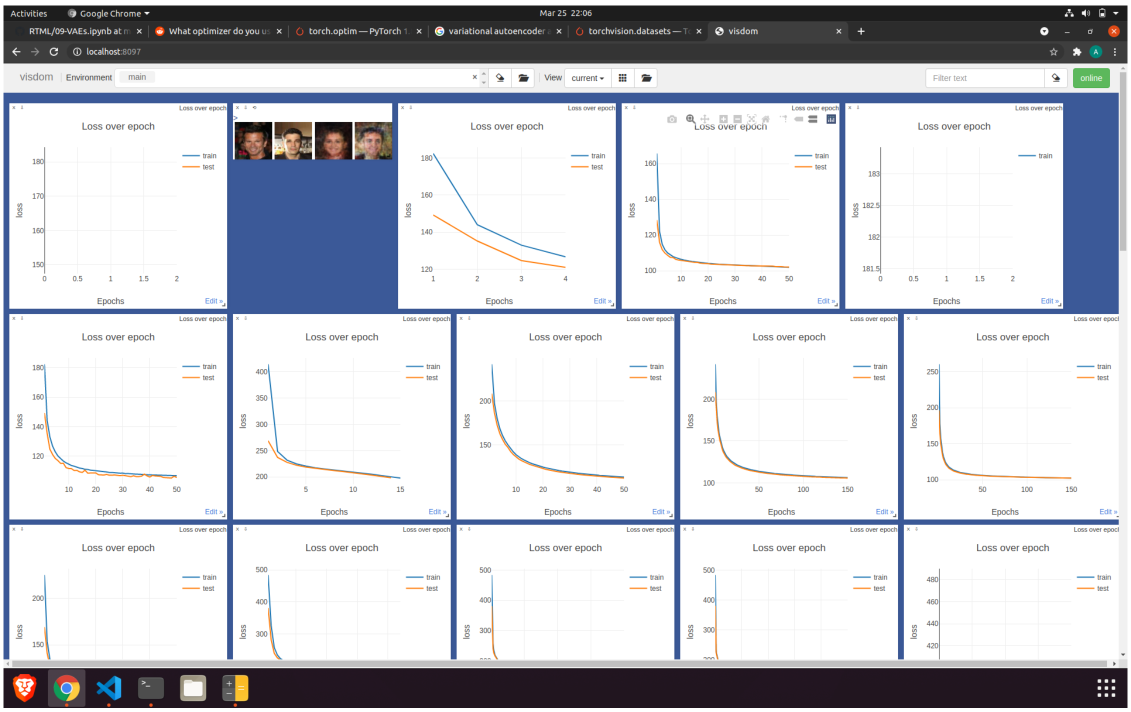
lab9 - report

March 25, 2021

First, here is a screenshots to prove that my visdom and python code is working.



1 Lab9 - st121413

1.1 1. VAE on MNIST

1.1.1 Experiment on Optimizer

I experiment on the Optimizer by changing it to simple SGD with 1-e3 learning rate but the VAE seems to dislike it because it prompt tons of error that can not understand. Therefore, I try to lower the learning rate more to 0.00001 and the training can started

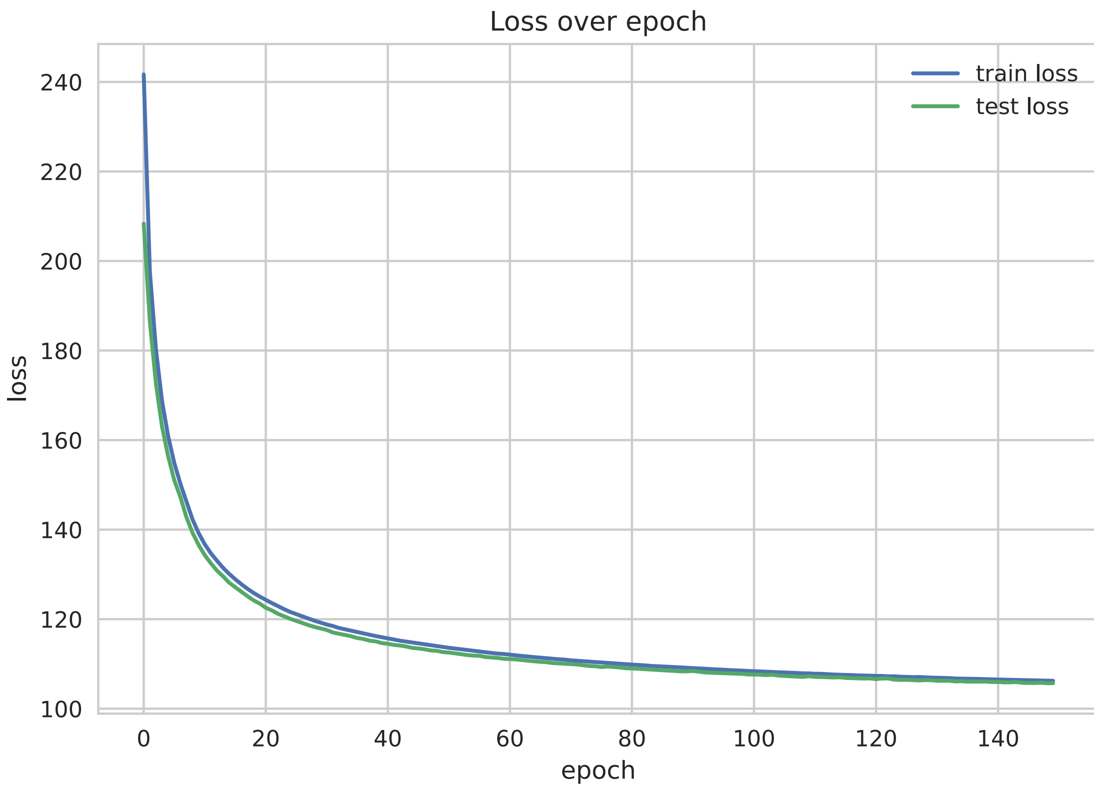
1. SGD

[ ]:
```
# optimizer = optim.SGD(model.parameters() , lr=0.00001)
```

[15]:
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
```

```
train_loss = [
241.6692,197.6623,180.2640,168.7409,160.8774,154.9138,150.2657,146.2110,142.
↪2606,139.2070,136.7275,134.7248,133.0525,131.4699,130.1271,128.9120,127.
↪8141,126.7948,125.8703,125.0563,124.3133,123.5895,122.8920,122.2317,121.
↪6181,121.1206,120.6180,120.1047,119.6469,119.2207,118.7935,118.4654,118.
↪0296,117.7202,117.3641,117.0923,116.7799,116.5009,116.2283,115.9694,115.
↪6658,115.4724,115.1954,114.9905,114.7907,114.5449,114.3486,114.1540,113.
↪9564,113.7474,113.6045,113.4330,113.2349,113.0573,112.9473,112.7594,112.
↪5878,112.4574,112.3068,112.2157,112.0577,111.9161,111.7719,111.6705,111.
↪5219,111.3930,111.2847,111.1681,111.0468,110.9618,110.8035,110.7047,110.
↪6241,110.5151,110.4019,110.3330,110.2350,110.1292,110.0374,109.9476,109.
↪8510,109.7813,109.6917,109.5687,109.5225,109.4301,109.3402,109.2963,109.
↪2187,109.1359,109.0492,108.9937,108.9292,108.8246,108.7832,108.6858,108.
↪6246,108.5820,108.5039,108.4434,108.3736,108.2836,108.2789,108.2068,108.
↪1420,108.1024,108.0189,107.9812,107.9022,107.8903,107.8033,107.7959,107.
↪7075,107.6322,107.5993,107.5349,107.5077,107.4420,107.4338,107.3789,107.
↪3198,107.3198,107.2189,107.2414,107.1394,107.1018,107.0572,107.0848,106.
↪9879,106.9443,106.9077,106.8396,106.8397,106.7400,106.7275,106.6971,106.
↪6788,106.6395,106.5835,106.5356,106.5160,106.4838,106.4663,106.3894,106.
↪3666,106.3392,106.3178,106.2862,106.2551,106.2333]

test_loss = [208.3556,186.5269,172.6224,163.0005,156.4162,151.0612,147.3203,142.
↪7491,139.2811,136.5707,134.3025,132.4728,130.8485,129.5700,128.1480,127.
↪1203,126.1207,125.1176,124.2125,123.4939,122.5394,121.9543,121.1945,120.
↪6296,120.0828,119.6035,119.0914,118.6723,118.2578,117.9082,117.5826,117.
↪0272,116.7472,116.4687,116.1818,115.7923,115.5663,115.2144,115.0357,114.
↪6624,114.4966,114.2532,114.1146,113.8916,113.5838,113.4604,113.2791,113.
↪0250,112.9336,112.6506,112.5503,112.3571,112.1486,111.9893,111.8691,111.
↪8126,111.5505,111.4464,111.3367,111.1569,111.0844,111.0138,110.8416,110.
↪7123,110.5797,110.4695,110.3866,110.2033,110.1279,110.0400,109.9853,109.
↪8822,109.7293,109.5457,109.4864,109.3150,109.4290,109.3427,109.2118,109.
↪0485,108.9613,108.9277,108.8481,108.7326,108.7306,108.6135,108.5248,108.
↪4399,108.3497,108.3548,108.4312,108.2661,108.0994,108.0626,107.9891,107.
↪9235,107.8814,107.8590,107.7937,107.6561,107.6427,107.6013,107.5043,107.
↪5956,107.3952,107.3352,107.2308,107.2222,107.1066,107.2454,107.1156,107.
↪0660,106.9998,106.9838,107.0135,106.8785,106.8134,106.7488,106.7342,106.
↪7716,106.6171,106.7417,106.7543,106.4948,106.4359,106.4597,106.3700,106.
↪3131,106.3843,106.3772,106.2306,106.2692,106.2588,106.1004,106.1479,106.
↪0466,106.0560,106.0278,106.0598,105.9590,105.9793,105.8857,105.8949,105.
↪9606,105.7790,105.7400,105.7458,105.8042,105.7067,105.6968]

plt.plot(train_loss, label="train loss")
plt.plot(test_loss, label="test loss")
plt.title("Loss over epoch")
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
```

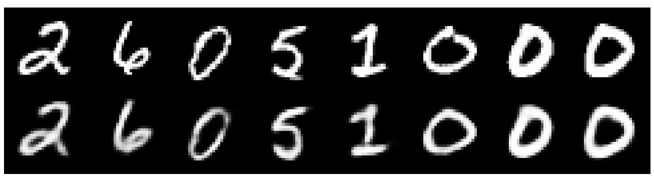
```
plt.show()
```



The reconstrction image at epoch 150

The sample image at epoch 150

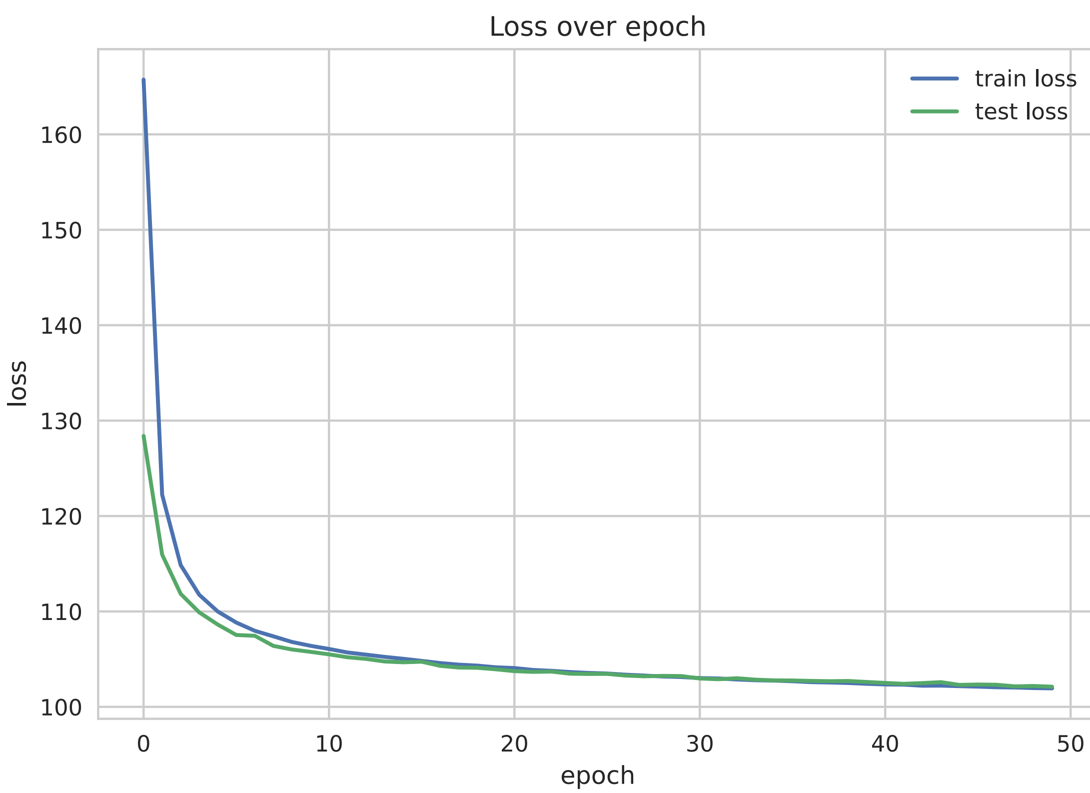
2. adam The reconstrction image at epoch 50



The sample image at epoch 50

[14]:

```
train_loss = [165.7384,122.2320,114.8598,111.7492,109.9953,108.8350,107.9703,107.3924,106.8022,106.4007,106.0683,105.7007,105.4731,105.2405,105.0402,104.8202,104.5905,104.4238,104.3238,104.1459,104.0693,103.8709,103.7769,103.6507,103.5552,103.4946,103.3701,103.2819,103.1801,103.1287,103.0157,102.9777,102.8690,102.7894,102.7525,102.6868,102.5920,102.5538,102.5005,102.4172,102.3497,102.3272,102.2171,102.2245,102.1698,102.1194,102.0547,102.0257,101.9690,101.9386]

test_loss = [128.3939,115.9546,111.8413,109.9077,108.6256,107.5334,107.4493,106.3886,106.0122,105.7681,105.5038,105.1988,105.0262,104.7575,104.6714,104.7331,104.2977,104.1203,104.0963,103.9370,103.7456,103.6680,103.7006,103.4767,103.4470,103.4569,103.2813,103.2026,103.2463,103.2269,102.9731,102.8965,103.0018,102.8504,102.7774,102.7654,102.7192,102.6903,102.7120,102.6097,102.5074,102.4142,102.4931,102.5930,102.3030,102.3414,102.3173,102.1466,102.1904,102.1141]

plt.plot(train_loss, label="train loss")
plt.plot(test_loss, label="test loss")
plt.title("Loss over epoch")
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```



Compare both optimizer, they have no noticable different.

From now, let stick with ADAM

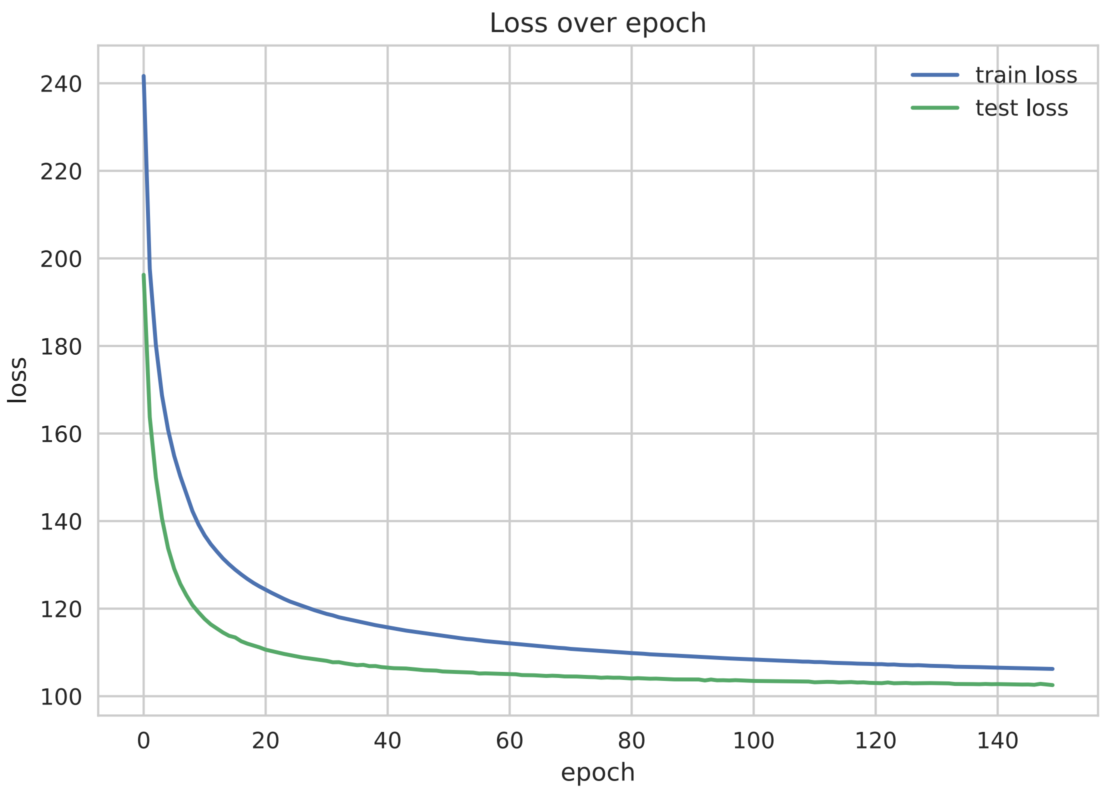
3. ADAM with less learning rate Here I use ADAM with learning rate of 0.0001 and see if it get any better

The reconstrction image at epoch 150

The sample image at epoch 150

[16]:

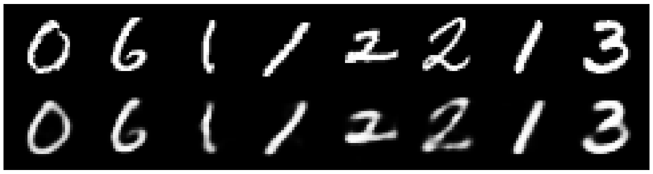
```
traiing_loss = [260.2390,177.6627,157.4811,146.0014,138.0080,132.4690,128.
↪3876,125.3330,122.9238,120.9596,119.3755,118.0202,116.9216,115.9662,115.
↪1456,114.3941,113.7619,113.1980,112.6609,112.1836,111.7669,111.3544,110.
↪9929,110.6233,110.2884,110.0235,109.7424,109.4822,109.2190,109.0095,108.
↪7865,108.5935,108.3913,108.1986,108.0376,107.8834,107.7132,107.6087,107.
↪4242,107.3104,107.1261,107.0453,106.8787,106.7874,106.7070,106.5712,106.
↪4744,106.3517,106.2887,106.1653,106.0867,106.0323,105.9252,105.8272,105.
↪7955,105.6964,105.6320,105.5516,105.4928,105.4186,105.3498,105.2924,105.
↪2507,105.1833,105.1290,105.0552,104.9912,104.9442,104.8969,104.8595,104.
↪7559,104.7178,104.6956,104.6199,104.5810,104.5727,104.5258,104.4710,104.
↪3945,104.3855,104.3072,104.2876,104.2273,104.2177,104.1761,104.1395,104.
↪0934,104.0323,104.0304,103.9988,103.9530,103.9316,103.8733,103.8399,103.
↪8317,103.7694,103.7385,103.7539,103.6859,103.6877,103.6153,103.5934,103.
↪5887,103.5514,103.5282,103.4798,103.4707,103.4721,103.3865,103.3960,103.
↪3608,103.3589,103.3034,103.2565,103.2546,103.1950,103.1933,103.1865,103.
↪1885,103.1458,103.0876,103.0982,103.0448,103.0550,102.9958,102.9919,102.
↪9933,102.9459,102.9448,102.9350,102.9090,102.8536,102.8445,102.8038,102.
↪8184,102.7638,102.7541,102.7732,102.7493,102.7298,102.6777,102.6630,102.
↪6510,102.6481,102.6074,102.5717,102.5502,102.5615,102.5639,102.5140]

test_loss = [196.2618,163.7477,149.9286,140.6152,133.8274,129.1168,125.6662,123.
↪0616,120.8106,119.1500,117.6406,116.4022,115.4655,114.5475,113.7945,113.
↪4280,112.5447,112.0039,111.5788,111.1504,110.6224,110.2895,109.9871,109.
↪6571,109.3719,109.1387,108.8457,108.6448,108.4858,108.2495,108.0750,107.
↪7525,107.7753,107.5018,107.3240,107.0889,107.1665,106.8724,106.8867,106.
↪6488,106.5272,106.3985,106.3761,106.3315,106.2032,106.0783,105.9397,105.
↪8828,105.8422,105.6463,105.5885,105.5436,105.4590,105.4043,105.3914,105.
↪1798,105.2176,105.1673,105.1398,105.0182,105.0045,105.0063,104.8257,104.
↪8298,104.7804,104.7175,104.6262,104.6809,104.6296,104.5143,104.5300,104.
↪4964,104.4169,104.3702,104.3379,104.2153,104.2808,104.2215,104.2297,104.
↪1440,104.0518,104.1252,104.0547,103.9955,104.0161,103.9520,103.8840,103.
↪8348,103.8259,103.8037,103.7571,103.8194,103.5956,103.8066,103.6289,103.
↪6473,103.5956,103.6595,103.5838,103.4680,103.4880,103.4699,103.4780,103.
↪4418,103.4373,103.4297,103.3613,103.3101,103.3154,103.3476,103.1764,103.
↪2461,103.2763,103.2602,103.1479,103.1805,103.2333,103.1251,103.1614,103.
↪0521,103.0015,102.9797,103.1299,102.9526,103.0086,103.0249,102.9445,102.
↪9733,103.0391,102.9957,102.9727,102.9137,102.9275,102.7889,102.7930,102.
↪8329,102.7627,102.7355,102.7921,102.7386,102.7655,102.7330,102.6662,102.
↪6463,102.6625,102.6752,102.6041,102.8225,102.6894,102.5369]

plt.plot(train_loss, label="train loss")
plt.plot(test_loss, label="test loss")
plt.title("Loss over epoch")
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```



Again, the graph might show some different but to the eye, there is no noticable different from reconstruction and sample image.

4. ADAM with larger batchsize The given first number of batch size is 128. I tried the 512 size and found that the learning is a bit more stable but takes a lot more time. So, I decide to just run size of 10000 and show the result here.

The reconstrction image at epoch 500



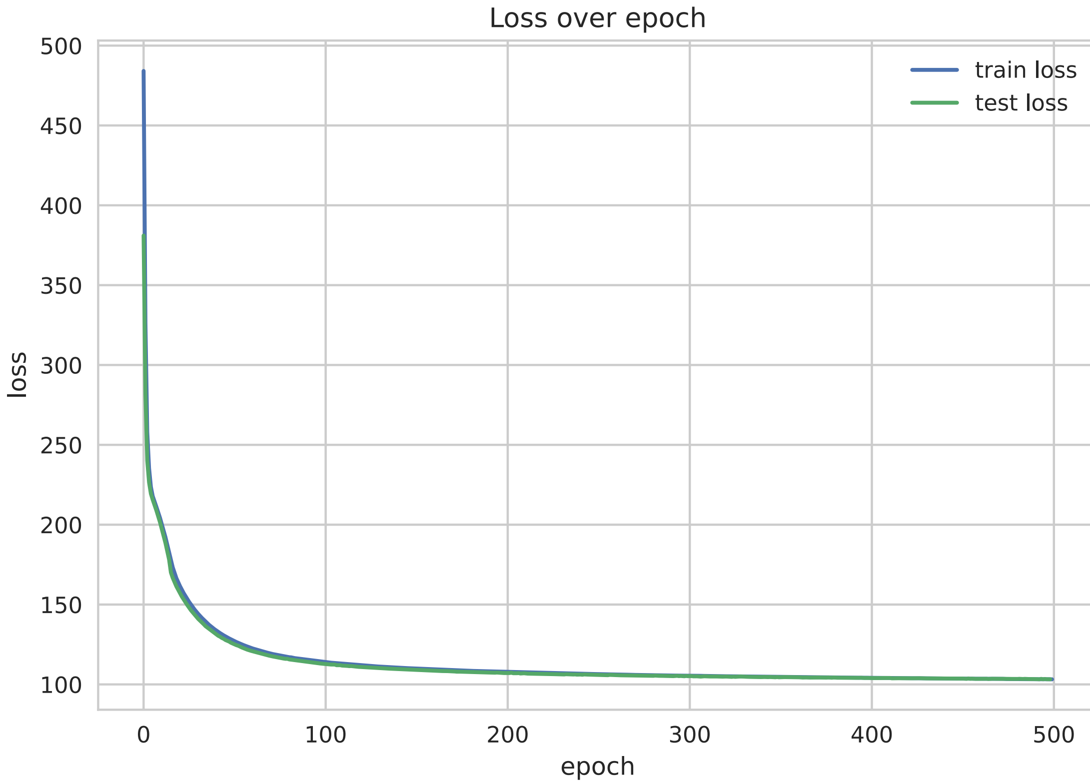
The sample image at epoch 500

```
[18]:
```

```
train_loss = [484.1251,326.3277,258.0663,235.3514,223.7051,217.8795,214.
↪4095,211.2266,207.9732,204.2786,200.3152,196.2618,192.1489,187.3353,182.
↪0585,177.4446,173.2794,169.6807,166.5235,163.8960,161.5048,159.2664,157.
↪1661,155.2518,153.3922,151.6439,150.0081,148.4227,146.8687,145.4351,144.
↪1288,142.8268,141.6441,140.4881,139.3997,138.2337,137.2386,136.2634,135.
↪4210,134.4995,133.6550,132.8465,132.1234,131.3575,130.6886,130.0157,129.
↪3834,128.7779,128.1844,127.6071,127.0768,126.5062,126.0596,125.5301,125.
↪0227,124.5571,124.0593,123.6918,123.2466,122.8865,122.4480,122.1069,121.
↪7571,121.3564,121.0728,120.7002,120.4128,120.1067,119.8339,119.6032,119.
↪2703,118.9848,118.7515,118.4819,118.2768,118.0227,117.8271,117.5981,117.
↪3958,117.2073,117.0116,116.9385,116.6387,116.4484,116.2990,116.0996,115.
↪9748,115.7287,115.5936,115.4251,115.2931,115.1053,114.9817,114.8371,114.
↪7241,114.5702,114.4610,114.3862,114.2215,114.0488,113.9071,113.8012,113.
↪6764,113.5764,113.4794,113.4148,113.2865,113.2296,113.0583,112.9248,112.
↪8133,112.6990,112.6167,112.5325,112.4209,112.2896,112.2830,112.1727,112.
↪0642,111.9705,111.8952,111.8045,111.7250,111.6781,111.5980,111.5178,111.
↪4558,111.3215,111.2629,111.1763,111.1848,111.0564,111.0116,110.9780,110.
↪8779,110.7862,110.7155,110.6629,110.6026,110.5205,110.4820,110.4318,110.
↪3102,110.2925,110.2449,110.1604,110.1087,110.1325,110.0050,109.9383,109.
↪9009,109.8039,109.7724,109.7644,109.6664,109.6535,109.5555,109.5361,109.
↪4384,109.4275,109.3442,109.2798,109.2906,109.2162,109.1704,109.0940,109.
↪0569,109.0452,109.0748,109.0307,108.9511,108.8834,108.8714,108.8305,108.
↪7101,108.6456,108.6087,108.5744,108.4985,108.4562,108.4476,108.4360,108.
↪4053,108.3598,108.3353,108.3162,108.2597,108.2370,108.1551,108.1690,108.
↪0740,108.0758,108.0173,107.9839,107.9380,107.9433,107.9103,107.8420,107.
↪7965,107.7867,107.7458,107.7346,107.6460,107.6836,107.7583,107.5804,107.
↪5489,107.5243,107.5517,107.4617,107.4197,107.4065,107.3729,107.2995,107.
↪2752,107.2802,107.2532,107.2346,107.1770,107.1281,107.1211,107.0967,107.
↪0839,107.0968,107.0711,106.9657,106.9815,106.9158,106.8674,106.8725,106.
↪8545,106.8139,106.7703,106.7597,106.7619,106.7447,106.7388,106.6961,106.
↪6958,106.6234,106.5861,106.6223,106.6096,106.5943,106.5440,106.4439,106.
↪4252,106.4972,106.5373,106.4395,106.4016,106.3805,106.3636,106.2938,106.
↪2997,106.2874,106.2166,106.2042,106.2378,106.2082,106.2066,106.1446,106.
↪1383,106.1617,106.0670,106.0784,106.0473,105.9869,106.0123,105.9814,105.
↪9282,105.9042,105.9089,105.8571,105.8897,105.8622,105.8374,105.8219,105.
↪8073,105.7539,105.7610,105.7519,105.7454,105.7393,105.6653,105.6940,105.
↪6352,105.6311,105.6238,105.6075,105.5634,105.5621,105.5219,105.5345,105.
↪5509,105.4864,105.4478,105.4425,105.4471,105.4528,105.4359,105.4560,105.
```

```
test_loss = [381.0698,281.8094,240.1786,226.4971,219.4196,215.6750,212.5502,209.
↪0035,205.3778,201.5719,197.2603,193.0351,188.5661,183.2154,178.1896,170.
↪0599,166.6968,163.8974,161.4006,159.1642,156.8700,154.9958,153.1722,151.
↪4939,149.6850,148.0142,146.4215,145.0320,143.5919,142.2788,141.0119,139.
↪9562,138.8311,137.6958,136.4908,135.6063,134.6323,133.6862,132.8384,132.
↪0499,131.3329,130.4438,129.8612,129.0778,128.5523,127.7815,127.2839,126.
↪8060,126.0766,125.5387,125.0730,124.5599,124.1979,123.5598,123.0992,122.
↪6040,122.2401,121.7322,121.4440,121.0181,120.7063,120.4356,120.1110,119.
↪6778,119.3847,119.0368,118.7629,118.5960,118.3424,117.9802,117.7968,117.
↪4840,117.2420,117.0412,116.7803,116.6737,116.4079,116.1644,115.9970,116.
↪0342,115.6405,115.4834,115.3531,115.0978,114.9500,114.7761,114.6444,114.
↪5317,114.3238,114.1784,113.9692,113.8672,113.8154,113.6789,113.4830,113.
↪3100,113.2257,113.0871,113.0176,112.9011,112.7452,112.6053,112.5074,112.
↪4302,112.4580,112.4210,112.0174,112.0316,112.0237,111.7793,111.7536,111.
↪7223,111.5681,111.4317,111.4877,111.3261,111.2810,111.0970,110.9899,110.
↪9318,110.8620,110.8655,110.7474,110.7052,110.5603,110.4866,110.4443,110.
↪3679,110.2961,110.2652,110.1955,110.1236,110.0191,109.9955,109.8629,109.
↪8481,109.7937,109.6892,109.6566,109.4925,109.5365,109.5114,109.3714,109.
↪3443,109.3405,109.3305,109.2955,109.1803,109.1947,109.1487,109.0288,109.
↪0417,108.9695,108.9615,108.8697,108.8401,108.8033,108.6734,108.6727,108.
↪5791,108.5514,108.5381,108.4908,108.4668,108.3561,108.3748,108.3595,108.
↪3008,108.2629,108.2504,108.2046,108.0789,107.9535,107.9987,107.9552,107.
↪9179,107.9501,107.8876,107.7878,107.8033,107.7596,107.6627,107.6620,107.
↪6945,107.7085,107.5850,107.5144,107.5828,107.4668,107.3946,107.4095,107.
↪4177,107.2592,107.3197,107.4142,107.3057,107.2515,107.1646,107.1114,107.
↪0897,107.0239,107.1181,107.3347,107.0019,106.9568,107.0024,107.0742,106.
↪8195,106.9150,106.9738,106.8264,106.7340,106.7192,106.7725,106.7251,106.
↪6711,106.6391,106.6506,106.5932,106.5653,106.5726,106.5740,106.5985,106.
↪4067,106.5126,106.4465,106.3897,106.3241,106.3534,106.3138,106.2554,106.
↪2310,106.3719,106.3137,106.2107,106.2230,106.1963,106.2490,106.1117,106.
↪1719,106.1702,106.0460,106.2279,106.1060,106.1174,106.0813,106.1053,106.
↪1084,105.9018,105.9569,105.9024,105.8273,105.8343,105.8754,105.8160,105.
↪7621,105.9612,105.8919,105.9352,105.7306,105.7497,105.7692,105.6896,105.
↪6693,105.7309,105.6003,105.6677,105.6613,105.6397,105.5747,105.5260,105.
↪4625,105.5452,105.5545,105.4718,105.4422,105.5004,105.3623,105.4165,105.
↪4061,105.3722,105.5112,105.4105,105.3659,105.4186,105.3407,105.2960,105.
↪2756,105.2546,105.2428,105.1997,105.1748,105.2799,105.3685,105.1956,105.
↪2798,105.1913,105.1012,105.2921,105.1402,105.1928,105.1871,105.0405,105.
```

```
plt.plot(train_loss, label="train loss")
plt.plot(test_loss, label="test loss")
plt.title("Loss over epoch")
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```



I come to the conclusion that the original given ADAM configuration is the best one.

I furthur did some research online to find which optimizer do people use with VAE and thry just use ADAM with default parameter.

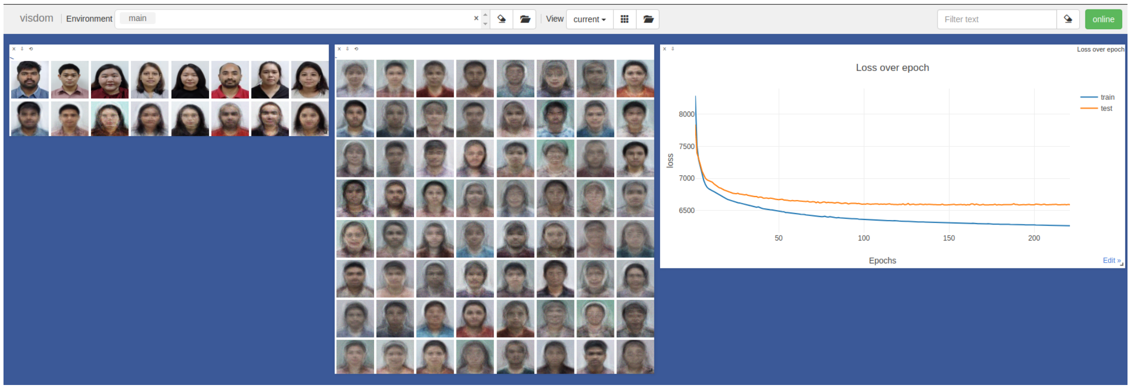
1.2 2. VAE on ICT

As a baseline, I ran the FC-VAE on ICT.

All photo are resized to 64 x 64 and convert to Tensor.

I modified the layer a bit so it is compatible with the new 3 x 64 x 64 data.

The result is as follow.



Now I modified the encoder to be AlexNet like and decoder as generator from DCGAN.

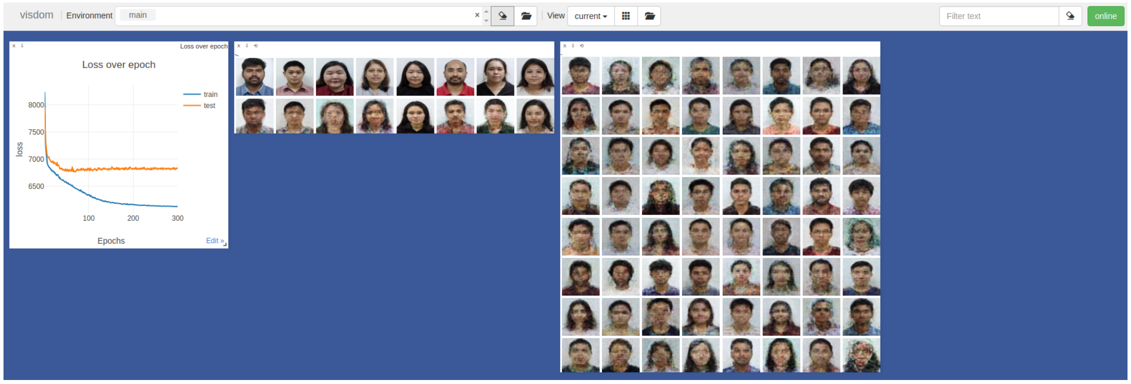
The network is as follow.

[ ]:
```
class VAEConv2(nn.Module):
    def __init__(self):
        super(VAEConv2, self).__init__()

        # for encoder
        self.fc1 = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(96, 256, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True)
            )
        self.fc21 = nn.Linear(4608//2, 20)
        self.fc22 = nn.Linear(4608//2, 20)

        # for decoder
        self.decoder_linear = torch.nn.Linear(20, 1024*4*4)

        self.decoder_conv1 = nn.Sequential(
            nn.ConvTranspose2d(
                in_channels=1024, out_channels=512, kernel_size=4,
                stride=2, padding=1, bias=False
            ),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True)
```

```
        )
        self.decoder_conv2 = nn.Sequential(
            nn.ConvTranspose2d(
                in_channels=512, out_channels=256, kernel_size=4,
                stride=2, padding=1, bias=False
            ),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
        )
        self.decoder_conv3 = nn.Sequential(
            nn.ConvTranspose2d(
                in_channels=256, out_channels=128, kernel_size=4,
                stride=2, padding=1, bias=False
            ),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )
        self.decoder_conv4 = nn.Sequential(
            nn.ConvTranspose2d(
                in_channels=128, out_channels=3, kernel_size=4,
                stride=2, padding=1, bias=False
            )
        )
        self.decoder_out = torch.nn.Sigmoid()

    def encode(self, x):
        x = self.fc1(x)
        h1 = F.relu(x).view(-1,4608//2)
        return self.fc21(h1), self.fc22(h1)

    def reparameterize(self, mu, logvar):
        # 0.5 for square root (variance to standard deviation)
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std

    def decode(self,z):
        # Project and reshape
        x = self.decoder_linear(z)
        x = x.view(-1, 1024, 4, 4)
        x = self.decoder_conv1(x)
        x = self.decoder_conv2(x)
        x = self.decoder_conv3(x)
        x = self.decoder_conv4(x)
        x = self.decoder_out(x)
        return x
```

```python
    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
```

Here is the result.



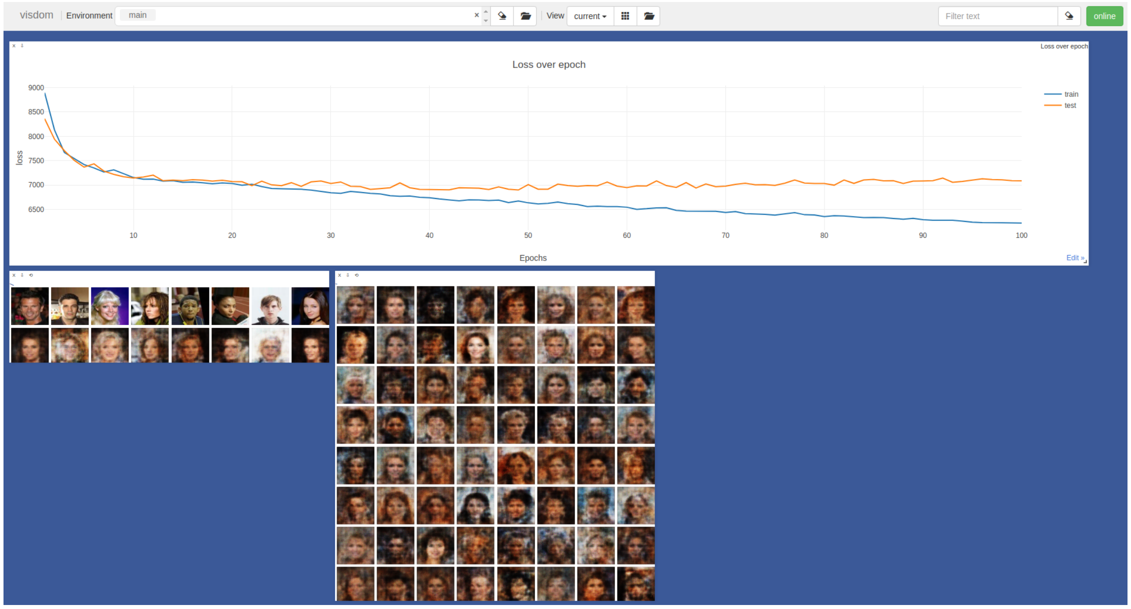
The result is as is regardless of how much I try.

- Chaning the hidden dimension (when it is sufficient, it is the same just need longer training time)
- Chaning the encoder or decoder.

Therefore, I ommit thoes result from here.

1.3 3. VAE on CelebA-317

Here is the result.



I tried with 300 epochs, it worse