



limhpone / computervision-final-prep

[Code](#)[Issues](#)[Pull requests](#)[Actions](#)[Projects](#)[Wiki](#)[Security](#)[In](#)

## [computervision-final-prep / lab / Lab 11 \(Generative Network\)-20251128 / AE-VAE.ipynb](#)



limhpone lab 11-GAN

0a969e1 · 2 hours ago



1241 lines (1241 loc) · 950 KB

[Preview](#)[Code](#)[Blame](#)

Raw



In [21]:

```
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

from torchvision.utils import make_grid
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchvision import datasets

import torch.nn as nn
import torch.nn.functional as F
import torch
```

In [22]:

```
device = 'cuda:2' if torch.cuda.is_available() else 'cpu'
print(device)

# Hyperparameter
lr = 2e-4
BATCH_SIZE = 256
NUM_EPOCHS = 50

logging_interval = 50
# save_model = True
```

cuda:2

In [23]:

```
import os
```

In [24]:

```
trf = transforms.Compose([
    transforms.ToTensor(),
])

train_dataset = datasets.MNIST(root='.',
                               train=True,
                               transform=trf,
                               download=True)

test_dataset = datasets.MNIST(root='.',
                             train=False,
                             transform=trf)
```

In [25]:

```
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=BATCH_SIZE,
                          num_workers=2,
                          shuffle=True)

test_loader = DataLoader(dataset=test_dataset,
                        batch_size=BATCH_SIZE,
                        num_workers=2,
                        shuffle=False)
```

```
print('Training Set:\n')
for images, labels in train_loader:
    print('Image batch dimensions:', images.size())
    print('Image label dimensions:', labels.size())
    break
```

Training Set:

```
Image batch dimensions: torch.Size([256, 1, 28, 28])
Image label dimensions: torch.Size([256])
```

In [27]:

```
plt.figure(figsize=(8, 8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(make_grid(images[:64],
                                  padding=2, normalize=True),
                      (1, 2, 0)))
plt.show()
```

Training Images



In [7]:

```
## Utility function to Reshape and Trim the images
```

```

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]

```

## Define a model

In [8]:

```

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

        self.encoder = nn.Sequential(
            nn.Conv2d(1, 32, stride=2, kernel_size=3, bias=False, padding=1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
            #
            nn.Conv2d(32, 64, stride=2, kernel_size=3, bias=False, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
            #
            nn.Conv2d(64, 64, stride=1, kernel_size=3, bias=False, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.1, inplace=True),
            #
            nn.Flatten(),
        )

        self.z_mean = torch.nn.Linear(3136, 2) # 7x7x64
        self.z_log_var = torch.nn.Linear(3136, 2)

        self.decoder = nn.Sequential(
            torch.nn.Linear(2, 3136),
            Reshape(-1, 64, 7, 7),
            #
            nn.ConvTranspose2d(64, 64, stride=2, kernel_size=3),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
            #
            nn.ConvTranspose2d(64, 64, stride=2, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Dropout2d(0.25),
            ""
        )

```

```

        #
        nn.ConvTranspose2d(64, 32, stride=2, kernel_size=3, padding=1),
        nn.BatchNorm2d(32),
        nn.LeakyReLU(0.1, inplace=True),
        nn.Dropout2d(0.25),
        #
        nn.ConvTranspose2d(32, 1, stride=2, kernel_size=3, padding=1),
        #
        Trim(), # 1x29x29 -> 1x28x28
        nn.Sigmoid()
    )

# Instead of sampling directly from N(z_mu, exp(z_log_var)), we sample from
# z = mu + sigma * epsilon; sigma = e^(0.5 * log(sigma^2))
def reparameterize(self, z_mu, z_log_var):
    eps = torch.randn(z_mu.size(0), z_mu.size(1)).to(z_mu.get_device())
    z = z_mu + eps * torch.exp(z_log_var/2.)
    return z

# Why reparameterize is necessary? - This allows backpropagation through stochastic sampling

# encoding_fn is used for visualizing our latent space
def encoding_fn(self, x):
    x = self.encoder(x)
    z_mean, z_log_var = self.z_mean(x), self.z_log_var(x)
    encoded = self.reparameterize(z_mean, z_log_var)
    return encoded

def forward(self, x):
    x = self.encoder(x)
    z_mean, z_log_var = self.z_mean(x), self.z_log_var(x)
    encoded = self.reparameterize(z_mean, z_log_var)
    decoded = self.decoder(encoded)
    return encoded, z_mean, z_log_var, decoded

```

In [9]:

```
model = VAE()
model.to(device)
```

Out[9]:

```
VAE(
  (encoder): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1, inplace=True)
    (3): Dropout2d(p=0.25, inplace=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): LeakyReLU(negative_slope=0.1, inplace=True)
    (7): Dropout2d(p=0.25, inplace=False)
    (8): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.1, inplace=True)
    (11): Flatten(start_dim=1, end_dim=-1)
  )
)
```

```

        )
        (z_mean): Linear(in_features=3136, out_features=2, bias=True)
        (z_log_var): Linear(in_features=3136, out_features=2, bias=True)
        (decoder): Sequential(
            (0): Linear(in_features=2, out_features=3136, bias=True)
            (1): Reshape()
            (2): ConvTranspose2d(64, 64, kernel_size=(3, 3), stride=(2, 2))
            (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ts=True)
            (4): LeakyReLU(negative_slope=0.1, inplace=True)
            (5): Dropout2d(p=0.25, inplace=False)
            (6): ConvTranspose2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
            (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ts=True)
            (8): LeakyReLU(negative_slope=0.1, inplace=True)
            (9): Dropout2d(p=0.25, inplace=False)
            (10): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
            (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ts=True)
            (12): LeakyReLU(negative_slope=0.1, inplace=True)
            (13): Dropout2d(p=0.25, inplace=False)
            (14): ConvTranspose2d(32, 1, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
            (15): Trim()
            (16): Sigmoid()
        )
    )
)

```

In [10]:

```

# Total num. of parameters
num_params = sum(p.numel() for p in model.parameters())
# Total num. of "trainable" parameters
num_trainable_params = sum(p.numel() for p in model.parameters() if p.requires_g
print(f'Total num. of parameters: {num_params}')
print(f'Total num. of Trainable parameters: {num_trainable_params}')

```



Total num. of parameters: 170789  
Total num. of Trainable parameters: 170789

In [11]:

```
loss_fn = F.mse_loss # evaluation metrics
```

In [12]:

```

def compute_epoch_loss_autoencoder(model, data_loader, loss_fn, device):
    model.eval()
    curr_loss, num_examples = 0., 0
    with torch.no_grad():
        for features, _ in data_loader:
            features = features.to(device)
            _, _, _, logits = model(features)
            loss = loss_fn(logits, features, reduction='sum')
            num_examples += features.size(0)
            curr_loss += loss

    curr_loss = curr_loss / num_examples
    return curr_loss

```

In [13]:

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

## Training loop

In [14]:

```
log_dict = {'train_combined_loss_per_batch': [],
            'train_combined_loss_per_epoch': [],
            'train_reconstruction_loss_per_batch': [],
            'train_kl_loss_per_batch': []}

skip_epoch_stats = False

start_time = time.time()
for epoch in range(NUM_EPOCHS):
    model.train()
    for batch_idx, (features, _) in enumerate(train_loader):

        features = features.to(device)

        # FORWARD AND BACK PROP
        encoded, z_mean, z_log_var, decoded = model(features)

        # total Loss = reconstruction loss + KL divergence

        kl_div = -0.5 * torch.sum(1 + z_log_var
                                  - z_mean**2
                                  - torch.exp(z_log_var),
                                  axis=1) # sum over latent dimension

        batchsize = kl_div.size(0)
        kl_div = kl_div.mean() # average over batch dimension

        pixelwise = loss_fn(decoded, features, reduction='none')
        pixelwise = pixelwise.view(batchsize, -1).sum(axis=1) # sum over pixels
        pixelwise = pixelwise.mean() # average over batch dimension

        loss = pixelwise + kl_div

        optimizer.zero_grad()

        loss.backward()
        # UPDATE MODEL PARAMETERS
        optimizer.step()

        # LOGGING
        log_dict['train_combined_loss_per_batch'].append(loss.item())
        log_dict['train_reconstruction_loss_per_batch'].append(pixelwise.item())
        log_dict['train_kl_loss_per_batch'].append(kl_div.item())

        if not batch_idx % logging_interval:
            print('Epoch: %03d/%03d | Batch %04d/%04d | Loss: %.4f'
                  % (epoch+1, NUM_EPOCHS, batch_idx,
                     len(train_loader), loss))

        if not skip_epoch_stats:
            model.eval()
```

```

        with torch.set_grad_enabled(False): # save memory during inference
            train_loss = compute_epoch_loss_autoencoder(
                model, train_loader, loss_fn, device)
            print('***Epoch: %03d/%03d | Loss: %.3f' % (
                epoch+1, NUM_EPOCHS, train_loss))
            log_dict['train_combined_loss_per_epoch'].append(train_loss.item())

        print('Time elapsed: %.2f min' % ((time.time() - start_time)/60))

    print('Total Training Time: %.2f min' % ((time.time() - start_time)/60))
    # if save_model is not None:
    #     torch.save({
    #         'model_state_dict': model.state_dict(),
    #         'optimizer_state_dict': optimizer_D.state_dict()
    #     }, 'vae_mnist.pt')

```

```

/tmp/ipykernel_1818094/595489086.py:47: UserWarning: Converting a tensor with requires_grad=True to a scalar may lead to unexpected behavior.
Consider using tensor.detach() first. (Triggered internally at /pytorch/torch/csrc/autograd/generated/python_variable_methods.cpp:835.)
    print('Epoch: %03d/%03d | Batch %04d/%04d | Loss: %.4f'
Epoch: 001/050 | Batch 0000/0235 | Loss: 255.0624
Epoch: 001/050 | Batch 0050/0235 | Loss: 84.3248
Epoch: 001/050 | Batch 0100/0235 | Loss: 66.1473
Epoch: 001/050 | Batch 0150/0235 | Loss: 59.4644
Epoch: 001/050 | Batch 0200/0235 | Loss: 55.9294
***Epoch: 001/050 | Loss: 46.000
Time elapsed: 0.17 min
Epoch: 002/050 | Batch 0000/0235 | Loss: 54.0511
Epoch: 002/050 | Batch 0050/0235 | Loss: 51.4986
Epoch: 002/050 | Batch 0100/0235 | Loss: 49.5837
Epoch: 002/050 | Batch 0150/0235 | Loss: 49.8921
Epoch: 002/050 | Batch 0200/0235 | Loss: 47.7225
***Epoch: 002/050 | Loss: 42.605
Time elapsed: 0.33 min
Epoch: 003/050 | Batch 0000/0235 | Loss: 49.0769
Epoch: 003/050 | Batch 0050/0235 | Loss: 48.0767
Epoch: 003/050 | Batch 0100/0235 | Loss: 48.2691
Epoch: 003/050 | Batch 0150/0235 | Loss: 47.5492
Epoch: 003/050 | Batch 0200/0235 | Loss: 46.7642
***Epoch: 003/050 | Loss: 41.028
Time elapsed: 0.49 min
Epoch: 004/050 | Batch 0000/0235 | Loss: 47.3210
Epoch: 004/050 | Batch 0050/0235 | Loss: 46.2264
Epoch: 004/050 | Batch 0100/0235 | Loss: 46.2966
Epoch: 004/050 | Batch 0150/0235 | Loss: 44.7269
Epoch: 004/050 | Batch 0200/0235 | Loss: 45.8496
***Epoch: 004/050 | Loss: 40.278
Time elapsed: 0.64 min
Epoch: 005/050 | Batch 0000/0235 | Loss: 46.4642
Epoch: 005/050 | Batch 0050/0235 | Loss: 45.4953
Epoch: 005/050 | Batch 0100/0235 | Loss: 44.7554
Epoch: 005/050 | Batch 0150/0235 | Loss: 46.0019
Epoch: 005/050 | Batch 0200/0235 | Loss: 43.9055
***Epoch: 005/050 | Loss: 39.567
Time elapsed: 0.80 min
Epoch: 006/050 | Batch 0000/0235 | Loss: 44.3087
Epoch: 006/050 | Batch 0050/0235 | Loss: 44.6984
Epoch: 006/050 | Batch 0100/0235 | Loss: 43.5045
Epoch: 006/050 | Batch 0150/0235 | Loss: 46.5066

```

```
Epoch: 006/050 | Batch 0200/0235 | Loss: 43.5113
***Epoch: 006/050 | Loss: 39.193
Time elapsed: 0.96 min
Epoch: 007/050 | Batch 0000/0235 | Loss: 42.7306
Epoch: 007/050 | Batch 0050/0235 | Loss: 43.6351
Epoch: 007/050 | Batch 0100/0235 | Loss: 45.3356
Epoch: 007/050 | Batch 0150/0235 | Loss: 44.4580
Epoch: 007/050 | Batch 0200/0235 | Loss: 43.7206
***Epoch: 007/050 | Loss: 38.687
Time elapsed: 1.12 min
Epoch: 008/050 | Batch 0000/0235 | Loss: 44.7416
Epoch: 008/050 | Batch 0050/0235 | Loss: 44.8916
Epoch: 008/050 | Batch 0100/0235 | Loss: 44.3987
Epoch: 008/050 | Batch 0150/0235 | Loss: 43.2250
Epoch: 008/050 | Batch 0200/0235 | Loss: 43.6281
***Epoch: 008/050 | Loss: 38.383
Time elapsed: 1.28 min
Epoch: 009/050 | Batch 0000/0235 | Loss: 43.9339
Epoch: 009/050 | Batch 0050/0235 | Loss: 42.8192
Epoch: 009/050 | Batch 0100/0235 | Loss: 43.8335
Epoch: 009/050 | Batch 0150/0235 | Loss: 42.7116
Epoch: 009/050 | Batch 0200/0235 | Loss: 42.3987
***Epoch: 009/050 | Loss: 38.116
Time elapsed: 1.44 min
Epoch: 010/050 | Batch 0000/0235 | Loss: 41.9803
Epoch: 010/050 | Batch 0050/0235 | Loss: 43.8122
Epoch: 010/050 | Batch 0100/0235 | Loss: 44.7661
Epoch: 010/050 | Batch 0150/0235 | Loss: 43.4683
Epoch: 010/050 | Batch 0200/0235 | Loss: 42.9096
***Epoch: 010/050 | Loss: 37.722
Time elapsed: 1.60 min
Epoch: 011/050 | Batch 0000/0235 | Loss: 42.5787
Epoch: 011/050 | Batch 0050/0235 | Loss: 44.4273
Epoch: 011/050 | Batch 0100/0235 | Loss: 42.1331
Epoch: 011/050 | Batch 0150/0235 | Loss: 43.2326
Epoch: 011/050 | Batch 0200/0235 | Loss: 43.4968
***Epoch: 011/050 | Loss: 37.626
Time elapsed: 1.76 min
Epoch: 012/050 | Batch 0000/0235 | Loss: 42.0469
Epoch: 012/050 | Batch 0050/0235 | Loss: 42.2037
Epoch: 012/050 | Batch 0100/0235 | Loss: 42.3549
Epoch: 012/050 | Batch 0150/0235 | Loss: 41.3022
Epoch: 012/050 | Batch 0200/0235 | Loss: 42.3972
***Epoch: 012/050 | Loss: 37.207
Time elapsed: 1.92 min
Epoch: 013/050 | Batch 0000/0235 | Loss: 43.2448
Epoch: 013/050 | Batch 0050/0235 | Loss: 42.1440
Epoch: 013/050 | Batch 0100/0235 | Loss: 42.8252
Epoch: 013/050 | Batch 0150/0235 | Loss: 41.8686
Epoch: 013/050 | Batch 0200/0235 | Loss: 41.7297
***Epoch: 013/050 | Loss: 36.993
Time elapsed: 2.09 min
Epoch: 014/050 | Batch 0000/0235 | Loss: 41.8763
Epoch: 014/050 | Batch 0050/0235 | Loss: 42.0802
Epoch: 014/050 | Batch 0100/0235 | Loss: 41.3353
Epoch: 014/050 | Batch 0150/0235 | Loss: 42.2503
Epoch: 014/050 | Batch 0200/0235 | Loss: 42.2639
***Epoch: 014/050 | Loss: 36.935
Time elapsed: 2.25 min
Epoch: 015/050 | Batch 0000/0235 | Loss: 40.8213
Epoch: 015/050 | Batch 0050/0235 | Loss: 42.8784
```

```
Epoch: 015/050 | Batch 0100/0235 | Loss: 41.6078
Epoch: 015/050 | Batch 0150/0235 | Loss: 43.0752
Epoch: 015/050 | Batch 0200/0235 | Loss: 40.9673
***Epoch: 015/050 | Loss: 36.561
Time elapsed: 2.42 min
Epoch: 016/050 | Batch 0000/0235 | Loss: 41.6636
Epoch: 016/050 | Batch 0050/0235 | Loss: 42.8638
Epoch: 016/050 | Batch 0100/0235 | Loss: 41.8576
Epoch: 016/050 | Batch 0150/0235 | Loss: 42.0306
Epoch: 016/050 | Batch 0200/0235 | Loss: 42.0040
***Epoch: 016/050 | Loss: 36.370
Time elapsed: 2.58 min
Epoch: 017/050 | Batch 0000/0235 | Loss: 41.6605
Epoch: 017/050 | Batch 0050/0235 | Loss: 41.2564
Epoch: 017/050 | Batch 0100/0235 | Loss: 42.0975
Epoch: 017/050 | Batch 0150/0235 | Loss: 41.6407
Epoch: 017/050 | Batch 0200/0235 | Loss: 42.2968
***Epoch: 017/050 | Loss: 36.234
Time elapsed: 2.75 min
Epoch: 018/050 | Batch 0000/0235 | Loss: 42.1136
Epoch: 018/050 | Batch 0050/0235 | Loss: 42.4410
Epoch: 018/050 | Batch 0100/0235 | Loss: 41.5587
Epoch: 018/050 | Batch 0150/0235 | Loss: 42.2724
Epoch: 018/050 | Batch 0200/0235 | Loss: 42.5564
***Epoch: 018/050 | Loss: 36.279
Time elapsed: 2.91 min
Epoch: 019/050 | Batch 0000/0235 | Loss: 42.6885
Epoch: 019/050 | Batch 0050/0235 | Loss: 41.3433
Epoch: 019/050 | Batch 0100/0235 | Loss: 42.2855
Epoch: 019/050 | Batch 0150/0235 | Loss: 40.6611
Epoch: 019/050 | Batch 0200/0235 | Loss: 41.2100
***Epoch: 019/050 | Loss: 35.931
Time elapsed: 3.08 min
Epoch: 020/050 | Batch 0000/0235 | Loss: 41.3141
Epoch: 020/050 | Batch 0050/0235 | Loss: 41.2292
Epoch: 020/050 | Batch 0100/0235 | Loss: 41.9318
Epoch: 020/050 | Batch 0150/0235 | Loss: 41.4969
Epoch: 020/050 | Batch 0200/0235 | Loss: 40.5901
***Epoch: 020/050 | Loss: 35.797
Time elapsed: 3.25 min
Epoch: 021/050 | Batch 0000/0235 | Loss: 42.1326
Epoch: 021/050 | Batch 0050/0235 | Loss: 41.8907
Epoch: 021/050 | Batch 0100/0235 | Loss: 42.1611
Epoch: 021/050 | Batch 0150/0235 | Loss: 41.4853
Epoch: 021/050 | Batch 0200/0235 | Loss: 41.1220
***Epoch: 021/050 | Loss: 35.662
Time elapsed: 3.43 min
Epoch: 022/050 | Batch 0000/0235 | Loss: 40.8813
Epoch: 022/050 | Batch 0050/0235 | Loss: 42.8904
Epoch: 022/050 | Batch 0100/0235 | Loss: 41.2094
Epoch: 022/050 | Batch 0150/0235 | Loss: 40.6341
Epoch: 022/050 | Batch 0200/0235 | Loss: 41.8519
***Epoch: 022/050 | Loss: 35.476
Time elapsed: 3.59 min
Epoch: 023/050 | Batch 0000/0235 | Loss: 40.0815
Epoch: 023/050 | Batch 0050/0235 | Loss: 41.6736
Epoch: 023/050 | Batch 0100/0235 | Loss: 41.3442
Epoch: 023/050 | Batch 0150/0235 | Loss: 41.8439
Epoch: 023/050 | Batch 0200/0235 | Loss: 42.2069
***Epoch: 023/050 | Loss: 35.344
```

```
Time elapsed: 3.76 min
Epoch: 024/050 | Batch 0000/0235 | Loss: 39.0077
Epoch: 024/050 | Batch 0050/0235 | Loss: 42.1516
Epoch: 024/050 | Batch 0100/0235 | Loss: 41.1327
Epoch: 024/050 | Batch 0150/0235 | Loss: 41.5365
Epoch: 024/050 | Batch 0200/0235 | Loss: 40.2786
***Epoch: 024/050 | Loss: 35.400
Time elapsed: 3.93 min
Epoch: 025/050 | Batch 0000/0235 | Loss: 42.1447
Epoch: 025/050 | Batch 0050/0235 | Loss: 41.6692
Epoch: 025/050 | Batch 0100/0235 | Loss: 40.8320
Epoch: 025/050 | Batch 0150/0235 | Loss: 40.4013
Epoch: 025/050 | Batch 0200/0235 | Loss: 39.1142
***Epoch: 025/050 | Loss: 35.254
Time elapsed: 4.09 min
Epoch: 026/050 | Batch 0000/0235 | Loss: 41.1271
Epoch: 026/050 | Batch 0050/0235 | Loss: 39.1342
Epoch: 026/050 | Batch 0100/0235 | Loss: 40.9424
Epoch: 026/050 | Batch 0150/0235 | Loss: 40.8151
Epoch: 026/050 | Batch 0200/0235 | Loss: 39.0052
***Epoch: 026/050 | Loss: 35.344
Time elapsed: 4.25 min
Epoch: 027/050 | Batch 0000/0235 | Loss: 41.7474
Epoch: 027/050 | Batch 0050/0235 | Loss: 39.0859
Epoch: 027/050 | Batch 0100/0235 | Loss: 41.3096
Epoch: 027/050 | Batch 0150/0235 | Loss: 41.6874
Epoch: 027/050 | Batch 0200/0235 | Loss: 39.4471
***Epoch: 027/050 | Loss: 34.893
Time elapsed: 4.41 min
Epoch: 028/050 | Batch 0000/0235 | Loss: 42.4002
Epoch: 028/050 | Batch 0050/0235 | Loss: 41.2694
Epoch: 028/050 | Batch 0100/0235 | Loss: 41.0438
Epoch: 028/050 | Batch 0150/0235 | Loss: 41.9139
Epoch: 028/050 | Batch 0200/0235 | Loss: 39.2109
***Epoch: 028/050 | Loss: 35.129
Time elapsed: 4.58 min
Epoch: 029/050 | Batch 0000/0235 | Loss: 39.1343
Epoch: 029/050 | Batch 0050/0235 | Loss: 39.6727
Epoch: 029/050 | Batch 0100/0235 | Loss: 39.0908
Epoch: 029/050 | Batch 0150/0235 | Loss: 40.9170
Epoch: 029/050 | Batch 0200/0235 | Loss: 39.5028
***Epoch: 029/050 | Loss: 34.977
Time elapsed: 4.74 min
Epoch: 030/050 | Batch 0000/0235 | Loss: 39.7044
Epoch: 030/050 | Batch 0050/0235 | Loss: 39.0123
Epoch: 030/050 | Batch 0100/0235 | Loss: 39.8663
Epoch: 030/050 | Batch 0150/0235 | Loss: 39.3814
Epoch: 030/050 | Batch 0200/0235 | Loss: 39.1825
***Epoch: 030/050 | Loss: 34.794
Time elapsed: 4.90 min
Epoch: 031/050 | Batch 0000/0235 | Loss: 41.2183
Epoch: 031/050 | Batch 0050/0235 | Loss: 38.8674
Epoch: 031/050 | Batch 0100/0235 | Loss: 40.9213
Epoch: 031/050 | Batch 0150/0235 | Loss: 40.8010
Epoch: 031/050 | Batch 0200/0235 | Loss: 40.1680
***Epoch: 031/050 | Loss: 34.746
Time elapsed: 5.06 min
Epoch: 032/050 | Batch 0000/0235 | Loss: 40.8648
Epoch: 032/050 | Batch 0050/0235 | Loss: 38.5969
Epoch: 032/050 | Batch 0100/0235 | Loss: 39.8466
```

```
Epoch: 032/050 | Batch 0150/0235 | Loss: 41.8637
Epoch: 032/050 | Batch 0200/0235 | Loss: 40.7991
***Epoch: 032/050 | Loss: 34.638
Time elapsed: 5.22 min
Epoch: 033/050 | Batch 0000/0235 | Loss: 40.7136
Epoch: 033/050 | Batch 0050/0235 | Loss: 39.8299
Epoch: 033/050 | Batch 0100/0235 | Loss: 39.9795
Epoch: 033/050 | Batch 0150/0235 | Loss: 41.9615
Epoch: 033/050 | Batch 0200/0235 | Loss: 39.1906
***Epoch: 033/050 | Loss: 34.652
Time elapsed: 5.40 min
Epoch: 034/050 | Batch 0000/0235 | Loss: 40.6842
Epoch: 034/050 | Batch 0050/0235 | Loss: 40.3172
Epoch: 034/050 | Batch 0100/0235 | Loss: 39.2567
Epoch: 034/050 | Batch 0150/0235 | Loss: 39.9506
Epoch: 034/050 | Batch 0200/0235 | Loss: 40.1054
***Epoch: 034/050 | Loss: 34.631
Time elapsed: 5.56 min
Epoch: 035/050 | Batch 0000/0235 | Loss: 39.9762
Epoch: 035/050 | Batch 0050/0235 | Loss: 40.8454
Epoch: 035/050 | Batch 0100/0235 | Loss: 40.8100
Epoch: 035/050 | Batch 0150/0235 | Loss: 40.6176
Epoch: 035/050 | Batch 0200/0235 | Loss: 41.7579
***Epoch: 035/050 | Loss: 34.578
Time elapsed: 5.72 min
Epoch: 036/050 | Batch 0000/0235 | Loss: 41.6314
Epoch: 036/050 | Batch 0050/0235 | Loss: 39.2654
Epoch: 036/050 | Batch 0100/0235 | Loss: 39.7288
Epoch: 036/050 | Batch 0150/0235 | Loss: 41.4062
Epoch: 036/050 | Batch 0200/0235 | Loss: 39.4609
***Epoch: 036/050 | Loss: 34.469
Time elapsed: 5.88 min
Epoch: 037/050 | Batch 0000/0235 | Loss: 40.3508
Epoch: 037/050 | Batch 0050/0235 | Loss: 40.1758
Epoch: 037/050 | Batch 0100/0235 | Loss: 38.5451
Epoch: 037/050 | Batch 0150/0235 | Loss: 39.9727
Epoch: 037/050 | Batch 0200/0235 | Loss: 38.8012
***Epoch: 037/050 | Loss: 34.483
Time elapsed: 6.04 min
Epoch: 038/050 | Batch 0000/0235 | Loss: 40.0443
Epoch: 038/050 | Batch 0050/0235 | Loss: 39.8396
Epoch: 038/050 | Batch 0100/0235 | Loss: 40.5535
Epoch: 038/050 | Batch 0150/0235 | Loss: 40.1700
Epoch: 038/050 | Batch 0200/0235 | Loss: 39.9460
***Epoch: 038/050 | Loss: 34.187
Time elapsed: 6.21 min
Epoch: 039/050 | Batch 0000/0235 | Loss: 41.3887
Epoch: 039/050 | Batch 0050/0235 | Loss: 41.2908
Epoch: 039/050 | Batch 0100/0235 | Loss: 41.1381
Epoch: 039/050 | Batch 0150/0235 | Loss: 40.3449
Epoch: 039/050 | Batch 0200/0235 | Loss: 40.3548
***Epoch: 039/050 | Loss: 34.223
Time elapsed: 6.37 min
Epoch: 040/050 | Batch 0000/0235 | Loss: 39.6550
Epoch: 040/050 | Batch 0050/0235 | Loss: 39.2371
Epoch: 040/050 | Batch 0100/0235 | Loss: 39.7899
Epoch: 040/050 | Batch 0150/0235 | Loss: 38.8216
Epoch: 040/050 | Batch 0200/0235 | Loss: 40.6724
***Epoch: 040/050 | Loss: 34.475
Time elapsed: 6.54 min
Epoch: 041/050 | Batch 0000/0235 | Loss: 41.0817
```

computervision-final-prep/lab/Lab 11 (Generative Network)-20251128/AE-VAE.ipynb at main · limhponer/computervision-final-prep

Epoch	Batch	Loss
041/050	0050/0235	40.5469
041/050	0100/0235	38.9778
041/050	0150/0235	40.6089
041/050	0200/0235	39.4389
***Epoch: 041/050   Loss: 34.332		
Time elapsed: 6.71 min		
042/050	0000/0235	41.5702
042/050	0050/0235	40.0644
042/050	0100/0235	39.8403
042/050	0150/0235	39.4446
042/050	0200/0235	40.8135
***Epoch: 042/050   Loss: 34.184		
Time elapsed: 6.87 min		
043/050	0000/0235	42.1320
043/050	0050/0235	40.0999
043/050	0100/0235	41.2374
043/050	0150/0235	40.0034
043/050	0200/0235	38.7538
***Epoch: 043/050   Loss: 34.098		
Time elapsed: 7.04 min		
044/050	0000/0235	38.5440
044/050	0050/0235	39.6562
044/050	0100/0235	38.1874
044/050	0150/0235	40.5302
044/050	0200/0235	41.5771
***Epoch: 044/050   Loss: 34.044		
Time elapsed: 7.21 min		
045/050	0000/0235	40.2022
045/050	0050/0235	39.7884
045/050	0100/0235	41.5700
045/050	0150/0235	40.2363
045/050	0200/0235	42.4268
***Epoch: 045/050   Loss: 33.967		
Time elapsed: 7.37 min		
046/050	0000/0235	40.1257
046/050	0050/0235	39.2605
046/050	0100/0235	37.5402
046/050	0150/0235	38.8494
046/050	0200/0235	40.4664
***Epoch: 046/050   Loss: 34.064		
Time elapsed: 7.53 min		
047/050	0000/0235	39.5175
047/050	0050/0235	41.4605
047/050	0100/0235	40.7118
047/050	0150/0235	39.8212
047/050	0200/0235	42.2698
***Epoch: 047/050   Loss: 34.181		
Time elapsed: 7.69 min		
048/050	0000/0235	39.4376
048/050	0050/0235	39.2865
048/050	0100/0235	40.6450
048/050	0150/0235	39.0865
048/050	0200/0235	40.7237
***Epoch: 048/050   Loss: 33.920		
Time elapsed: 7.85 min		
049/050	0000/0235	41.2092
049/050	0050/0235	39.9116
049/050	0100/0235	39.1651
049/050	0150/0235	41.0919
049/050	0200/0235	40.3547

```
***Epoch: 049/050 | Loss: 33.902
Time elapsed: 8.01 min
Epoch: 050/050 | Batch 0000/0235 | Loss: 39.2820
Epoch: 050/050 | Batch 0050/0235 | Loss: 40.7778
Epoch: 050/050 | Batch 0100/0235 | Loss: 39.1879
Epoch: 050/050 | Batch 0150/0235 | Loss: 40.3235
Epoch: 050/050 | Batch 0200/0235 | Loss: 39.8487
***Epoch: 050/050 | Loss: 33.925
Time elapsed: 8.17 min
Total Training Time: 8.17 min
```

## Plot loss

In [15]:

```
def plot_training_loss(minibatch_losses, num_epochs, averaging_iterations=100, c
    iter_per_epoch = len(minibatch_losses) // num_epochs

    plt.figure()
    ax1 = plt.subplot(1, 1, 1)
    ax1.plot(range(len(minibatch_losses)),
              (minibatch_losses), label=f'Minibatch Loss{custom_label}')
    ax1.set_xlabel('Iterations')
    ax1.set_ylabel('Loss')

    if len(minibatch_losses) < 1000:
        num_losses = len(minibatch_losses) // 2
    else:
        num_losses = 1000

    ax1.set_ylim([
        0, np.max(minibatch_losses[num_losses:])*1.5
    ])

    ax1.plot(np.convolve(minibatch_losses,
                         np.ones(averaging_iterations,)/averaging_iterations,
                         mode='valid'),
             label=f'Running Average{custom_label}')
    ax1.legend()

#####
# Set second x-axis
ax2 = ax1.twiny()
newlabel = list(range(num_epochs+1))

newpos = [e*iter_per_epoch for e in newlabel]

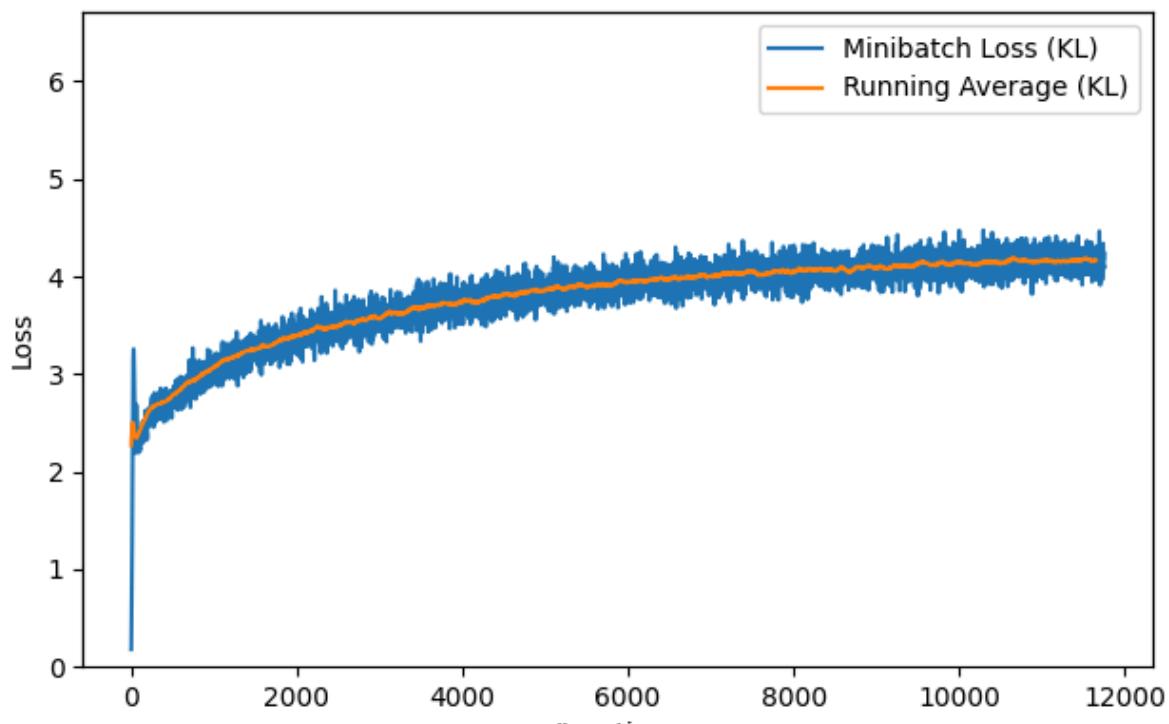
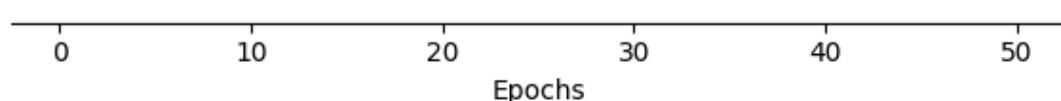
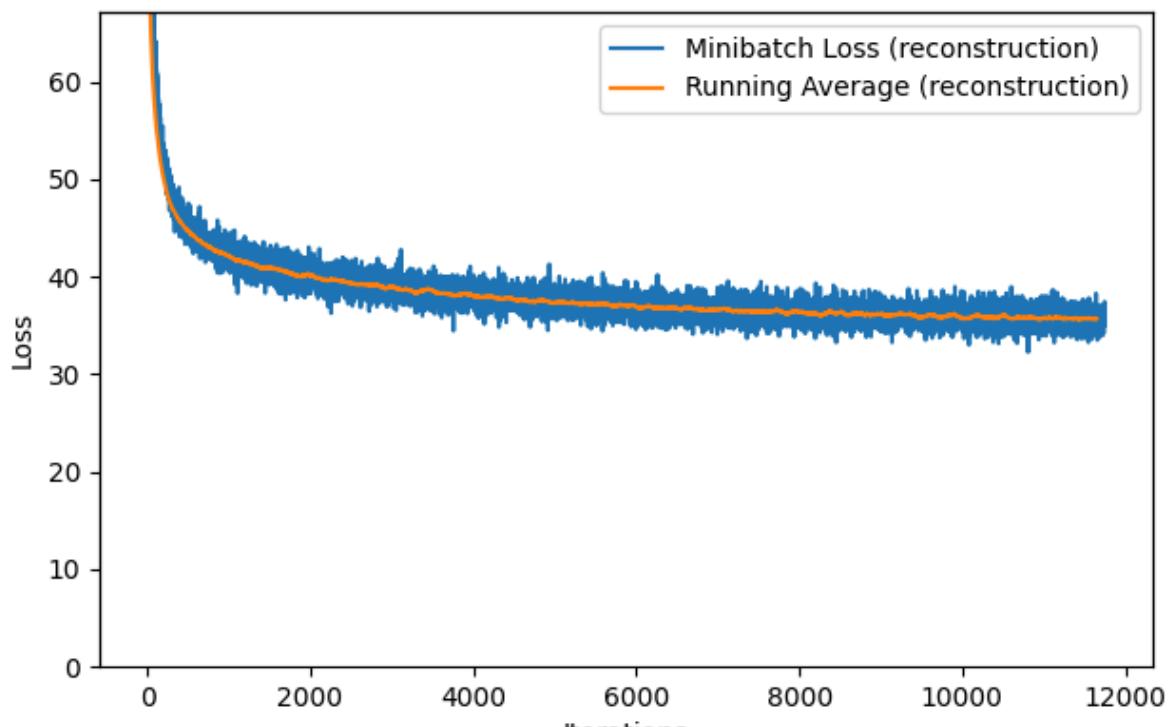
ax2.set_xticks(newpos[::10])
ax2.set_xticklabels(newlabel[::10])

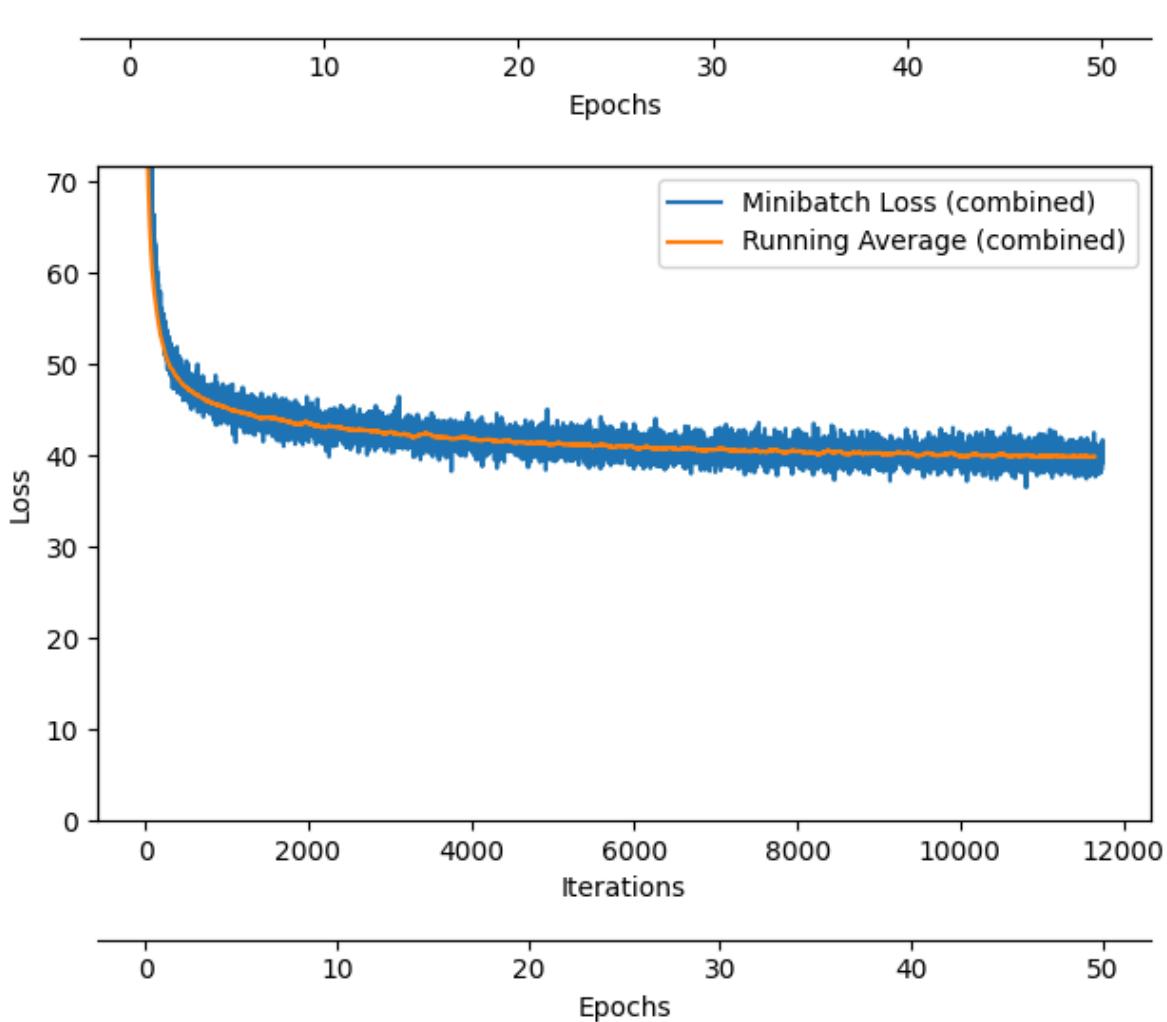
ax2.xaxis.set_ticks_position('bottom')
ax2.xaxis.set_label_position('bottom')
ax2.spines['bottom'].set_position(('outward', 45))
ax2.set_xlabel('Epochs')
ax2.set_xlim(ax1.get_xlim())
#####

plt.tight_layout()
```

In [16]:

```
plot_training_loss(log_dict['train_reconstruction_loss_per_batch'], NUM_EPOCHS,
plot_training_loss(log_dict['train_kl_loss_per_batch'], NUM_EPOCHS, custom_label
plot_training_loss(log_dict['train_combined_loss_per_batch'], NUM_EPOCHS, custom
plt.show()
```





In [17]:

```
def plot_generated_images(data_loader, model, device,
                         unnormalizer=None,
                         figsize=(20, 2.5), n_images=15):

    fig, axes = plt.subplots(nrows=2, ncols=n_images,
                           sharex=True, sharey=True, figsize=figsize)

    for batch_idx, (features, _) in enumerate(data_loader):

        features = features.to(device)

        color_channels = features.shape[1]
        image_height = features.shape[2]
        image_width = features.shape[3]

        with torch.no_grad():
            encoded, z_mean, z_log_var, decoded_images = model(features)[:n_images]

        orig_images = features[:n_images]
        break

    for i in range(n_images):
        for ax, img in zip(axes, [orig_images, decoded_images]):
            curr_img = img[i].detach().to(torch.device('cpu'))
            if unnormalizer is not None:
                curr_img = unnormalizer(curr_img)
```

```

        if color_channels > 1:
            curr_img = np.transpose(curr_img, (1, 2, 0))
            ax[i].imshow(curr_img)
        else:
            ax[i].imshow(curr_img.view((image_height, image_width)), cmap='binary')

def plot_latent_space_with_labels(num_classes, data_loader, encoding_fn, device):
    d = {i:[] for i in range(num_classes)}

    with torch.no_grad():
        for i, (features, targets) in enumerate(data_loader):
            features = features.to(device)
            targets = targets.to(device)

            embedding = encoding_fn(features)

            for i in range(num_classes):
                if i in targets:
                    mask = targets == i
                    d[i].append(embedding[mask].to('cpu').numpy())

    colors = list(mcolors.TABLEAU_COLORS.items())
    for i in range(num_classes):
        d[i] = np.concatenate(d[i])
        plt.scatter(
            d[i][:, 0], d[i][:, 1],
            color=colors[i][1],
            label=f'{i}',
            alpha=0.5)

    plt.legend()

def plot_images_sampled_from_vae(model, device, latent_size, unnormalizer=None,
                                  with torch.no_grad():
                                      #####
                                      ### RANDOM SAMPLE
                                      #####
                                      rand_features = torch.randn(num_images, latent_size).to(device)
                                      new_images = model.decoder(rand_features)
                                      color_channels = new_images.shape[1]
                                      image_height = new_images.shape[2]
                                      image_width = new_images.shape[3]

                                      #####
                                      ### VISUALIZATION
                                      #####
                                      image_width = 28

                                      fig, axes = plt.subplots(nrows=1, ncols=num_images, figsize=(10, 2.5), sharex=True)
                                      decoded_images = new_images[:num_images]

                                      for ax, img in zip(axes, decoded_images):
                                          curr_img = img.detach().to(torch.device('cpu'))
                                          if unnormalizer is not None:
                                              curr_img = unnormalizer(curr_img)
                                          ax.imshow(curr_img)

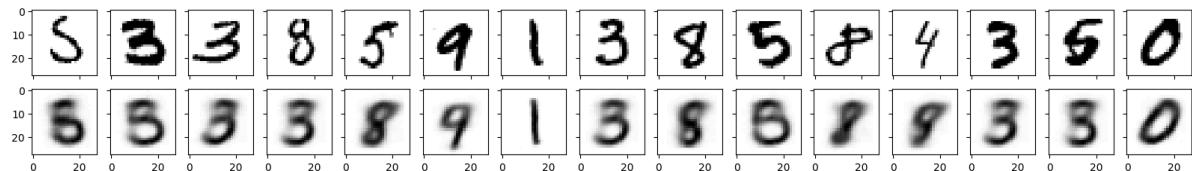
```

```

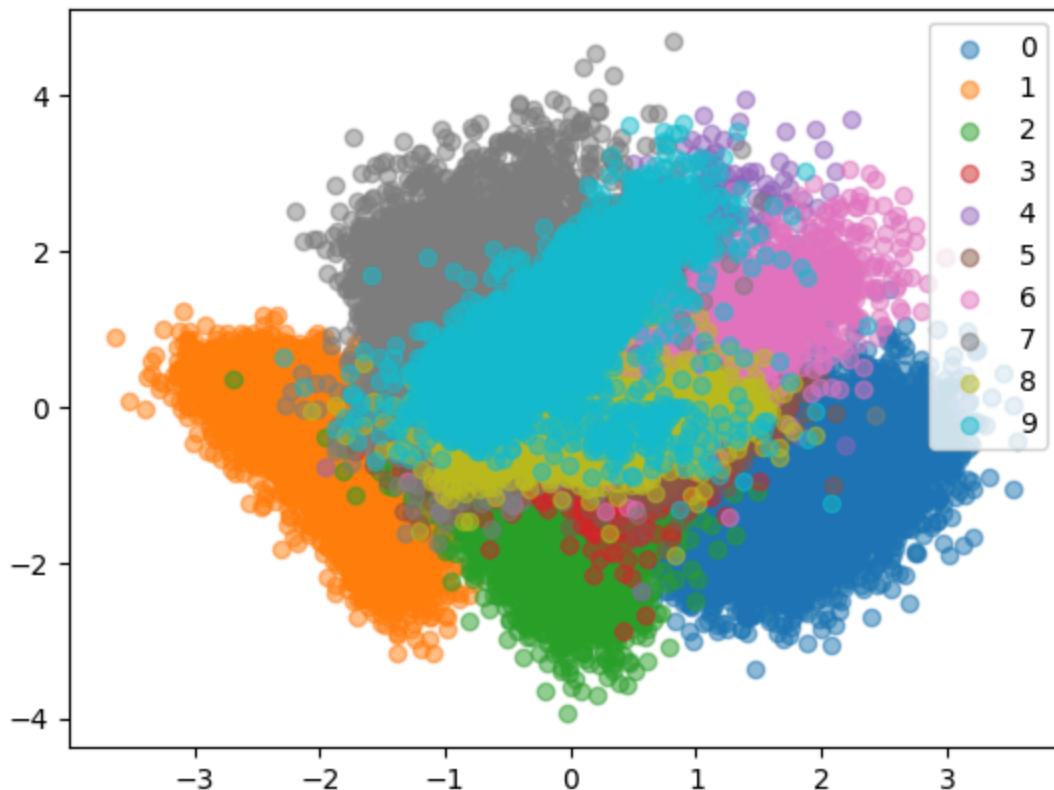
if color_channels > 1:
    curr_img = np.transpose(curr_img, (1, 2, 0))
    ax.imshow(curr_img)
else:
    ax.imshow(curr_img.view((image_height, image_width)), cmap='binary')

```

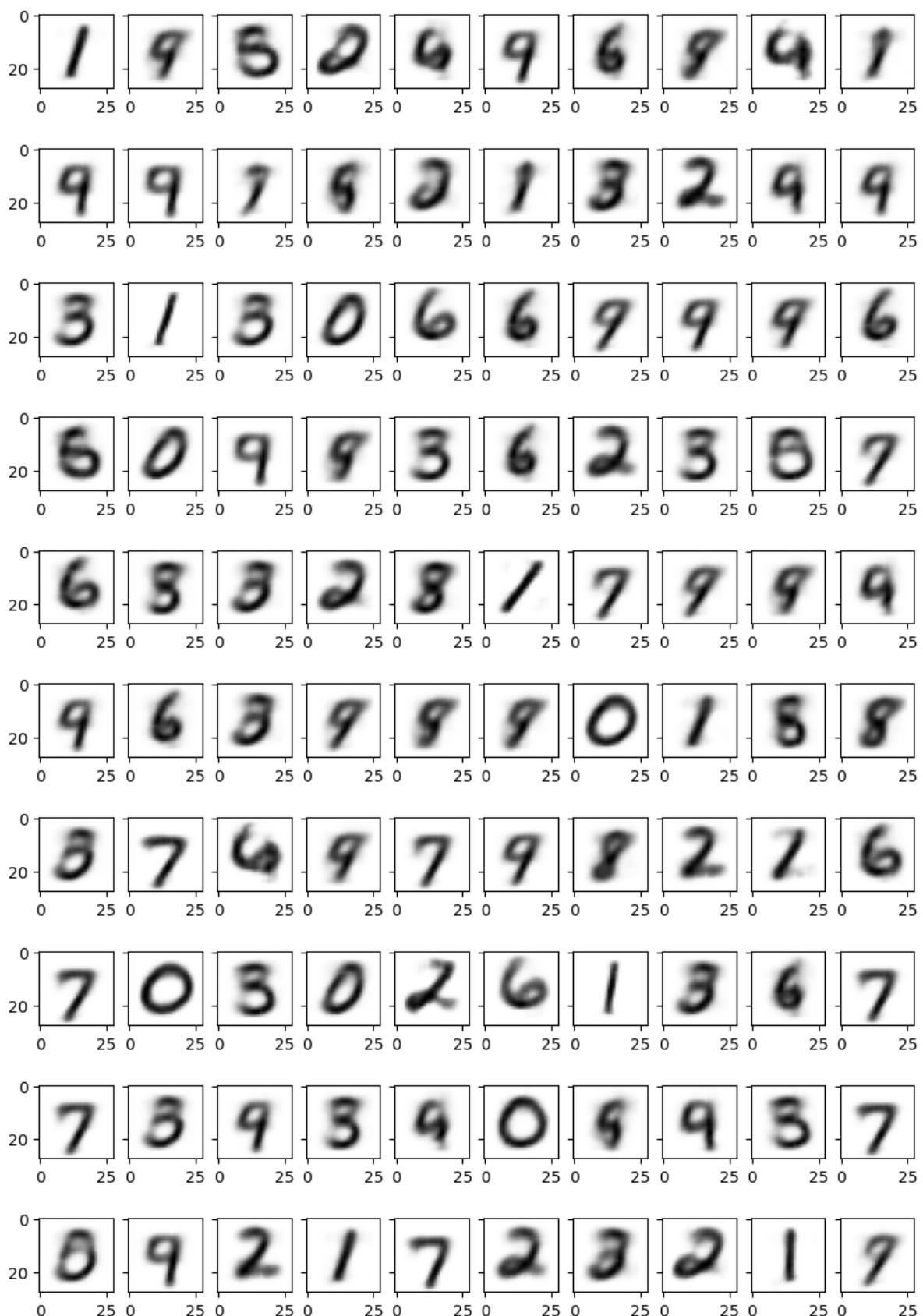
In [18]: `plot_generated_images(data_loader=train_loader, model=model, device=device)`



In [19]: `plot_latent_space_with_labels(num_classes=10, data_loader=train_loader, encoding_fn=model.encoding_fn, device=device)`  
`plt.legend()`  
`plt.show()`



In [20]: `for i in range(10):`  
`plot_images_sampled_from_vae(model=model, device=device, latent_size=2)`  
`plt.show()`



In [ ]:

