Answer 1

Firstly I tried keeping increasing the latent space to 4 dimensions and then also decreasing the filter sizes for each convolution layer. The results weren't very satisfactory, they were blurry and some of them were decoded wrongly.

Imports

```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
```

Import utility functions

```
from helper_data import get_dataloaders_mnist
from helper_train import train_autoencoder_v1
from helper_utils import set_deterministic, set_all_seeds
from helper_plotting import plot_training_loss
from helper_plotting import plot_generated_images
from helper_plotting import plot_latent_space_with_labels
```

```
##########################
### SETTINGS
##########################

# Device
CUDA_DEVICE_NUM = 3
NUM_CLASSES = 10
DEVICE = torch.device(f'cuda:{0}' if torch.cuda.is_available() else 'cpu')
print('Device:', DEVICE)

# Hyperparameters
RANDOM_SEED = 123
LEARNING_RATE = 0.0005
BATCH_SIZE = 32
NUM_EPOCHS = 10
```

```
Device: cuda:0
```

```
set_deterministic
set_all_seeds(RANDOM_SEED)
```

Dataset

```
##########################
### Dataset
##########################

train_loader, valid_loader, test_loader = get_dataloaders_mnist(
    batch_size=BATCH_SIZE,
    num_workers=2,
    validation_fraction=0.)
```

```
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
100%  9912422/9912422 [00:00<00:00, 53996188.70it/s]
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
100%  28881/28881 [00:00<00:00, 608905.67it/s]
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
100%  1648877/1648877 [00:00<00:00, 21106334.13it/s]
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
100%  4542/4542 [00:00<00:00, 154347.78it/s]
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
```

```
# Checking the dataset
print('Training Set:\n')
for images, labels in train_loader:
    print('Image batch dimensions:', images.size())
    print('Image label dimensions:', labels.size())
    print(labels[:10])
    break

# Checking the dataset
print('\nValidation Set:')
for images, labels in valid_loader:
    print('Image batch dimensions:', images.size())
    print('Image label dimensions:', labels.size())
    print(labels[:10])
    break

# Checking the dataset
print('\nTesting Set:')
for images, labels in test_loader:
    print('Image batch dimensions:', images.size())
    print('Image label dimensions:', labels.size())
    print(labels[:10])
    break
```

```
Training Set:

Image batch dimensions: torch.Size([32, 1, 28, 28])
Image label dimensions: torch.Size([32])
tensor([1, 2, 1, 9, 0, 6, 9, 8, 0, 1])

Validation Set:

Testing Set:
Image batch dimensions: torch.Size([32, 1, 28, 28])
Image label dimensions: torch.Size([32])
tensor([7, 2, 1, 0, 4, 1, 4, 9, 5, 9])
```

Model

```
##########################
### MODEL
##########################


class Reshape(nn.Module):
    def __init__(self, *args):
        super().__init__()
        self.shape = args

    def forward(self, x):
        return x.view(self.shape)
```

```
class Trim(nn.Module):
    def __init__(self, *args):
        super().__init__()

    def forward(self, x):
        return x[:, :, :28, :28]


class AutoEncoder(nn.Module):
    def __init__(self):
        super().__init__()

        self.encoder = nn.Sequential( #784
                nn.Conv2d(1, 16, stride=(1, 1), kernel_size=(3, 3), padding=1),
                nn.LeakyReLU(0.01),
                nn.Conv2d(16, 32, stride=(2, 2), kernel_size=(3, 3), padding=1),
                nn.LeakyReLU(0.01),
                nn.Conv2d(32, 64, stride=(2, 2), kernel_size=(3, 3), padding=1),
                nn.LeakyReLU(0.01),
                nn.Conv2d(64, 64, stride=(1, 1), kernel_size=(3, 3), padding=1),
                nn.Flatten(),
                nn.Linear(64 * 7 * 7, 4)
        )
        self.decoder = nn.Sequential(
                torch.nn.Linear(4, 64 * 7 * 7),
                Reshape(-1, 64, 7, 7),
                nn.ConvTranspose2d(64, 64, stride=(1, 1), kernel_size=(3, 3), padding=1),
                nn.LeakyReLU(0.01),
                nn.ConvTranspose2d(64, 32, stride=(2, 2), kernel_size=(3, 3), padding=1),
                nn.LeakyReLU(0.01),
                nn.ConvTranspose2d(32, 16, stride=(2, 2), kernel_size=(3, 3), padding=0),
                nn.LeakyReLU(0.01),
                nn.ConvTranspose2d(16, 1, stride=(1, 1), kernel_size=(3, 3), padding=0),
                Trim(),  # 1x29x29 -> 1x28x28
                nn.Sigmoid()
                )

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


set_all_seeds(RANDOM_SEED)

model = AutoEncoder()
model.to(DEVICE)

optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

```
sum(p.numel() for p in model.parameters() if p.requires_grad)
```

```
148613
```

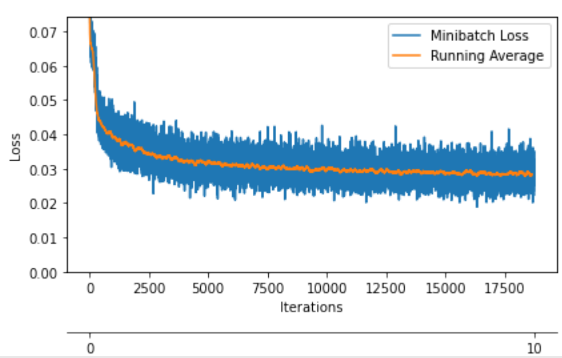
Training

```
log_dict = train_autoencoder_v1(num_epochs=NUM_EPOCHS, model=model,
                                optimizer=optimizer, device=DEVICE,
                                train_loader=train_loader,
                                skip_epoch_stats=True,
                                logging_interval=250)
```

```
Epoch: 006/010 | Batch 0000/1875 | Loss: 0.0274
Epoch: 006/010 | Batch 0250/1875 | Loss: 0.0317
Epoch: 006/010 | Batch 0500/1875 | Loss: 0.0326
Epoch: 006/010 | Batch 0750/1875 | Loss: 0.0283
Epoch: 006/010 | Batch 1000/1875 | Loss: 0.0314
Epoch: 006/010 | Batch 1250/1875 | Loss: 0.0285
Epoch: 006/010 | Batch 1500/1875 | Loss: 0.0317
Epoch: 006/010 | Batch 1750/1875 | Loss: 0.0282
Time elapsed: 4.48 min
Epoch: 007/010 | Batch 0000/1875 | Loss: 0.0291
Epoch: 007/010 | Batch 0250/1875 | Loss: 0.0345
Epoch: 007/010 | Batch 0500/1875 | Loss: 0.0228
Epoch: 007/010 | Batch 0750/1875 | Loss: 0.0300
Epoch: 007/010 | Batch 1000/1875 | Loss: 0.0351
Epoch: 007/010 | Batch 1250/1875 | Loss: 0.0276
Epoch: 007/010 | Batch 1500/1875 | Loss: 0.0289
Epoch: 007/010 | Batch 1750/1875 | Loss: 0.0290
Time elapsed: 5.21 min
Epoch: 008/010 | Batch 0000/1875 | Loss: 0.0255
Epoch: 008/010 | Batch 0250/1875 | Loss: 0.0292
Epoch: 008/010 | Batch 0500/1875 | Loss: 0.0273
Epoch: 008/010 | Batch 0750/1875 | Loss: 0.0284
Epoch: 008/010 | Batch 1000/1875 | Loss: 0.0281
Epoch: 008/010 | Batch 1250/1875 | Loss: 0.0268
Epoch: 008/010 | Batch 1500/1875 | Loss: 0.0287
Epoch: 008/010 | Batch 1750/1875 | Loss: 0.0277
Time elapsed: 5.94 min
Epoch: 009/010 | Batch 0000/1875 | Loss: 0.0282
Epoch: 009/010 | Batch 0250/1875 | Loss: 0.0267
Epoch: 009/010 | Batch 0500/1875 | Loss: 0.0314
Epoch: 009/010 | Batch 0750/1875 | Loss: 0.0304
Epoch: 009/010 | Batch 1000/1875 | Loss: 0.0263
Epoch: 009/010 | Batch 1250/1875 | Loss: 0.0246
Epoch: 009/010 | Batch 1500/1875 | Loss: 0.0291
Epoch: 009/010 | Batch 1750/1875 | Loss: 0.0287
Time elapsed: 6.68 min
Epoch: 010/010 | Batch 0000/1875 | Loss: 0.0265
Epoch: 010/010 | Batch 0250/1875 | Loss: 0.0303
Epoch: 010/010 | Batch 0500/1875 | Loss: 0.0244
Epoch: 010/010 | Batch 0750/1875 | Loss: 0.0271
Epoch: 010/010 | Batch 1000/1875 | Loss: 0.0262
Epoch: 010/010 | Batch 1250/1875 | Loss: 0.0316
Epoch: 010/010 | Batch 1500/1875 | Loss: 0.0280
Epoch: 010/010 | Batch 1750/1875 | Loss: 0.0263
Time elapsed: 7.45 min
Total Training Time: 7.45 min
```

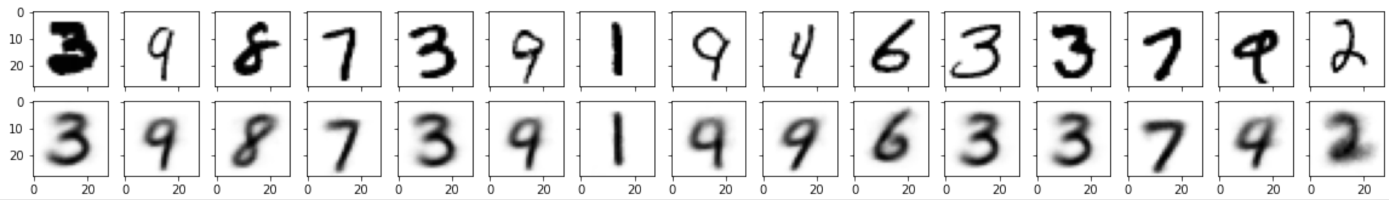
Evaluation

```
plot_training_loss(log_dict['train_loss_per_batch'], NUM_EPOCHS)
plt.show()
```

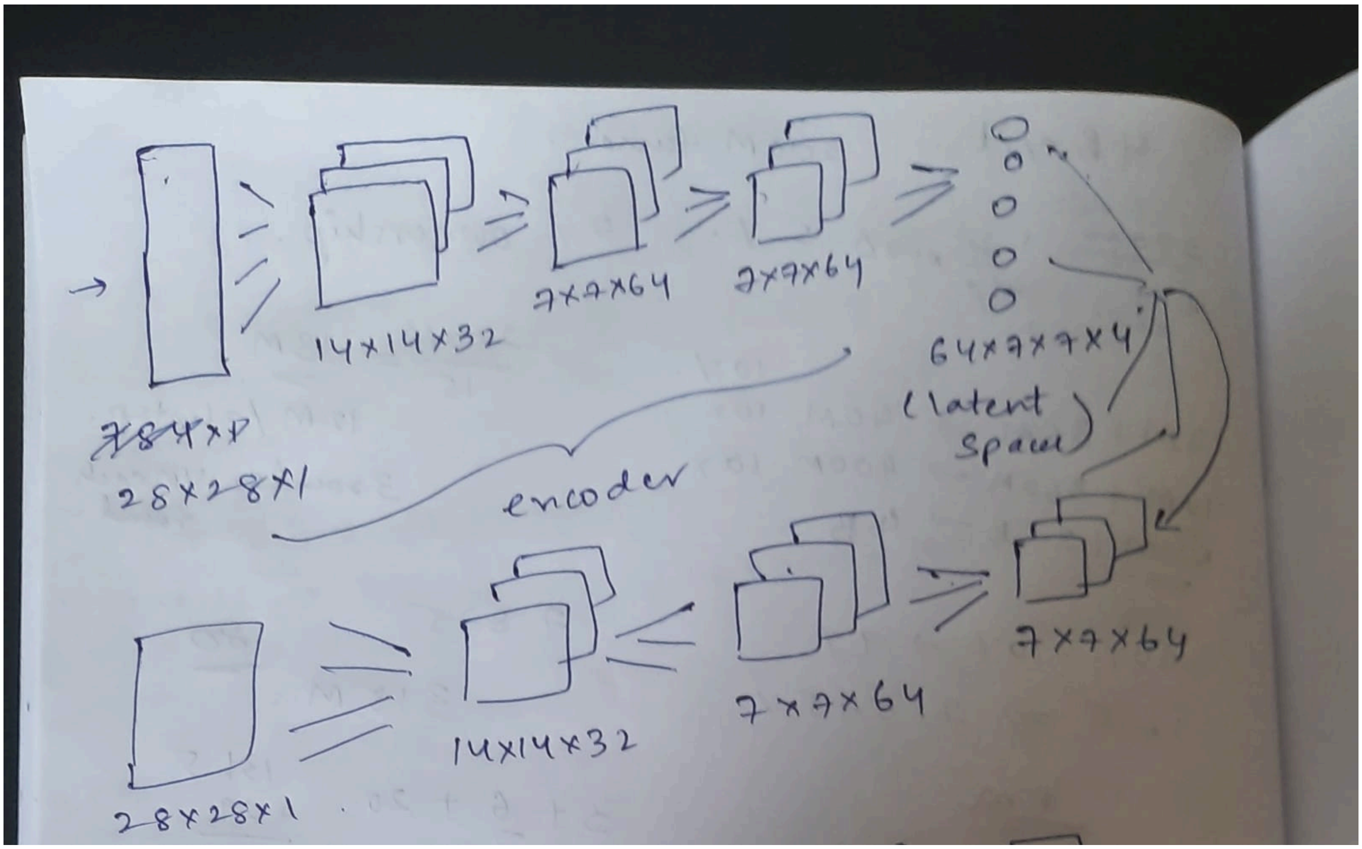


Original plotting with the given model in lecture

```
plot_generated_images(data_loader=train_loader, model=model, device=DEVICE)
```

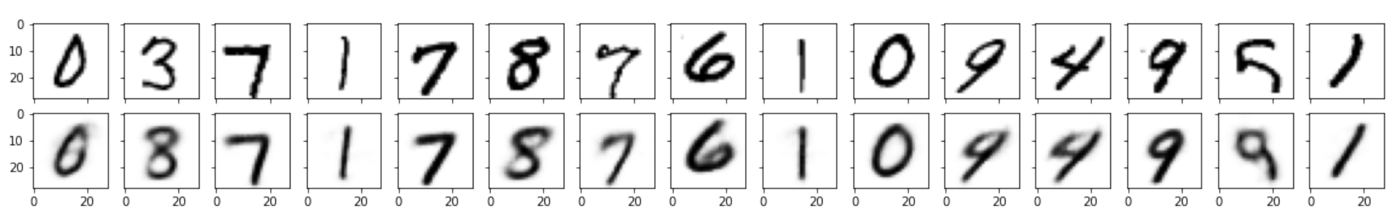


This is the pipeline, for this model, it has the encoder, decoder and the latent space in between



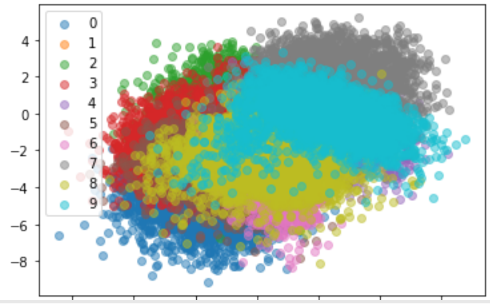
This model with 4 dim latent space

```
plot_generated_images(data_loader=train_loader, model=model, device=DEVICE)
```



```
plot_latent_space_with_labels(
    num_classes=NUM_CLASSES,
    data_loader=train_loader,
    model=model,
    device=DEVICE)

plt.legend()
plt.show()
```



Decoding one particular image and a new image (Not part of exercise)

```
sample_img = None
sample_labels = None
for images, labels in train_loader:
    sample_img = images
    sample_labels = labels
    break
```

```
sample_img = sample_img.to(DEVICE)
encoded_sample =  model.encoder(sample_img)
```

```
len(encoded_sample)
```

```
32
```

```
len(encoded_sample[7])
```

```
4
```

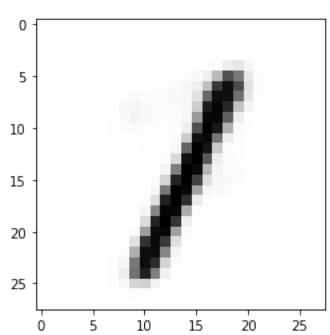
```
sample_labels[7]
```

```
tensor(1)
```

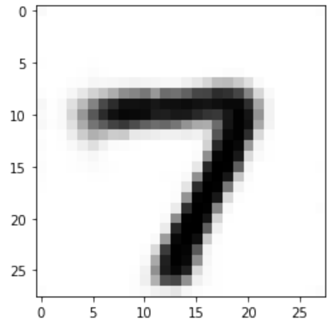
```
encoded_sample[7]
```

```
tensor([ 2.7196,  0.3216,  1.3547, -1.8533], device='cuda:0',
       grad_fn=<SelectBackward0>)
```

```
with torch.no_grad():
    new_image = model.decoder(encoded_sample[7].to(DEVICE))
    new_image.squeeze_(0)
    new_image.squeeze_(0)
plt.imshow(new_image.to('cpu').numpy(), cmap='binary')
plt.show()
```




```
with torch.no_grad():
    new_image = model.decoder(torch.tensor([3.7196,  2.3216,  -1.3547, 1.8533]).to(DEVICE))
    new_image.squeeze_(0)
    new_image.squeeze_(0)
plt.imshow(new_image.to('cpu').numpy(), cmap='binary')
plt.show()
```



Start coding or generate with AI.