МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

ОТЧЕТ

Лабораторная работа №5

по курсу «Проектирование интеллектуальных систем»

Тема: «Вариационный автоэнкодер»

ИСПОЛНИТЕЛЬ: группа ИУ5-22

__Петрова Ирина___
ФИО

_____
подпись

"06"___05____2020 г.

ПРЕПОДАВАТЕЛЬ: _____
ФИО

_____
подпись

"__"_____2020 г.

Москва - 2020

Задание

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

```
[ ]  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(),
               # Без активации
               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'),
               # Без активации
               tf.keras.layers.Conv2DTranspose(
                   filters=1, kernel_size=3, strides=(1, 1), padding="SAME"),
             ]
         )

       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 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
```

```
[ ]  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)

     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)

     def compute_gradients(model, x):
       with tf.GradientTape() as tape:
         loss = compute_loss(model, x)
       return tape.gradient(loss, model.trainable_variables), loss

     def apply_gradients(optimizer, gradients, variables):
       optimizer.apply_gradients(zip(gradients, variables))
```

```
[ ]  epochs = 40
     latent_dim = 2
     num_examples_to_generate = 25

     # сохраняем случайный вектор постоянным для генерации (предсказания),
     # так будет легче увидеть улучшения.
     random_vector_for_generation = tf.random.normal(
         shape=[num_examples_to_generate, latent_dim])
     model = CVAE(latent_dim)
```

```
[ ]  def generate_and_save_images(model, epoch, test_input):
       predictions = model.sample(test_input)
       fig = plt.figure(figsize=(25,25))

       for i in range(predictions.shape[0]):
           plt.subplot(25, 25, i+1)
           plt.imshow(predictions[i, :, :, 0], cmap='gray')
           plt.axis('off')

       # tight_layout минимизирует перекрытия
       plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
       plt.show()
```

```
[ ] 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:
        gradients, loss = compute_gradients(model, train_x)
        apply_gradients(optimizer, gradients, model.trainable_variables)
      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()
        display.clear_output(wait=False)
        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: 40, Test set ELBO: -152.89031982421875, time elapse for current epoch 98.93774938583374
```

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

```
[ ] generate_and_save_images(model, 0, t_vector)
```