МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Искусственные нейронные сети»
Тема: «Распознавание рукописных символов»

Студент гр. 7382 _____ Государкин Я.С.
Преподаватель _____ Жангиров Т.Р

Санкт-Петербург
2020

Задание

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9).

1. Найти архитектуру сети, при которой точность будет выше 95%
2. Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения
3. Написать функцию, которая позволит загружать пользовательское изображение не из датасета

Теоретическая часть

Данная задача — задач многоклассовой классификации изображений. Одно изображение из датасета MNIST, на котором будет обучаться НС — имеет размерность 28 * 28 * 1.

Ход работы

1. Найти архитектуру с точностью выше 95%

Найденная архитектура даёт точность ~ 98%.

- Оптимизатор: Adam, lr=0.001
- Инициализация весов: normal
- batch_size = 100
- loss_func: categorical_crossentropy
- epochs = 6
- Layers:
  - `Dense(input_dim=28 * 28, units=28 * 14, activation='relu', kernel_initializer=self.initializer),`
  - `Dense(input_dim=28 * 14, units=28 * 5, activation='sigmoid', kernel_initializer=self.initializer)`
  - `Dense(input_dim=28 * 5, units=self.num_classes, activation='softmax')`

Графики точности и ошибки см. ниже.

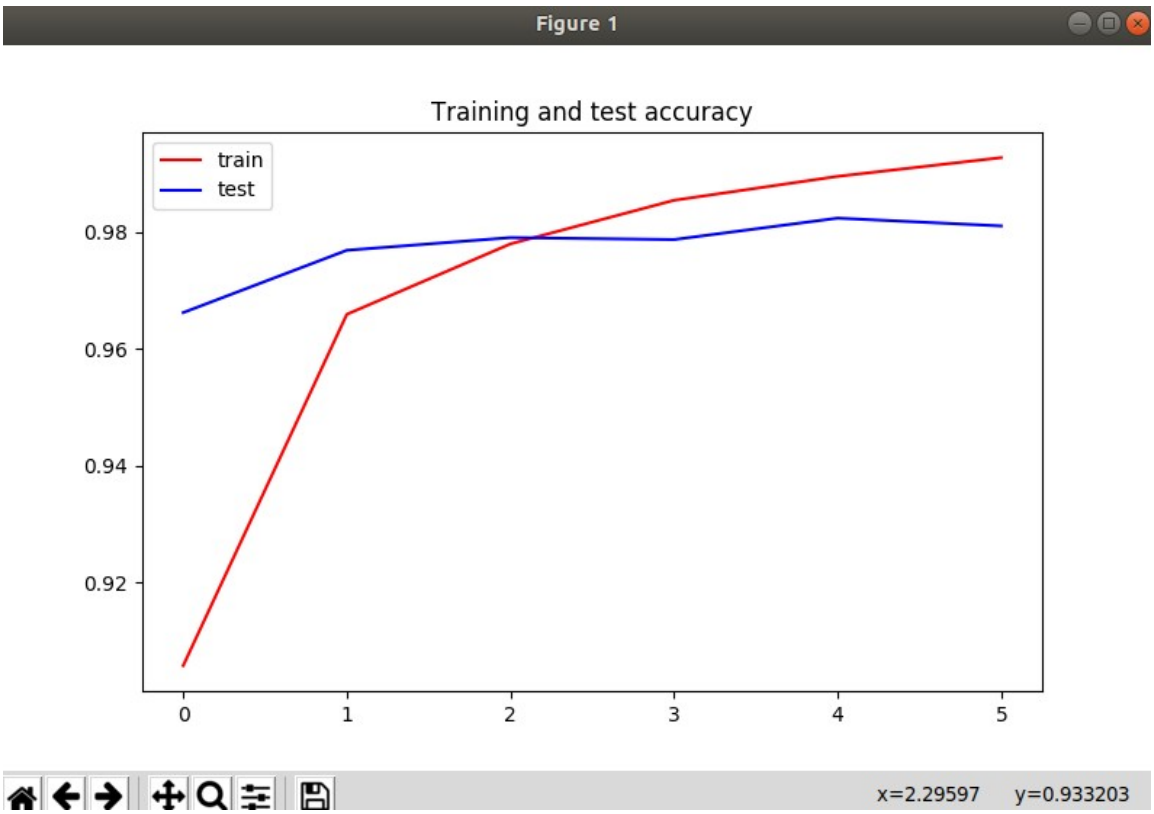


Figure 1
Training and test accuracy
train
test
0.98
0.96
0.94
0.92
0
1
2
3
4
5
x=2.29597
y=0.933203

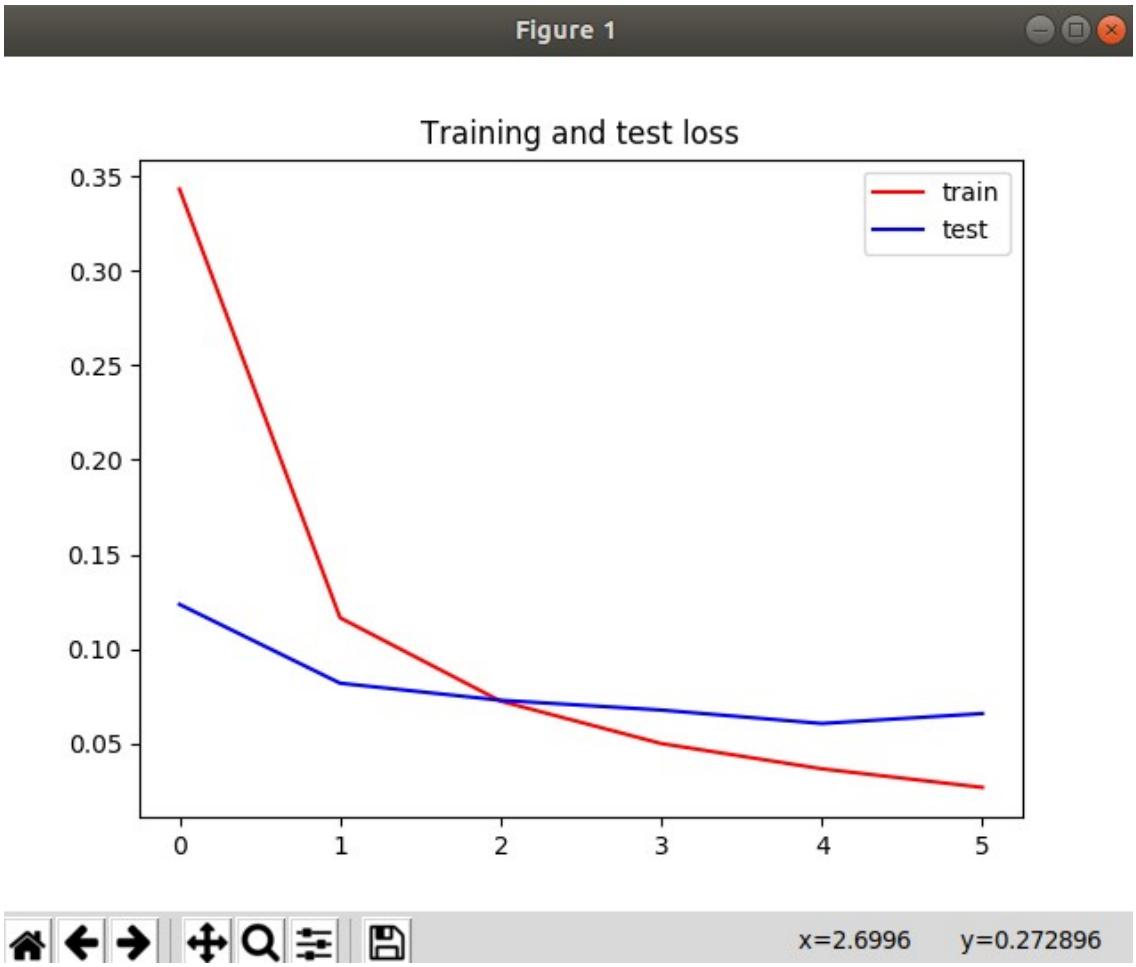


Figure 1
Training and test loss
train
test
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
1
2
3
4
5
x=2.6996
y=0.272896

2. Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения

Исследуем следующие конфигурации оптимизаторов (кол-во эпох, batch-size оставляем те же)

- SGD(lr=0.001, momentum=0.3)

lr — скорость обучения, momentum — коэффициент сохранения импульса

Средняя точность составила 67%

- Adagrad(lr=0.001)

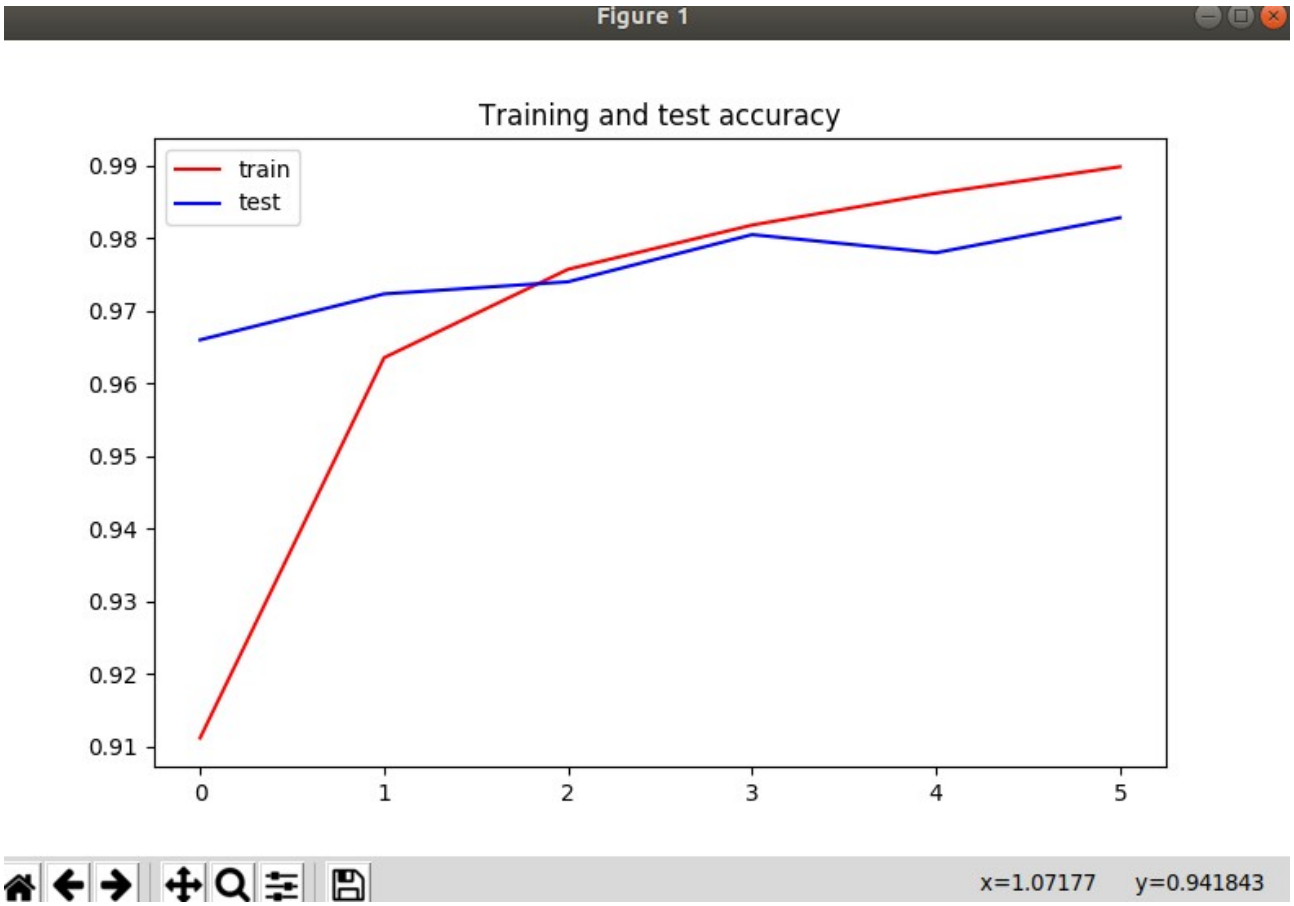
lr — скорость обучения

Средняя точность после обучения сети составила ~ 82%

- RMSProp(lr=0.001, rho=0.8)

lr — скорость обучения, rho — коэффициент затухания скользящего среднего значения градиента

Средняя точность классификации на тестовых данных после обучения сети составила ~ 98%



3. Написать функцию, которая позволит загружать пользовательское изображение не из датасета

Код:

```
import numpy as np
from PIL import Image


# function uploading image
def uploadImage(path):
    image = Image.open(path)
    data = np.asarray(image)
    return data

# uploading custom picture
data = uploadImage('pic.png')
print("Uploading TEST: " + data.shape)
```

Вывод в консоль:

(549, 788, 4) — x,y, и кол-во каналов (RGBA). Тестовое изображение есть в репо.

Она преобразует картинку в numpy array, который пригоден для использования keras-ом при обучении нейронных сетей.

Выводы.

В ходе лабораторной работы мы ознакомились с задачей классификации рукописных цифр, подобрали архитектуру, дающую точность на тестовой выборке — 98%. Было показано, что SGD и Adagrad сходятся хуже чем Adam и RMSProp. Также написана функция загрузки изображения в память программы. Код в приложении А.

ПРИЛОЖЕНИЕ А

```
import numpy as np
```

```
import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormalization,
Input
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import activations
from tensorflow.keras import initializers
from tensorflow.keras import optimizers
from tensorflow.keras import losses
import matplotlib.pyplot as plt
from PIL import Image
input_dim = 28 * 28
learn_rate = 0.001
epochs = 6
batch_size = 100
class Classifier(keras.Model):
    def __init__(self, num_classes=10):
        super(Classifier, self).__init__(name='example')
        self.initializer = initializers.normal
        self.num_classes = num_classes
        self.features = Sequential([
            Dense(input_dim=28 * 28, units=28 * 14, activation='relu',
kernel_initializer=self.initializer),
            Dense(input_dim=28 * 14, units=28 * 5, activation='sigmoid',
kernel_initializer=self.initializer),
            Dense(input_dim=28 * 5, units=self.num_classes, activation='softmax')
        ])
    def call(self, inputs):
        return self.features(inputs)
# function uploading image
def uploadImage(path):
    image = Image.open(path)
    data = np.asarray(image)
    return data
# uploading custom picture
data = uploadImage('pic.png')
print("Uploading TEST: " + data.shape)
# load dataset
mnist = keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
# normalize
x_train = x_train.reshape(-1, 28 * 28 * 1)
x_test = x_test.reshape(-1, 28 * 28 * 1)
x_train = x_train / 255.0
x_test = x_test / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
# init model
model = Classifier()
optimizer = optimizers.Adam(lr=learn_rate)
test1_opt = optimizers.SGD(lr=learn_rate, momentum=0.3)
test2_opt = optimizers.Adagrad(learn_rate)
test3_opt = optimizers.RMSprop(lr=learn_rate, rho=0.8)
loss = losses.CategoricalCrossentropy()
model.compile(optimizer=test3_opt, loss=loss, metrics=['accuracy'])
H = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
# testing
test_loss, test_acc = model.evaluate(x_test, y_test)
print('test_acc:', test_acc)
# plot
plt.figure(1,figsize=(8,5))
```

```
plt.title("Training and test accuracy")
plt.plot(H.history['acc'], 'r', label='train')
plt.plot(H.history['val_acc'], 'b', label='test')
plt.legend()
plt.show()
plt.clf()
plt.figure(1,figsize=(8,5))
plt.title("Training and test loss")
plt.plot(H.history['loss'], 'r', label='train')
plt.plot(H.history['val_loss'], 'b', label='test')
plt.legend()
plt.show()
plt.clf()
```