In [51]:
```
!pip install time
!pip install tqdm
```

```
ERROR: Could not find a version that satisfies the requirement time
(from versions: none)
ERROR: No matching distribution found for time
Collecting tqdm
  Downloading tqdm-4.64.0-py2.py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.4/78.4 kB 795.0 kB/
s eta 0:00:0031m1.5 MB/s eta 0:00:01m
Installing collected packages: tqdm
Successfully installed tqdm-4.64.0
```

In [1]:
```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

```
2022-07-13 08:09:56.547867: W tensorflow/stream_executor/platform/d
efault/dso_loader.cc:64] Could not load dynamic library 'libcudart.
so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object fil
e: No such file or directory
2022-07-13 08:09:56.547890: I tensorflow/stream_executor/cuda/cudar
t_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU
set up on your machine.

TensorFlow version: 2.9.1
```

In [54]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
from tqdm import tqdm
```

In [410]:
```
data = pd.read_csv('data/handtyped_nn/train.csv')
label = data['label']
```

In [411]:
```
data /= 254
data['label'] = label
```

In [412]:
```
data.head()
```

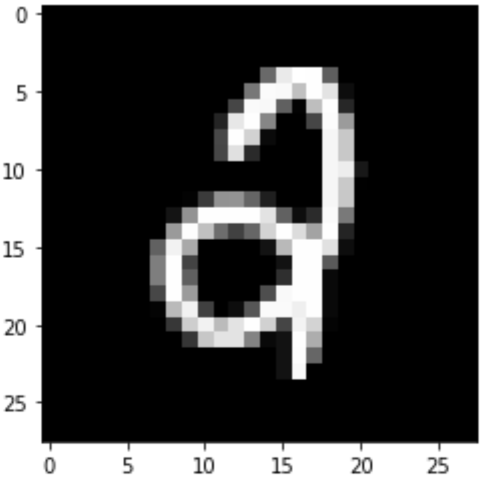
Out[412]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixe
0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
1	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
2	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
3	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
4	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	

5 rows × 785 columns

In [413]:
```
plt.imshow(data.values[10000][1:].reshape(28, 28), cmap='gray')
```

Out[413]:
```
<matplotlib.image.AxesImage at 0x7fb71ea7ff40>
```



In [414]:

```
data.values[10000][1:].reshape(28, 28)
```

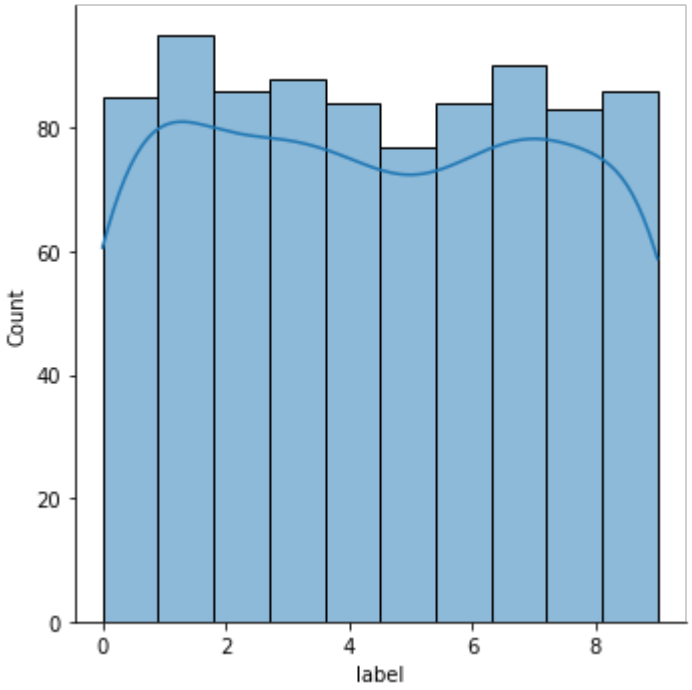
Out[414]:

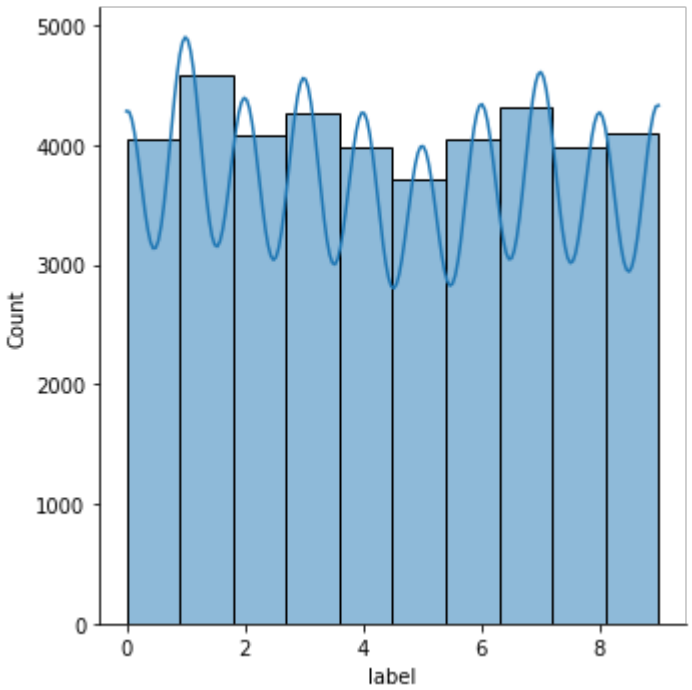
```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
```

In [415]:

```
def split_data_1to50(data):
    test = data.sort_values(by=['label'])[::49]
    train = pd.concat([data, test]).drop_duplicates(keep=False)
    sns.displot(test['label'], bins=10, kde=True);
    sns.displot(train['label'], bins=10, kde=True);
    return test, train
```

In [417]:
```
1 test, train = split_data_1to50(data)
```





In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [425]:
```
1 class MyNN:
2     def __init__(self):
3         self.learn_rate = 0.01
4 
5         self._weights0 = np.random.rand(10, 16)-0.5
6         self._weights1 = np.random.rand(16, 49)-0.5
7         self._weights2 = np.random.rand(49, 784)-0.5
```

```

        self._input  = np.zeros([10])
        self._layer1 = np.zeros([16])
        self._layer2 = np.zeros([49])
        self._output = np.zeros([784])

        self._correction_0 = np.zeros([10, 16])
        self._correction_1 = np.zeros([16, 49])
        self._correction_2 = np.zeros([49, 784])

        self._activation0 = self.act_relu
        self._activation1 = self.act_relu
        self._activation2 = self.act_relu

        self._der_act_0 = self.der_act_relu
        self._der_act_1 = self.der_act_relu
        self._der_act_2 = self.der_act_relu

    def zero(self):
        self._input  = np.zeros([10])
        self._layer1 = np.zeros([16])
        self._layer2 = np.zeros([49])
        self._output = np.zeros([784])

    def act_sigmoid(self, layer):
        return np.array([1/(1+np.exp(-x)) for x in layer])

    def act_relu(self, layer):
        return np.array([x if x > 0 else 0 for x in layer])

    def act_softplus(self, layer):
        return np.array([np.ln(1+np.exp(x)) for x in layer])

    def der_act_sigmoid(self, l):
        _exp = np.exp(-l)
        return _exp / (1+_exp)**2

    def der_act_relu(self, l):
        if l > 0:
            return 1
        else:
            return 0

    def def_act_softplus(self, l):
        return 1 / (1+np.exp(-l))

    def predict(self, num):
        self.zero()
        self._input = np.array([100 if k == num[0] else 0 for k i
        #print(num, self._input)

        self._layer1 = np.dot(self._input, self._weights0)
        self._layer1 = self._activation0(self._layer1)
        #print("   ", self._layer1)


        self._layer2 = np.dot(self._layer1, self._weights1)
        self._layer2 = self._activation1(self._layer2)

        self._output = np.dot(self._layer2, self._weights2)
```

```
68          self._output = self._activation2(self._output)
69
70          return self._output
71
72      def learn(self, labels):
73
74          main_delta   = self._output - labels
75          delta_output = main_delta * [self._der_act_2(x) for x in
76          delta_w_2    = np.dot(delta_output.reshape(len(delta_outp
77                                  self._layer2.reshape(1, len(self._l
78
79          delta_layer2 = np.dot(self._weights2, delta_output) * [se
80          delta_w_1    = np.dot(delta_layer2.reshape(len(delta_laye
81                                  self._layer1.reshape(1, len(self._l
82
83          delta_layer1 = np.dot(self._weights1, delta_layer2) * [se
84          delta_w_0    = np.dot(delta_layer1.reshape(len(delta_laye
85                                  self._input.reshape(1, len(self._in
86
87          self._correction_0 += np.transpose(delta_w_0)
88          self._correction_1 += np.transpose(delta_w_1)
89          self._correction_2 += np.transpose(delta_w_2)
90
91      def update(self):
92          self._weights0 = self._weights0 - self.learn_rate * self.
93          self._weights1 = self._weights1 - self.learn_rate * self.
94          self._weights2 = self._weights2 - self.learn_rate * self.
95
96          self._correction_0 = np.zeros([10, 16])
97          self._correction_1 = np.zeros([16, 49])
98          self._correction_2 = np.zeros([49, 784])
99
```

In [426]:

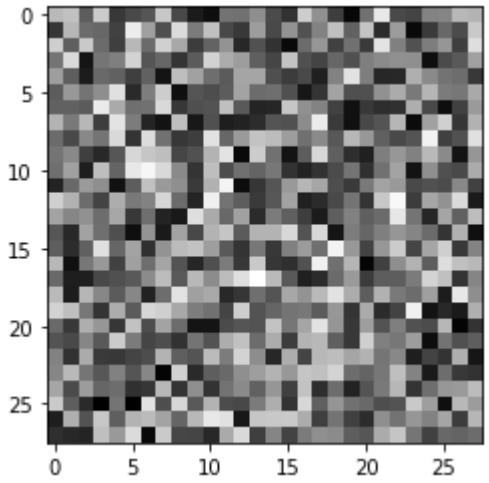
```
1  NN = MyNN()
2
3  NN._activation0 = NN.act_sigmoid
4  NN._activation1 = NN.act_sigmoid
5  NN._activation2 = NN.act_sigmoid
6
7  NN._der_act_0 = NN.der_act_sigmoid
8  NN._der_act_1 = NN.der_act_sigmoid
9  NN._der_act_2 = NN.der_act_sigmoid
10
```

In [ ]:

```
1
```

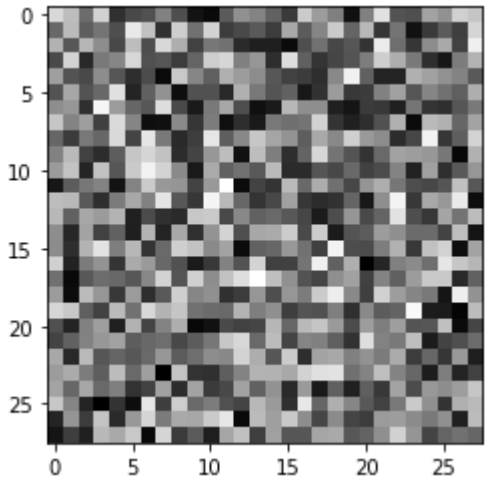
In [427]:
```
res = NN.predict([0])
plt.imshow(res.reshape(28, 28), cmap='gray')
```

Out[427]: <matplotlib.image.AxesImage at 0x7fb71fa89190>



In [428]:
```
res = NN.predict([1])
plt.imshow(res.reshape(28, 28), cmap='gray')
```

Out[428]: <matplotlib.image.AxesImage at 0x7fb71f0f4250>



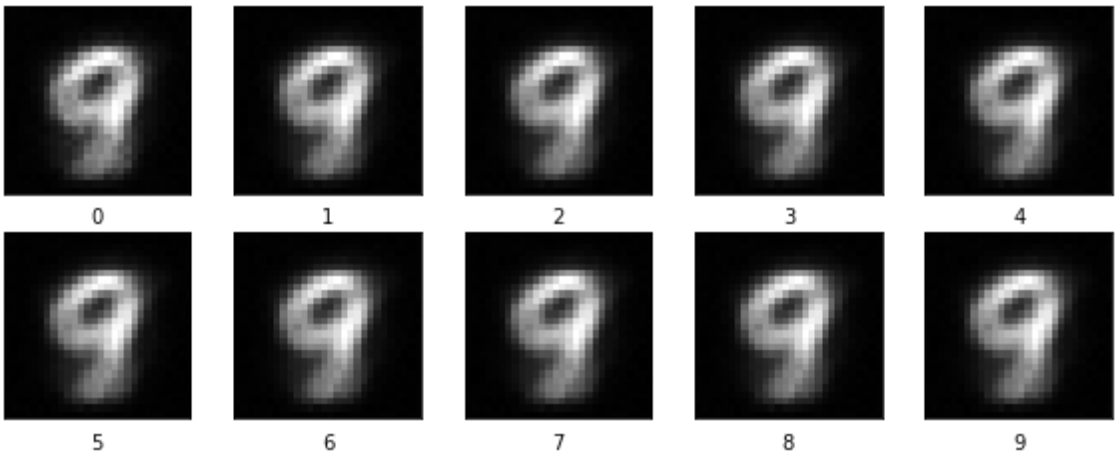
In [429]:
```
def print_predictions(nn):
    plt.figure(figsize=(10,10))
    for i in range(10):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        res = nn.predict([i])
        plt.imshow(res.reshape(28, 28), cmap='gray')
        plt.xlabel(i)
    plt.show()

```

In [430]:
```
#data = data.sort_values(by=['label'])

```

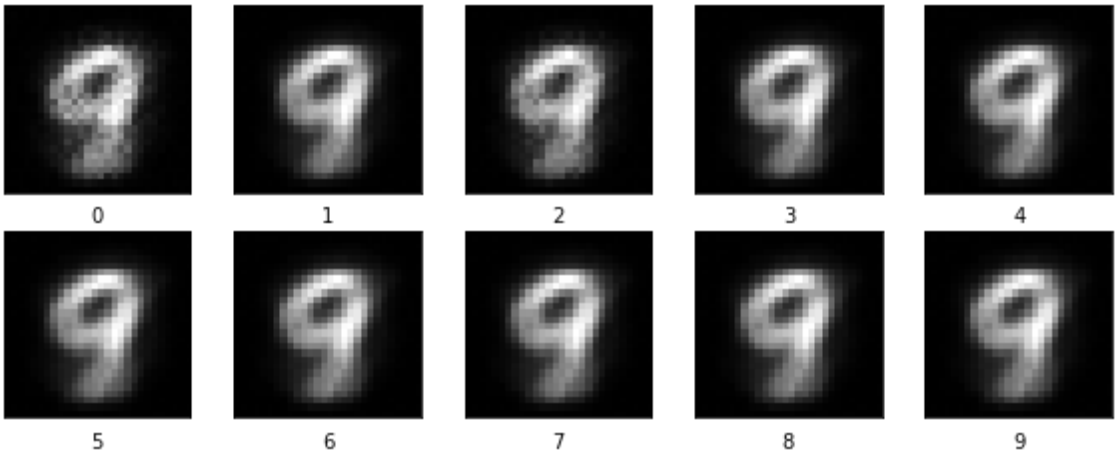
In [431]:
```
epohs = 100
uprate = 5
```

```
print_flag = 5000
_data = test
for e in range(epohs):
    print('E:', e)
    for i in tqdm(range(len(_data))):
        NN.predict( _data.values[i][:1] )
        expect = _data.values[i][1:]
        NN.learn(expect)
        if not i % uprate:
            NN.update()
        #if not i % print_flag:
        #    print_predictions(NN)
    print_predictions(NN)
```

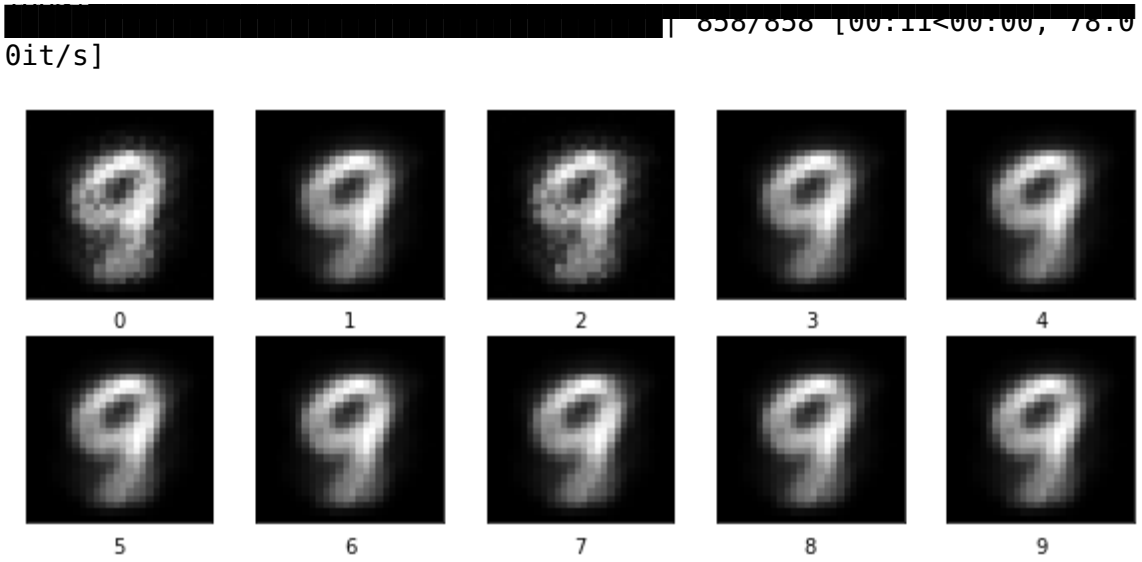
```
E: 0

  4%|
| 32/858 [00:00<00:16, 49.42it/s]/tmp/ipykernel_5437/3794245865.py:
33: RuntimeWarning: overflow encountered in exp
  return np.array([1/(1+np.exp(-x)) for x in layer])
100%|
| 858/858 [00:11<00:00, 74.2
7it/s]
```



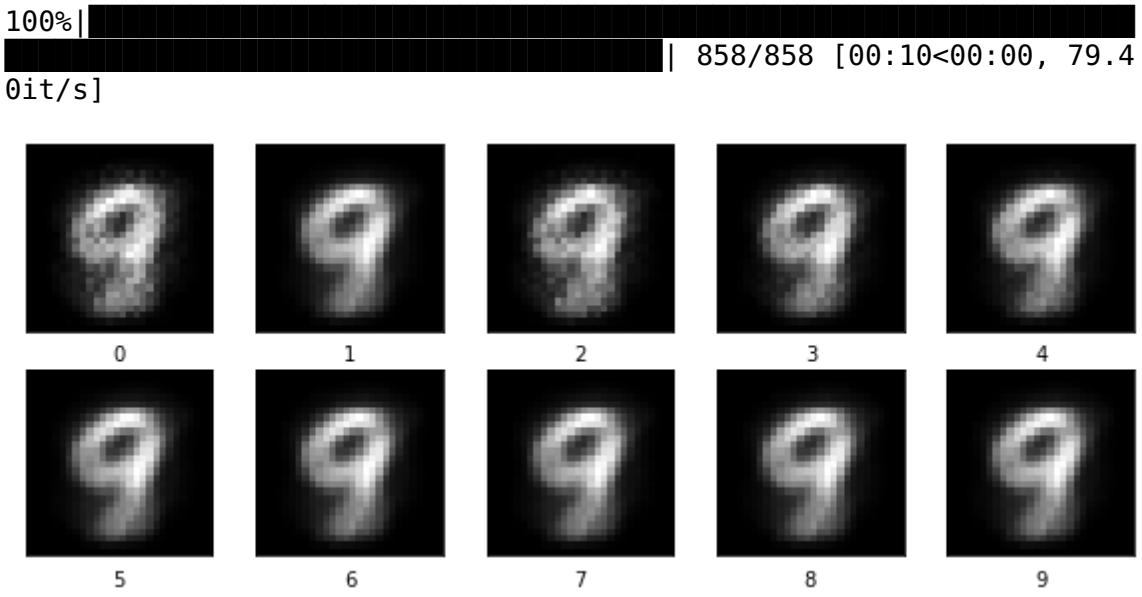

```
E: 1

100%|
| 858/858 [00:10<00:00, 78.8
2it/s]
```



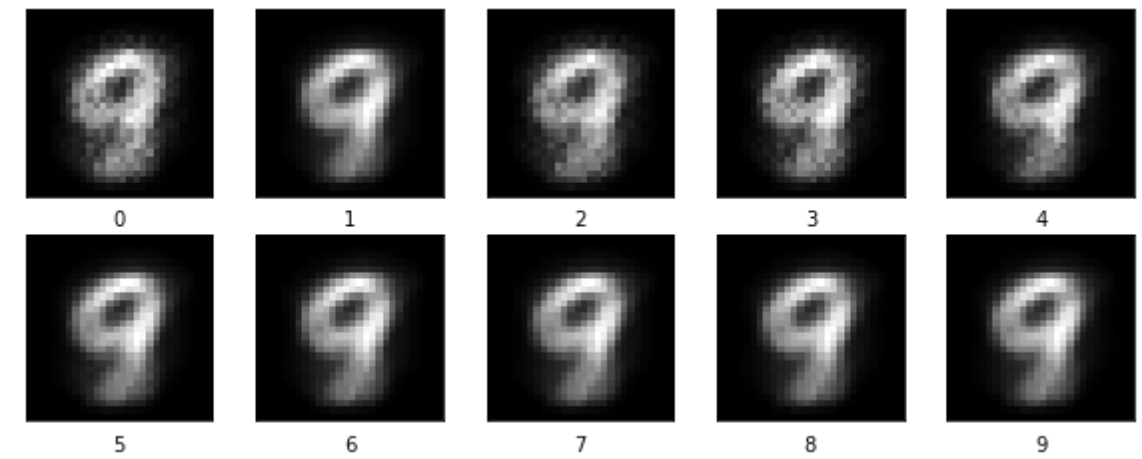

```
E: 2

100%|
```



E: 3

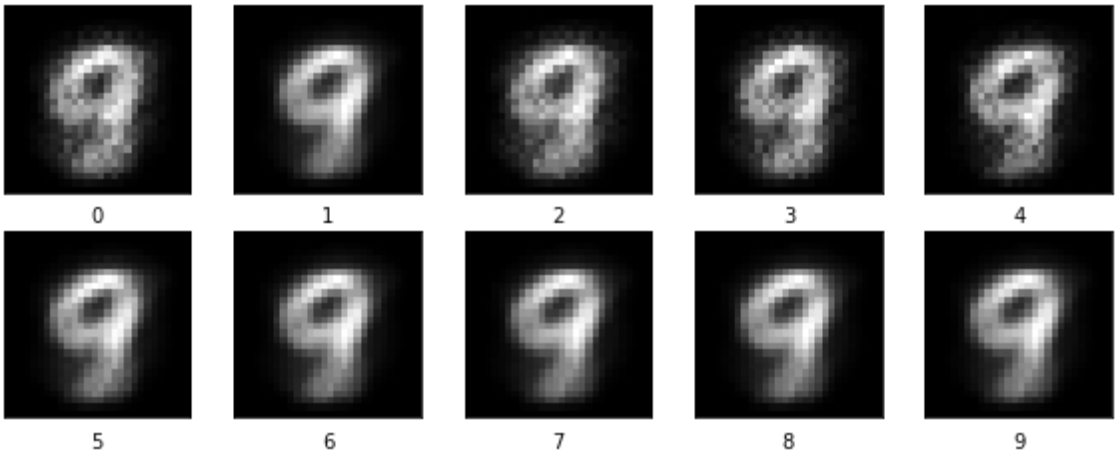


E: 4



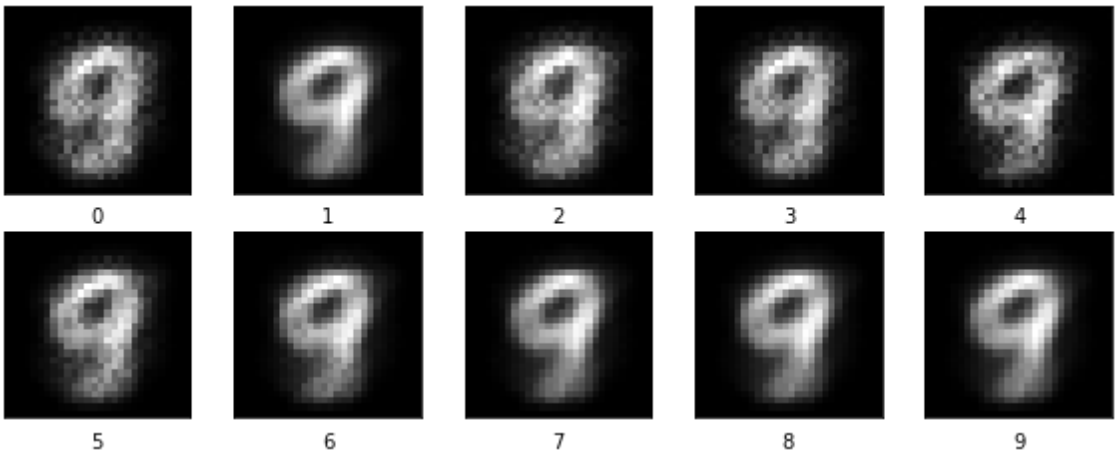
E: 5

| 858/858 [00:10<00:00, 80.31it/s]



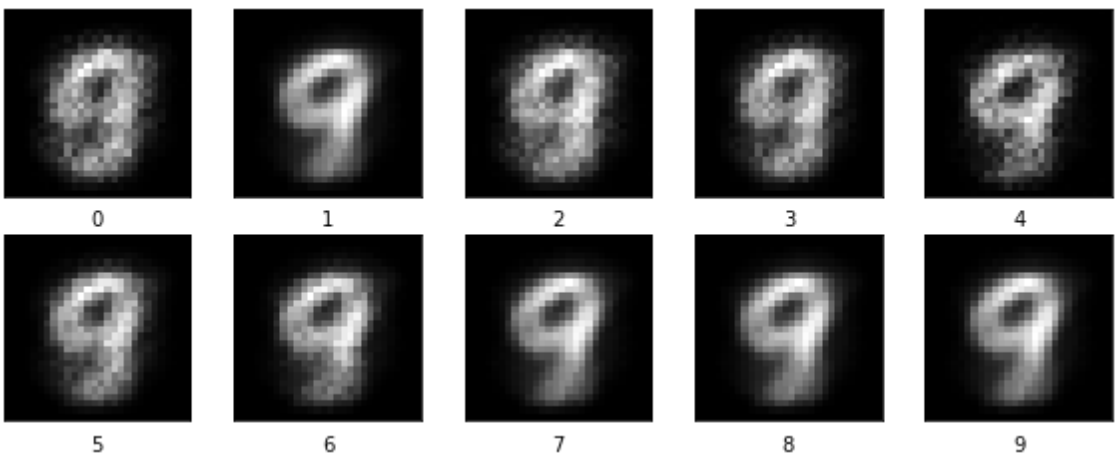
E: 6

100%| | 858/858 [00:10<00:00, 82.13it/s]



E: 7

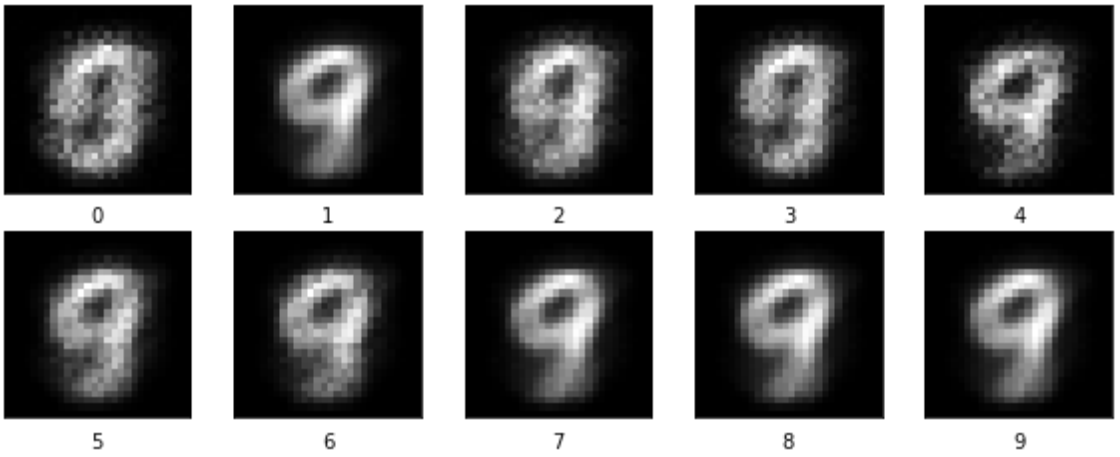
100%| | 858/858 [00:10<00:00, 81.09it/s]



E: 8

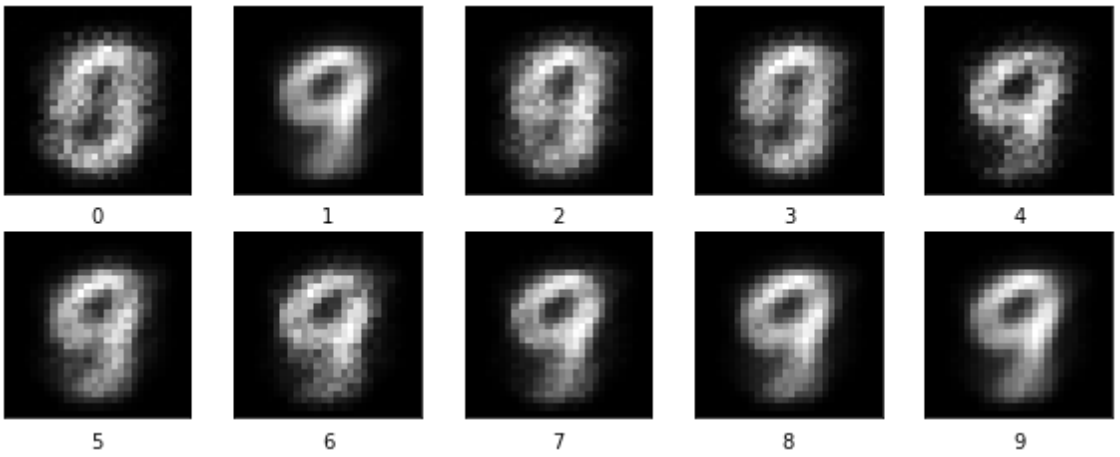
100%|

| 858/858 [00:10<00:00, 81.79it/s]



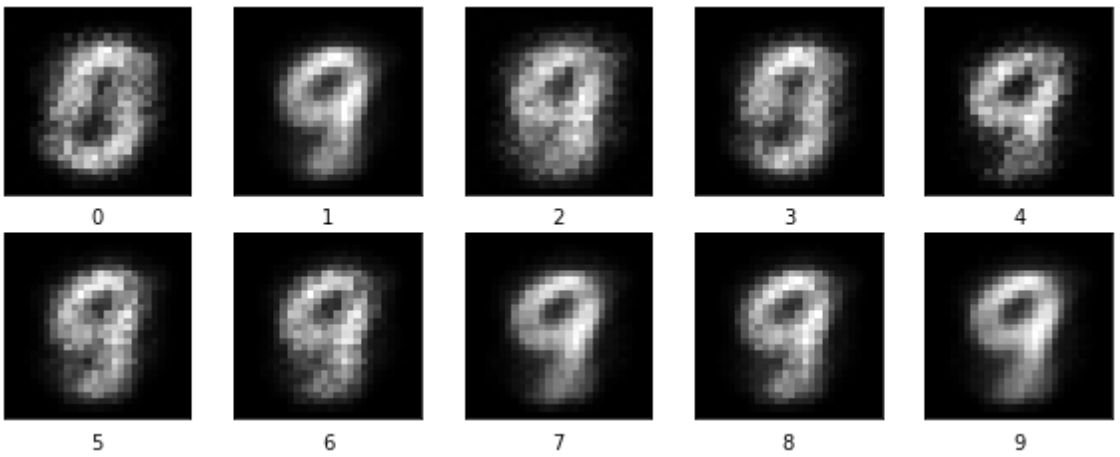
E: 9

100%| | 858/858 [00:11<00:00, 75.43it/s]



E: 10

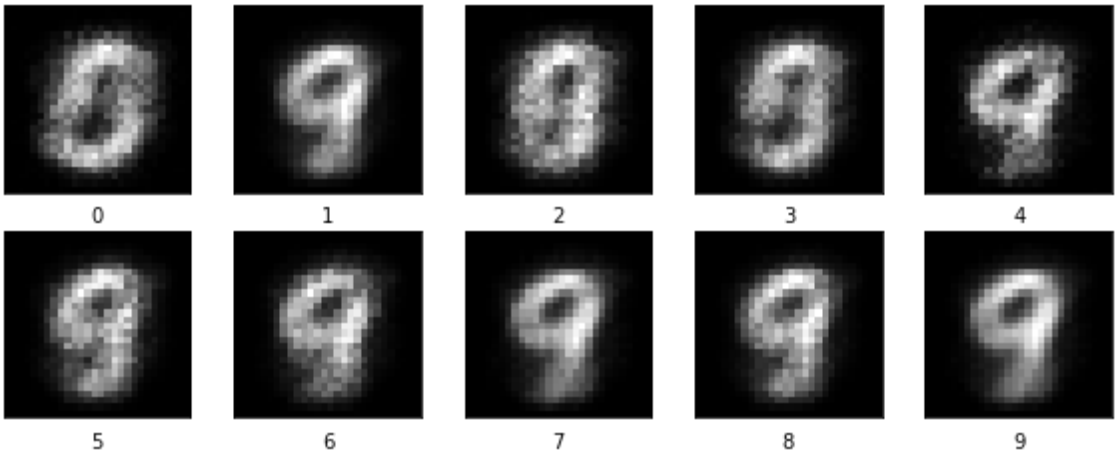
100%| | 858/858 [00:10<00:00, 82.71it/s]



E: 11

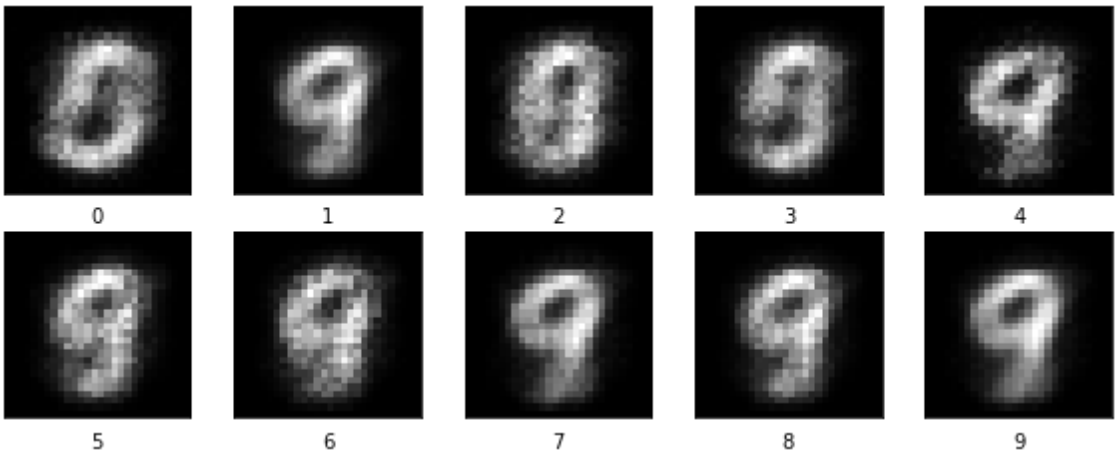
100%|

| 858/858 [00:10<00:00, 82.3
1it/s]



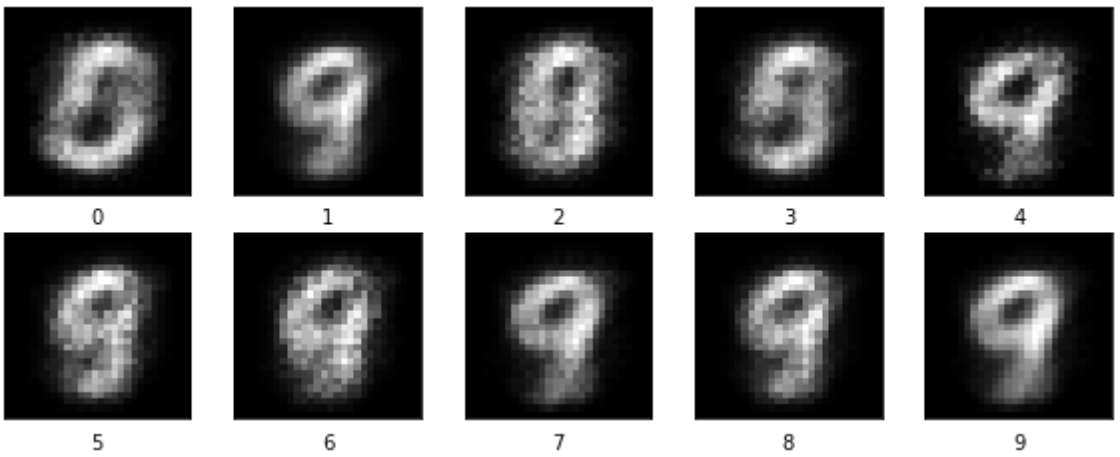
E: 12

100%|
| 858/858 [00:10<00:00, 80.5
5it/s]



E: 13

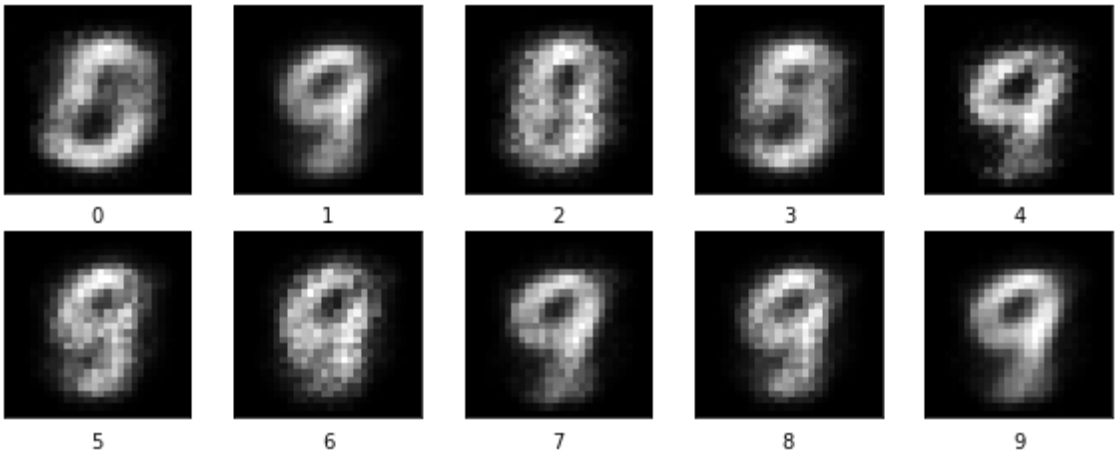
100%|
| 858/858 [00:10<00:00, 81.6
4it/s]



E: 14

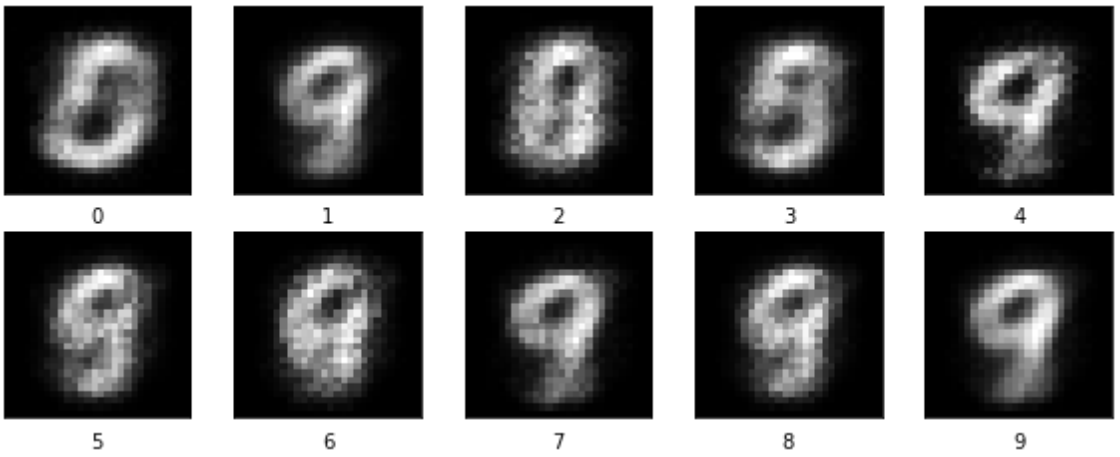
100%|

| 858/858 [00:10<00:00, 82.03it/s]



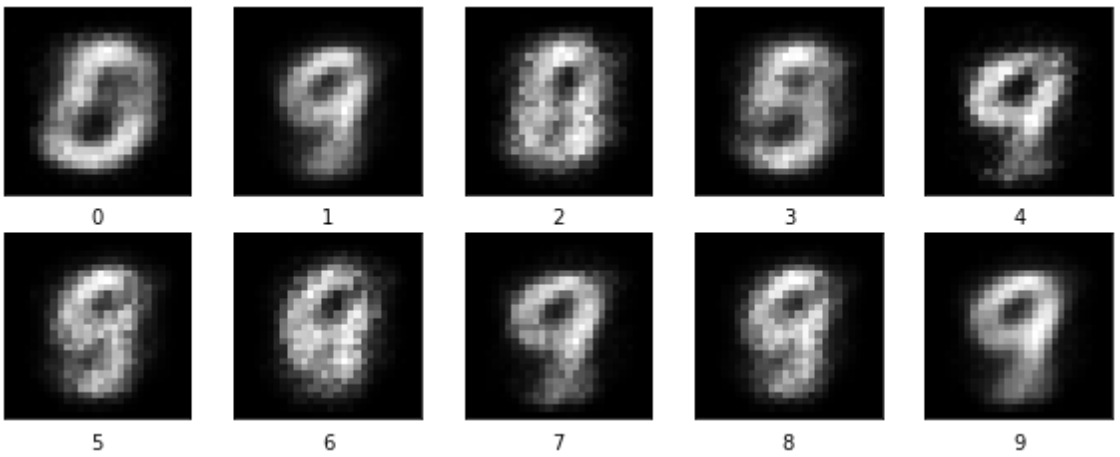
E: 15

100%| | 858/858 [00:10<00:00, 80.77it/s]



E: 16

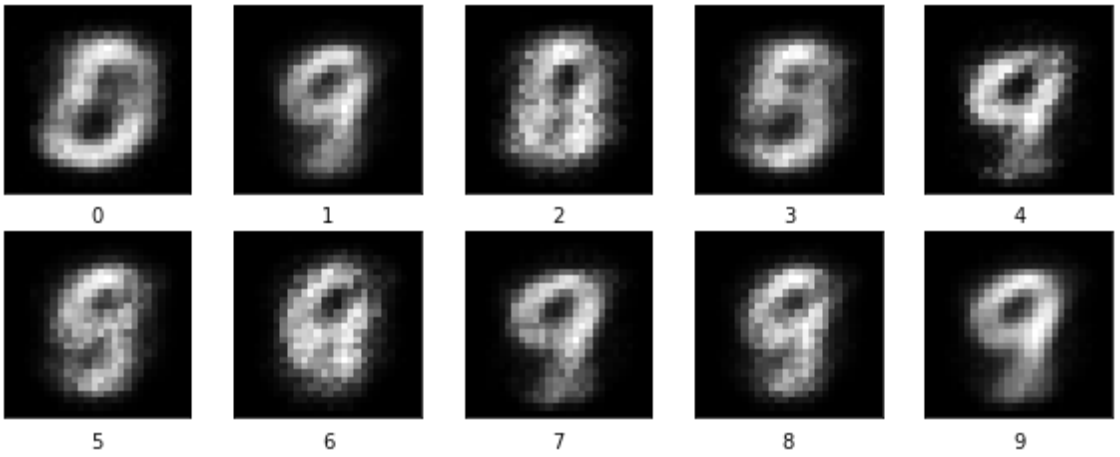
100%| | 858/858 [00:10<00:00, 81.15it/s]



E: 17

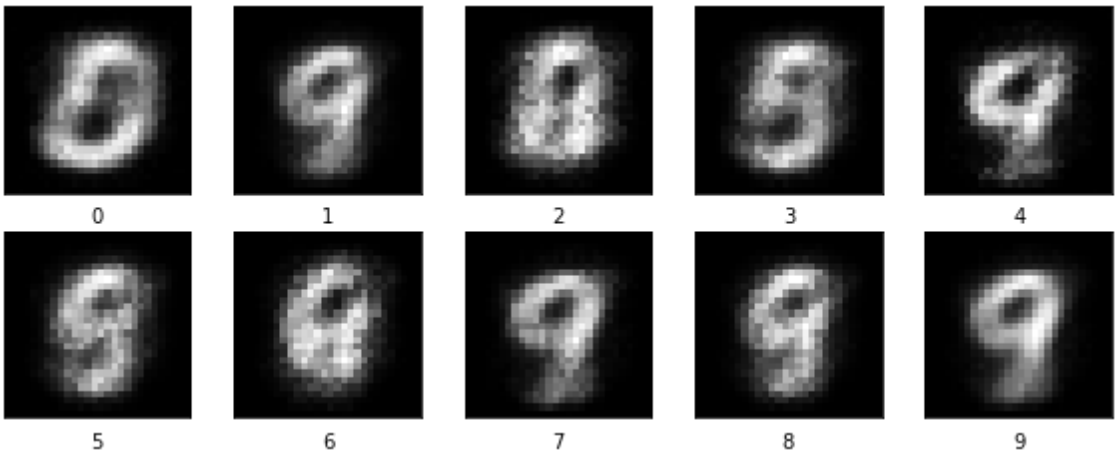
100%|

| 858/858 [00:10<00:00, 81.09it/s]



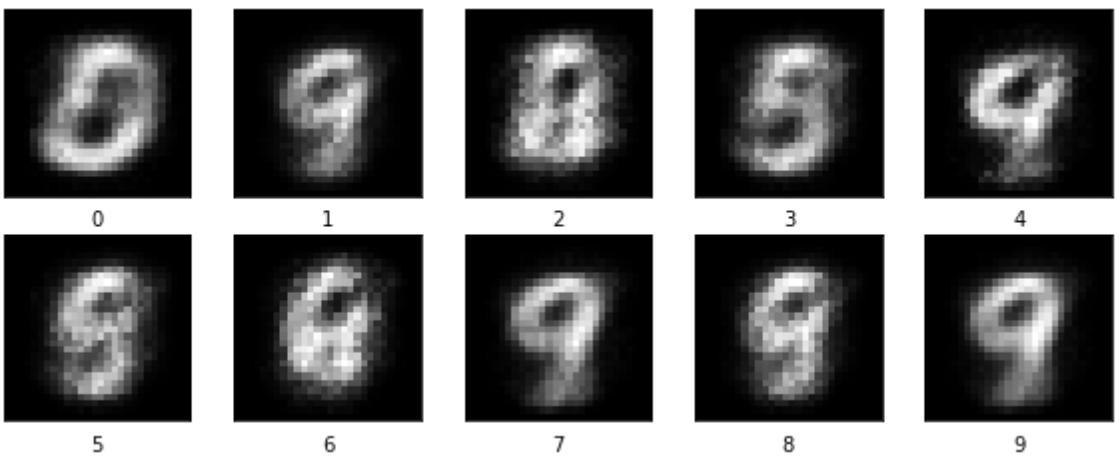
E: 18

100%| | 858/858 [00:10<00:00, 79.07it/s]



E: 19

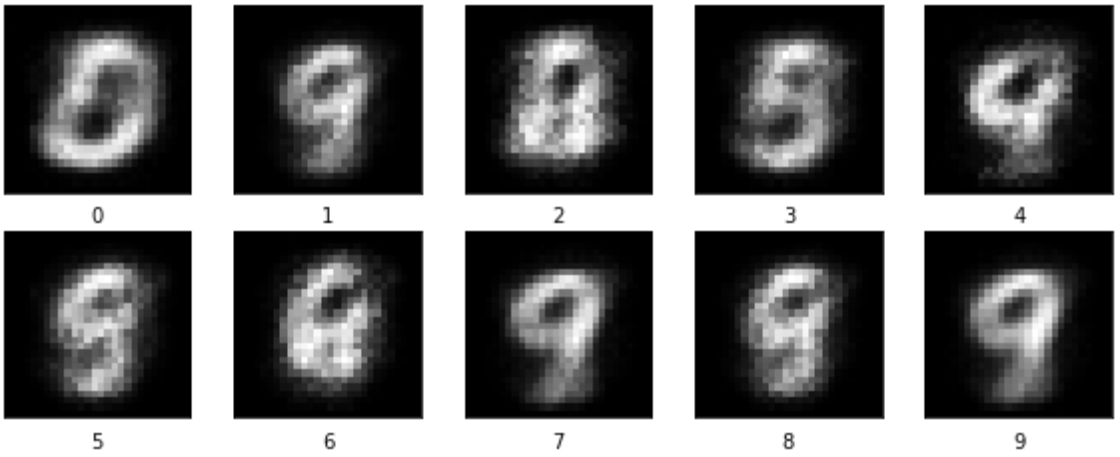
100%| | 858/858 [00:10<00:00, 80.38it/s]



E: 20

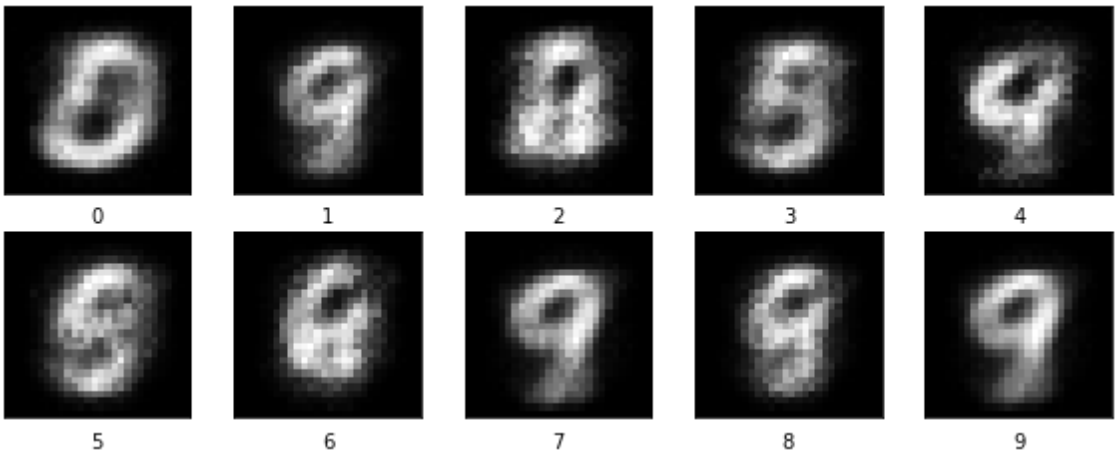
100%|

| 858/858 [00:10<00:00, 78.39it/s]



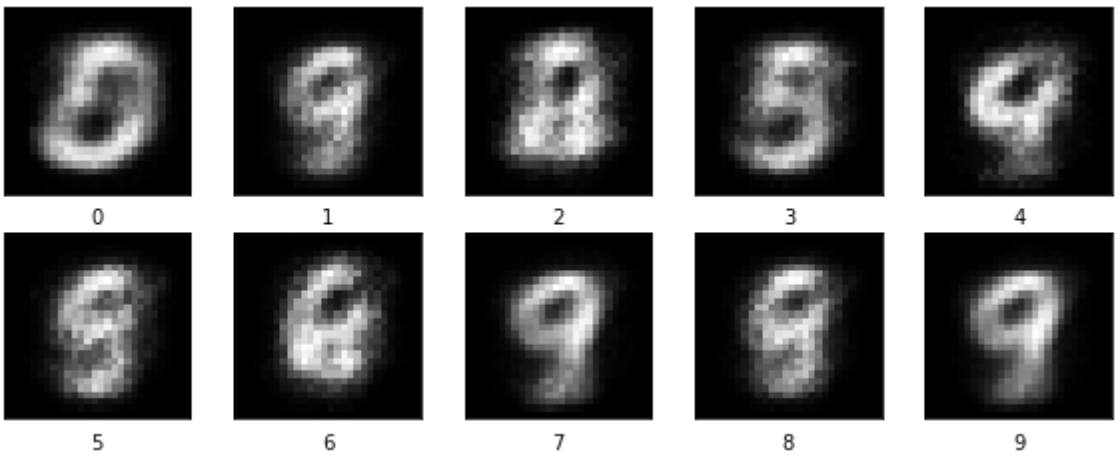
E: 21

100%| | 858/858 [00:10<00:00, 80.34it/s]



E: 22

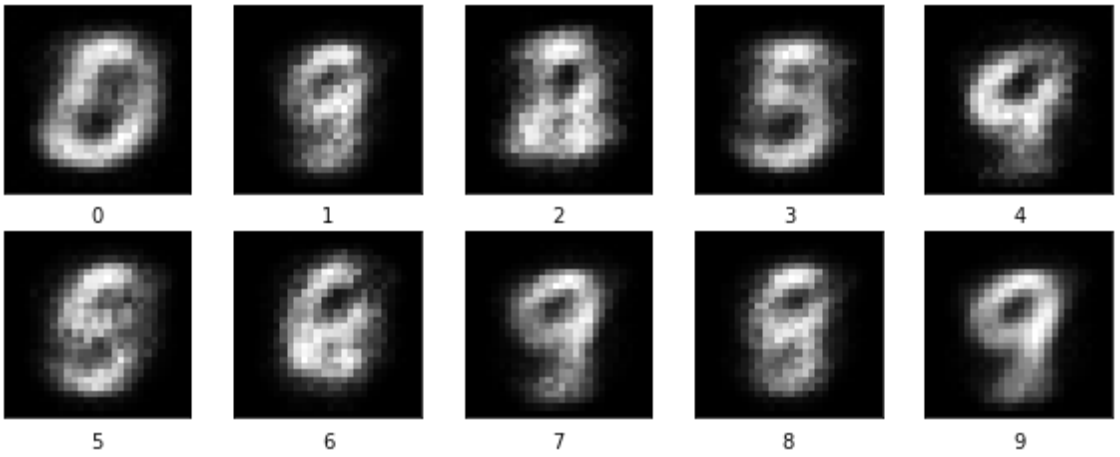
100%| | 858/858 [00:11<00:00, 76.74it/s]



E: 23

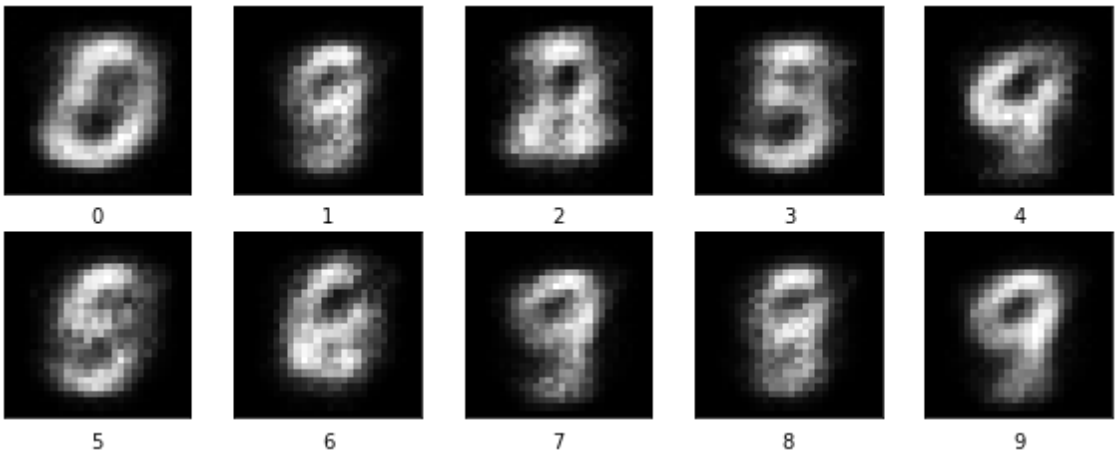
100%|

| 858/858 [00:10<00:00, 82.02it/s]



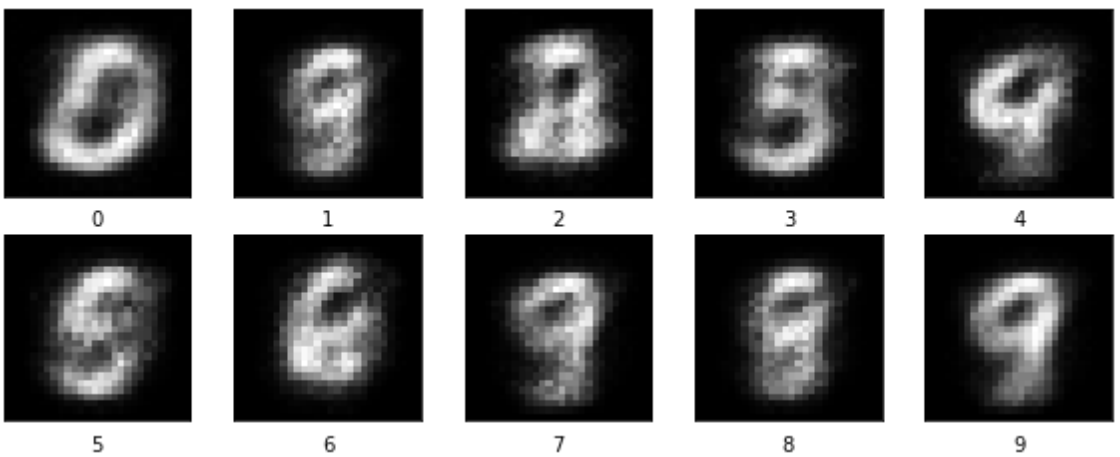
E: 24

100%| | 858/858 [00:10<00:00, 79.35it/s]



E: 25

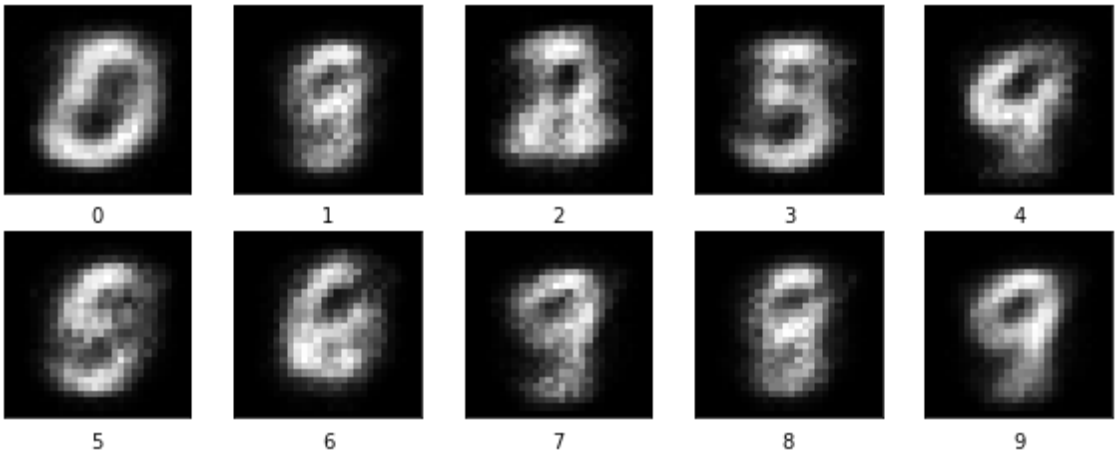
100%| | 858/858 [00:10<00:00, 82.48it/s]



E: 26

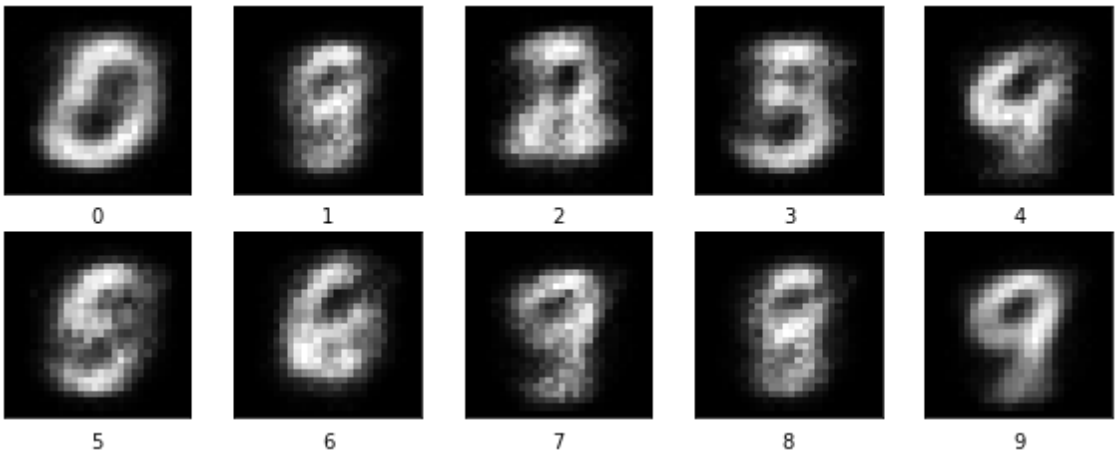
100%|

| 858/858 [00:10<00:00, 80.05it/s]



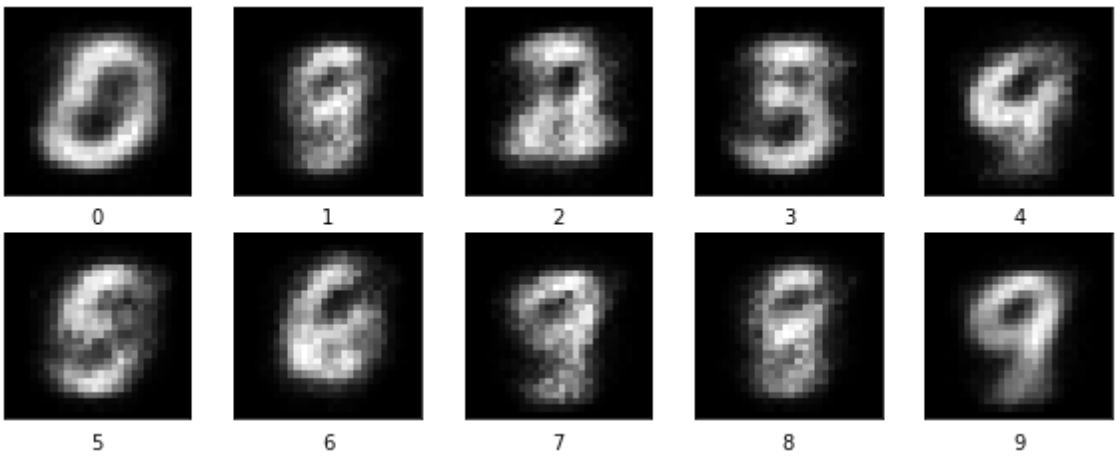
E: 27

100%| | 858/858 [00:10<00:00, 80.54it/s]



E: 28

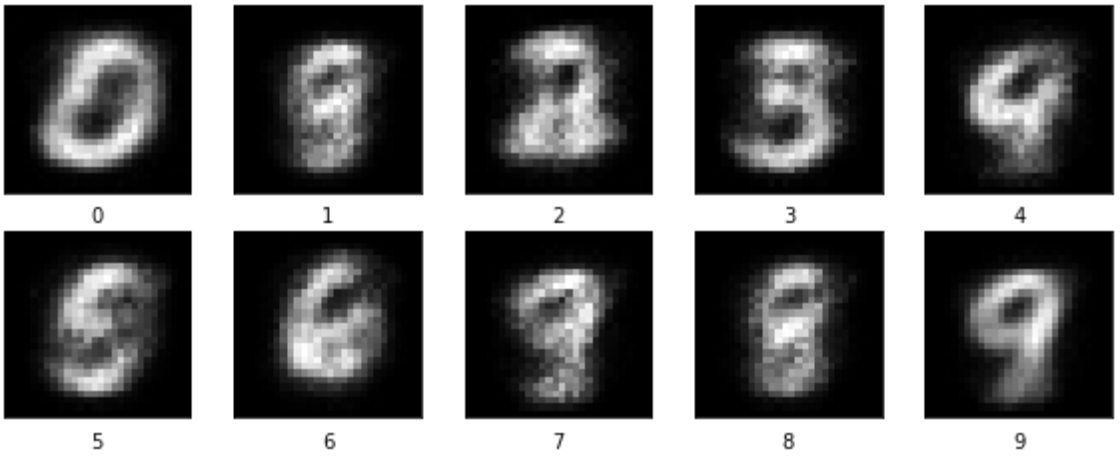
100%| | 858/858 [00:10<00:00, 80.30it/s]



E: 29

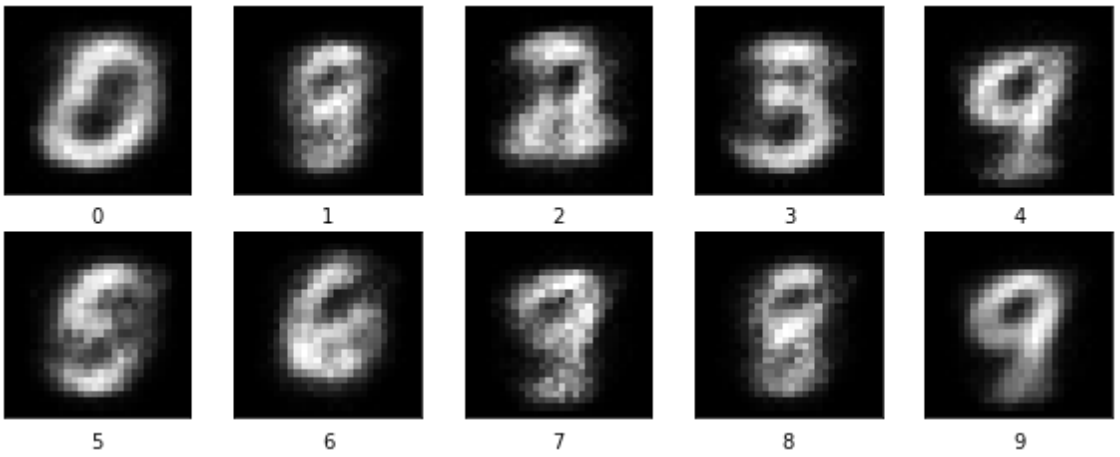
100%|

| 858/858 [00:10<00:00, 81.15it/s]



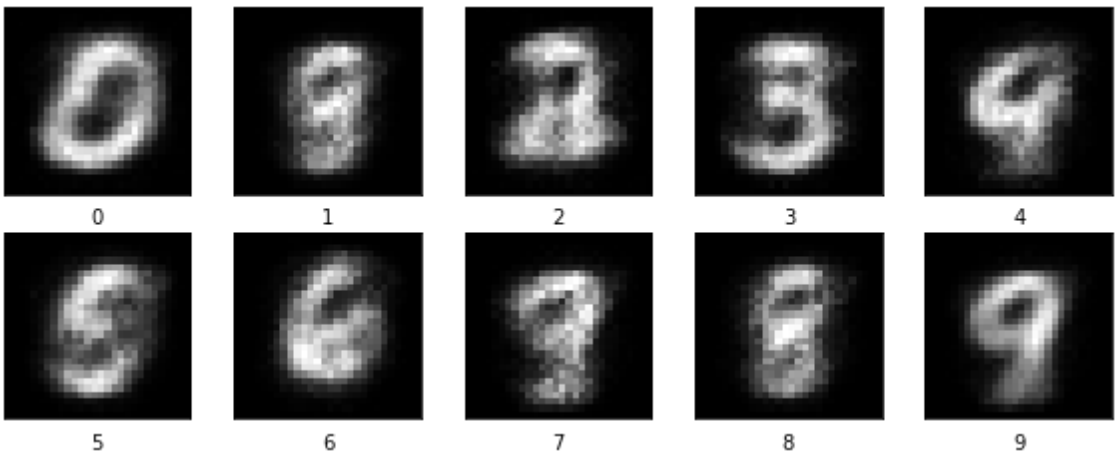
E: 30

100%| | 858/858 [00:10<00:00, 81.32it/s]



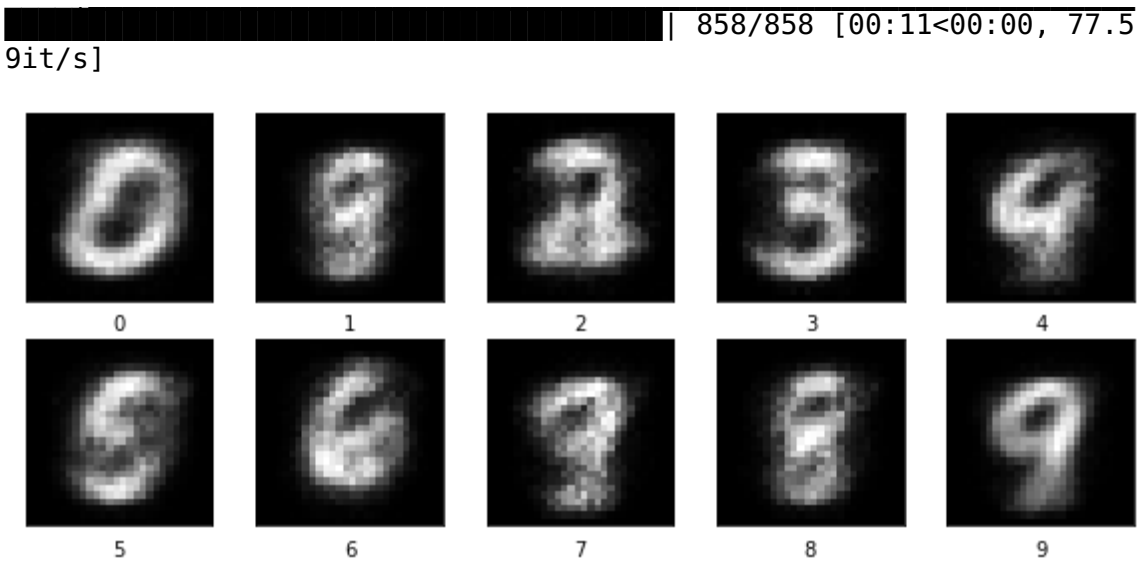
E: 31

100%| | 858/858 [00:10<00:00, 80.88it/s]

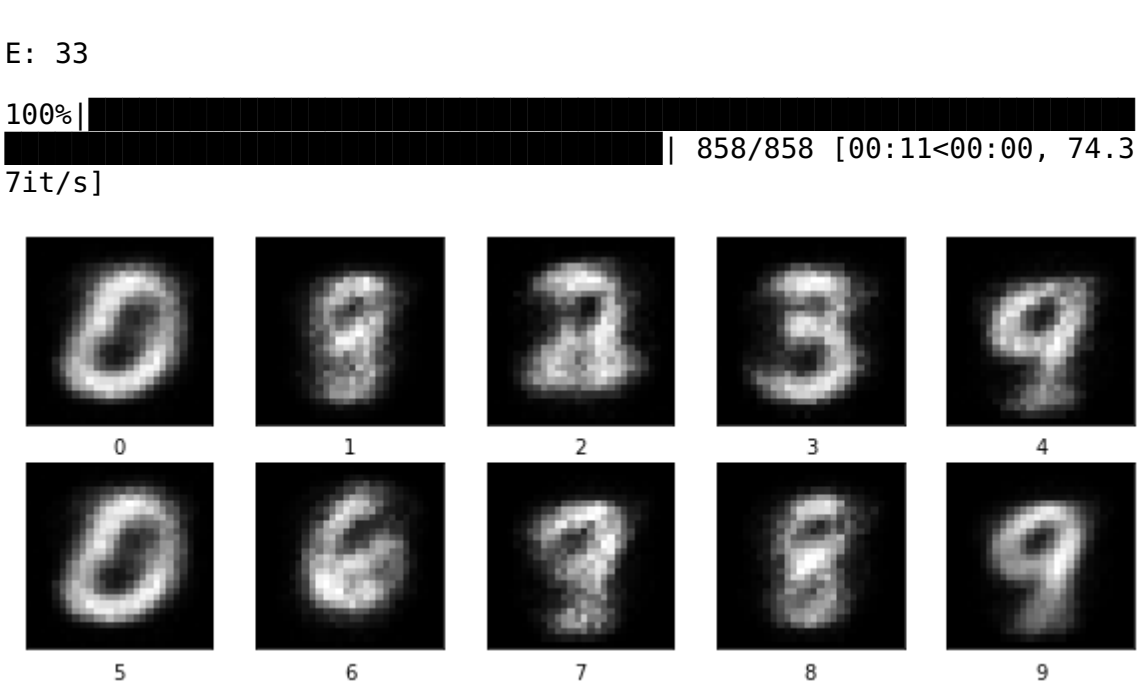


E: 32

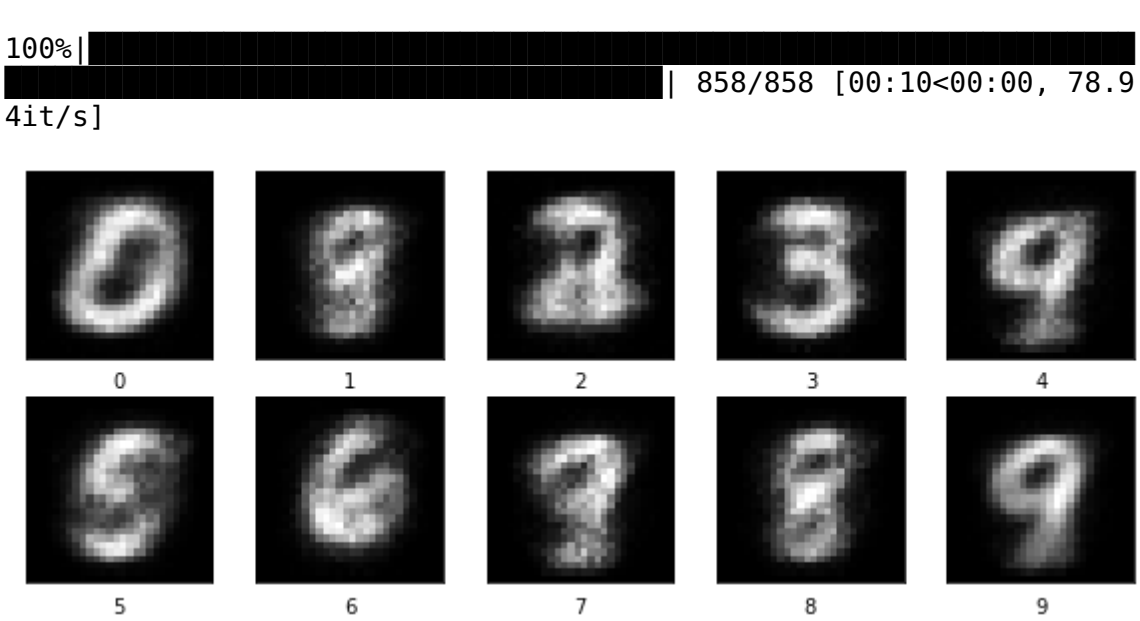
100%|



E: 33

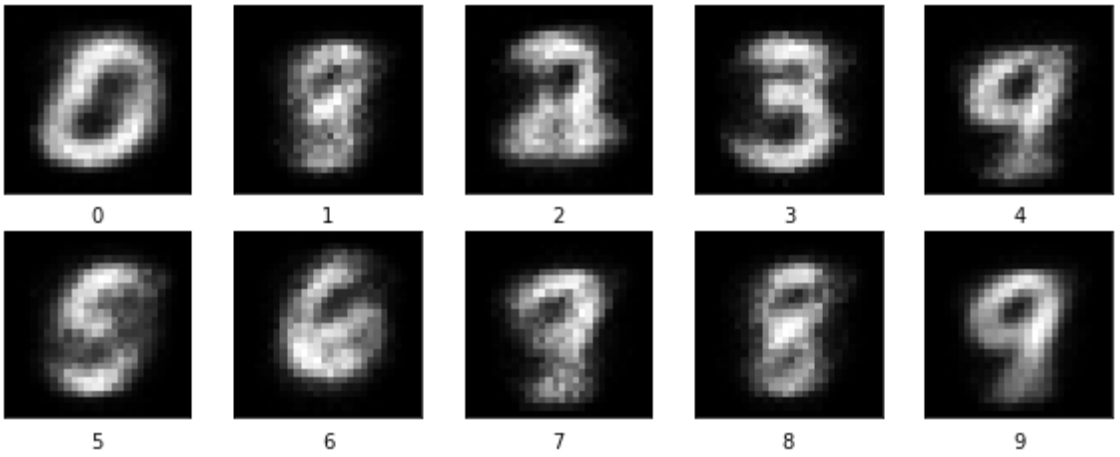


E: 34



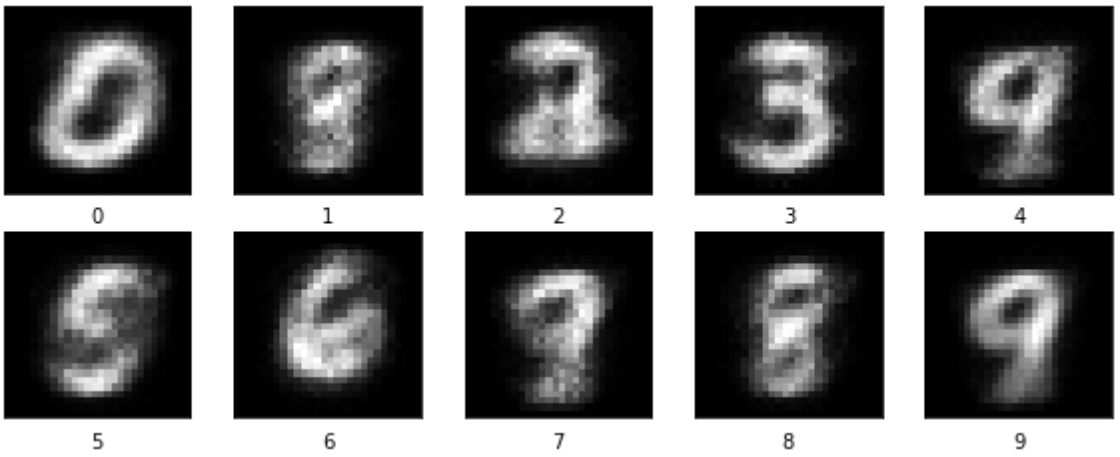
E: 35

| 858/858 [00:10<00:00, 80.0
1it/s]



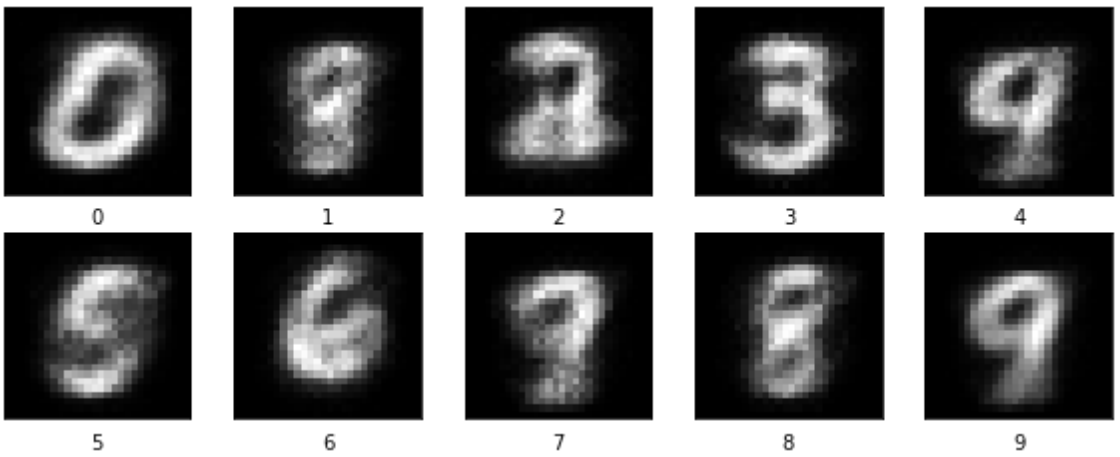
E: 36

100%|
| 858/858 [00:10<00:00, 80.6
0it/s]



E: 37

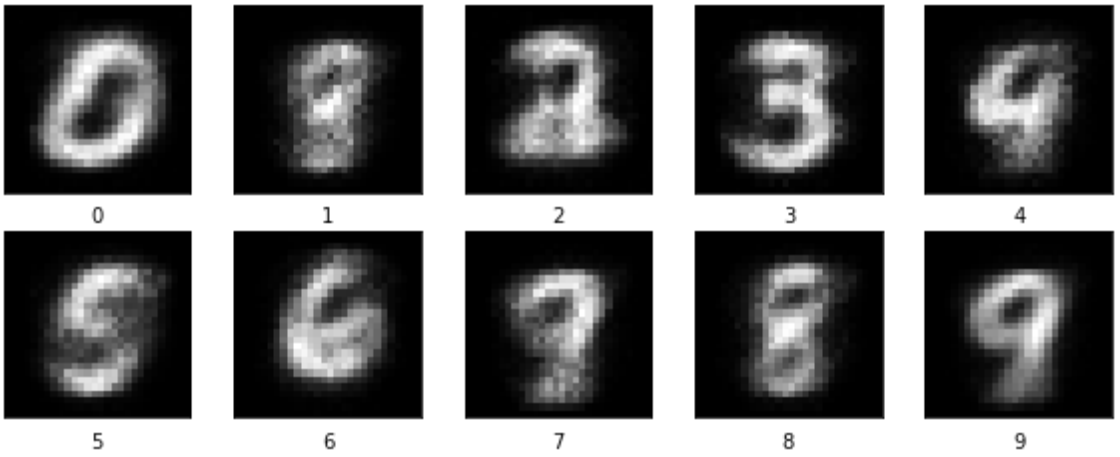
100%|
| 858/858 [00:11<00:00, 76.3
6it/s]



E: 38

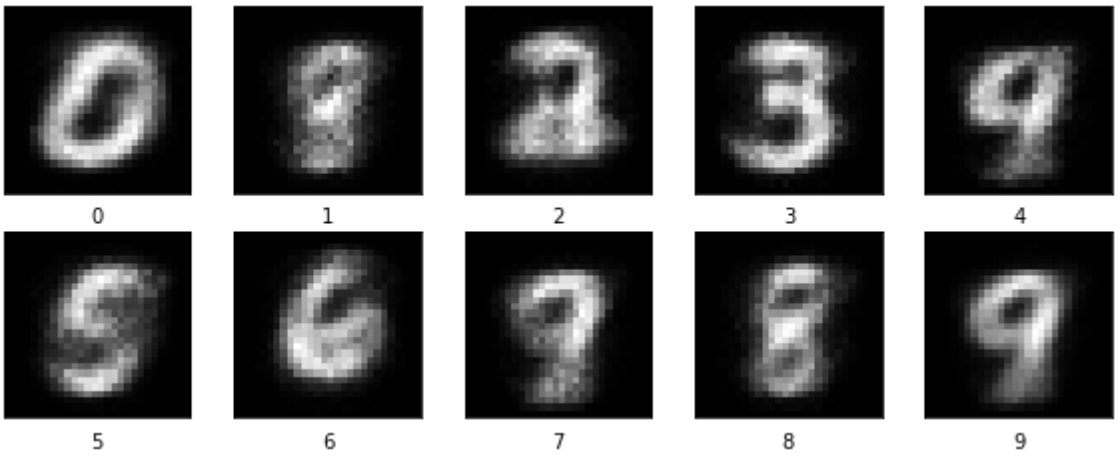
100%|

| 858/858 [00:11<00:00, 77.94it/s]



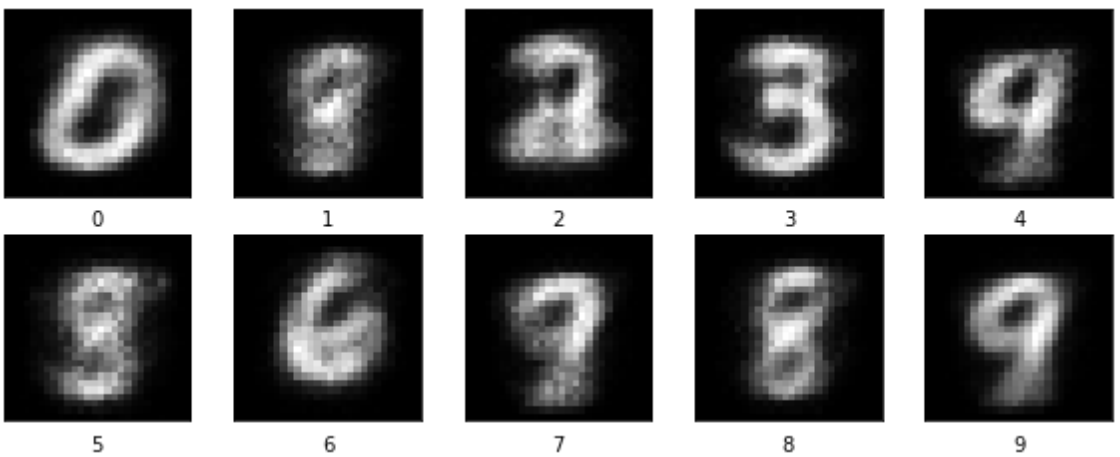
E: 39

100%| | 858/858 [00:11<00:00, 75.77it/s]



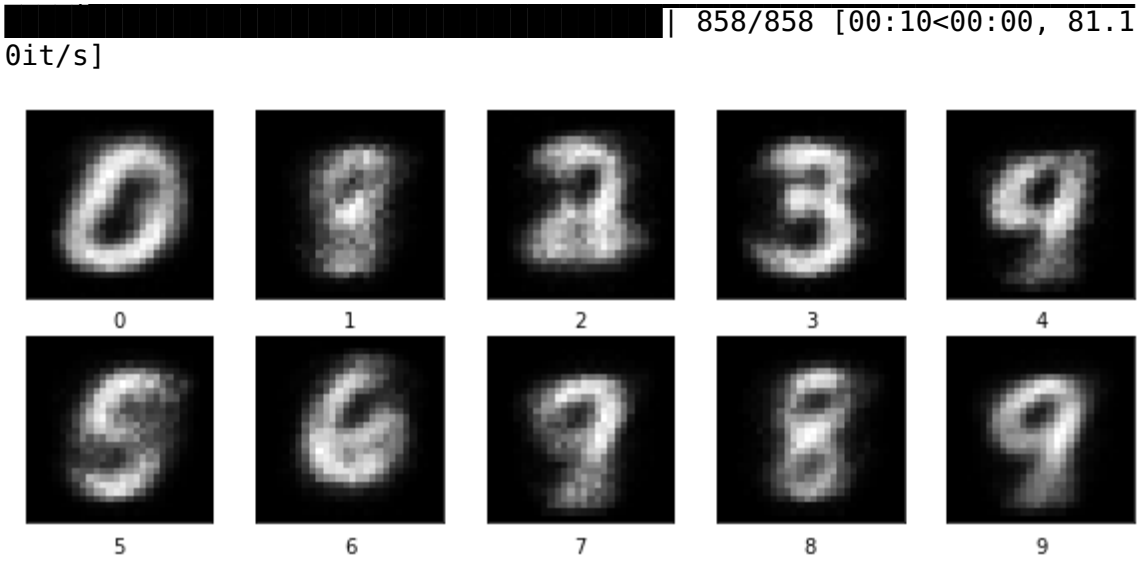
E: 40

100%| | 858/858 [00:10<00:00, 80.89it/s]

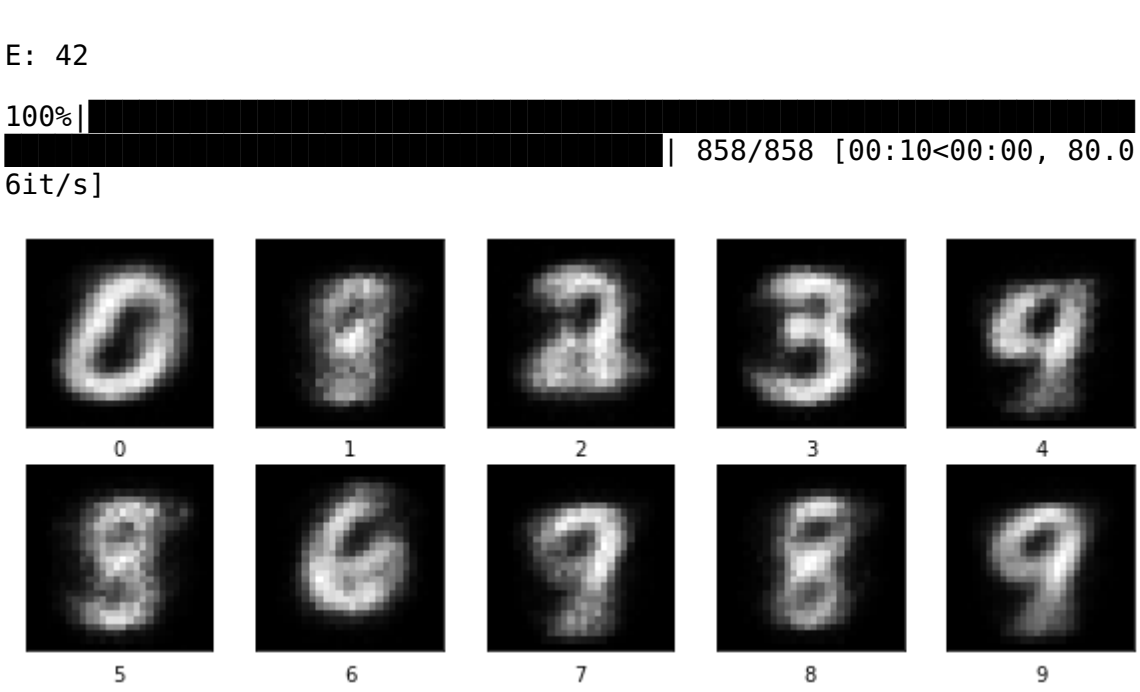


E: 41

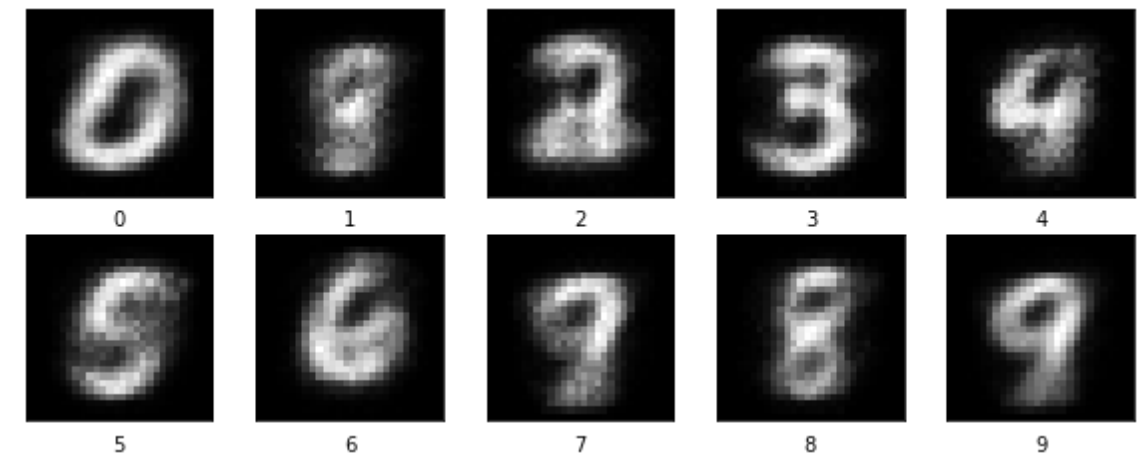
100%|



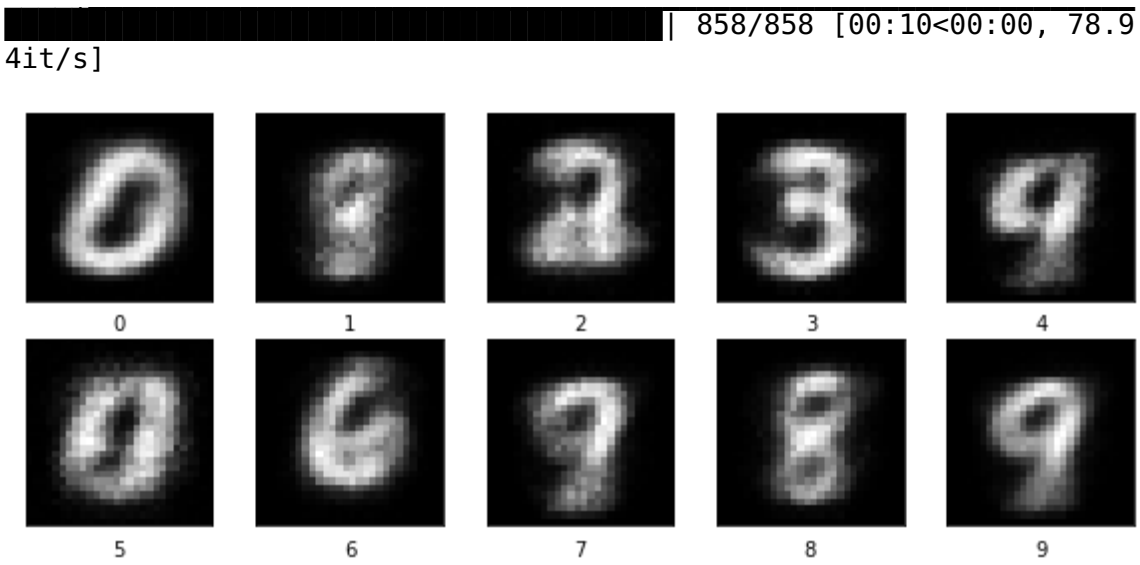
E: 42



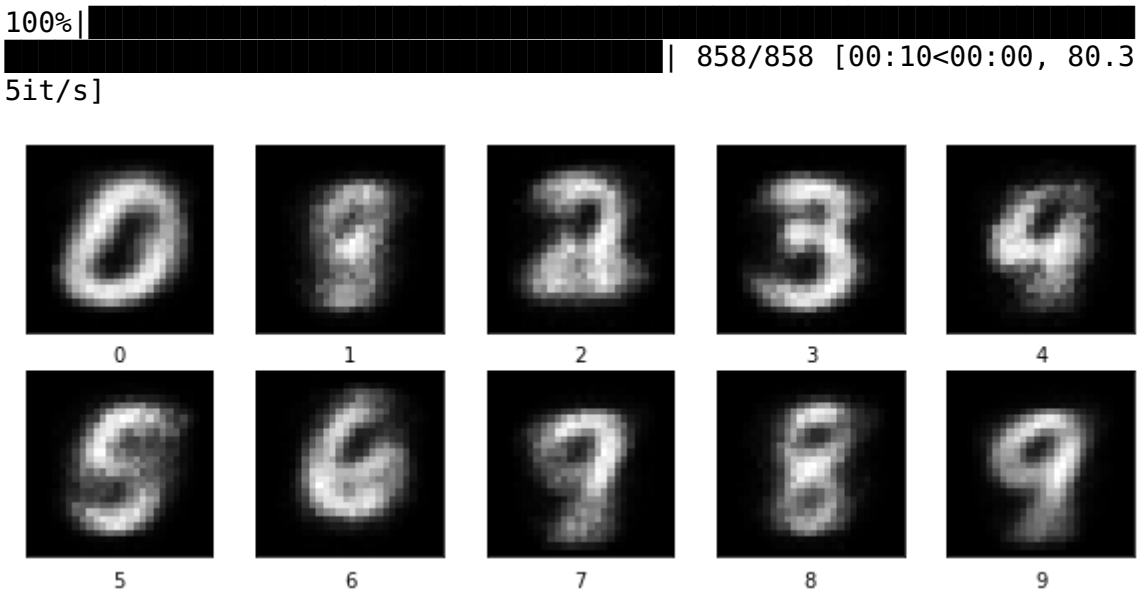
E: 43



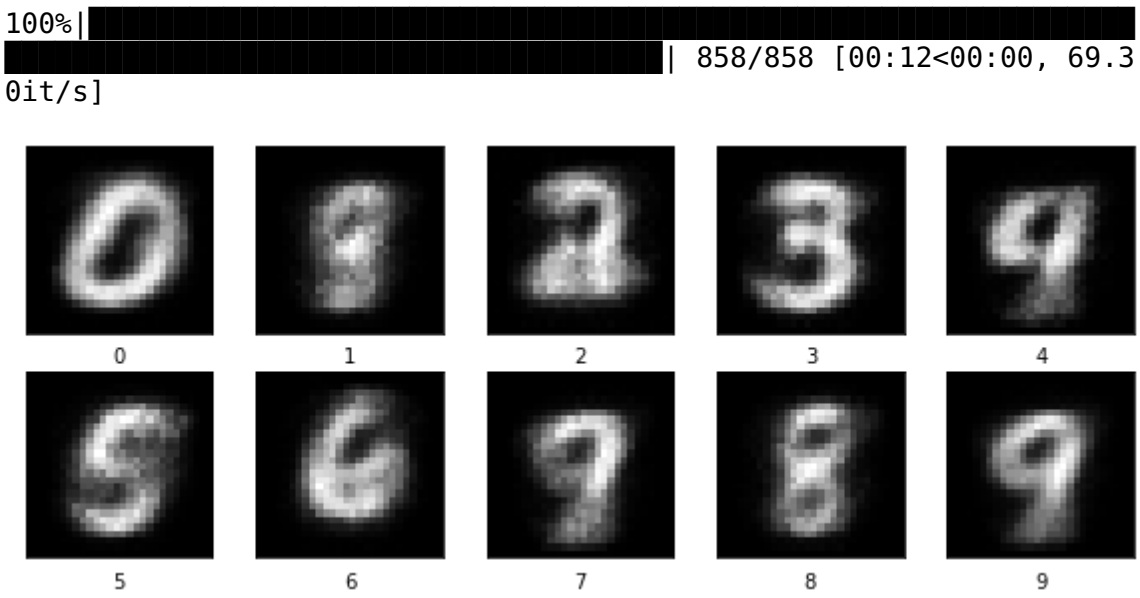
E: 44


| 858/858 [00:10<00:00, 78.94it/s]
0
1
2
3
4
5
6
7
8
9

E: 45


100%|
| 858/858 [00:10<00:00, 80.35it/s]
0
1
2
3
4
5
6
7
8
9

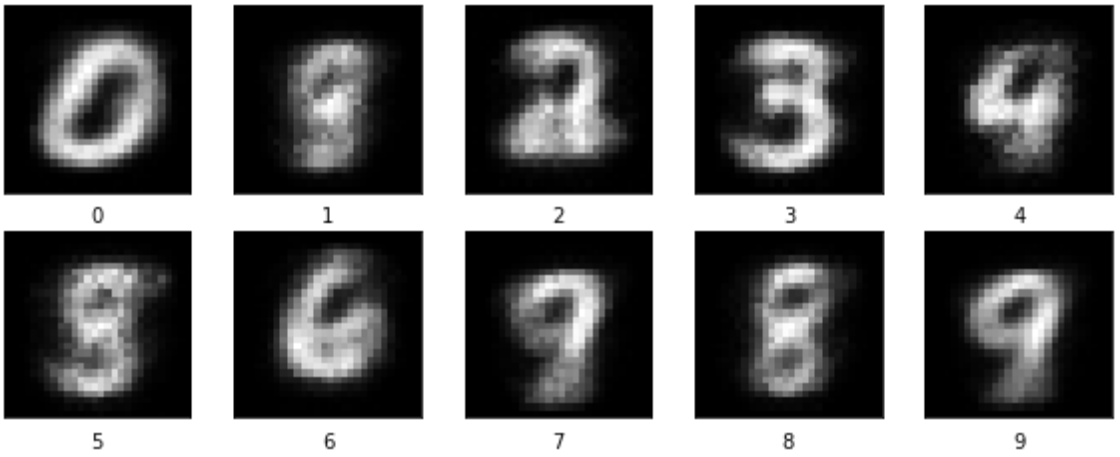
E: 46


100%|
| 858/858 [00:12<00:00, 69.30it/s]
0
1
2
3
4
5
6
7
8
9

E: 47

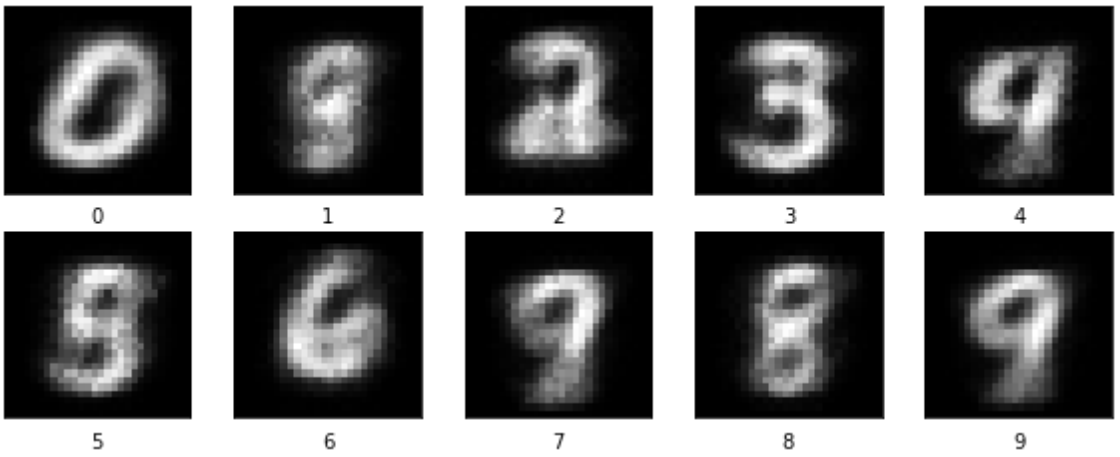

100%|

| 858/858 [00:11<00:00, 75.3
1it/s]



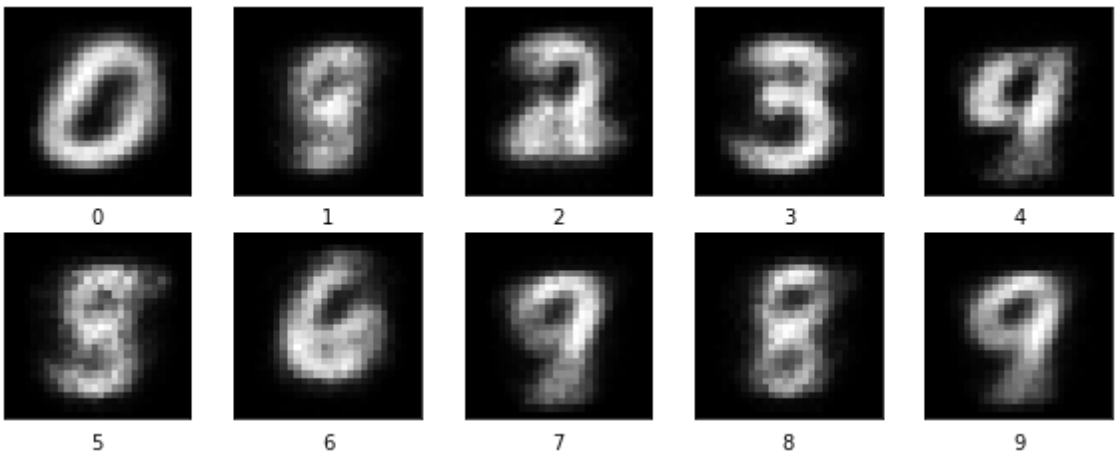
E: 48

100%|
| 858/858 [00:10<00:00, 79.8
3it/s]



E: 49

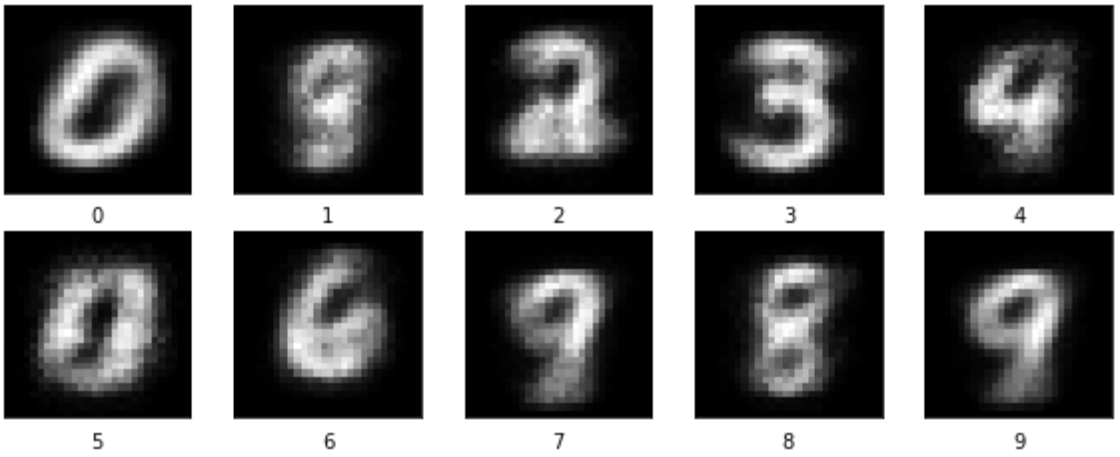
100%|
| 858/858 [00:10<00:00, 78.8
9it/s]



E: 50

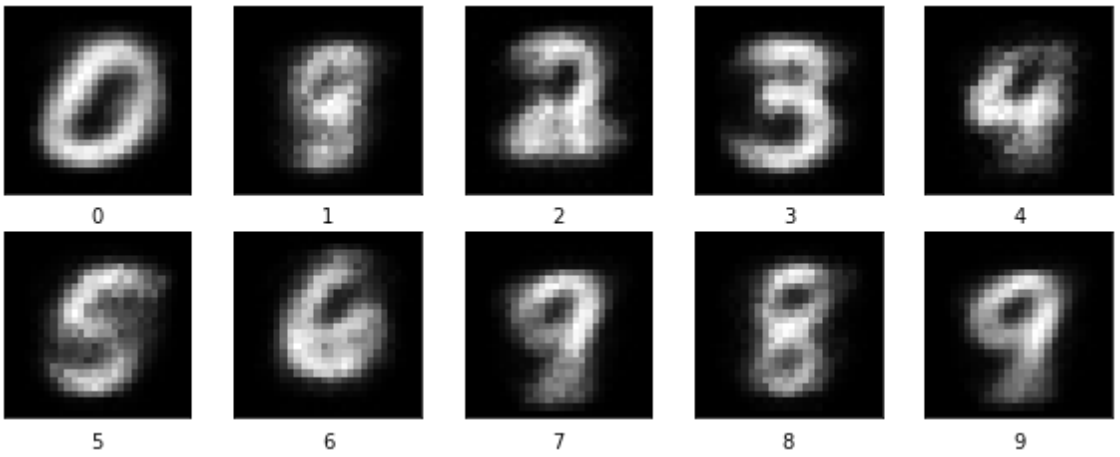
100%|

| 858/858 [00:12<00:00, 71.25it/s]



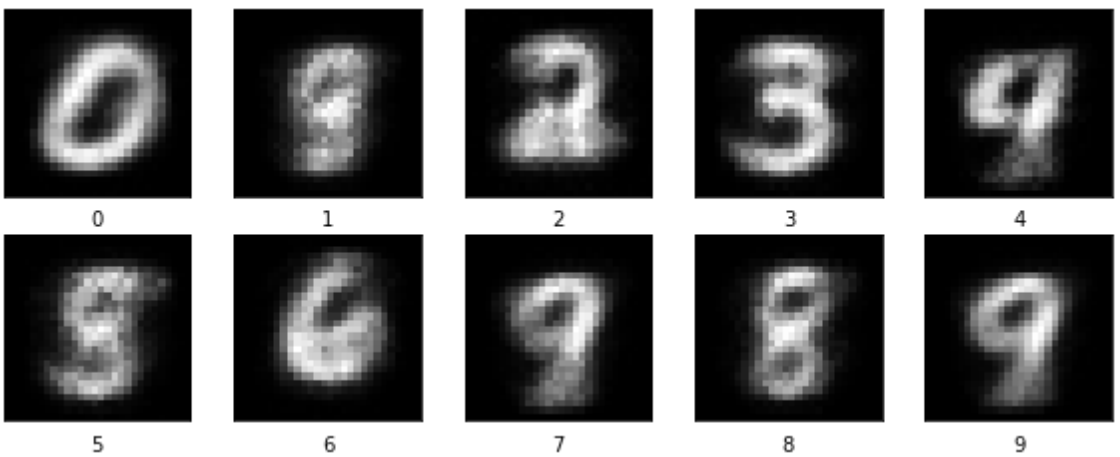
E: 51

100%| | 858/858 [00:14<00:00, 59.54it/s]



E: 52

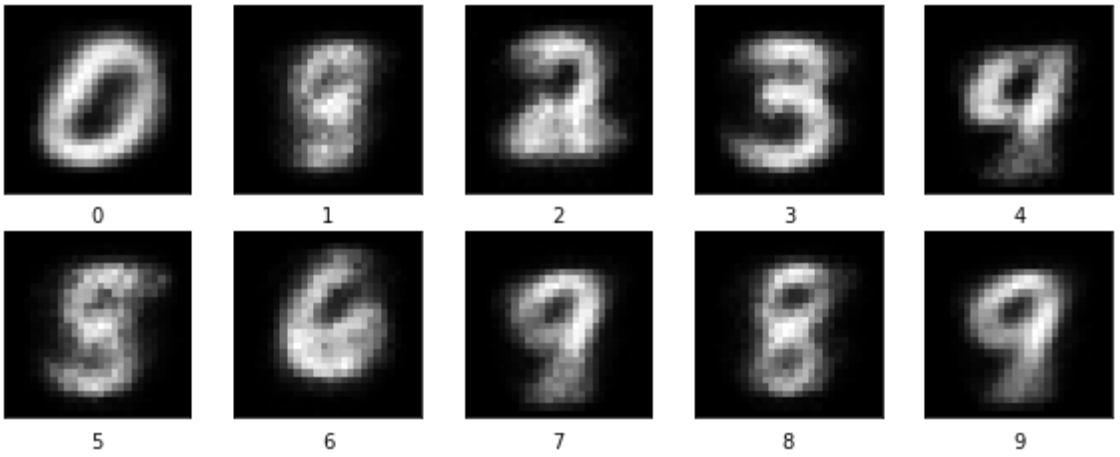
100%| | 858/858 [00:11<00:00, 71.97it/s]



E: 53

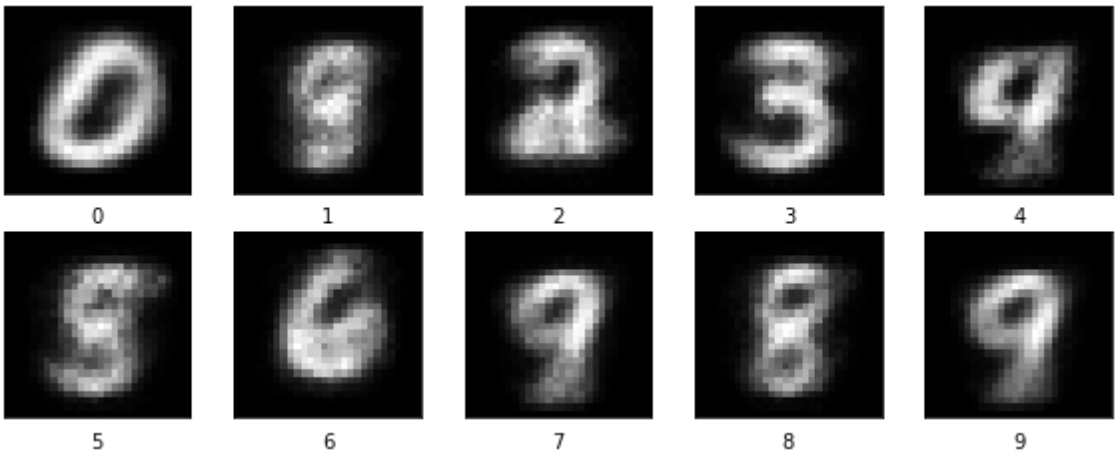
100%|

100%|| 858/858 [00:10<00:00, 79.02it/s]



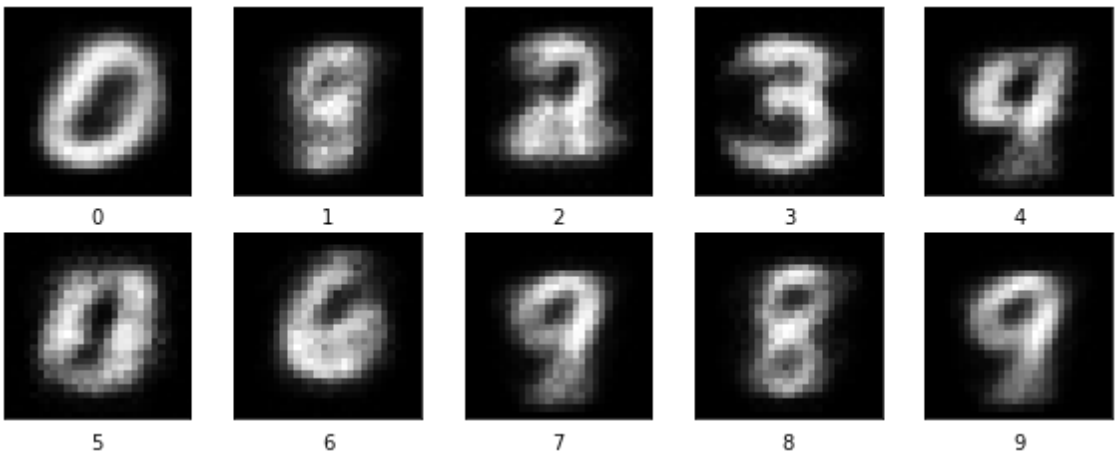
E: 54

100%|| 858/858 [00:10<00:00, 79.00it/s]



E: 55

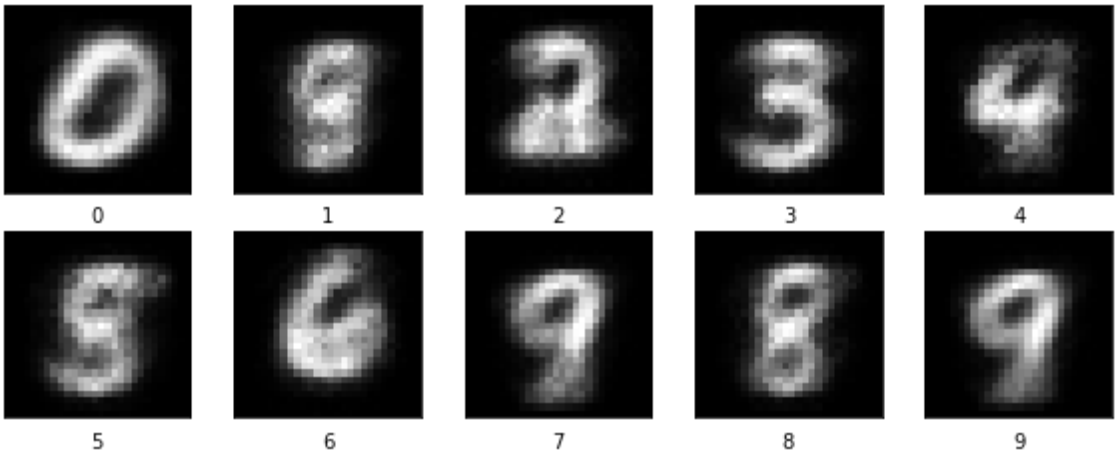
100%|| 858/858 [00:10<00:00, 79.55it/s]



E: 56

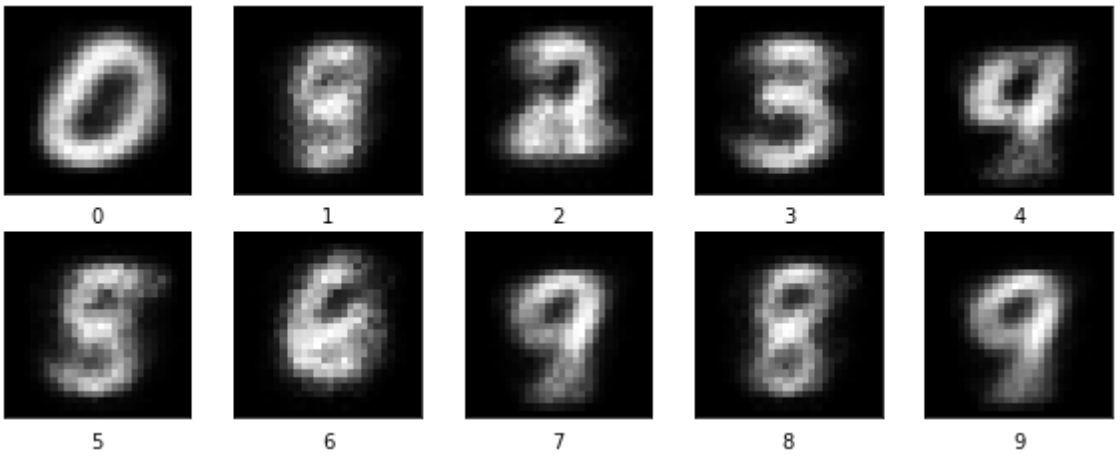
100%|

| 858/858 [00:10<00:00, 83.05it/s]



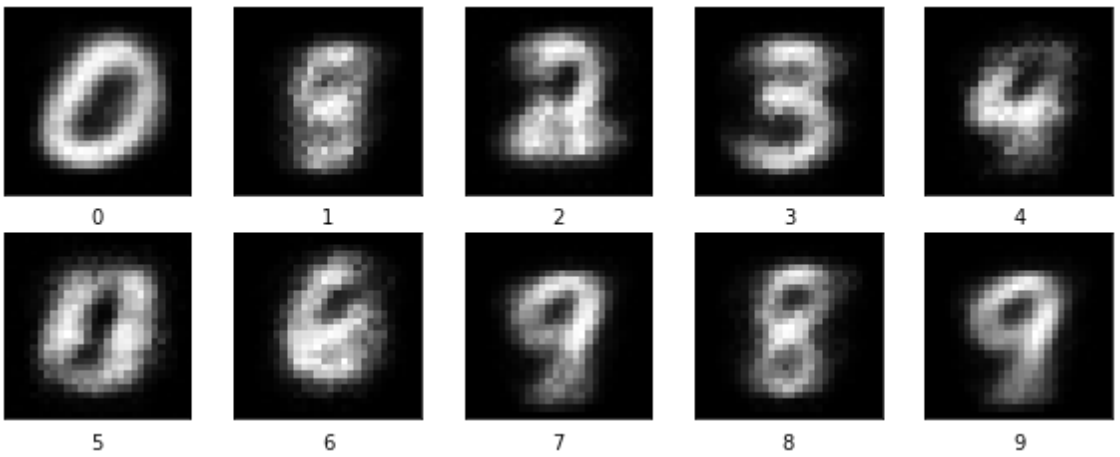
E: 57

100%| | 858/858 [00:10<00:00, 83.16it/s]



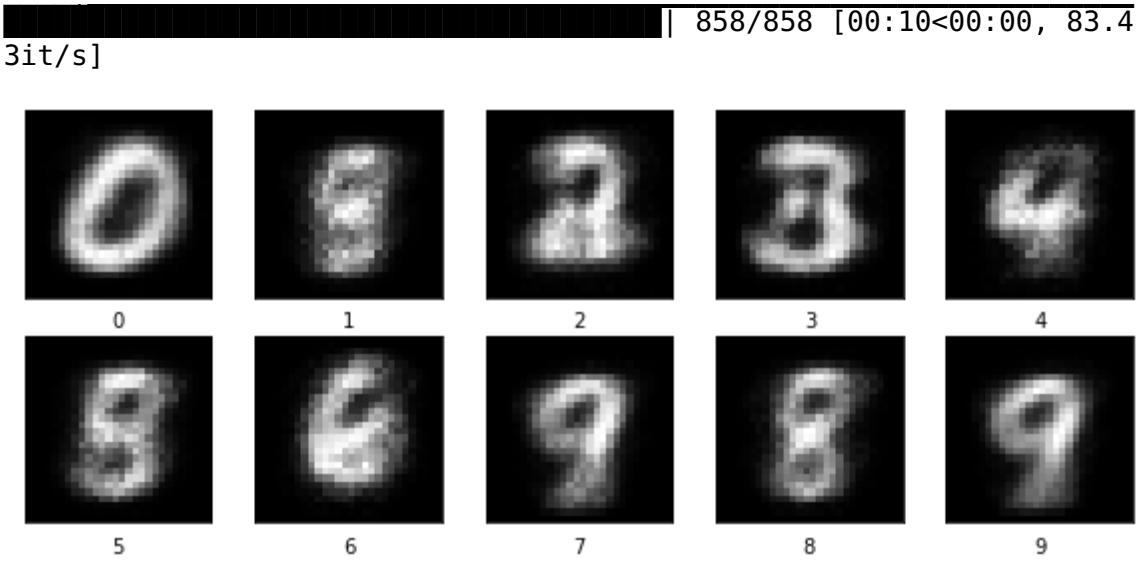
E: 58

100%| | 858/858 [00:10<00:00, 81.91it/s]

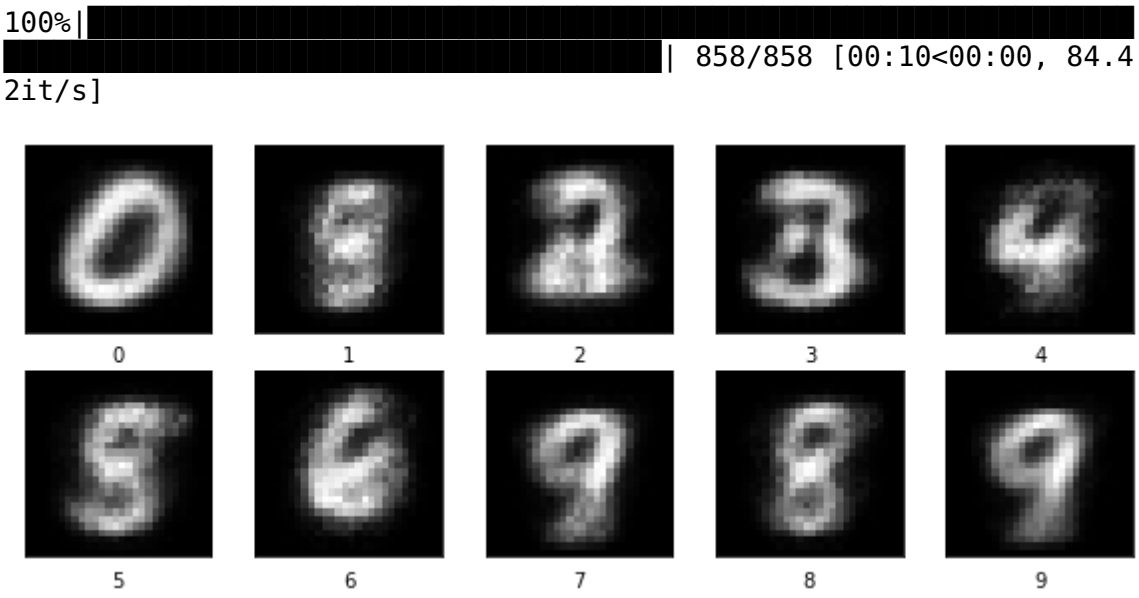


E: 59

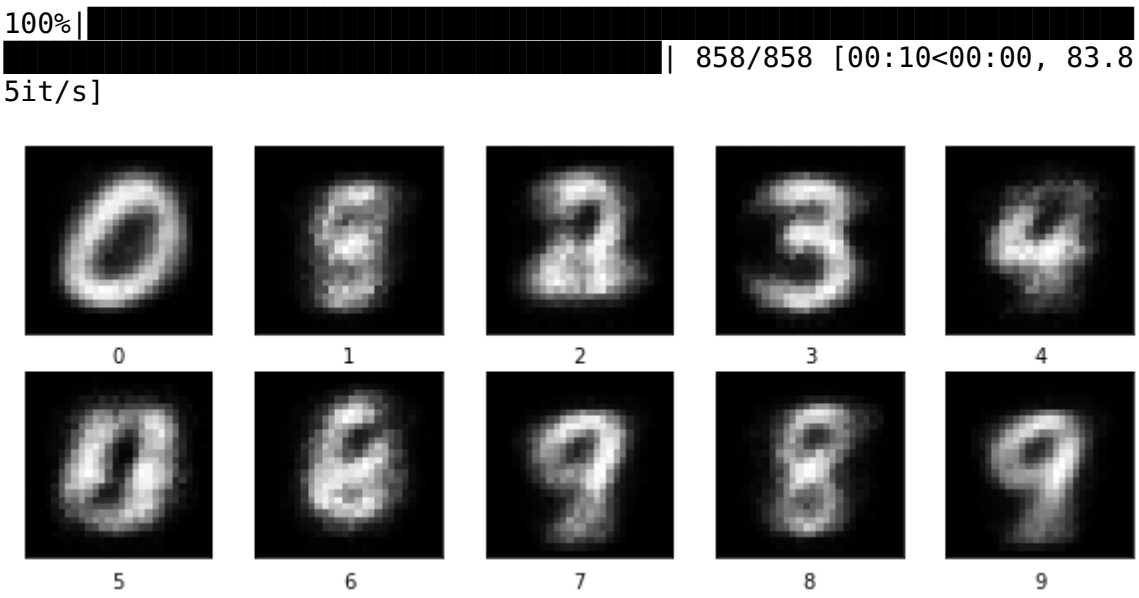
100%|


| 858/858 [00:10<00:00, 83.43it/s]
0
1
2
3
4
5
6
7
8
9

E: 60


100%|
| 858/858 [00:10<00:00, 84.42it/s]
0
1
2
3
4
5
6
7
8
9

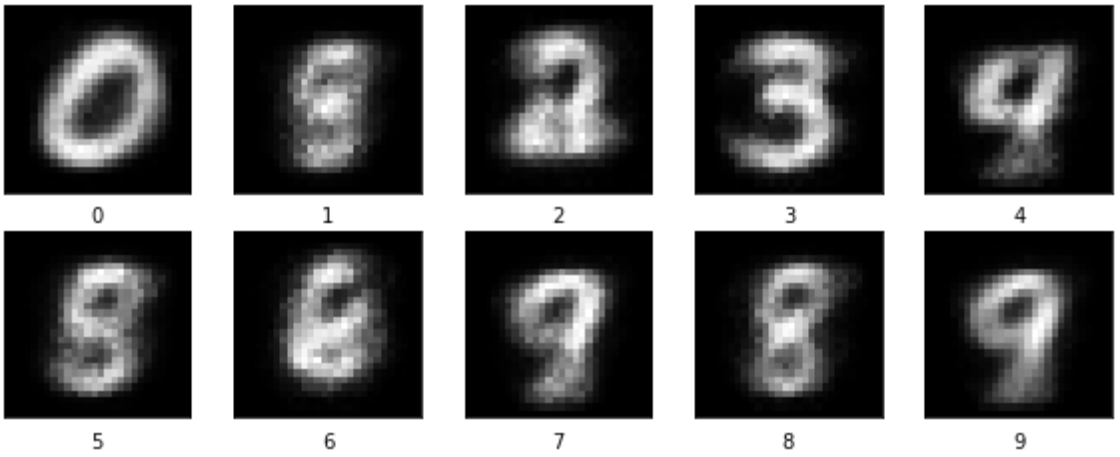
E: 61


100%|
| 858/858 [00:10<00:00, 83.85it/s]
0
1
2
3
4
5
6
7
8
9

E: 62

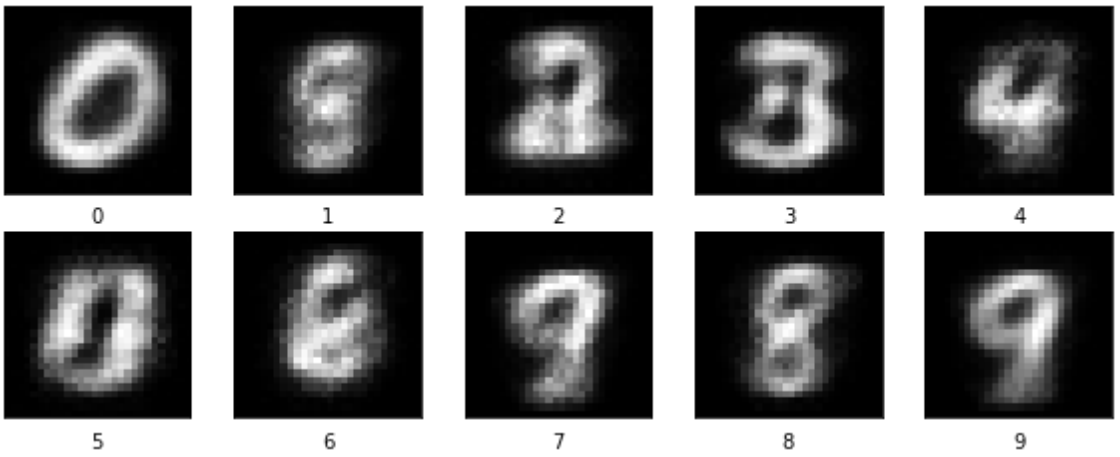

100%|

| 858/858 [00:10<00:00, 81.20it/s]



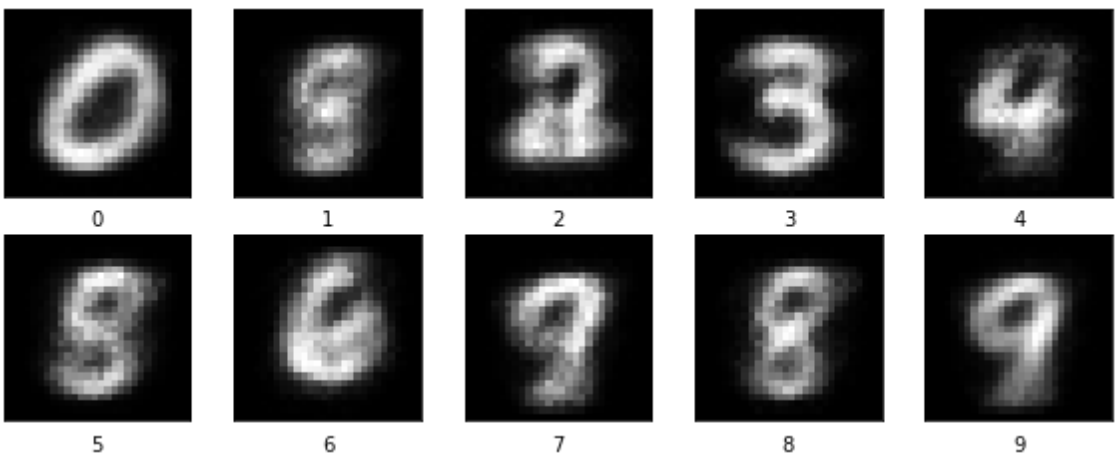
E: 63

100%| | 858/858 [00:11<00:00, 76.11it/s]



E: 64

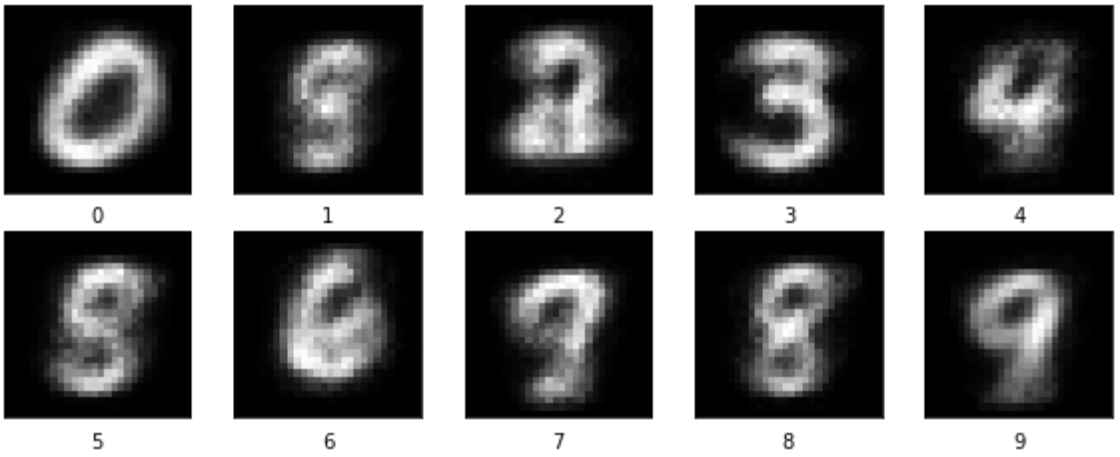
100%| | 858/858 [00:10<00:00, 78.45it/s]



E: 65

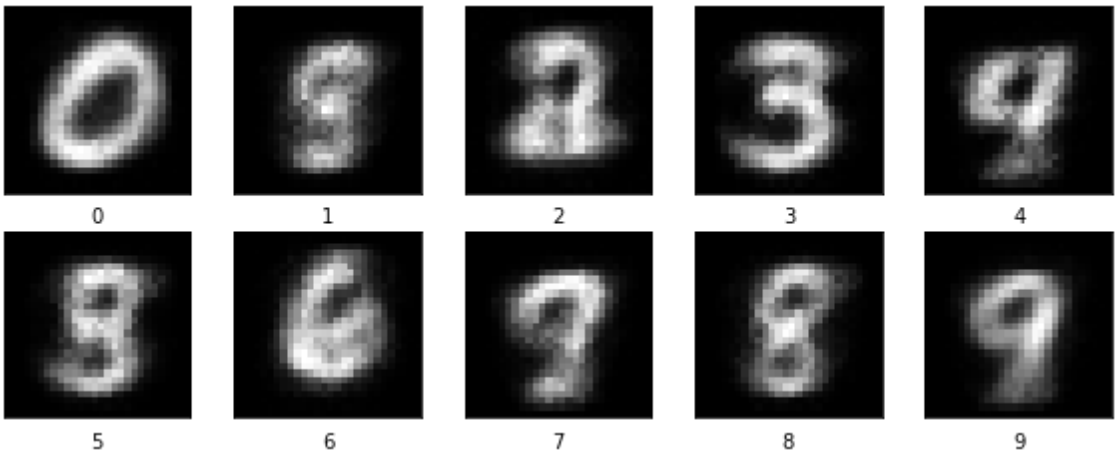
100%|

| 858/858 [00:10<00:00, 84.1
1it/s]



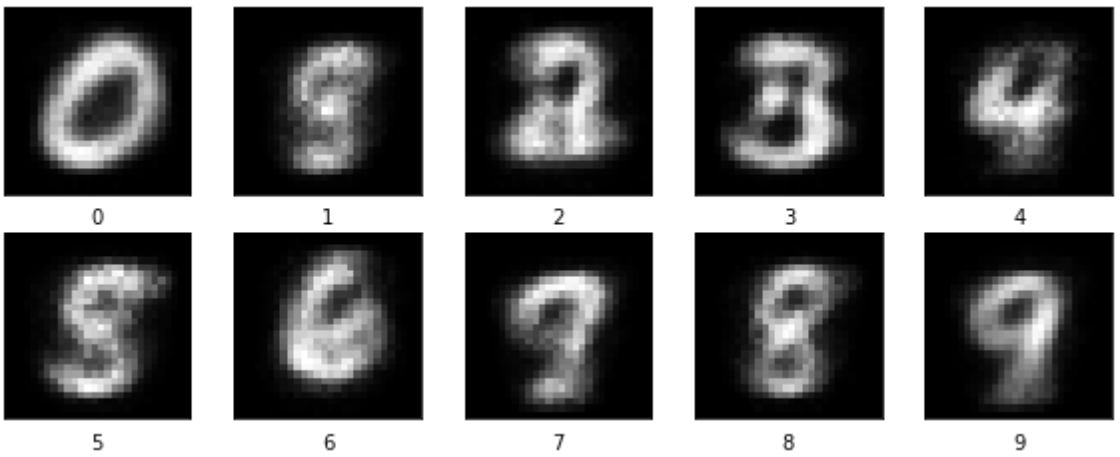
E: 66

100%|
| 858/858 [00:10<00:00, 83.1
5it/s]



E: 67

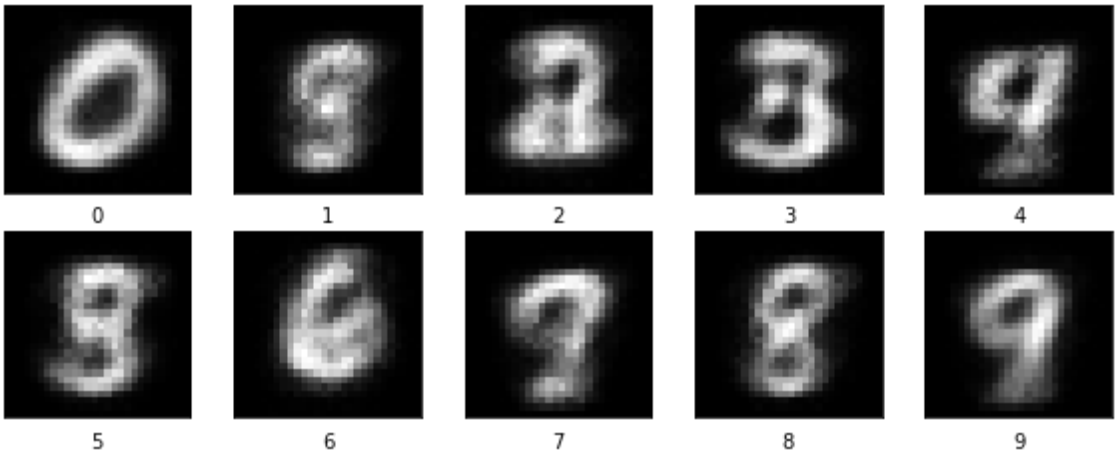
100%|
| 858/858 [00:10<00:00, 82.4
2it/s]



E: 68

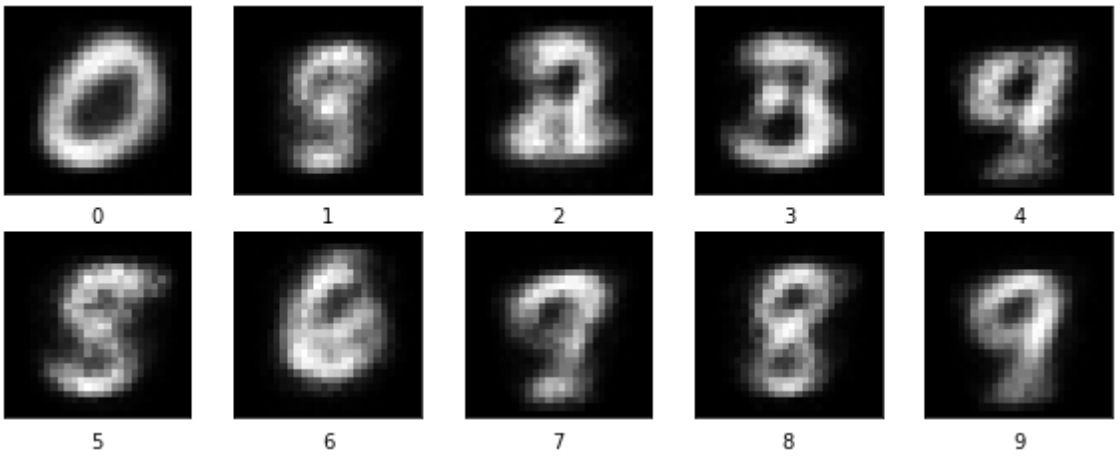
100%|

| 858/858 [00:10<00:00, 82.38it/s]



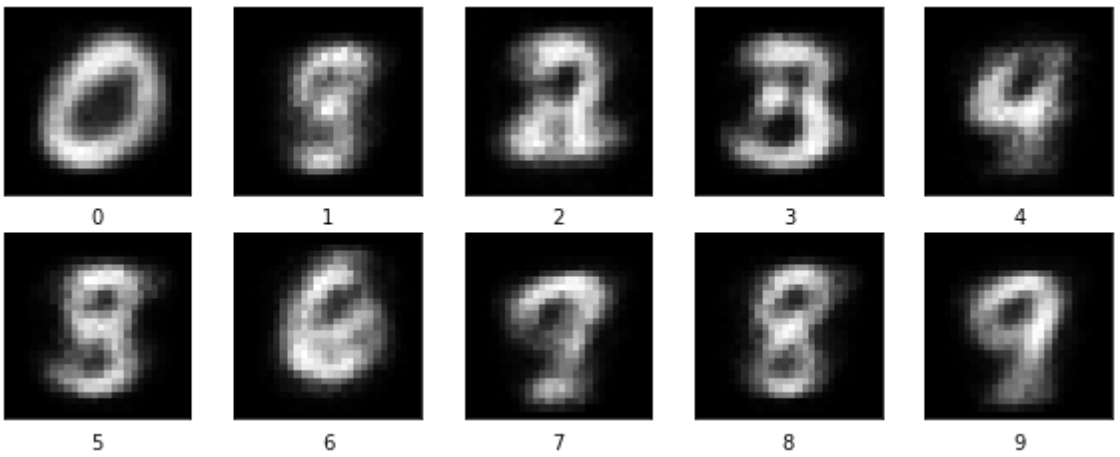
E: 69

100%| | 858/858 [00:10<00:00, 83.46it/s]



E: 70

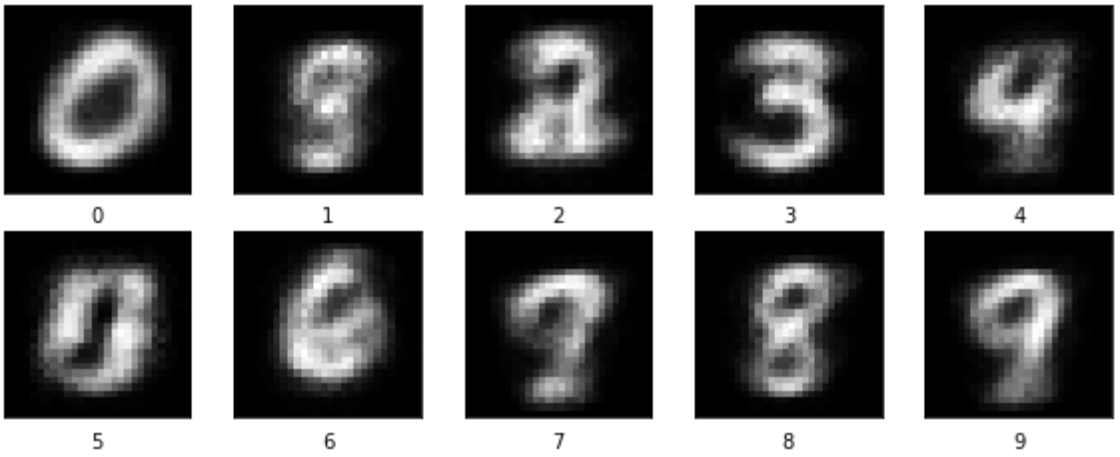
100%| | 858/858 [00:10<00:00, 80.79it/s]



E: 71

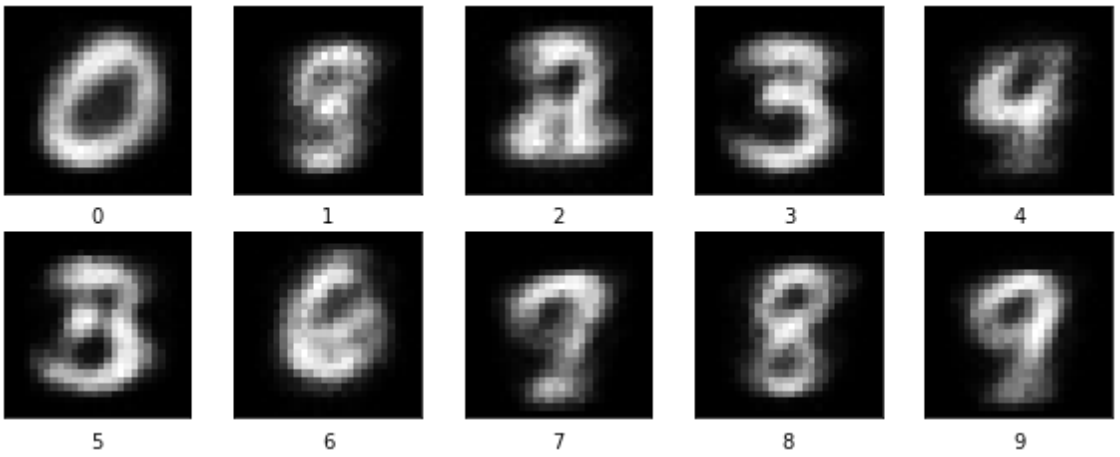
100%|

| 858/858 [00:11<00:00, 75.24it/s]



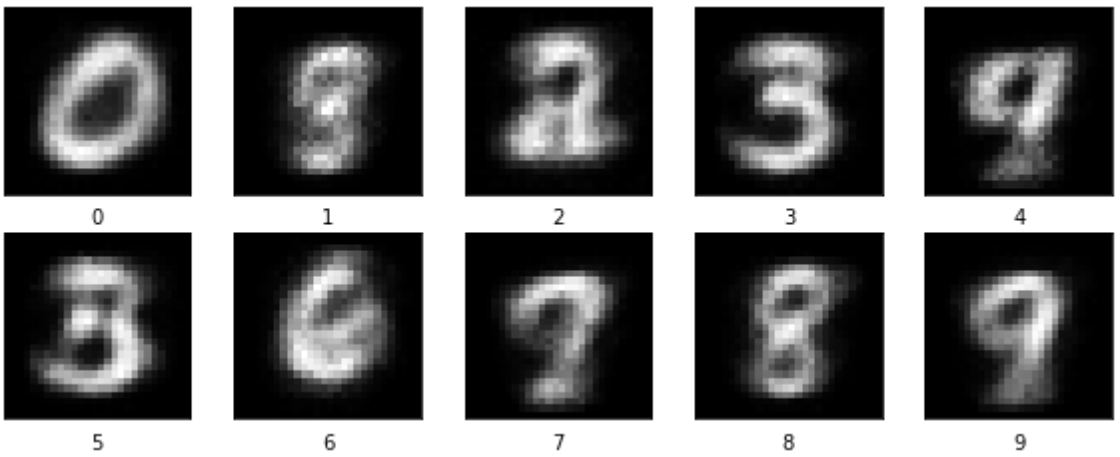
E: 72

100%| | 858/858 [00:11<00:00, 76.42it/s]



E: 73

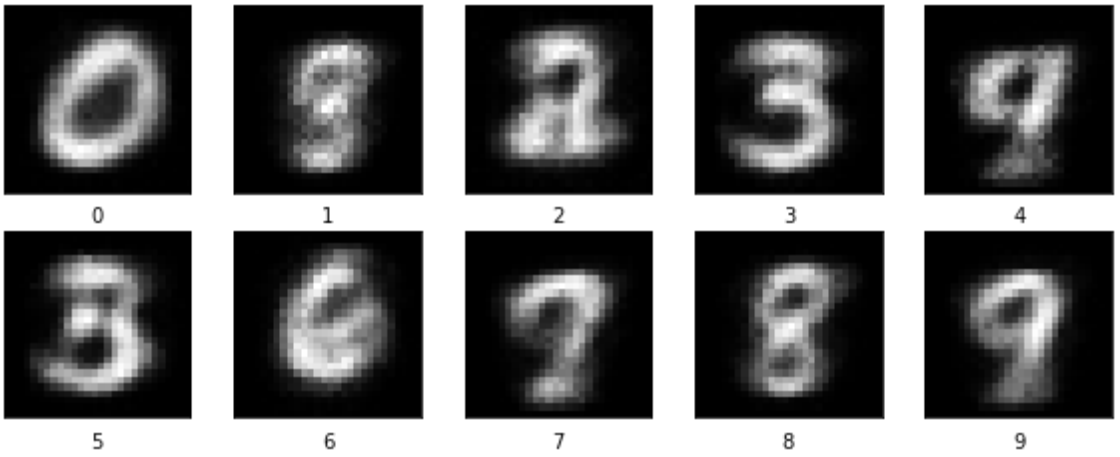
100%| | 858/858 [00:10<00:00, 79.17it/s]



E: 74

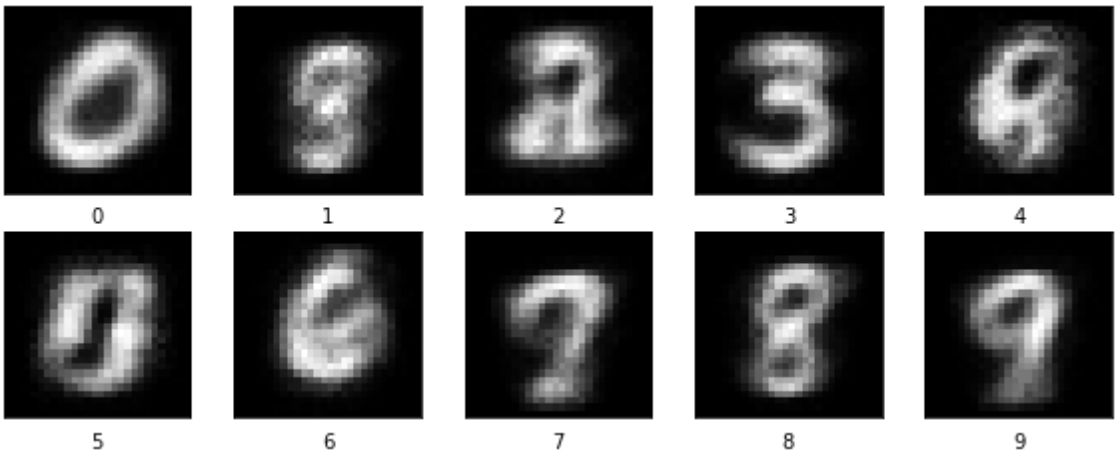
100%|

| 858/858 [00:10<00:00, 84.3
3it/s]



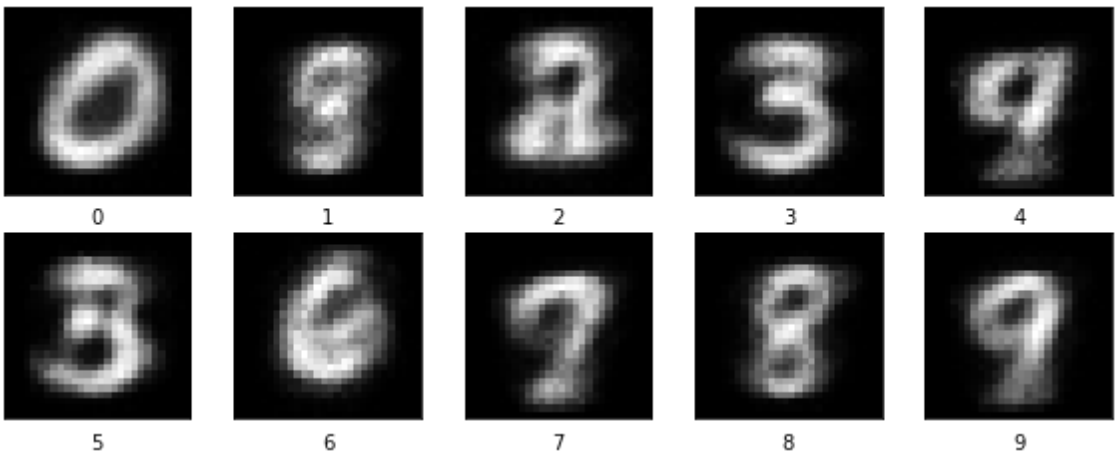
E: 75

100%|
| 858/858 [00:11<00:00, 76.7
3it/s]



E: 76

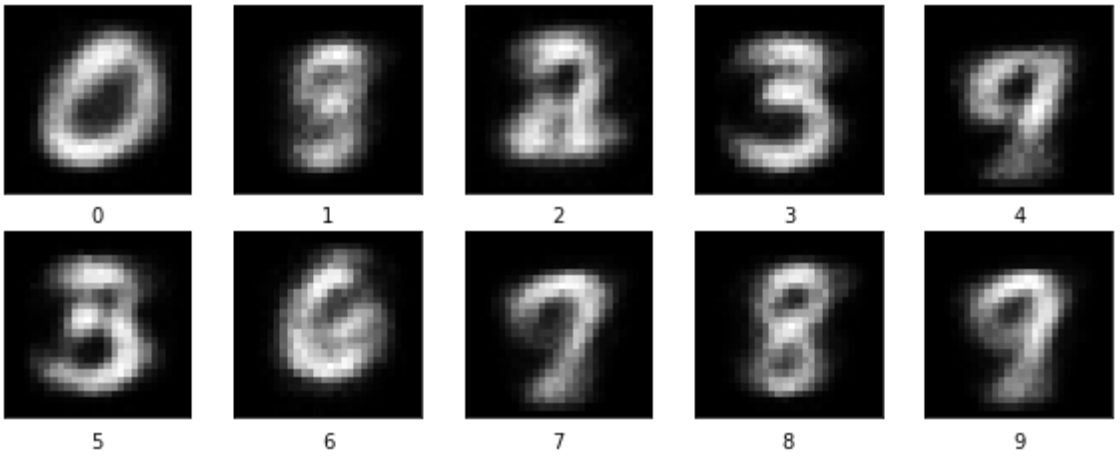
100%|
| 858/858 [00:11<00:00, 73.8
3it/s]



E: 77

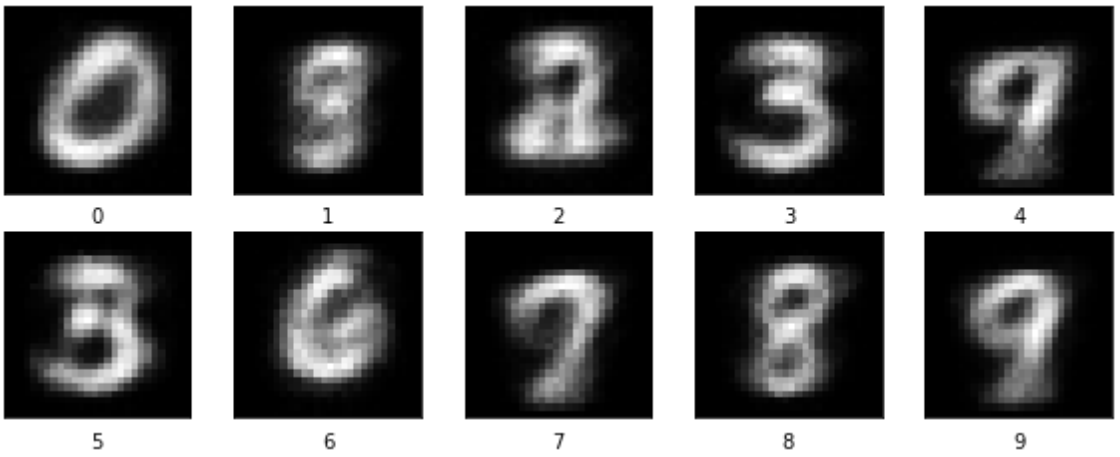
100%|

| 858/858 [00:10<00:00, 82.74it/s]



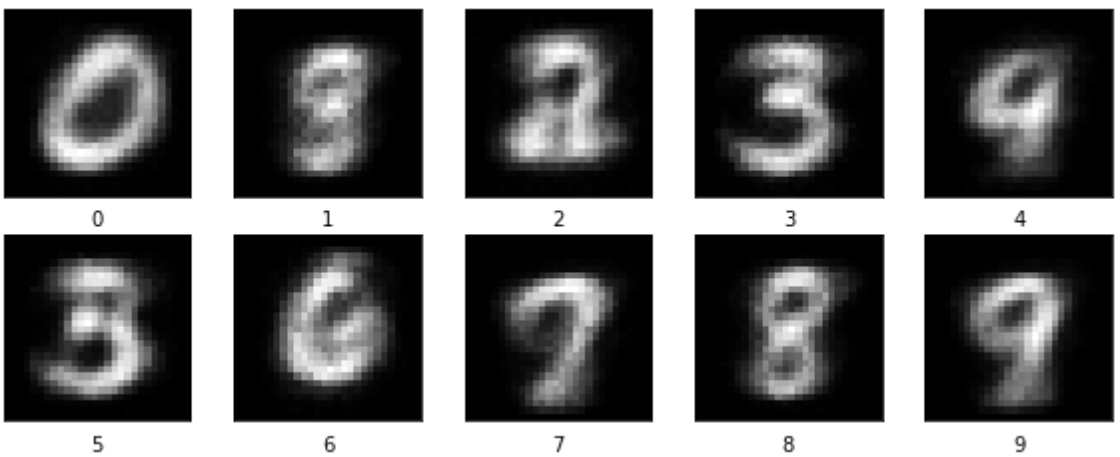
E: 78

100%| | 858/858 [00:10<00:00, 79.39it/s]



E: 79

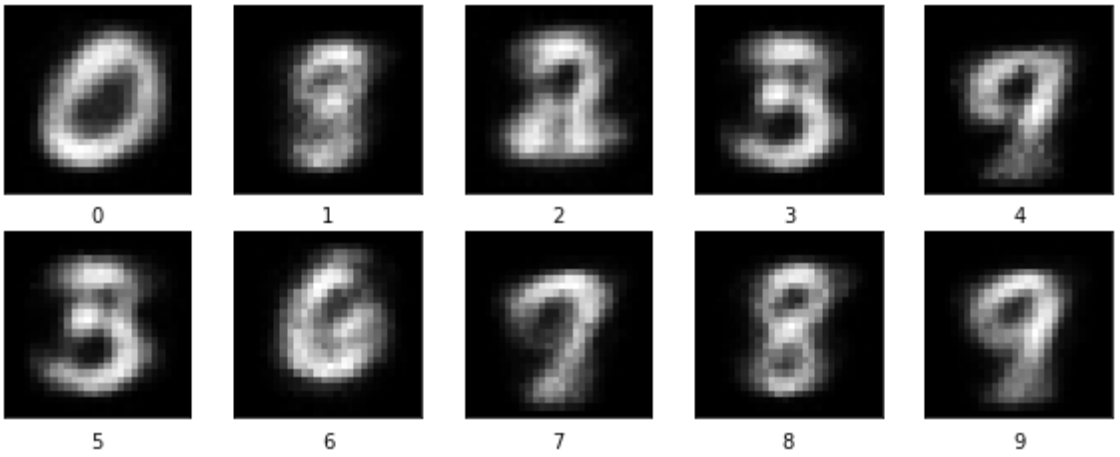
100%| | 858/858 [00:10<00:00, 82.74it/s]



E: 80

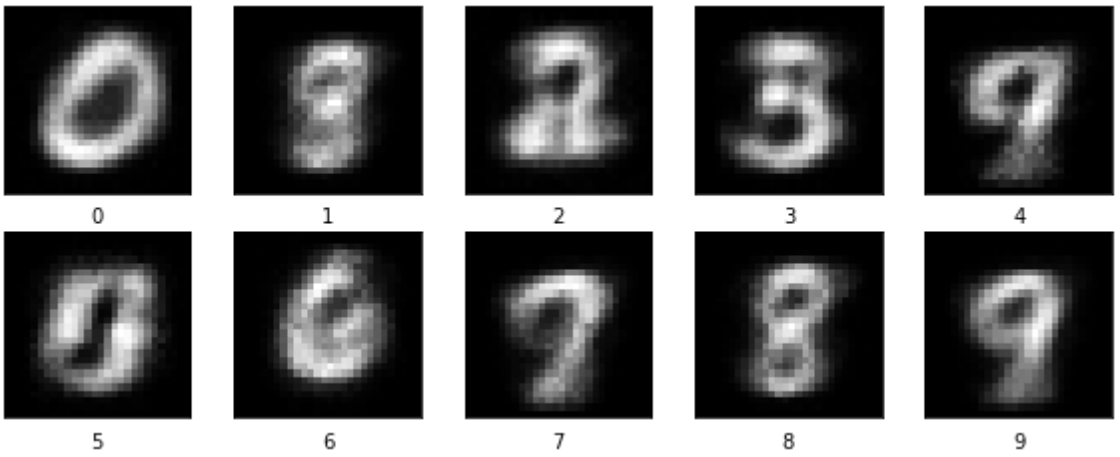
100%|

| 858/858 [00:10<00:00, 82.17it/s]



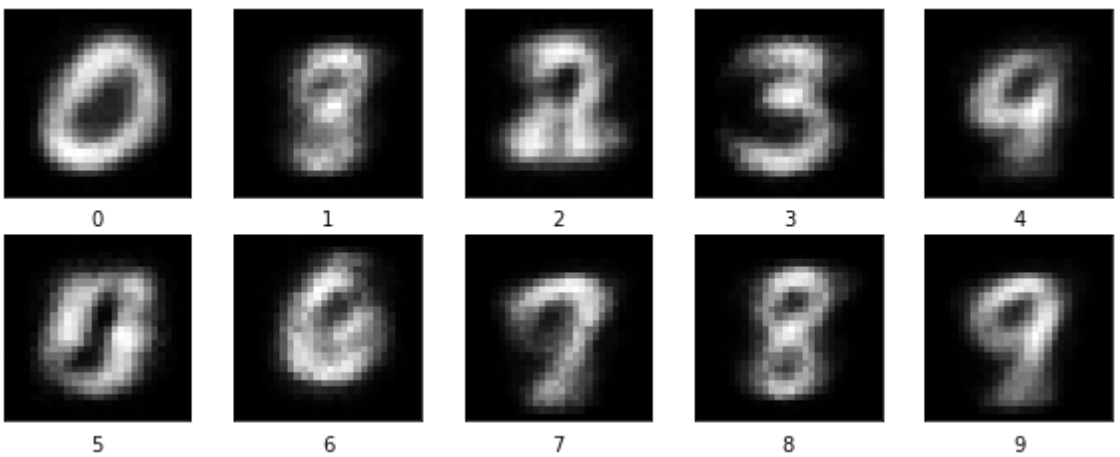
E: 81

100%| | 858/858 [00:10<00:00, 80.85it/s]



E: 82

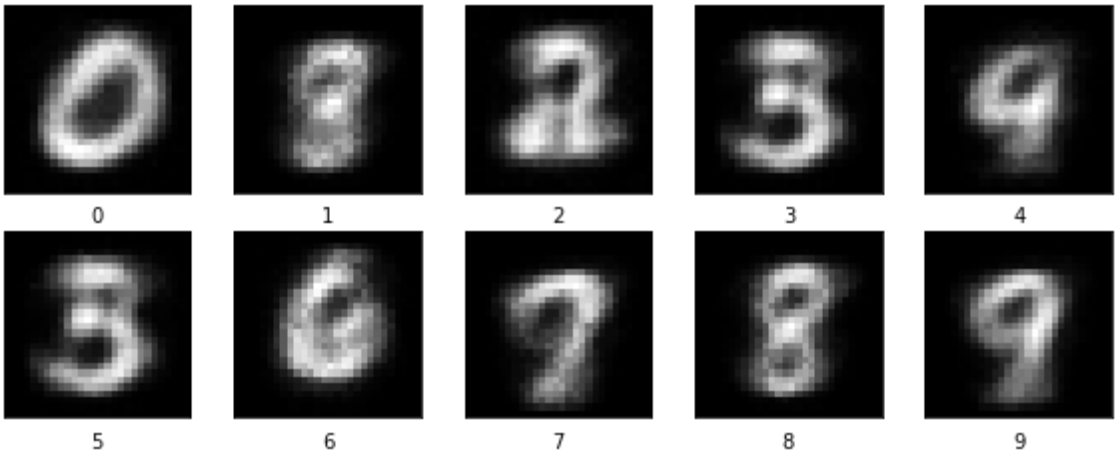
100%| | 858/858 [00:10<00:00, 82.79it/s]



E: 83

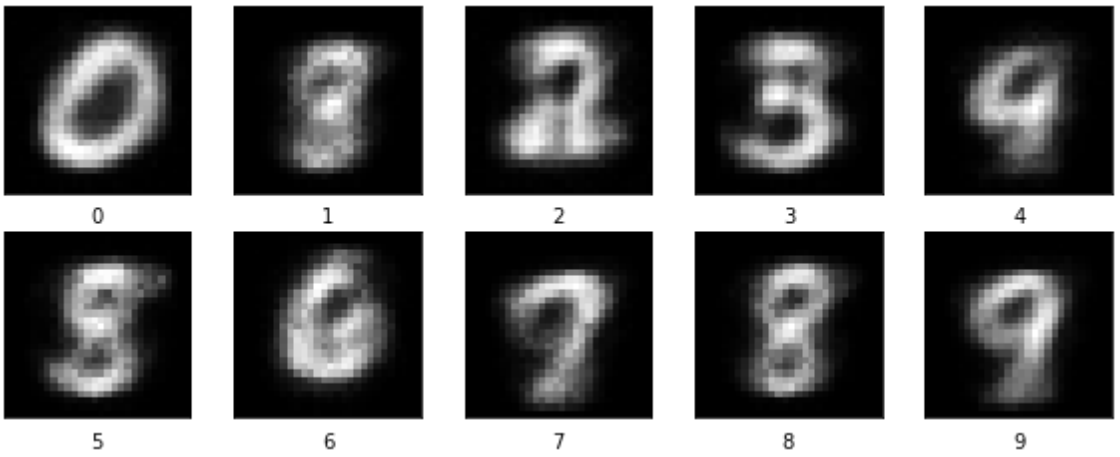
100%|

| 858/858 [00:10<00:00, 78.87it/s]



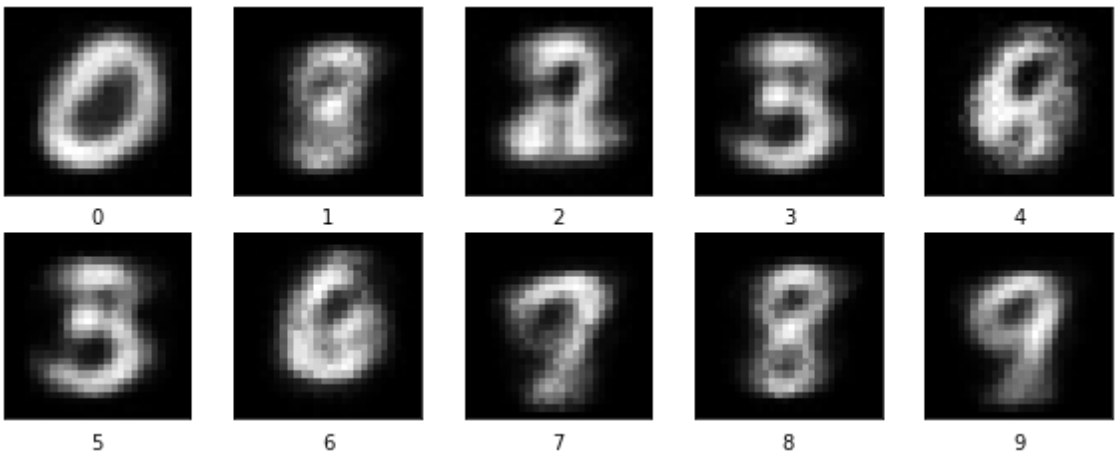
E: 84

100%| | 858/858 [00:11<00:00, 74.31it/s]



E: 85

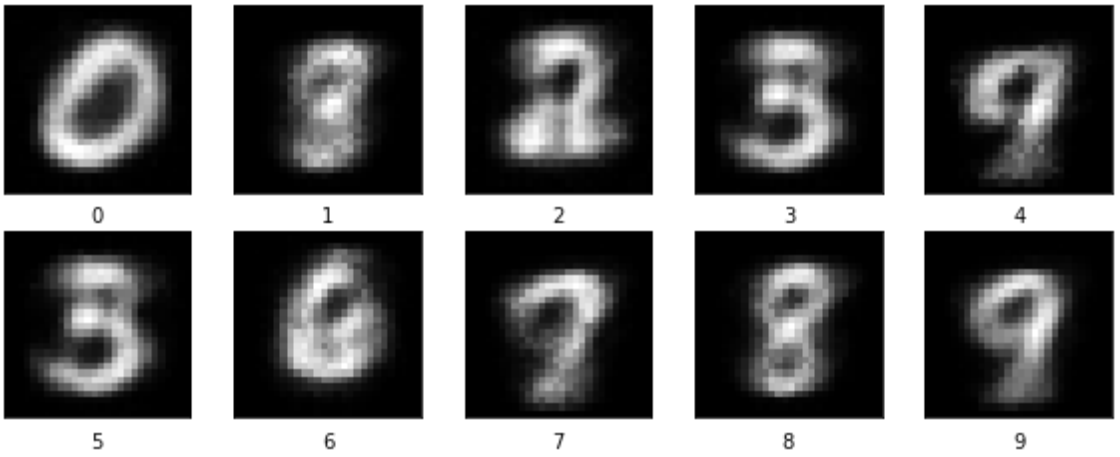
100%| | 858/858 [00:11<00:00, 74.31it/s]



E: 86

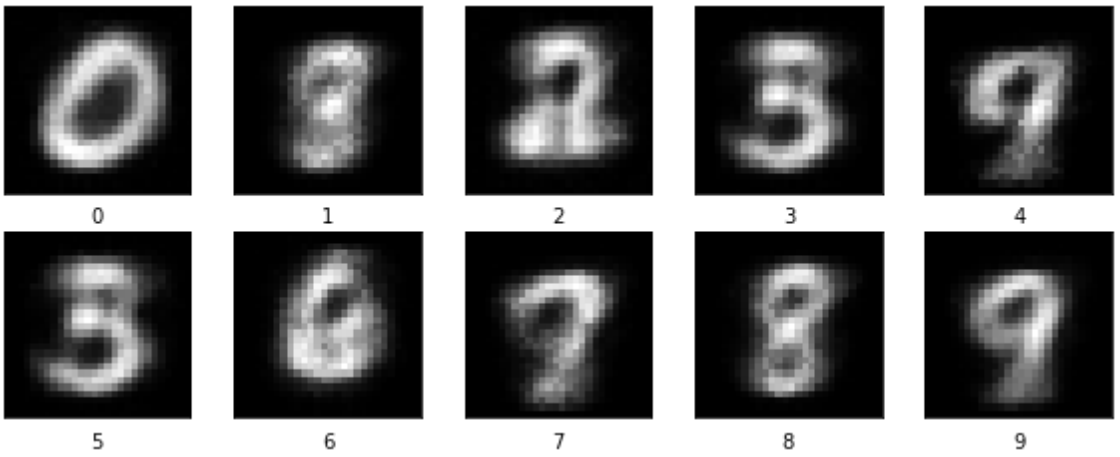
100%|

| 858/858 [00:11<00:00, 76.38it/s]



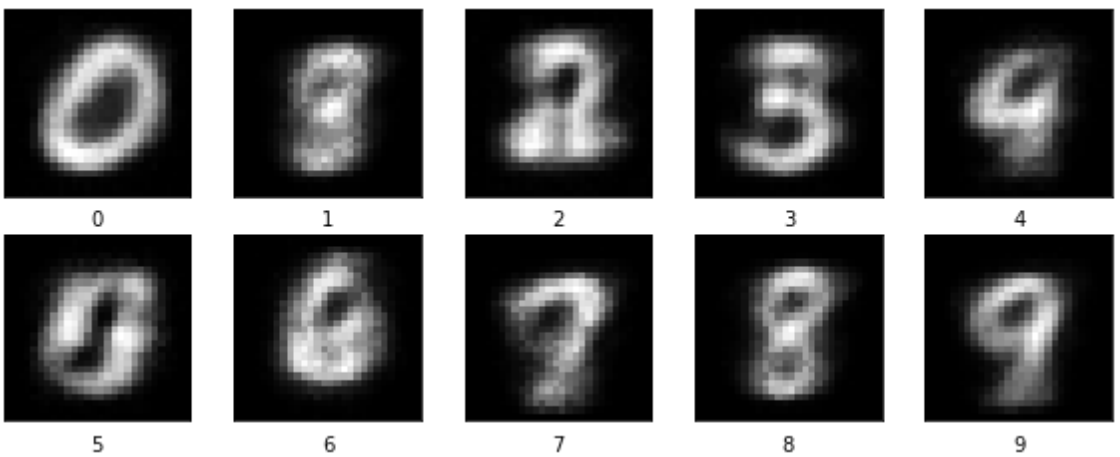
E: 87

100%| | 858/858 [00:11<00:00, 75.72it/s]



E: 88

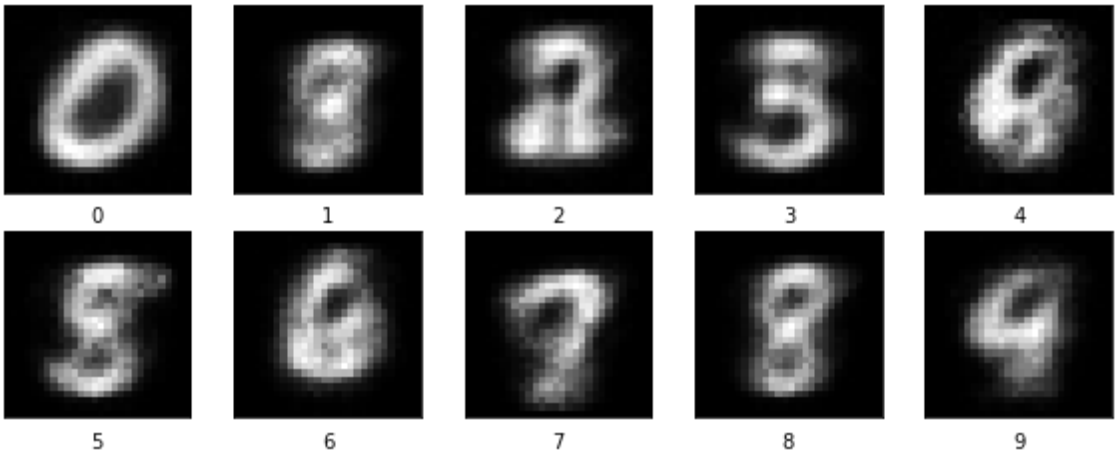
100%| | 858/858 [00:11<00:00, 73.04it/s]



E: 89

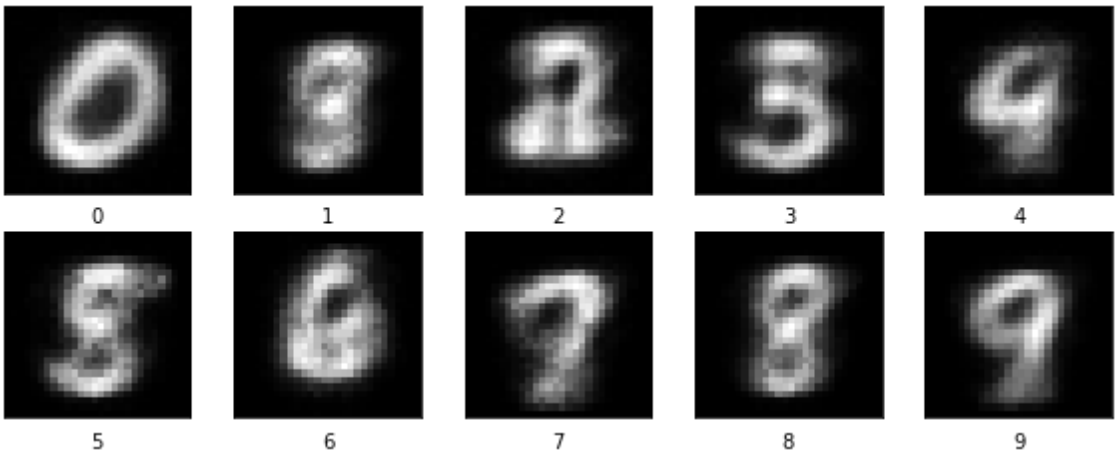
100%|

| 858/858 [00:10<00:00, 79.43it/s]



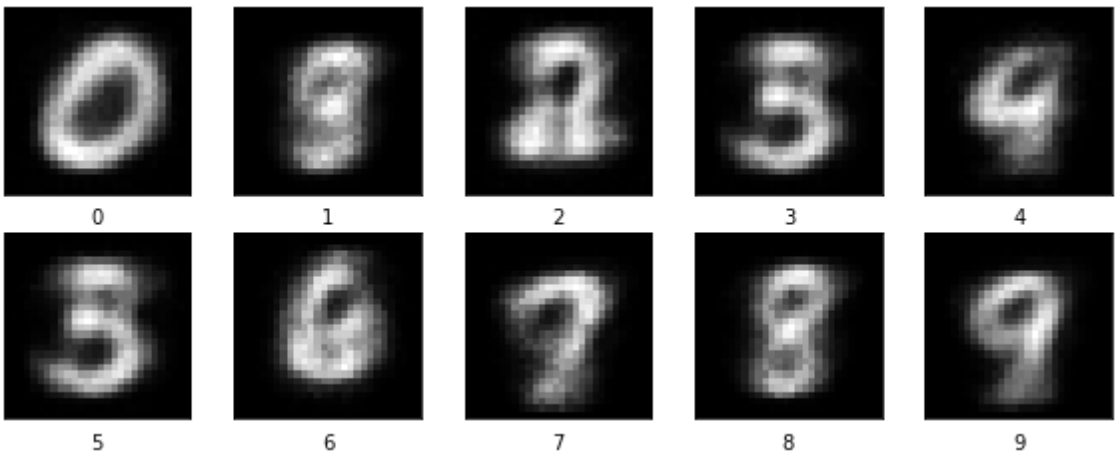
E: 90

100%| | 858/858 [00:10<00:00, 83.24it/s]



E: 91

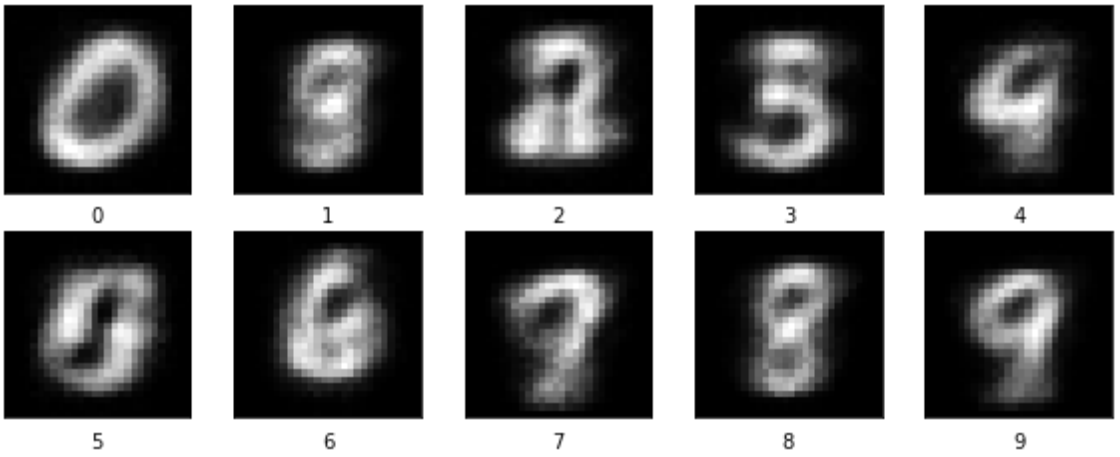
100%| | 858/858 [00:11<00:00, 71.90it/s]



E: 92

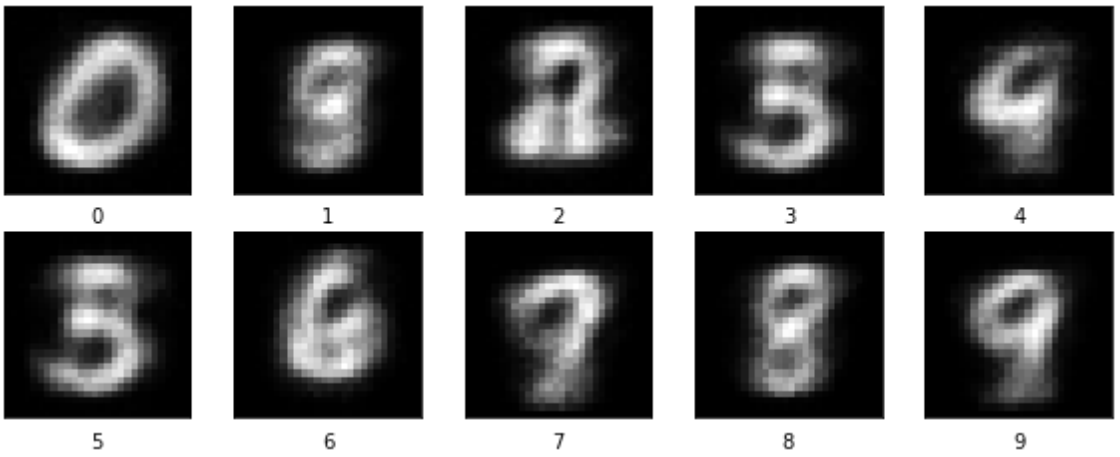
100%|

| 858/858 [00:10<00:00, 78.04it/s]



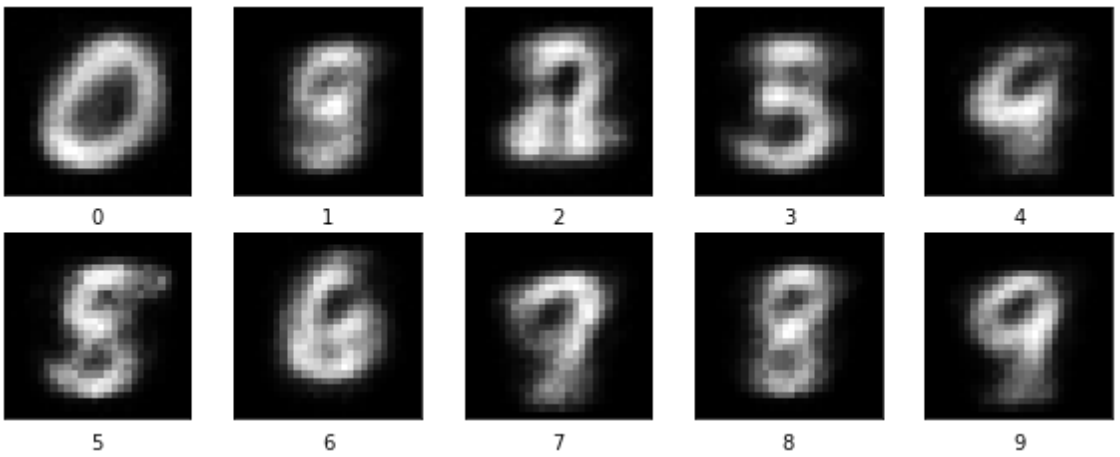
E: 93

100%| | 858/858 [00:13<00:00, 65.73it/s]



E: 94

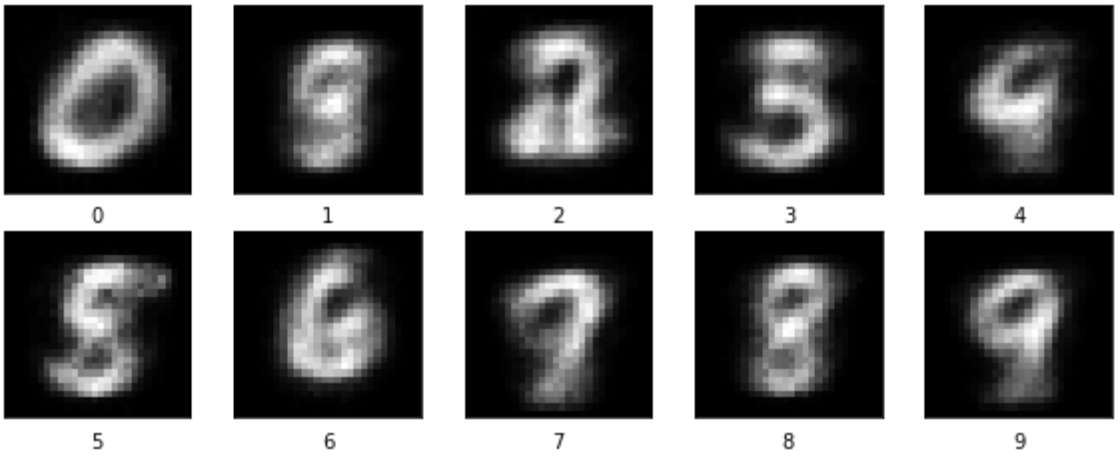
100%| | 858/858 [00:14<00:00, 58.14it/s]



E: 95

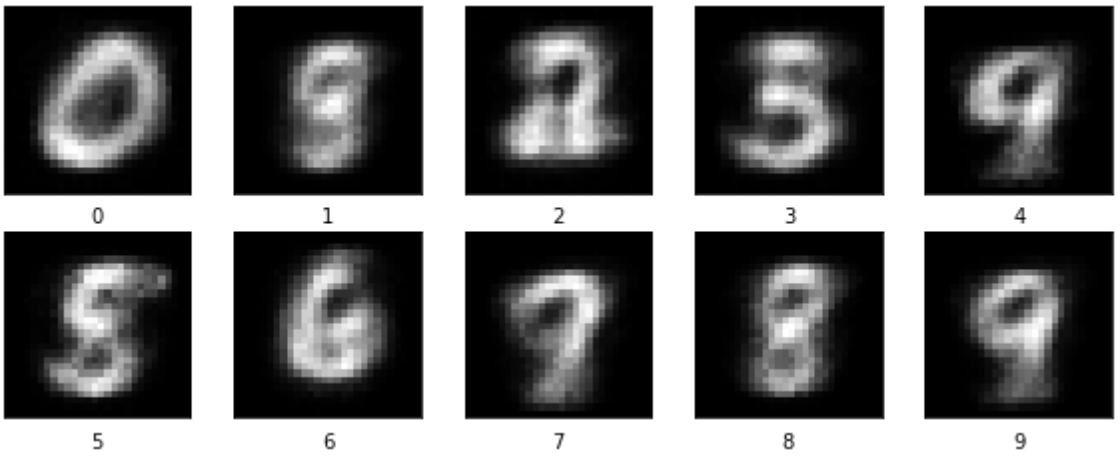
100%|

| 858/858 [00:14<00:00, 57.99it/s]



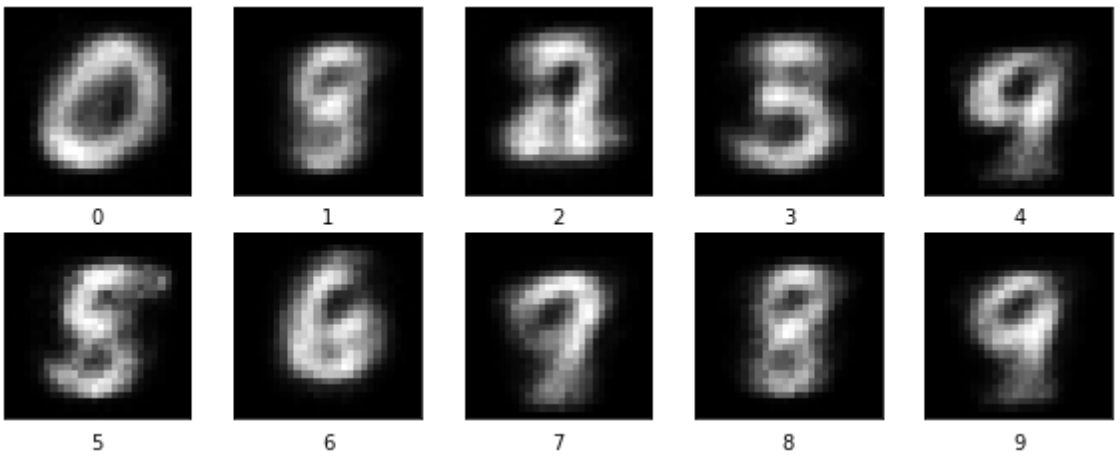
E: 96

100%| | 858/858 [00:10<00:00, 81.56it/s]



E: 97

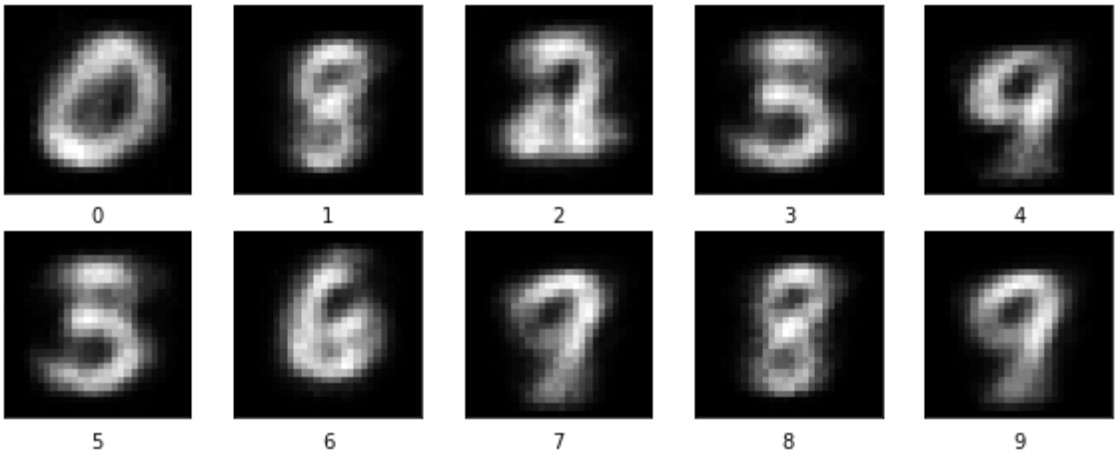
100%| | 858/858 [00:10<00:00, 78.76it/s]



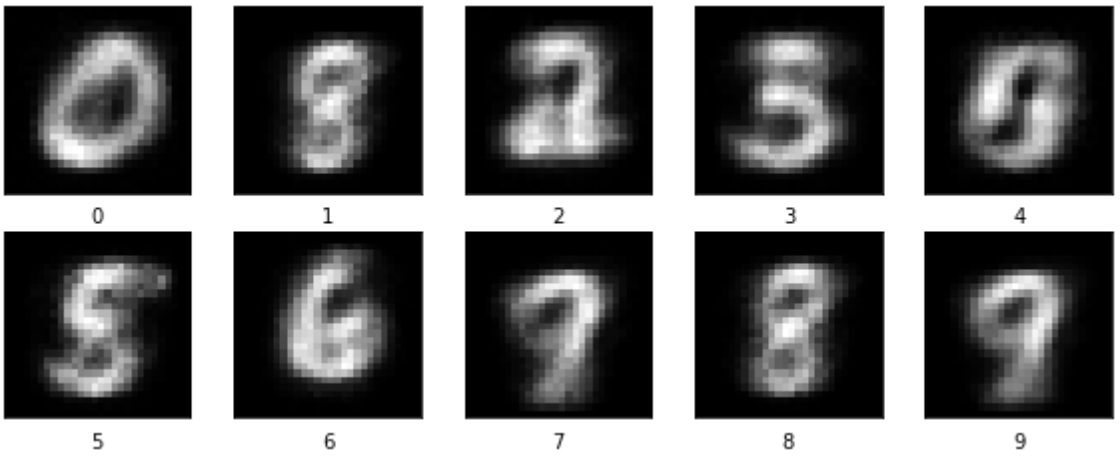
E: 98

100%|

```
| 858/858 [00:11<00:00, 74.26it/s]
```

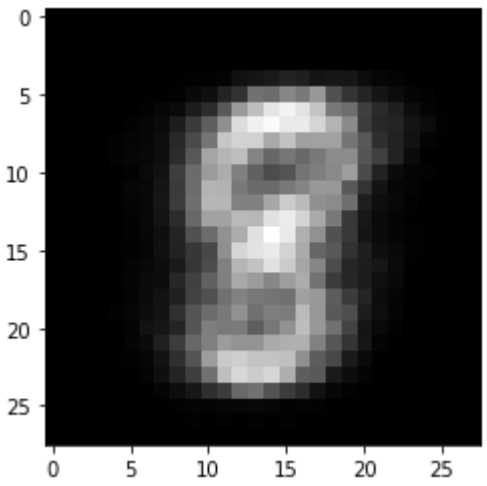



```
E: 99
100%| | 858/858 [00:11<00:00, 75.00it/s]
```



In [432]:
```
res = NN.predict([1])
plt.imshow(res.reshape(28, 28), cmap='gray')
```

Out[432]: <matplotlib.image.AxesImage at 0x7fb71ff178b0>



In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [325]:
```
1
```

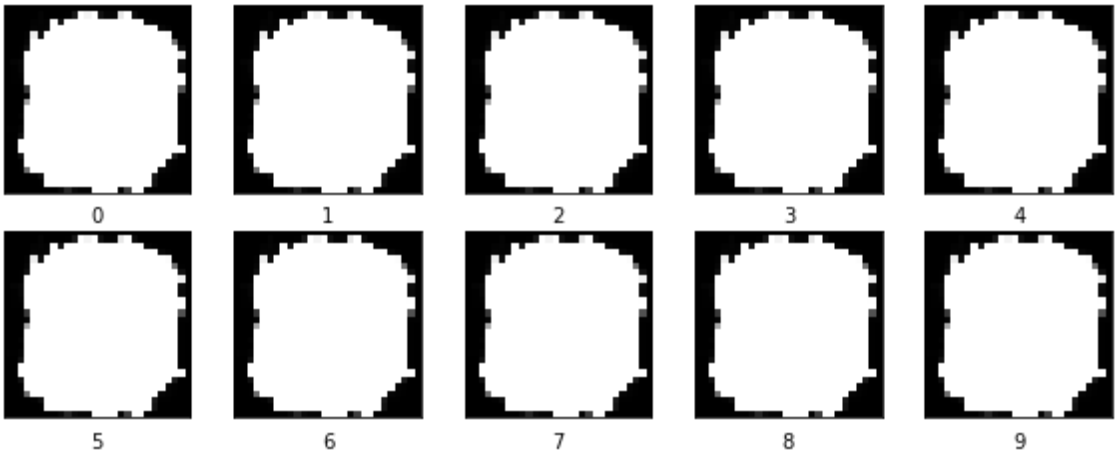
```
/tmp/ipykernel_5437/153577755.py:1: RuntimeWarning: invalid value e
ncountered in log
  np.log(-1)
```

Out[325]:
```
nan
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [57]:
```
1  plt.figure(figsize=(10,10))
2  for i in range(10):
3      plt.subplot(5,5,i+1)
4      plt.xticks([])
5      plt.yticks([])
6      plt.grid(False)
7      res = NN.predict([i])
8      plt.imshow(res.reshape(28, 28), cmap='gray')
9      plt.xlabel(i)
10 plt.show()
11
```



In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
class MyNN:
    def __init__(self):
        self.learn_rate = 0.01

        self._weights0 = np.random.rand(784, 49)*0.1 - 0.05
        self._weights1 = np.random.rand(49, 16)*0.1 - 0.05
        self._weights2 = np.random.rand(16, 10)*0.1 - 0.05

        self._input  = np.zeros([784])
        self._layer1 = np.zeros([49])
        self._layer2 = np.zeros([16])
        self._output = np.zeros([10])

        self._correction_0 = np.zeros([784, 49])
        self._correction_1 = np.zeros([49, 16])
        self._correction_2 = np.zeros([16, 10])

        self._activation0 = self.act_relu
        self._activation1 = self.act_relu
        self._activation2 = self.act_relu

        self._der_act_0 = self.der_act_relu
        self._der_act_1 = self.der_act_relu
        self._der_act_2 = self.der_act_relu

    def zero(self):
        self._input  = np.zeros([784])
        self._layer1 = np.zeros([49])
        self._layer2 = np.zeros([16])
        self._output = np.zeros([10])

    def act_sigmoid(self, layer):
        return np.array([1/(1+np.exp(-x)) for x in layer])

    def act_relu(self, layer):
        return np.array([x if x > 0 else 0 for x in layer])

    def act_softplus(self, layer):
        return np.array([np.ln(1+np.exp(x)) for x in layer])

    def der_act_sigmoid(self, l):
        _exp = np.exp(-l)
        return _exp / (1+_exp)**2

    def der_act_relu(self, l):
        if l > 0:
            return 1
        else:
            return 0

```

```
    def def_act_softplus(self, l):
        return 1 / (1+np.exp(-l))

    def predict(self, img):
        self.zero()
        self._input = img

        self._layer1 = np.dot(self._input, self._weights0)
        self._layer1 = self._activation0(self._layer1)

        self._layer2 = np.dot(self._layer1, self._weights1)
        self._layer2 = self._activation1(self._layer2)

        self._output = np.dot(self._layer2, self._weights2)
        self._output = self._activation2(self._output)

        return self._output

    def learn(self, labels):
        main_delta   = self._output - labels
        delta_output = main_delta * [self._der_act_2(x) for x in
        delta_w_2    = np.dot(delta_output.reshape(len(delta_out
                               self._layer2.reshape(1, len(self._

        delta_layer2 = np.dot(self._weights2, delta_output) * [s
        delta_w_1    = np.dot(delta_layer2.reshape(len(delta_lay
                               self._layer1.reshape(1, len(self._

        delta_layer1 = np.dot(self._weights1, delta_layer2) * [s
        delta_w_0    = np.dot(delta_layer1.reshape(len(delta_lay
                               self._input.reshape(1, len(self._i

        self._correction_0 += np.transpose(delta_w_0)
        self._correction_1 += np.transpose(delta_w_1)
        self._correction_2 += np.transpose(delta_w_2)

    def update(self):
        self._weights0 = self._weights0 - self.learn_rate * self
        self._weights1 = self._weights1 - self.learn_rate * self
        self._weights2 = self._weights2 - self.learn_rate * self

        self._correction_0 = np.zeros([784, 49])
        self._correction_1 = np.zeros([49, 16])
        self._correction_2 = np.zeros([16, 10])

    def check_correct(self, test):
        correct = 0
        for i in range(len(test)):
            res = np.argmax(self.predict(test.values[i][1:]))
            if res == test.values[i][:1]:
                correct+=1
        return correct/len(test)

    def fit(self, test, train, epochs, learn_rate, batch):
        self.learn_rate = learn_rate
        for e in range(epochs):
            for i in range(len(train)):
                NN.predict( train.values[i][1:] )
                expect = [1 if k == train.values[i][:1] else 0 f
                NN.learn(expect)
```

```
111                 if not i %batch:
112                     NN.update()
113             print("e{}, corr = {}".format(e, NN.check_correct(te
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```