Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Лабораторная работа №5 по дисциплине**

«Проектирование интеллектуальных систем»

**ИСПОЛНИТЕЛЬ:**

Чечнев А.А.

Группа ИУ5-23М

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"\_\_"\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Москва 2020

1. **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
   2. Создать сетку из 25 изображений, где по оси Х изменяется значение первого элемента **z**, а по оси Y - второго элемента **z**
2. **Решение**

**Импорт библиотек**

from packaging import version

import tensorflow as tf

from tensorflow import keras

import datetime as dt

print("TensorFlow version: ", tf.\_\_version\_\_)

assert version.parse(tf.\_\_version\_\_).release[0] >= 2, \

    "This notebook requires TensorFlow 2.0 or above."

from packaging import version

import tensorflow as tf

from tensorflow import keras

import datetime as dt

print("TensorFlow version: ", tf.\_\_version\_\_)

assert version.parse(tf.\_\_version\_\_).release[0] >= 2, \

    "This notebook requires TensorFlow 2.0 or above."

**Загрузка данных MNIST и создание**[**Dataset**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset)**.**

(x\_train, \_), (x\_test, \_) = keras.datasets.mnist.load\_data()

train\_dataset = tf.data.Dataset.from\_tensor\_slices((x\_train)).batch(32).shuffle(10000)

train\_dataset = train\_dataset.map(lambda x: (tf.math.divide(tf.cast(x, tf.float32), 255.0)))

#Бинаризация

train\_dataset = train\_dataset.map(lambda x: (tf.round(x)))

train\_dataset = train\_dataset.map(lambda x: (tf.image.random\_flip\_left\_right(x)))

test\_dataset = tf.data.Dataset.from\_tensor\_slices((x\_test)).batch(32)

test\_dataset = test\_dataset.map(lambda x: (tf.math.divide(tf.cast(x, tf.float32), 255.0)))

#Бинаризация

test\_dataset = test\_dataset.map(lambda x: (tf.round(x)))

**Класс энкодер**

class MnistEncoder(keras.Model):

    def \_\_init\_\_(self):

        super(MnistEncoder, self).\_\_init\_\_(name='mnist\_encoder')

        with tf.device('/device:GPU:0'):

            self.flatten = keras.layers.Reshape(target\_shape=(28, 28, 1))

            self.conv2d\_1 = keras.layers.Conv2D(filters=32,

                                                kernel\_size = (3,3),

                                                padding='same',

                                                activation='relu')

            self.mxpool = keras.layers.MaxPool2D(pool\_size=(2, 2))

            self.flatten2 = keras.layers.Flatten()

            self.fc2 = keras.layers.Dense(64,

                                         activation=tf.nn.relu,

                                         kernel\_initializer=tf.initializers.RandomNormal,

                                         kernel\_regularizer=keras.regularizers.l2(l=0.001))

            self.dropout = keras.layers.Dropout(0.5)

            self.fc\_mu = keras.layers.Dense(2,

                                            activation=None,

                                            kernel\_initializer=tf.initializers.RandomNormal,

                                            kernel\_regularizer=keras.regularizers.l2(l=0.001))

            self.fc\_var = keras.layers.Dense(2,

                                             activation=None,

                                             kernel\_initializer=tf.initializers.RandomNormal,

                                             kernel\_regularizer=keras.regularizers.l2(l=0.001))

    def call(self, x, training=False):

        if training:

            x = self.dropout(

                  self.fc2(

                      self.flatten2(

                          self.mxpool(

                              self.conv2d\_1(

                                 self.flatten(x) ) ) ) ) )

        else:

            x = self.fc2(

                  self.flatten2(

                      self.mxpool(

                          self.conv2d\_1(

                              self.flatten(x) ) ) ) )

        mu = self.fc\_mu(x)

        var = self.fc\_var(x)

        return mu, var

**Класс декодер**

class MnistDecoder(keras.Model):

  def \_\_init\_\_(self):

    super(MnistDecoder, self).\_\_init\_\_(name='mnist\_decoder')

    with tf.device('/device:GPU:0'):

      self.decoder\_input = keras.layers.InputLayer(input\_shape=(2,))

      self.fc1 = keras.layers.Dense(units=7\*7\*32,

                                    activation="relu")

      self.reshape = keras.layers.Reshape(target\_shape=(7, 7, 32))

      self.conv2trd1 = keras.layers.Conv2DTranspose(filters=64,

                                                    kernel\_size=3,

                                                    strides = (2,2),

                                                    padding="same",

                                                    activation="relu")

      self.dropout = keras.layers.Dropout(rate=0.5)

      self.flatten = keras.layers.Flatten()

      self.fc2 = keras.layers.Dense(784)

      self.resh = keras.layers.Reshape(target\_shape=(28, 28))

  def call(self, x, training=False):

    x = self.decoder\_input(x)

    x = self.fc1(x)

    x = self.reshape(x)

    x = self.conv2trd1(x)

    x = self.flatten(x)

    x = self.fc2(x)

    #x = self.conv2dtr2(x)

    if training:

      x = self.dropout(x)

    x = self.resh(x)

    return x

**Класс вариационный автоэнкодер**

class VAE(keras.Model):

    # Функции от keras.Model

        def \_\_init\_\_(self):

            super(VAE, self).\_\_init\_\_(name='mnist\_vae')

            with tf.device('/device:GPU:0'):

                self.encoder = MnistEncoder()

                self.decoder = MnistDecoder()

        def call(self, x, training=False):

            mu, var = self.encoder(x,training)

            z = self.reparametrize(mu, var)

            x\_hat = self.decoder(z,training)

            return x\_hat, mu, var

    # Функция выполняющая репараметризацию

        def reparametrize(self, mu, var):

            eps = tf.random.normal(shape=mu.shape)

            return eps \* var + mu

    # Функция выполняющая получение изображение из вектора z. Заметим, что

    # здесь присутствует сигмоида

        def sample(self, eps=None):

            if eps is None:

                eps = tf.random.normal(shape=(100, self.latent\_dim))

            return tf.sigmoid(self.decoder(eps,False))

**Обучение**

from datetime import datetime

import numpy as np

optimizer = tf.keras.optimizers.Adam(1e-4)

summary\_writer = tf.summary.create\_file\_writer('logs', flush\_millis=10000)

summary\_writer.set\_as\_default()

@tf.function

def log\_normal\_pdf(mean, var, raxis=1):

    arg1 = tf.multiply(tf.pow(mean,2) + tf.pow(var,2) - tf.math.log(tf.pow(var,2)) - 1,0.5)

    return tf.reduce\_sum(arg1,axis=1)

@tf.function

def compute\_loss(model, x):

    x\_hat, mu, var = model(x,False)

    ## сигмоида встроена в функцию потери

    cross\_ent = tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits=x\_hat, labels=x)

    logpx\_z = -tf.reduce\_sum(cross\_ent, axis=[1, 2])

    logqz\_x = log\_normal\_pdf(mu, var)

    return -tf.reduce\_mean(logpx\_z  - logqz\_x)

@tf.function

def compute\_gradients(model, x):

    with tf.device('/gpu:0'):

        with tf.GradientTape() as tape:

            loss = compute\_loss(model, x)

        return tape.gradient(loss, model.trainable\_variables), loss

@tf.function

def apply\_gradients(optimizer, gradients, variables):

    optimizer.apply\_gradients(zip(gradients, variables))

def generate\_and\_save\_images(model, epoch, test\_input, file\_writer):

    '''

    Генерирует примеры на основе test\_input и записывает их в file\_writer для

    визуализации в tensorboard

    '''

    predictions = model.sample(test\_input)

    with file\_writer.as\_default():

        images = np.reshape(predictions, (-1, 28, 28, 1)) \* 255

        images[images < 0] = 0

        images[images > 255] = 255

        tf.summary.image("16 generated data examples", images, max\_outputs=16, step=epoch)

epochs = 40

latent\_dim = 2

num\_examples\_to\_generate = 16

# Создадим случайный вектор

random\_vector\_for\_generation = tf.random.normal(

    shape=[num\_examples\_to\_generate, latent\_dim])

model = VAE()

import time

current\_time = datetime.now().strftime("%Y%m%d-%H%M%S")

train\_log\_dir = 'logs/gradient\_tape/' + current\_time + '/train'

test\_log\_dir = 'logs/gradient\_tape/' + current\_time + '/test'

train\_summary\_writer = tf.summary.create\_file\_writer(train\_log\_dir)

test\_summary\_writer = tf.summary.create\_file\_writer(test\_log\_dir)

generate\_and\_save\_images(model, 0, random\_vector\_for\_generation, test\_summary\_writer)

# Создаем метрики для вычисления.

train\_loss = tf.keras.metrics.Mean('train\_loss', dtype=tf.float32)

test\_loss = tf.keras.metrics.Mean('test\_loss', dtype=tf.float32)

%tensorboard --logdir logs/gradient\_tape

for epoch in range(1, epochs + 1):

    for train\_x in train\_dataset:

        gradients, loss = compute\_gradients(model, train\_x)

        train\_loss(loss)

        apply\_gradients(optimizer, gradients, model.trainable\_variables)

    with train\_summary\_writer.as\_default():

        tf.summary.scalar('loss', train\_loss.result(), step=epoch)

    print(epoch,loss)

    if epoch % 1 == 0:

#         for test\_x in test\_dataset:

#             test\_loss(compute\_loss(model, test\_x))

#         with test\_summary\_writer.as\_default():

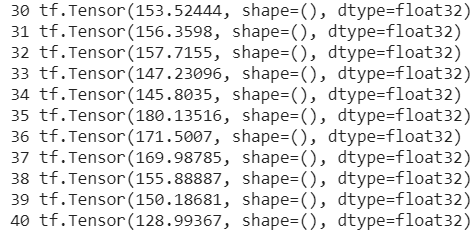
#             tf.summary.scalar('loss', test\_loss.result(), step=epoch)

        generate\_and\_save\_images(model, epoch, random\_vector\_for\_generation, test\_summary\_writer)

    # Обнуляем метрики на каждой эпохе

    train\_loss.reset\_states()

    test\_loss.reset\_states()



**Создание сетки изображений**

from scipy.stats import norm

import matplotlib.pyplot as plt

# Display a 2D manifold of the digits

n = 25  # figure with 25x25 digits

digit\_size = 28

batch\_size = 32

figure = np.zeros((digit\_size \* n, digit\_size \* n))

# Construct grid of latent variable values

grid\_x = norm.ppf(np.linspace(0.05, 0.95, n))

grid\_y = norm.ppf(np.linspace(0.05, 0.95, n))

# decode for each square in the grid

for i, yi in enumerate(grid\_x):

    for j, xi in enumerate(grid\_y):

        z\_sample = np.array([[xi, yi]])

        z\_sample = np.tile(z\_sample, batch\_size).reshape(batch\_size, 2)

        x\_decoded = model.sample(z\_sample)

        digit = tf.reshape(x\_decoded[0], (digit\_size, digit\_size))

        figure[i \* digit\_size: (i + 1) \* digit\_size,

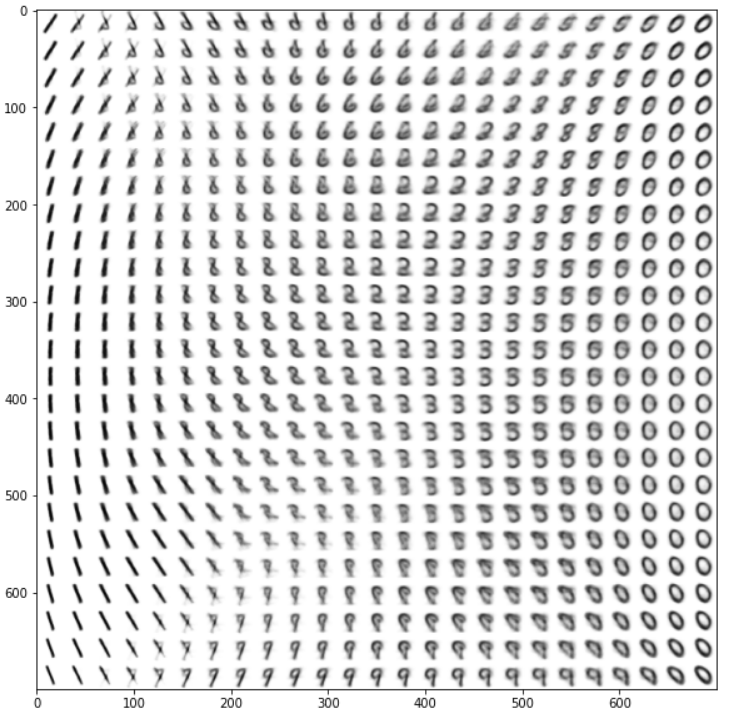
              j \* digit\_size: (j + 1) \* digit\_size] = digit

plt.figure(figsize=(10, 10))

plt.imshow(figure, cmap='binary')

plt.show()

**Результат**



# **Список литературы**

1. Терехов В.И., Черненький И.М., Методические указания к лабораторной работе №5 - М, 2020
2. TensorFlow [Электронный ресурс] - Режим доступа - <https://www.tensorflow.org/>