Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №5
по дисциплине
«Методы машинного обучения»
на тему
«Вариацонный энкодер»

Выполнил:
студент группы ИУ5-24М
Зубаиров В. А.

Москва — 2020 г.

1. Задание на лабораторную работу

1.1. 1. Создать вариационный автоэнкодер с использованием сверток (Conv2d) в энкодере (слои отвечающие за среднее и отклонение остаются полносвязными), и с развертками (Conv2dTranspose) в декодере. Размерность скрытого вектора равна двум

1.2. 2. Создать сетку из 25 изображений, где по оси X изменяется значение первого элемента z, а по оси Y - второго элемента z

1.2.1. 1) Иморт необходимых библиотек

[15]:
```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:80% !important; }</style>"))
!pip3 install -q imageio
import PIL
import imageio
```

<IPython.core.display.HTML object>

[2]:
```
import numpy as np
np.set_printoptions(linewidth=110)
```

[3]:
```
from packaging import version
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import datetime as dt
import time

print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >= 2, \
    "This notebook requires TensorFlow 2.0 or above."
```

TensorFlow version: 2.1.0

1.2.2. 2) Подготовка датасета МНИСТ

- разделение на тестовую и обучающую выборку
- нормализация изображений
- нарезка на части

[4]:
```
(train_images, _), (test_images, _) = tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1).astype('float32')

# Normalizing the images to the range of [0., 1.]
```

```
train_images /= 255.
test_images /= 255.

# Binarization
train_images[train_images >= .5] = 1.
train_images[train_images < .5] = 0.
test_images[test_images >= .5] = 1.
test_images[test_images < .5] = 0.

TRAIN_BUF = 60000
BATCH_SIZE = 32

TEST_BUF = 10000

train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(TRAIN_BUF).
  ↪batch(BATCH_SIZE)
test_dataset = tf.data.Dataset.from_tensor_slices(test_images).shuffle(TEST_BUF).
  ↪batch(BATCH_SIZE)
```

1.2.3. Код энкодера-декодера

[5]:
```
class CVAE(tf.keras.Model):
    def __init__(self, latent_dim):
        super(CVAE, self).__init__()
        self.latent_dim = latent_dim
        self.inference_net = tf.keras.Sequential(
            [
                tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),
                tf.keras.layers.Conv2D(filters=32, kernel_size=3, strides=(2, 2), activation='relu'),
                tf.keras.layers.Conv2D(
                filters=64, kernel_size=3, strides=(2, 2), activation='relu'),
                tf.keras.layers.Flatten(),
                  # No activation
                tf.keras.layers.Dense(latent_dim + latent_dim),
            ]
        )

        self.generative_net = tf.keras.Sequential(
            [
              tf.keras.layers.InputLayer(input_shape=(latent_dim,)),
              tf.keras.layers.Dense(units=7*7*32, activation=tf.nn.relu),
              tf.keras.layers.Reshape(target_shape=(7, 7, 32)),
              tf.keras.layers.Conv2DTranspose(filters=64, kernel_size=3, strides=(2, 2),
  ↪padding="SAME", activation='relu'),
              tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=3, strides=(2, 2),
  ↪padding="SAME", activation='relu'),
          # No activation
              tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=3, strides=(1, 1),
  ↪padding="SAME"),
```

```
        ]
    )

  @tf.function
  def sample(self, eps=None):
    if eps is None:
      eps = tf.random.normal(shape=(100, self.latent_dim))
    return self.decode(eps, apply_sigmoid=True)
  def image_grid(self, z):
    return self.decode(eps)

  def encode(self, x):
    mean, logvar = tf.split(self.inference_net(x), num_or_size_splits=2, axis=1)
    return mean, logvar

  def reparameterize(self, mean, logvar):
    eps = tf.random.normal(shape=mean.shape)
    return eps * tf.exp(logvar * .5) + mean

  def decode(self, z, apply_sigmoid=False):
    logits = self.generative_net(z)
    if apply_sigmoid:
      probs = tf.sigmoid(logits)
      return probs

    return logits
```

1.2.4. Вычисление и применение градиентов

[6]:
```
optimizer = tf.keras.optimizers.Adam(1e-4)

def log_normal_pdf(sample, mean, logvar, raxis=1):
  log2pi = tf.math.log(2. * np.pi)
  return tf.reduce_sum(-.5 * ((sample - mean) ** 2. * tf.exp(-logvar) + logvar + log2pi),
 ↪axis=raxis)

@tf.function
def compute_loss(model, x):
  mean, logvar = model.encode(x)
  z = model.reparameterize(mean, logvar)
  x_logit = model.decode(z)

  cross_ent = tf.nn.sigmoid_cross_entropy_with_logits(logits=x_logit, labels=x)
  logpx_z = -tf.reduce_sum(cross_ent, axis=[1, 2, 3])
  logpz = log_normal_pdf(z, 0., 0.)
  logqz_x = log_normal_pdf(z, mean, logvar)
  return -tf.reduce_mean(logpx_z + logpz - logqz_x)

@tf.function
```

```
def compute_apply_gradients(model, x, optimizer):
    with tf.GradientTape() as tape:
        loss = compute_loss(model, x)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

1.2.5. Установка количества эпох, измерения, количества необходимых примеров

[16]:
```
epochs = 40
latent_dim = 2
num_examples_to_generate = 16

#начальный случайный вектор для генерации
random_vector_for_generation = tf.random.normal(shape=[num_examples_to_generate,
    latent_dim])
model = CVAE(latent_dim)
```

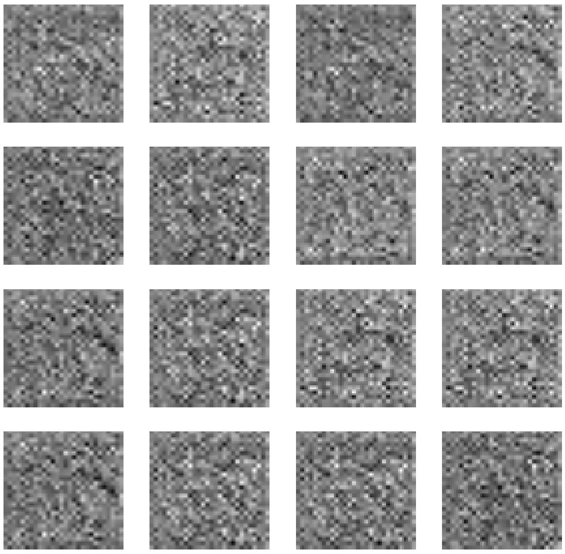
1.2.6. Функция для сохранения и вывода 16 изображений в каждой эпохе

[8]:
```
def generate_and_save_images(model, epoch, test_input):
    predictions = model.sample(test_input)
    fig = plt.figure(figsize=(5,5))

    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow(predictions[i, :, :, 0], cmap='gray')
        plt.axis('off')

    # tight_layout minimizes the overlap between 2 sub-plots
    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
    plt.show()
```

1.2.7. Обучение модели

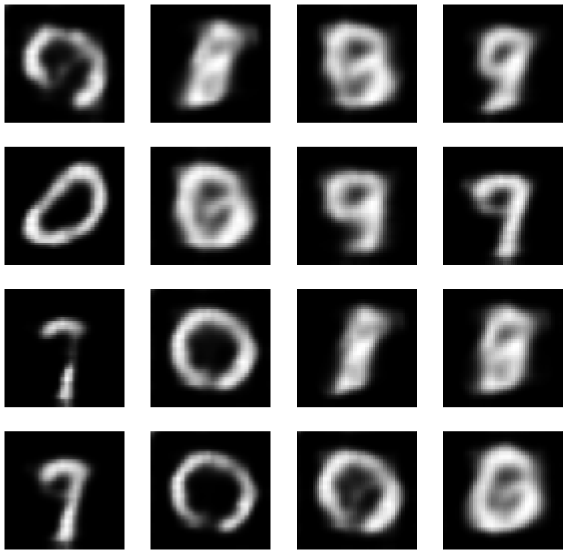
[9]:
```
generate_and_save_images(model, 0, random_vector_for_generation)

for epoch in range(1, epochs + 1):
    start_time = time.time()
    for train_x in train_dataset:
        compute_apply_gradients(model, train_x, optimizer)
    end_time = time.time()

    if epoch % 1 == 0:
        loss = tf.keras.metrics.Mean()
        for test_x in test_dataset:
            loss(compute_loss(model, test_x))
        elbo = -loss.result()
        print('Epoch: {}, Test set ELBO: {}, '
```

```
        'time elapse for current epoch {}'.format(epoch,
                                                  elbo,
                                                  end_time - start_time))
    generate_and_save_images(model, epoch, random_vector_for_generation)
```

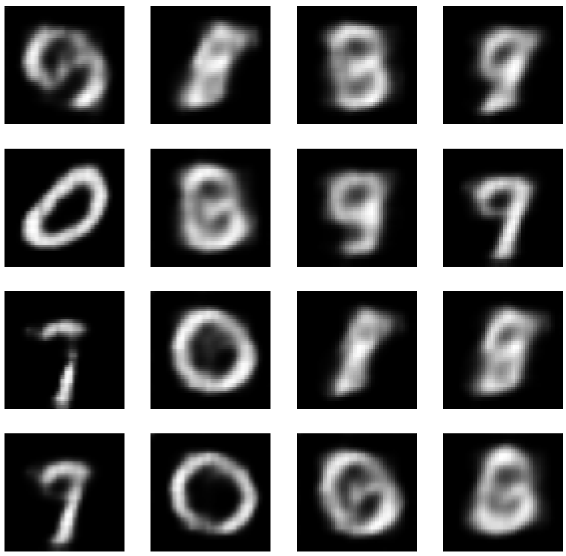


Epoch: 1, Test set ELBO: -178.18820190429688, time elapse for current epoch 35.0986590385437

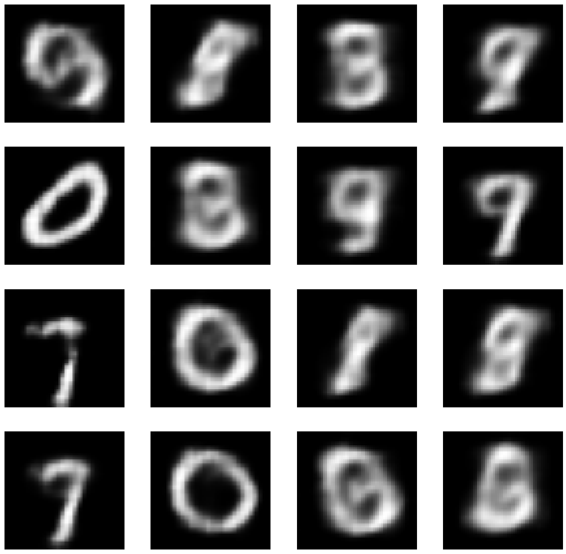
Epoch: 2, Test set ELBO: -170.4047393798828, time elapse for current epoch 34.096349000930786



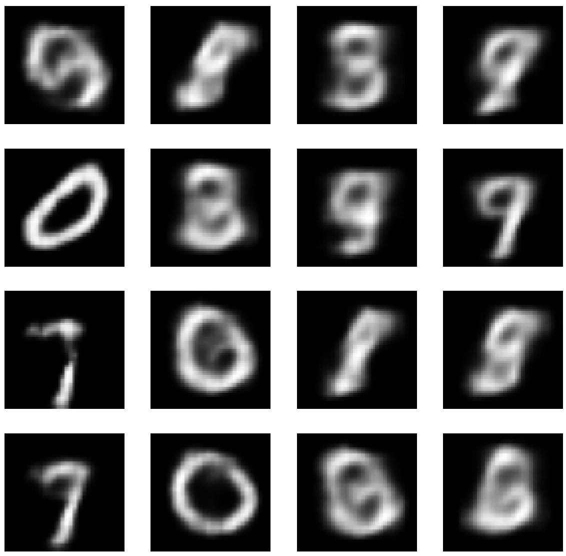
Epoch: 3, Test set ELBO: -166.2039794921875, time elapse for current epoch 37.713661193847656



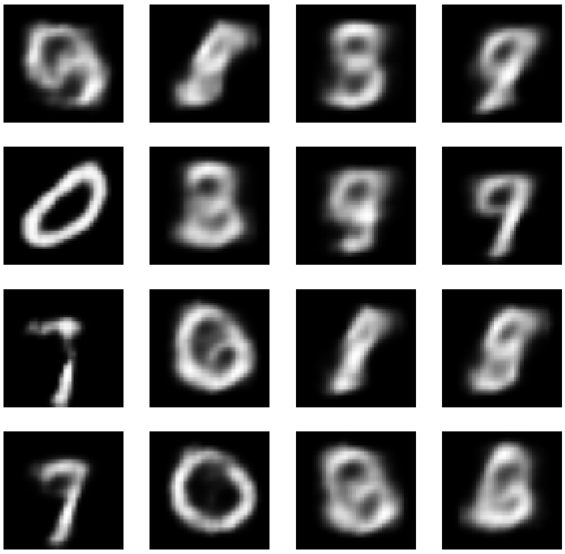
Epoch: 4, Test set ELBO: -163.1304168701172, time elapse for current epoch 37.33688712120056



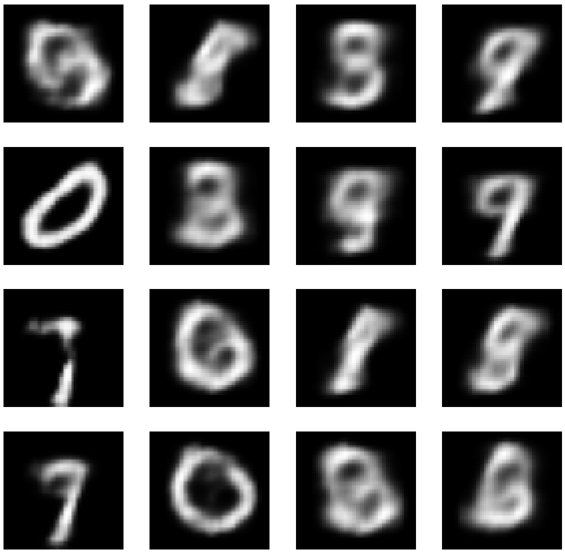
Epoch: 5, Test set ELBO: -161.2280731201172, time elapse for current epoch 37.575697898864746



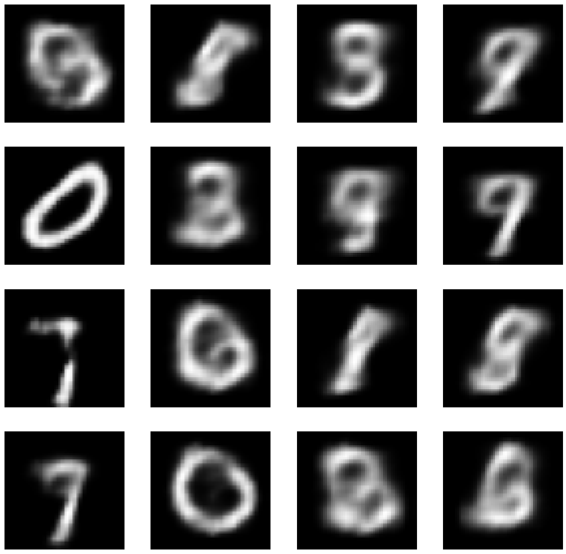
Epoch: 6, Test set ELBO: -159.64608764648438, time elapse for current epoch 39.950963258743286



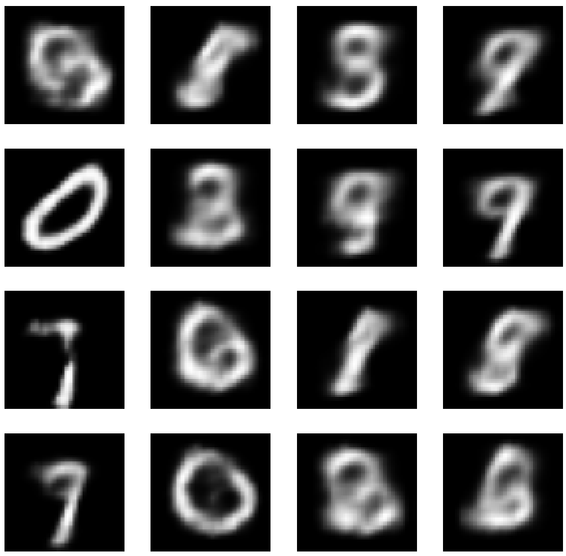
Epoch: 7, Test set ELBO: -158.7156219482422, time elapse for current epoch 38.471580266952515



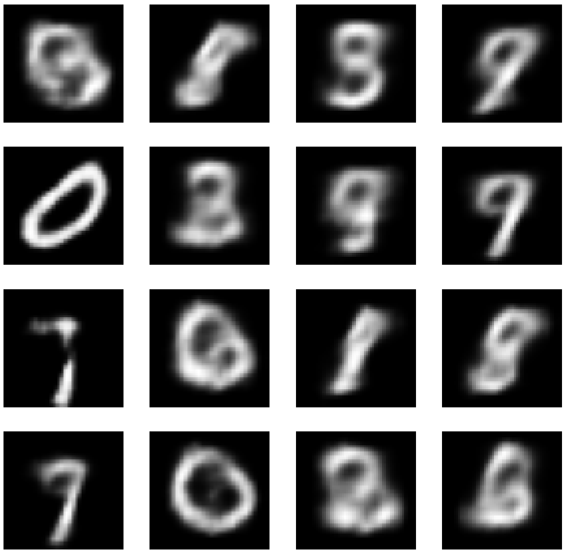
Epoch: 8, Test set ELBO: -158.0225830078125, time elapse for current epoch 38.16924500465393



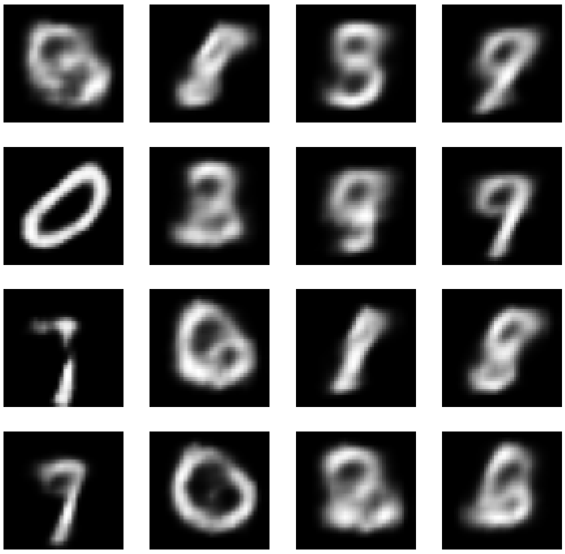
Epoch: 9, Test set ELBO: -157.2595672607422, time elapse for current epoch 39.61945199966431



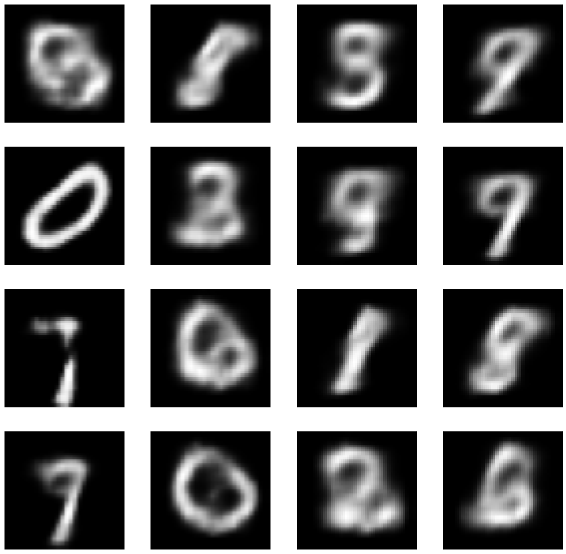
Epoch: 10, Test set ELBO: -156.91648864746094, time elapse for current epoch 38.59878873825073



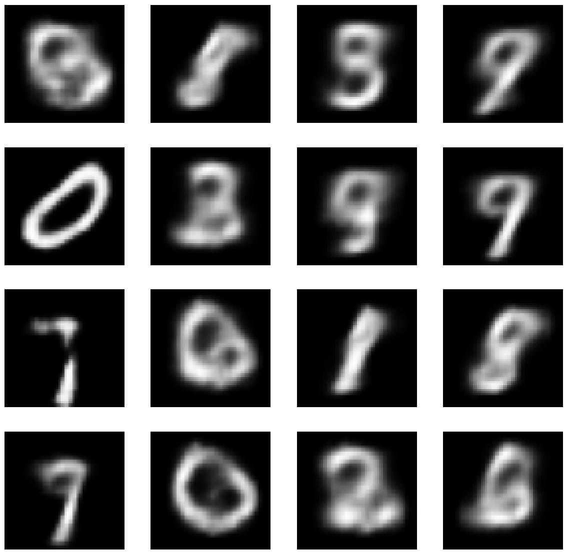
Epoch: 11, Test set ELBO: -156.31771850585938, time elapse for current epoch 40.30605912208557



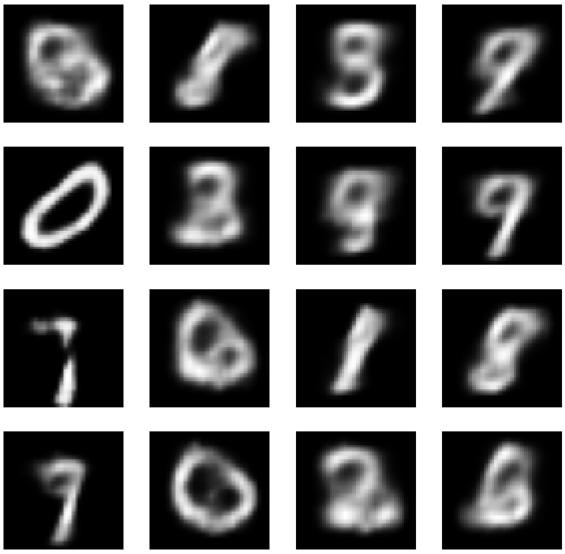
Epoch: 12, Test set ELBO: -155.97549438476562, time elapse for current epoch 39.399182081222534



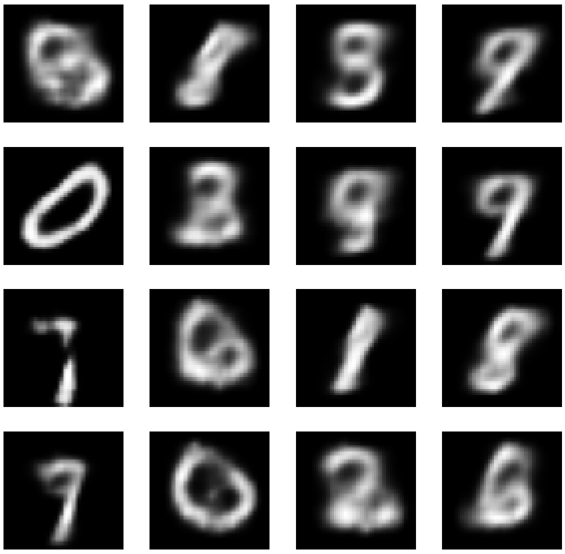
Epoch: 13, Test set ELBO: -155.43975830078125, time elapse for current epoch 39.15373420715332



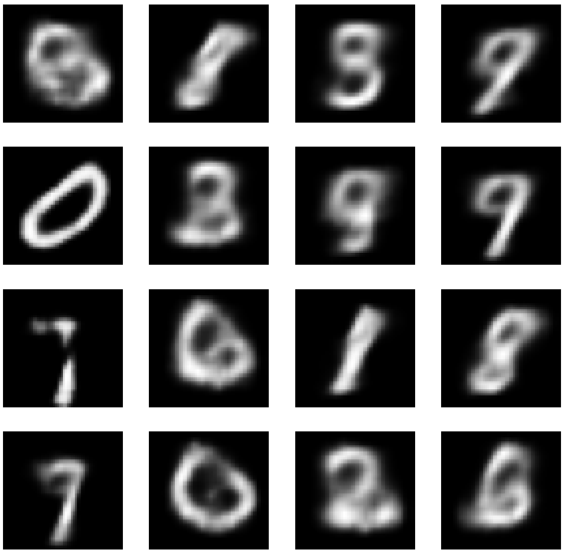
Epoch: 14, Test set ELBO: -154.96592712402344, time elapse for current epoch 39.7727370262146



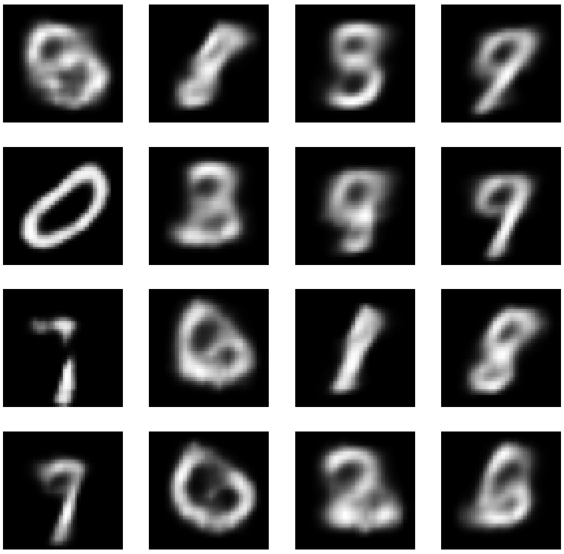
Epoch: 15, Test set ELBO: -154.9392547607422, time elapse for current epoch 40.330471992492676



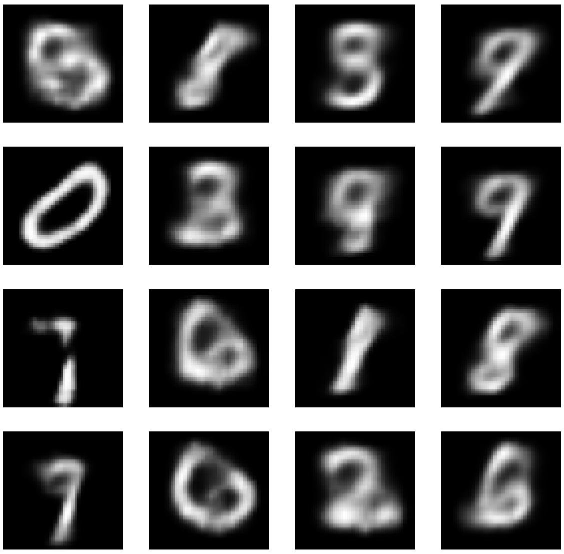
Epoch: 16, Test set ELBO: -154.94970703125, time elapse for current epoch 40.96620011329651



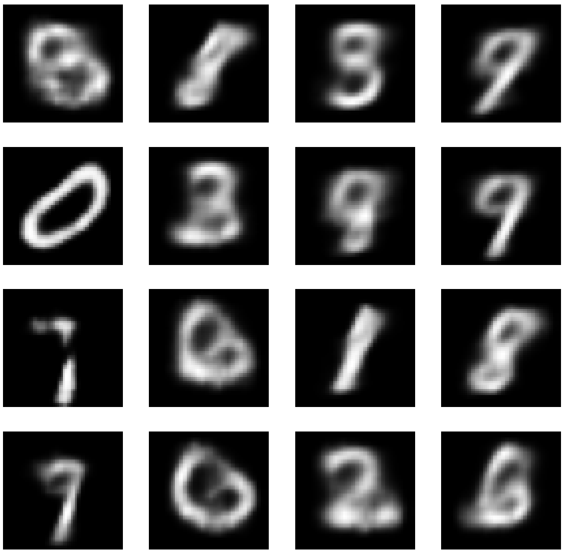
Epoch: 17, Test set ELBO: -154.17176818847656, time elapse for current epoch 41.17062187194824



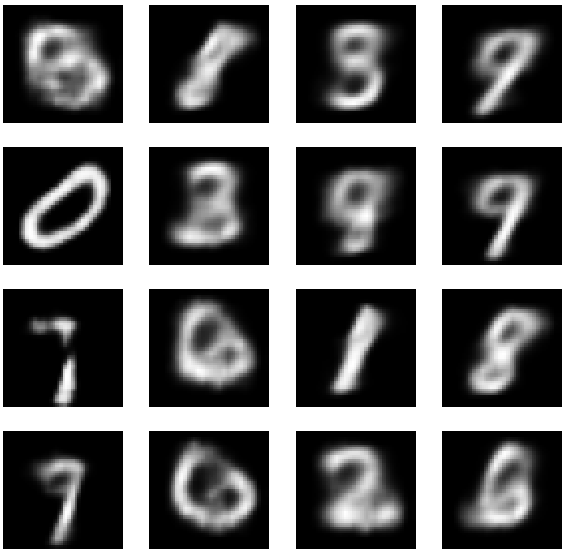
Epoch: 18, Test set ELBO: -154.01446533203125, time elapse for current epoch 41.020140171051025



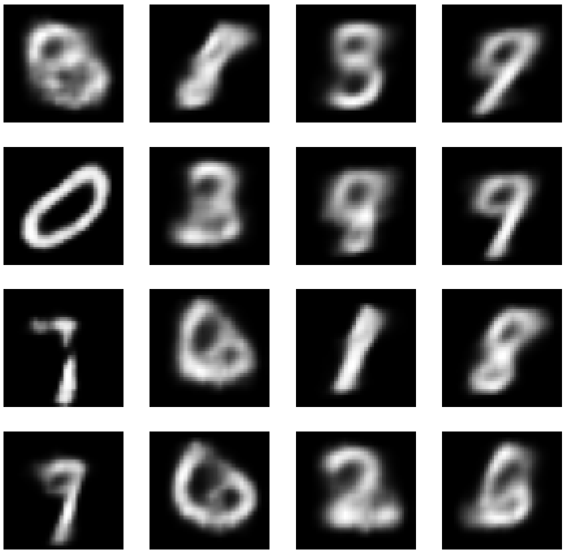
Epoch: 19, Test set ELBO: -154.122314453125, time elapse for current epoch 41.73163890838623



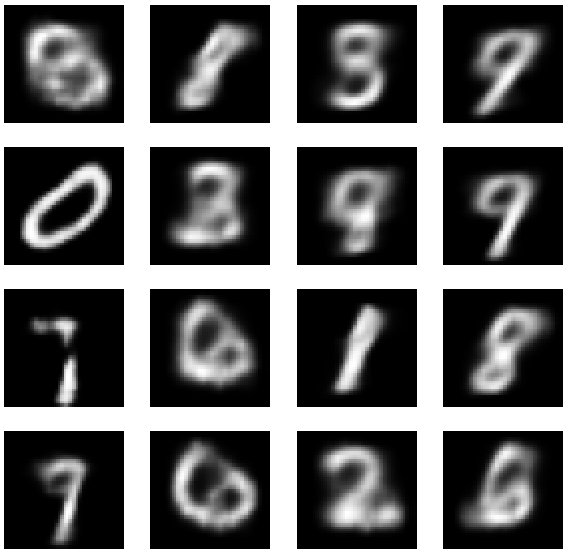
Epoch: 20, Test set ELBO: -153.5556182861328, time elapse for current epoch 41.888553857803345



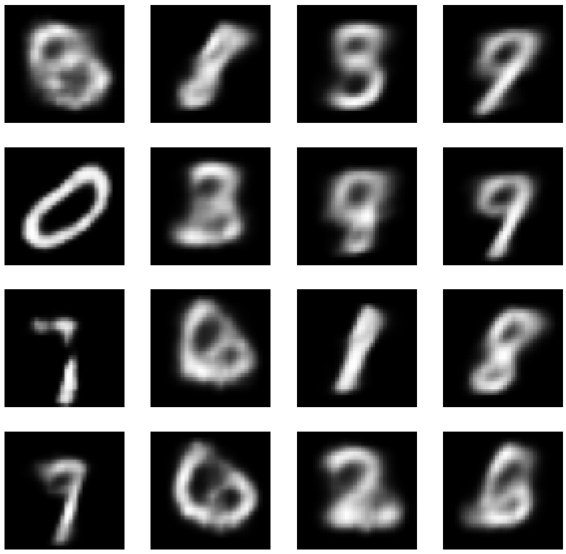
Epoch: 21, Test set ELBO: -153.3916778564453, time elapse for current epoch 42.22235894203186



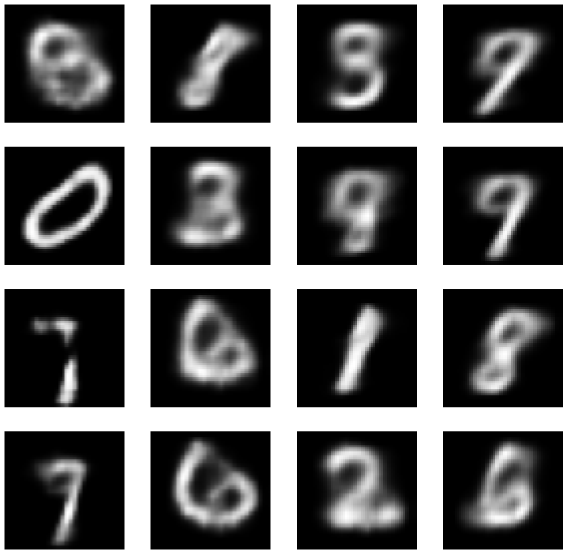
Epoch: 22, Test set ELBO: -153.05711364746094, time elapse for current epoch 44.0643572807312



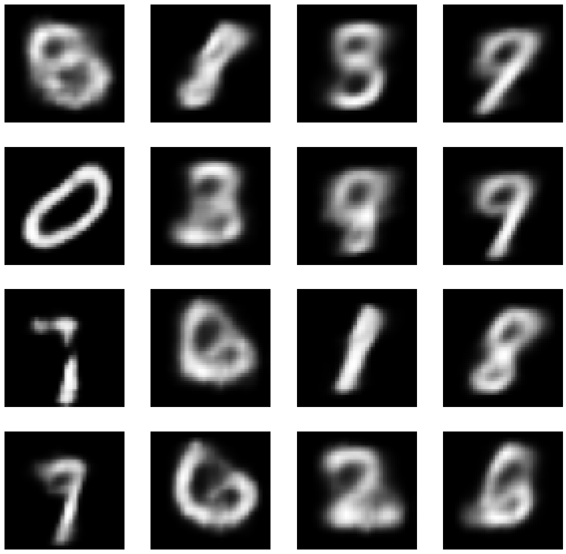
Epoch: 23, Test set ELBO: -152.83651733398438, time elapse for current epoch 42.495583295822144



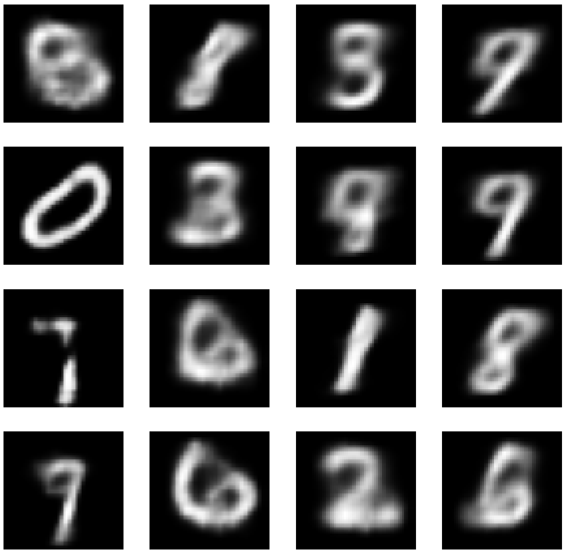
Epoch: 24, Test set ELBO: -153.29937744140625, time elapse for current epoch 41.68933820724487



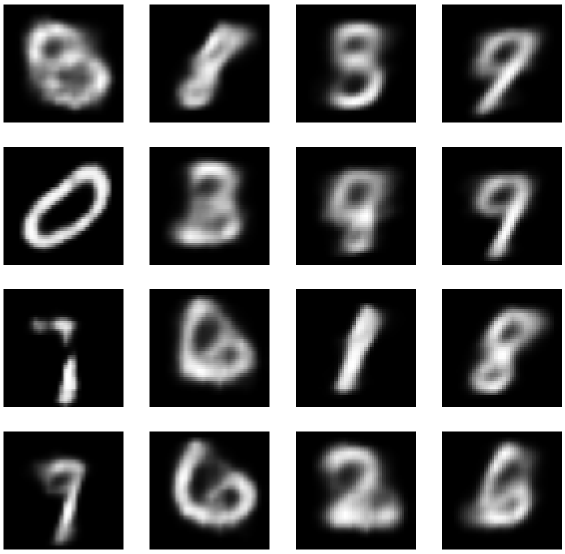
Epoch: 25, Test set ELBO: -152.5604248046875, time elapse for current epoch 44.11400103569031



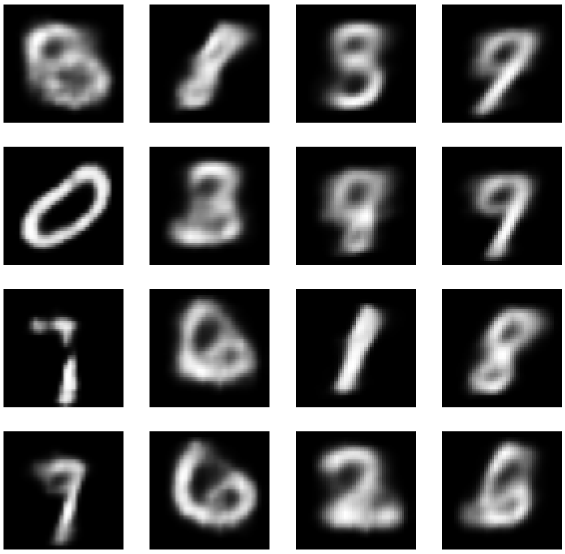
Epoch: 26, Test set ELBO: -152.51963806152344, time elapse for current epoch 44.22315788269043



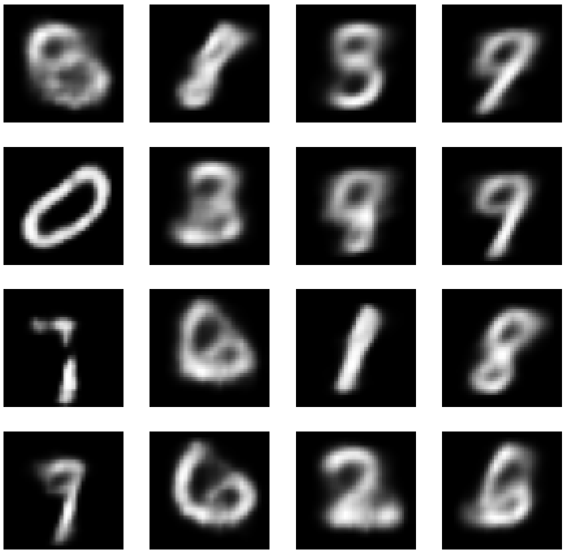
Epoch: 27, Test set ELBO: -152.21954345703125, time elapse for current epoch 43.319597005844116



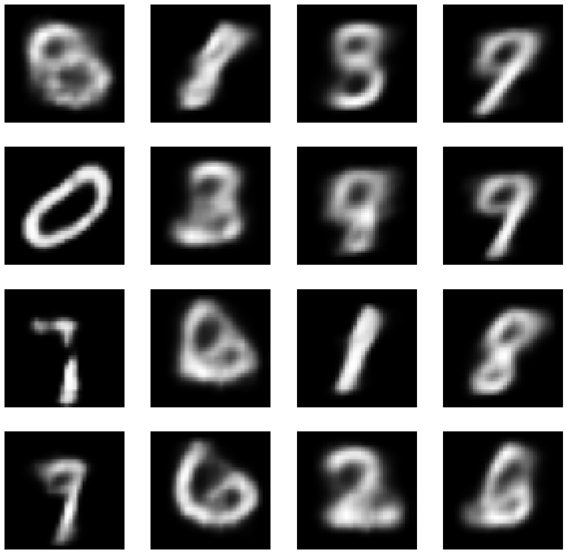
Epoch: 28, Test set ELBO: -152.51275634765625, time elapse for current epoch 43.853729009628296



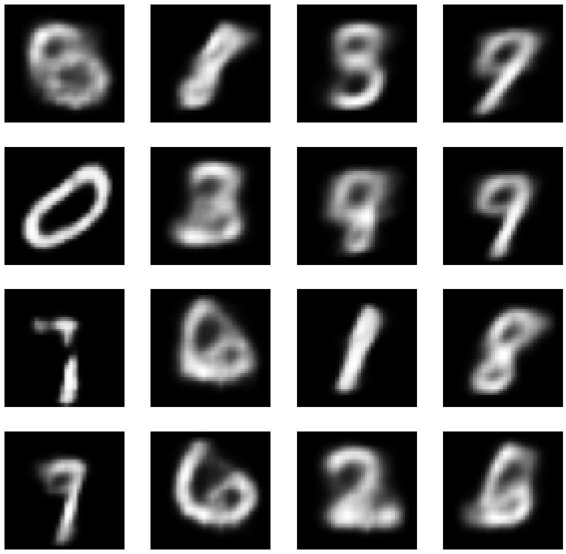
Epoch: 29, Test set ELBO: -152.14027404785156, time elapse for current epoch 43.36322617530823



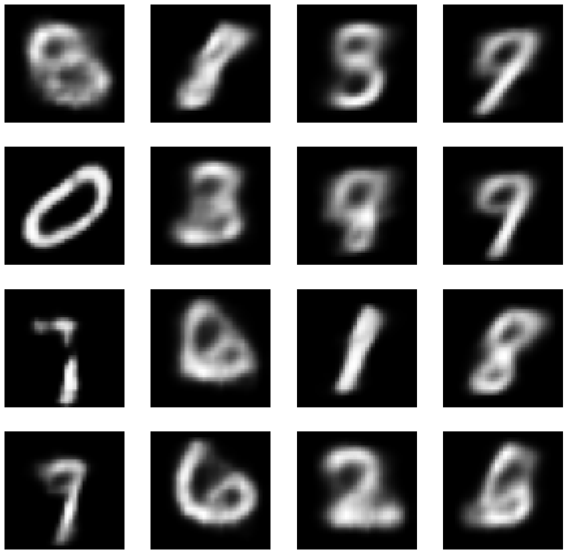
Epoch: 30, Test set ELBO: -151.83157348632812, time elapse for current epoch 44.08106207847595



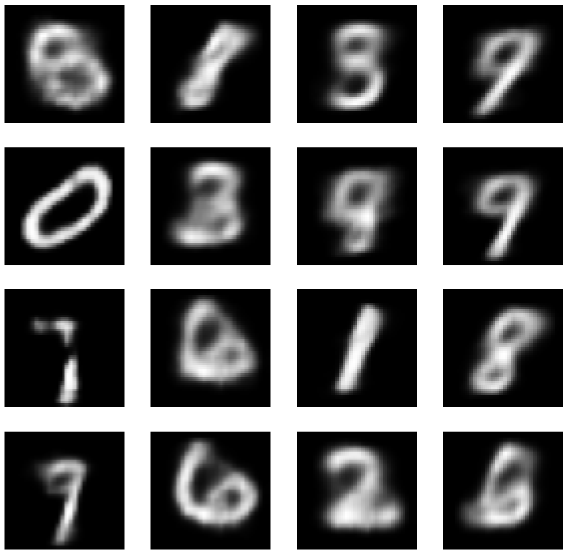
Epoch: 31, Test set ELBO: -151.69577026367188, time elapse for current epoch 41.45243978500366



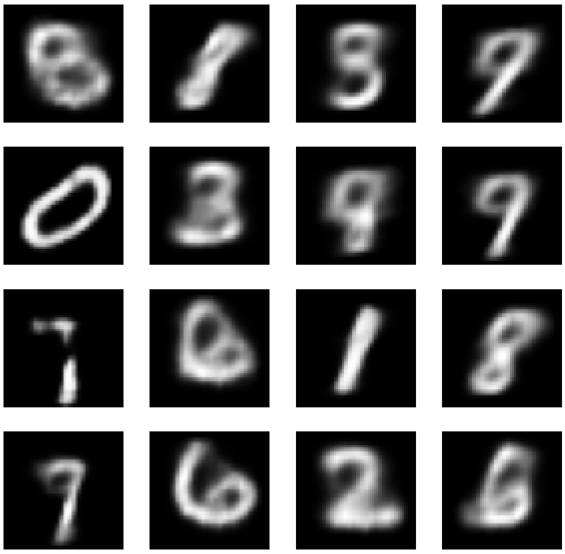
Epoch: 32, Test set ELBO: -152.06387329101562, time elapse for current epoch 43.3482940196991



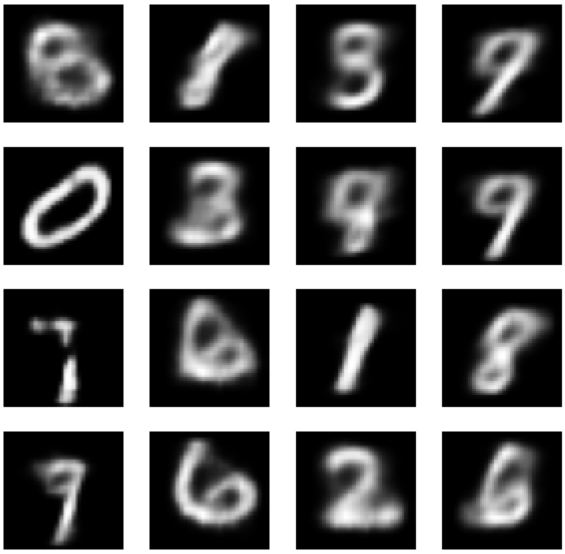
Epoch: 33, Test set ELBO: -151.56871032714844, time elapse for current epoch 42.48015904426575



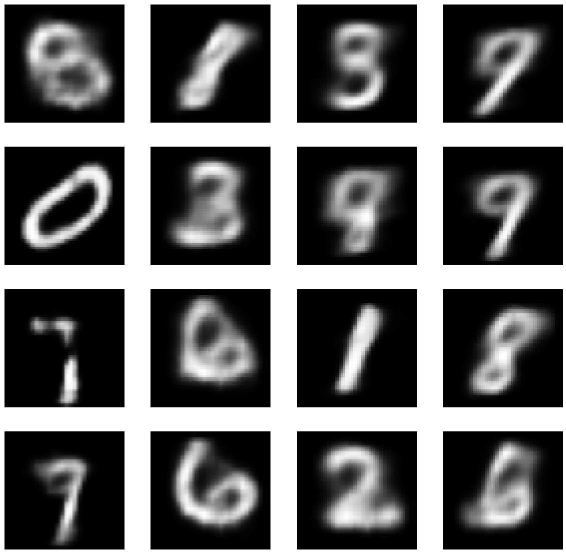
Epoch: 34, Test set ELBO: -151.5661163330078, time elapse for current epoch 44.75703501701355



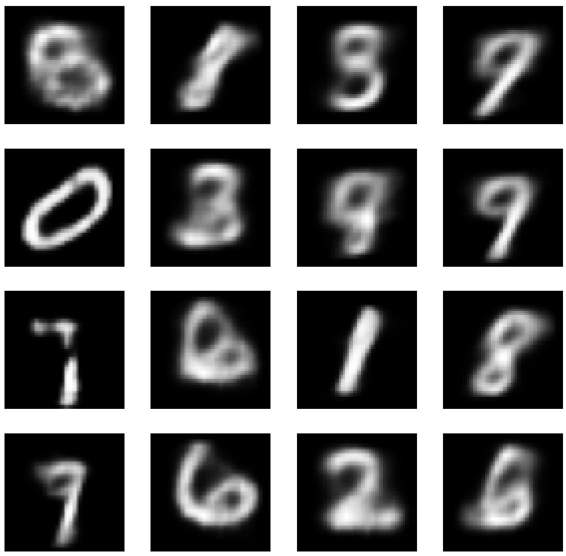
Epoch: 35, Test set ELBO: -151.44589233398438, time elapse for current epoch 43.75524377822876



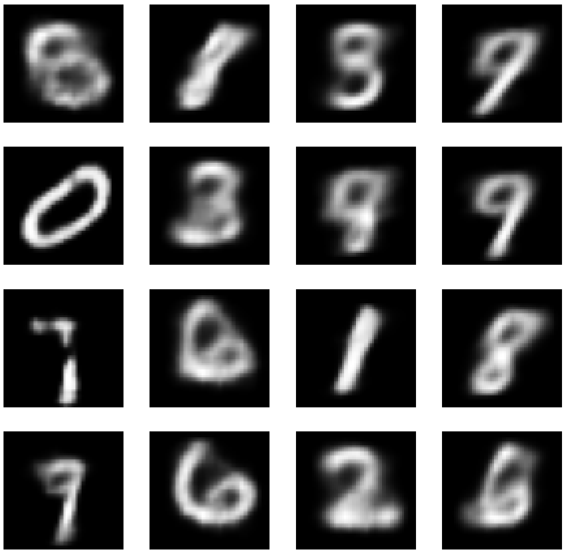
Epoch: 36, Test set ELBO: -151.24319458007812, time elapse for current epoch 47.255741119384766



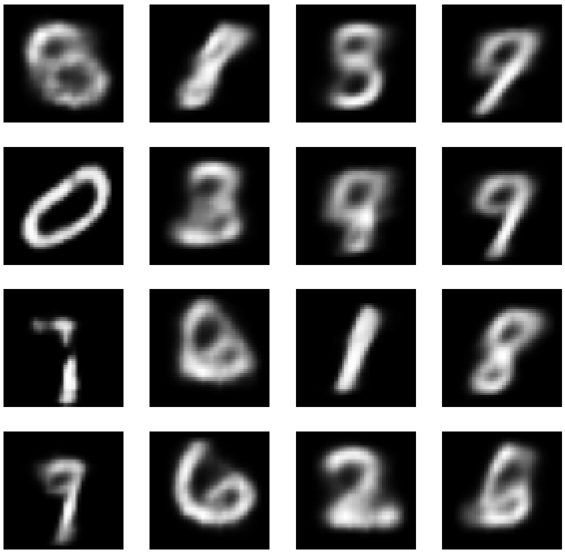
Epoch: 37, Test set ELBO: -150.97320556640625, time elapse for current epoch 46.38107085227966



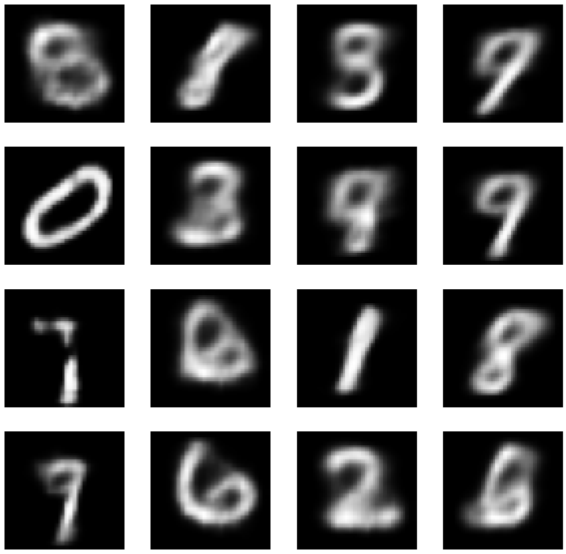
Epoch: 38, Test set ELBO: -150.91412353515625, time elapse for current epoch 46.91331195831299



Epoch: 39, Test set ELBO: -150.95849609375, time elapse for current epoch 45.48463296890259



Epoch: 40, Test set ELBO: -150.8723602294922, time elapse for current epoch 41.40005803108215

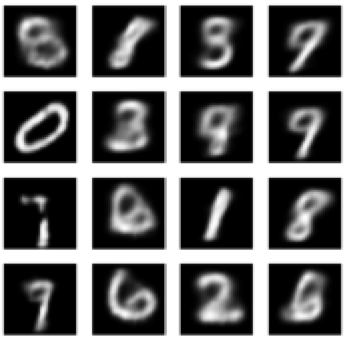


1.2.8. Изображение, которое получилось на 40-ой эпохе

[10]:
```
def display_image(epoch_no):
    return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
```

[11]:
```
plt.imshow(display_image(epochs))
plt.axis('off')# Display images
```

[11]: (-0.5, 359.5, 359.5, -0.5)



1.2.9. Создание gif из картинок с эпохами

[12]:
```
import glob
anim_file = 'cvae.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
```

```
    filenames = glob.glob('image*.png')
    filenames = sorted(filenames)
    last = -1
    for i,filename in enumerate(filenames):
        frame = 2*(i**0.5)
        if round(frame) > round(last):
            last = frame
        else:
            continue
        image = imageio.imread(filename)
        writer.append_data(image)
    image = imageio.imread(filename)
    writer.append_data(image)

from IPython import display
import IPython
if IPython.version_info >= (6,2,0,''):
    display.Image(filename=anim_file)
```

Гифка лежит в папке с лабой

1.2.10. Функция для вывода сетки изображений с варьируемым параметром

[13]:
```
n = 25
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))
grid_x = np.linspace(-3, 3, n)
grid_y = np.linspace(-3, 3, n)
def generate_images(model, epoch, writer):
    for i, yi in enumerate(grid_y):
        for j, xi in enumerate(grid_x):
            z_sample = np.array([[xi, yi]])
            x_decoded = model.sample(z_sample).numpy()
            digit = x_decoded[0].reshape(digit_size, digit_size)
            figure[i * digit_size : (i + 1) * digit_size, j * digit_size : (j+1) * digit_size] = digit

    with writer.as_default():
        image = np.reshape(figure, (1, digit_size*n, digit_size*n, 1))
        tf.summary.image("GEN DATA", image, step=epoch)

    plot_image(figure)

def plot_image(figure):
    plt.figure(figsize=(n //2, n//2))
    start_range = digit_size // 2
    end_range = (n - 1) * digit_size + start_range + 1
    pixel_range = np.arange(start_range, end_range, digit_size)
    sample_range_x = np.round(grid_x, 1)
    sample_range_y = np.round(grid_y, 1)
    plt.xticks(pixel_range, sample_range_x)
```

```
    plt.xticks(pixel_range, sample_range_y)
    plt.xlabel("Z[0]")
    plt.ylabel("Z[1]")
    plt.imshow(figure, cmap="Greys_r")
    plt.show()
```

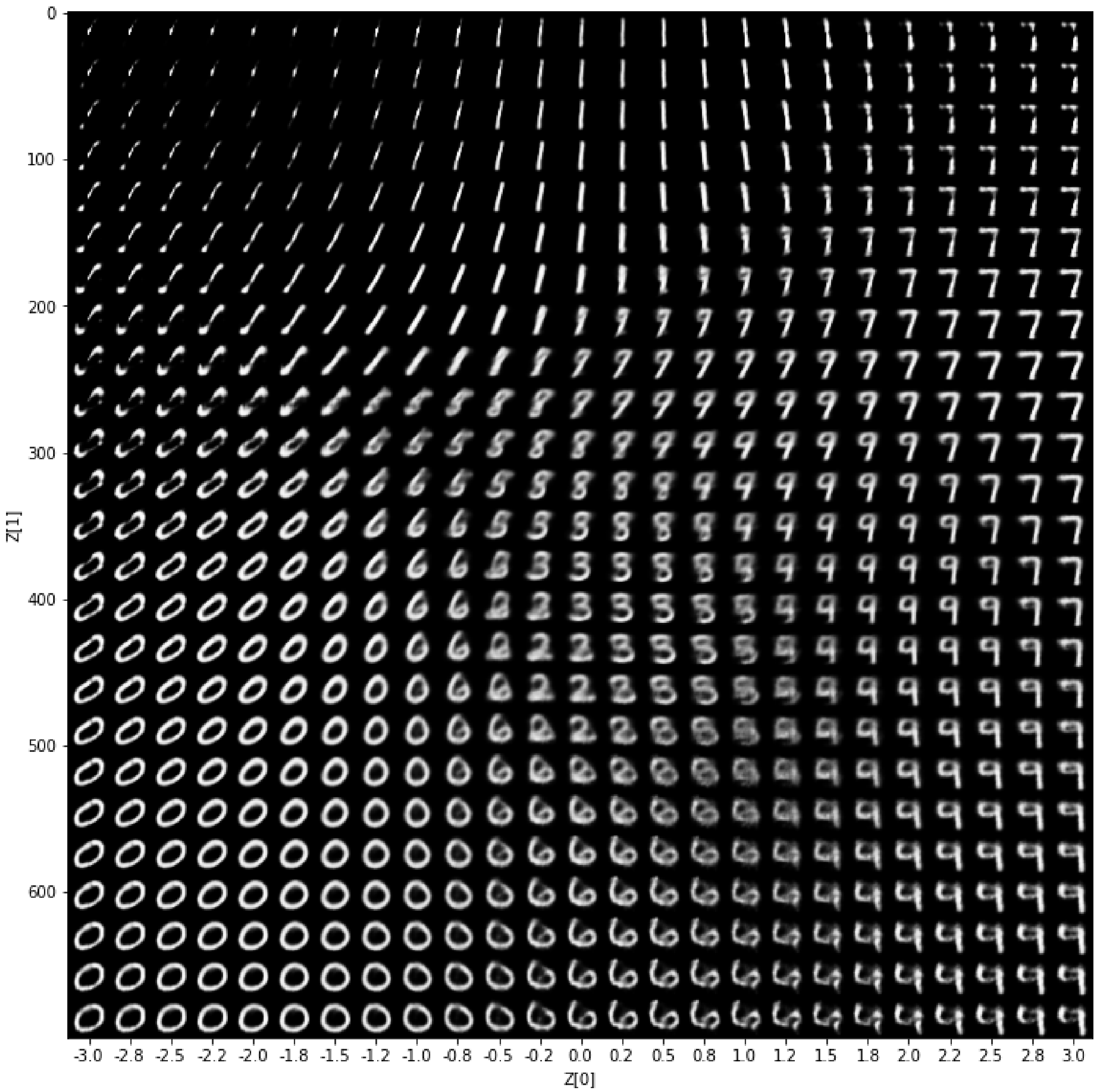
1.2.11. Получившаяся сетка с изображениями

[14]:
```
test_summary_writer = tf.summary.create_file_writer("./TEST")
generate_images(model, 40, test_summary_writer)
```

WARNING:tensorflow:Layer dense_1 is casting an input tensor from dtype float64 to the layer's dtype of float32, which is new behavior in TensorFlow 2. The layer has dtype float32 because it's dtype defaults to floatx.

If you intended to run this layer in float32, you can safely ignore this warning. If in doubt, this warning is likely only an issue if you are porting a TensorFlow 1.X model to TensorFlow 2.

To change all layers to have dtype float64 by default, call `tf.keras.backend.set_floatx('float64')`. To change just this layer, pass dtype='float64' to the layer constructor. If you are the author of this layer, you can disable autocasting by passing autocast=False to the base Layer constructor.



1.3. Вывод.

В ходе данной лабораторной работы написали вариационный автоэнкодер со сверточными слоями, убедились в его работоспособности

1.4. Список литературы

[1] Google. Tensorflow. 2018. Feb. url - https://www.tensorflow.org/install/install_windows.
[2] url - https://virtualenv.pypa.io/en/stable/userguide/.
[3] Microsoft. about_Execution_Policies. 2018. url - https://technet.microsoft.com/en-us/library/dd347641.aspx.
[4] Jupyter Project. Installing Jupyter. 2018. url - http://jupyter.org/install.

[ ]: