It 6700: Reconstruction Loss: 0.004582060966640711
It 6800: Reconstruction Loss: 0.004014728590846062
It 6900: Reconstruction Loss: 0.00393039220944047
It 7000: Reconstruction Loss: 0.00466529093682766
It 7100: Reconstruction Loss: 0.00452636880800128
It 7200: Reconstruction Loss: 0.004711611662060022
It 7300: Reconstruction Loss: 0.0037244956474751234
It 7400: Reconstruction Loss: 0.0045255329459905624
It 7500: Reconstruction Loss: 0.00461358530446887
Run Epoch 8
It 7600: Reconstruction Loss: 0.004474017769098282
It 7700: Reconstruction Loss: 0.0038130246102809906
It 7800: Reconstruction Loss: 0.0045503731817007065
It 7900: Reconstruction Loss: 0.004730247426778078
It 8000: Reconstruction Loss: 0.004492415580898523
It 8100: Reconstruction Loss: 0.004131972324103117
It 8200: Reconstruction Loss: 0.0039808182045817375
It 8300: Reconstruction Loss: 0.004443623591214418
It 8400: Reconstruction Loss: 0.004379660822451115
Run Epoch 9
It 8500: Reconstruction Loss: 0.004476477392017841
It 8600: Reconstruction Loss: 0.004415799863636494
It 8700: Reconstruction Loss: 0.003854422364383936
It 8800: Reconstruction Loss: 0.004195955116301775
It 8900: Reconstruction Loss: 0.003928026184439659
It 9000: Reconstruction Loss: 0.00427329121157527
It 9100: Reconstruction Loss: 0.004124037455767393
It 9200: Reconstruction Loss: 0.003656135406345129
It 9300: Reconstruction Loss: 0.0038930936716496944
```

Verifying reconstructions [0pt]

Now that we trained the auto-encoder we can visualize some of the reconstructions on the test set to verify that it is converged and did not overfit. **Before continuing, make sure that your auto-encoder is able to reconstruct these samples near-perfectly.**

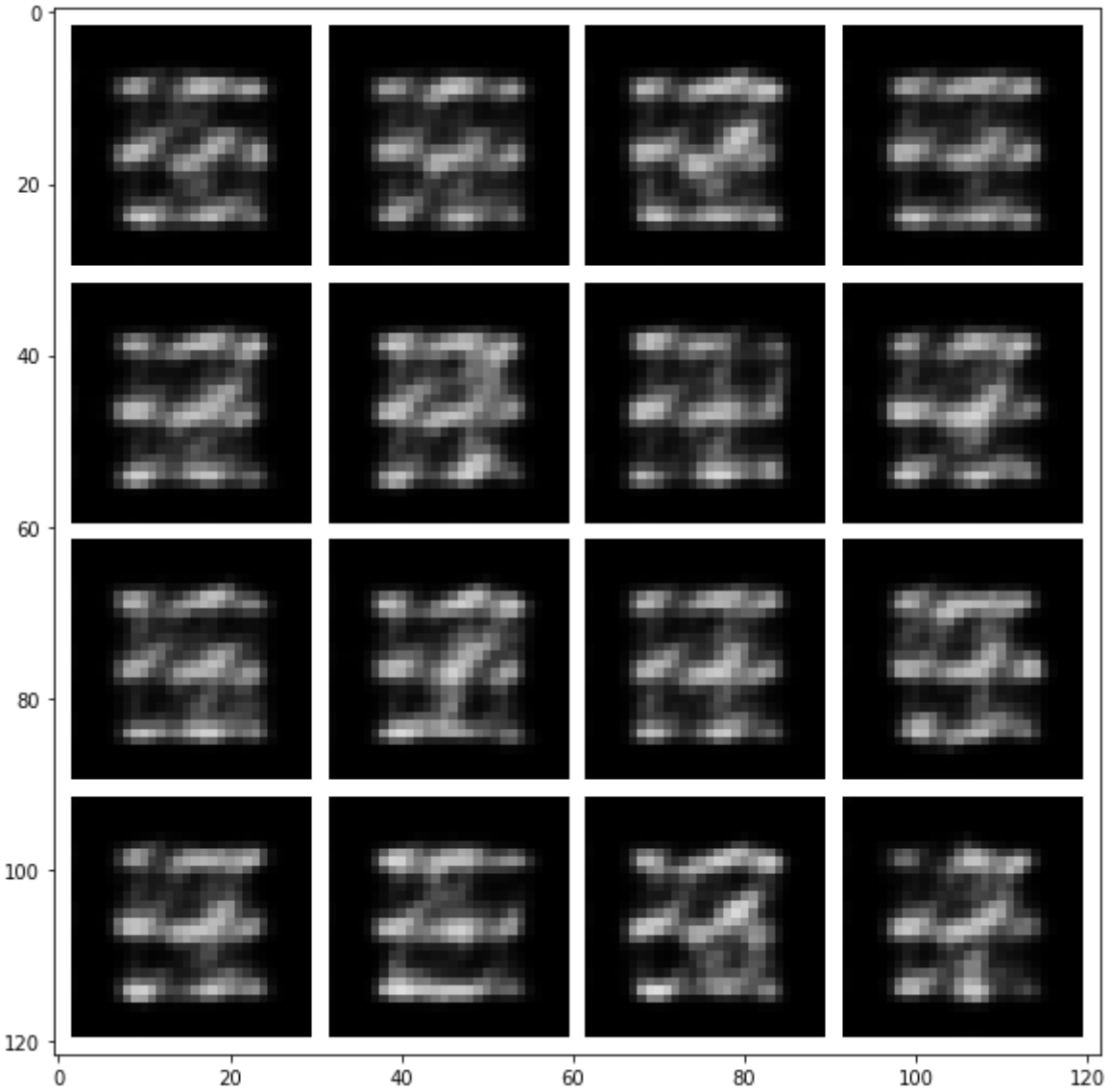
In [ ]:
```
# visualize test data reconstructions
def vis_reconstruction(model):
  # download MNIST test set + build Dataset object
  mnist_test = torchvision.datasets.MNIST(root='./data',
                                          train=False,
                                          download=True,
                                          transform=torchvision.transforms.
  mnist_test_iter = iter(mnist_test)
  model.eval()      # set model in evalidation mode (eg freeze batchnorm pa
  input_imgs, test_reconstructions = [], []
  for _ in range(5):
    input_img = np.asarray(next(mnist_test_iter)[0])
    reconstruction = model.reconstruct(torch.tensor(input_img[None], device
    input_imgs.append(input_img[0])
    test_reconstructions.append(reconstruction[0, 0].data.cpu().numpy())

  fig = plt.figure(figsize = (20, 50))
  ax1 = plt.subplot(111)
  ax1.imshow(np.concatenate([np.concatenate(input_imgs, axis=1),
                             np.concatenate(test_reconstructions, axis=1)],
  plt.show()

vis_reconstruction(ae_model)
```

Sampling from the Auto-Encoder [2pt]

To test whether the auto-encoder is useful as a generative model, we can use it like any other generative model: draw embedding samples from a prior distribution and decode them through the decoder network. We will choose a unit Gaussian prior to allow for easy comparison to the VAE later.

In [ ]:

```
# we will sample N embeddings, then decode and visualize them
def vis_samples(model):
  ######################################## TODO ##############################
  # Sample embeddings from a diagonal unit Gaussian distribution and decode
  # using the model.
  # HINT: The sampled embeddings should have shape [batch_size, nz]. Diagon
  #       Gaussians have mean 0 and a covariance matrix with ones on the di
  #       and zeros everywhere else.
  # HINT: If you are unsure whether you sampled the correct distribution, y
  #       sample a large batch and compute the empirical mean and variance
  #       .mean() and .var() functions.
  # HINT: You can directly use model.decoder() to decode the samples.
  ############################################################################
  sampled_embeddings = torch.randn(batch_size, nz).to(device)    # sample b
  decoded_samples = model.decoder(sampled_embeddings)       # decoder outpu
  ######################################## END TODO ##########################

  fig = plt.figure(figsize = (10, 10))
  ax1 = plt.subplot(111)
  ax1.imshow(torchvision.utils.make_grid(decoded_samples[:16], nrow=4, pad_
                .data.cpu().numpy().transpose(1, 2, 0), cmap='gray')
  plt.show()

vis_samples(ae_model)
```



> **Inline Question: Describe your observations, why do you think they occur? [2pt]** \ (please limit your answer to <150 words) \ **Answer:** The images generated by the model are not distinct. To be specific the model is trying to generate digits but it is not clearly able to generate a single digit. It seems that two or more digits which are partially generated are overlapping. For example I can see, traces of 3 and 5 overlapping, or may be 4 is converting to 7 or vice versa. Another example is first image in second row has traces of 3 and 6. It can also be noted that the images generated are not in shape and are very blurry.

3. Variational Auto-Encoder (VAE)

Variational auto-encoders use a very similar architecture to deterministic auto-encoders, but are inherently storchastic models, i.e. we perform a stochastic sampling operation during the forward pass, leading to different different outputs every time we run the network for the same input. This sampling is required to optimize the VAE objective also known as the evidence lower bound (ELBO):

$$ p(x) > \underbrace{\mathbb{E}_{z\sim q(z\vert x)} p(x \vert z)}_{\text{reconstruction}} - \underbrace{D_{\text{KL}}\big(q(z \vert x), p(z)\big)}_{\text{prior divergence}} $$
Here, $D_{\text{KL}}(q, p)$ denotes the Kullback-Leibler (KL) divergence between the posterior distribution $q(z \vert x)$, i.e. the output of our encoder, and $p(z)$, the prior over the embedding variable z, which we can choose freely.

For simplicity, we will again choose a unit Gaussian prior. The left term is the reconstruction term we already know from training the auto-encoder. When assuming a Gaussian output distribution for both encoder $q(z \vert x)$ and decoder $p(x \vert z)$ the objective reduces to:

$$ \mathcal{L}_{\text{VAE}} = \sum_{x\sim \mathcal{D}} (x - \hat{x})^2 - \beta \cdot D_{\text{KL}}\big(\mathcal{N}(\mu_q, \sigma_q), \mathcal{N}(0, I)\big) $$
Here, $\hat{x}$ is the reconstruction output of the decoder. In comparison to the auto-encoder objetive, the VAE adds a regularizing term between the output of the encoder and a chosen prior distribution, effectively forcing the encoder output to not stray too far from the prior during training. As a result the decoder gets trained with samples that look pretty similar to samples from the prior, which will hopefully allow us to generate better images when using the VAE as a generative model and actually feeding it samples from the prior (as we have done for the AE before).

The coefficient β is a scalar weighting factor that trades off between reconstruction and regularization objective. We will investigate the influence of this factor in out experiments below.

If you need a refresher on VAEs you can check out this tutorial paper: https://arxiv.org/abs/1606.05908

Reparametrization Trick

The sampling procedure inside the VAE's forward pass for obtaining a sample z from the posterior distribution $q(z \vert x)$, when implemented naively, is non-differentiable. However, since $q(z\vert x)$ is parametrized with a Gaussian function, there is a simple trick to obtain a differentiable sampling operator, known as the *reparametrization trick*.

Instead of directly sampling $z \sim \mathcal{N}(\mu_q, \sigma_q)$ we can "separate" the network's predictions and the random sampling by computing the sample as:

$$ z = \mu_q + \sigma_q * \epsilon , \quad \epsilon \sim \mathcal{N}(0, I) $$
Note that in this equation, the sample z is computed as a deterministic function of the network's predictions μ_q and σ_q and therefore allows to propagate gradients through the sampling procedure.

Note: While in the equations above the encoder network parametrizes the standard deviation σ_q of the Gaussian posterior distribution, in practice we usually parametrize the **logarithm of the standard deviation** $\log \sigma_q$ for numerical stability. Before

Defining the VAE Model [7pt]

In [ ]:

```
def kl_divergence(mu1, log_sigma1, mu2, log_sigma2):
  """Computes KL[p||q] between two Gaussians defined by [mu, log_sigma]."""
  return (log_sigma2 - log_sigma1) + (torch.exp(log_sigma1) ** 2 + (mu1 - m
              / (2 * torch.exp(log_sigma2) ** 2) - 0.5


class VAE(nn.Module):
  def __init__(self, nz, beta=1.0):
    super().__init__()
    self.beta = beta          # factor trading off between two loss compone
    ######################################## TODO ############################
    # Instantiate Encoder and Decoder.
    # HINT: Remember that the encoder is now parametrizing a Gaussian distr
    #       mean and log_sigma, so the dimensionality of the output embeddi
    #       double.
    ##########################################################################
    self.encoder = Encoder(nz * 2)
    self.decoder = Decoder(nz)
    self.nz = nz
    ######################################## END TODO ########################

  def forward(self, x):
    ######################################## TODO ############################
    # Implement the forward pass of the VAE.
    # HINT: Your code should implement the following steps:
    #          1. encode input x, split encoding into mean and log_sigma of
    #          2. sample z from inferred posterior distribution using
    #             reparametrization trick
    #          3. decode the sampled z to obtain the reconstructed image
    ##########################################################################
    # encode input into posterior distribution q(z | x)
    q = self.encoder(x)       # output of encoder (concatenated mean and lo

    # sample latent variable z with reparametrization
    mu = q[:, : self.nz]
    log_sigma = q[:, self.nz: ]

    std = torch.exp(log_sigma)
    eps = torch.randn_like(std)
    z = mu + (eps * std)       # batch of sampled embeddings

    # compute reconstruction
    reconstruction = self.decoder(z)   # decoder reconstruction from embedd
    ######################################## END TODO ########################

    return {'q': q,
            'rec': reconstruction}

  def loss(self, x, outputs):
    ######################################## TODO ############################
    # Implement the loss computation of the VAE.
    # HINT: Your code should implement the following steps:
    #          1. compute the image reconstruction loss, similar to AE we u
    #          2. compute the KL divergence loss between the inferred poste
    #             distribution and a unit Gaussian prior; you can use the p
    #             function above for computing the KL divergence between tw
    #             parametrized by mean and log_sigma
    # HINT: Make sure to compute the KL divergence in the correct order sin
    #       not symmetric, ie. KL(p, q) != KL(q, p)!
    ##########################################################################
    # compute reconstruction loss
    criterion = nn.MSELoss()
    rec_loss = criterion(x, outputs['rec'])
```

```
    rec_loss = criterion(x, outputs['rec'])

    # compute KL divergence loss
    q = outputs['q']
    mu1 = q[:, 0 : self.nz]
    log_sigma1 = q[:, self.nz: ]

    # Create a unit gaussian prior
    mu2 = torch.zeros_like(mu1)
    log_sigma2 = torch.zeros_like(log_sigma1)

    kl_loss = kl_divergence(mu1, log_sigma1, mu2, log_sigma2).sum(dim = 1).
    ######################################## END TODO ##########################

    # return weihgted objective
    return rec_loss + self.beta * kl_loss, \
           {'rec_loss': rec_loss, 'kl_loss': kl_loss}

  def reconstruct(self, x):
    """Use mean of posterior estimate for visualization reconstruction."""
    ######################################## TODO ##############################
    # This function is used for visualizing reconstructions of our VAE mode
    # obtain the maximum likelihood estimate we bypass the sampling procedu
    # inferred latent and instead directly use the mean of the inferred pos
    # HINT: encode the input image and then decode the mean of the posteric
    #       the reconstruction.
    ############################################################################
    q = self.encoder(x)

    total_len = q.shape
    mid = total_len[1] // 2
    mu1 = q[:, 0 : mid]

    reconstruction = self.decoder(mu1)
    ######################################## END TODO ##########################
    return reconstruction
```

Setting up the VAE Training Loop [4pt]

Let's start training the VAE model! We will first verify our implementation by setting $\beta = 0$.

In [ ]:

```
learning_rate = 1e-3
nz = 64

######################################## TODO ##############################
# Tune the beta parameter to obtain good VAE training results. However, for
# initial experiments leave beta = 0 in order to verify our implementation.
############################################################################
epochs = 32          # using 5 epochs is sufficient for the first two experi
                    # for the experiment where you tune beta, 20 epochs are
beta = 32e-4
####################################### END TODO ###########################

# build VAE model
vae_model = VAE(nz, beta).to(device)    # transfer model to GPU if availabl
vae_model = vae_model.train()   # set model in train mode (eg batchnorm par

# build optimizer and loss function
######################################## TODO ##############################
# Build the optimizer for the vae_model. We will again use the Adam optimiz
# the given learning rate and otherwise default parameters.
############################################################################
opt = torch.optim.Adam(vae_model.parameters(), lr=learning_rate)
####################################### END TODO ###########################

train_it = 0
rec_loss, kl_loss = [], []
for ep in range(epochs):
  print("Run Epoch {}".format(ep))
  ######################################## TODO ##############################
  # Implement the main training loop for the VAE model.
  # HINT: Your training loop should sample batches from the data loader, ru
  #       forward pass of the VAE, compute the loss, perform the backward p
  #       perform one gradient step with the optimizer.
  # HINT: Don't forget to erase old gradients before performing the backwar
  # HINT: This time we will use the loss() function of our model for comput
  #       training loss. It outputs the total training loss and a dict cont
  #       the breakdown of reconstruction and KL loss.
  ############################################################################
  for sample_img, sample_label in mnist_data_loader:
    # add VAE training loop commands here
    # ...

    # move to device
    sample_img = sample_img.to(device)

    opt.zero_grad()
    outputs = vae_model.forward(sample_img)
    total_loss, losses = vae_model.loss(sample_img, outputs)
    losses['rec_loss'].backward()
    opt.step()

    losses['rec_loss'] = losses['rec_loss'].detach().cpu()
    losses['kl_loss'] = losses['kl_loss'].detach().cpu()

    ####################################### END TODO ###########################

    rec_loss.append(losses['rec_loss']); kl_loss.append(losses['kl_loss'])
    if train_it % 100 == 0:
      print("It {}: Total Loss: {}, \t Rec Loss: {},\t KL Loss: {}"\
            .format(train_it, total_loss, losses['rec_loss'], losses['kl_lo
    train_it += 1

print("Done!")
```

```
print("Done!")

# log the loss training curves
fig = plt.figure(figsize = (10, 5))
ax1 = plt.subplot(121)
ax1.plot(rec_loss)
ax1.title.set_text("Reconstruction Loss")
ax2 = plt.subplot(122)
ax2.plot(kl_loss)
ax2.title.set_text("KL Loss")
plt.show()
```

```
Run Epoch 0
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
  cpuset_checked))
It 0: Total Loss: 0.2774864435195923,    Rec Loss: 0.21992328763008118,  KL Loss: 17.988483428955078
It 100: Total Loss: 0.4779055714607239,          Rec Loss: 0.030961547046899796,    KL Loss: 139.67001342773438
It 200: Total Loss: 0.7319027185440063,          Rec Loss: 0.020007126033306122,    KL Loss: 222.46737670898438
It 300: Total Loss: 0.8326473236083984,          Rec Loss: 0.014163658954203129,    KL Loss: 255.77615356445312
It 400: Total Loss: 0.9007357358932495,          Rec Loss: 0.011664076708257198,    KL Loss: 277.83489990234375
It 500: Total Loss: 0.9605934023857117,          Rec Loss: 0.012227545492351055,    KL Loss: 296.3643493652344
It 600: Total Loss: 1.0072956085205078,          Rec Loss: 0.011902923695743084,    KL Loss: 311.0602111816406
It 700: Total Loss: 1.0459448099136353,          Rec Loss: 0.011220490559933557,    KL Loss: 323.35137939453125
It 800: Total Loss: 1.0774550437927246,          Rec Loss: 0.011518415063619614,    KL Loss: 333.105224609375
It 900: Total Loss: 1.102566123008728,   Rec Loss: 0.009032628498971462,
         KL Loss: 341.7292175292969
Run Epoch 1
It 1000: Total Loss: 1.130315899848938,          Rec Loss: 0.008511689491569996,    KL Loss: 350.5638427734375
It 1100: Total Loss: 1.151742935180664,          Rec Loss: 0.009394929744303226,    KL Loss: 356.9837646484375
It 1200: Total Loss: 1.1788643598556519,         Rec Loss: 0.007960422895848751,    KL Loss: 365.907470703125
It 1300: Total Loss: 1.2035024166107178,         Rec Loss: 0.008567946963012218,    KL Loss: 373.41705322265625
It 1400: Total Loss: 1.2265050411224365,         Rec Loss: 0.008329198695719242,    KL Loss: 380.6799621582031
It 1500: Total Loss: 1.2449636459350586,         Rec Loss: 0.008045295253396034,    KL Loss: 386.5369873046875
It 1600: Total Loss: 1.2702120542526245,         Rec Loss: 0.008302547968924046,    KL Loss: 394.34674072265625
It 1700: Total Loss: 1.2851167917251587,         Rec Loss: 0.006771530490368605,    KL Loss: 399.48291015625
It 1800: Total Loss: 1.306889295578003,          Rec Loss: 0.007455016952008009,    KL Loss: 406.0732421875
Run Epoch 2
It 1900: Total Loss: 1.323966383934021,          Rec Loss: 0.007028057239949703,    KL Loss: 411.543212890625
It 2000: Total Loss: 1.3477437496185303,         Rec Loss: 0.007306357845664024,    KL Loss: 418.8866882324219
```

```
It 2100: Total Loss: 1.3664422035217285,         Rec Loss: 0.00658883480355
14355,   KL Loss: 424.9541931152344
It 2200: Total Loss: 1.38427734375,      Rec Loss: 0.006209683604538441,
         KL Loss: 430.6461486816406
It 2300: Total Loss: 1.4032166004180908,         Rec Loss: 0.00718410266563
2963,    KL Loss: 436.2601623535156
It 2400: Total Loss: 1.4290953874588013,         Rec Loss: 0.00711045693606
1382,    KL Loss: 444.37030029296875
It 2500: Total Loss: 1.4324791431427002,         Rec Loss: 0.00560893444344
4014,    KL Loss: 445.8969421386719
It 2600: Total Loss: 1.4486968517303467,         Rec Loss: 0.00665817270055
41325,   KL Loss: 450.6370849609375
It 2700: Total Loss: 1.4847843647003174,         Rec Loss: 0.00676075648516
41655,   KL Loss: 461.8824157714844
It 2800: Total Loss: 1.5002268552780151,         Rec Loss: 0.00663118204101
9201,    KL Loss: 466.7486877441406
Run Epoch 3
It 2900: Total Loss: 1.512379765510559,          Rec Loss: 0.00589347071945
6673,    KL Loss: 470.7769775390625
It 3000: Total Loss: 1.5423152446746826,         Rec Loss: 0.00712254922837
019,     KL Loss: 479.74774169921875
It 3100: Total Loss: 1.5565309524536133,         Rec Loss: 0.00572328083217
144,     KL Loss: 484.62744140625
It 3200: Total Loss: 1.577323079109192,          Rec Loss: 0.00692464830353
8561,    KL Loss: 490.74951171875
It 3300: Total Loss: 1.5972665548324585,         Rec Loss: 0.00765071902424
0971,    KL Loss: 496.75494384765625
It 3400: Total Loss: 1.6176013946533203,         Rec Loss: 0.00693783676251
7691,    KL Loss: 503.3323669433594
It 3500: Total Loss: 1.613166093826294,          Rec Loss: 0.00564650585874
9151,    KL Loss: 502.34991455078125
It 3600: Total Loss: 1.6424843072891235,         Rec Loss: 0.00537592638283
968,     KL Loss: 511.59637451171875
It 3700: Total Loss: 1.6607036590576172,         Rec Loss: 0.00585298426449
2989,    KL Loss: 517.140869140625
Run Epoch 4
It 3800: Total Loss: 1.668525218963623,          Rec Loss: 0.00663072336465
1203,    KL Loss: 519.342041015625
It 3900: Total Loss: 1.6916613578796387,         Rec Loss: 0.00615688459947
7053,    KL Loss: 526.7201538085938
It 4000: Total Loss: 1.714827299118042,          Rec Loss: 0.00542154768481
8506,    KL Loss: 534.1893310546875
It 4100: Total Loss: 1.7321176528930664,         Rec Loss: 0.00566127803176
6415,    KL Loss: 539.5176391601562
It 4200: Total Loss: 1.7553743124008179,         Rec Loss: 0.00607981579378
2473,    KL Loss: 546.654541015625
It 4300: Total Loss: 1.7606512308120728,         Rec Loss: 0.00482209911569
953,     KL Loss: 548.6965942382812
It 4400: Total Loss: 1.7789299488067627,         Rec Loss: 0.00541949039325
1181,    KL Loss: 554.2220458984375
It 4500: Total Loss: 1.786085844039917,          Rec Loss: 0.00494317244738
3404,    KL Loss: 556.6071166992188
It 4600: Total Loss: 1.8270859718322754,         Rec Loss: 0.00565514015033
8411,    KL Loss: 569.1971435546875
Run Epoch 5
It 4700: Total Loss: 1.843637466430664,          Rec Loss: 0.00556664774194
3598,    KL Loss: 574.3971557617188
It 4800: Total Loss: 1.858289122581482,          Rec Loss: 0.00632513640448
451,     KL Loss: 578.73876953125
It 4900: Total Loss: 1.8836029767990112,         Rec Loss: 0.00499674677848
8159,    KL Loss: 587.064453125
It 5000: Total Loss: 1.902522087097168,          Rec Loss: 0.00481174793094
3966,    KL Loss: 593.0344848632812
```

```
It 5100: Total Loss: 1.9192194938659668,         Rec Loss: 0.00485937297344
2078,    KL Loss: 598.237548828125
It 5200: Total Loss: 1.936258316040039,          Rec Loss: 0.00491855246946
2156,    KL Loss: 603.543701171875
It 5300: Total Loss: 1.9503813982009888,         Rec Loss: 0.00473559694364
6669,    KL Loss: 608.0143432617188
It 5400: Total Loss: 1.975095510482788,          Rec Loss: 0.00558425579220
0565,    KL Loss: 615.4722900390625
It 5500: Total Loss: 1.9986612796783447,         Rec Loss: 0.00564643926918
5066,    KL Loss: 622.817138671875
It 5600: Total Loss: 2.0086140632629395,         Rec Loss: 0.00522227492183
4469,    KL Loss: 626.0599365234375
Run Epoch 6
It 5700: Total Loss: 2.02952241897583,   Rec Loss: 0.0044502634555101395,
         KL Loss: 632.8350830078125
It 5800: Total Loss: 2.0493009090423584,         Rec Loss: 0.00451282458379
8647,    KL Loss: 638.9962768554688
It 5900: Total Loss: 2.0689947605133057,         Rec Loss: 0.00495099276304
245,     KL Loss: 645.013671875
It 6000: Total Loss: 2.094130039215088,          Rec Loss: 0.00469426112249
4936,    KL Loss: 652.94873046875
It 6100: Total Loss: 2.114453077316284,          Rec Loss: 0.00565573014318
943,     KL Loss: 658.9991455078125
It 6200: Total Loss: 2.1350691318511963,         Rec Loss: 0.00495060579851
2697,    KL Loss: 665.662109375
It 6300: Total Loss: 2.1467838287353516,         Rec Loss: 0.00519000971689
8203,    KL Loss: 669.2481079101562
It 6400: Total Loss: 2.187685489654541,          Rec Loss: 0.00567197659984
231,     KL Loss: 681.8792724609375
It 6500: Total Loss: 2.180095911026001,          Rec Loss: 0.00508099375292
6588,    KL Loss: 679.6921997070312
Run Epoch 7
It 6600: Total Loss: 2.2090466022491455,         Rec Loss: 0.00447110831737
5183,    KL Loss: 688.9298706054688
It 6700: Total Loss: 2.2277493476867676,         Rec Loss: 0.00436911592260
0031,    KL Loss: 694.8063354492188
It 6800: Total Loss: 2.2506725788116455,         Rec Loss: 0.00470764003694
0575,    KL Loss: 701.8640747070312
It 6900: Total Loss: 2.2832400798797607,         Rec Loss: 0.00491534499451
5181,    KL Loss: 711.9765014648438
It 7000: Total Loss: 2.2968590259552,    Rec Loss: 0.0053110974840819836,
         KL Loss: 716.1087646484375
It 7100: Total Loss: 2.3148977756500244,         Rec Loss: 0.00547448825091
1236,    KL Loss: 721.6947631835938
It 7200: Total Loss: 2.3346030712127686,         Rec Loss: 0.00531032169237
7329,    KL Loss: 727.9039916992188
It 7300: Total Loss: 2.373723268508911,          Rec Loss: 0.00464492803439
49795,   KL Loss: 740.3370361328125
It 7400: Total Loss: 2.3963327407836914,         Rec Loss: 0.00494982907548
5468,    KL Loss: 747.3071899414062
It 7500: Total Loss: 2.4058775901794434,         Rec Loss: 0.00478109065443
2774,    KL Loss: 750.3427124023438
Run Epoch 8
It 7600: Total Loss: 2.436552047729492,          Rec Loss: 0.00479691987857
2226,    KL Loss: 759.9234619140625
It 7700: Total Loss: 2.4529664516448975,         Rec Loss: 0.00493241194635
62965,   KL Loss: 765.0106811523438
It 7800: Total Loss: 2.481835126876831,          Rec Loss: 0.00486395834013
8197,    KL Loss: 774.0535278320312
It 7900: Total Loss: 2.5118844509124756,         Rec Loss: 0.00450830161571
5027,    KL Loss: 783.5551147460938
It 8000: Total Loss: 2.5399680137634277,         Rec Loss: 0.00536237331107
2588,    KL Loss: 792.0643310546875
```

```
It 8100: Total Loss: 2.552361249923706,         Rec Loss: 0.00502146454527
9741,    KL Loss: 796.043701171875
It 8200: Total Loss: 2.5769786834716797,        Rec Loss: 0.00447487132623
7917,    KL Loss: 803.907470703125
It 8300: Total Loss: 2.589060068130493,         Rec Loss: 0.00425026845186
9488,    KL Loss: 807.7530517578125
It 8400: Total Loss: 2.6047215461730957,        Rec Loss: 0.00404173368588
0899,    KL Loss: 812.7124633789062
Run Epoch 9
It 8500: Total Loss: 2.637462854385376,         Rec Loss: 0.00411361269652
8435,    KL Loss: 822.921630859375
It 8600: Total Loss: 2.6765949726104736,        Rec Loss: 0.00460720062255
8594,    KL Loss: 834.9962158203125
It 8700: Total Loss: 2.699052095413208,         Rec Loss: 0.00471660494804
3823,    KL Loss: 841.9798583984375
It 8800: Total Loss: 2.7238516807556152,        Rec Loss: 0.00470954738557
3387,    KL Loss: 849.73193359375
It 8900: Total Loss: 2.749663829803467,         Rec Loss: 0.00418854970484
972,     KL Loss: 857.9610595703125
It 9000: Total Loss: 2.7636239528656006,        Rec Loss: 0.00443407986313
1046,    KL Loss: 862.246826171875
It 9100: Total Loss: 2.79076886177063,   Rec Loss: 0.004247513134032488,
         KL Loss: 870.7879638671875
It 9200: Total Loss: 2.802940845489502,         Rec Loss: 0.00416485872119
66515,   KL Loss: 874.6174926757812
It 9300: Total Loss: 2.832237720489502,         Rec Loss: 0.00455129146575
9277,    KL Loss: 883.652099609375
Run Epoch 10
It 9400: Total Loss: 2.8359737396240234,        Rec Loss: 0.00471646245568
9907,    KL Loss: 884.7679443359375
It 9500: Total Loss: 2.893840789794922,         Rec Loss: 0.00469406228512
5256,    KL Loss: 902.8583984375
It 9600: Total Loss: 2.921339273452759,         Rec Loss: 0.00460641644895
07675,   KL Loss: 911.47900390625
It 9700: Total Loss: 2.93939471244812,   Rec Loss: 0.004112901631742716,
         KL Loss: 917.2755737304688
It 9800: Total Loss: 2.965461254119873,         Rec Loss: 0.00404720706865
1915,    KL Loss: 925.44189453125
It 9900: Total Loss: 2.9857583045959473,        Rec Loss: 0.00437527196481
8239,    KL Loss: 931.6822509765625
It 10000: Total Loss: 3.025643825531006,        Rec Loss: 0.00448724534362
55455,   KL Loss: 944.1114501953125
It 10100: Total Loss: 3.027818202972412,        Rec Loss: 0.00398918474093
0796,    KL Loss: 944.9465942382812
It 10200: Total Loss: 3.0687053203582764,       Rec Loss: 0.00510968640446
6629,    KL Loss: 957.3736572265625
It 10300: Total Loss: 3.0727362632751465,       Rec Loss: 0.00390825280919
6711,    KL Loss: 959.0087890625
Run Epoch 11
It 10400: Total Loss: 3.0769262313842773,       Rec Loss: 0.00363808963447
8092,    KL Loss: 960.402587890625
It 10500: Total Loss: 3.110595464706421,        Rec Loss: 0.00417428417131
3047,    KL Loss: 970.7566528320312
It 10600: Total Loss: 3.137021541595459,        Rec Loss: 0.00387955061160
028,     KL Loss: 979.1068725585938
It 10700: Total Loss: 3.1840057373046875,       Rec Loss: 0.00458821700885
8919,    KL Loss: 993.5680541992188
It 10800: Total Loss: 3.1893532276153564,       Rec Loss: 0.00411293981596
8275,    KL Loss: 995.3876342773438
It 10900: Total Loss: 3.2263028621673584,       Rec Loss: 0.00430353032425
046,     KL Loss: 1006.8748168945312
It 11000: Total Loss: 3.259866714477539,        Rec Loss: 0.00410383008420
4674,    KL Loss: 1017.4259033203125
```

```
It 11100: Total Loss: 3.28562068939209,         Rec Loss: 0.00440895557403
56445,   KL Loss: 1025.378662109375
It 11200: Total Loss: 3.315359115600586,        Rec Loss: 0.00440288055688
1428,    KL Loss: 1034.673828125
Run Epoch 12
It 11300: Total Loss: 3.3569817543029785,       Rec Loss: 0.00398242566734
5524,    KL Loss: 1047.812255859375
It 11400: Total Loss: 3.338604211807251,        Rec Loss: 0.00413873139768
8389,    KL Loss: 1042.0205078125
It 11500: Total Loss: 3.363919734954834,        Rec Loss: 0.00406595086678
8626,    KL Loss: 1049.954345703125
It 11600: Total Loss: 3.395744562149048,        Rec Loss: 0.00359237054362
8931,    KL Loss: 1060.047607421875
It 11700: Total Loss: 3.398911952972412,        Rec Loss: 0.00386428157798
9459,    KL Loss: 1060.952392578125
It 11800: Total Loss: 3.428546905517578,        Rec Loss: 0.00414969585835
9337,    KL Loss: 1070.1241455078125
It 11900: Total Loss: 3.4480550289154053,       Rec Loss: 0.00434812204912
3049,    KL Loss: 1076.158447265625
It 12000: Total Loss: 3.465754508972168,        Rec Loss: 0.00433083157986
4025,    KL Loss: 1081.6949462890625
It 12100: Total Loss: 3.4740633964538574,       Rec Loss: 0.00394451199099
4215,    KL Loss: 1084.4122314453125
Run Epoch 13
It 12200: Total Loss: 3.522129535675049,        Rec Loss: 0.00379054783843
4577,    KL Loss: 1099.48095703125
It 12300: Total Loss: 3.5470144748687744,       Rec Loss: 0.00434980448335
4092,    KL Loss: 1107.082763671875
It 12400: Total Loss: 3.564781427383423,        Rec Loss: 0.00418167188763
6185,    KL Loss: 1112.6875
It 12500: Total Loss: 3.568160057067871,        Rec Loss: 0.00371964578516
78133,   KL Loss: 1113.8876953125
It 12600: Total Loss: 3.611981153488159,        Rec Loss: 0.00360085163265
4667,    KL Loss: 1127.618896484375
It 12700: Total Loss: 3.632220506668091,        Rec Loss: 0.00383919430896
63982,   KL Loss: 1133.869140625
It 12800: Total Loss: 3.644129991531372,        Rec Loss: 0.00366319133900
10595,   KL Loss: 1137.6458740234375
It 12900: Total Loss: 3.6695828437805176,       Rec Loss: 0.00382528663612
90216,   KL Loss: 1145.54931640625
It 13000: Total Loss: 3.68013596534729,         Rec Loss: 0.00386013649404
0489,    KL Loss: 1148.836181640625
It 13100: Total Loss: 3.7122390270233154,       Rec Loss: 0.00390179106034
33847,   KL Loss: 1158.85546875
Run Epoch 14
It 13200: Total Loss: 3.7361929416656494,       Rec Loss: 0.00338671100325
88243,   KL Loss: 1166.501953125
It 13300: Total Loss: 3.7613213062286377,       Rec Loss: 0.00353970471769
5713,    KL Loss: 1174.3067626953125
It 13400: Total Loss: 3.7764878273010254,       Rec Loss: 0.00460598105564
7135,    KL Loss: 1178.713134765625
It 13500: Total Loss: 3.7983922958374023,       Rec Loss: 0.00405638245865
7026,    KL Loss: 1185.72998046875
It 13600: Total Loss: 3.820723295211792,        Rec Loss: 0.00390685116872
1914,    KL Loss: 1192.755126953125
It 13700: Total Loss: 3.8380980491638184,       Rec Loss: 0.00377457402646
5416,    KL Loss: 1198.22607421875
It 13800: Total Loss: 3.876309871673584,        Rec Loss: 0.00327199837192
89303,   KL Loss: 1210.3243408203125
It 13900: Total Loss: 3.8860723972320557,       Rec Loss: 0.00424073822796
3448,    KL Loss: 1213.0723876953125
It 14000: Total Loss: 3.918910264968872,        Rec Loss: 0.00380454794503
7484,    KL Loss: 1223.4705810546875
```

```
Run Epoch 15
It 14100: Total Loss: 3.926501750946045,         Rec Loss: 0.00470618205145
0014,    KL Loss: 1225.5611572265625
It 14200: Total Loss: 3.955382823944092,         Rec Loss: 0.00367433740757
40576,   KL Loss: 1234.908935546875
It 14300: Total Loss: 3.9658830165863037,        Rec Loss: 0.00349806458689
2724,    KL Loss: 1238.245361328125
It 14400: Total Loss: 4.005494594573975,         Rec Loss: 0.00379849388264
11963,   KL Loss: 1250.530029296875
It 14500: Total Loss: 4.01389741897583,          Rec Loss: 0.00404831254854
7983,    KL Loss: 1253.077880859375
It 14600: Total Loss: 3.988095760345459,         Rec Loss: 0.00315840169787
4069,    KL Loss: 1245.29296875
It 14700: Total Loss: 4.025870323181152,         Rec Loss: 0.00391772342845
7975,    KL Loss: 1256.8602294921875
It 14800: Total Loss: 4.045266151428223,         Rec Loss: 0.00383653445169
32964,   KL Loss: 1262.94677734375
It 14900: Total Loss: 4.0716352462768555,        Rec Loss: 0.00375193660147
4881,    KL Loss: 1271.213623046875
It 15000: Total Loss: 4.083459377288818,         Rec Loss: 0.00322805601172
1492,    KL Loss: 1275.072265625
Run Epoch 16
It 15100: Total Loss: 4.131476402282715,         Rec Loss: 0.00419882638379
93145,   KL Loss: 1289.774169921875
It 15200: Total Loss: 4.128103733062744,         Rec Loss: 0.00355011830106
3776,    KL Loss: 1288.923095703125
It 15300: Total Loss: 4.160480499267578,         Rec Loss: 0.00359271909110
2481,    KL Loss: 1299.027587890625
It 15400: Total Loss: 4.171641826629639,         Rec Loss: 0.00324987969361
24563,   KL Loss: 1302.62255859375
It 15500: Total Loss: 4.200206279754639,         Rec Loss: 0.00409222627058
6252,    KL Loss: 1311.28564453125
It 15600: Total Loss: 4.196268081665039,         Rec Loss: 0.00379667477682
23286,   KL Loss: 1310.1473388671875
It 15700: Total Loss: 4.225154399871826,         Rec Loss: 0.00394685612991
4522,    KL Loss: 1319.12744140625
It 15800: Total Loss: 4.241907119750977,         Rec Loss: 0.00366440415382
38525,   KL Loss: 1324.4508056640625
It 15900: Total Loss: 4.257602214813232,         Rec Loss: 0.00419574417173
8625,    KL Loss: 1329.1895751953125
Run Epoch 17
It 16000: Total Loss: 4.241065979003906,         Rec Loss: 0.00319479615427
55365,   KL Loss: 1324.3348388671875
It 16100: Total Loss: 4.294215679168701,         Rec Loss: 0.00347163435071
70677,   KL Loss: 1340.857421875
It 16200: Total Loss: 4.313192844390869,         Rec Loss: 0.00362501037307
0836,    KL Loss: 1346.739990234375
It 16300: Total Loss: 4.309237003326416,         Rec Loss: 0.00331999291665
85207,   KL Loss: 1345.5989990234375
It 16400: Total Loss: 4.360329627990723,         Rec Loss: 0.00361820380203
4259,    KL Loss: 1361.472412109375
It 16500: Total Loss: 4.3884077072143555,        Rec Loss: 0.00327215087600
05236,   KL Loss: 1370.35498046875
It 16600: Total Loss: 4.380279064178467,         Rec Loss: 0.00317736994475
12627,   KL Loss: 1367.8443603515625
It 16700: Total Loss: 4.441582202911377,         Rec Loss: 0.00375045533291
99553,   KL Loss: 1386.822509765625
It 16800: Total Loss: 4.444010257720947,         Rec Loss: 0.00412761187180
8767,    KL Loss: 1387.46337890625
Run Epoch 18
It 16900: Total Loss: 4.457760334014893,         Rec Loss: 0.00359787698835
1345,    KL Loss: 1391.92578125
It 17000: Total Loss: 4.467045307159424,         Rec Loss: 0.00335479271598
```

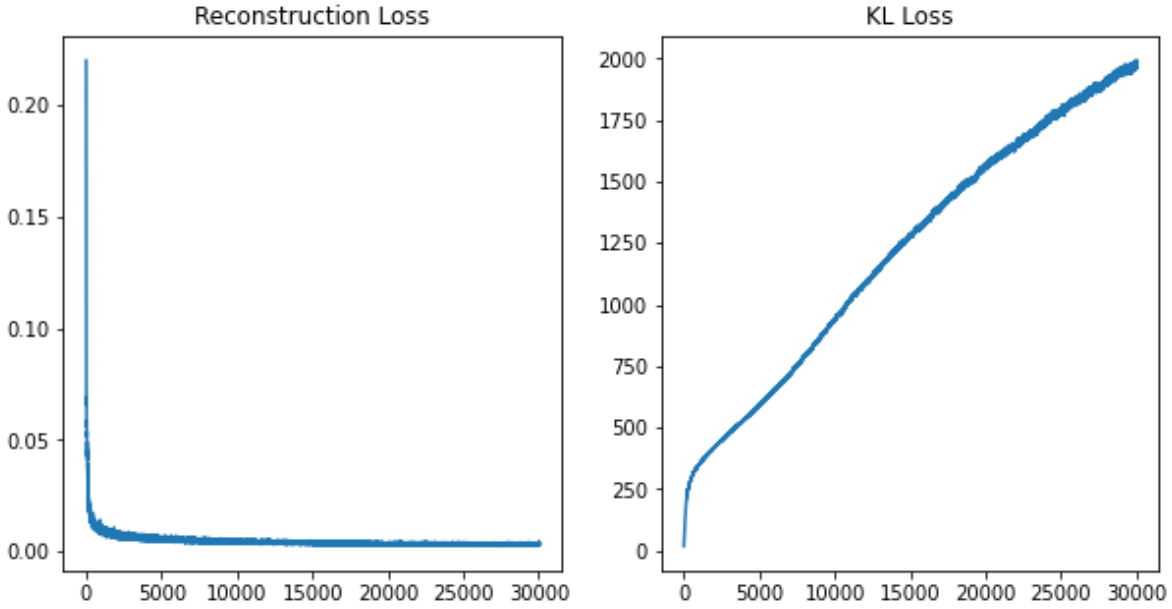
```
16027,   KL Loss: 1394.9033203125
It 17100: Total Loss: 4.48316764831543,         Rec Loss: 0.00354333245195
4484,    KL Loss: 1399.882568359375
It 17200: Total Loss: 4.505029678344727,        Rec Loss: 0.00370266474783
42056,   KL Loss: 1406.664794921875
It 17300: Total Loss: 4.521445274353027,        Rec Loss: 0.00320015475153
92303,   KL Loss: 1411.95166015625
It 17400: Total Loss: 4.561147212982178,        Rec Loss: 0.00346717284992
33723,   KL Loss: 1424.275146484375
It 17500: Total Loss: 4.568474292755127,        Rec Loss: 0.00393438804894
68575,   KL Loss: 1426.418701171875
It 17600: Total Loss: 4.566519737243652,        Rec Loss: 0.00310497893951
83325,   KL Loss: 1426.067138671875
It 17700: Total Loss: 4.6302008628845215,       Rec Loss: 0.00376658840104
93755,   KL Loss: 1445.7607421875
It 17800: Total Loss: 4.61329984664917,         Rec Loss: 0.00343401194550
097,     KL Loss: 1440.5830078125
Run Epoch 19
It 17900: Total Loss: 4.634116172790527,        Rec Loss: 0.00329333101399
24288,   KL Loss: 1447.132080078125
It 18000: Total Loss: 4.658857822418213,        Rec Loss: 0.00377780036069
45276,   KL Loss: 1454.7125244140625
It 18100: Total Loss: 4.636510372161865,        Rec Loss: 0.00332528166472
91183,   KL Loss: 1447.870361328125
It 18200: Total Loss: 4.667378902435303,        Rec Loss: 0.00306428619660
4371,    KL Loss: 1457.598388671875
It 18300: Total Loss: 4.7315449714660645,       Rec Loss: 0.00312126125209
0335,    KL Loss: 1477.6324462890625
It 18400: Total Loss: 4.714032173156738,        Rec Loss: 0.00345015758648
51475,   KL Loss: 1472.056884765625
It 18500: Total Loss: 4.771265029907227,        Rec Loss: 0.00411878898739
8148,    KL Loss: 1489.733154296875
It 18600: Total Loss: 4.783430099487305,        Rec Loss: 0.00331601058132
9465,    KL Loss: 1493.78564453125
It 18700: Total Loss: 4.754288673400879,        Rec Loss: 0.00334454490803
18213,   KL Loss: 1484.6700439453125
Run Epoch 20
It 18800: Total Loss: 4.758532524108887,        Rec Loss: 0.00304546183906
49557,   KL Loss: 1486.0897216796875
It 18900: Total Loss: 4.797486782073975,        Rec Loss: 0.00365958921611
30905,   KL Loss: 1498.0709228515625
It 19000: Total Loss: 4.823488712310791,        Rec Loss: 0.00389542733319
1037,    KL Loss: 1506.123046875
It 19100: Total Loss: 4.8221354484558105,       Rec Loss: 0.00345384585671
12684,   KL Loss: 1505.838134765625
It 19200: Total Loss: 4.8300299644470215,       Rec Loss: 0.00369184021838
00936,   KL Loss: 1508.230712890625
It 19300: Total Loss: 4.846795082092285,        Rec Loss: 0.00357978488318
6221,    KL Loss: 1513.5048828125
It 19400: Total Loss: 4.872220516204834,        Rec Loss: 0.00309509271755
81455,   KL Loss: 1521.6016845703125
It 19500: Total Loss: 4.8929243087768555,       Rec Loss: 0.00336132780648
76795,   KL Loss: 1527.988525390625
It 19600: Total Loss: 4.938602924346924,        Rec Loss: 0.00323445652611
55367,   KL Loss: 1542.302734375
Run Epoch 21
It 19700: Total Loss: 4.973816394805908,        Rec Loss: 0.00356012815609
5743,    KL Loss: 1553.205078125
It 19800: Total Loss: 4.950754642486572,        Rec Loss: 0.00343057210557
16276,   KL Loss: 1546.038818359375
It 19900: Total Loss: 4.954117774963379,        Rec Loss: 0.00357583700679
2426,    KL Loss: 1547.04443359375
It 20000: Total Loss: 4.984787940979004,        Rec Loss: 0.00306142703630
```

```
03016,   KL Loss: 1556.7896728515625
It 20100: Total Loss: 5.045109748840332,         Rec Loss: 0.00375230936333
5371,    KL Loss: 1575.42431640625
It 20200: Total Loss: 5.040319442749023,         Rec Loss: 0.00305088004097
3425,    KL Loss: 1574.146484375
It 20300: Total Loss: 5.0455169677734375,        Rec Loss: 0.00350734149105
8469,    KL Loss: 1575.6280517578125
It 20400: Total Loss: 5.068269729614258,         Rec Loss: 0.00281632854603
23095,   KL Loss: 1582.9542236328125
It 20500: Total Loss: 5.051268577575684,         Rec Loss: 0.00332032400183
37965,   KL Loss: 1577.48388671875
It 20600: Total Loss: 5.095432281494141,         Rec Loss: 0.00339265260845
42274,   KL Loss: 1591.262451171875
Run Epoch 22
It 20700: Total Loss: 5.107320308685303,         Rec Loss: 0.00321612157858
90818,   KL Loss: 1595.0325927734375
It 20800: Total Loss: 5.112602710723877,         Rec Loss: 0.00294342194683
8498,    KL Loss: 1596.7685546875
It 20900: Total Loss: 5.127891540527344,         Rec Loss: 0.00366360181942
58213,   KL Loss: 1601.3212890625
It 21000: Total Loss: 5.144627571105957,         Rec Loss: 0.00329956132918
59627,   KL Loss: 1606.6650390625
It 21100: Total Loss: 5.138282299041748,         Rec Loss: 0.00347360759042
20343,   KL Loss: 1604.627685546875
It 21200: Total Loss: 5.182568073272705,         Rec Loss: 0.00297306175343
6923,    KL Loss: 1618.62353515625
It 21300: Total Loss: 5.208220481872559,         Rec Loss: 0.00352404336445
03355,   KL Loss: 1626.4677734375
It 21400: Total Loss: 5.195047855377197,         Rec Loss: 0.00312364823184
90744,   KL Loss: 1622.476318359375
It 21500: Total Loss: 5.180099010467529,         Rec Loss: 0.00319016794674
0985,    KL Loss: 1617.7840576171875
Run Epoch 23
It 21600: Total Loss: 5.222617149353027,         Rec Loss: 0.00316396402195
096,     KL Loss: 1631.0792236328125
It 21700: Total Loss: 5.205334186553955,         Rec Loss: 0.00311193917877
97213,   KL Loss: 1625.694580078125
It 21800: Total Loss: 5.28769063949585,          Rec Loss: 0.00329313962720
33453,   KL Loss: 1651.374267578125
It 21900: Total Loss: 5.246123790740967,         Rec Loss: 0.00366227468475
69942,   KL Loss: 1638.269287109375
It 22000: Total Loss: 5.244804859161377,         Rec Loss: 0.00291131762787
6997,    KL Loss: 1638.091796875
It 22100: Total Loss: 5.304995059967041,         Rec Loss: 0.00339138205163
18083,   KL Loss: 1656.751220703125
It 22200: Total Loss: 5.3204498291015625,        Rec Loss: 0.00332047441042
9597,    KL Loss: 1661.6029052734375
It 22300: Total Loss: 5.310042381286621,         Rec Loss: 0.00308415642939
50796,   KL Loss: 1658.4244384765625
It 22400: Total Loss: 5.332967281341553,         Rec Loss: 0.00300709996372
4613,    KL Loss: 1665.6126708984375
It 22500: Total Loss: 5.369832992553711,         Rec Loss: 0.00318542239256
20317,   KL Loss: 1677.077392578125
Run Epoch 24
It 22600: Total Loss: 5.353749752044678,         Rec Loss: 0.00263645336963
23633,   KL Loss: 1672.222900390625
It 22700: Total Loss: 5.3684258460998535,        Rec Loss: 0.00329772662371
397,     KL Loss: 1676.6025390625
It 22800: Total Loss: 5.376986026763916,         Rec Loss: 0.00296518625691
5331,    KL Loss: 1679.381591796875
It 22900: Total Loss: 5.406606674194336,         Rec Loss: 0.00307878735475
24214,   KL Loss: 1688.6024169921875
It 23000: Total Loss: 5.449256896972656,         Rec Loss: 0.00325480941683
```

```
05397,   KL Loss: 1701.8756103515625
It 23100: Total Loss: 5.427193641662598,         Rec Loss: 0.00277926912531
25668,   KL Loss: 1695.12939453125
It 23200: Total Loss: 5.422658920288086,         Rec Loss: 0.00322388252243
3996,    KL Loss: 1693.573486328125
It 23300: Total Loss: 5.464754581451416,         Rec Loss: 0.00295797525905
0727,    KL Loss: 1706.8115234375
It 23400: Total Loss: 5.454076290130615,         Rec Loss: 0.00294203218072
6528,    KL Loss: 1703.4794921875
Run Epoch 25
It 23500: Total Loss: 5.493338108062744,         Rec Loss: 0.00315974792465
5676,    KL Loss: 1715.680908203125
It 23600: Total Loss: 5.506298065185547,         Rec Loss: 0.00301530095748
60334,   KL Loss: 1719.77587890625
It 23700: Total Loss: 5.51902961730957,          Rec Loss: 0.00297467363998
29388,   KL Loss: 1723.7672119140625
It 23800: Total Loss: 5.533520221710205,         Rec Loss: 0.00300449272617
6977,    KL Loss: 1728.2861328125
It 23900: Total Loss: 5.537171840667725,         Rec Loss: 0.00313666020520
03145,   KL Loss: 1729.385986328125
It 24000: Total Loss: 5.57315731048584,          Rec Loss: 0.00295172957703
4712,    KL Loss: 1740.6893310546875
It 24100: Total Loss: 5.6025214195251465,        Rec Loss: 0.00330907804891
4671,    KL Loss: 1749.75390625
It 24200: Total Loss: 5.6038079261779785,        Rec Loss: 0.00301627232693
13574,   KL Loss: 1750.2474365234375
It 24300: Total Loss: 5.628164291381836,         Rec Loss: 0.00320328609086
57312,   KL Loss: 1757.80029296875
Run Epoch 26
It 24400: Total Loss: 5.657005310058594,         Rec Loss: 0.00267876894213
2592,    KL Loss: 1766.97705078125
It 24500: Total Loss: 5.696140766143799,         Rec Loss: 0.00300815701484
68018,   KL Loss: 1779.1038818359375
It 24600: Total Loss: 5.697236061096191,         Rec Loss: 0.00340454280376
43433,   KL Loss: 1779.3223876953125
It 24700: Total Loss: 5.65086030960083,          Rec Loss: 0.00317359459586
4415,    KL Loss: 1764.902099609375
It 24800: Total Loss: 5.666318416595459,         Rec Loss: 0.00311747193336
4868,    KL Loss: 1769.7503662109375
It 24900: Total Loss: 5.6986517906188965,        Rec Loss: 0.00320714921690
52362,   KL Loss: 1779.826416015625
It 25000: Total Loss: 5.73121976852417,          Rec Loss: 0.00319123500958
08506,   KL Loss: 1790.0089111328125
It 25100: Total Loss: 5.746094703674316,         Rec Loss: 0.00329975970089
43558,   KL Loss: 1794.62353515625
It 25200: Total Loss: 5.728720664978027,         Rec Loss: 0.00322350906208
15754,   KL Loss: 1789.2178955078125
It 25300: Total Loss: 5.741361141204834,         Rec Loss: 0.00297206337563
69352,   KL Loss: 1793.24658203125
Run Epoch 27
It 25400: Total Loss: 5.778127670288086,         Rec Loss: 0.00284880748949
94497,   KL Loss: 1804.7747802734375
It 25500: Total Loss: 5.769482135772705,         Rec Loss: 0.00323353148996
83,      KL Loss: 1801.9527587890625
It 25600: Total Loss: 5.798570156097412,         Rec Loss: 0.00299098505638
5398,    KL Loss: 1811.1185302734375
It 25700: Total Loss: 5.844773769378662,         Rec Loss: 0.00312819890677
92892,   KL Loss: 1825.514404296875
It 25800: Total Loss: 5.849793910980225,         Rec Loss: 0.00311080995015
8,       KL Loss: 1827.0885009765625
It 25900: Total Loss: 5.842171669006348,         Rec Loss: 0.00352330692112
44583,   KL Loss: 1824.57763671875
It 26000: Total Loss: 5.879512310028076,         Rec Loss: 0.00305236107669
```

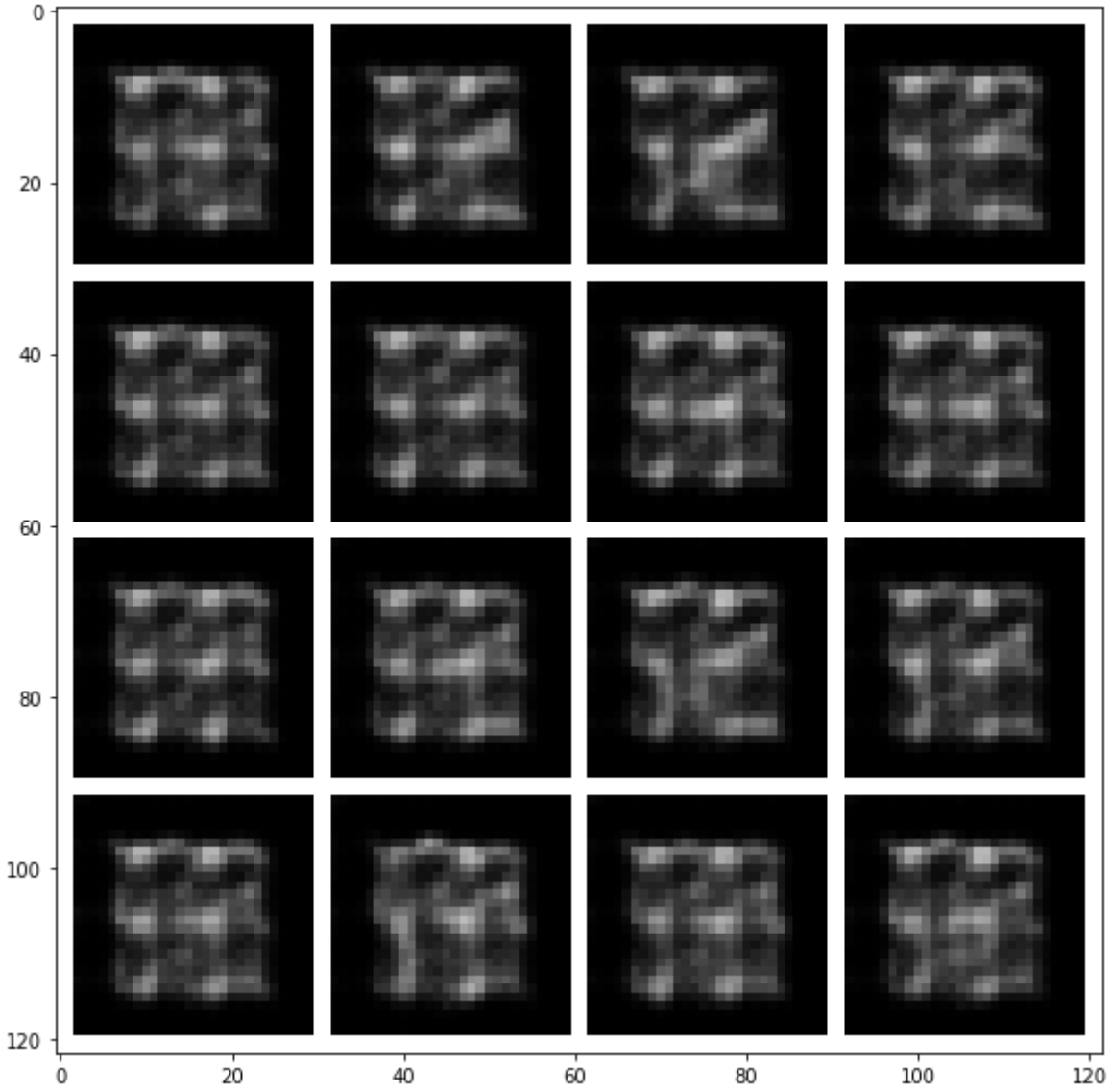
```
7707,    KL Loss: 1836.393798828125
It 26100: Total Loss: 5.864948272705078,         Rec Loss: 0.00284939259290
6952,    KL Loss: 1831.9058837890625
It 26200: Total Loss: 5.877519607543945,         Rec Loss: 0.00302530359476
80473,   KL Loss: 1835.7794189453125
Run Epoch 28
It 26300: Total Loss: 5.892045974731445,         Rec Loss: 0.00335674569942
05713,   KL Loss: 1840.21533203125
It 26400: Total Loss: 5.90395975112915,          Rec Loss: 0.00322253606282
1746,    KL Loss: 1843.98046875
It 26500: Total Loss: 5.902001857757568,         Rec Loss: 0.00301938992924
98827,   KL Loss: 1843.43212890625
It 26600: Total Loss: 5.908114433288574,         Rec Loss: 0.00308642443269
4912,    KL Loss: 1845.3212890625
It 26700: Total Loss: 5.970120429992676,         Rec Loss: 0.00384446303360
16417,   KL Loss: 1864.4613037109375
It 26800: Total Loss: 6.005579471588135,         Rec Loss: 0.00330705242231
48823,   KL Loss: 1875.710205078125
It 26900: Total Loss: 5.960150718688965,         Rec Loss: 0.00314529985189
43787,   KL Loss: 1861.564208984375
It 27000: Total Loss: 5.999385356903076,         Rec Loss: 0.00285297073423
86246,   KL Loss: 1873.91650390625
It 27100: Total Loss: 5.988559246063232,         Rec Loss: 0.00302097154781
22234,   KL Loss: 1870.4808349609375
It 27200: Total Loss: 6.058098793029785,         Rec Loss: 0.00286664580926
2991,    KL Loss: 1892.260009765625
Run Epoch 29
It 27300: Total Loss: 6.001765251159668,         Rec Loss: 0.00290407496504
4856,    KL Loss: 1874.6441650390625
It 27400: Total Loss: 6.031906604766846,         Rec Loss: 0.00294439122080
8029,    KL Loss: 1884.05078125
It 27500: Total Loss: 6.041461944580078,         Rec Loss: 0.00325835496187
2101,    KL Loss: 1886.938720703125
It 27600: Total Loss: 6.0324387550354,   Rec Loss: 0.0029912670142948627,
         KL Loss: 1884.202392578125
It 27700: Total Loss: 6.0444722175598145,        Rec Loss: 0.00308113498613
23833,   KL Loss: 1887.9346923828125
It 27800: Total Loss: 6.091704368591309,         Rec Loss: 0.00359268812462
6875,    KL Loss: 1902.5350341796875
It 27900: Total Loss: 6.084866046905518,         Rec Loss: 0.00303338211961
09056,   KL Loss: 1900.57275390625
It 28000: Total Loss: 6.114672660827637,         Rec Loss: 0.00314617576077
58045,   KL Loss: 1909.85205078125
It 28100: Total Loss: 6.101531028747559,         Rec Loss: 0.00298368977382
7791,    KL Loss: 1905.796142578125
Run Epoch 30
It 28200: Total Loss: 6.138578414916992,         Rec Loss: 0.00302832946181
2973,    KL Loss: 1917.3594970703125
It 28300: Total Loss: 6.1455864906311035,        Rec Loss: 0.00306647038087
2488,    KL Loss: 1919.53759765625
It 28400: Total Loss: 6.1561055183410645,        Rec Loss: 0.00339682307094
33556,   KL Loss: 1922.721435546875
It 28500: Total Loss: 6.136283874511719,         Rec Loss: 0.00308250333182
5137,    KL Loss: 1916.62548828125
It 28600: Total Loss: 6.202587127685547,         Rec Loss: 0.00316999573260
54573,   KL Loss: 1937.31787109375
It 28700: Total Loss: 6.182847023010254,         Rec Loss: 0.00299033219926
05925,   KL Loss: 1931.205322265625
It 28800: Total Loss: 6.226450443267822,         Rec Loss: 0.00296075944788
754,     KL Loss: 1944.840576171875
It 28900: Total Loss: 6.240612983703613,         Rec Loss: 0.00293598882853
98483,   KL Loss: 1949.274169921875
It 29000: Total Loss: 6.235678672790527,         Rec Loss: 0.00286587630398
```

```
5715,    KL Loss: 1947.7540283203125
Run Epoch 31
It 29100: Total Loss: 6.244664192199707,          Rec Loss: 0.00291323591955
00612,   KL Loss: 1950.54736328125
It 29200: Total Loss: 6.265792369842529,          Rec Loss: 0.00290708034299
31402,   KL Loss: 1957.151611328125
It 29300: Total Loss: 6.26191520690918,           Rec Loss: 0.00299979047849
77436,   KL Loss: 1955.9111328125
It 29400: Total Loss: 6.2388482093811035,         Rec Loss: 0.00285888137295
84217,   KL Loss: 1948.74658203125
It 29500: Total Loss: 6.270743370056152,          Rec Loss: 0.00304029113613
06906,   KL Loss: 1958.6572265625
It 29600: Total Loss: 6.274444103240967,          Rec Loss: 0.00265380553901
19553,   KL Loss: 1959.9345703125
It 29700: Total Loss: 6.288321018218994,          Rec Loss: 0.00346044008620
08333,   KL Loss: 1964.01904296875
It 29800: Total Loss: 6.30757474899292,           Rec Loss: 0.00320998579263
68713,   KL Loss: 1970.114013671875
It 29900: Total Loss: 6.318593502044678,          Rec Loss: 0.00256506889127
1949,    KL Loss: 1973.759033203125
It 30000: Total Loss: 6.315262794494629,          Rec Loss: 0.00257863500155
```

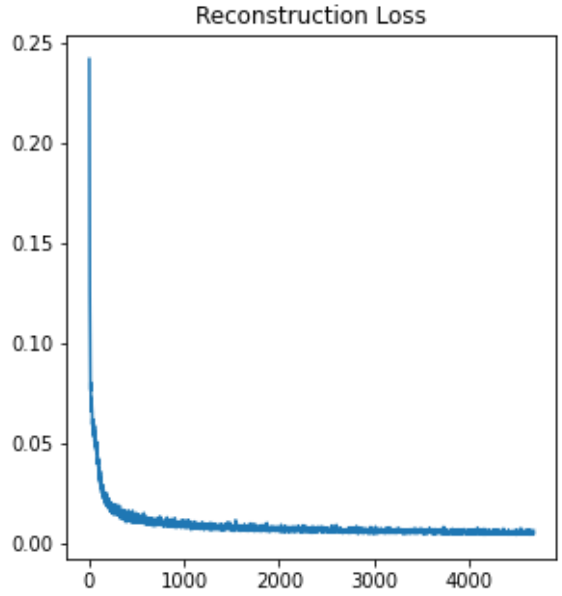


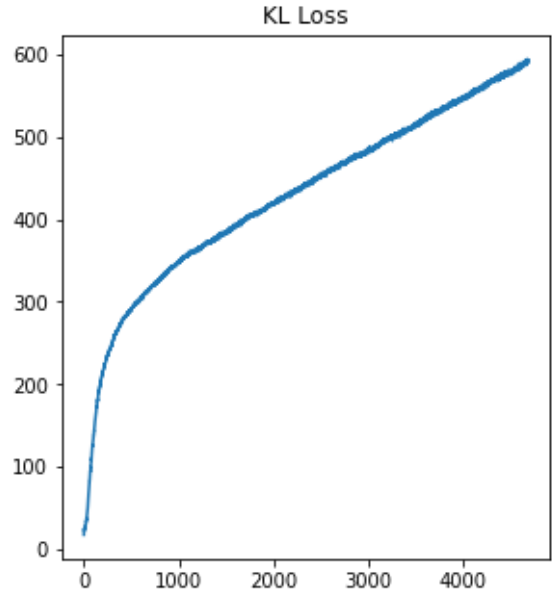
Let's look at some reconstructions and decoded embedding samples!

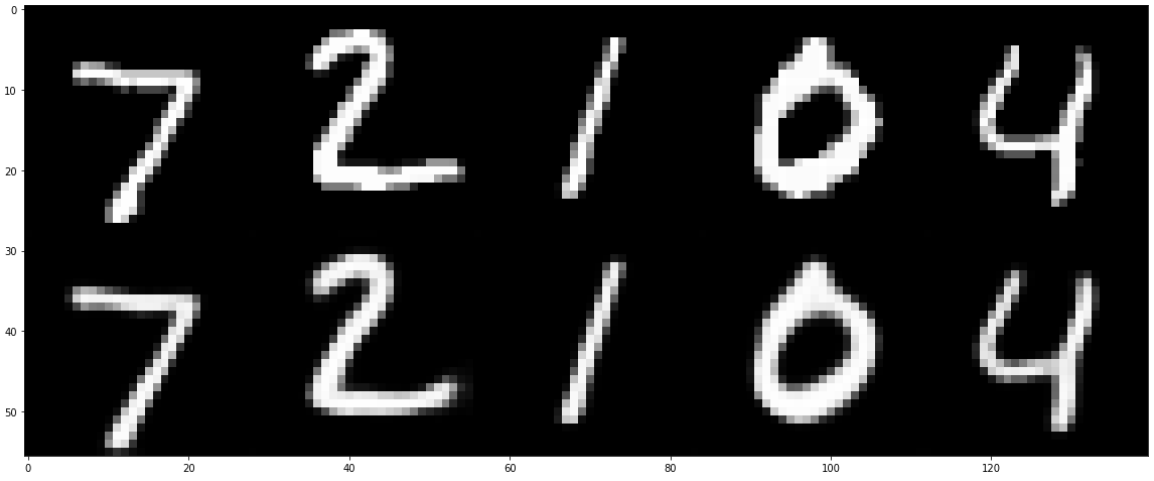
In [ ]:
```
# visualize VAE reconstructions and samples from the generative model
vis_reconstruction(vae_model)
vis_samples(vae_model)
```

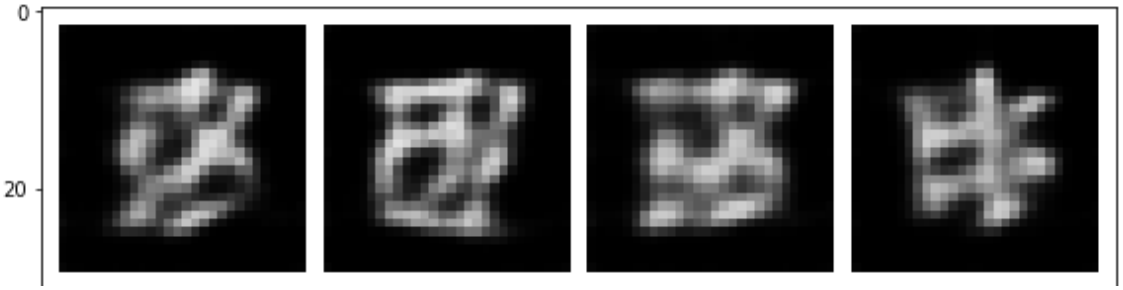


Inline Question: What can you observe when setting $\beta = 0$? Explain your observations! [3pt] \ (please limit your answer to <150 words) \ **Answer:** The model generates the original input almost precisely except some blurriness in the edges. I think it is almost similar to an autoencoder. For the generative part of the model where we sample from unit diagonal gaussian, the images generated are slightly more distinct and clear than the one generated by autoencoders. It seems as if the model is trying to generated some digits but most of them are deformed/skewed.


Reconstruction Loss
0.25
0.20
0.15
0.10
0.05
0.00
0
1000
2000
3000
4000

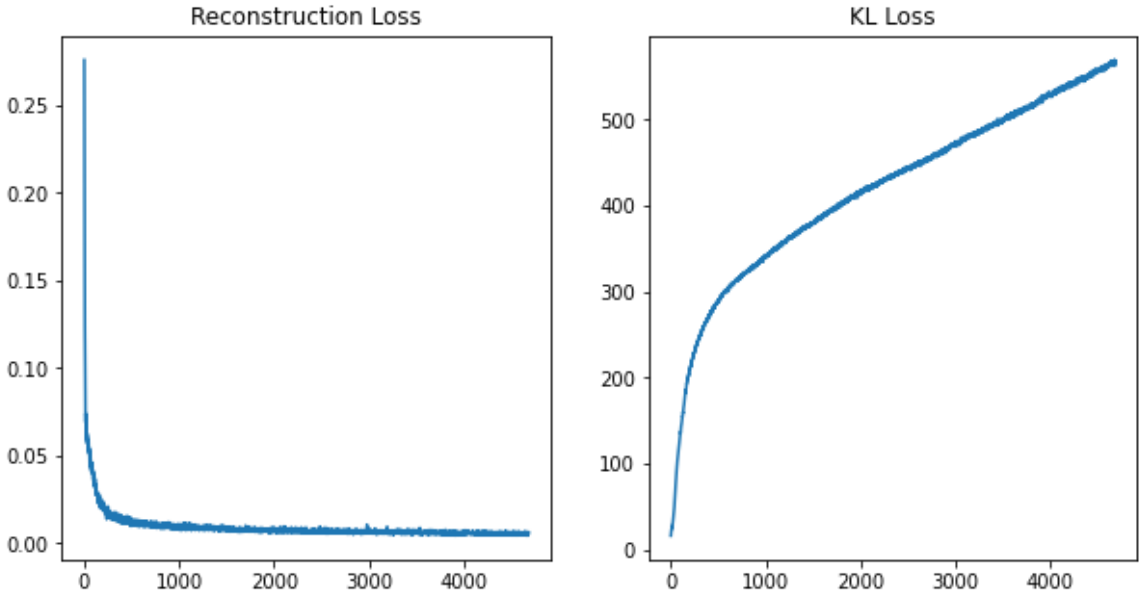

KL Loss
600
500
400
300
200
100
0
0
1000
2000
3000
4000

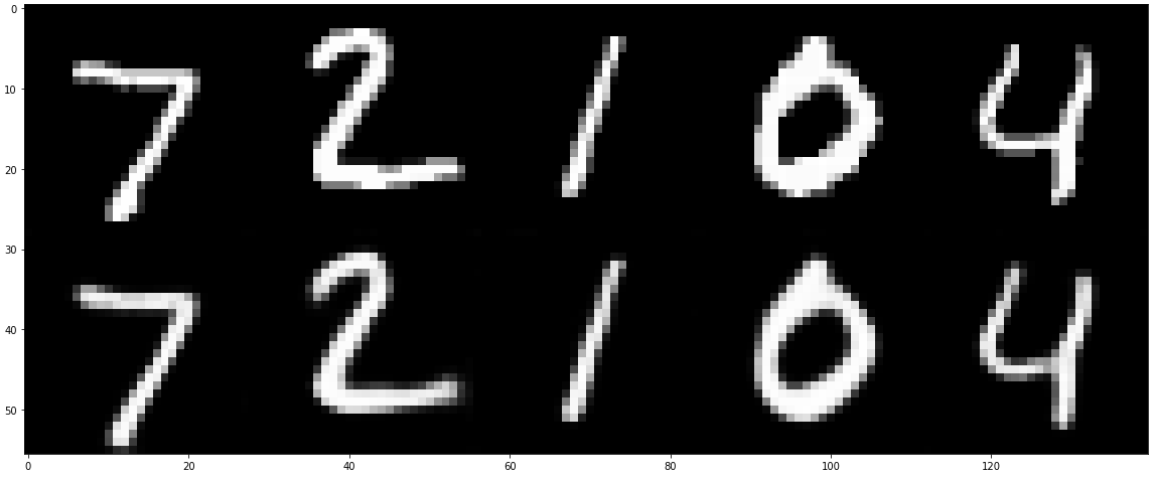

0
10
20
30
40
50
0
20
40
60
80
100
120

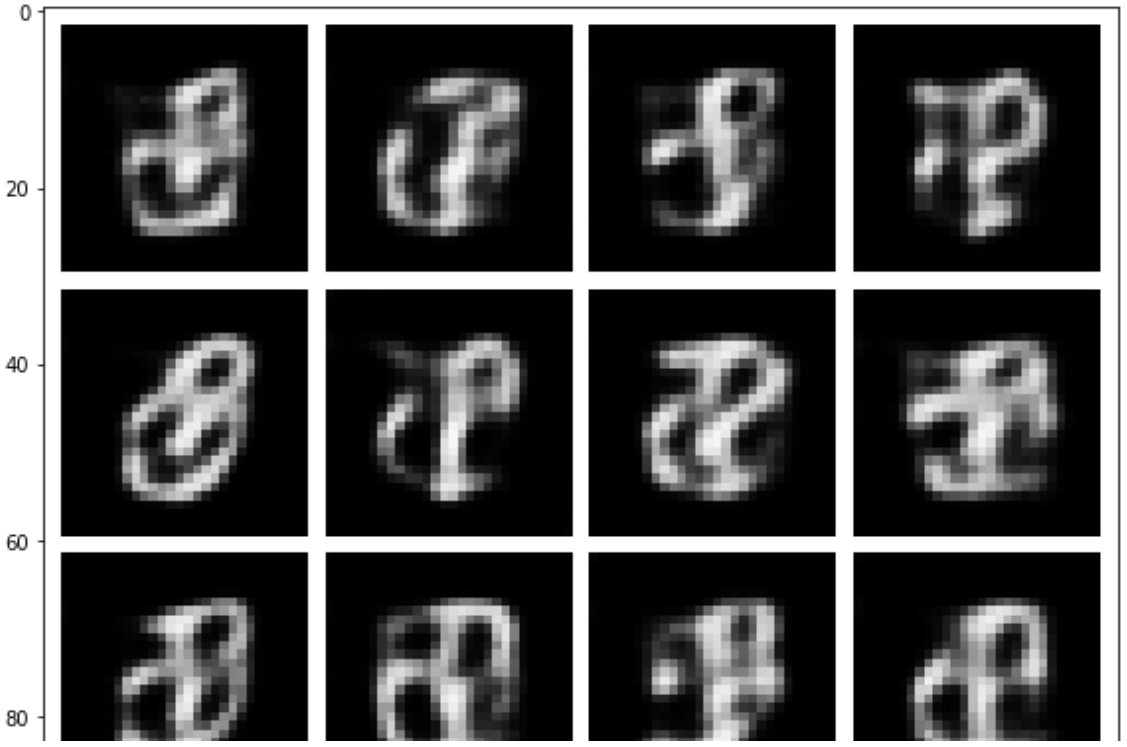


Let's repeat the same experiment for $\beta = 10$, a very high value for the coefficient. You can modify the β value in the cell above and rerun it (it is okay to overwrite the outputs of the previous experiment, but **make sure to copy the visualizations of training curves, reconstructions and samples for $\beta = 0$ into your solution PDF** before deleting them).

> **Inline Question: What can you observe when setting $\beta = 10$? Explain your observations! [3pt]** \ (please limit your answer to <200 words) \
> **Answer**: It seems like at `beta=10` , the digits in the reconstructed image are getting a little blurry. While the overall digits are almost the same, we can see additional pixels around the digits. Moroever, the samples of generation are also very blurry, resulting into almost undistinguishable digits.It seems that the digits that are generated are getting skewed in the space and not readable in many of the cases. It is like one digit is morphing into the next digit that is similar to it.







Now we can start tuning the beta value to achieve a good result. First describe what a "good result" would look like (focus what you would expect for reconstructions and sample quality).

> **Inline Question: Characterize what properties you would expect for reconstructions (1pt) and samples (2pt) of a well-tuned VAE! [3pt]** \
> (please limit your answer to <200 words) \ **Answer**: A well-tuned VAE should have sharper reconstructions and should have almost negligible blur around the digits. For the samples that are generated by a VAE, they should be more distinct and clear with respect to what digit they represent. Moreover, the skewness of the digits should not be too much that they are unreadable or unbelievable that a human would have written it.

Tuning the β-factor [5pt]

Now that you know what outcome we would like to obtain, try to tune β to achieve this result.

(logarithmic search in steps of 10x will be helpful, good results can be achieved after ~20 epochs of training). It is again okay to overwrite the results of the previous $\beta=10$ experiment after copying them to the solution PDF.

Your final notebook should include the visualizations of your best tuned VAE

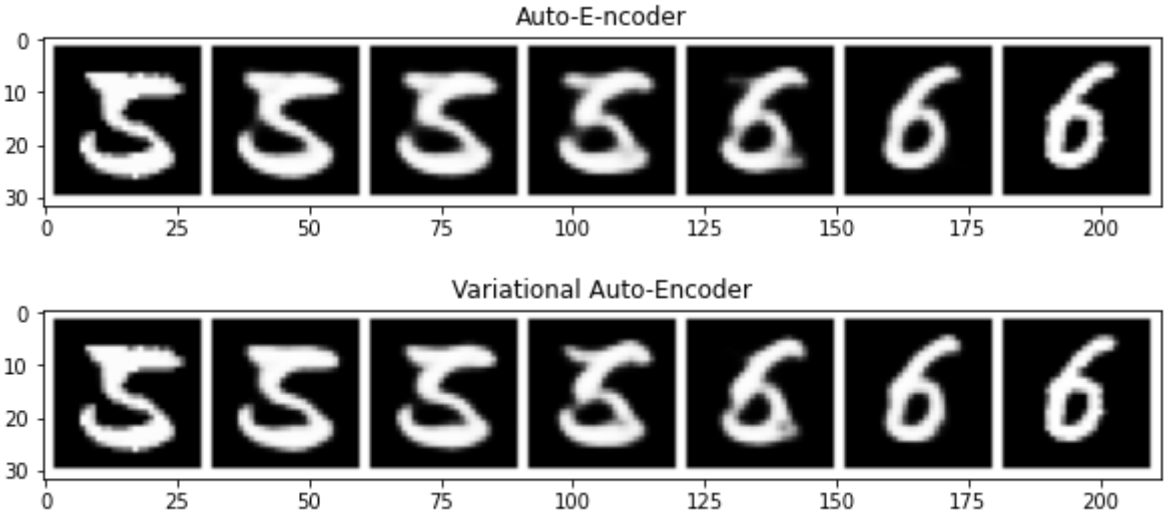
4. Embedding Space Interpolation [3pt]

As mentioned in the introduction, AEs and VAEs cannot only be used to generate images, but also to learn low-dimensional representations of their inputs. In this final section we will investigate the representations we learned with both models by **interpolating in embedding space** between different images. We will encode two images into their low-dimensional embedding representations, then interpolate these embeddings and reconstruct the result.

In [ ]:
```
START_LABEL = 5
END_LABEL = 6
nz=64

def get_image_with_label(target_label):
  """Returns a random image from the training set with the requested digit.
  for img_batch, label_batch in mnist_data_loader:
    for img, label in zip(img_batch, label_batch):
      if label == target_label:
        return img.to(device)

def interpolate_and_visualize(model, tag, start_img, end_img):
  """Encodes images and performs interpolation. Displays decodings."""
  model.eval()    # put model in eval mode to avoid updating batchnorm

  # encode both images into embeddings (use posterior mean for interpolatio
  z_start = model.encoder(start_img[None])[..., :nz]
  z_end = model.encoder(end_img[None])[..., :nz]

  # compute interpolated latents
  N_INTER_STEPS = 5
  z_inter = [z_start + i/N_INTER_STEPS * (z_end - z_start) for i in range(N

  # decode interpolated embeddings (as a single batch)
  img_inter = model.decoder(torch.cat(z_inter))

  # reshape result and display interpolation
  vis_imgs = torch.cat([start_img[None], img_inter, end_img[None]])
  fig = plt.figure(figsize = (10, 10))
  ax1 = plt.subplot(111)
  ax1.imshow(torchvision.utils.make_grid(vis_imgs, nrow=N_INTER_STEPS+2, pa
                  .data.cpu().numpy().transpose(1, 2, 0), cmap='gray')
  plt.title(tag)
  plt.show()


# sample two training images with given labels
start_img = get_image_with_label(START_LABEL)
end_img = get_image_with_label(END_LABEL)

# visualize interpolations for AE and VAE models
interpolate_and_visualize(ae_model, "Auto-E-ncoder", start_img, end_img)
interpolate_and_visualize(vae_model, "Variational Auto-Encoder", start_img,
```

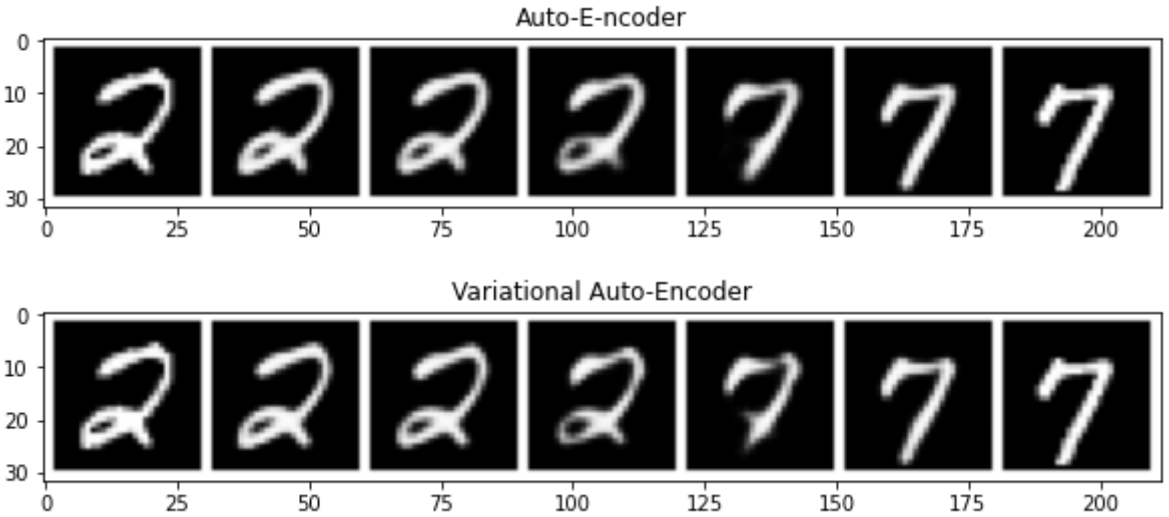
```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481:
```

```
UserWarning: This DataLoader will create 4 worker processes in total. Our s
uggested max number of worker in current system is 2, which is smaller than
what this DataLoader is going to create. Please be aware that excessive wor
ker creation might get DataLoader running slow or even freeze, lower the wo
rker number to avoid potential slowness/freeze if necessary.
  cpuset_checked))
```



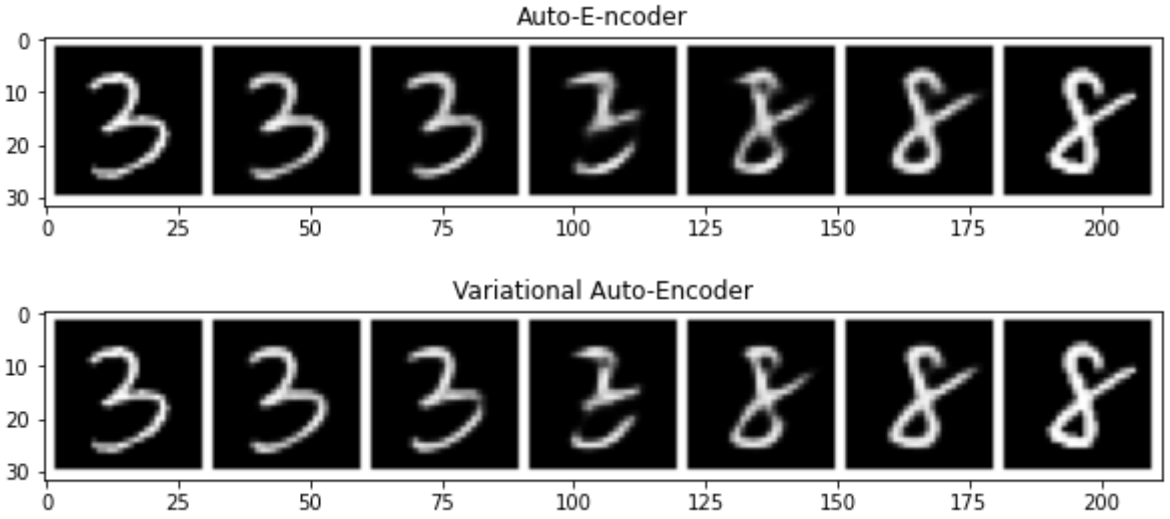
In [ ]:
```
# sample two training images with given labels
start_img = get_image_with_label(2)
end_img = get_image_with_label(7)

# visualize interpolations for AE and VAE models
interpolate_and_visualize(ae_model, "Auto-E-ncoder", start_img, end_img)
interpolate_and_visualize(vae_model, "Variational Auto-Encoder", start_img,
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481:
UserWarning: This DataLoader will create 4 worker processes in total. Our s
uggested max number of worker in current system is 2, which is smaller than
what this DataLoader is going to create. Please be aware that excessive wor
ker creation might get DataLoader running slow or even freeze, lower the wo
rker number to avoid potential slowness/freeze if necessary.
  cpuset_checked))
```



In [ ]:
```
# sample two training images with given labels
start_img = get_image_with_label(3)
end_img = get_image_with_label(8)

# visualize interpolations for AE and VAE models
interpolate_and_visualize(ae_model, "Auto-E-ncoder", start_img, end_img)
interpolate_and_visualize(vae_model, "Variational Auto-Encoder", start_img,
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481:
UserWarning: This DataLoader will create 4 worker processes in total. Our s
uggested max number of worker in current system is 2, which is smaller than
what this DataLoader is going to create. Please be aware that excessive wor
ker creation might get DataLoader running slow or even freeze, lower the wo
rker number to avoid potential slowness/freeze if necessary.
  cpuset_checked))
```



Repeat the experiment for different start / end labels and different samples. Describe your observations.

> **Inline Question: Repeat the interpolation experiment with different start / end labels and multiple samples. Describe your observations! Focus on**: \
>
> 1. **How do AE and VAE embedding space interpolations differ?** \
> 2. **How do you expect these differences to affect the usefulness of the learned representation for downstream learning?** \ (please limit your answer to <300 words)
>
> **Answer**: The interpolations of AE have more randomness compared to the embedding space interpolations from VAE. Comparing both the visualizations we can see that the latent space of VAE has more inclination to the MNIST like characters while AE is very blurry. Using AE and VAE in downstream learning will provide drastically different results because a random value in case of VAE generates more meaningful output than AE in somecases where is feels like AE is trying to morph one character into another or the latent space is overlapping.

Submission PDF

As in assignment 1, please prepare a separate submission PDF for each problem. **Do not simply render your notebook as a pdf**. For this problem, please include the following plots & answers in a PDF called `problem_1_solution.pdf` :

1. Auto-encoder samples and AE sampling inline question answer.
2. VAE training curves, reconstructions and samples for:
    - $\beta = 0$
    - $\beta = 10$
    - your tuned β (also listing the tuned value for β)
3. Answers to all inline questions in VAE section (ie 4 inline questions).
4. Three representative interpolation comparisons that show AE and VAE embedding interpolation between the same images.
5. Answer to interpolation inline question.

Note that you still need to submit the jupyter notebook with all generated solutions. We will randomly pick submissions and check that the plots in the PDF and in the notebook are equivalent (except for those $\beta=0\;/\;10$ plots that we allowed to overwrite).