nvae-on-mnist

September 21, 2021

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

```
Mon Sep 20 17:15:22 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   64C    P8    33W / 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*0.001*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: 552.5988113978068 recons_loss: 0.08149009006818135 kl1_loss:
8993000.035677018
epoch: 1 kl0_loss: 228.62604755045572 recons_loss: 0.18202623312473298 kl1_loss:
40437.194376920575
epoch: 2 kl0_loss: 48.95514374898275 recons_loss: 0.20301823207537334 kl1_loss:
399.9756480061849
epoch: 3 kl0_loss: 19.733062393697104 recons_loss: 0.20592214629650116 kl1_loss:
148.10959665120444
epoch: 4 kl0_loss: 10.35953073272705 recons_loss: 0.19865384487311044 kl1_loss:
55.09836978149414
epoch: 5 kl0_loss: 7.125041558837891 recons_loss: 0.19905106410185497 kl1_loss:
19.358863272094727
epoch: 6 kl0_loss: 5.818726324971517 recons_loss: 0.2011004822254181 kl1_loss:
7.192104116439819
epoch: 7 kl0_loss: 5.02292872873942 recons_loss: 0.20027347994645436 kl1_loss:
3.40001966603597
epoch: 8 kl0_loss: 4.589922513326009 recons_loss: 0.19524338253339132 kl1_loss:
1.7755799393335978
epoch: 9 kl0_loss: 4.379409768931071 recons_loss: 0.19027946609656016 kl1_loss:
0.9343214878241222
epoch: 10 kl0_loss: 4.118248423258463 recons_loss: 0.1897810166120529 kl1_loss:
0.49708919458389283
```

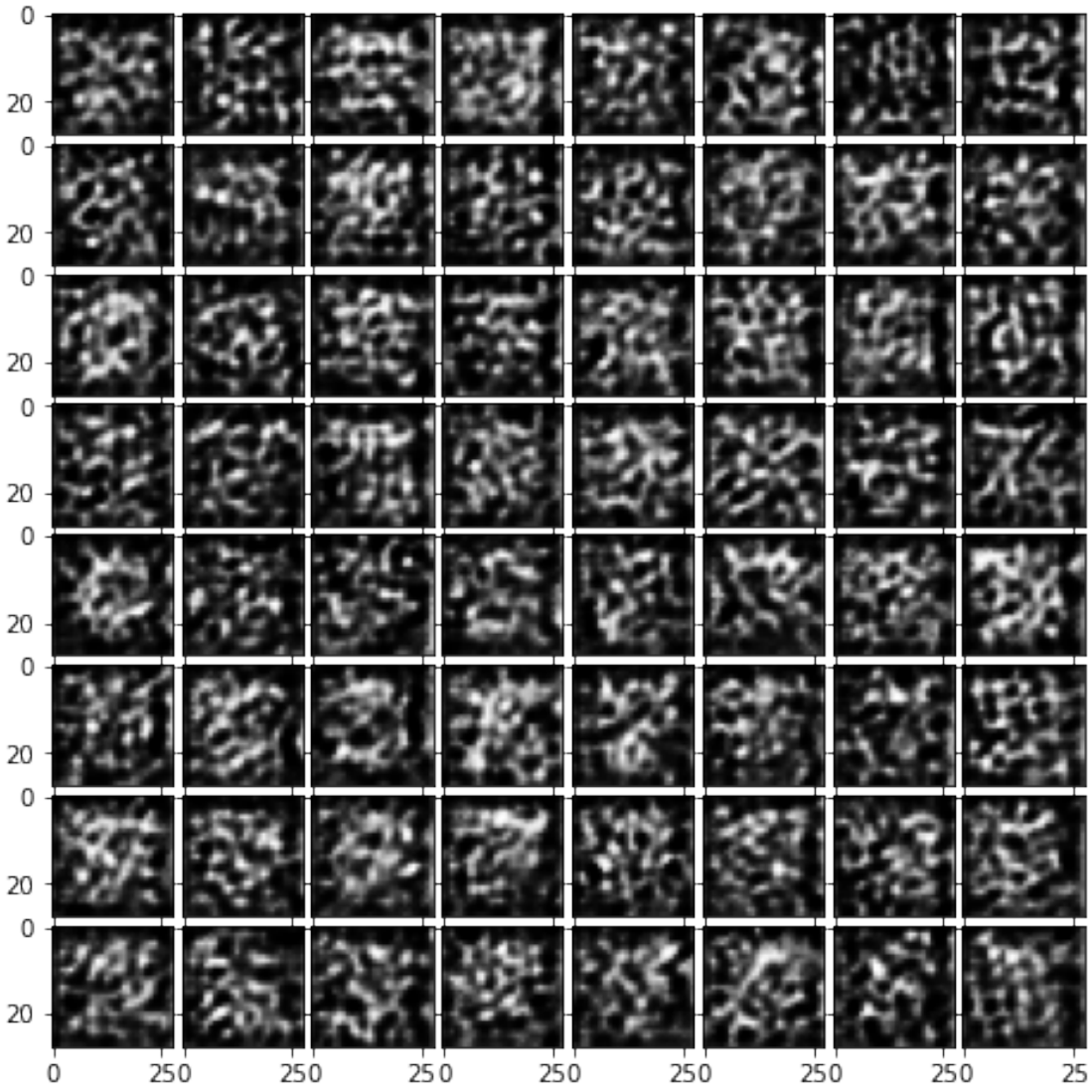
```
epoch: 11 kl0_loss: 3.8215887528737387 recons_loss: 0.19110879057248434
kl1_loss: 0.32747052941322324
epoch: 12 kl0_loss: 3.5498366305033366 recons_loss: 0.19304395826657614
kl1_loss: 0.22306393117904663
epoch: 13 kl0_loss: 3.296511098607381 recons_loss: 0.1953794021209081 kl1_loss:
0.17252056606610616
epoch: 14 kl0_loss: 3.053325121561686 recons_loss: 0.19807046367327372 kl1_loss:
0.12402575489679972
epoch: 15 kl0_loss: 2.801732429631551 recons_loss: 0.20151449382305145 kl1_loss:
0.09840667513211568
epoch: 16 kl0_loss: 2.5869005533854166 recons_loss: 0.20452761386235555
kl1_loss: 0.07483918245633443
epoch: 17 kl0_loss: 2.3606560024261474 recons_loss: 0.20798216117223103
kl1_loss: 0.05998423682848612
epoch: 18 kl0_loss: 2.148428874460856 recons_loss: 0.21159059694608054 kl1_loss:
0.05094919376373291
epoch: 19 kl0_loss: 1.9318052917480468 recons_loss: 0.2154268349011739 kl1_loss:
0.042695477231343586
```

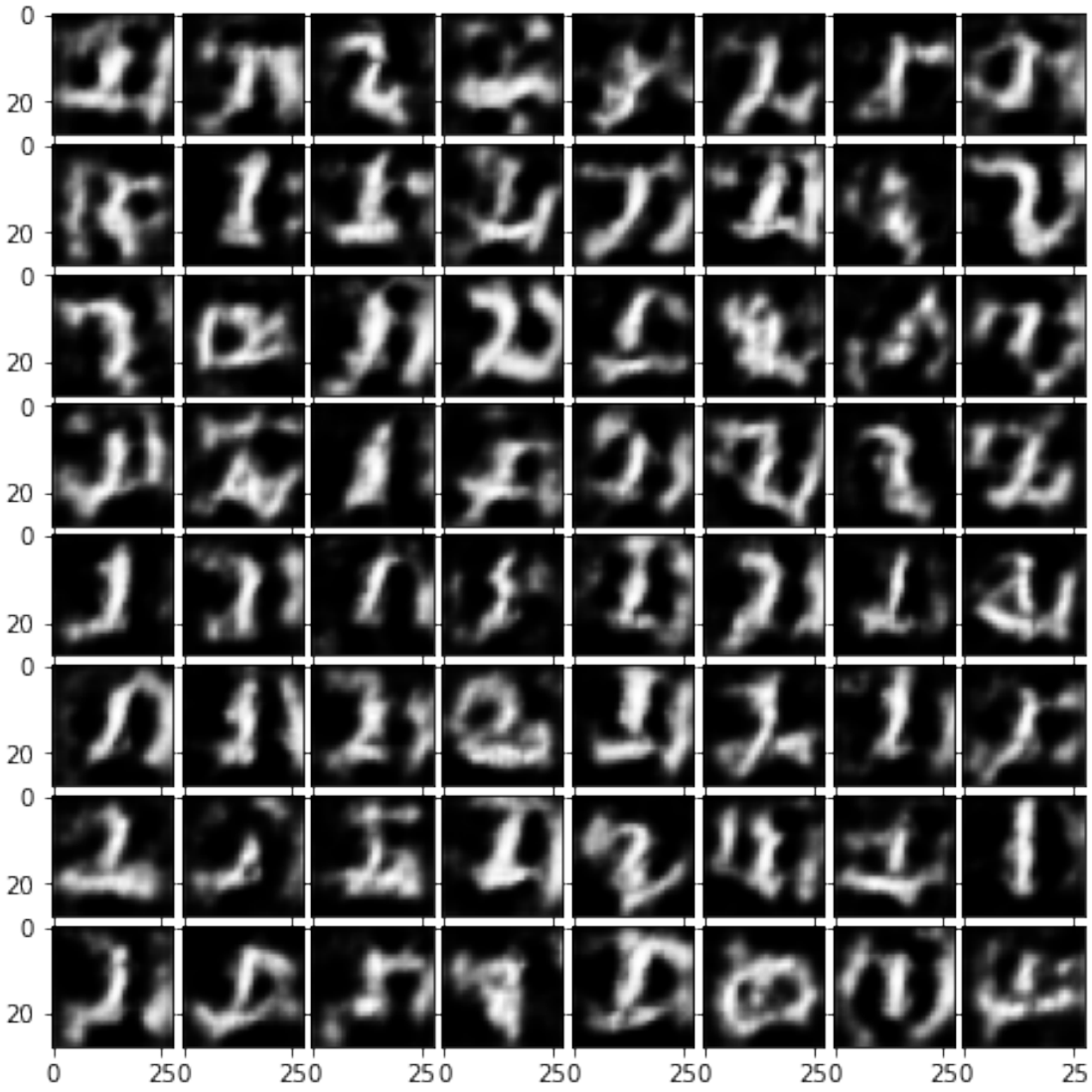
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22: RuntimeWarning:
More than 20 figures have been opened. Figures created through the pyplot
interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and
may consume too much memory. (To control this warning, see the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

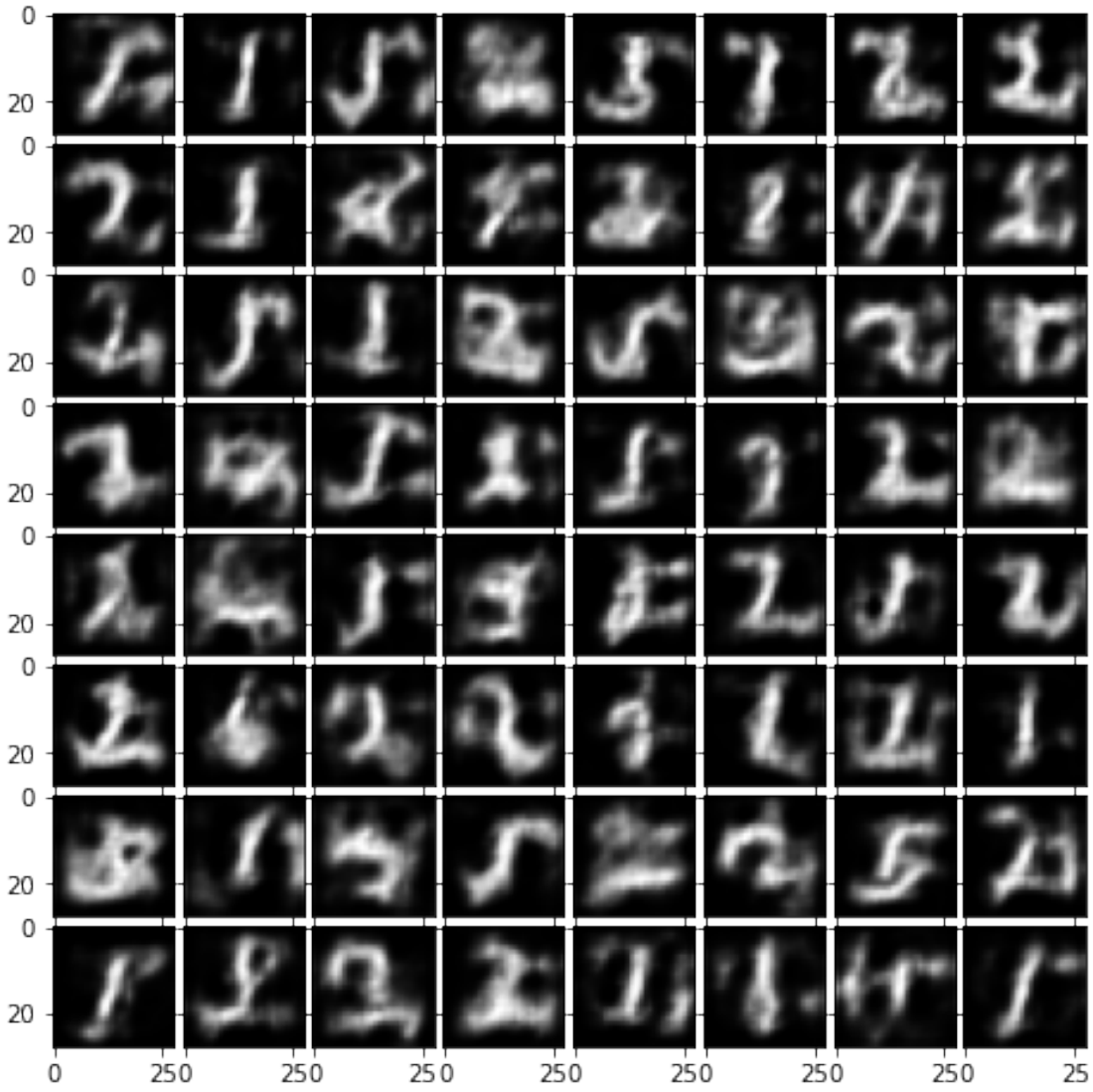
```
epoch: 20 kl0_loss: 1.7362581377029418 recons_loss: 0.21901256087621054
kl1_loss: 0.035107791646321614
epoch: 21 kl0_loss: 1.5411536390940348 recons_loss: 0.22344956988493603
kl1_loss: 0.030008816734949748
epoch: 22 kl0_loss: 1.344944715499878 recons_loss: 0.22727305982112886 kl1_loss:
0.024766757329305014
epoch: 23 kl0_loss: 1.1489202901204427 recons_loss: 0.23169128551483154
kl1_loss: 0.02133170042037964
epoch: 24 kl0_loss: 0.9609287005106608 recons_loss: 0.235880291056633 kl1_loss:
0.017633835792541505
epoch: 25 kl0_loss: 0.7843037874539693 recons_loss: 0.24026916534900666
kl1_loss: 0.014048905849456787
epoch: 26 kl0_loss: 0.6375093693733216 recons_loss: 0.24379087624549867
kl1_loss: 0.011795634031295776
epoch: 27 kl0_loss: 0.5062767448743184 recons_loss: 0.24737420587539674
kl1_loss: 0.009501897716522217
epoch: 28 kl0_loss: 0.4132228946208954 recons_loss: 0.24971380554835002
kl1_loss: 0.007667074076334635
epoch: 29 kl0_loss: 0.3333115641117096 recons_loss: 0.2519855671564738 kl1_loss:
0.0061984070618947345
epoch: 30 kl0_loss: 0.27195795777638754 recons_loss: 0.2537960555553436
kl1_loss: 0.005078613376617432
epoch: 31 kl0_loss: 0.2308945845444997 recons_loss: 0.2550339027404785 kl1_loss:
0.004634869416554769
```

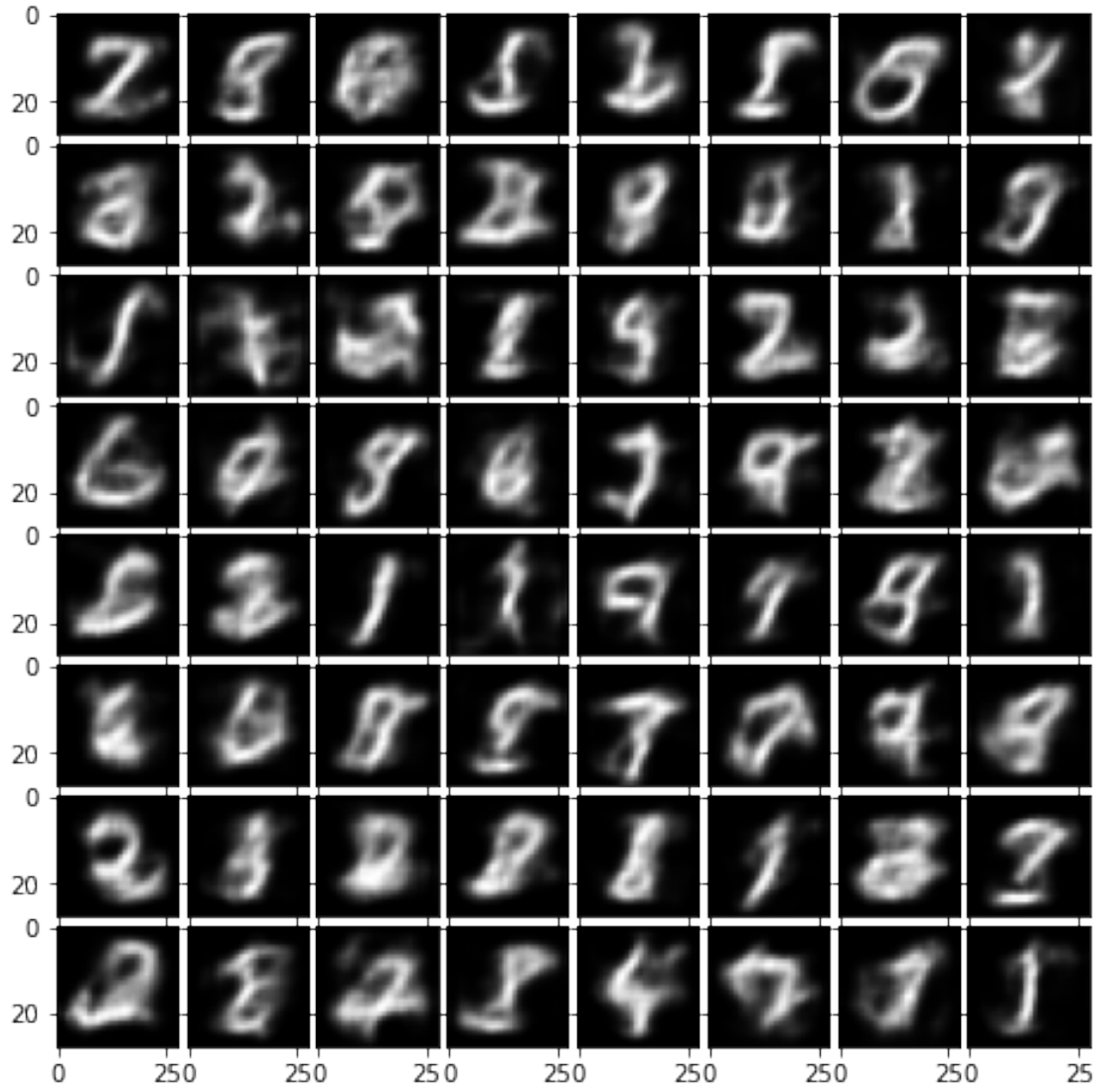
```
epoch: 32 kl0_loss: 0.18818732148011524 recons_loss: 0.25634173990885417
kl1_loss: 0.003628303337097168
epoch: 33 kl0_loss: 0.15807033089001973 recons_loss: 0.25725031161308287
kl1_loss: 0.003013256565729777
epoch: 34 kl0_loss: 0.1297524777452151 recons_loss: 0.25820223147074384
kl1_loss: 0.0023426968256632487
epoch: 35 kl0_loss: 0.11354501698414485 recons_loss: 0.2587387716770172
kl1_loss: 0.0021249086380004884
epoch: 36 kl0_loss: 0.08936313715775808 recons_loss: 0.2595750279744466
kl1_loss: 0.0013996047655741373
epoch: 37 kl0_loss: 0.0693962350110213 recons_loss: 0.2603385655085246 kl1_loss:
0.0013686153729756674
epoch: 38 kl0_loss: 0.05697437961101532 recons_loss: 0.2607404406229655
kl1_loss: 0.0007266483783721924
epoch: 39 kl0_loss: 0.025881875984867415 recons_loss: 0.2620153597354889
kl1_loss: 0.0007585242907206217
epoch: 40 kl0_loss: 0.012879262799024582 recons_loss: 0.2624769925117493
kl1_loss: 0.0007720725377400716
epoch: 41 kl0_loss: 0.010224225065608819 recons_loss: 0.262556133111318
kl1_loss: 0.0007175376194000244
epoch: 42 kl0_loss: 0.00369331260373195 recons_loss: 0.26281385396321616
kl1_loss: 0.0009236536343892415
epoch: 43 kl0_loss: 0.006084038613736629 recons_loss: 0.2626700224399567
kl1_loss: -2.4632374445597332e-05
epoch: 44 kl0_loss: 0.0022775954986612003 recons_loss: 0.26284793461958567
kl1_loss: 0.0011894471486409506
epoch: 45 kl0_loss: 0.0012298369571566582 recons_loss: 0.2628934335867564
kl1_loss: -0.00011633087793986003
epoch: 46 kl0_loss: 0.0007021887605388959 recons_loss: 0.2629090718428294
kl1_loss: -7.367533047993978e-05
epoch: 47 kl0_loss: 0.0005239377667506536 recons_loss: 0.26291763345400493
kl1_loss: 0.0007418174107869466
epoch: 48 kl0_loss: 0.0007759620154897371 recons_loss: 0.26287784128189084
kl1_loss: -8.412297566731771e-05
epoch: 49 kl0_loss: 0.0009865226725737253 recons_loss: 0.26288132621447247
kl1_loss: 0.001106262747446696
```

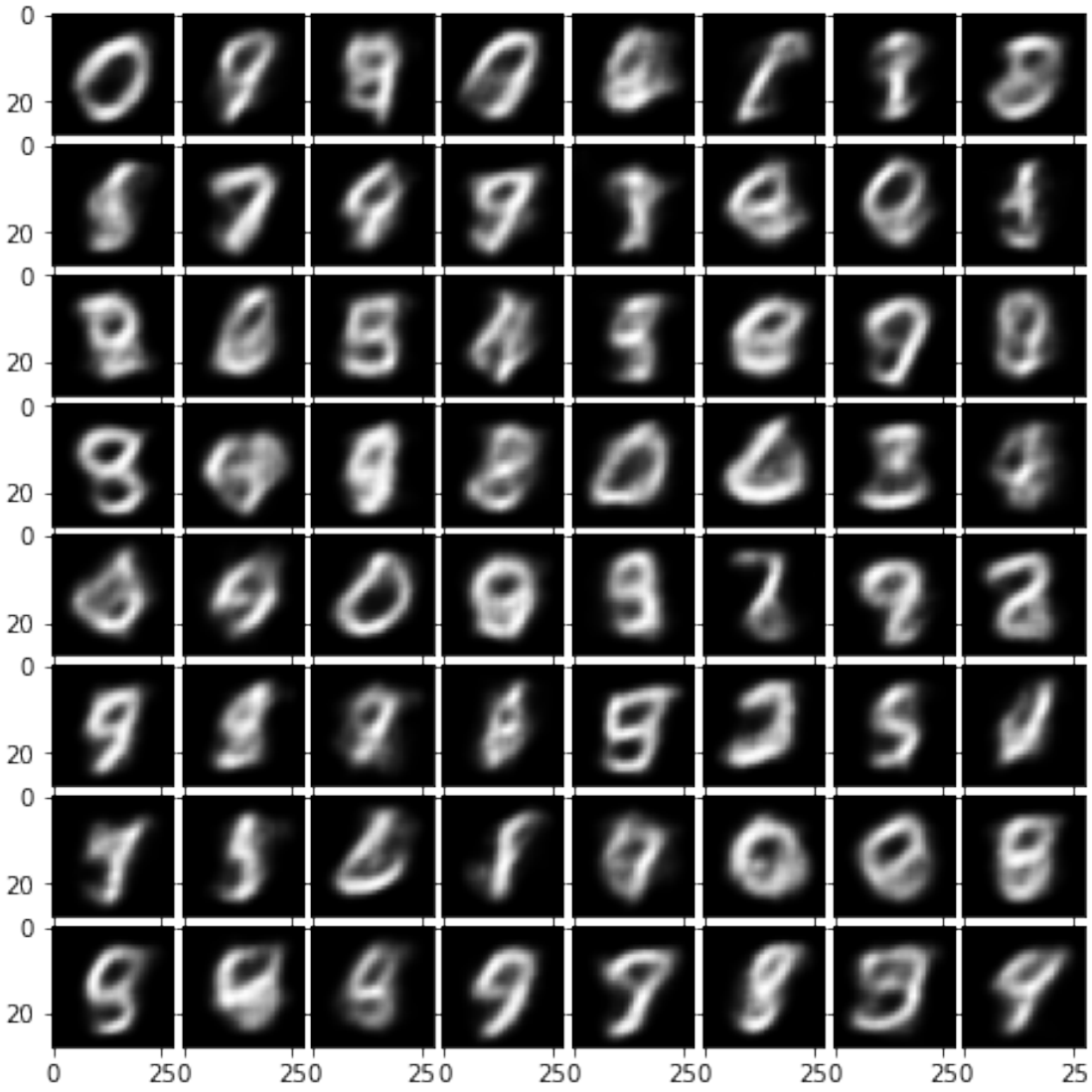
<Figure size 432x288 with 0 Axes>

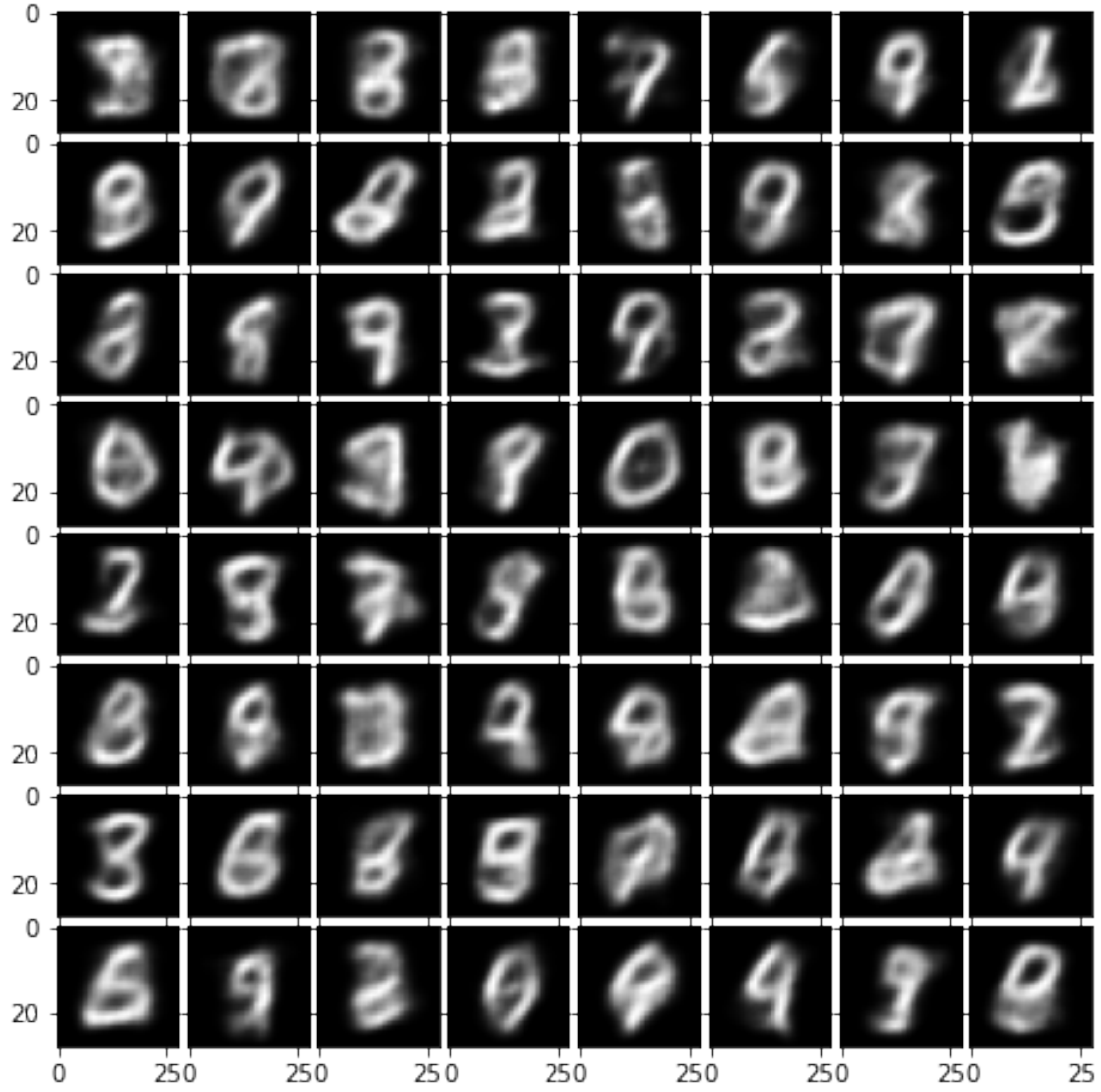

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

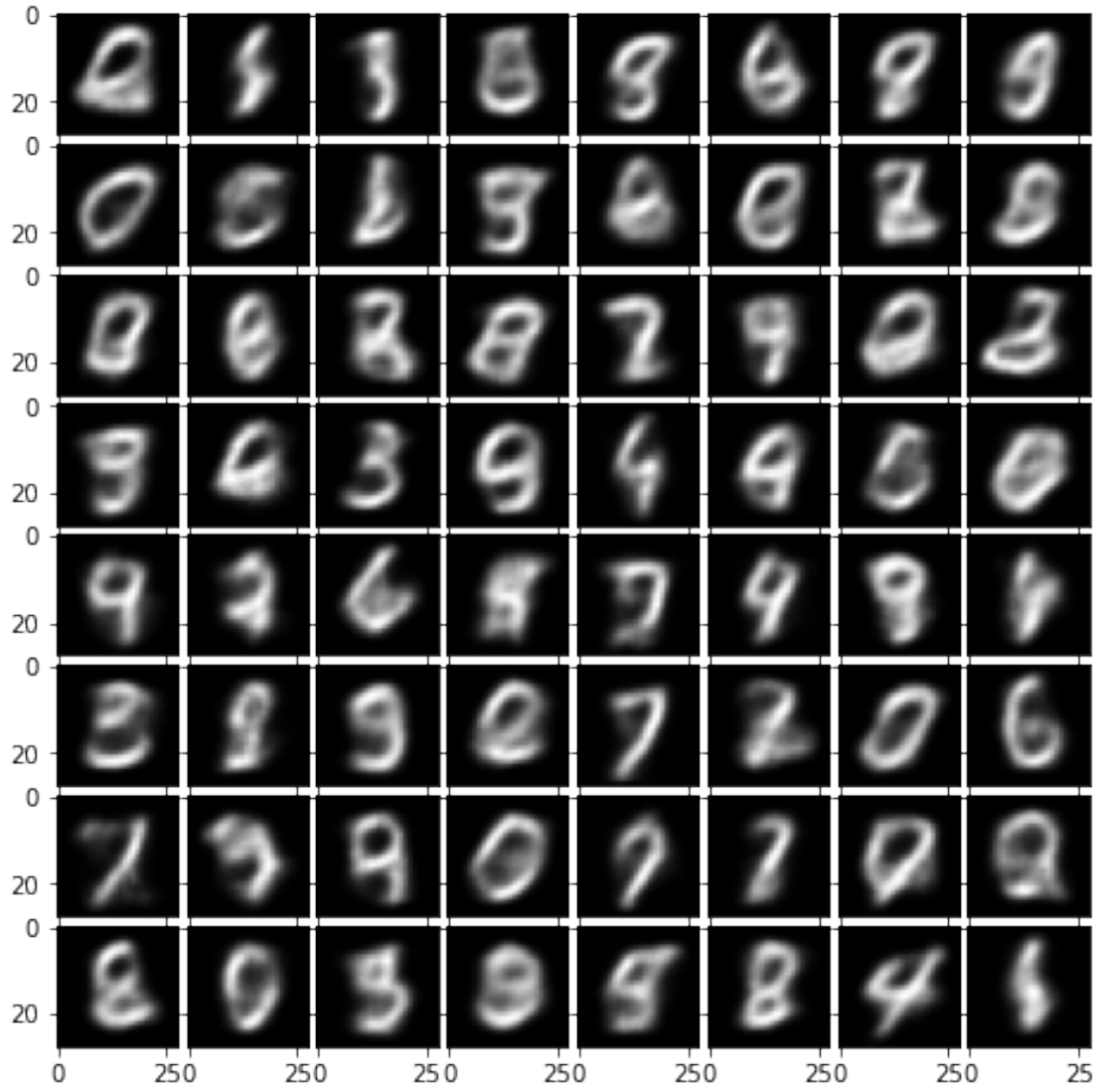

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

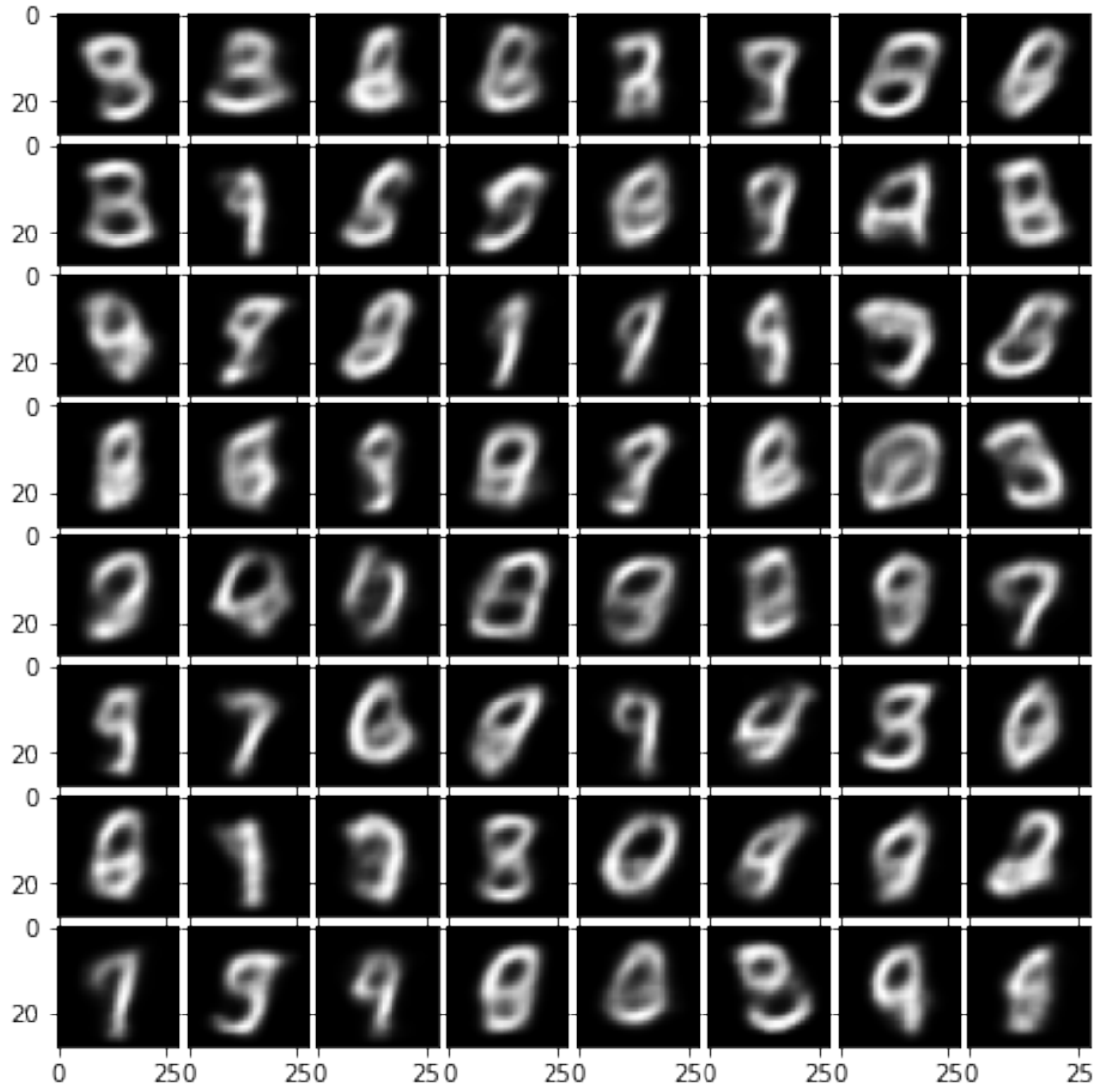

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

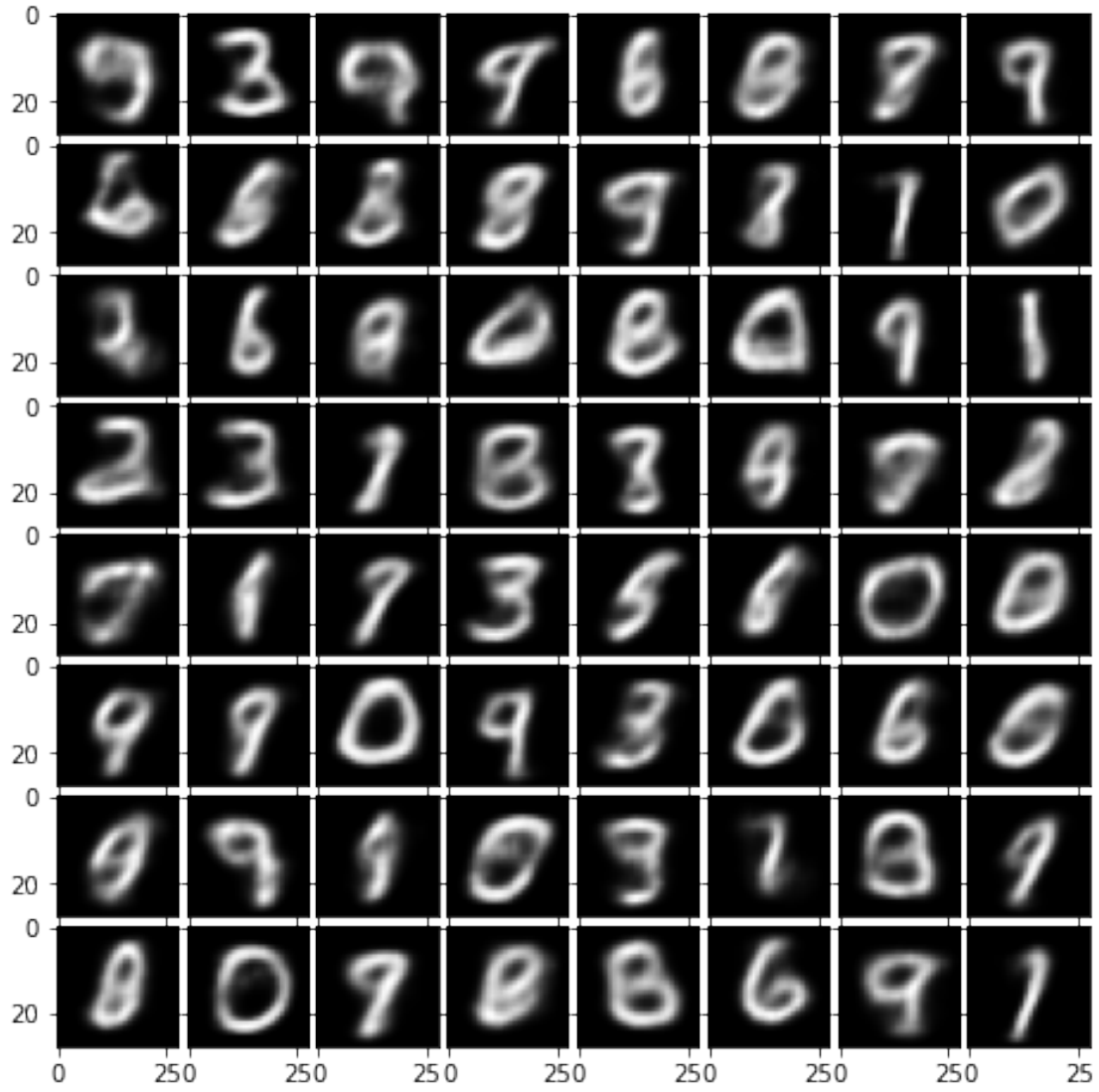

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

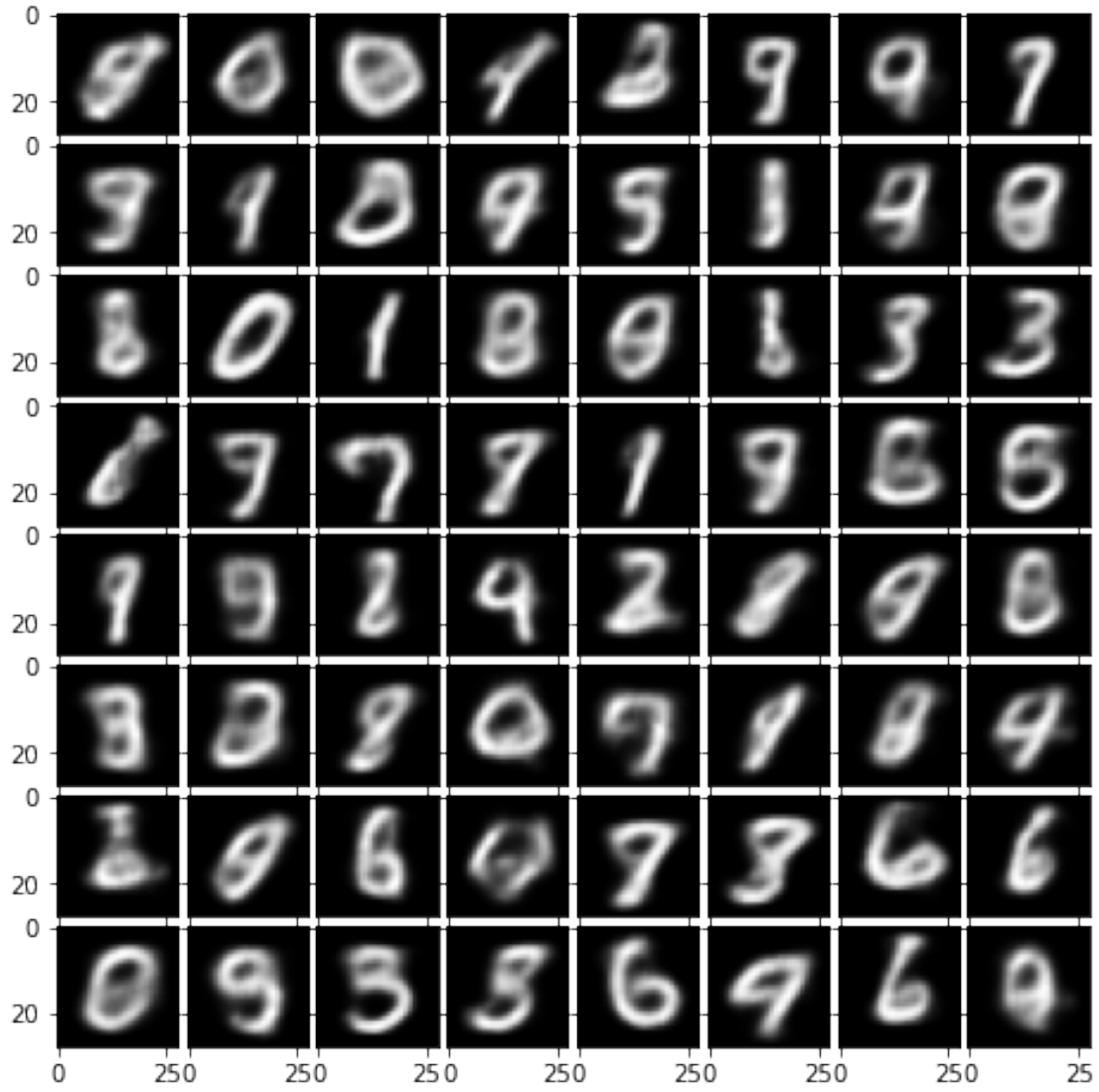

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

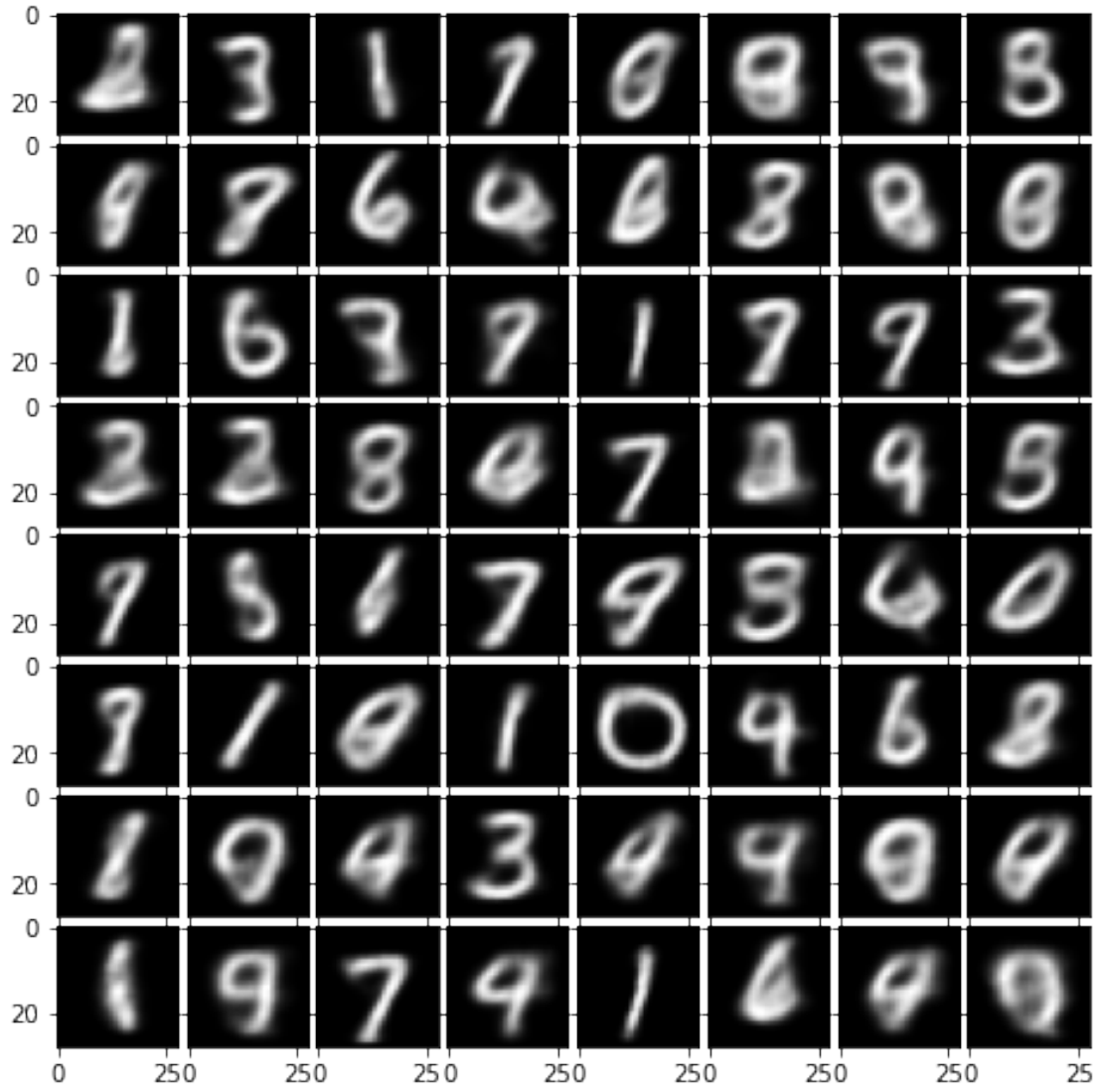

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

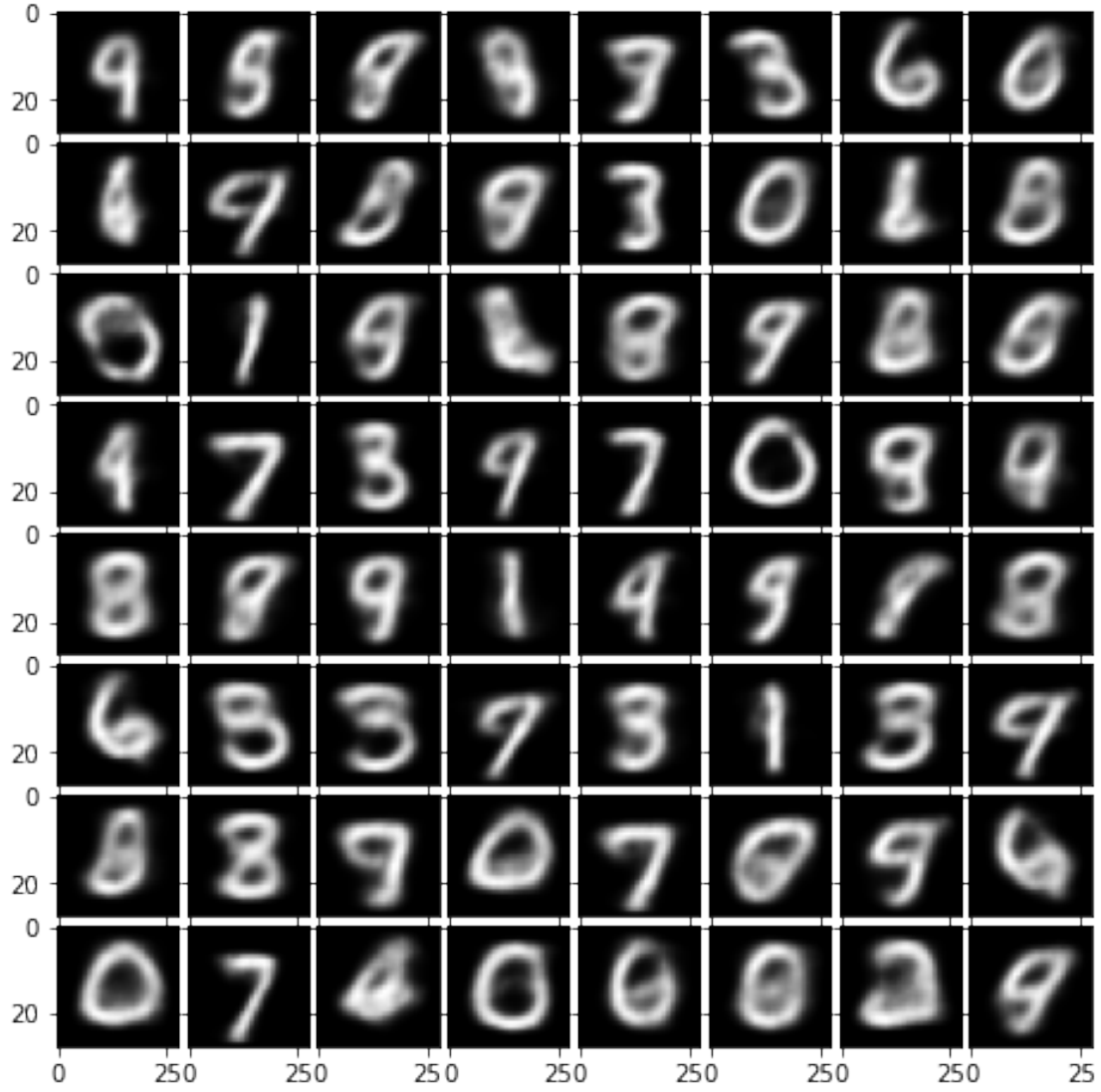

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

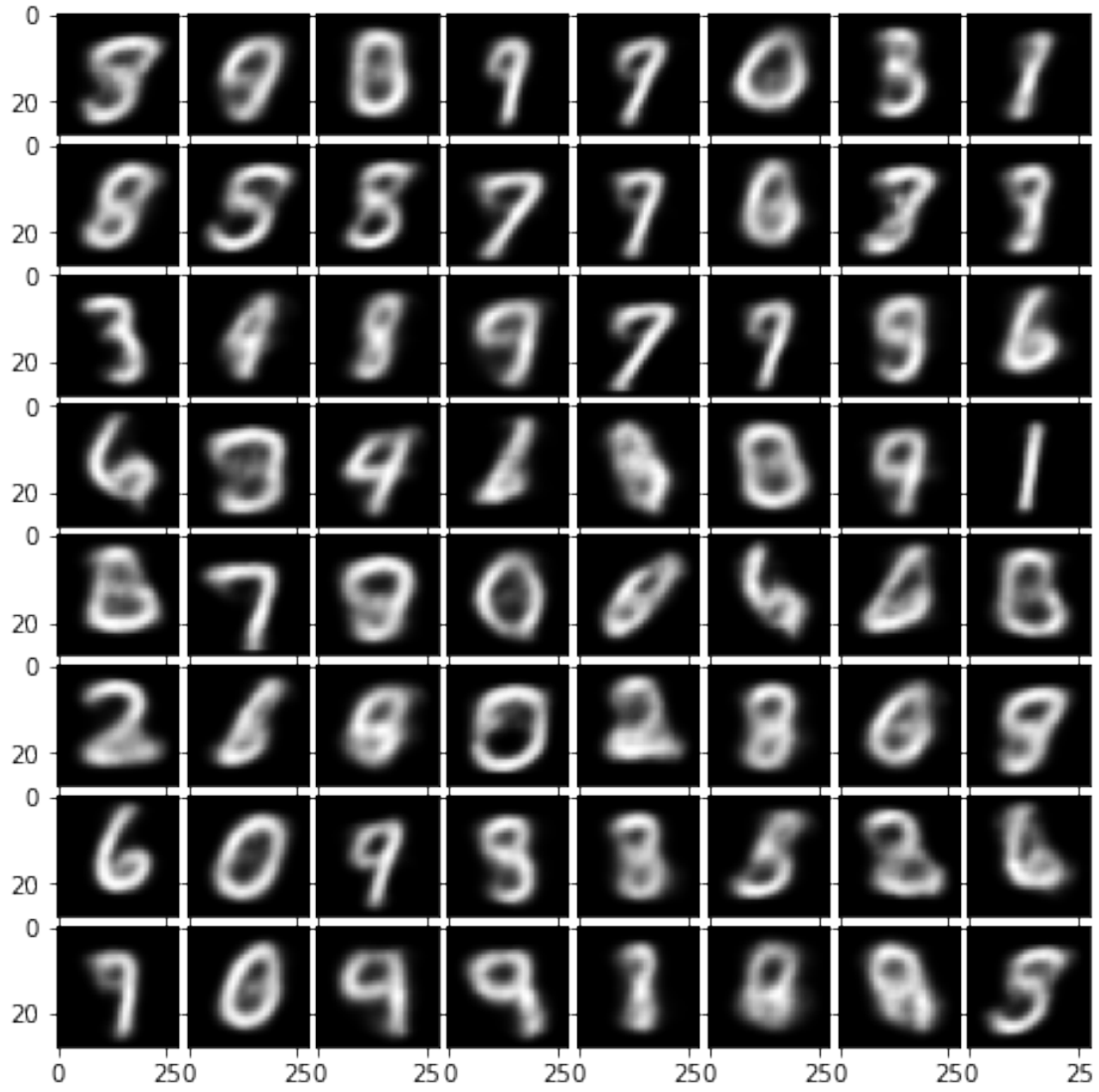

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

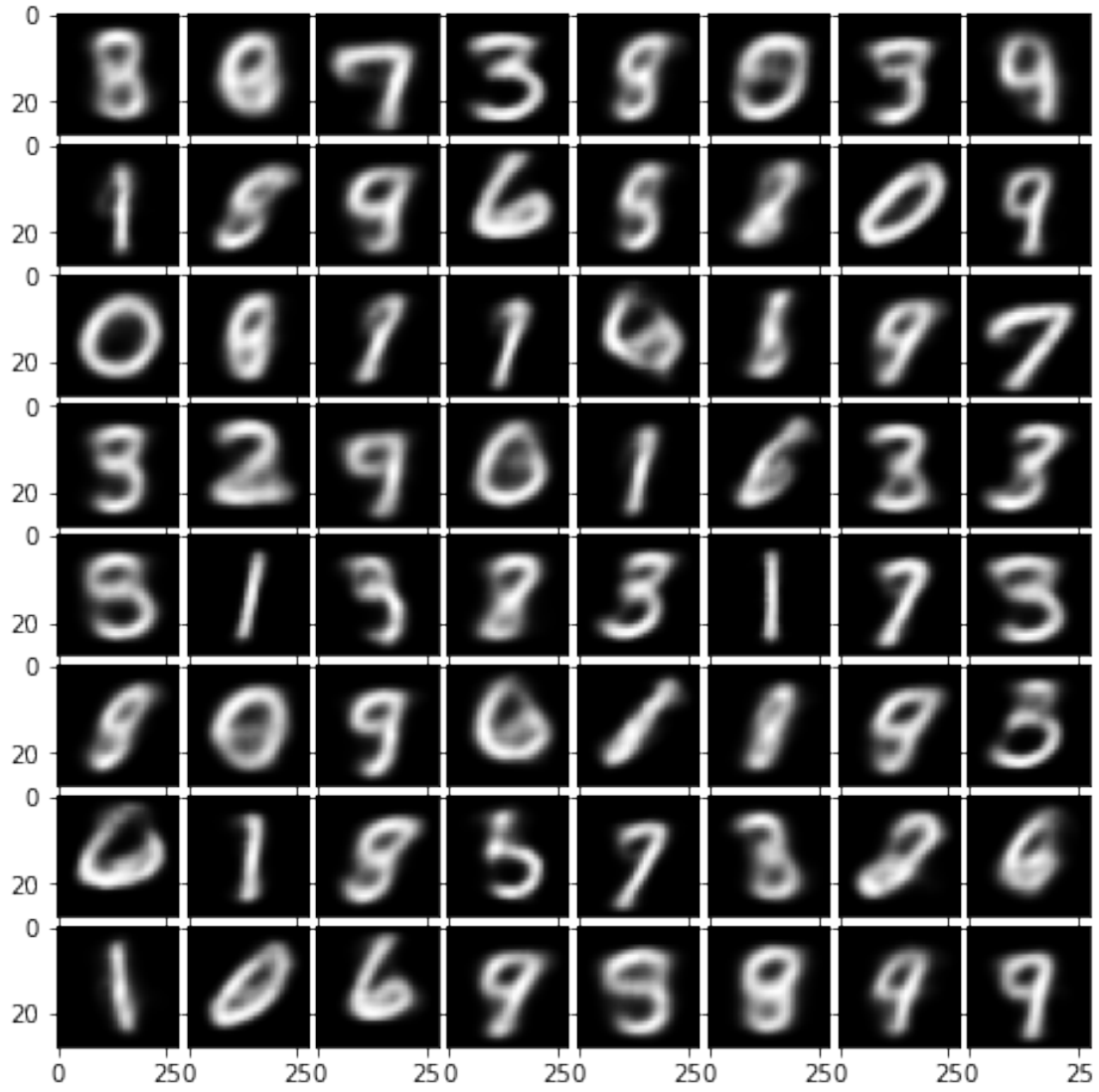

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

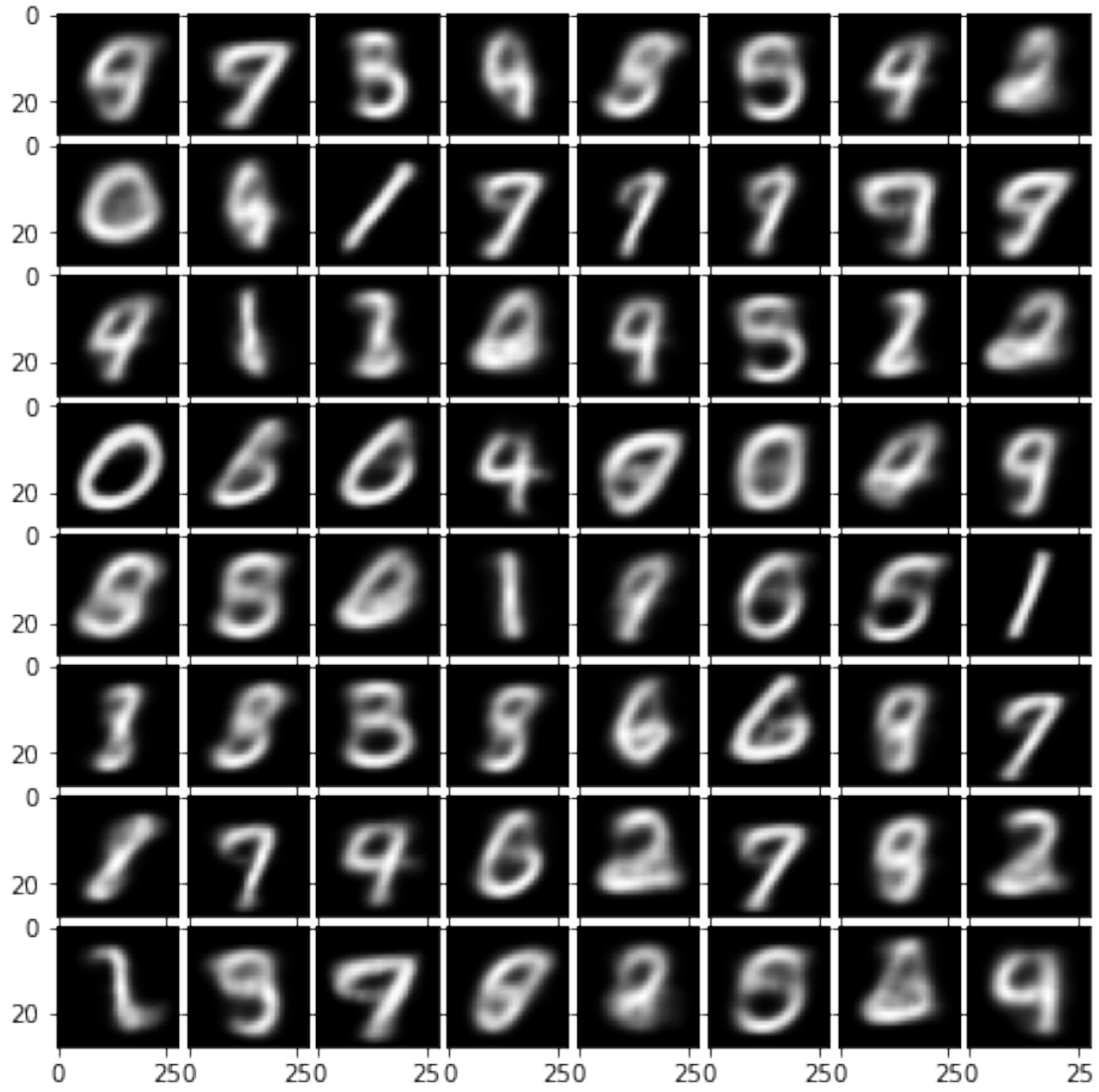

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

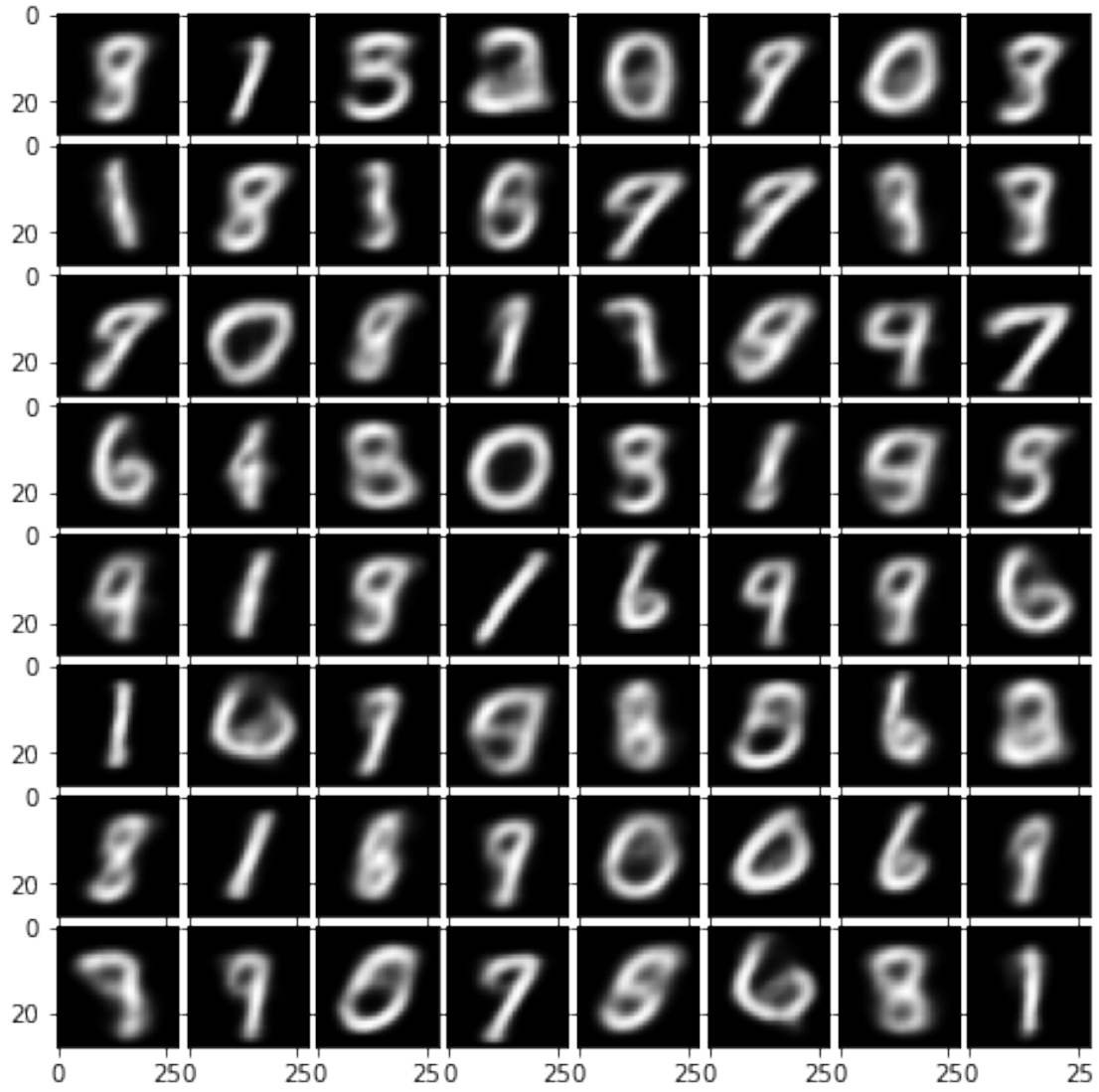

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

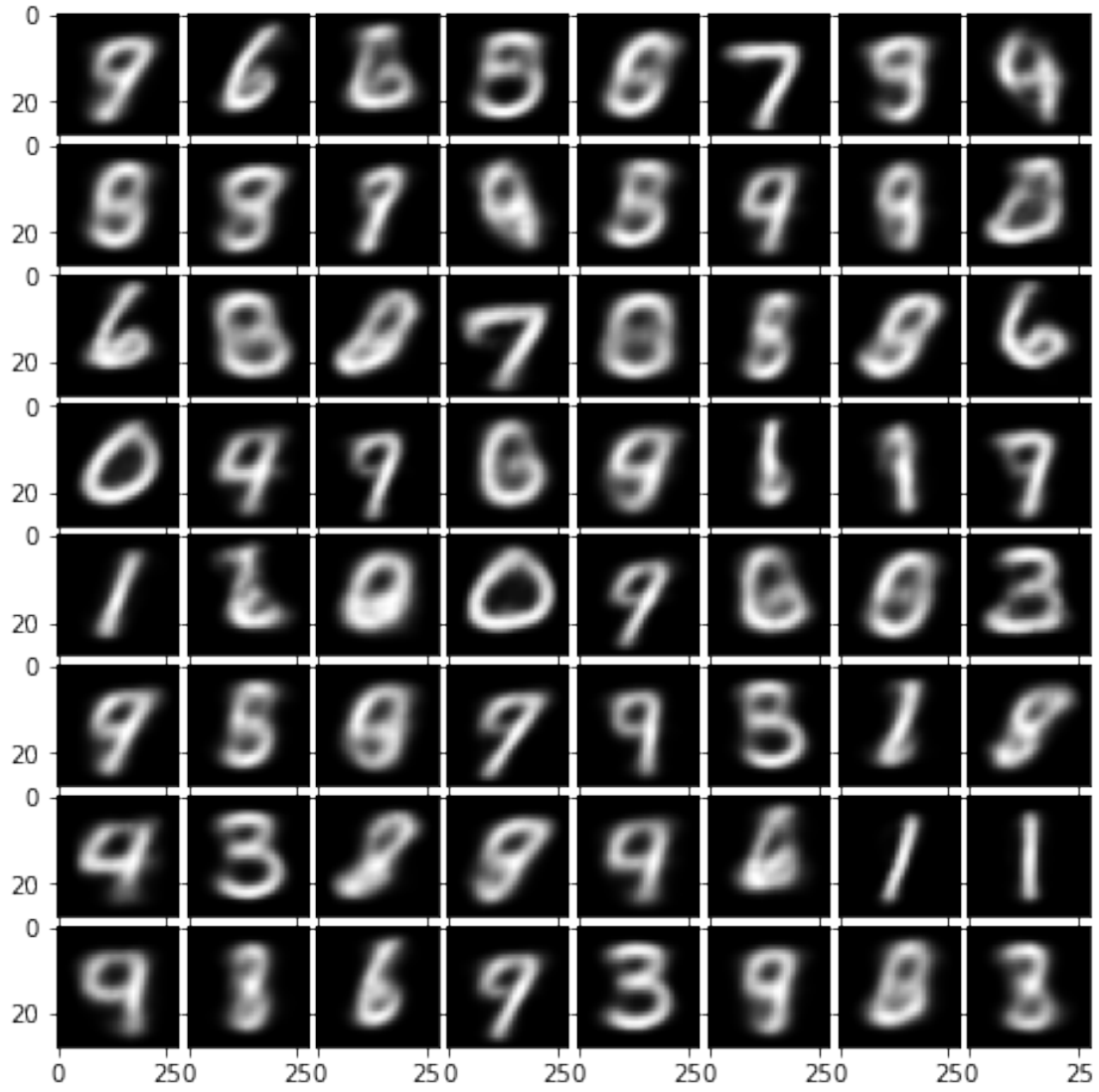

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

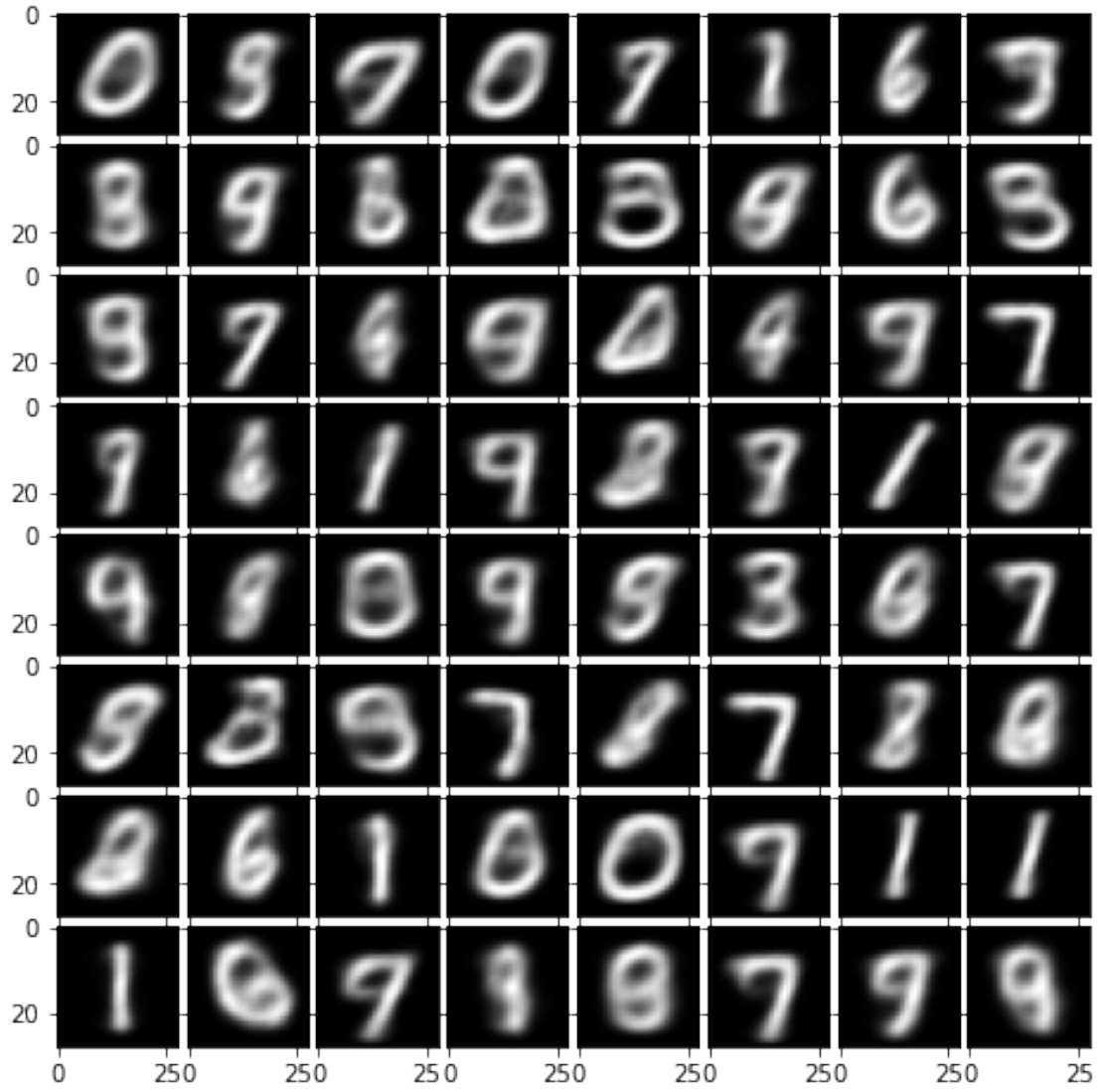

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

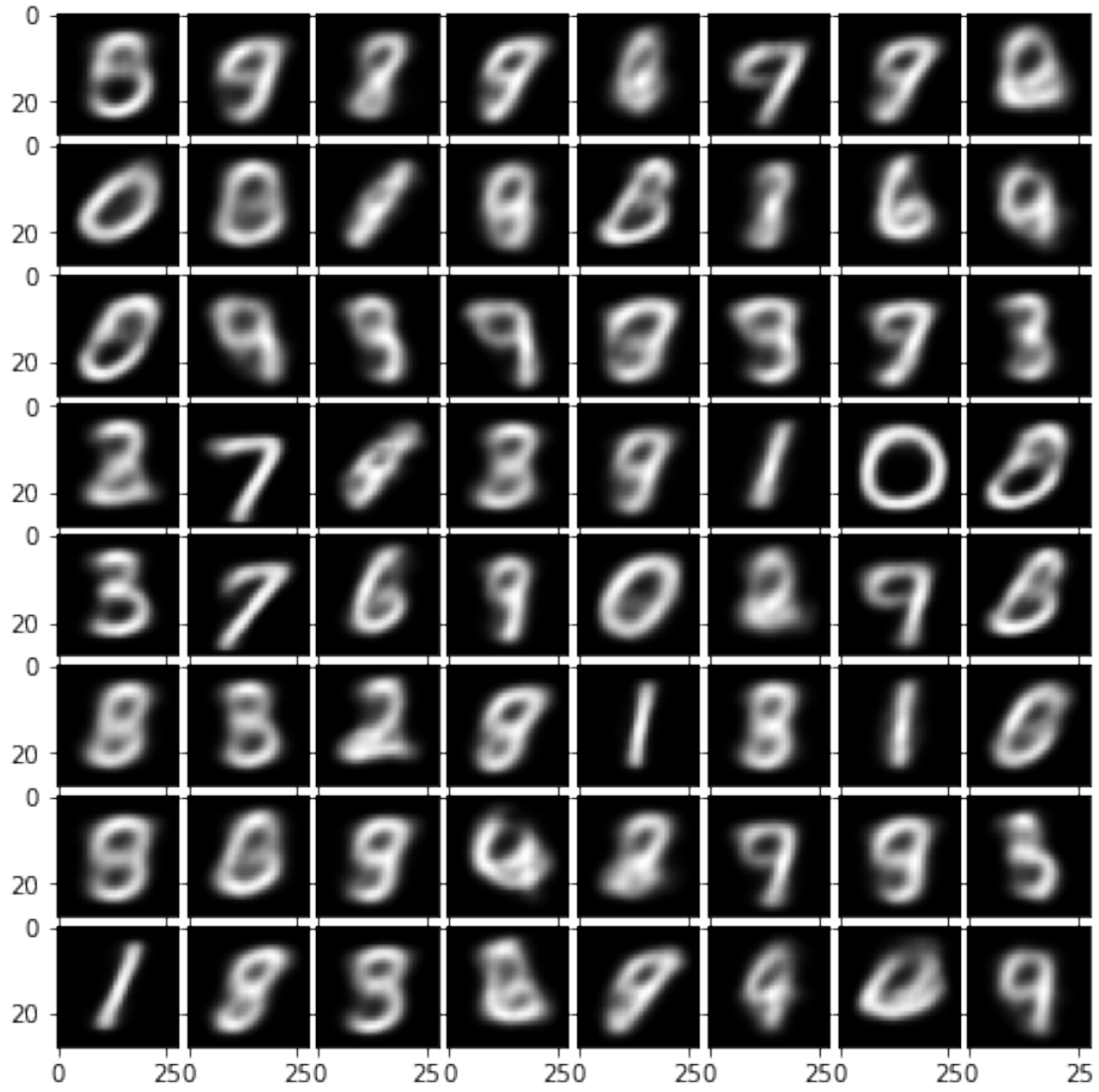

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

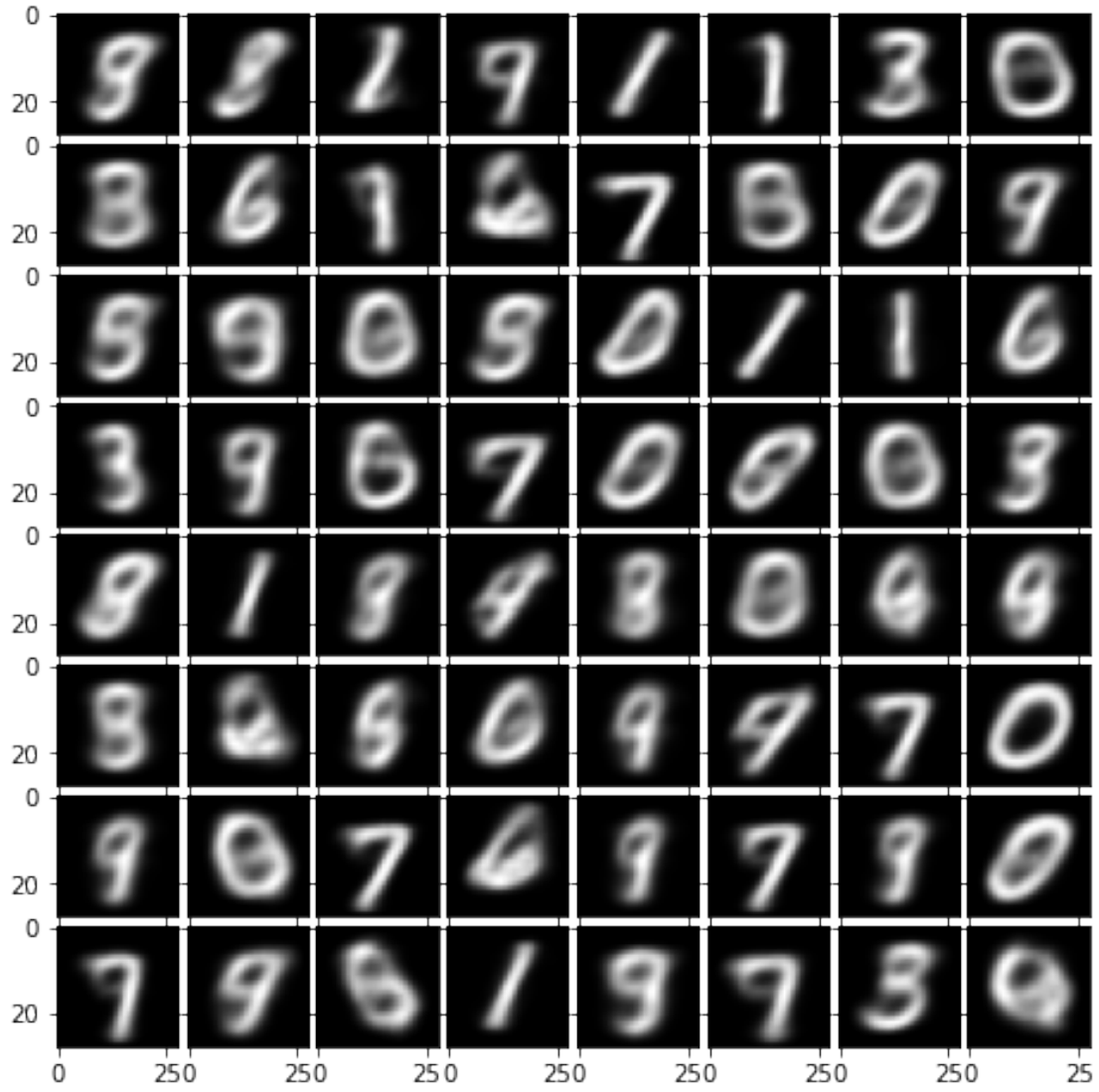

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

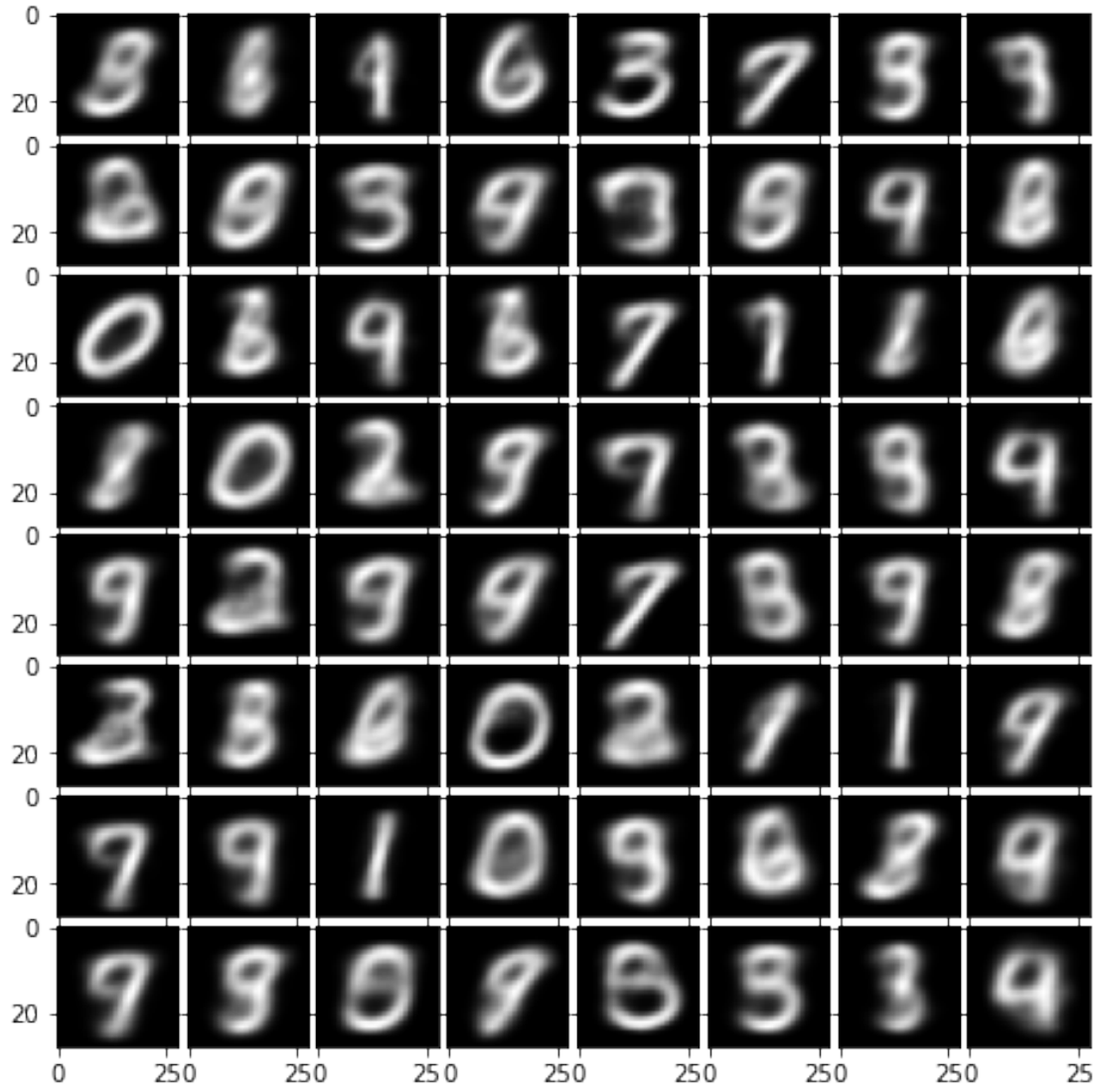

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

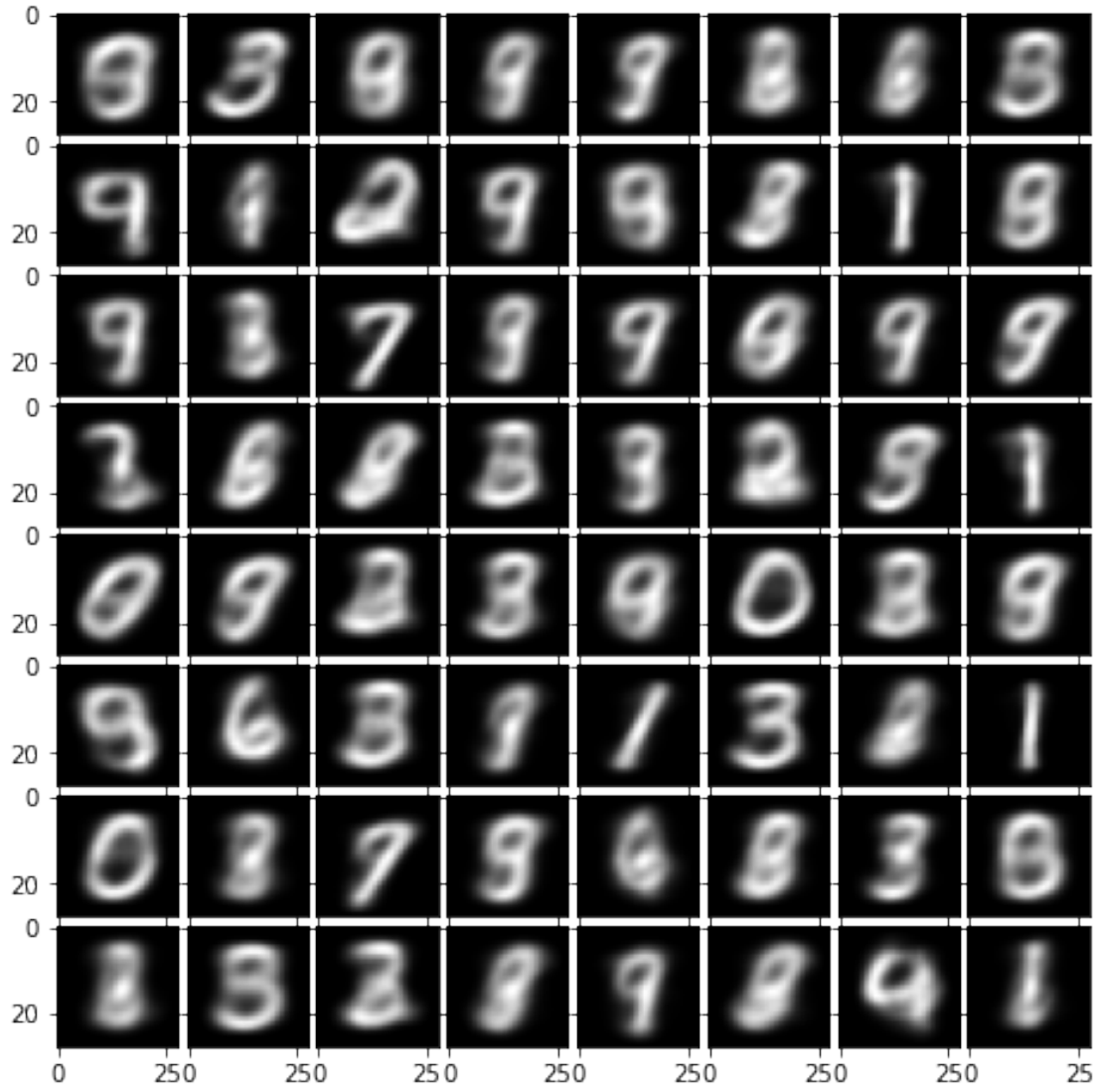

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

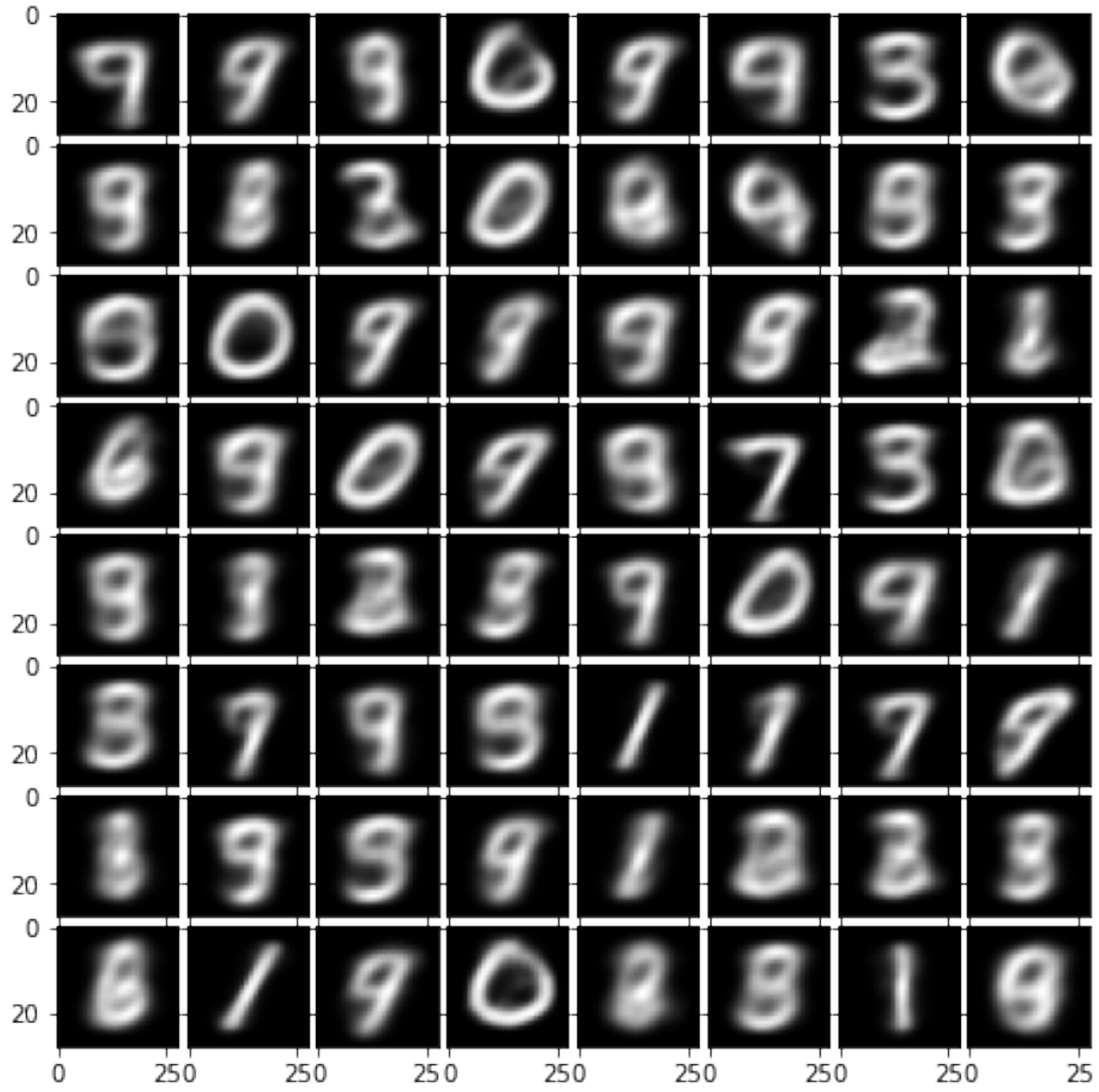

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

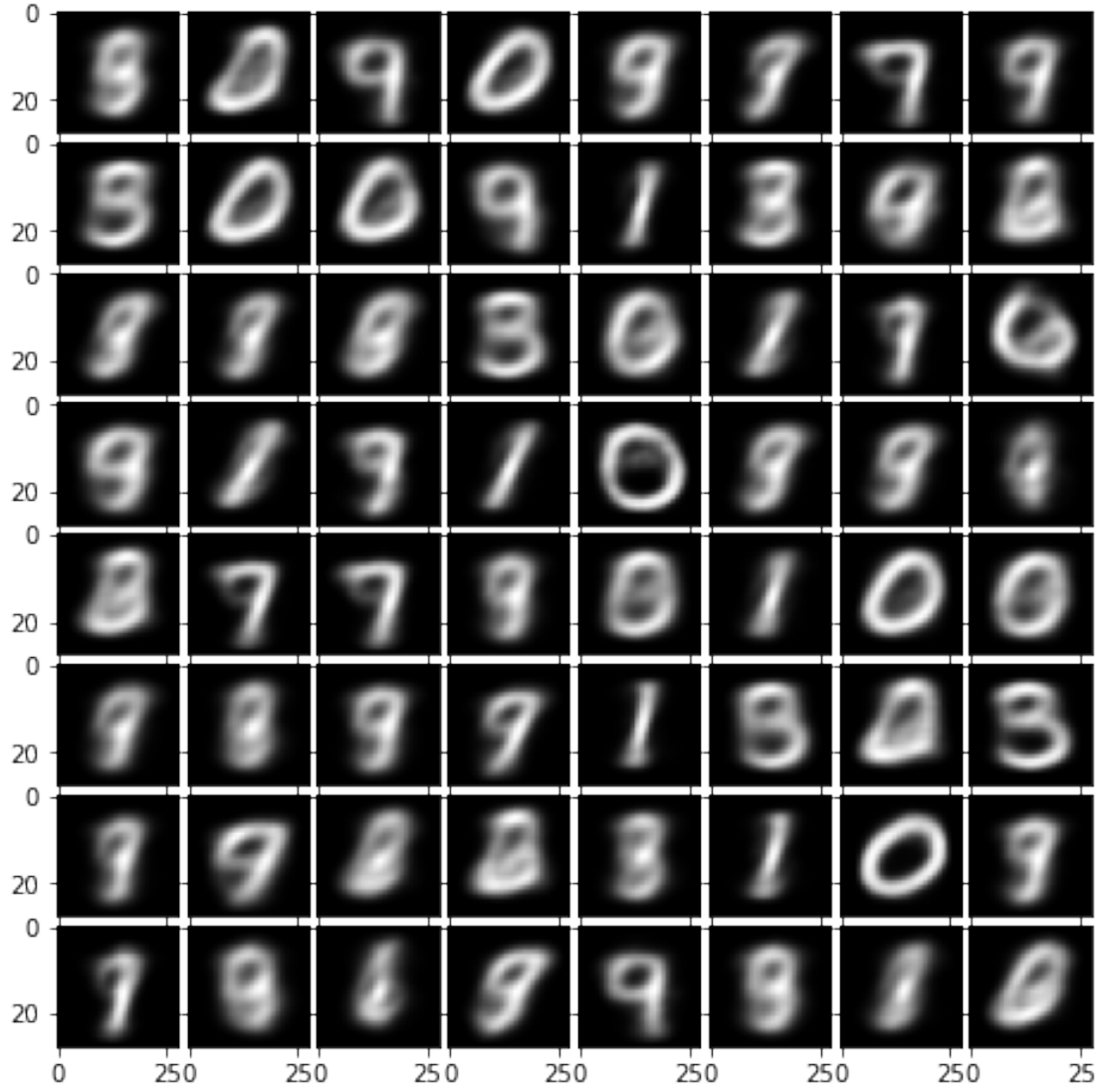

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

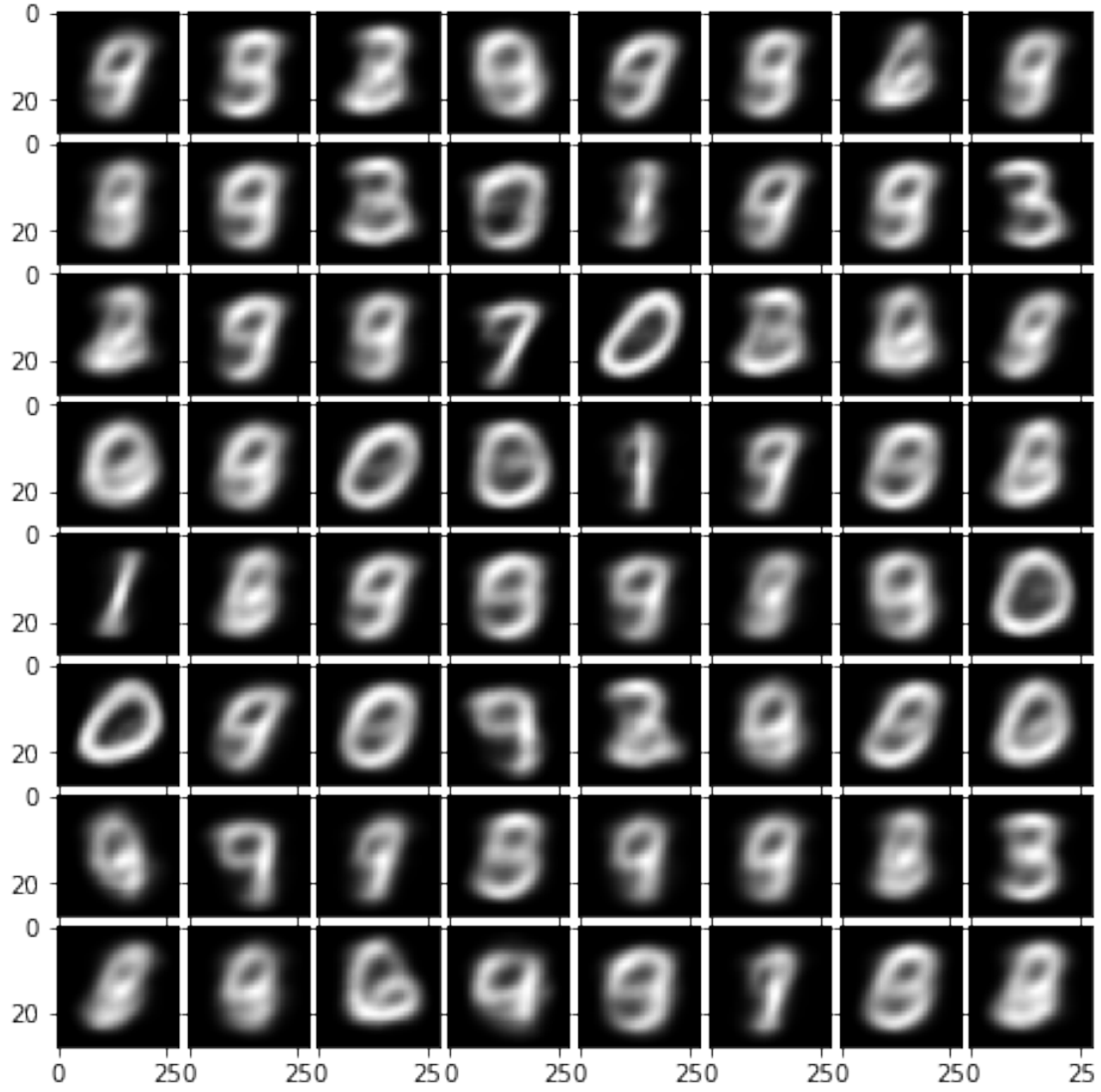

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

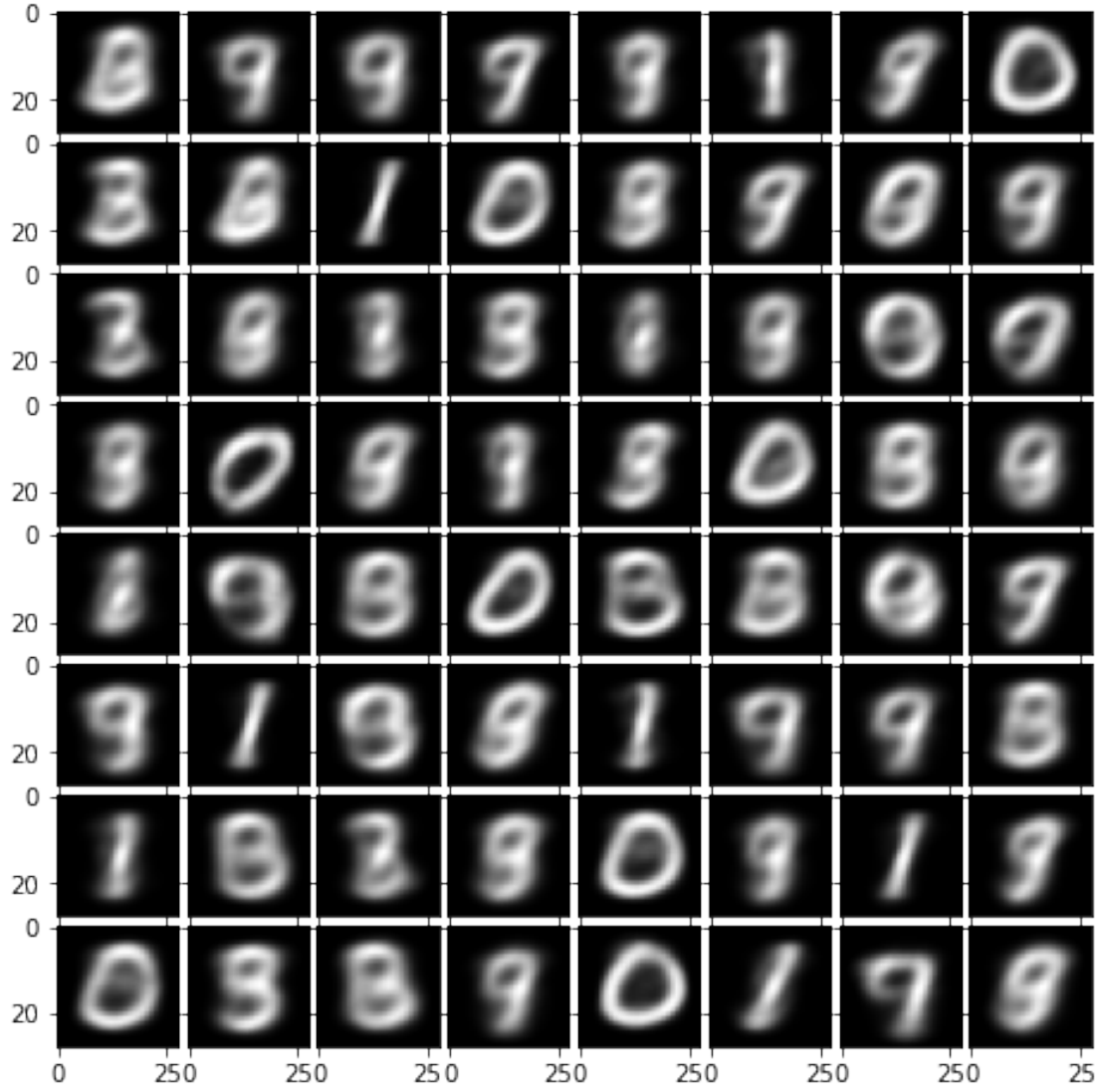

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

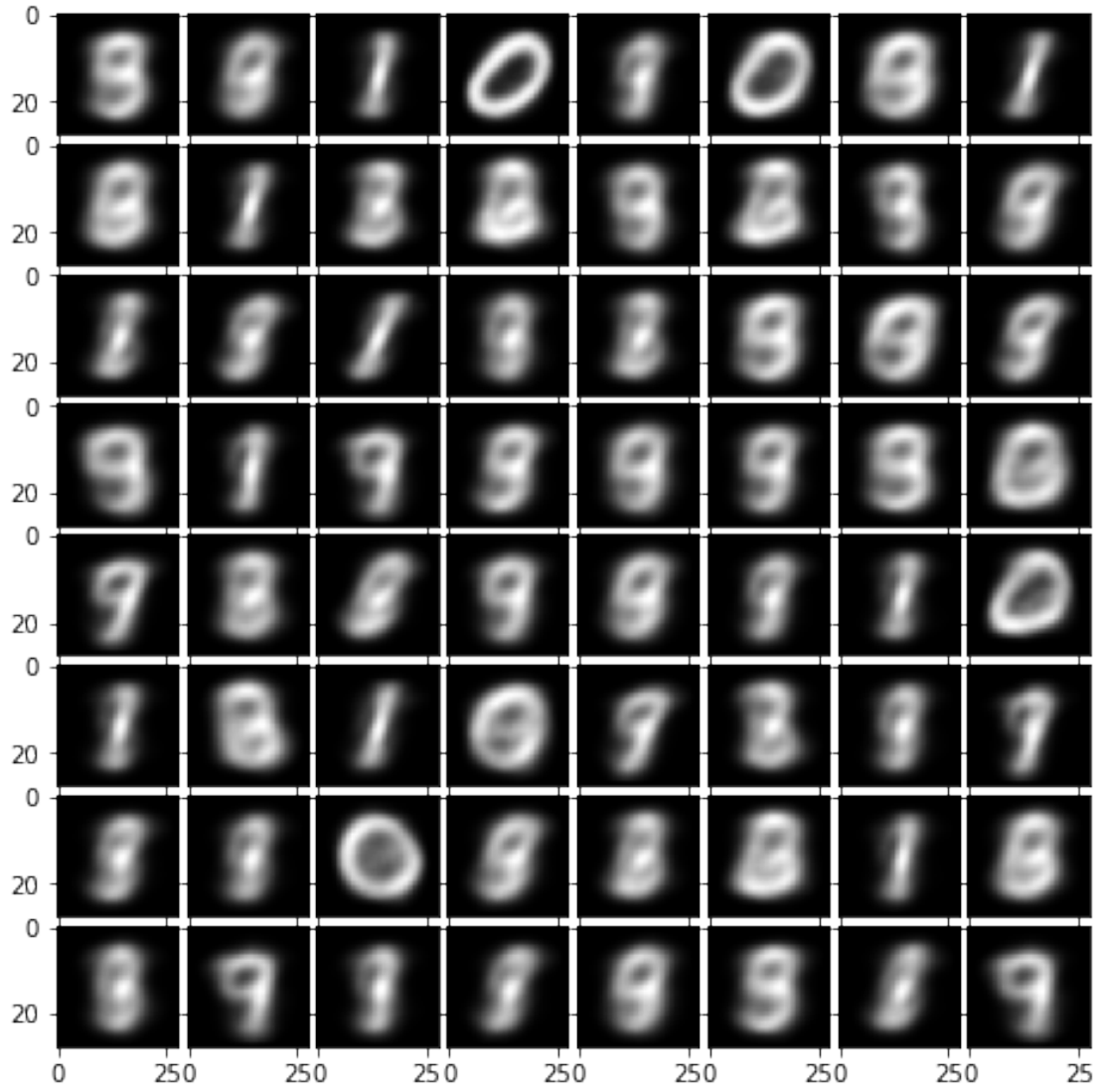

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

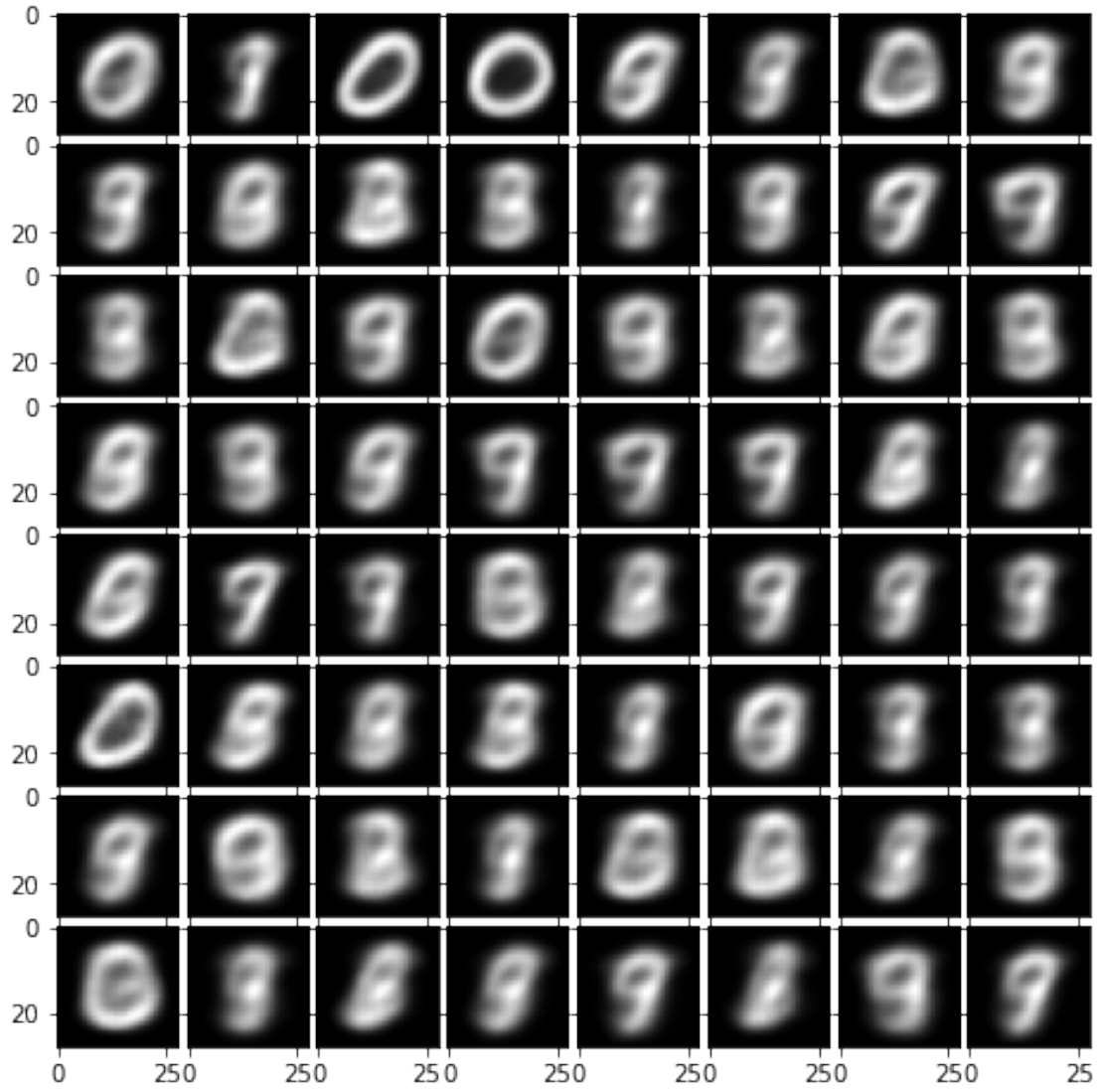

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

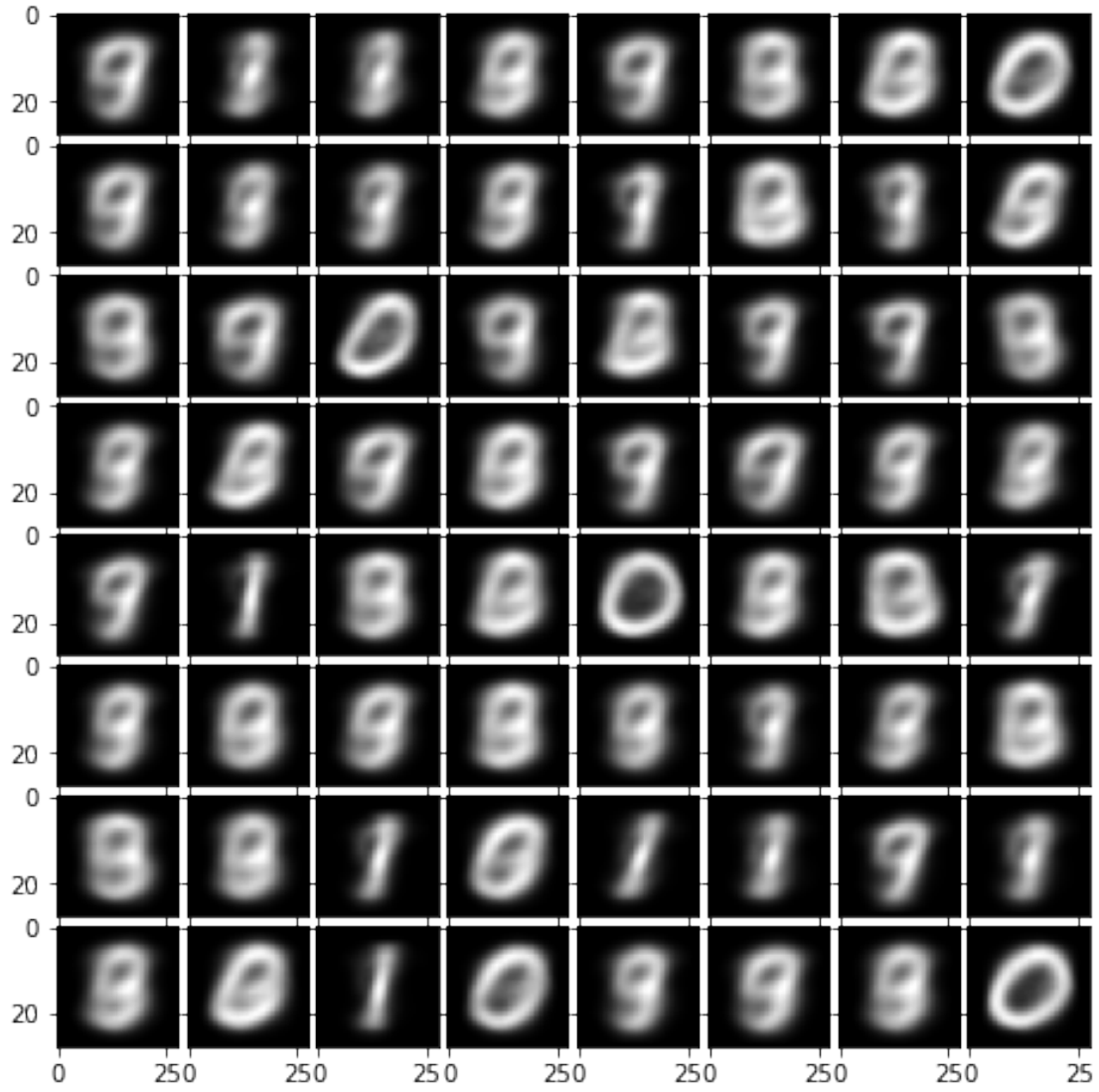

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

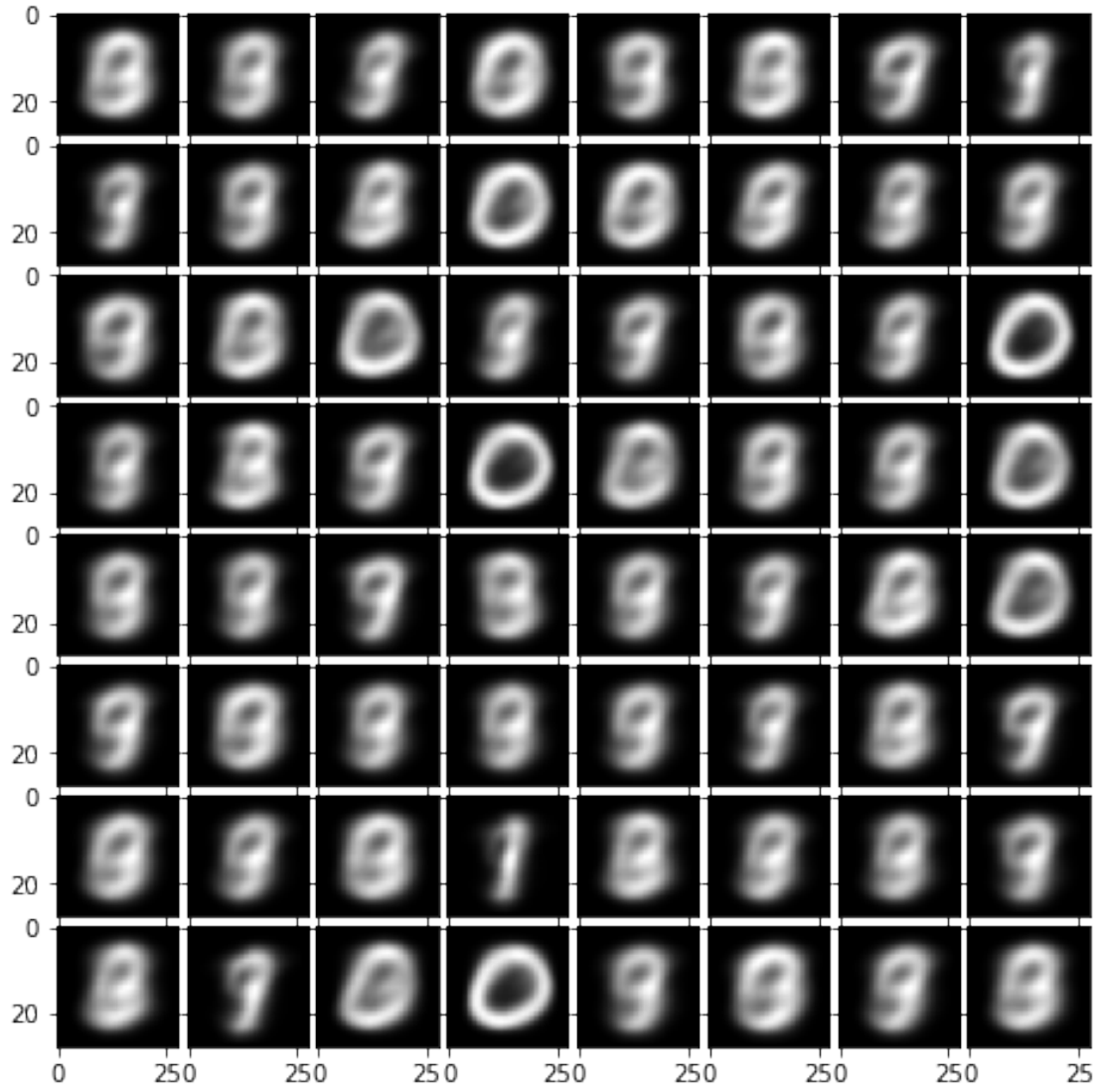

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

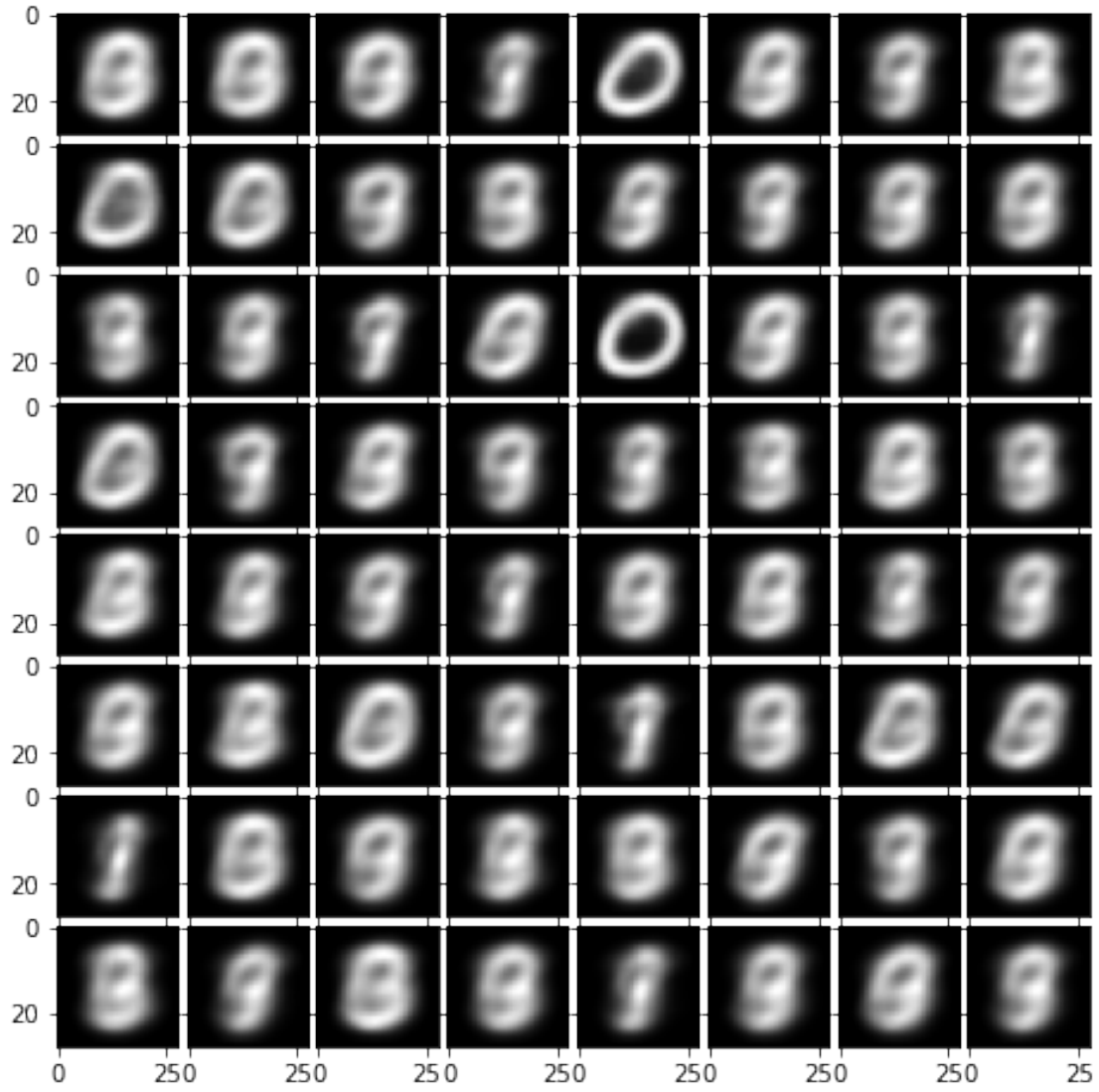

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

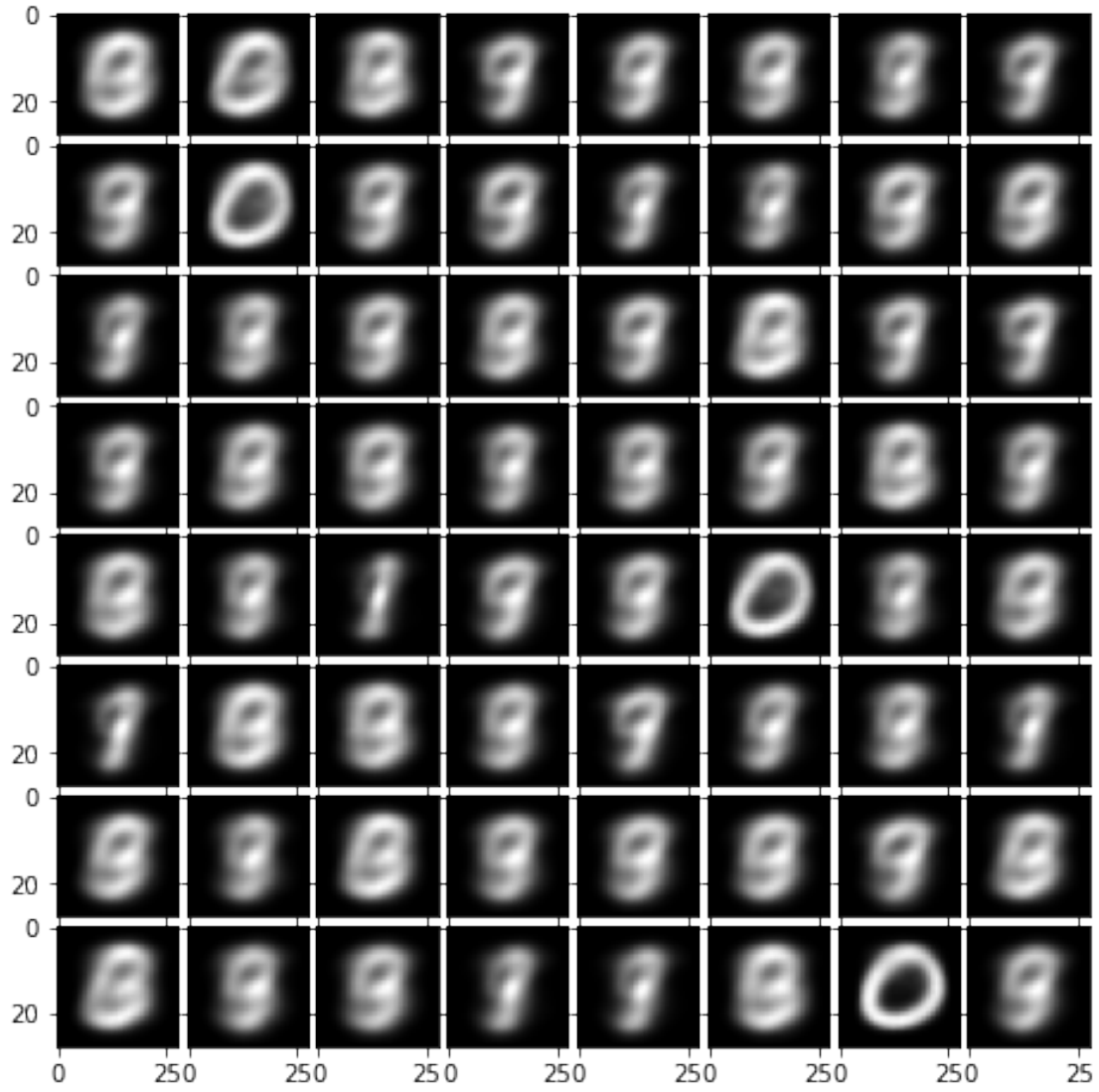

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

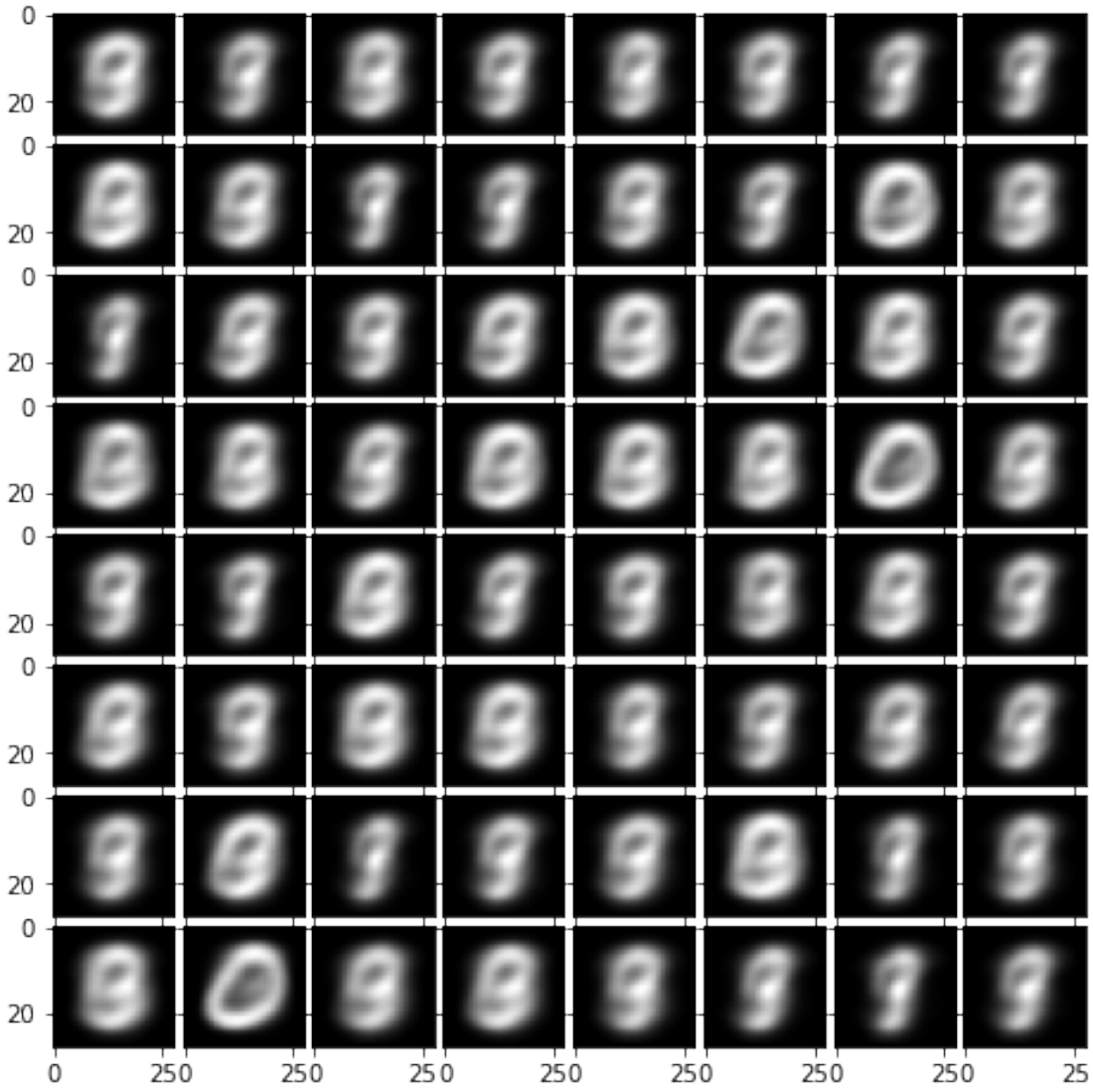

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

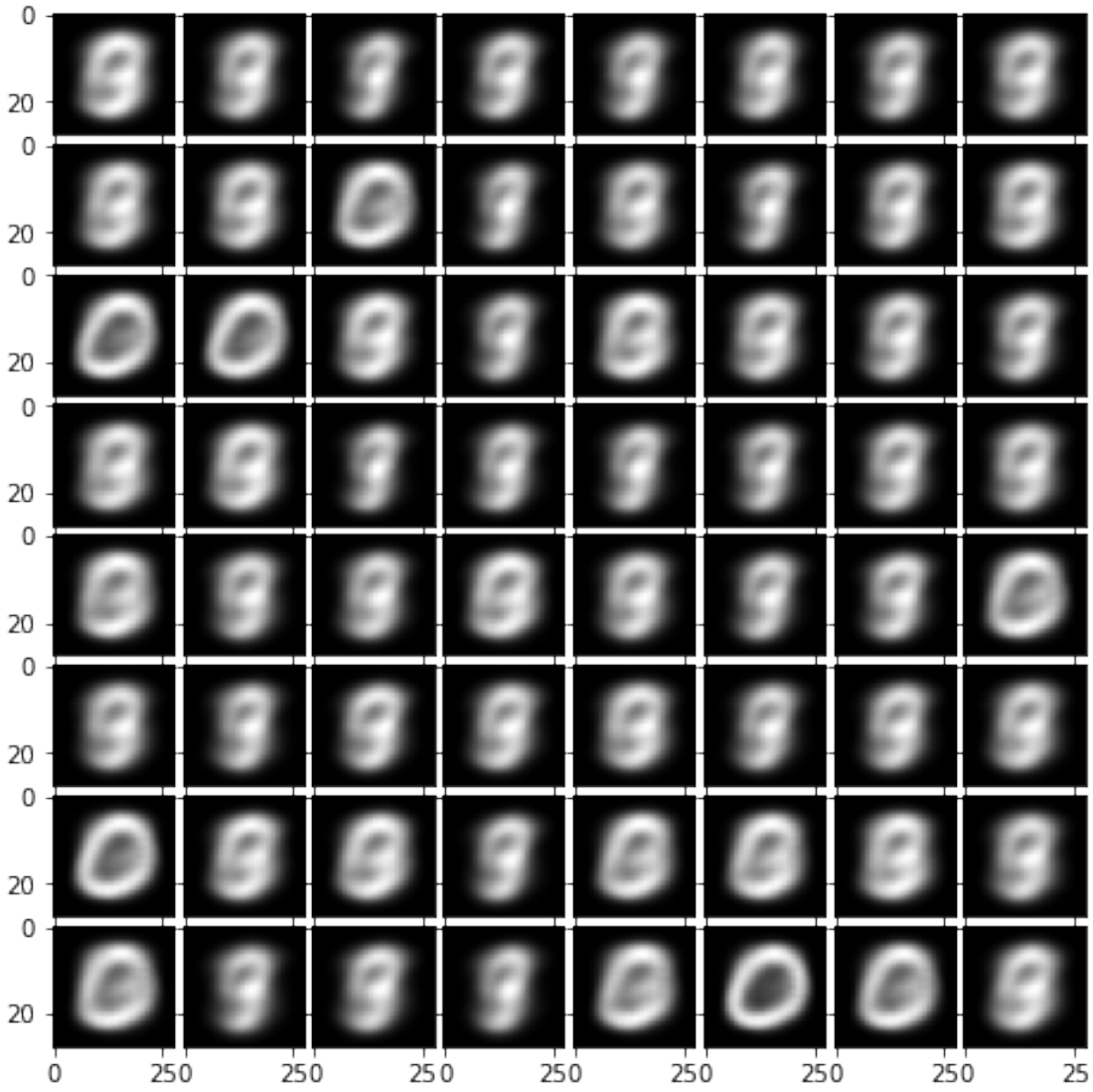

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

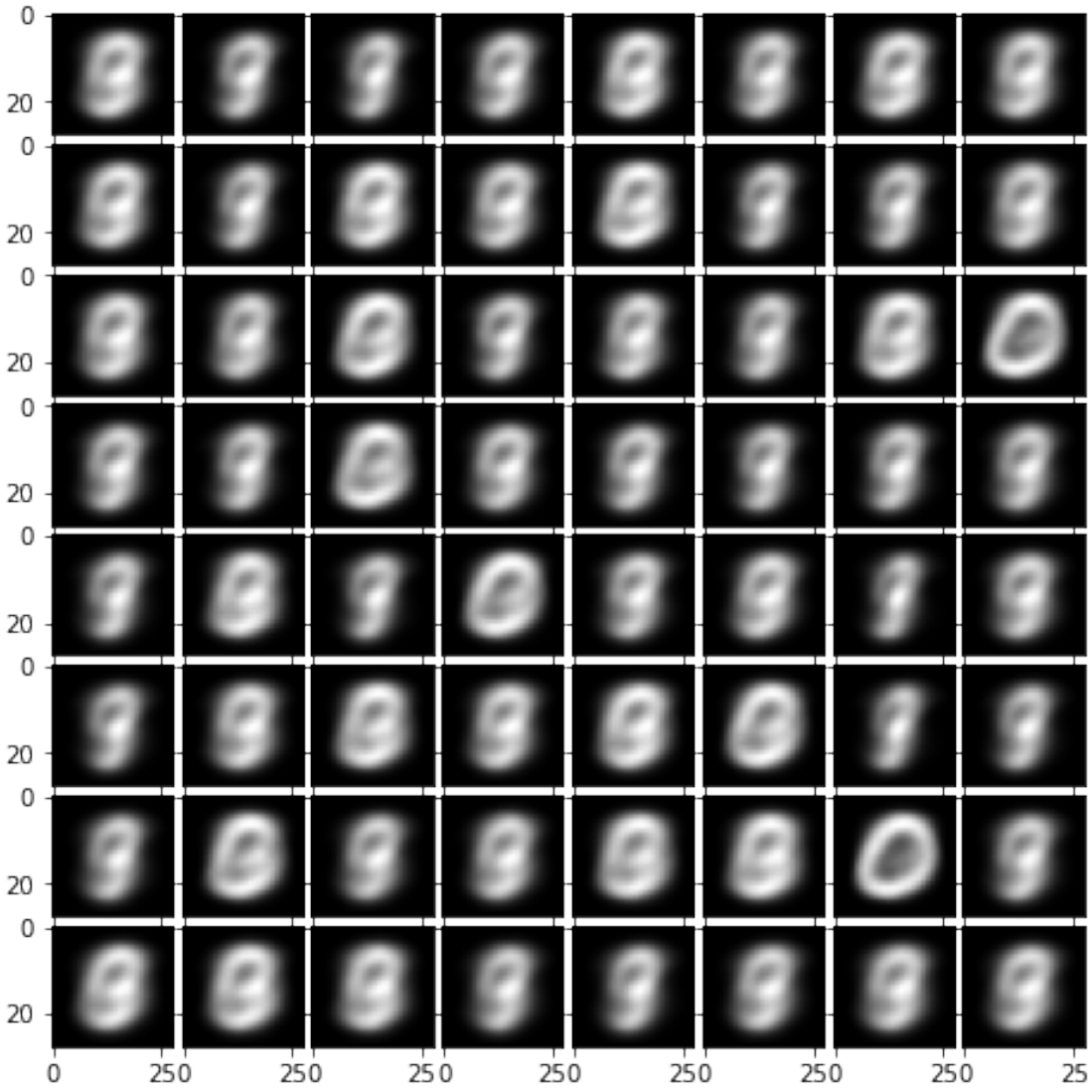

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

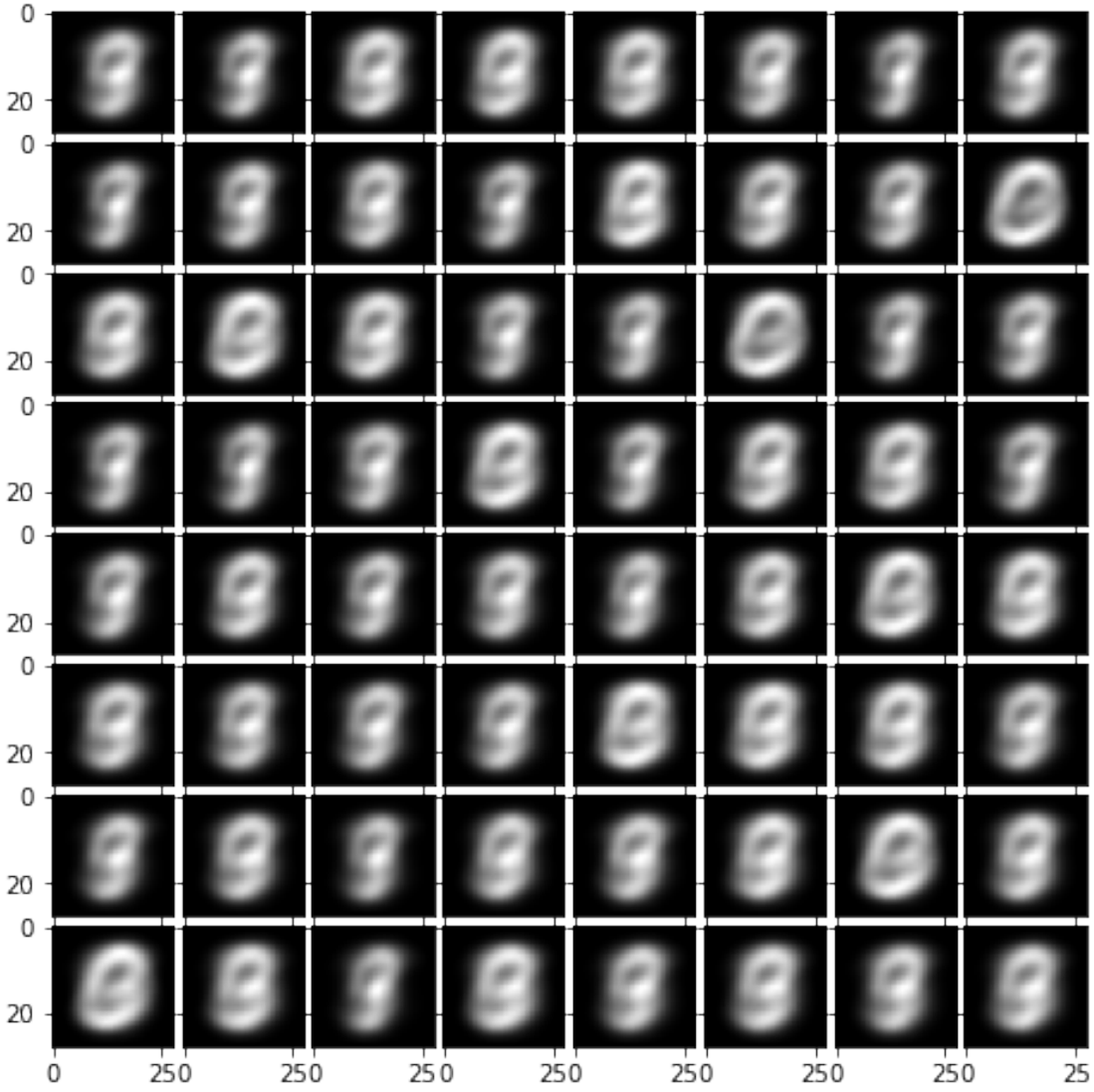

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

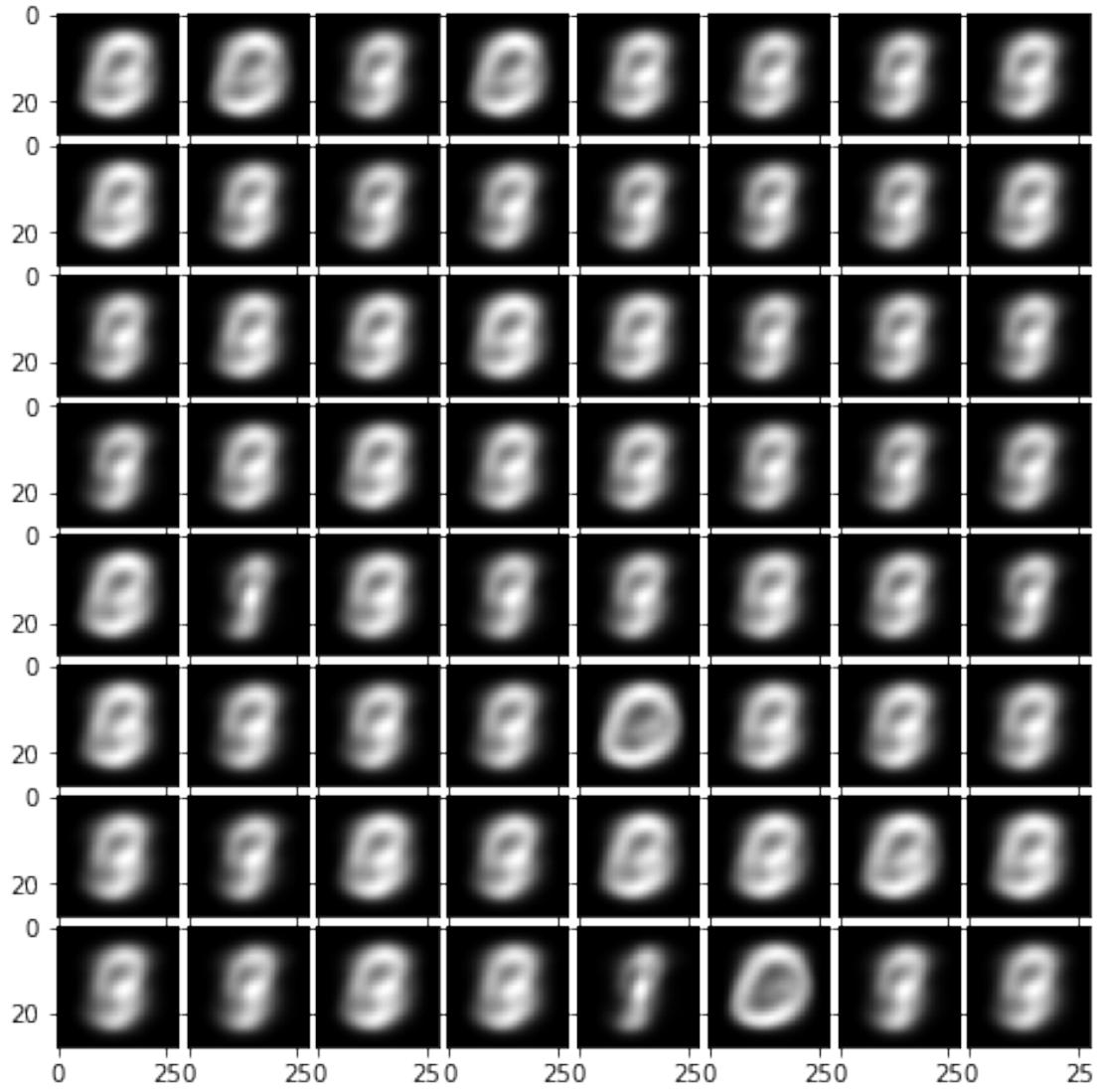

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

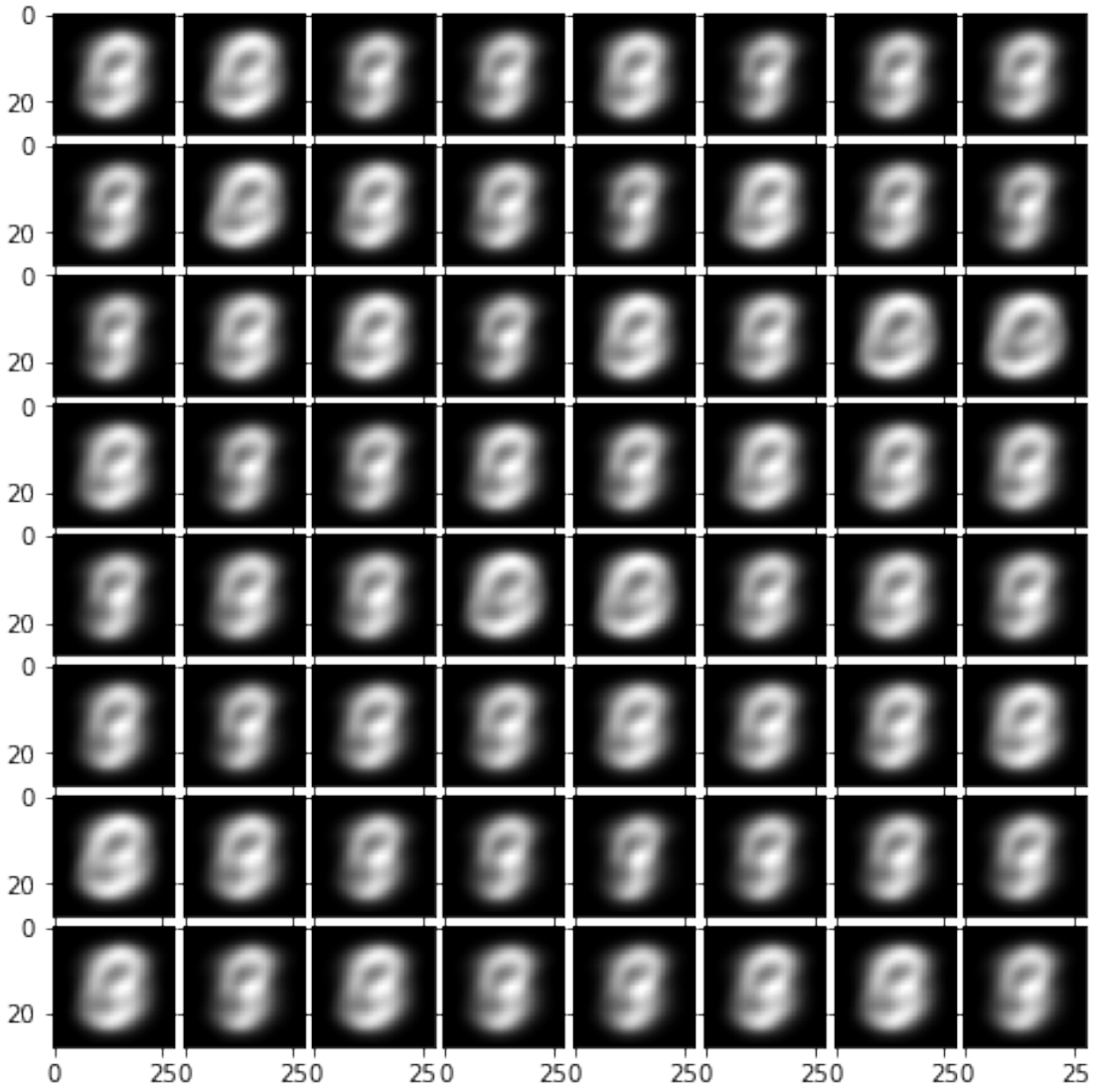

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

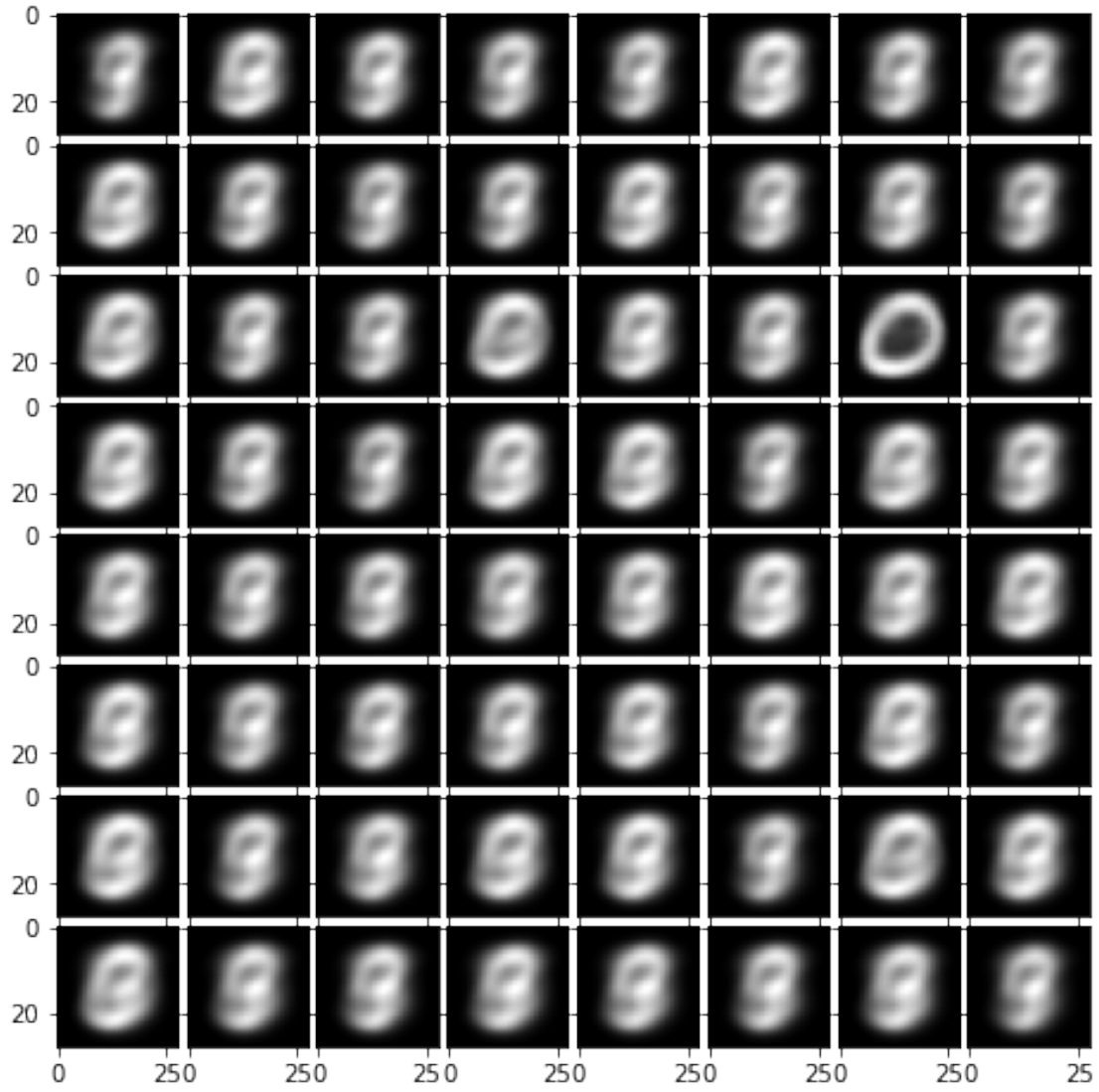

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

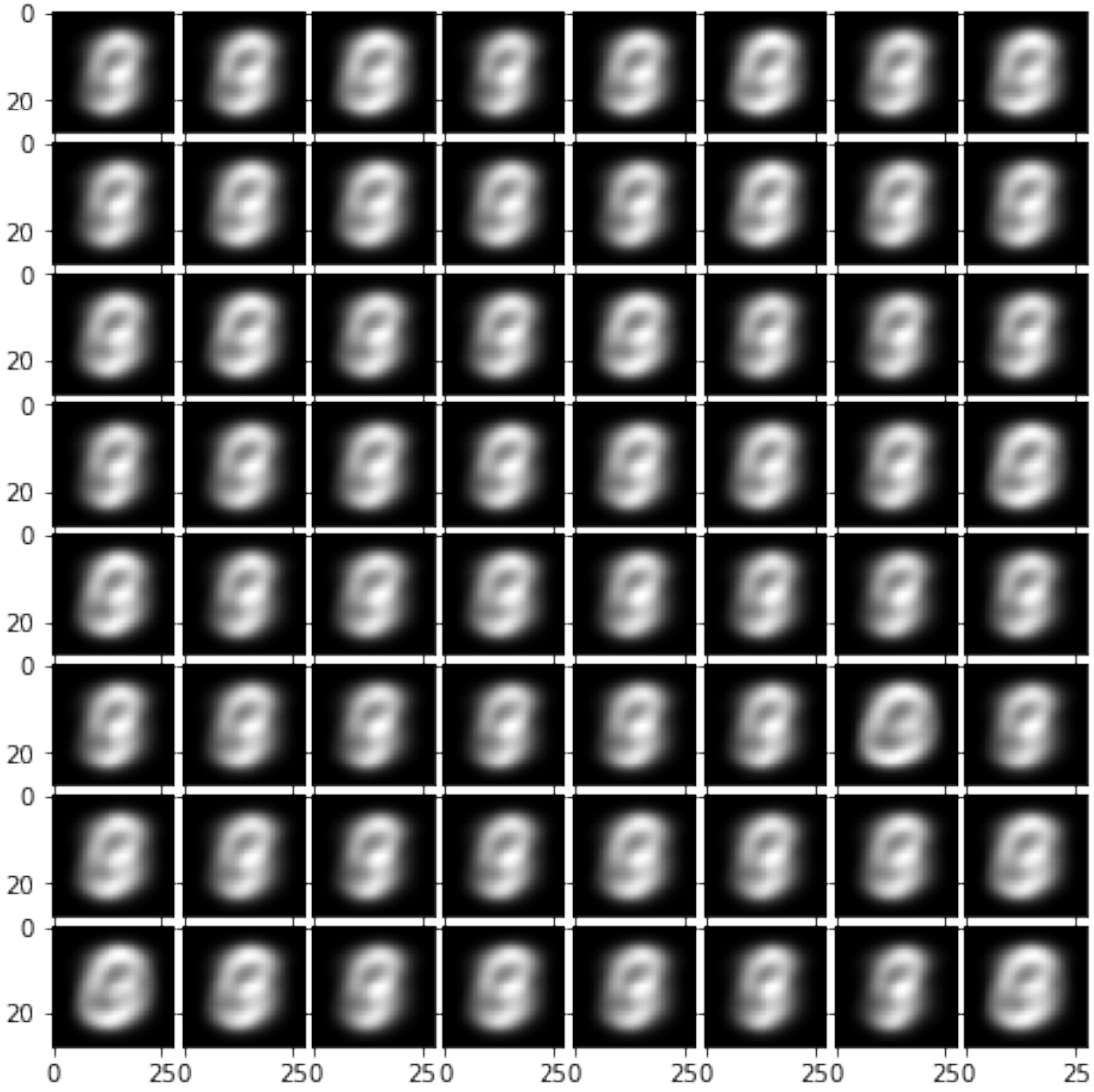

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

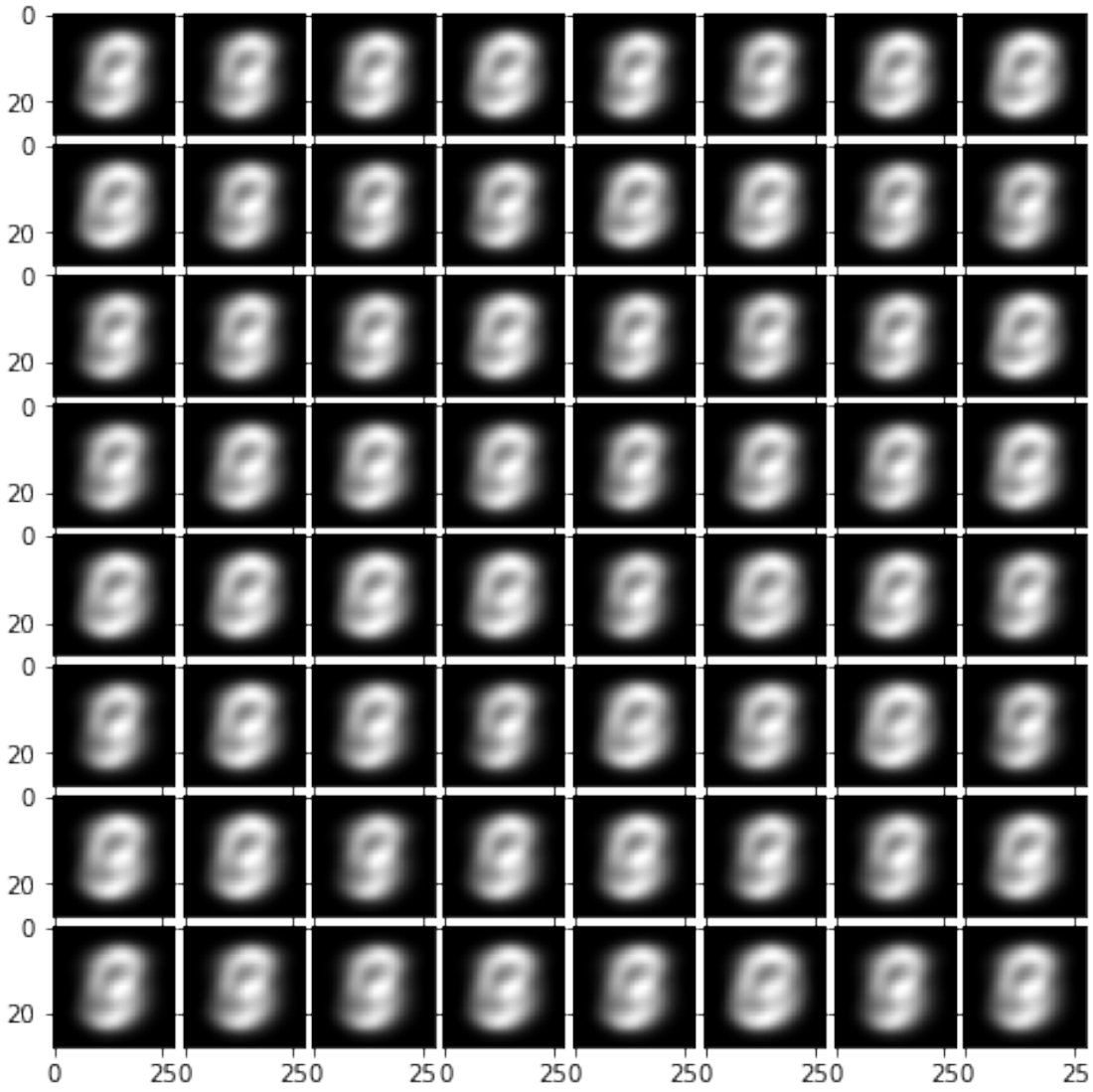

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

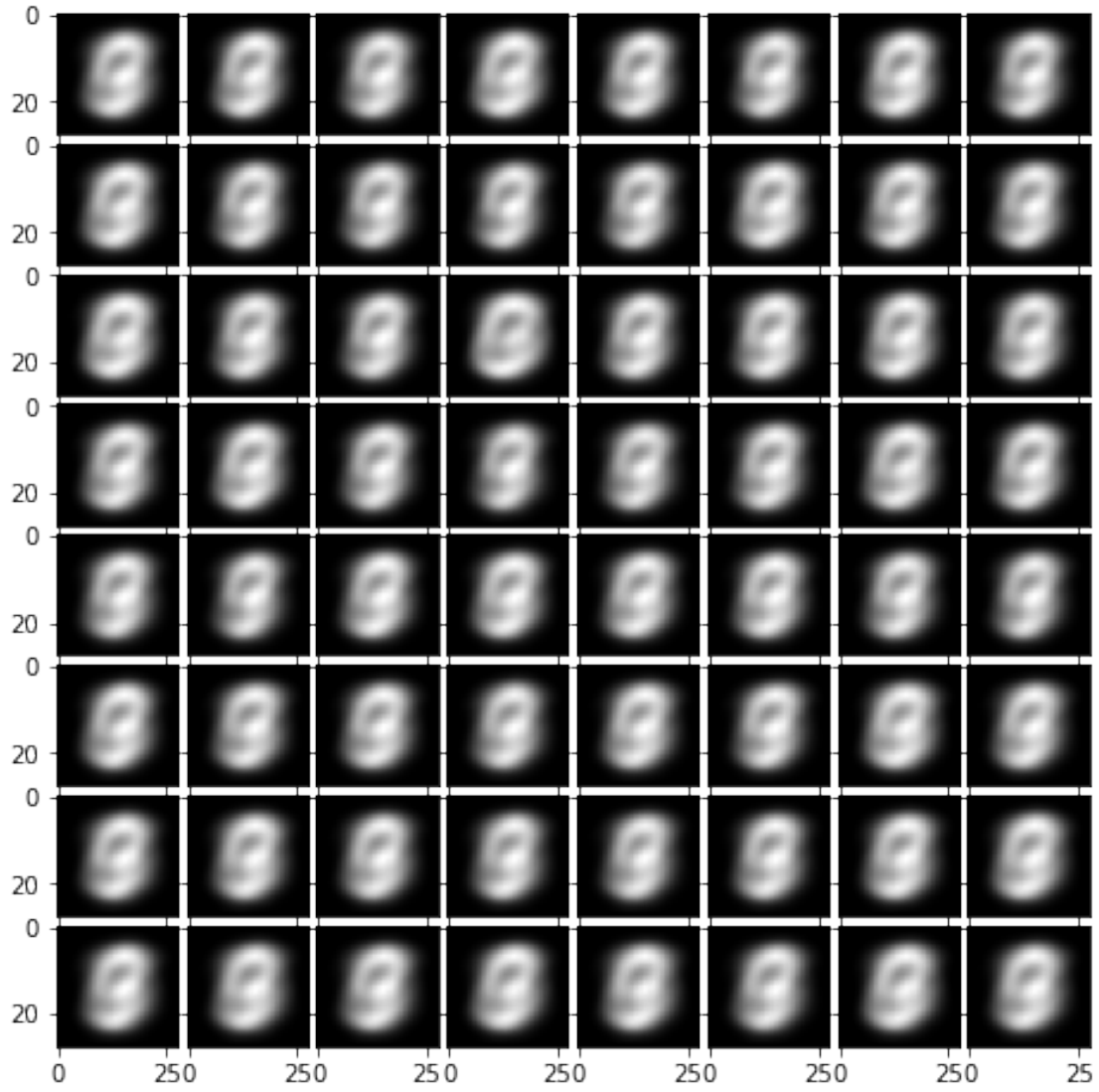

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

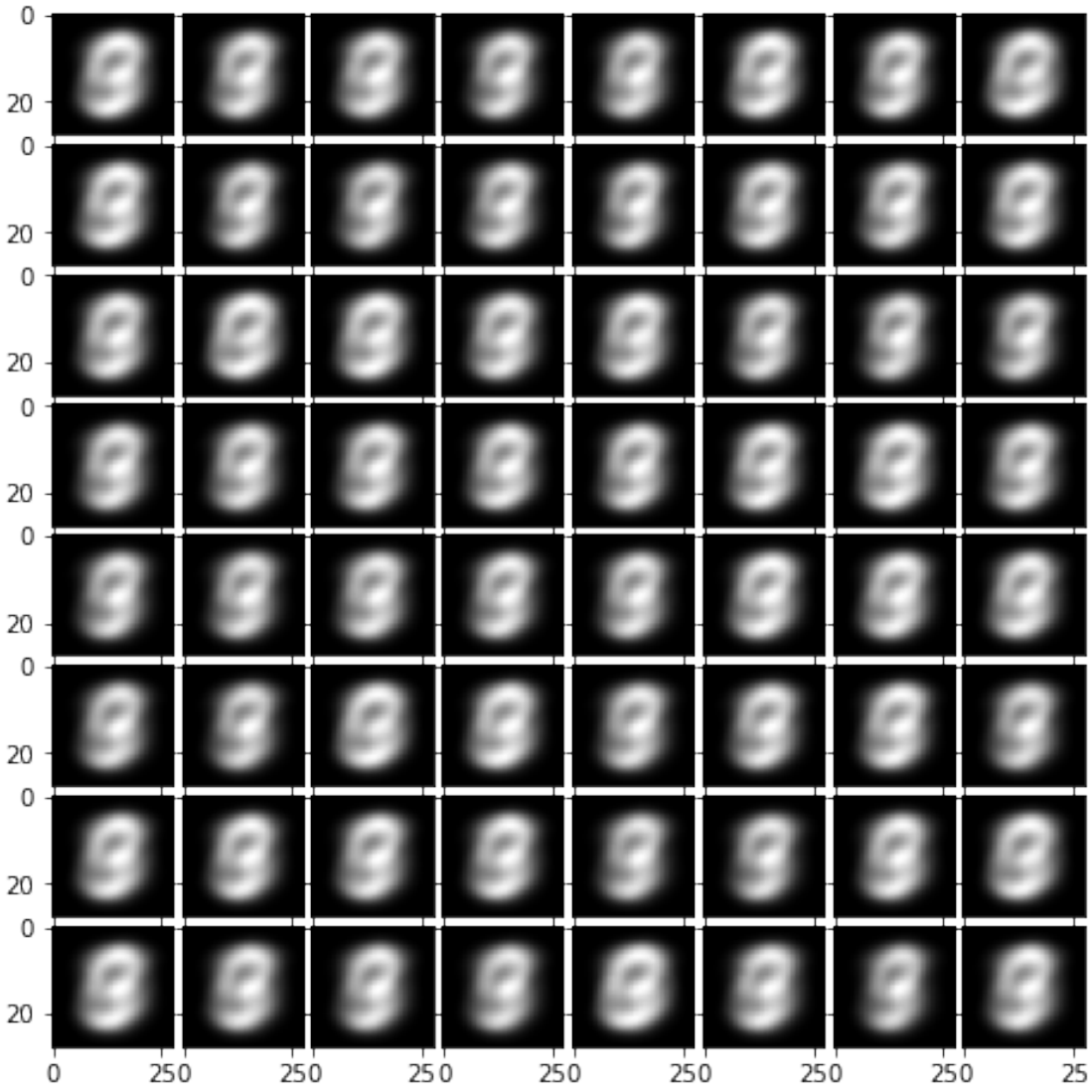

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

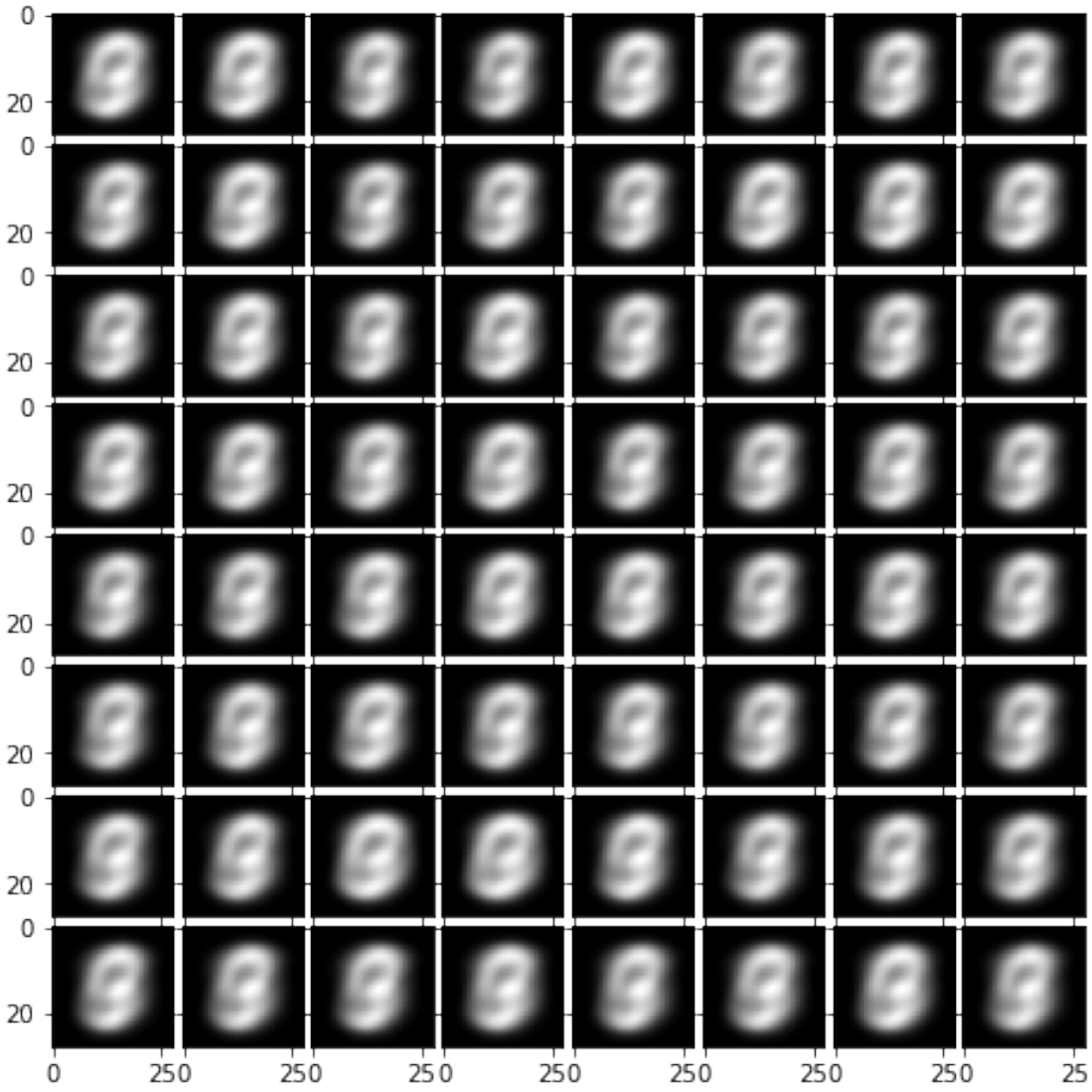

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
250
250
250
250
250
250
250
25

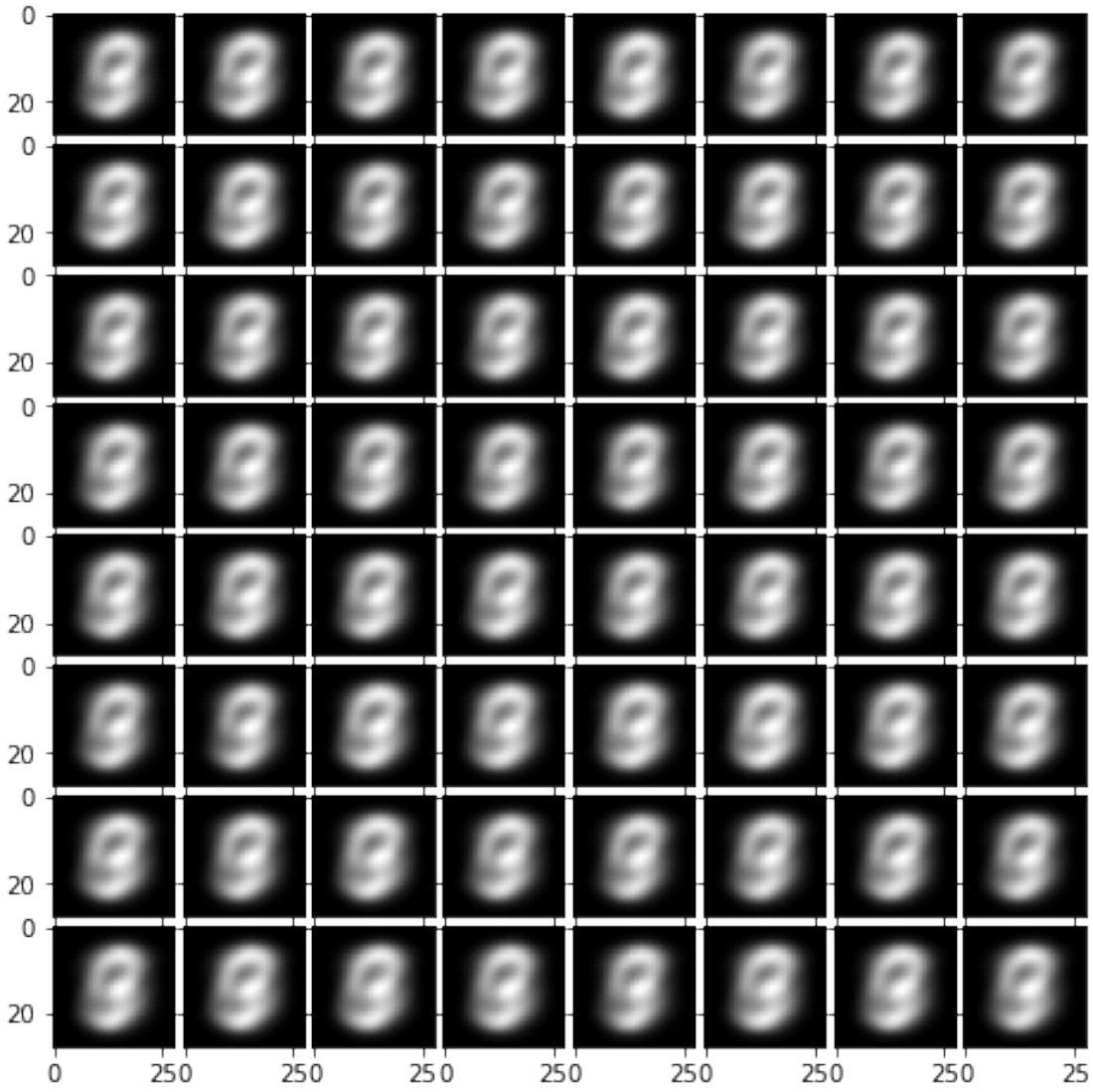

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

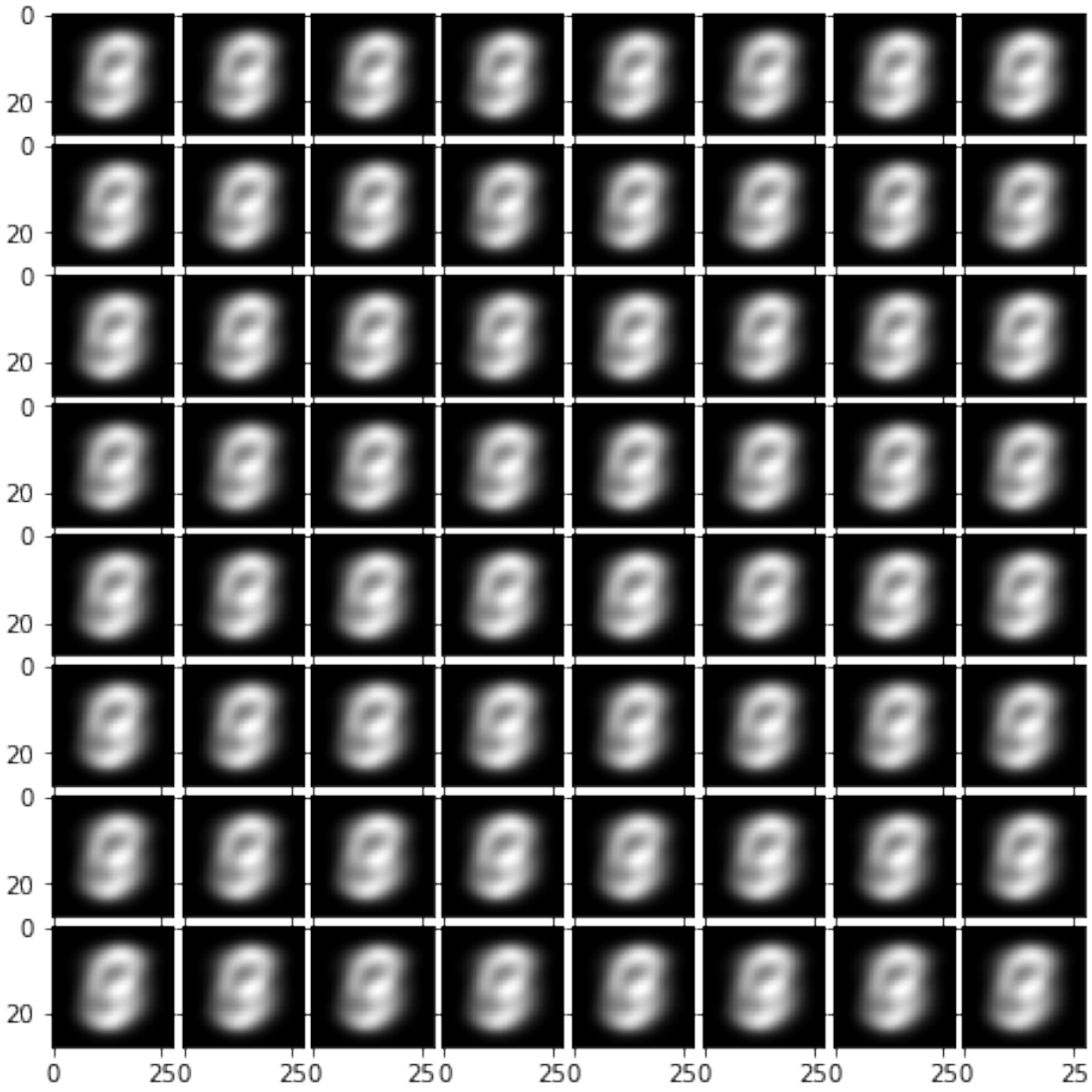

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

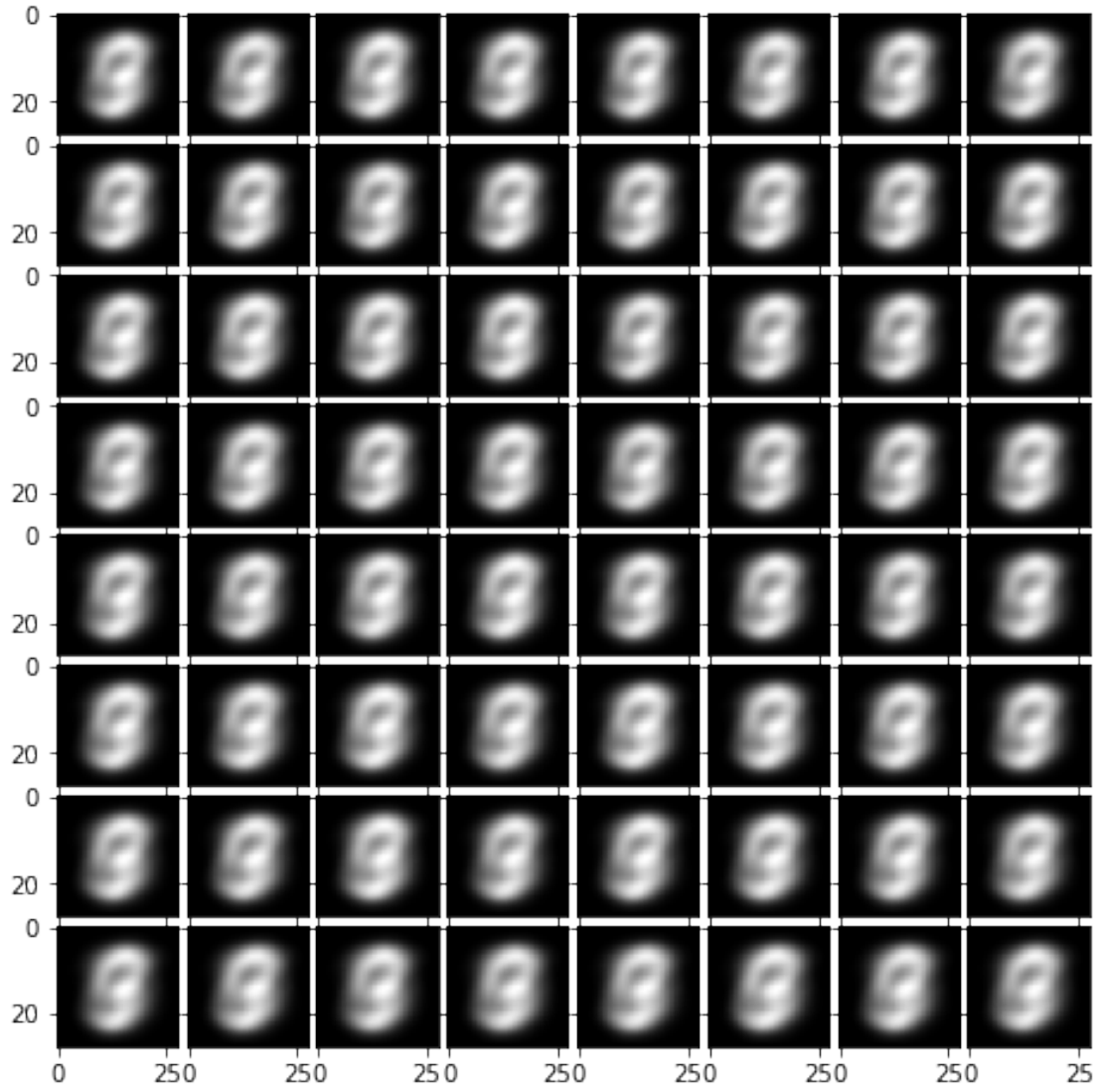

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

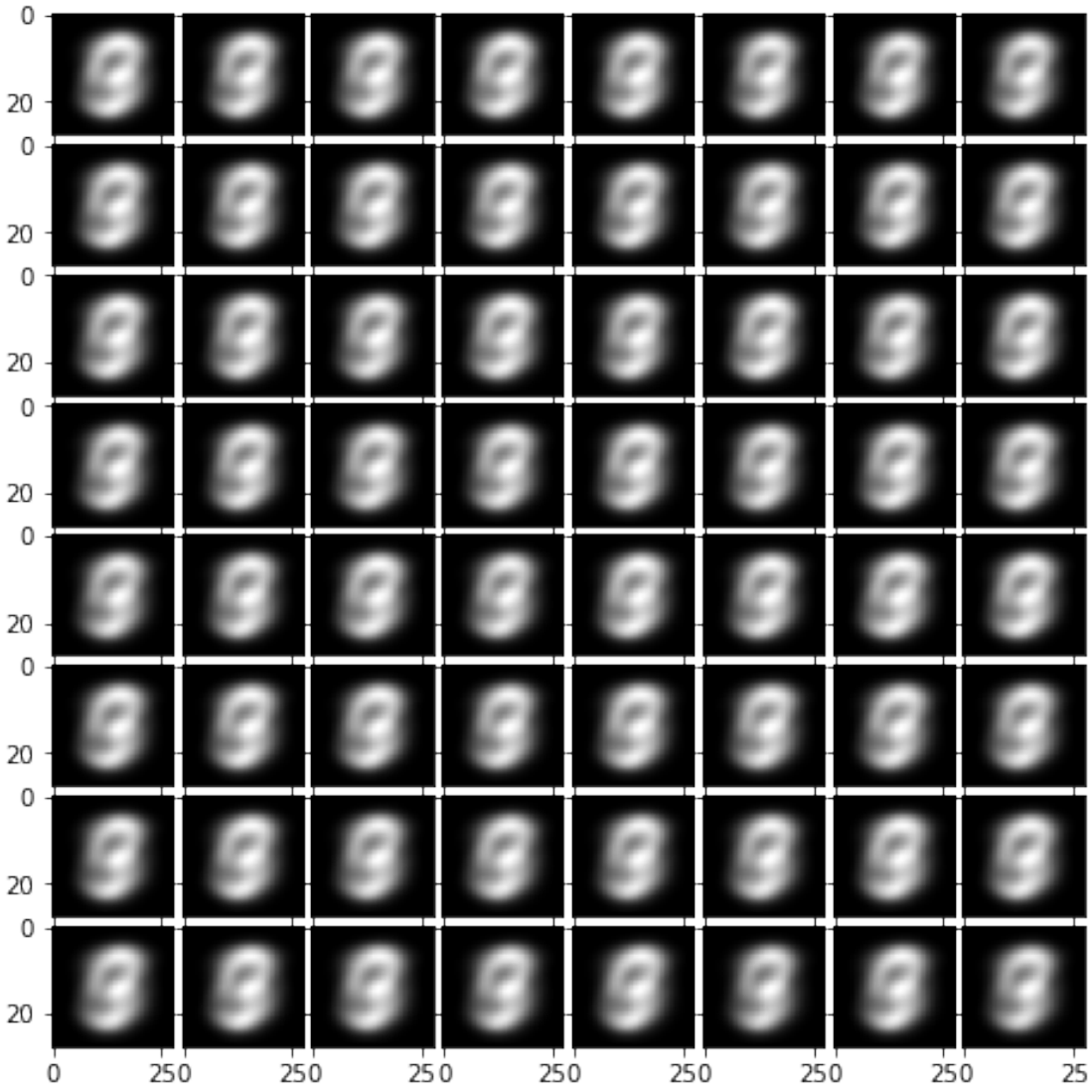

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

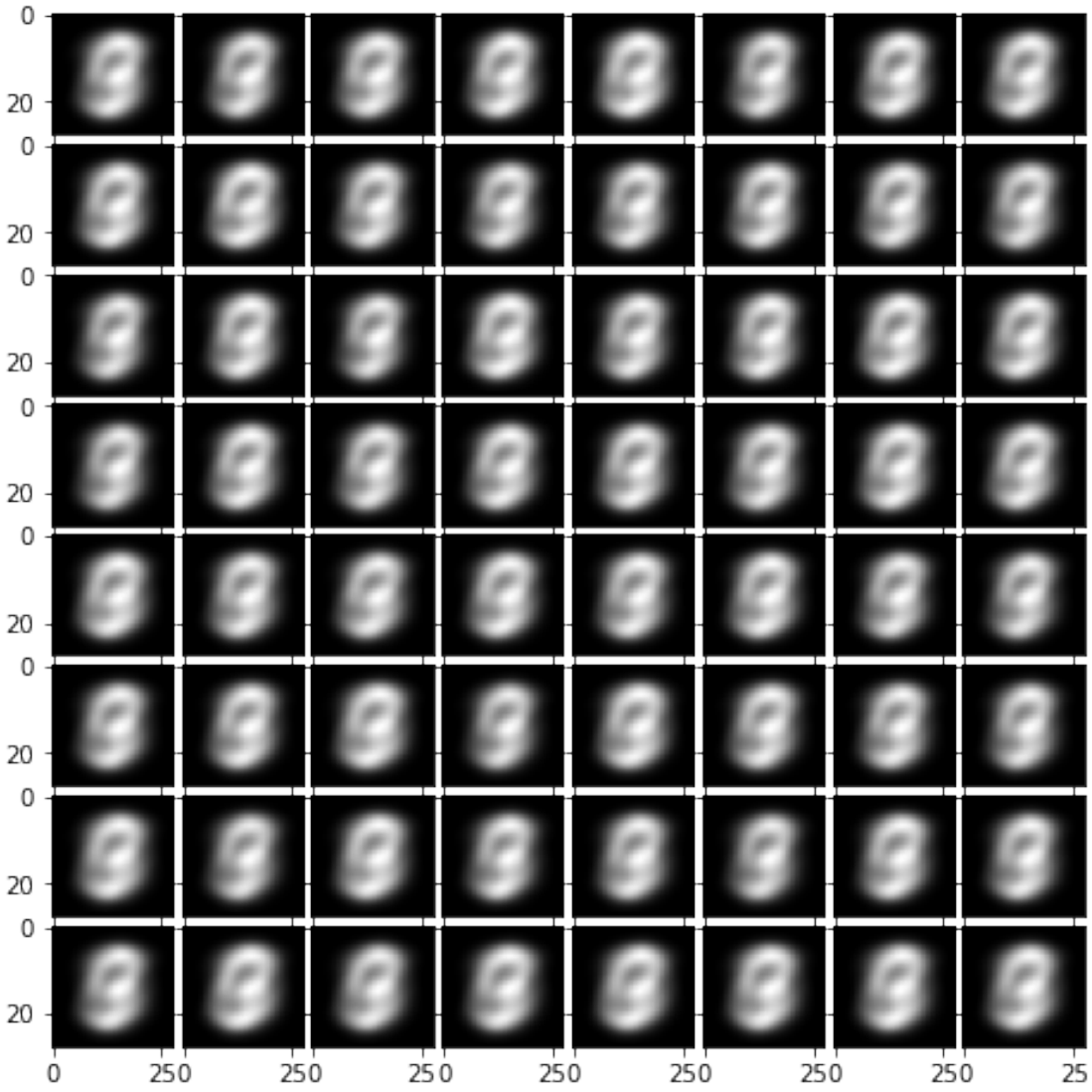

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25

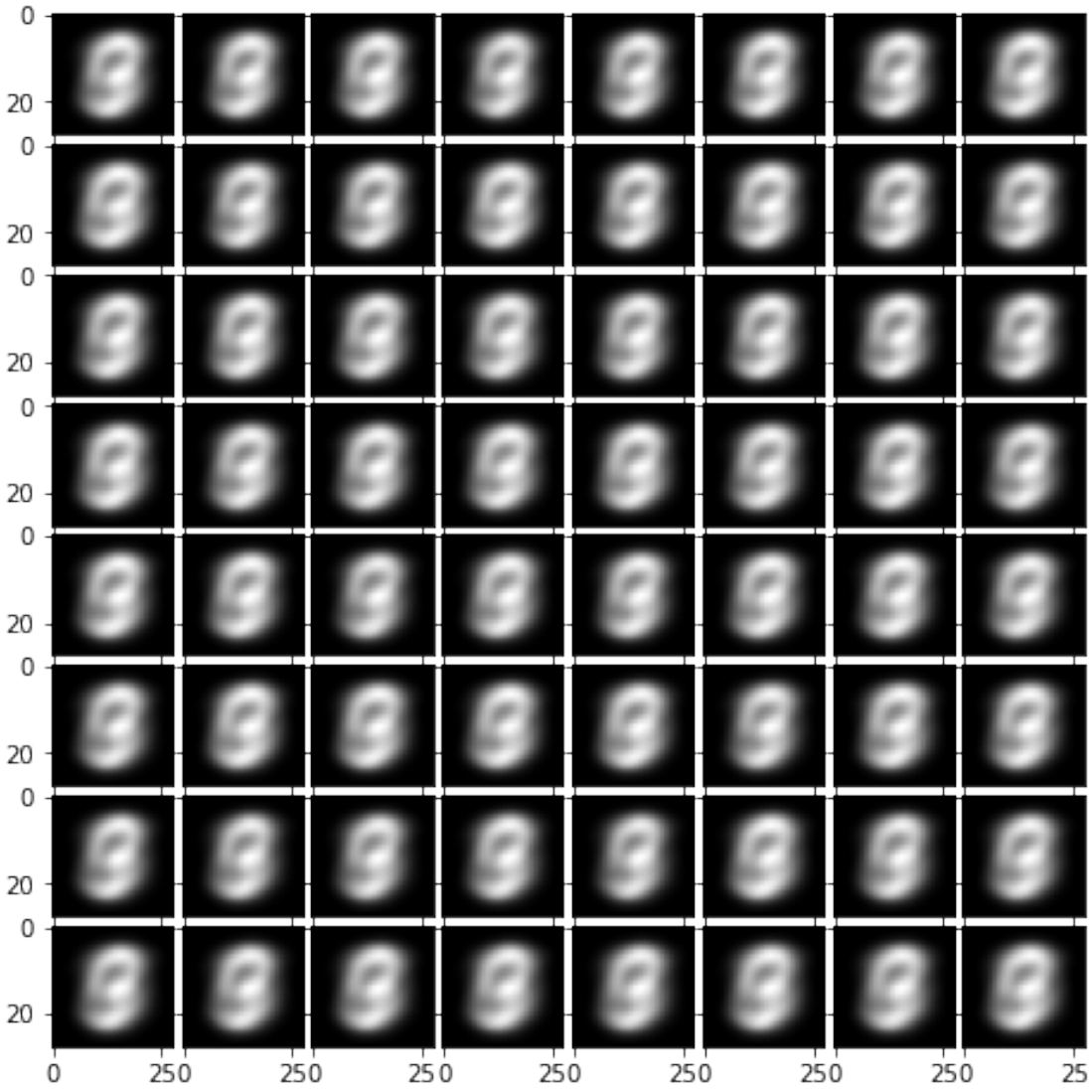

0
20
0
20
0
20
0
20
0
20
0
20
0
20
0
20
0 250 250 250 250 250 250 250 25


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
```

[ ]:
```
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>
```