nvae-on-mnist

September 25, 2021

[ ]:
```
!nvidia-smi
```

```
Sat Sep 25 10:01:55 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   73C    P8    34W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[ ]:
```
import cv2
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch
import torchvision
import torch.nn as nn
from torchvision import transforms
import torch.nn.functional as F
from torchvision.utils import save_image
```

[ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[ ]:
```
device
```

[ ]:
```
device(type='cuda')
```

0.1 Depth Wise Seprable Convolution

This convolution originated from the idea that depth and spatial dimension of a filter can be separated- thus the name separable. You can separate the height and width dimension of these filters. Gx filter (see fig 3) can be viewed as matrix product of [1 2 1] transpose with [-1 0 1].

that the filter had disguised itself. It shows it had 9 parameters but it has actually 6. This has been possible because of separation of its height and width dimensions. The same idea applied to separate depth dimension from horizontal (width*height) gives us depth-wise separable convolution whare we perform depth-wise convolution and after that we use a 1*1 filter to cover the depth dimension.

0.1.1 Torch Implementation Related Documen

`torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)`

Parameters:

`stride controls the stride for the cross-correlation, a single number or a tuple.`

`padding controls the amount of padding applied to the input. It can be either a string {‘valid`

`dilation controls the spacing between the kernel points; also known as the à trous algorithm.`

`groups controls the connections between inputs and outputs. in_channels and out_channels must`

[ ]:
```
class depthwise_separable_conv(nn.Module):
    def __init__(self, nin, kernels_per_layer, nout):
        super(depthwise_separable_conv, self).__init__()
        self.depthwise = nn.Conv2d(nin, nin * kernels_per_layer, kernel_size=5,␣
↪padding=2, groups=nin)
        self.pointwise = nn.Conv2d(nin * kernels_per_layer, nout, kernel_size=1)

    def forward(self, x):
        out = self.depthwise(x)
        out = self.pointwise(out)
        return out
```

0.2 Swish Activation Function

Swish is a smooth, non-monotonic function that consistently matches or outperforms ReLU on deep networks applied to a variety of challenging domains such as Image classification and Machine translation. It is unbounded above and bounded below & it is the non-monotonic attribute that actually creates the difference. With self-gating, it requires just a scalar input whereas in multi-gating scenario, it would require multiple two-scalar input.

[ ]:
```
def swish(x):
    return x * torch.sigmoid(x)
```

0.3 Squeeze and Excitation Networks

For more information see this paper.

Squeeze-and-Excitation Networks (SENets) introduce a building block for CNNs that improves channel interdependencies at almost no computational cost.

The transformation simply corresponds with the operation that the network where you are going to implement the SE block would perform in its natural scheme. For instance, if you are in a block within a ResNet, the Ftr term will correspond with the process of the entire residual block (convolution, batch normalization, ReLU…).

The squeezing step is probably the most simply one. It basically performs a average pooling at each channel to create a 1x1 squeezed representation of the volume U.

The authors introduce a new parameter called the reduction ratio r, to introduce a first fully connected (FC) layer with a ReLU activation, before the gating network with the sigmoid activation.

The reason to do this is to introduce a bottleneck that allows us to reduce the dimensionality at the same time that introduce new non-linearities.

Furthermore, we can have better control on the model complexity and aid the generalization property of the network.

Having two FC layers will result on having 2 matrices of weights that will be learned by the network during the training in an end-to-end fashion (all of them are backpropagated together with the convolutional kernels).

The last step, scaling, is indeed a re-scaling operation. We are going to give the squeezed vector its original shape, keeping the information obtained during the excitation step.

Mathematically, the scaling is achieved by simple scalar product of each channel on the input volume with the corresponding channel on the activated 1x1 squeezed vector.

[ ]:
```
class ChannelSELayer(nn.Module):
    def __init__(self, num_channels, reduction_ratio=2):
        super(ChannelSELayer, self).__init__()
        num_channels_reduced = num_channels // reduction_ratio
        self.reduction_ratio = reduction_ratio
        self.fc1 = nn.Linear(num_channels, num_channels_reduced, bias=True)
        self.fc2 = nn.Linear(num_channels_reduced, num_channels, bias=True)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_tensor):
        batch_size, num_channels, H, W = input_tensor.size()
        squeeze_tensor = input_tensor.view(batch_size, num_channels, -1).
↪mean(dim=2)
```

```
        fc_out_1 = self.relu(self.fc1(squeeze_tensor))
        fc_out_2 = self.sigmoid(self.fc2(fc_out_1))

        a, b = squeeze_tensor.size()
        output_tensor = torch.mul(input_tensor, fc_out_2.view(a, b, 1, 1))
        return output_tensor
```

[ ]:
```
class dec_res(nn.Module):
  def __init__(self,in_channel):
    super(dec_res,self).__init__()
    self.bn1 = nn.BatchNorm2d(in_channel)
    self.c1 = nn.
↪Conv2d(in_channels=in_channel,out_channels=2*in_channel,kernel_size=1,stride=1,padding=0)
    self.bn2 = nn.BatchNorm2d(2*in_channel)
    self.dc1 =␣
↪depthwise_separable_conv(nin=2*in_channel,kernels_per_layer=3,nout=2*in_channel)
    self.bn3 = nn.BatchNorm2d(2*in_channel)
    self.c2 = nn.
↪Conv2d(in_channels=2*in_channel,out_channels=in_channel,kernel_size=1,stride=1,padding=0)
    self.bn4 = nn.BatchNorm2d(in_channel)
    self.SE = ChannelSELayer(in_channel)
  def forward(self,x1):
    x = self.c1(self.bn1(x1))
    x = swish(self.bn2(x))
    x = self.dc1(x)
    x = swish(self.bn3(x))
    x = self.bn4(self.c2(x))
    x = self.SE(x)
    return x+x1
```

[ ]:
```
class enc_res(nn.Module):
  def __init__(self,in_channel):
    super(enc_res,self).__init__()
    self.bn1 = nn.BatchNorm2d(in_channel)
    self.c1 = nn.
↪Conv2d(in_channels=in_channel,out_channels=2*in_channel,kernel_size=3,stride=1,padding=1)
    self.bn2 = nn.BatchNorm2d(2*in_channel)
    self.c2 = nn.
↪Conv2d(in_channels=2*in_channel,out_channels=in_channel,kernel_size=3,stride=1,padding=1)
    self.bn3 = nn.BatchNorm2d(in_channel)
    self.SE = ChannelSELayer(in_channel)
  def forward(self,x1):
    x = self.c1(swish(self.bn1(x1)))
    x = self.c2(swish(self.bn2(x)))
    x = self.SE(x)
    return x+x1
```

```
[ ]: class NVAE(nn.Module):
       def __init__(self,start_channel,original_dim):
         super(NVAE,self).__init__()
         self.original_dim = original_dim
         self.conv1 = nn.
     ↪Conv2d(in_channels=start_channel,out_channels=8,kernel_size=3,stride=1,padding=1)
         self.encblock1 = enc_res(8)
         self.dsconv1 = nn.
     ↪Conv2d(in_channels=8,out_channels=8,kernel_size=2,stride=2,padding=0)
         self.encblock2 = enc_res(8)
         self.dsconv2 = nn.
     ↪Conv2d(in_channels=8,out_channels=8,kernel_size=2,stride=2,padding=0)

         self.qmu1 = nn.
     ↪Linear(original_dim*original_dim*2,original_dim*original_dim*2)
         self.qvar1 = nn.
     ↪Linear(original_dim*original_dim*2,original_dim*original_dim*2)

         self.qmu0 = nn.Linear(original_dim*original_dim//
     ↪2,original_dim*original_dim//2)
         self.qvar0 = nn.Linear(original_dim*original_dim//
     ↪2,original_dim*original_dim//2)

         self.pmu1 = nn.
     ↪Linear(original_dim*original_dim*2,original_dim*original_dim*2)
         self.pvar1 = nn.
     ↪Linear(original_dim*original_dim*2,original_dim*original_dim*2)

         self.decblock1 = dec_res(8)
         self.usconv1 = nn.
     ↪ConvTranspose2d(in_channels=8,out_channels=8,kernel_size=2,stride=2,padding=0)
         self.decblock2 = dec_res(16)
         self.usconv2 = nn.
     ↪ConvTranspose2d(in_channels=16,out_channels=16,kernel_size=2,stride=2,padding=0)
         self.decblock3 = dec_res(16)
         self.finconv = nn.
     ↪Conv2d(in_channels=16,out_channels=start_channel,kernel_size=3,stride=1,padding=1)

       def forward(self,x):
         z1 = self.dsconv1(self.encblock1(self.conv1(x)))
         z0 = self.dsconv2(self.encblock2(z1))

         qmu0 = self.qmu0(z0.reshape(z0.shape[0],self.original_dim*self.original_dim/
     ↪/2))
         qvar0 = self.qvar0(z0.reshape(z0.shape[0],self.original_dim*self.
     ↪original_dim//2))
```

```
   qmu1 = self.qmu1(z1.reshape(z1.shape[0],self.original_dim*self.
↪original_dim*2))
   qvar1 = self.qvar1(z1.reshape(z1.shape[0],self.original_dim*self.
↪original_dim*2))

   stdvar0 = qvar0.mul(0.5).exp_()
   stdvar1 = qvar1.mul(0.5).exp_()

   e0 = torch.randn(qmu0.shape).to(device)
   ez0 = qmu0+e0*stdvar0
   ez0 = ez0.reshape(ez0.shape[0],8,self.original_dim//4,self.original_dim//4)
   ez1 = self.usconv1(self.decblock1(ez0))

   pmu1 = self.pmu1(ez1.reshape(ez1.shape[0],self.original_dim*self.
↪original_dim*2))
   pvar1 = self.pvar1(ez1.reshape(ez1.shape[0],self.original_dim*self.
↪original_dim*2))

   pstdvar1 = pvar1.mul(0.5).exp_()

   e2 = torch.randn(qmu1.shape).to(device)
   ez2 = pmu1+qmu1 + e2*pstdvar1*stdvar1
   ez2 = ez2.reshape(ez2.shape[0],8,self.original_dim//2,self.original_dim//2)

   final = torch.cat((ez1,ez2),1)

   recons = nn.Sigmoid()(self.finconv(self.decblock3(self.usconv2(self.
↪decblock2(final)))))

   return qmu0,qvar0,qmu1,qvar1,pmu1,pvar1,recons

 def sample(self,bs):
   e = torch.randn([bs,8,self.original_dim//4,self.original_dim//4]).to(device)
   ez1 = self.usconv1(self.decblock1(e))

   pmu1 = self.pmu1(ez1.reshape(ez1.shape[0],self.original_dim*self.
↪original_dim*2))
   pvar1 = self.pvar1(ez1.reshape(ez1.shape[0],self.original_dim*self.
↪original_dim*2))

   stdvar1 = pvar1.mul(0.5).exp_()

   e1 = torch.randn([ez1.shape[0],self.original_dim*self.original_dim*2]).
↪to(device)
   e1 = pmu1 + e1*stdvar1
```

```
    e1 = e1.reshape(e1.shape[0],8,self.original_dim//2,self.original_dim//2)
    recons = nn.Sigmoid()(self.finconv(self.decblock3(self.usconv2(self.
↪decblock2(torch.cat((ez1,e1),1))))))

    return recons

  def loss(self,x):
    qmu0,qvar0,qmu1,qvar1,pmu1,pvar1,recons = self.forward(x)
    klz0 = 0.5*torch.sum(torch.square(qmu0)+qvar0.exp()-qvar0-1)/x.shape[0]
    klz1 = 0.5*torch.sum(torch.square(qmu1)/pvar1.exp()+qvar1.exp()-qvar1-1)
    reconsloss = nn.BCELoss()(recons,x)
    return klz0,klz1,reconsloss
```

[ ]:
```
batch_size=64
```

[ ]:
```
transform = transforms.Compose([transforms.ToTensor()])
```

[ ]:
```
train_dataset = torchvision.datasets.MNIST(root='data/mnist',
                                           train=True,
                                           transform=transform,
                                           download=True)

test_dataset = torchvision.datasets.MNIST(root='data/mnist',
                                          train=False,
                                          transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=True)
```

```
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/MNIST/raw/train-images-idx3-ubyte.gz
  0%|          | 0/9912422 [00:00<?, ?it/s]
Extracting data/mnist/MNIST/raw/train-images-idx3-ubyte.gz to
data/mnist/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/MNIST/raw/train-labels-idx1-ubyte.gz
  0%|          | 0/28881 [00:00<?, ?it/s]
```

```
Extracting data/mnist/MNIST/raw/train-labels-idx1-ubyte.gz to
data/mnist/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/MNIST/raw/t10k-images-idx3-ubyte.gz

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

Extracting data/mnist/MNIST/raw/t10k-images-idx3-ubyte.gz to
data/mnist/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/MNIST/raw/t10k-labels-idx1-ubyte.gz

  0%|          | 0/4542 [00:00<?, ?it/s]

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


/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:498:
UserWarning: The given NumPy array is not writeable, and PyTorch does not
support non-writeable tensors. This means you can write to the underlying
(supposedly non-writeable) NumPy array using the tensor. You may want to copy
the array to protect its data or make it writeable before converting it to a
tensor. This type of warning will be suppressed for the rest of this program.
(Triggered internally at  /pytorch/torch/csrc/utils/tensor_numpy.cpp:180.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

[ ]:
```
model = NVAE(1,28).to(device)
```

AdaMax is a generalisation of Adam from $l2$ the norm $l\infty$ to the norm.

`torch.optim.Adamax(params, lr=0.002, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)`

Implements Adamax algorithm (a variant of Adam based on infinity norm).

[ ]:
```
optim = torch.optim.Adamax(model.parameters())
```

[ ]:
```
epochs=50
```

[ ]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
```

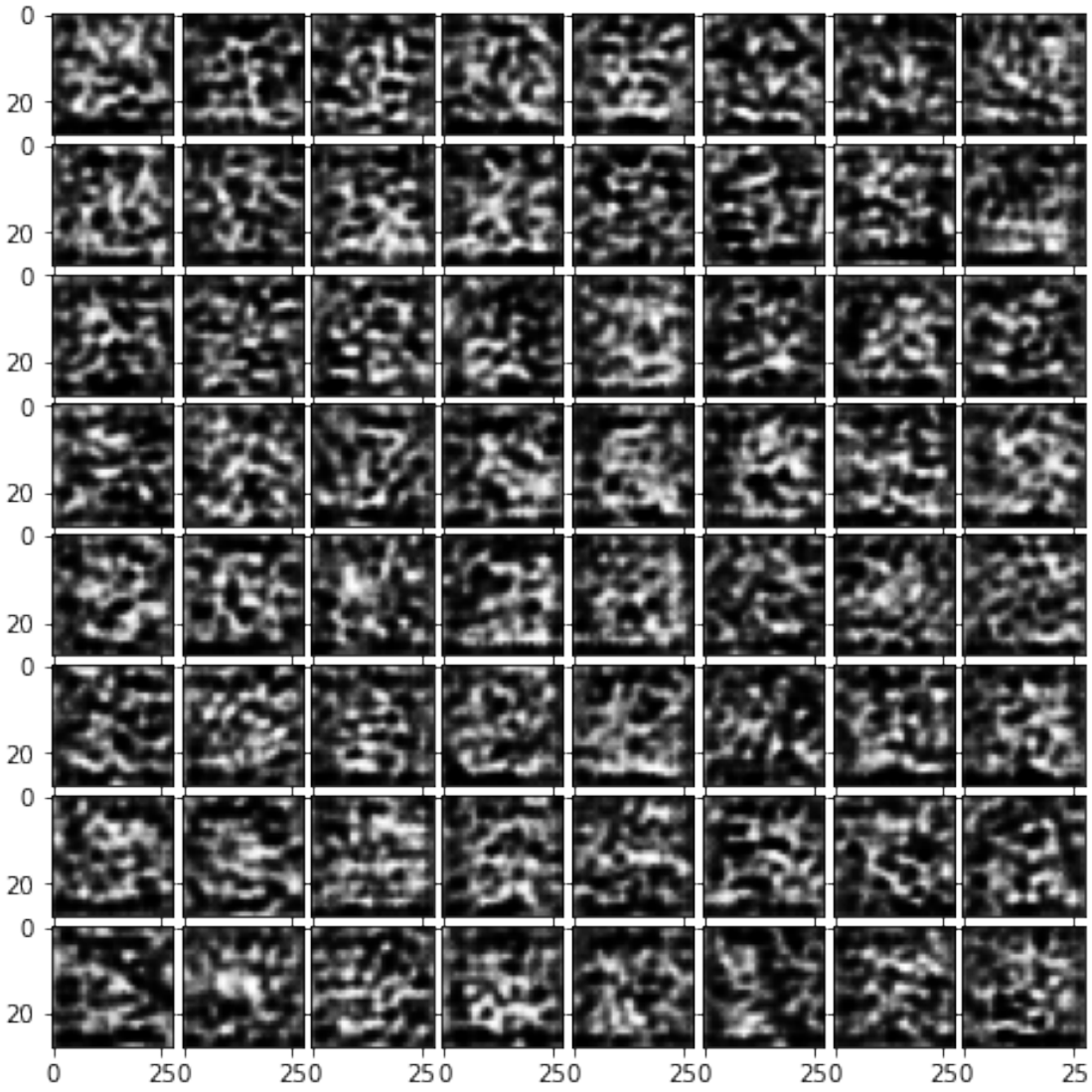
[ ]:
```
for epoch in range(epochs):
    minloss = 1
    running_kl0_loss=0
    running_recons_loss=0
    running_kl1_loss=0
```

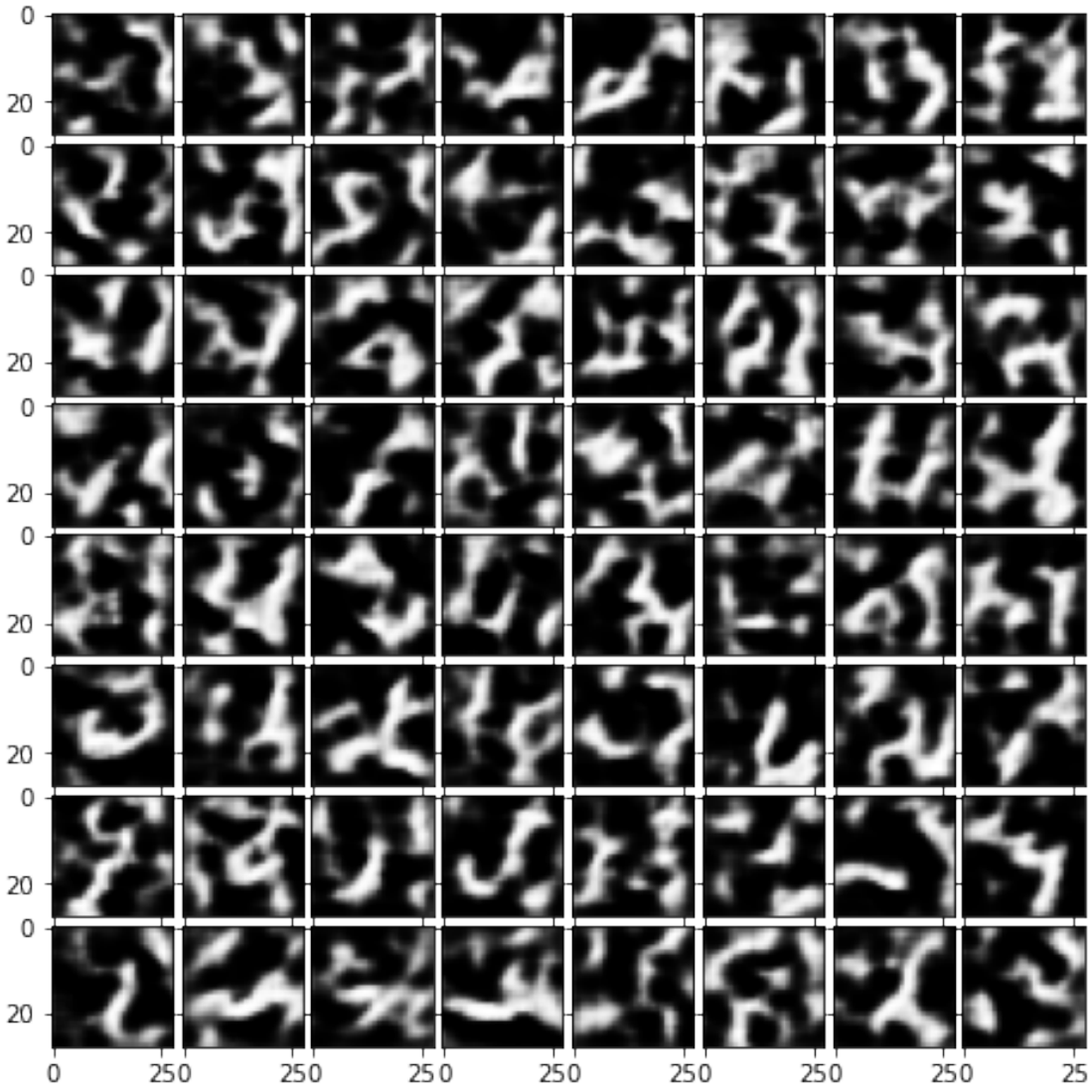
```
    num_images=0
    for i,(img,label) in enumerate(train_loader):
      img = img.to(device)
      optim.zero_grad()
      klz0,klz1,recons = model.loss(img)
      loss=recons+epoch*0.0001*klz0+epoch*0.0001*klz1
      loss.backward()
      optim.step()
      running_kl0_loss = running_kl0_loss + klz0.item()*len(img)
      running_kl1_loss = running_kl1_loss + klz1.item()*len(img)
      running_recons_loss = running_recons_loss + recons.item()*len(img)

      num_images= num_images+len(img)
    print('epoch: '+str(epoch)+' kl0_loss: '+str(running_kl0_loss/num_images)+'␣
↪recons_loss: '+str(running_recons_loss/num_images)+' kl1_loss:␣
↪'+str(running_kl1_loss/num_images))
    imgs = model.sample(64).cpu().detach().reshape(64,28,28)
    plt.gray()
    fig = plt.figure(figsize=(8., 8.))
    grid = ImageGrid(fig, 111, nrows_ncols=(8, 8), axes_pad=0.05)

    for ax, im in zip(grid, imgs):
        ax.imshow(im)
    plt.savefig(str(epoch)+".png")
```

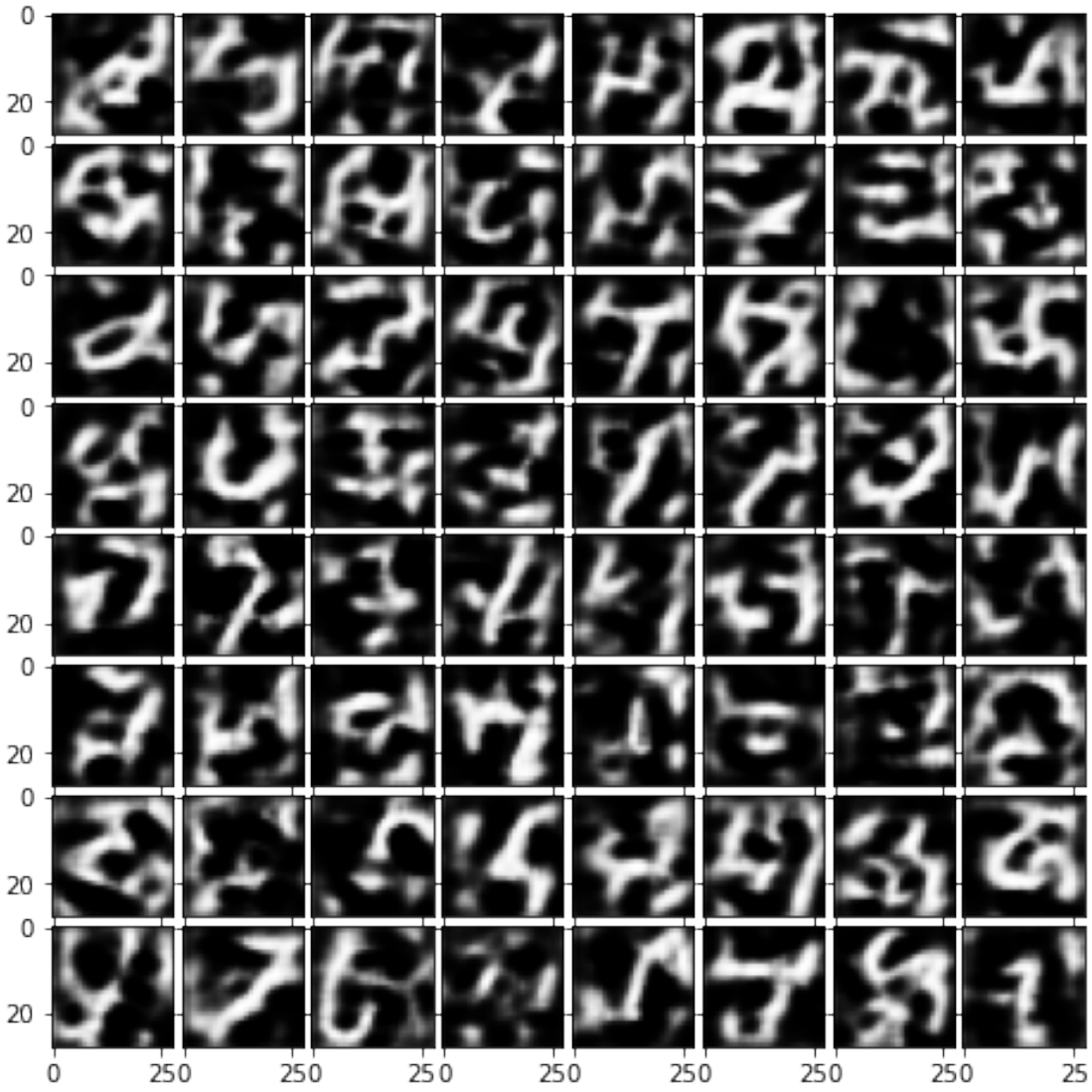
```
epoch: 0 kl0_loss: 751.782080317688 recons_loss: 0.08109162319699924 kl1_loss:
8917296.777019141
epoch: 1 kl0_loss: 570.4323500651042 recons_loss: 0.15105966951052346 kl1_loss:
38005.96190992839
epoch: 2 kl0_loss: 402.2469321289062 recons_loss: 0.14689265320301056 kl1_loss:
259.6337049641927
epoch: 3 kl0_loss: 213.27998752441405 recons_loss: 0.1471416516462962 kl1_loss:
115.2509824239095
epoch: 4 kl0_loss: 115.06704755045573 recons_loss: 0.1378506362915039 kl1_loss:
52.97208299967448
epoch: 5 kl0_loss: 73.83459227701823 recons_loss: 0.12621081261634826 kl1_loss:
22.230131729888917
epoch: 6 kl0_loss: 57.30363482259114 recons_loss: 0.1184653028011322 kl1_loss:
9.794979366556804
epoch: 7 kl0_loss: 48.309914426676436 recons_loss: 0.1144391380906105 kl1_loss:
4.822097951634725
epoch: 8 kl0_loss: 41.855485040283206 recons_loss: 0.11107408939599991 kl1_loss:
2.3236456545511883
epoch: 9 kl0_loss: 36.97051285603841 recons_loss: 0.1051474888086319 kl1_loss:
1.2710862868467967
epoch: 10 kl0_loss: 33.71670335693359 recons_loss: 0.0998356648683548 kl1_loss:
0.6939344835599264
```

```
epoch: 11 kl0_loss: 31.774424300130207 recons_loss: 0.09806553760369618
kl1_loss: 0.4290548406124115
epoch: 12 kl0_loss: 30.256083868408204 recons_loss: 0.09812449281613032
kl1_loss: 0.2950733585993449
epoch: 13 kl0_loss: 28.753330624389648 recons_loss: 0.09869587037960688
kl1_loss: 0.2302874468008677
epoch: 14 kl0_loss: 27.493141733805338 recons_loss: 0.09959638084967931
kl1_loss: 0.18545938824017844
epoch: 15 kl0_loss: 26.28852609049479 recons_loss: 0.10048087741136551 kl1_loss:
0.15588229813575744
epoch: 16 kl0_loss: 25.20113154602051 recons_loss: 0.1014794246951739 kl1_loss:
0.13966897044181822
epoch: 17 kl0_loss: 24.198837790934245 recons_loss: 0.10256501171588897
kl1_loss: 0.12259357266426087
epoch: 18 kl0_loss: 23.263466990152995 recons_loss: 0.10358741597731908
kl1_loss: 0.10926611018180847
epoch: 19 kl0_loss: 22.410162530517578 recons_loss: 0.10468868658542634
kl1_loss: 0.10576434677441915

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22: RuntimeWarning:
More than 20 figures have been opened. Figures created through the pyplot
interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and
may consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).

epoch: 20 kl0_loss: 21.57954174499512 recons_loss: 0.1056950667778651 kl1_loss:
0.09912984641393026
epoch: 21 kl0_loss: 20.86861962381999 recons_loss: 0.10672866642077763 kl1_loss:
0.09103862768809001
epoch: 22 kl0_loss: 20.19998764851888 recons_loss: 0.10775813192129136 kl1_loss:
0.08282333766619364
epoch: 23 kl0_loss: 19.529254646809896 recons_loss: 0.10877485410372416
kl1_loss: 0.07920589922269185
epoch: 24 kl0_loss: 18.907623767089845 recons_loss: 0.10982415846586227
kl1_loss: 0.07466760719617208
epoch: 25 kl0_loss: 18.370249978637695 recons_loss: 0.1106940640091896 kl1_loss:
0.07422551207542419
epoch: 26 kl0_loss: 17.837274108886717 recons_loss: 0.11170441991885503
kl1_loss: 0.0666663032690684
epoch: 27 kl0_loss: 17.28653235066732 recons_loss: 0.11273801556825638 kl1_loss:
0.06336531389554341
epoch: 28 kl0_loss: 16.822976717122394 recons_loss: 0.11374140033721925
kl1_loss: 0.06198535777727763
epoch: 29 kl0_loss: 16.36267199198405 recons_loss: 0.11461928412914277 kl1_loss:
0.06130962732632955
epoch: 30 kl0_loss: 15.941362764485676 recons_loss: 0.11557901817162831
kl1_loss: 0.058592124366760256
epoch: 31 kl0_loss: 15.480578638203939 recons_loss: 0.11652140124638875
kl1_loss: 0.05679292494455973
```

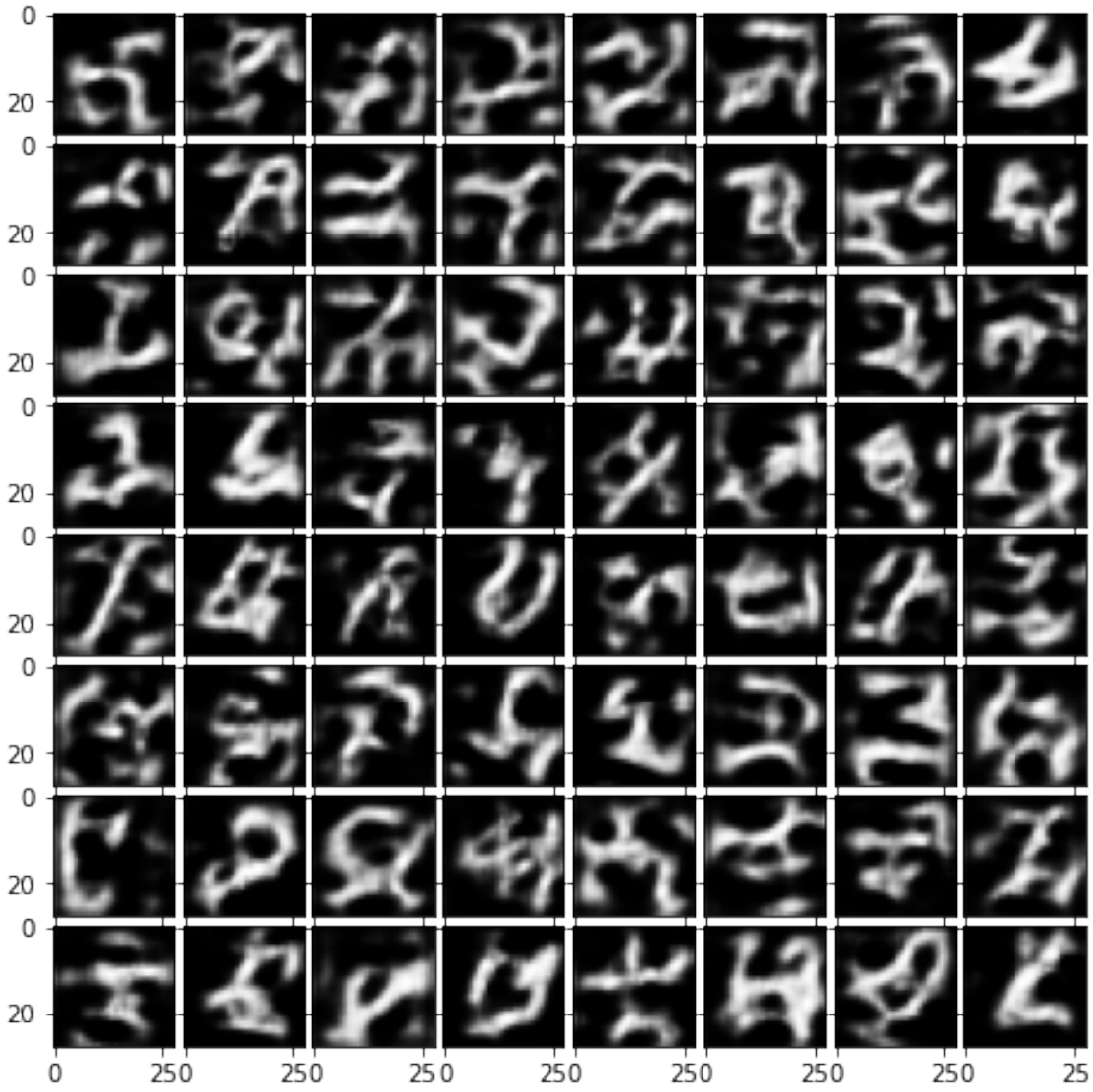
```
epoch: 32 kl0_loss: 15.145828164164225 recons_loss: 0.11731142841180166
kl1_loss: 0.05196991376876831
epoch: 33 kl0_loss: 14.746272914632161 recons_loss: 0.11820075264771779
kl1_loss: 0.05115602682431539
epoch: 34 kl0_loss: 14.374892031351726 recons_loss: 0.11906812372207641
kl1_loss: 0.04865165758132935
epoch: 35 kl0_loss: 14.046451326497396 recons_loss: 0.11994835286537806
kl1_loss: 0.04605357297261556
epoch: 36 kl0_loss: 13.687309195963541 recons_loss: 0.12078918993473053
kl1_loss: 0.044615216700236
epoch: 37 kl0_loss: 13.387060451253255 recons_loss: 0.12152494059006373
kl1_loss: 0.04352628908157349
epoch: 38 kl0_loss: 13.072342435709636 recons_loss: 0.12241315124432246
kl1_loss: 0.04461188589731852
epoch: 39 kl0_loss: 12.81632827351888 recons_loss: 0.12308639443715413 kl1_loss:
0.04126805640856425
epoch: 40 kl0_loss: 12.538095556132 recons_loss: 0.12386690600713095 kl1_loss:
0.03961378099123637
epoch: 41 kl0_loss: 12.294520878092447 recons_loss: 0.12465567227204641
kl1_loss: 0.04051355160077413
epoch: 42 kl0_loss: 12.053678917948405 recons_loss: 0.12528542629877726
kl1_loss: 0.038296764373779296
epoch: 43 kl0_loss: 11.83693717549642 recons_loss: 0.12615564269224802 kl1_loss:
0.03759724310239156
epoch: 44 kl0_loss: 11.628213116455077 recons_loss: 0.1268004652897517 kl1_loss:
0.0377593921661377
epoch: 45 kl0_loss: 11.384304349263509 recons_loss: 0.12758560822804768
kl1_loss: 0.03565698954264323
epoch: 46 kl0_loss: 11.206871402994791 recons_loss: 0.12825051782131194
kl1_loss: 0.03503577399253845
epoch: 47 kl0_loss: 11.008865227254232 recons_loss: 0.12891203769048054
kl1_loss: 0.03407250299453735
epoch: 48 kl0_loss: 10.820389528401693 recons_loss: 0.12958042811552684
kl1_loss: 0.032992552121480306
epoch: 49 kl0_loss: 10.675146875 recons_loss: 0.13029568121433258 kl1_loss:
0.03283892561594645

<Figure size 432x288 with 0 Axes>
```

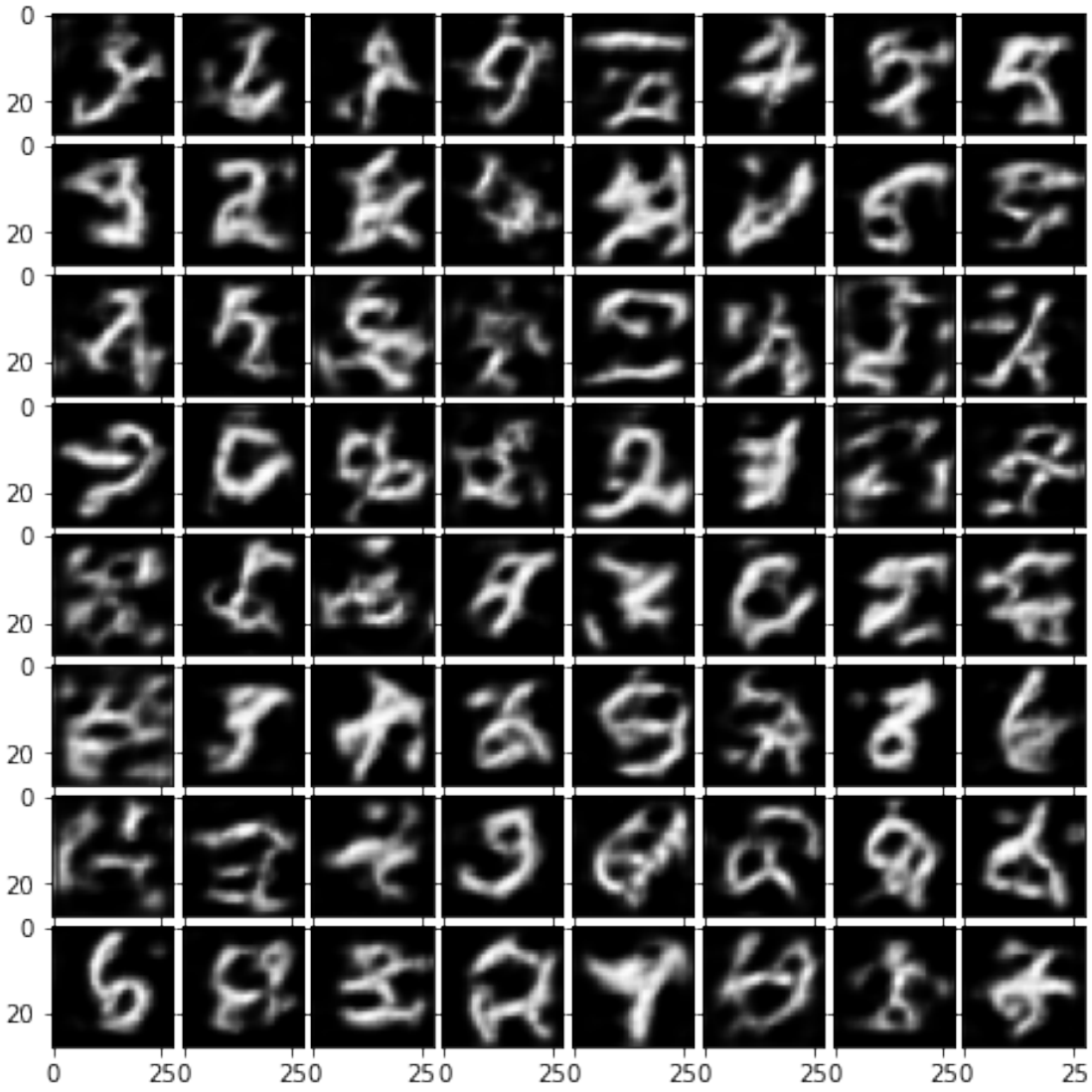

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

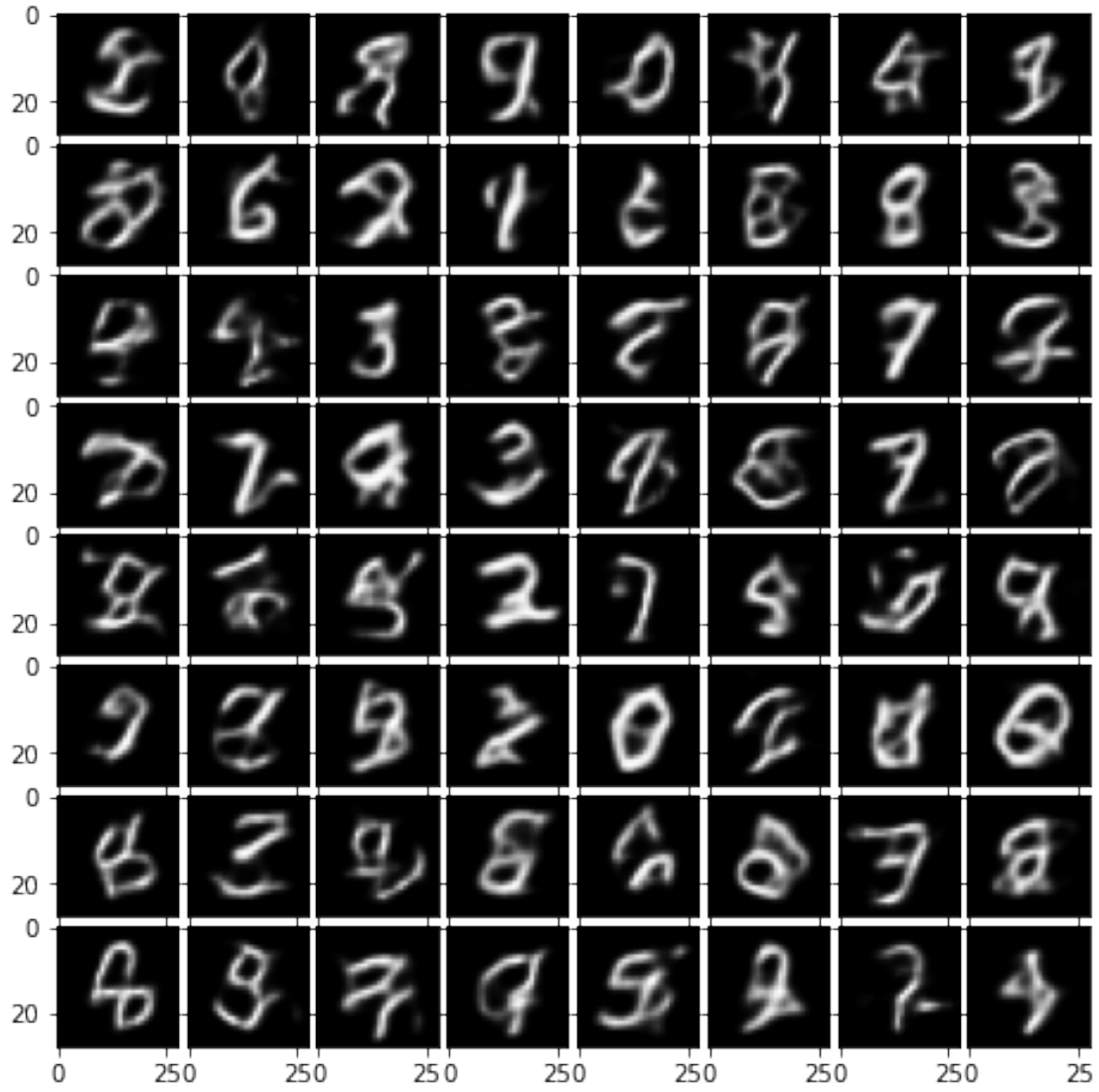

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

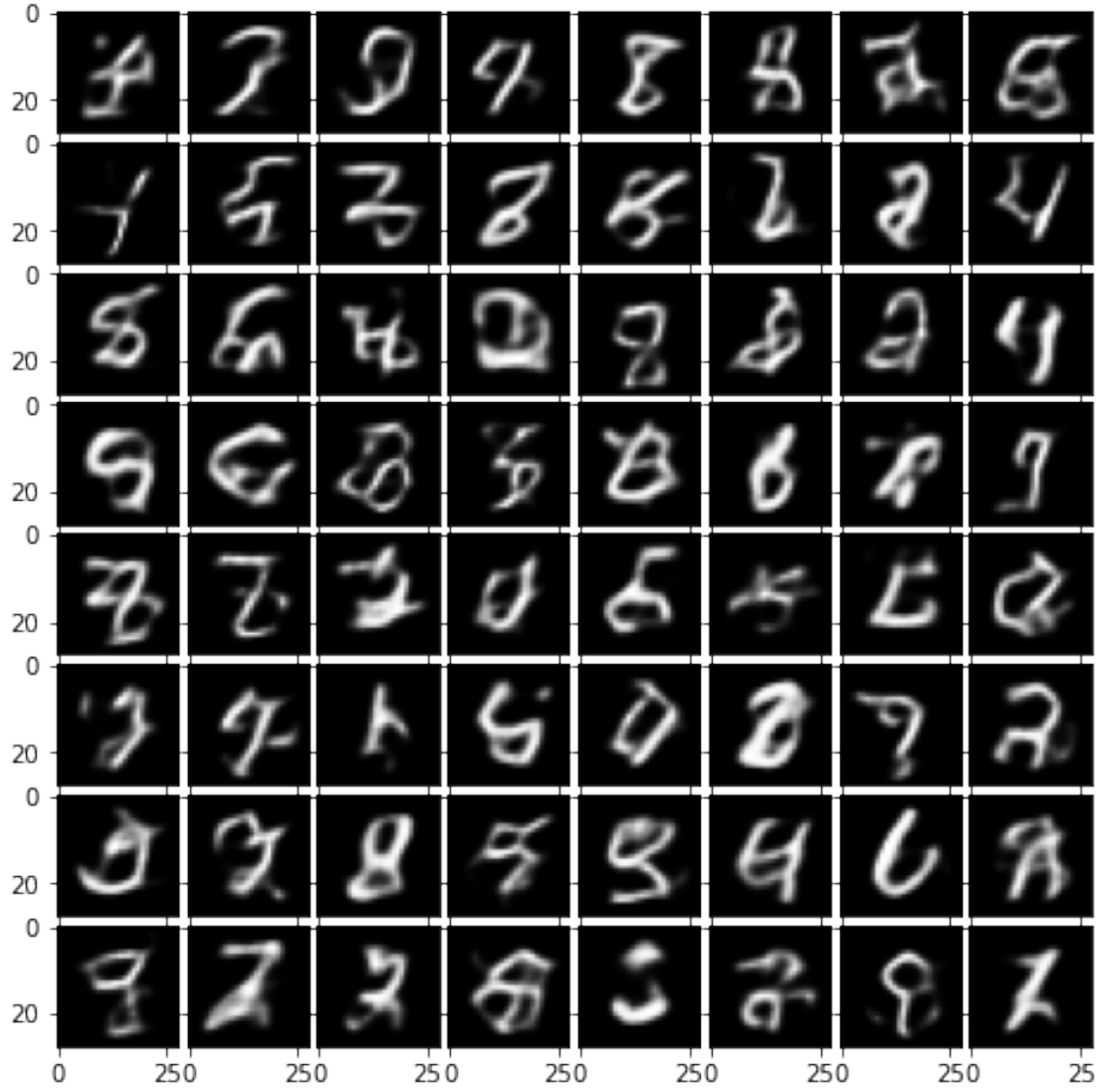

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

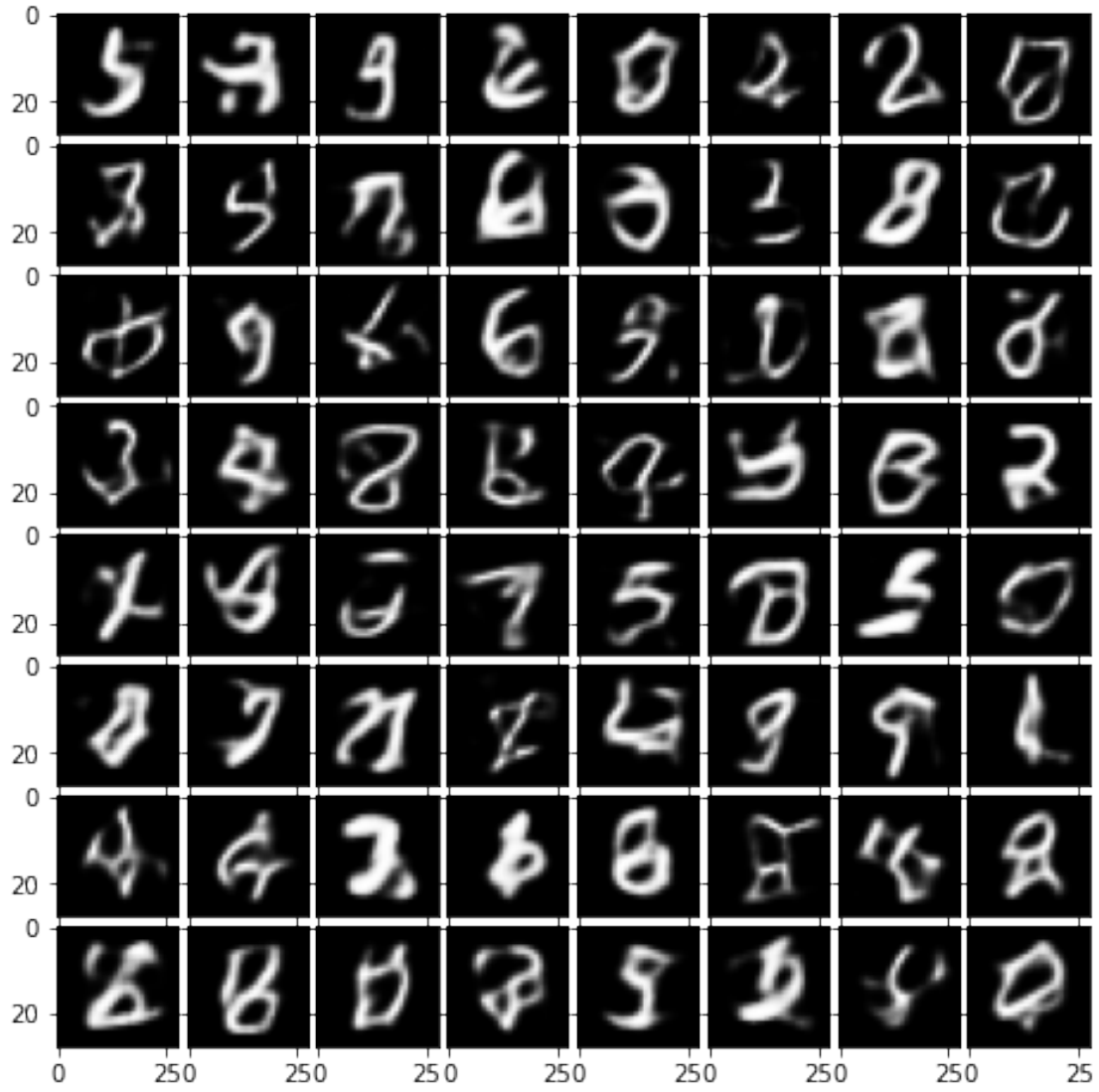

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

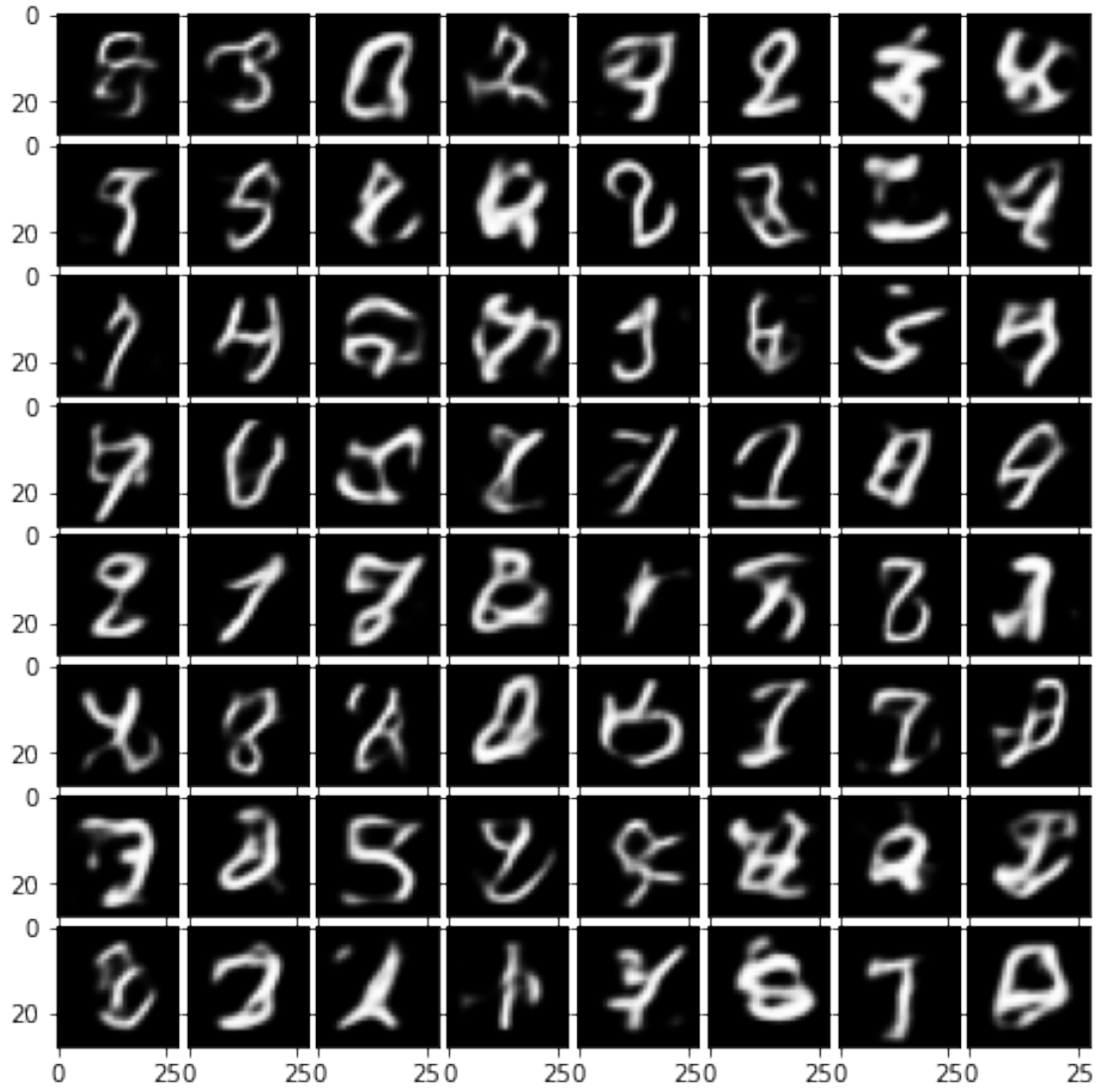

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

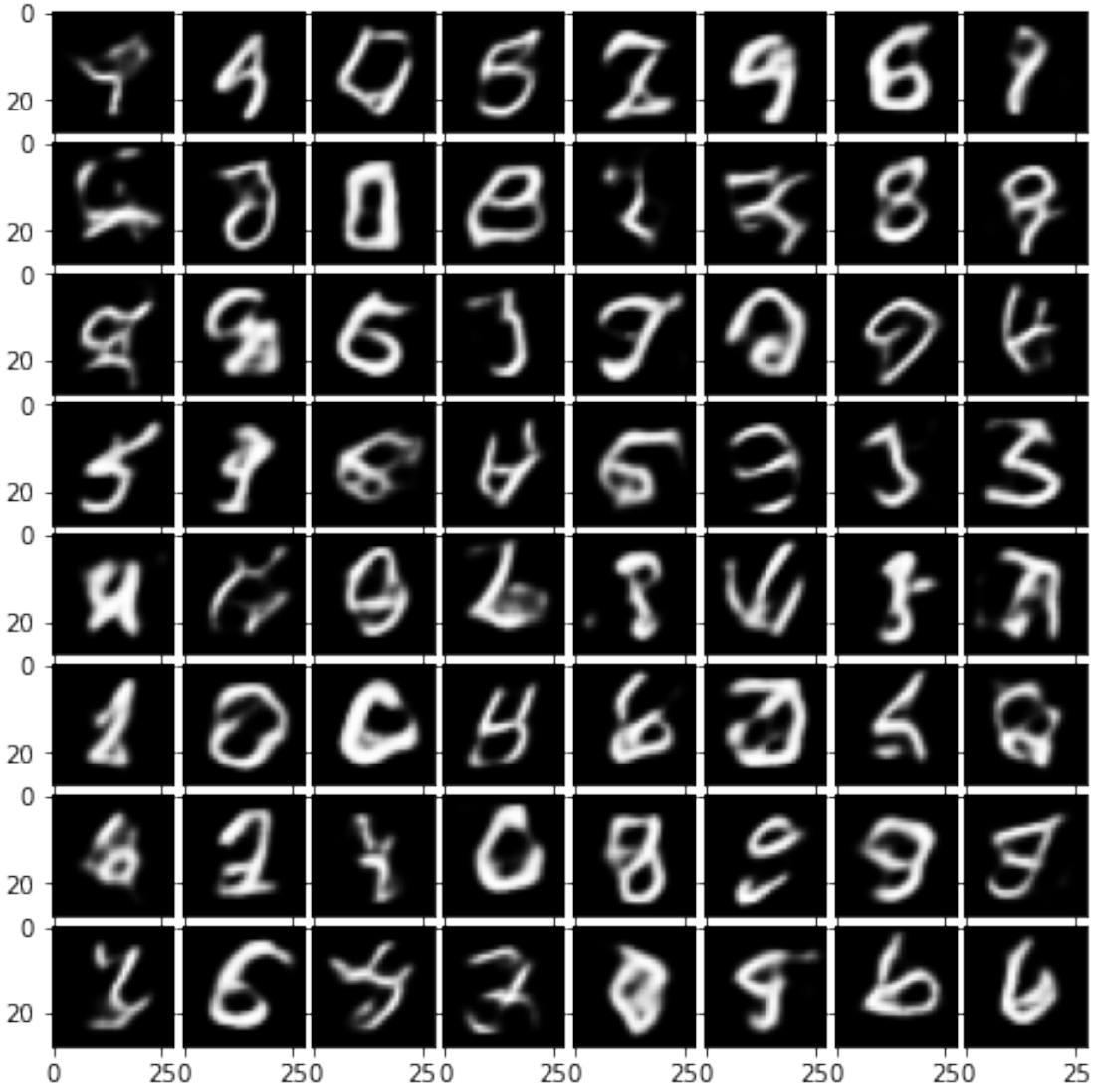

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

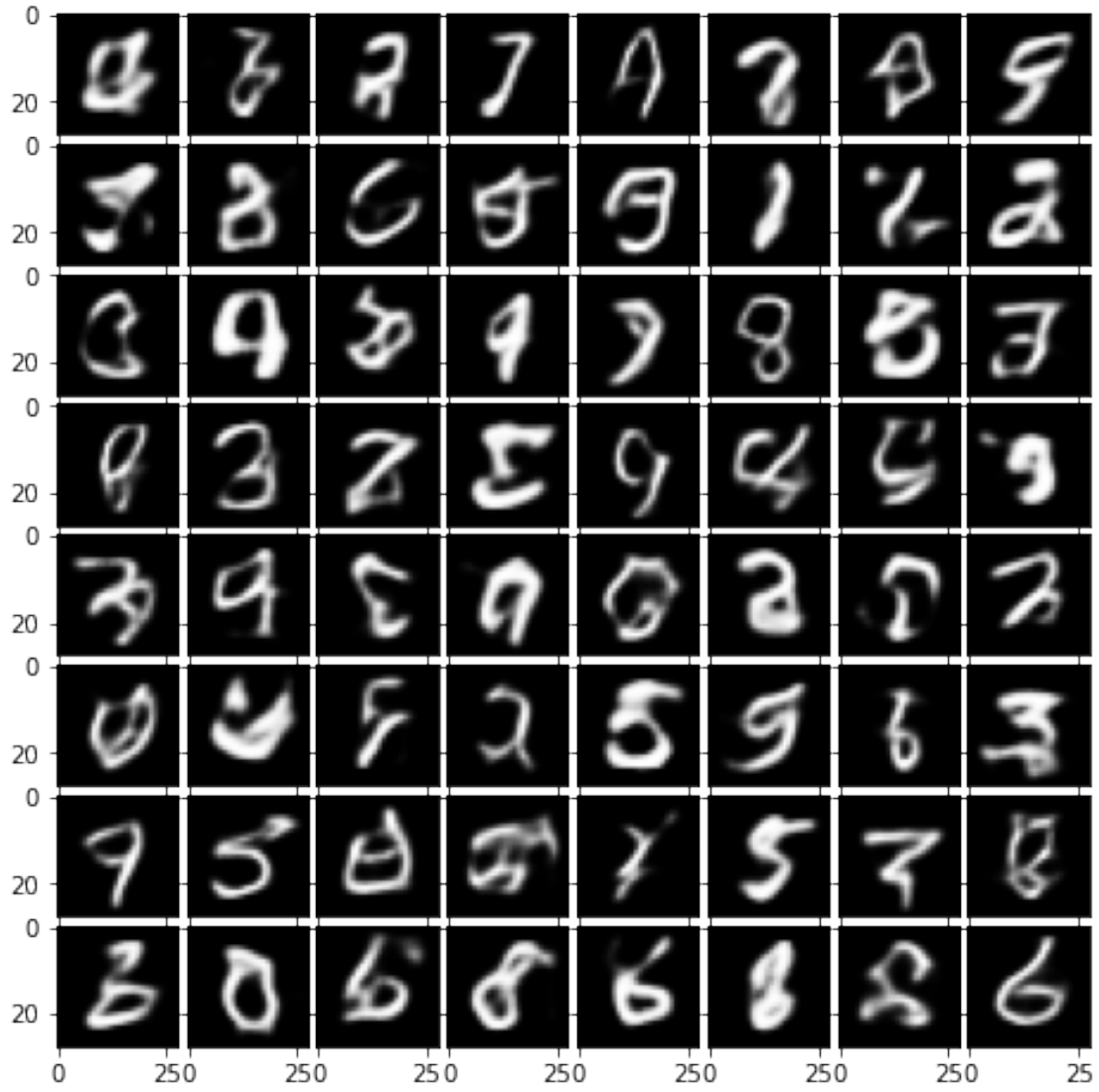

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

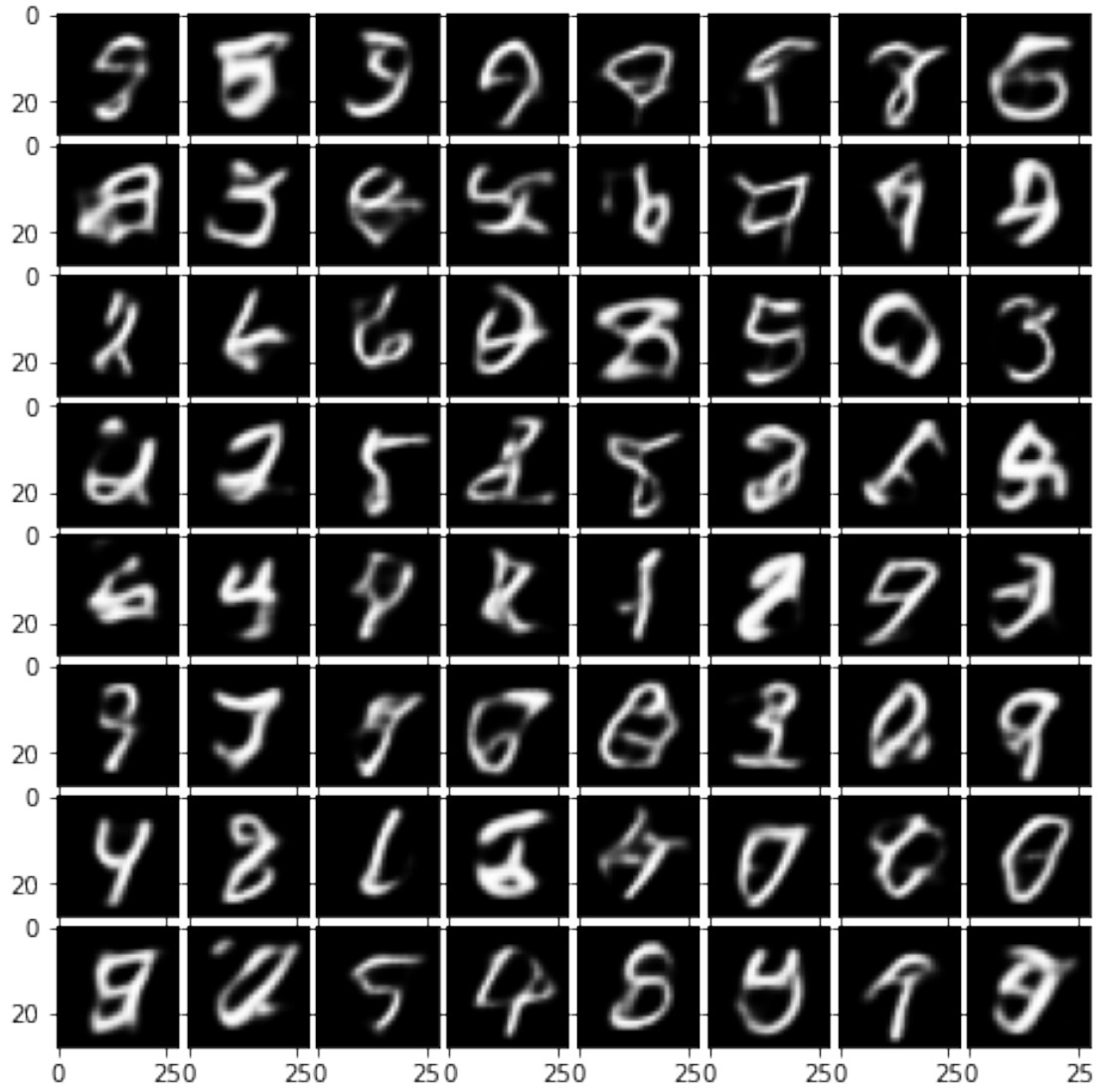

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

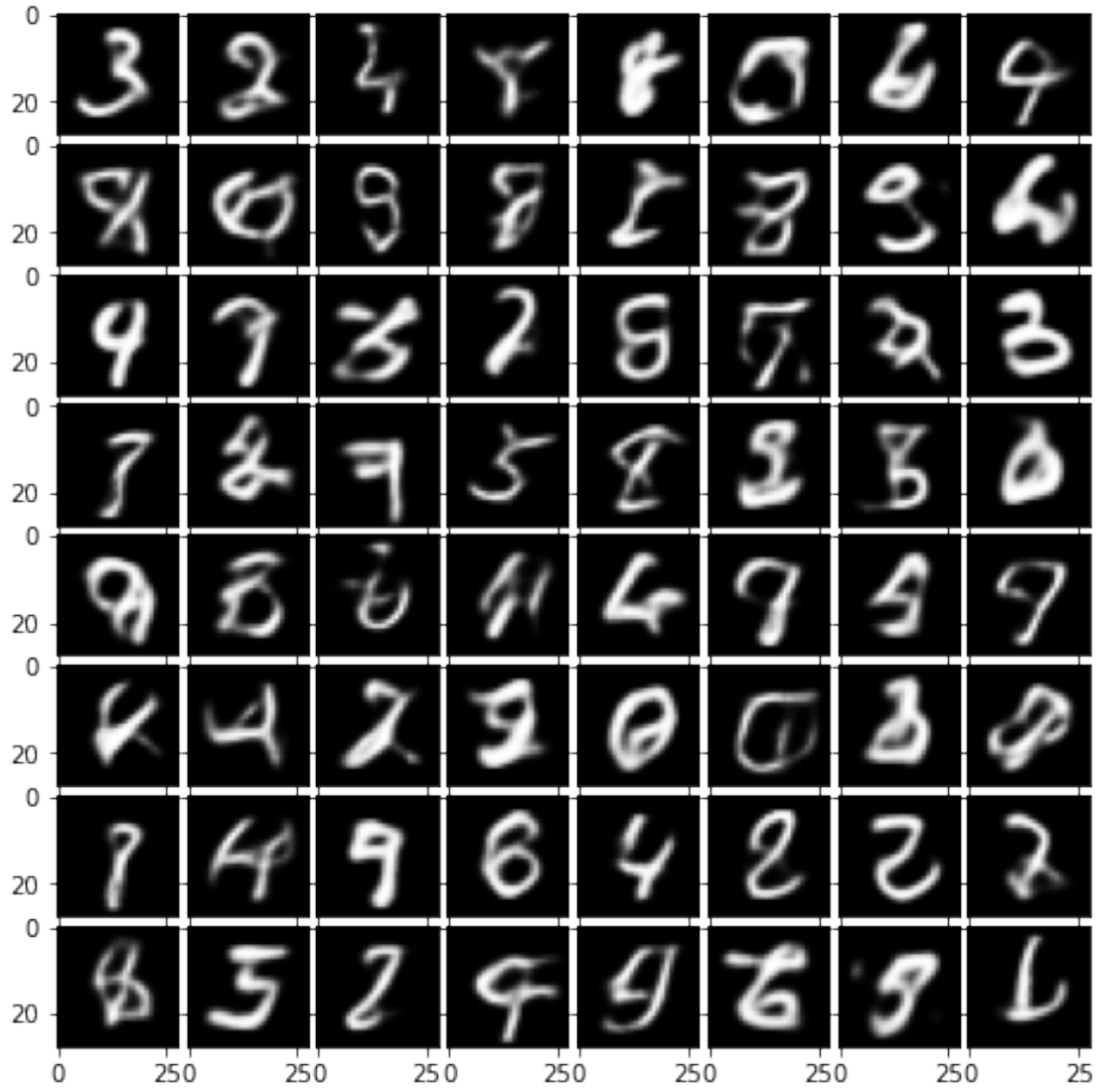

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

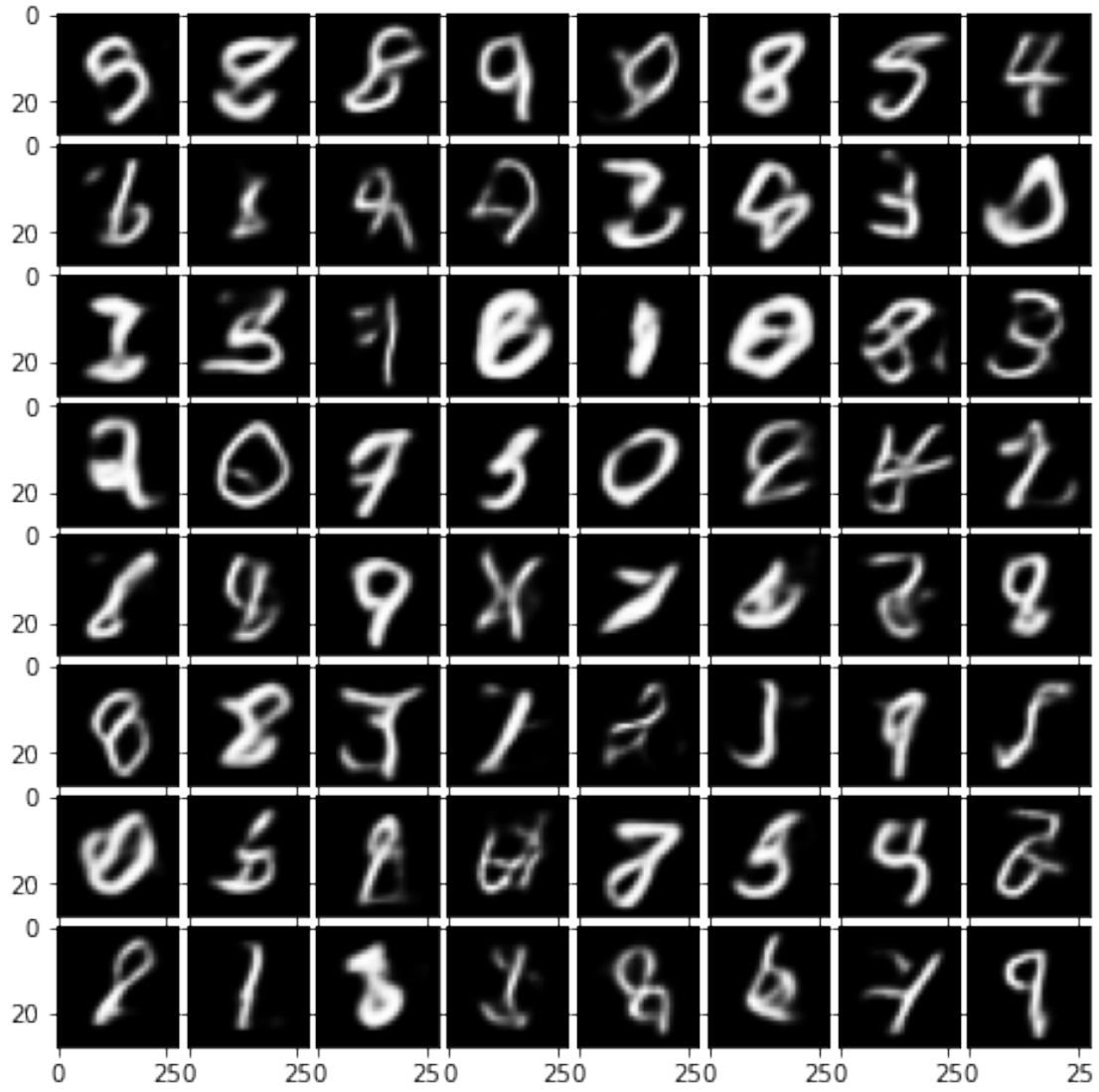

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

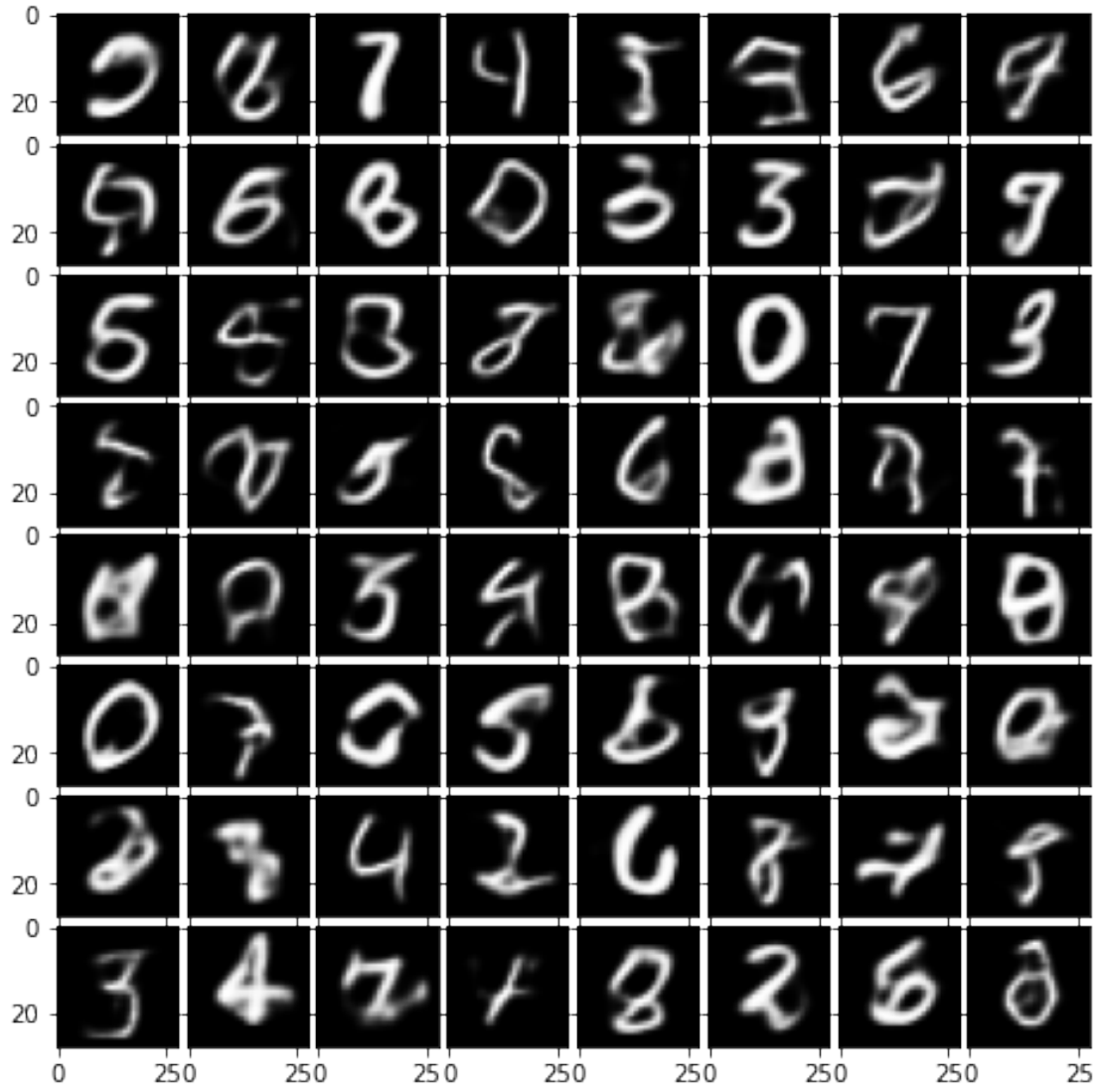

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

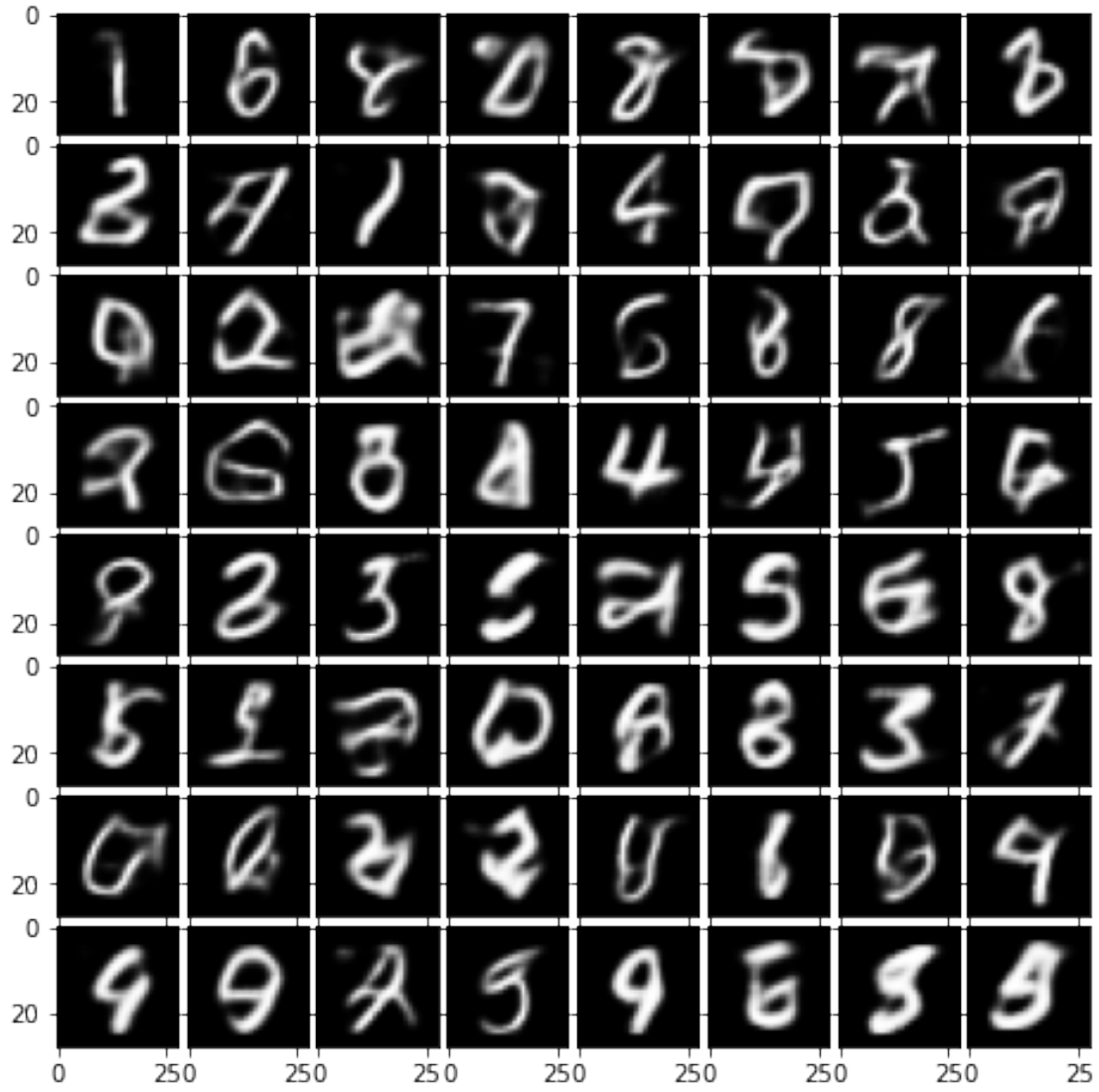

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

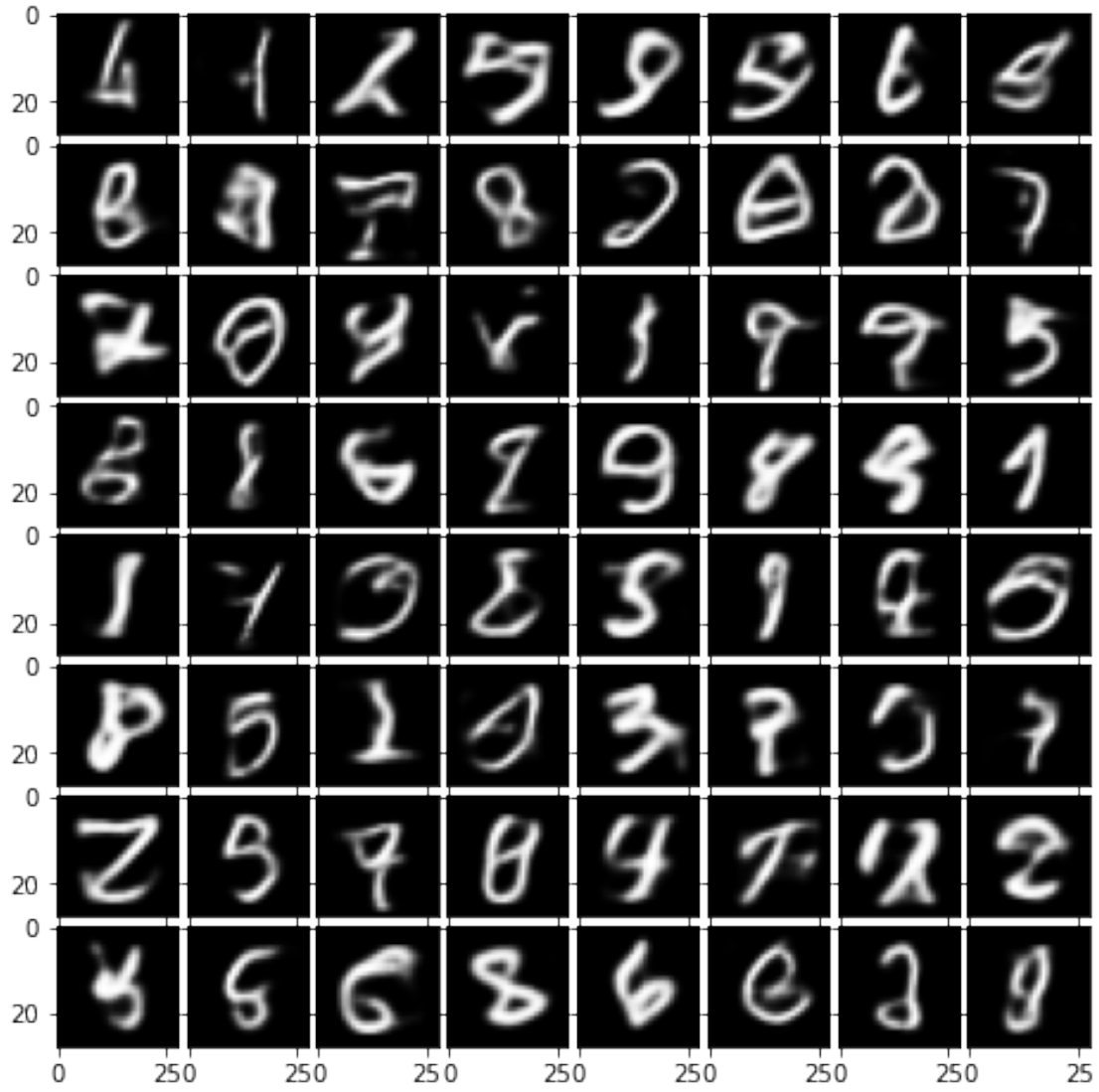

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

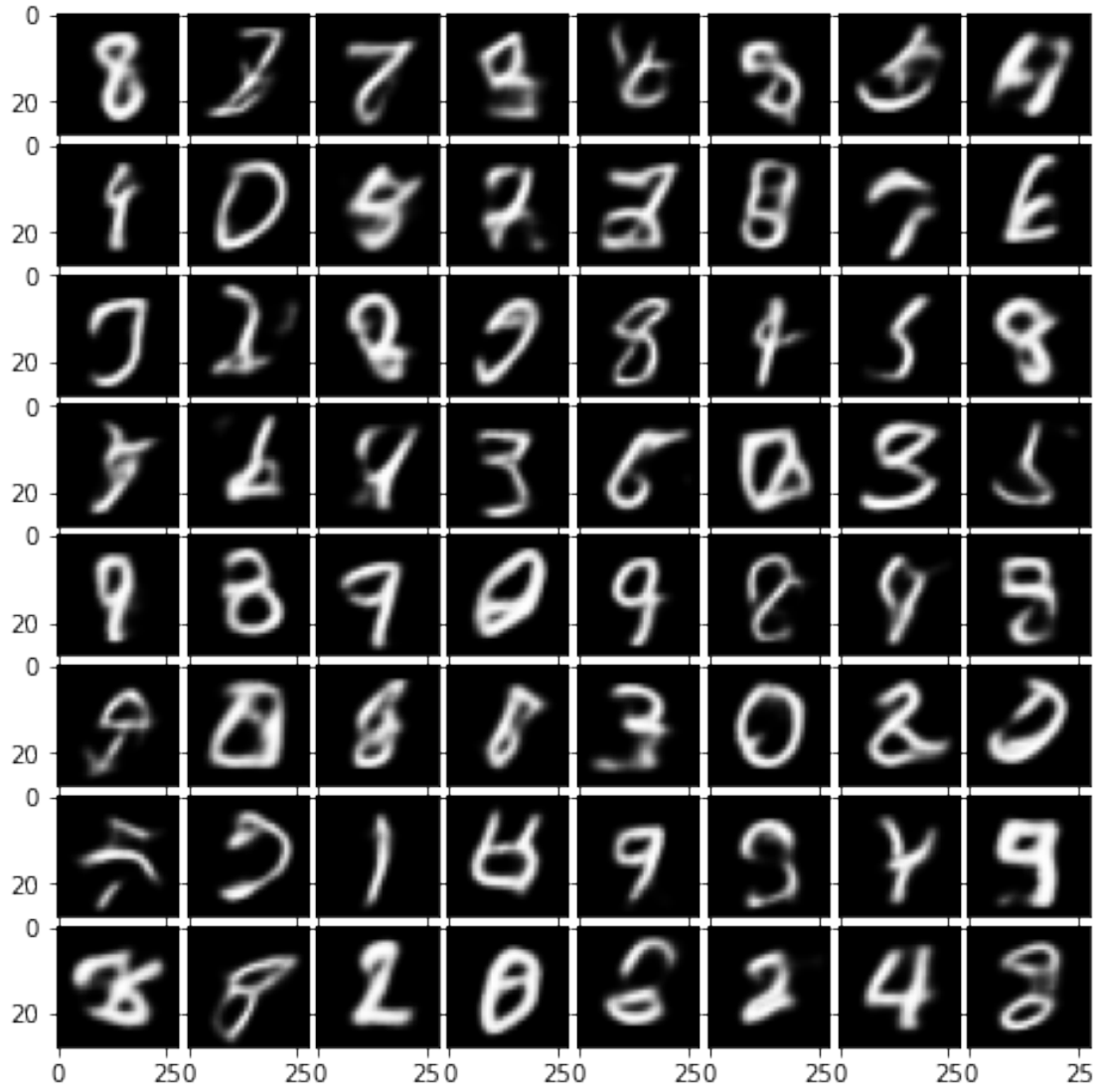

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

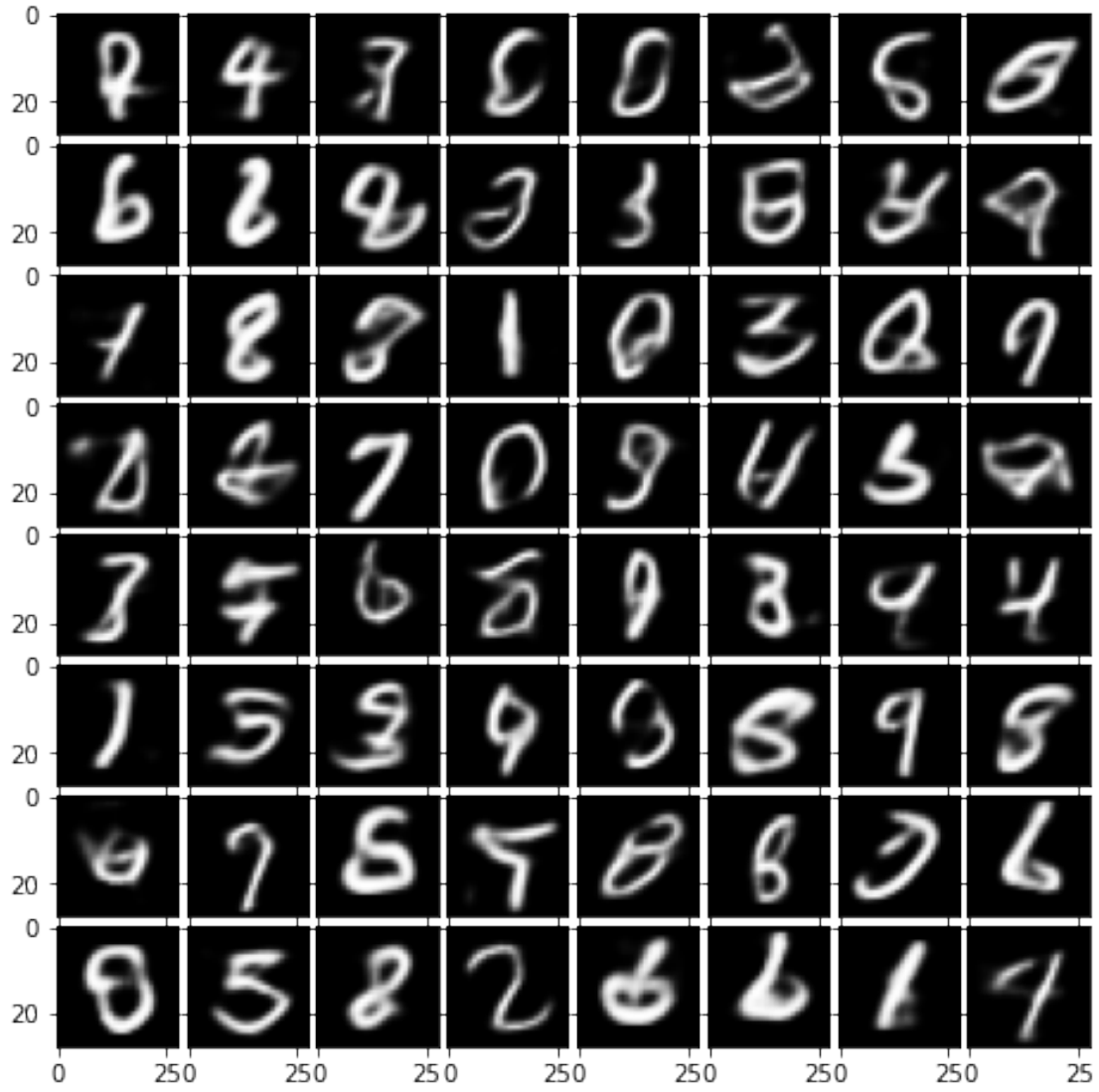

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

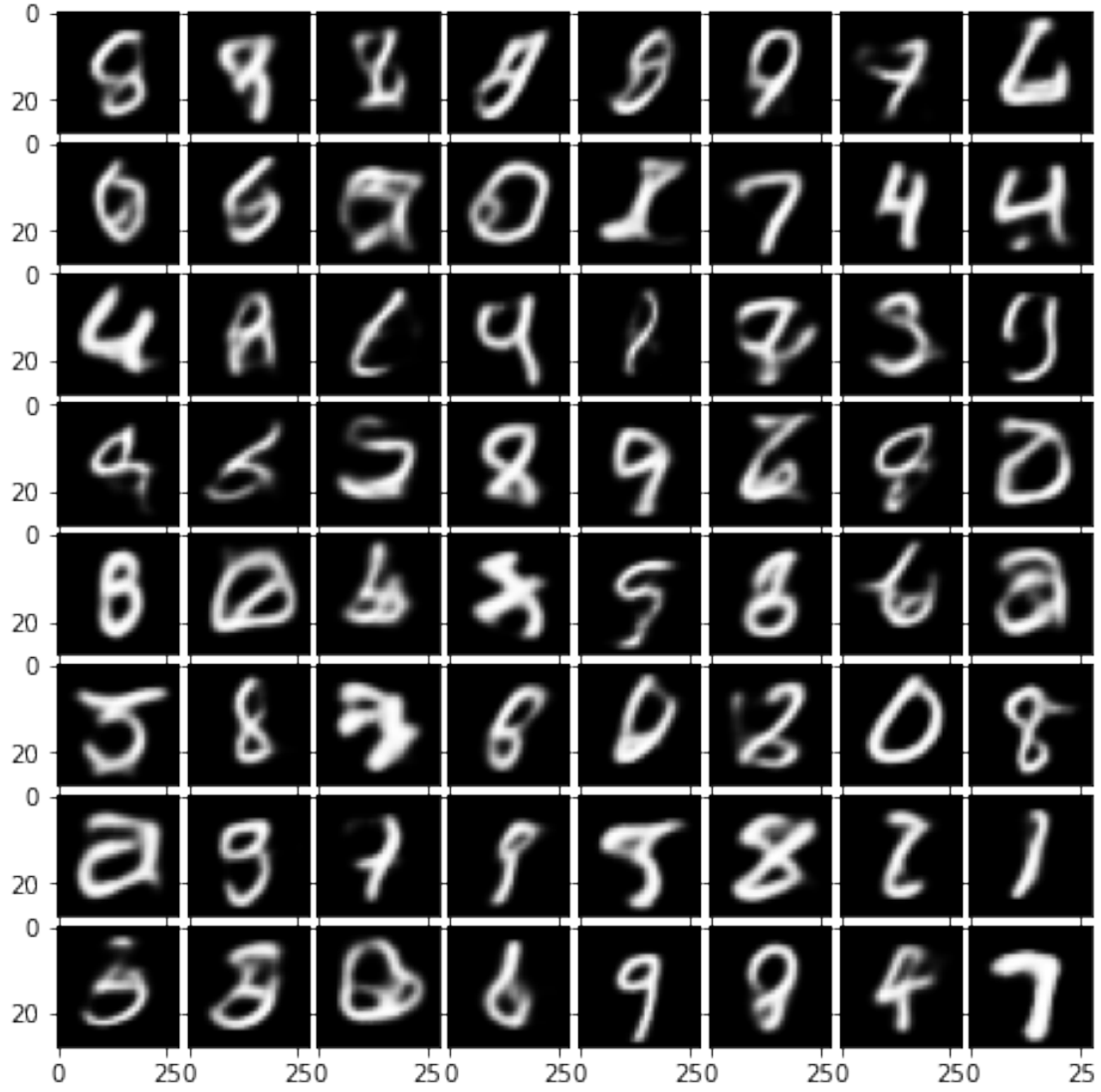

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

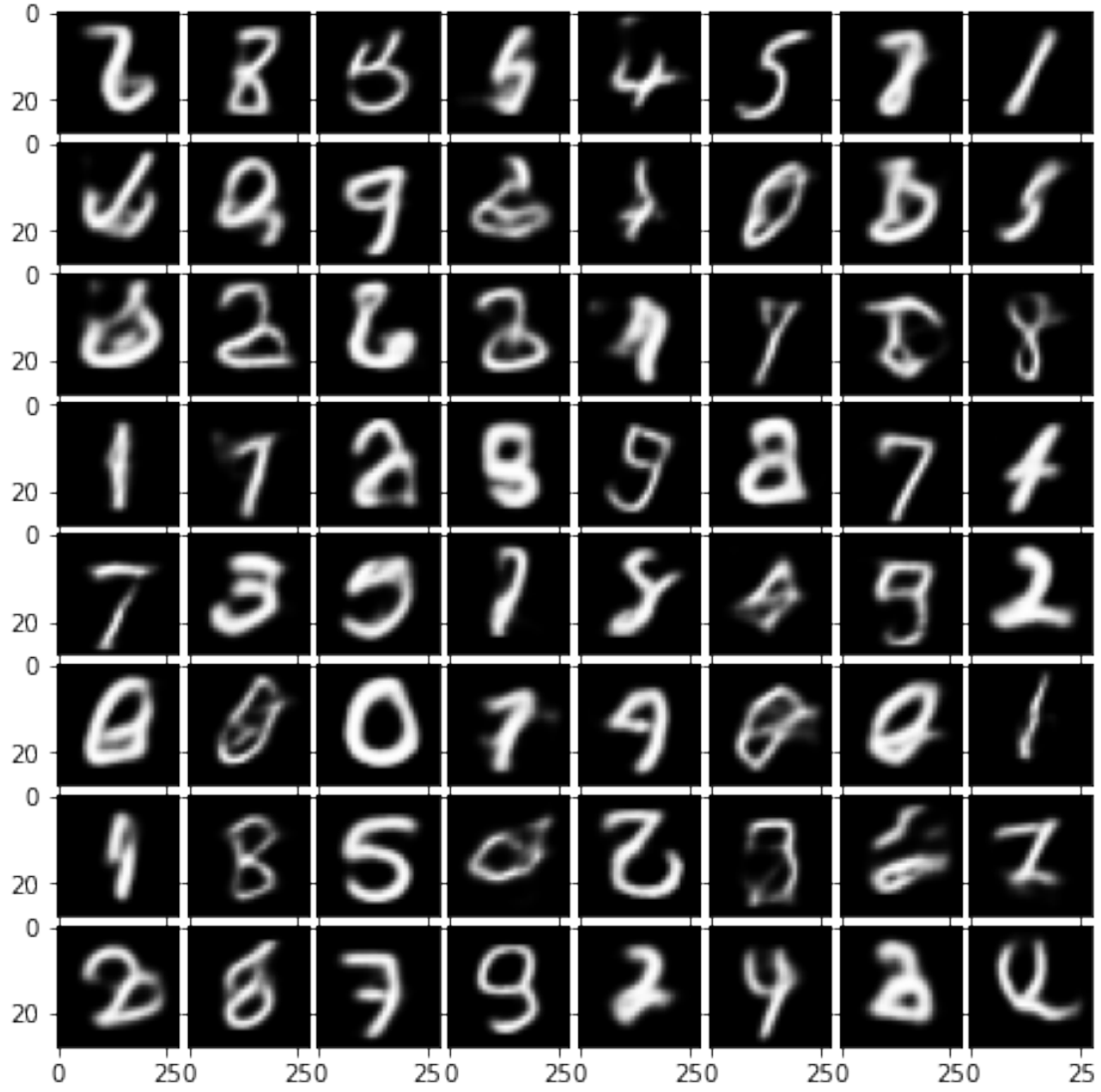

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

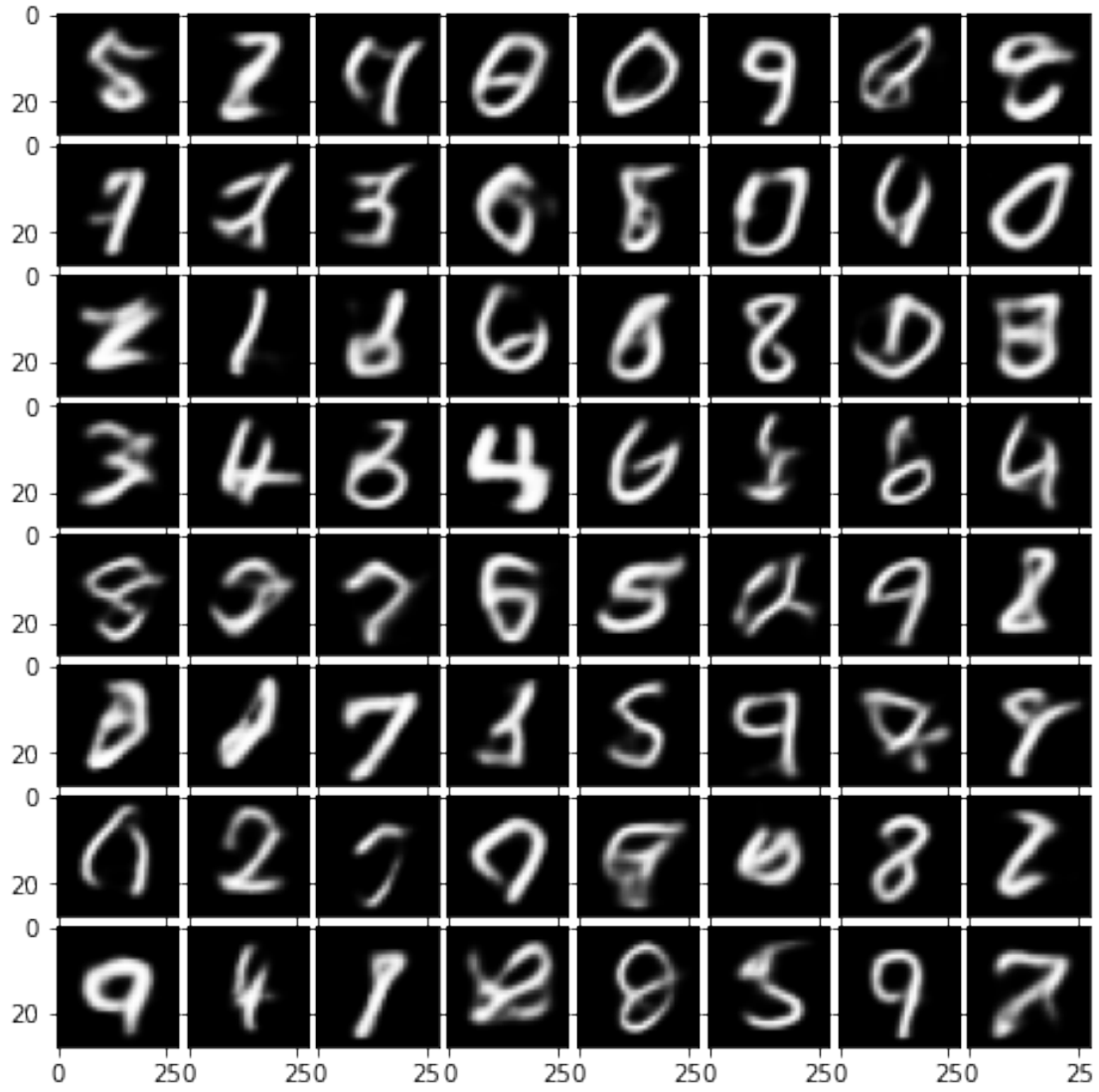

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

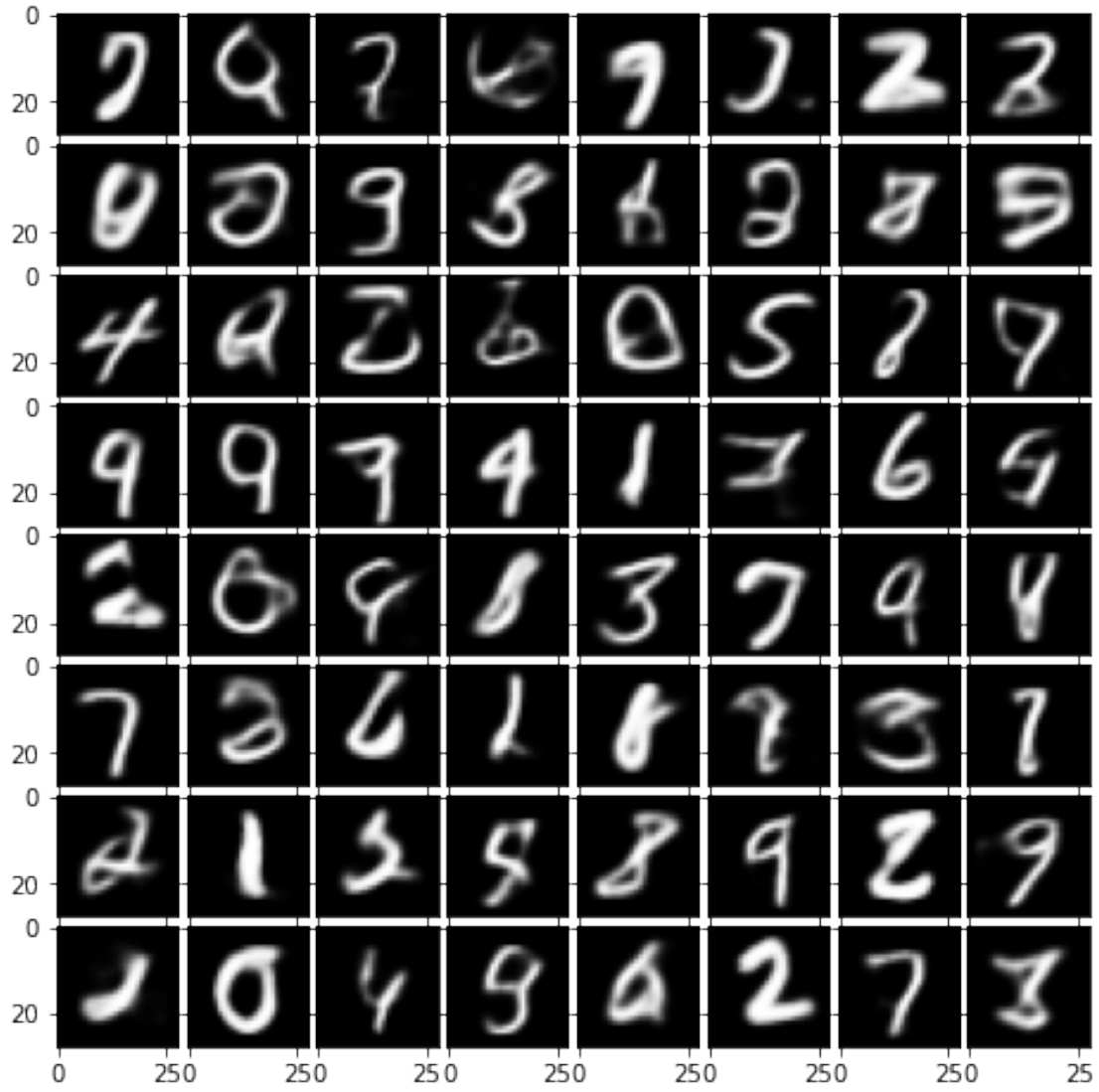

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

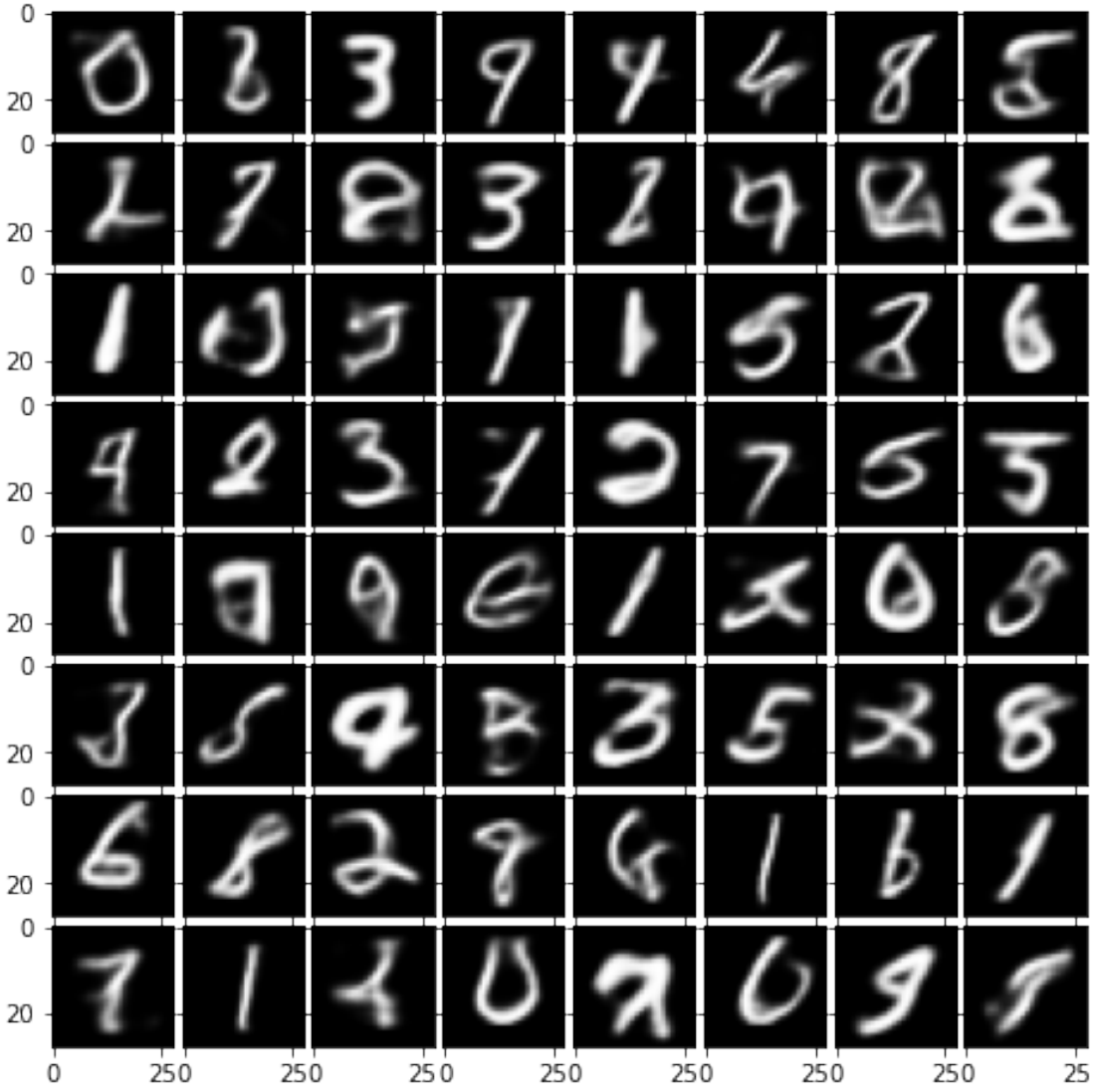

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

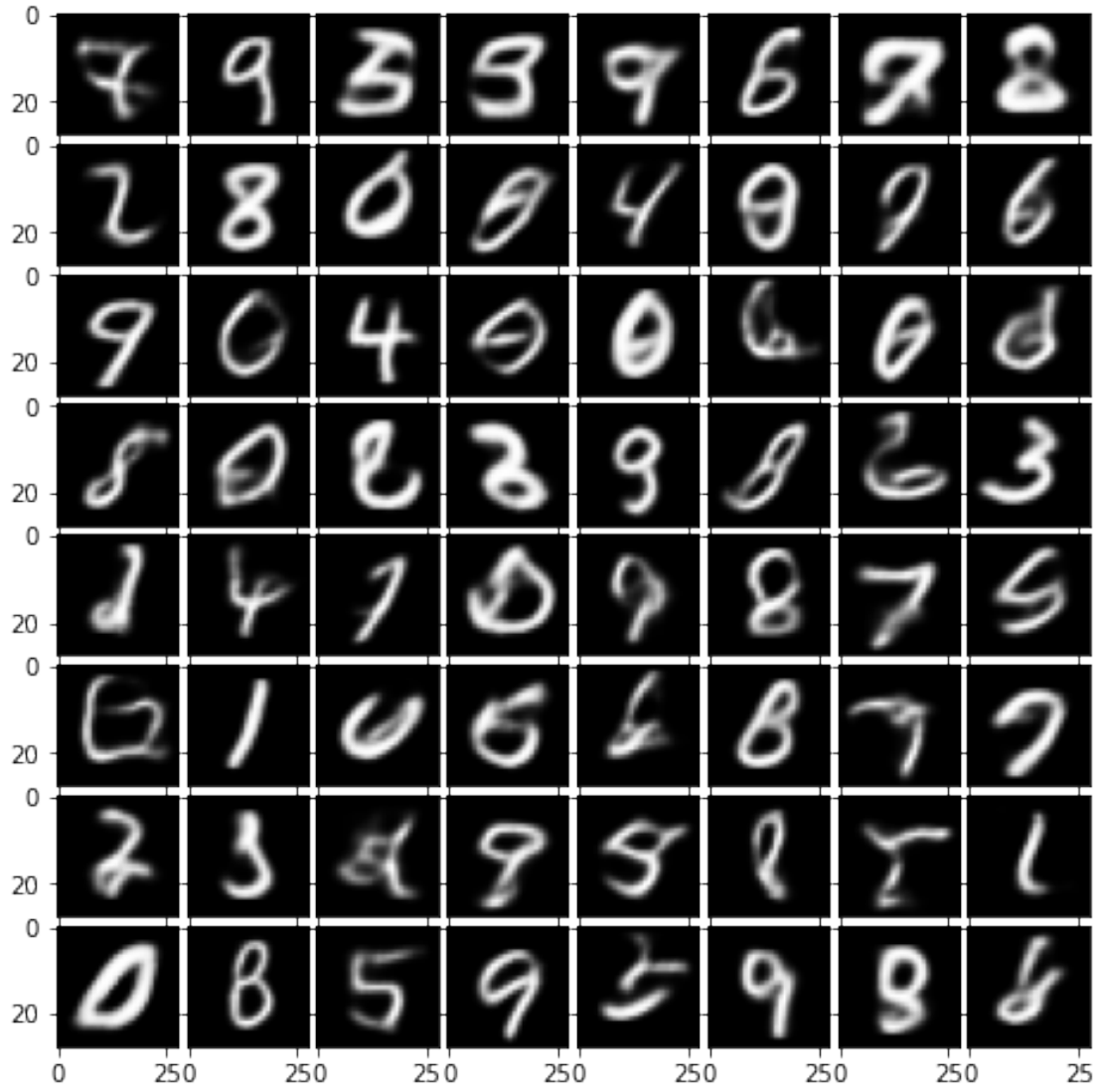

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

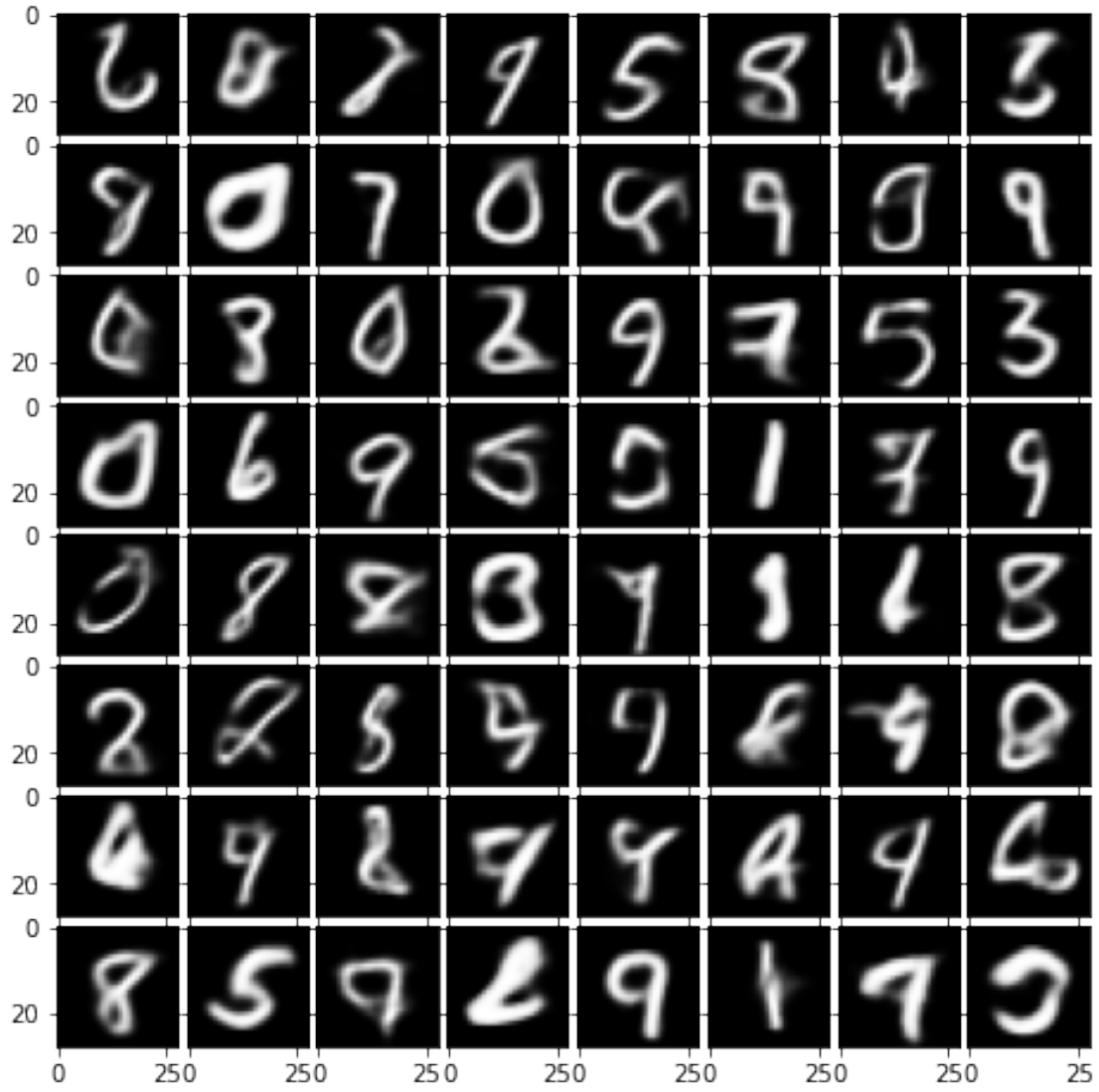

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

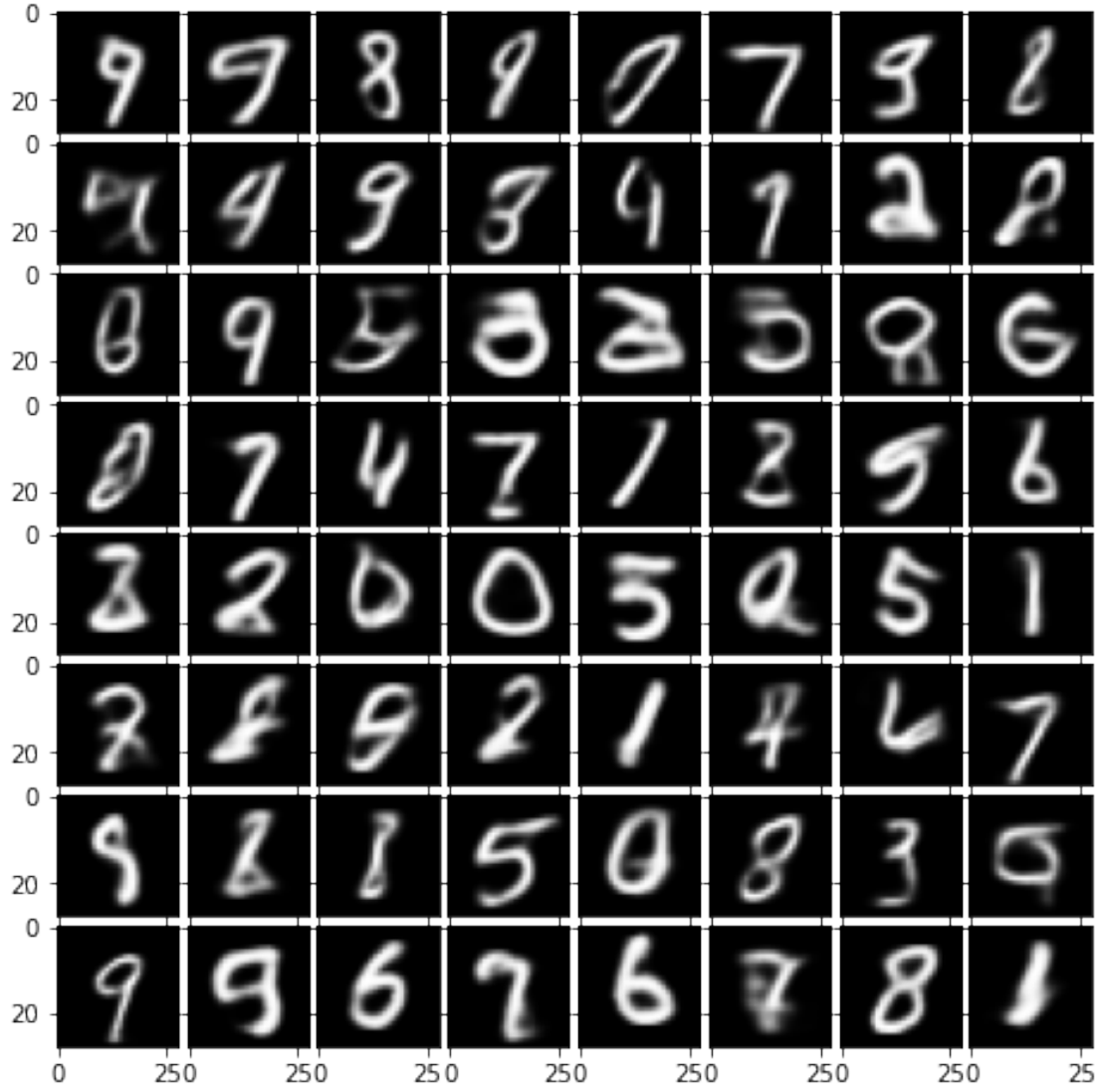

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

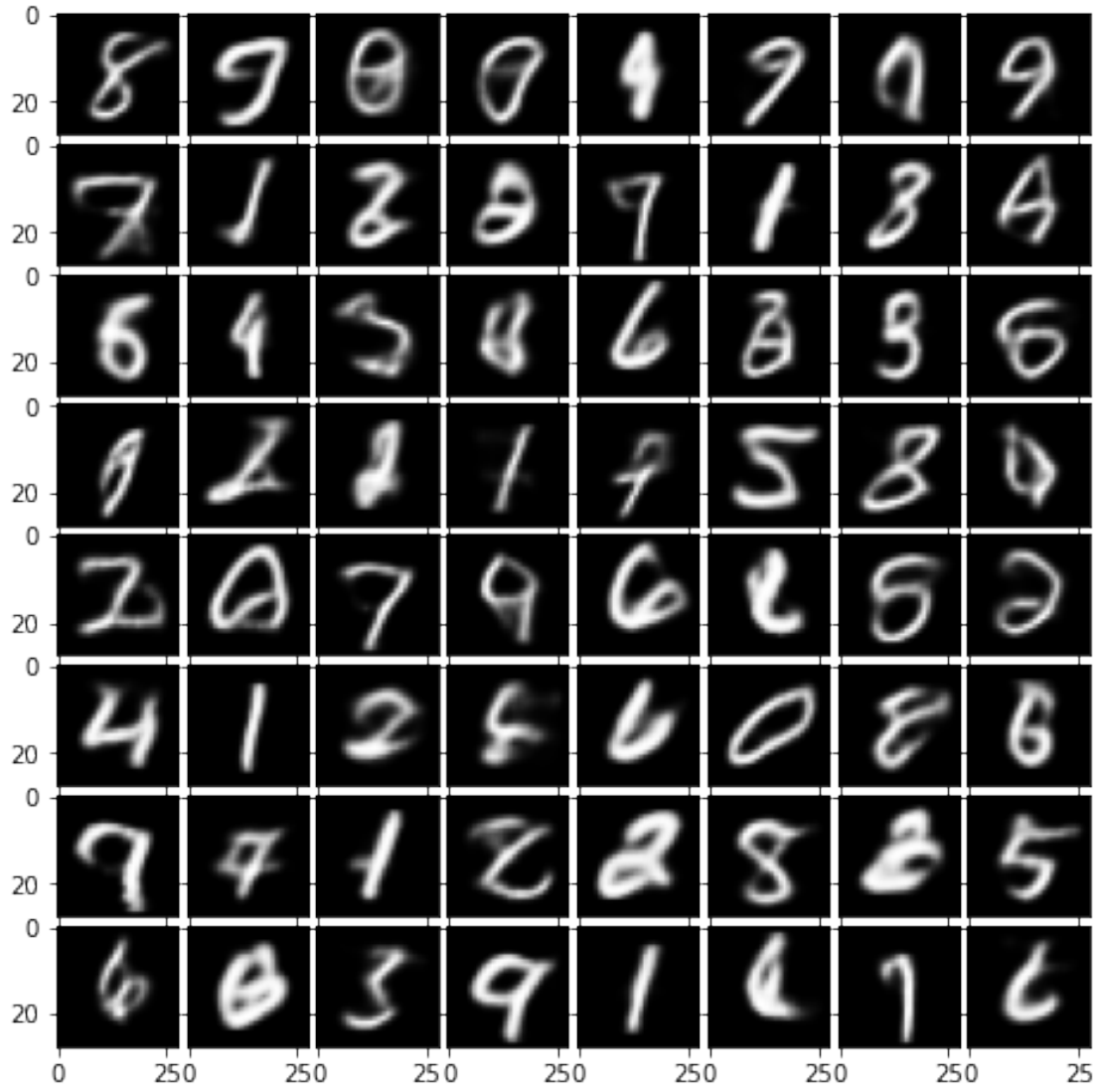

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

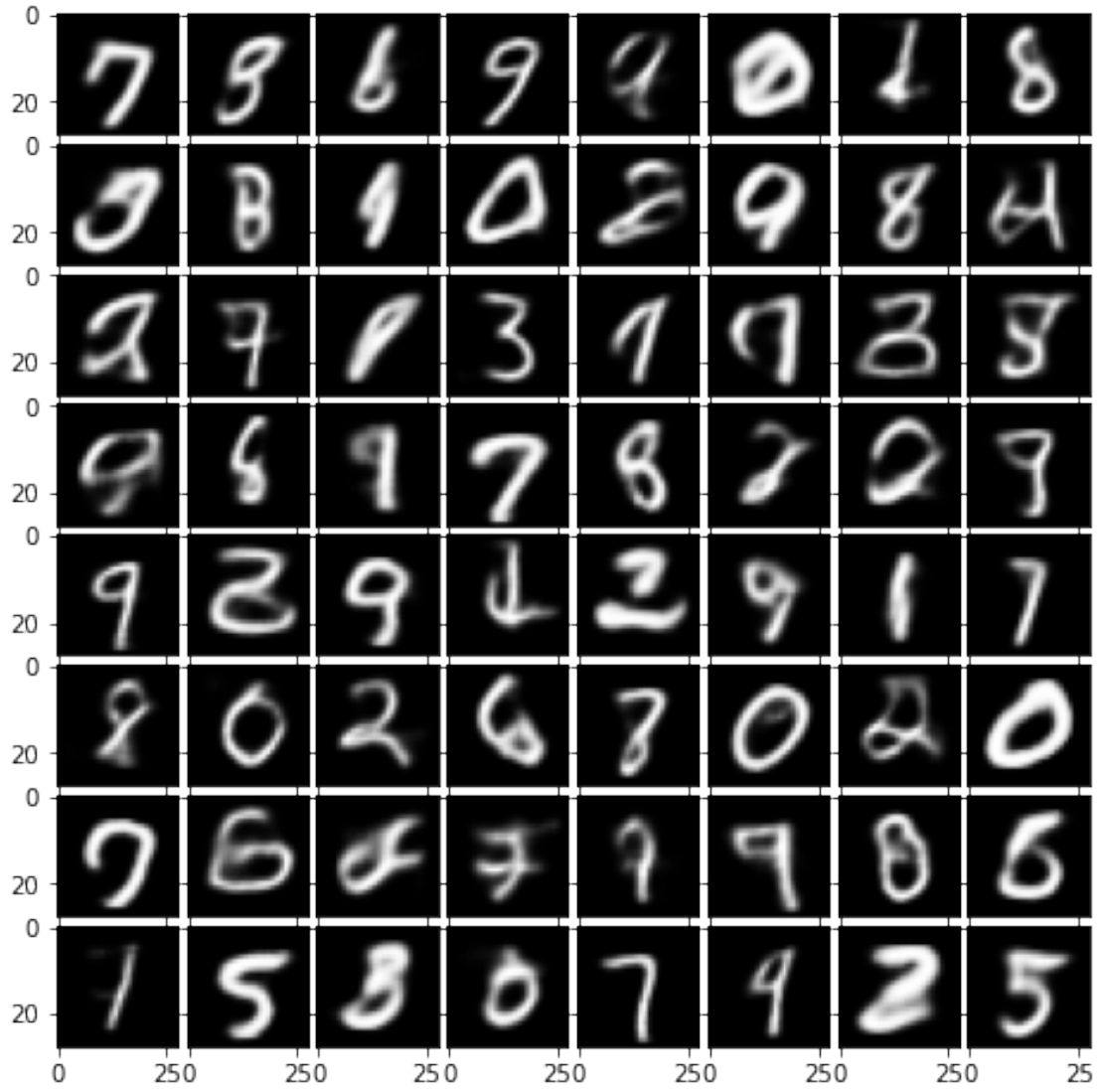

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

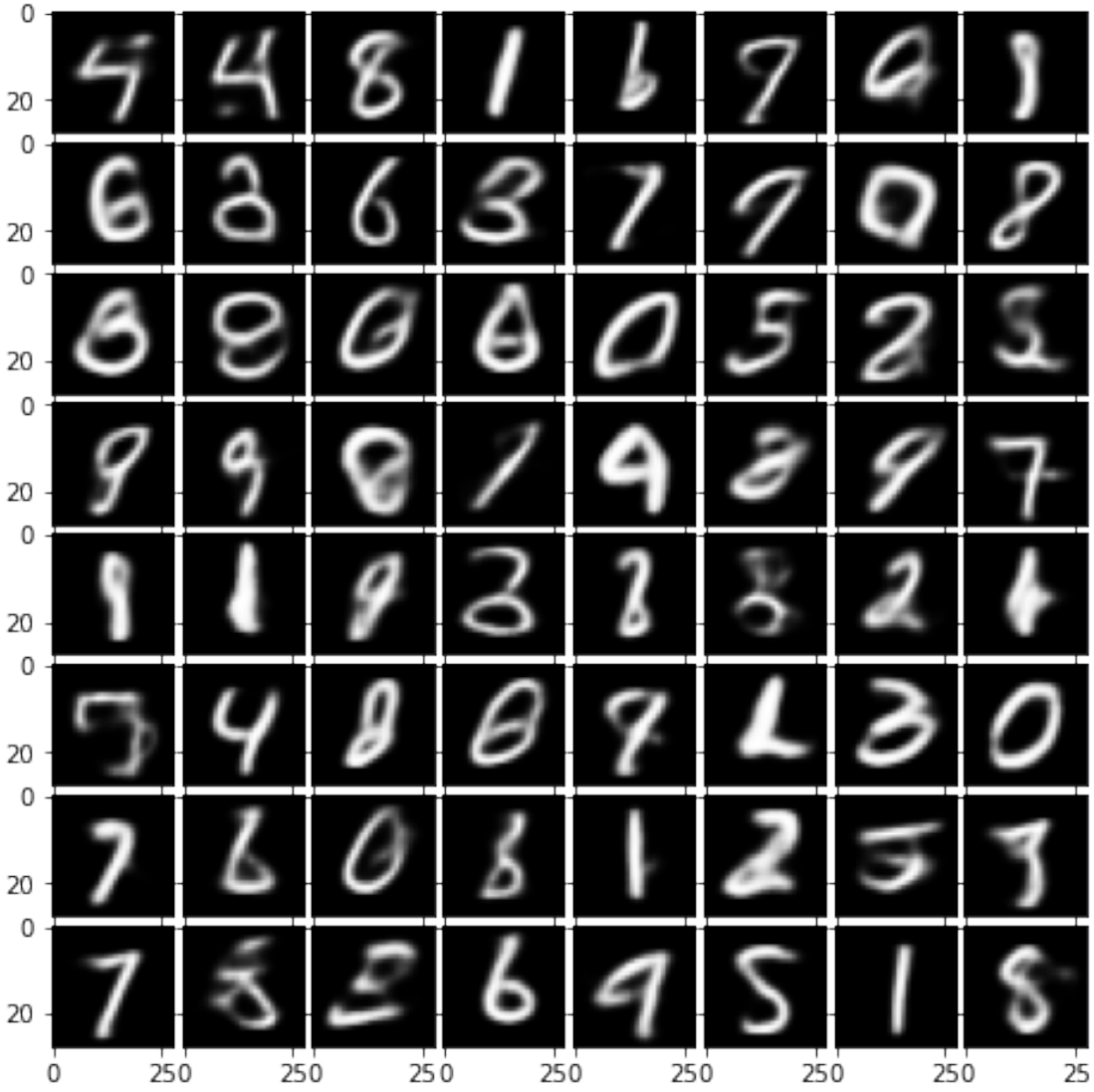

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

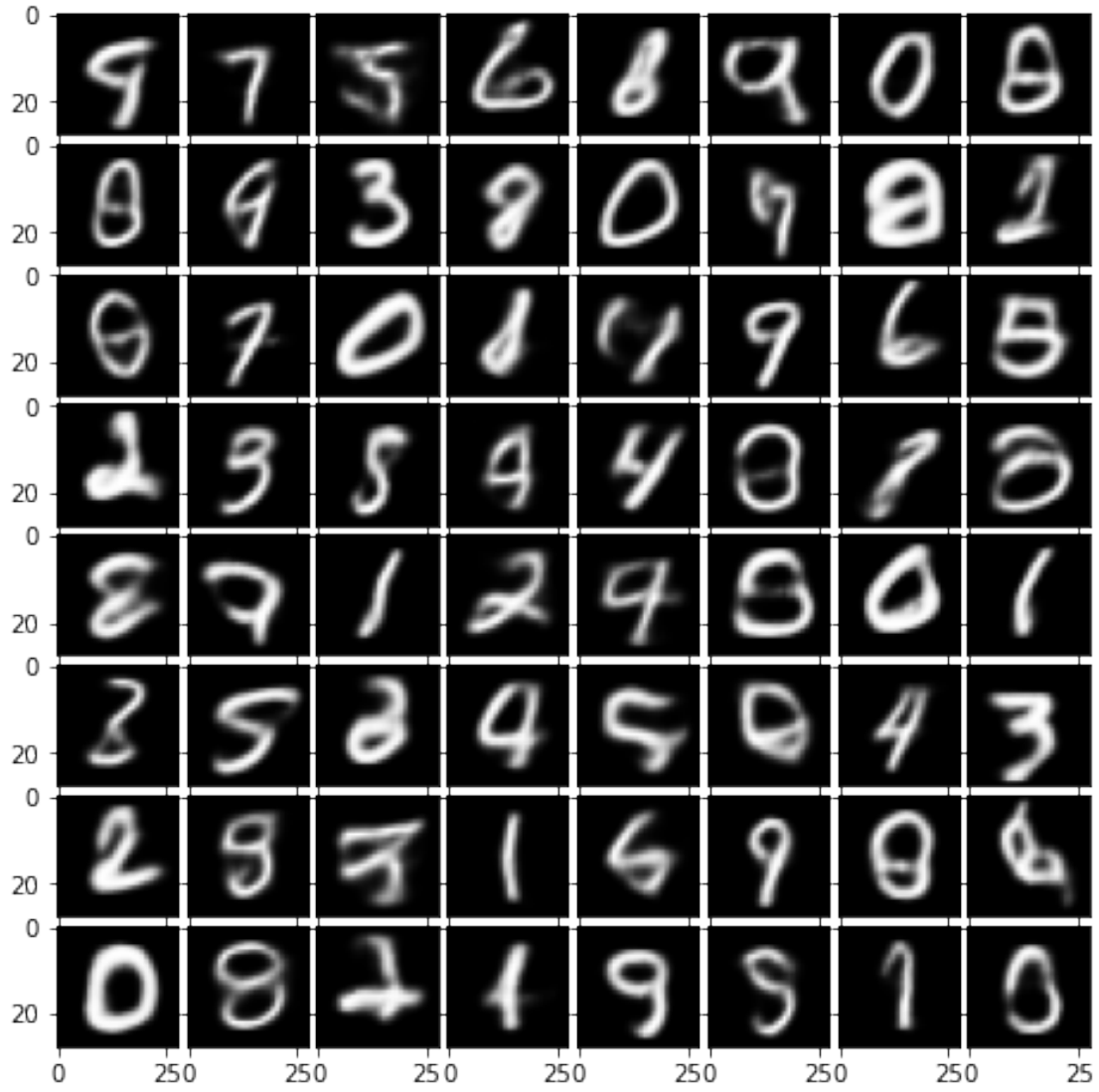

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

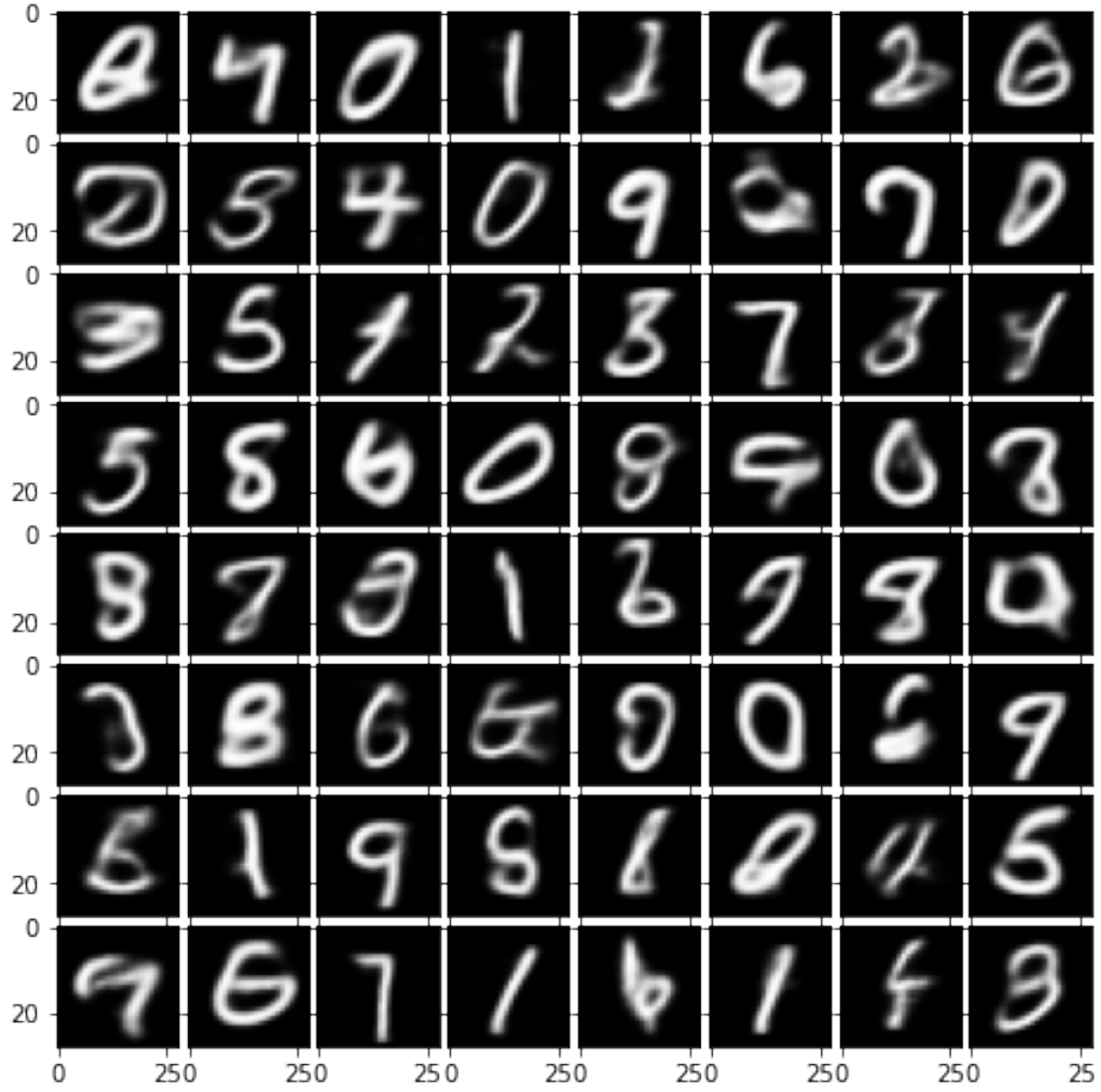

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

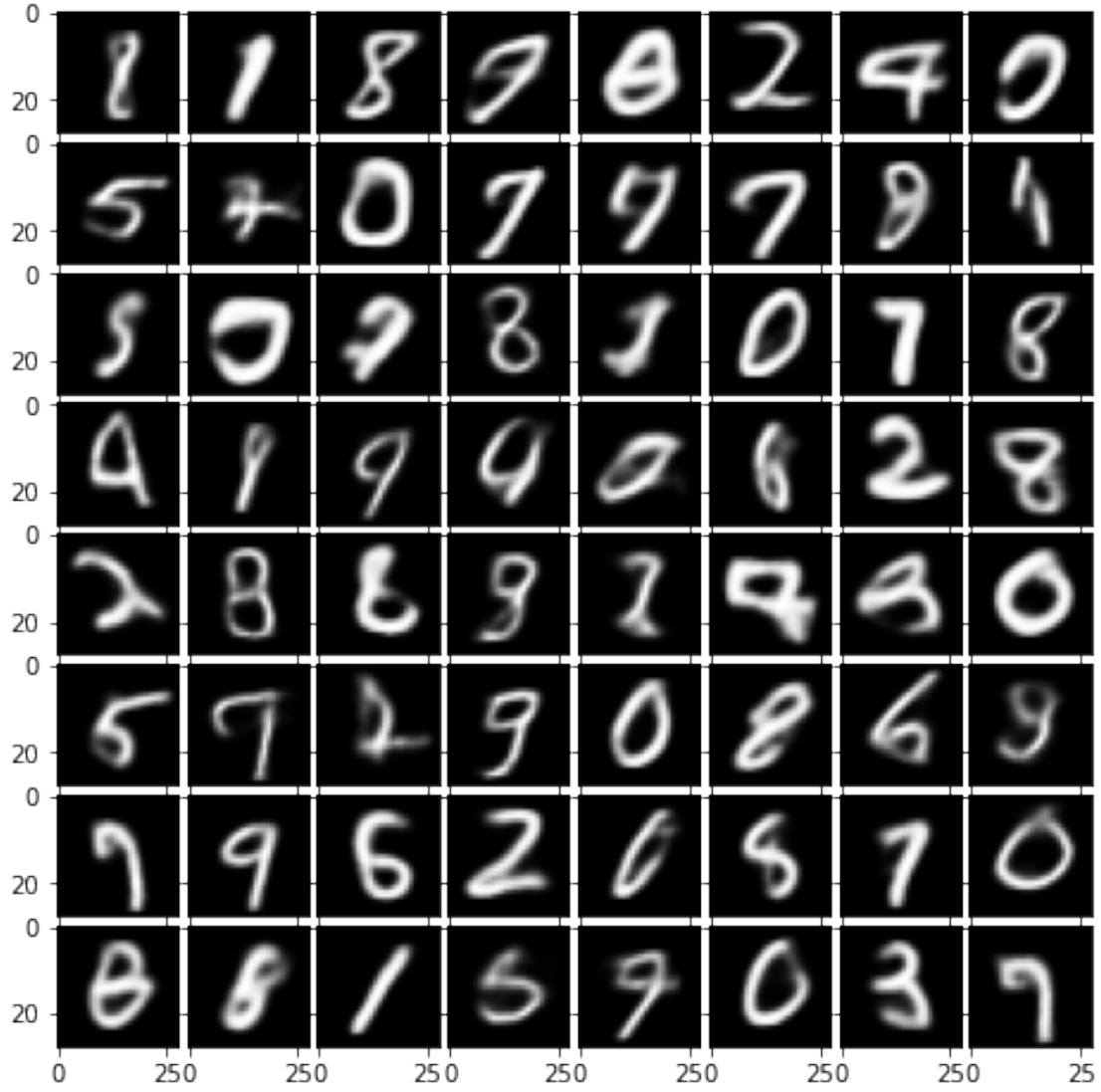

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

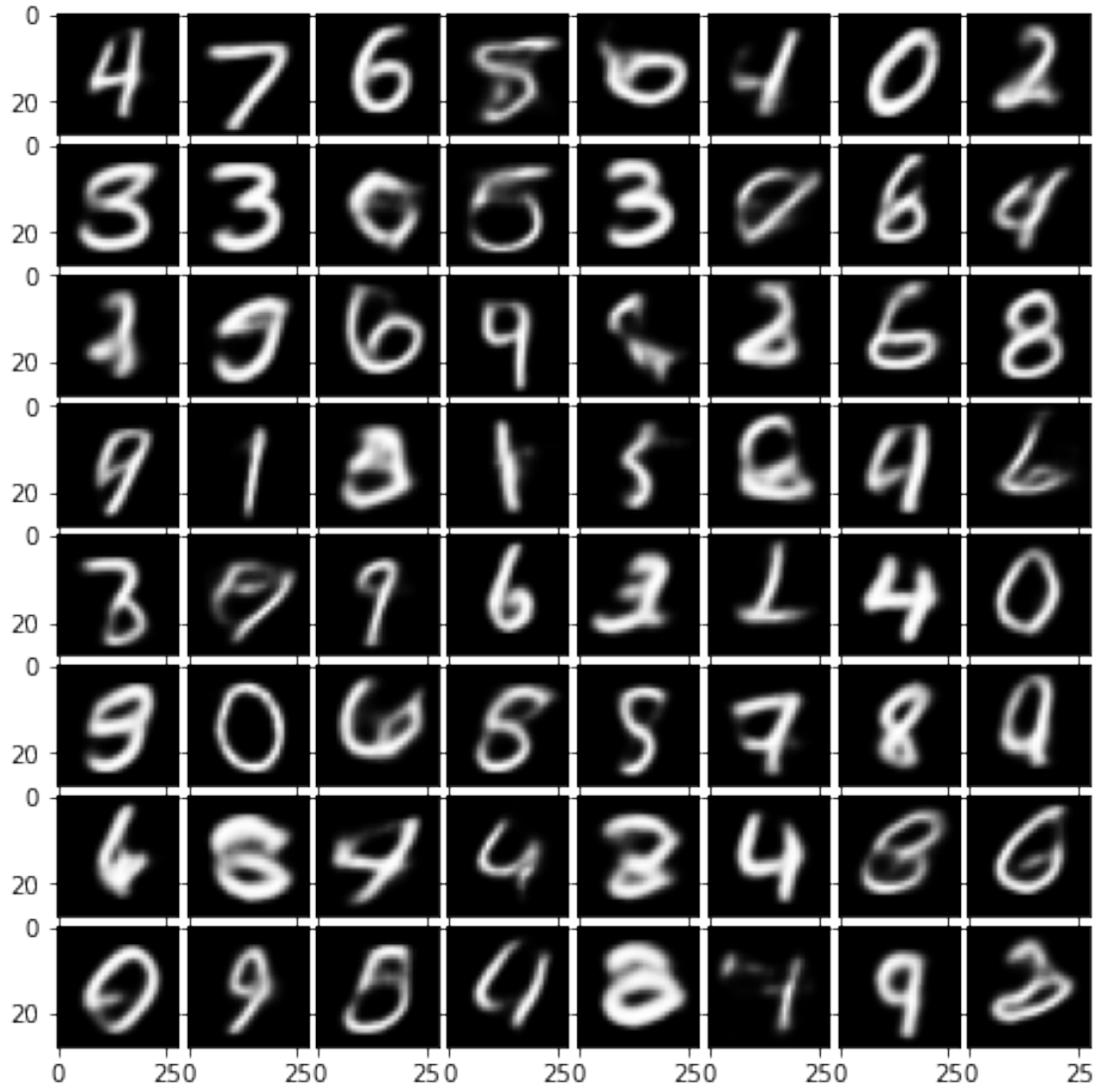

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

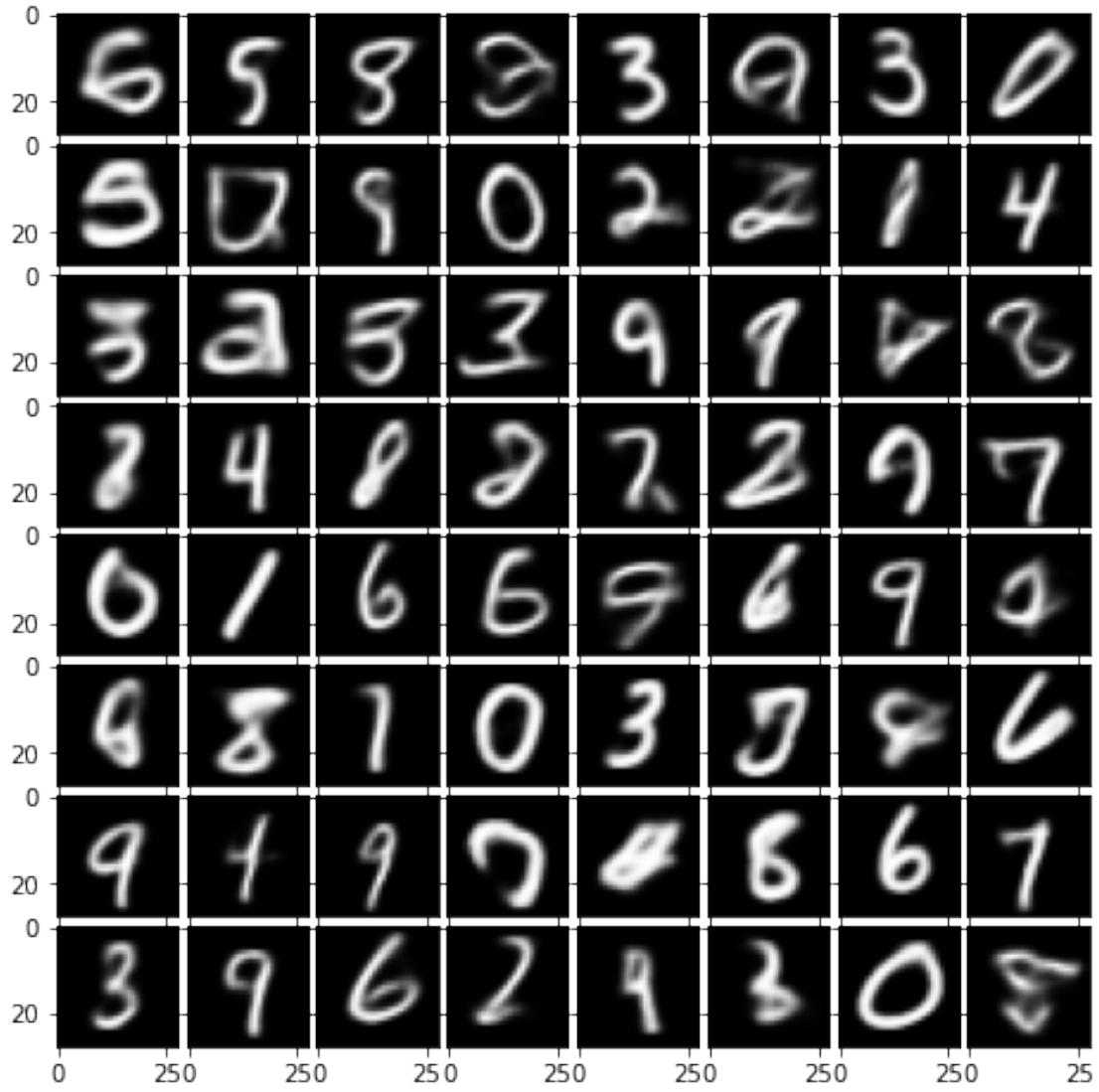

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

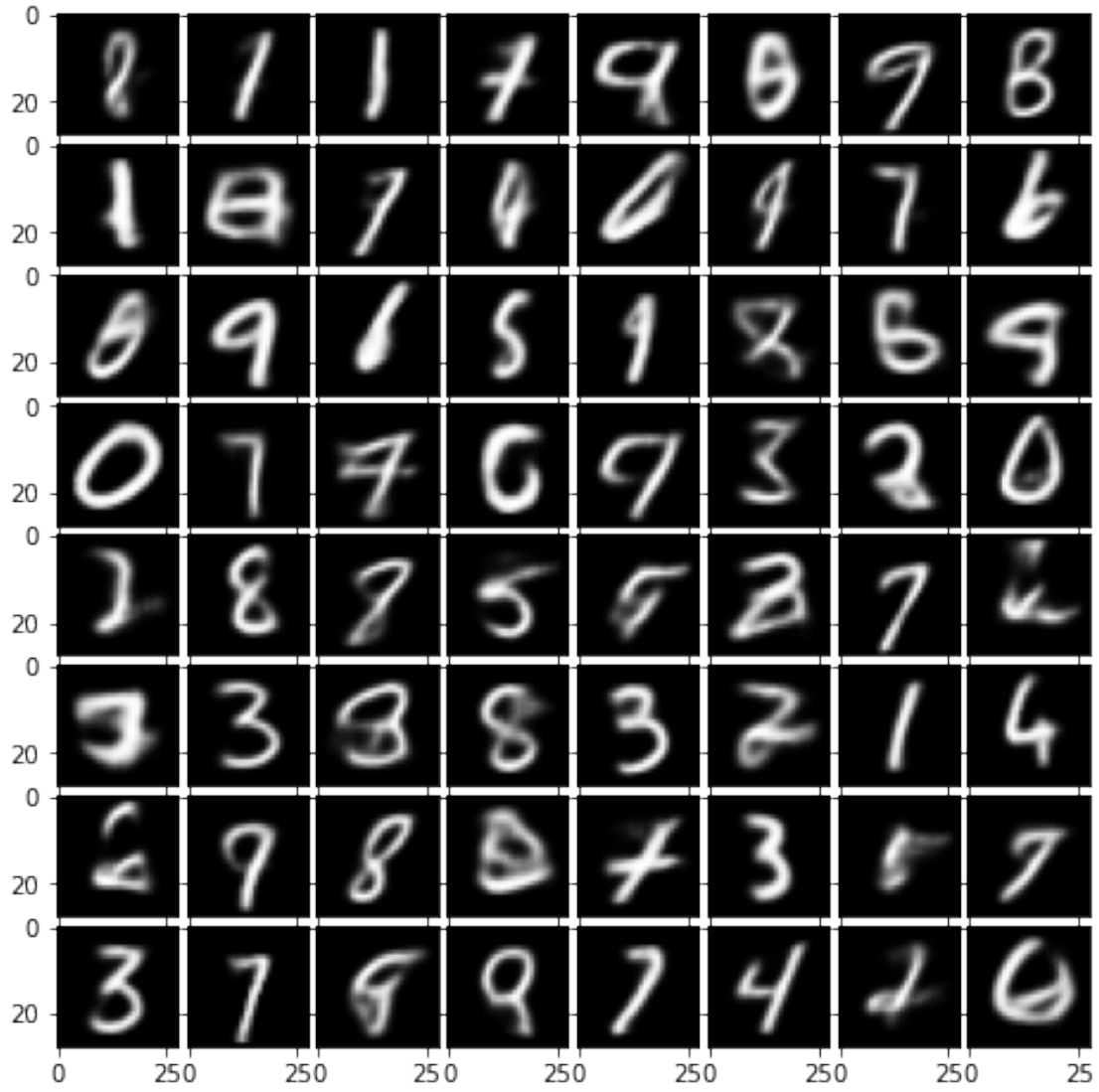

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

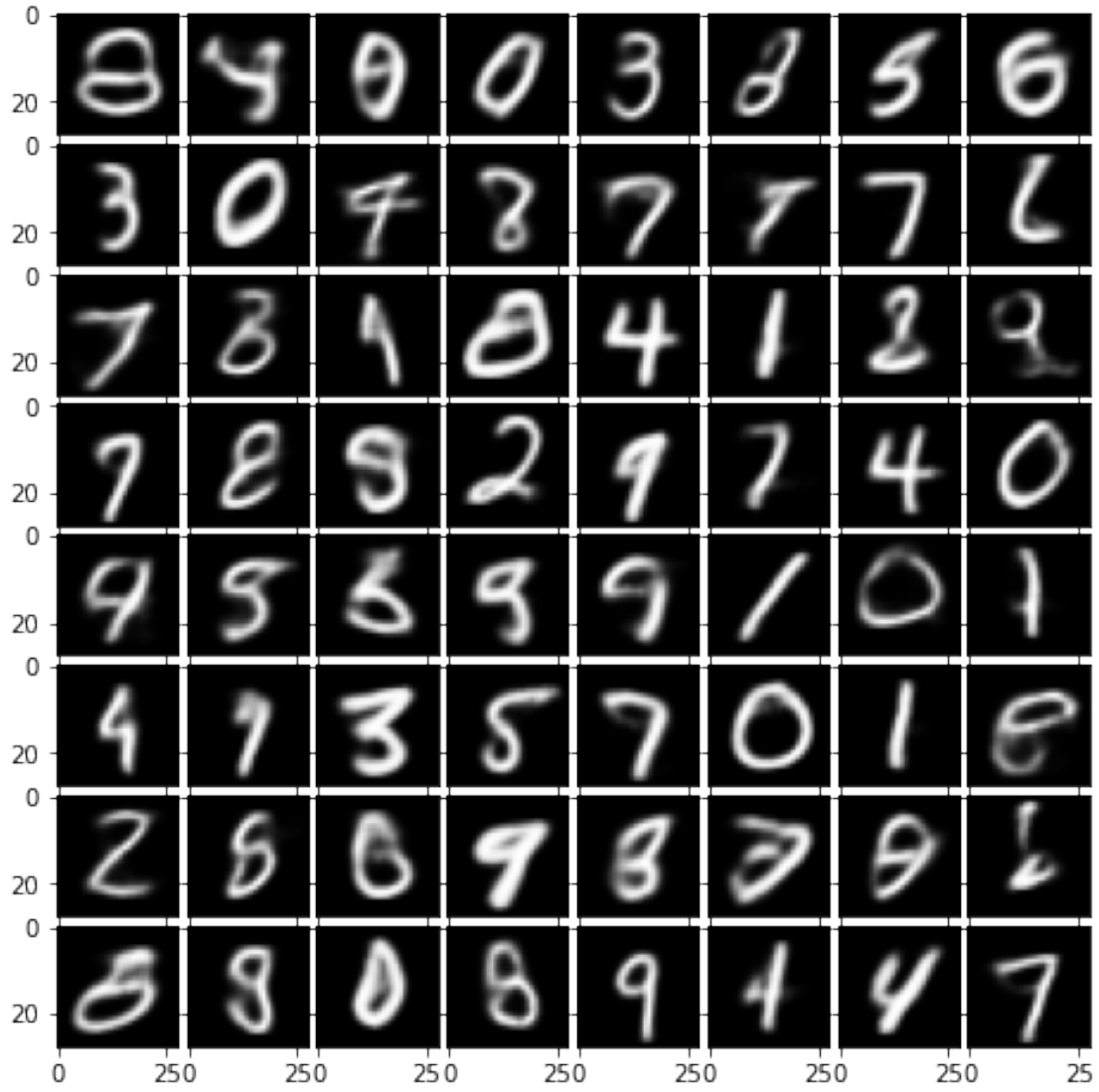

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

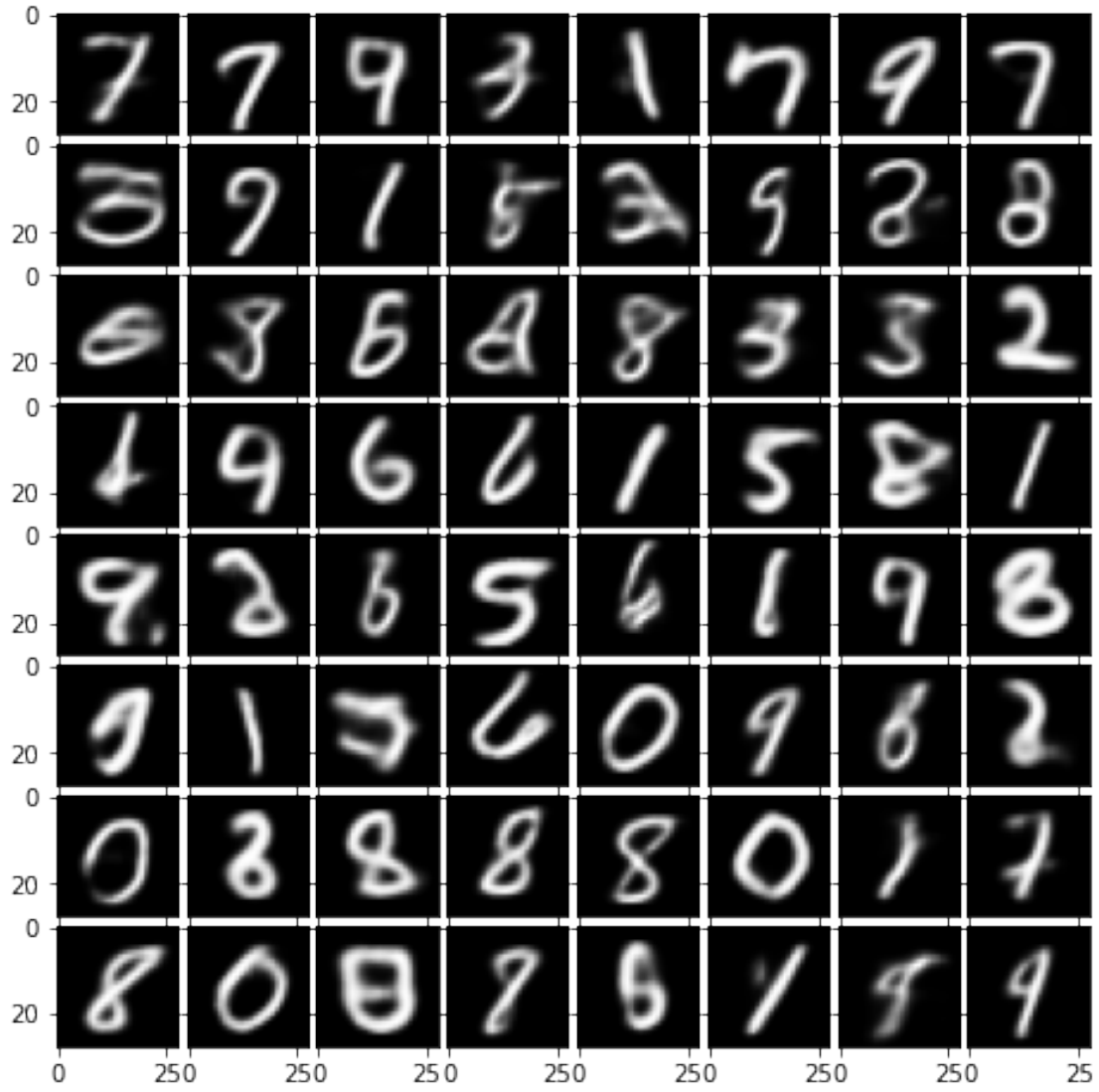

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

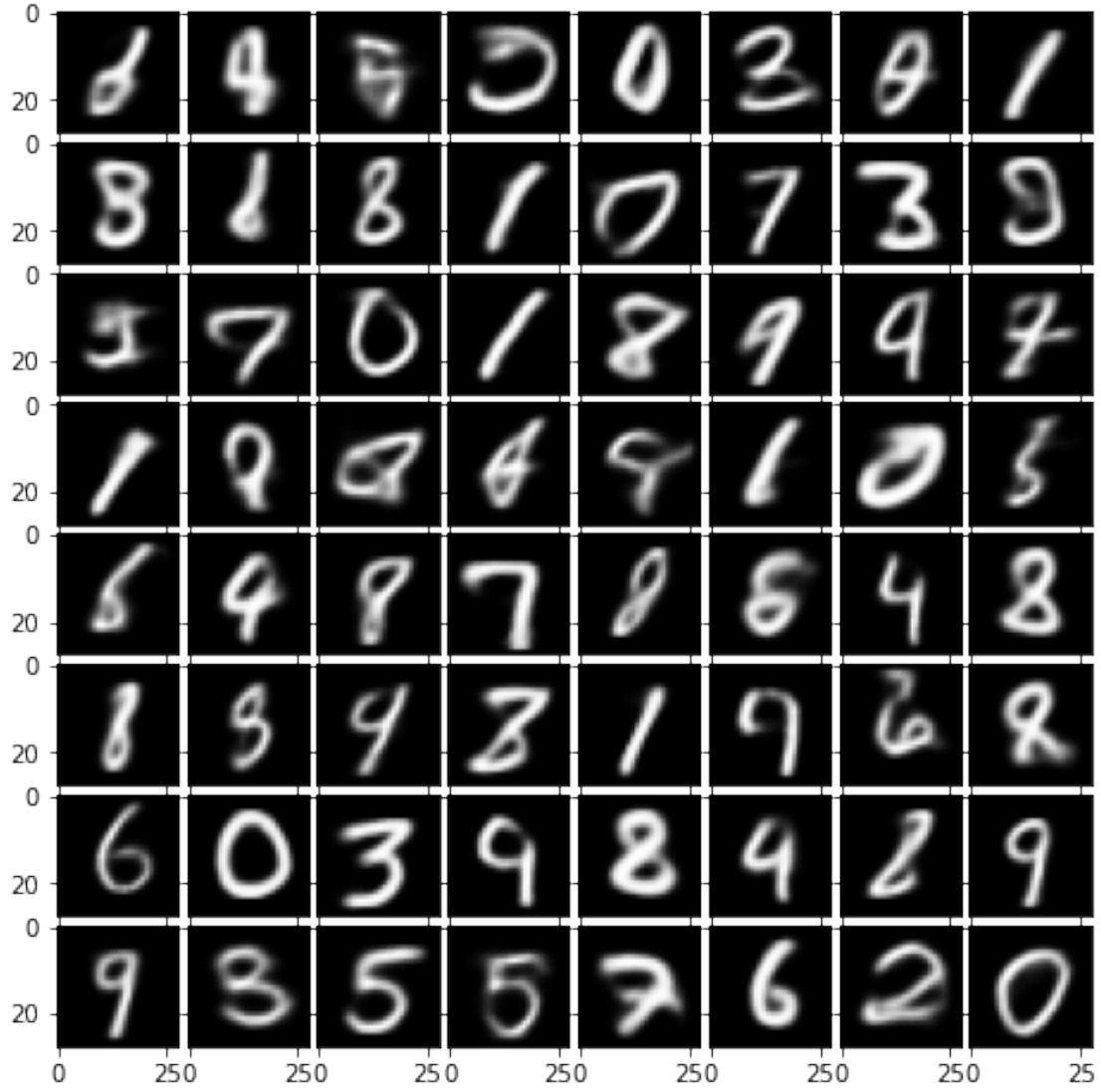

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

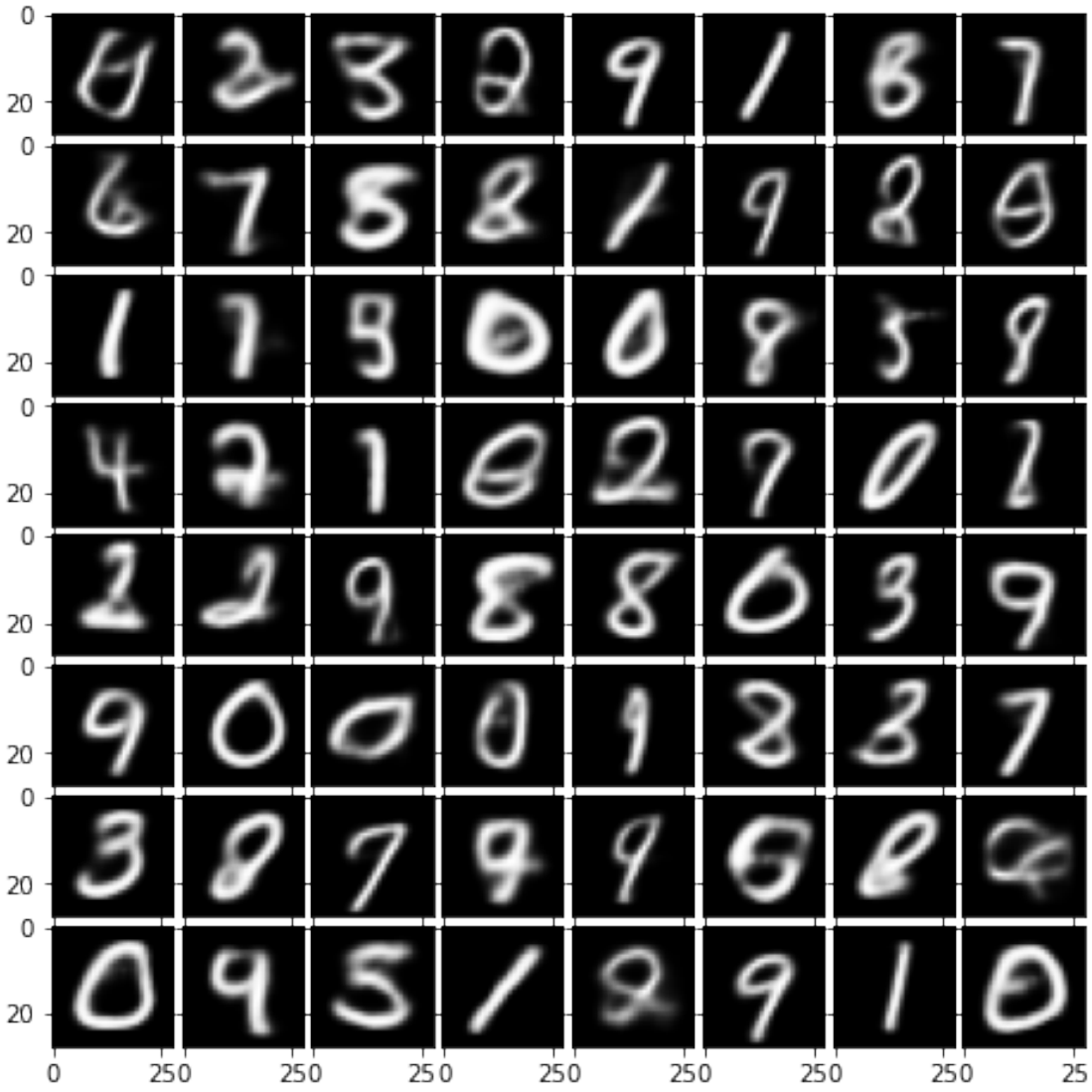

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25


0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25



[ ]:
```
imgs = model.sample(64).cpu().detach().reshape(64,28,28)
```

[ ]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
```

[22]:
```
plt.gray()
fig = plt.figure(figsize=(8., 8.))
grid = ImageGrid(fig, 111,  # similar to subplot(111)
                 nrows_ncols=(8, 8),  # creates 2x2 grid of axes
                 axes_pad=0.05  # pad between axes in inch.
                 )

for ax, im in zip(grid, imgs):
```

```
    # Iterating over the grid returns the Axes.
    ax.imshow(im)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```