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

ОТЧЕТ

по лабораторной работе 4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студентка гр. 7383 _____ Иолшина В.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург

2020

Цель работы.

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

Задачи.

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения

Требования.

- Написать функцию, позволяющая загружать изображение пользователи и классифицировать его
- Найти архитектуру сети, при которой точность классификации будет не менее 95%
- Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения
- Написать функцию, которая позволит загружать пользовательское изображение не из датасета

Ход работы.

Была создана и обучена модель искусственной нейронной сети (код программы представлен в приложении А).

Исследуем влияние различных оптимизаторов, а также их параметров, на процесс обучения. Для этого рассмотрим оптимизаторы Adagrad, Adam, RMSprop и SGD с разными входными параметрами.

На рис. 1-8 представлены графики ошибок и точности для разных оптимизаторов.

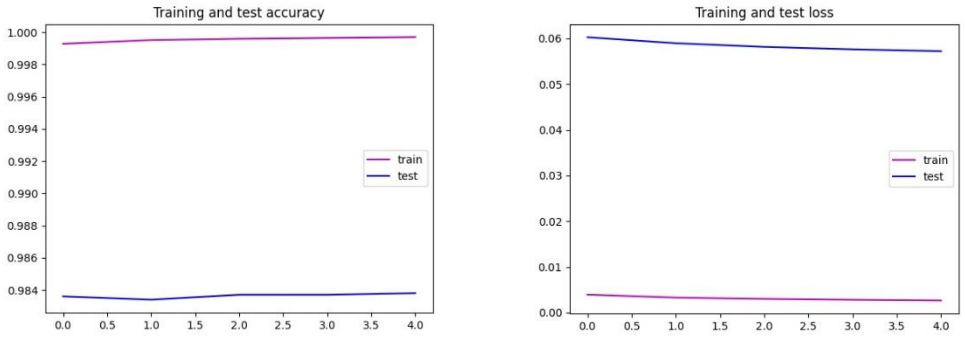

Рисунок 1 – Adagrad, learning_rate = 0.01

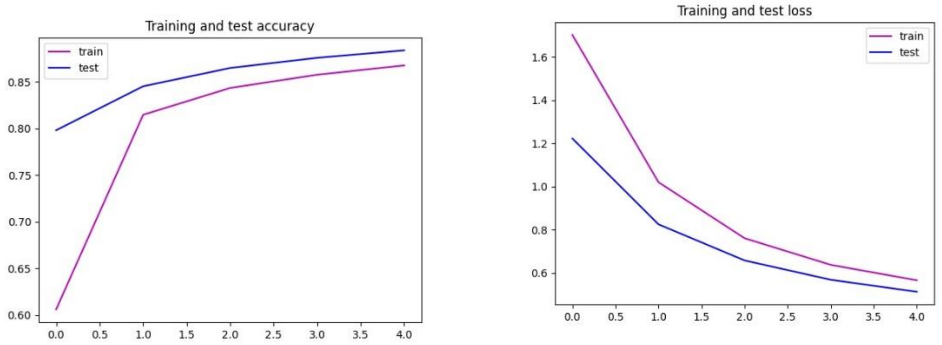

Рисунок 2– Adagrad, learning_rate = 0.001

Из графиков видно, что результаты работы сети улучшаются с увеличением скорости обучения. Наилучшая точность, равная 0.9838, для оптимизатора Adagrad была достигнута при скорости обучения, равной 0.01.

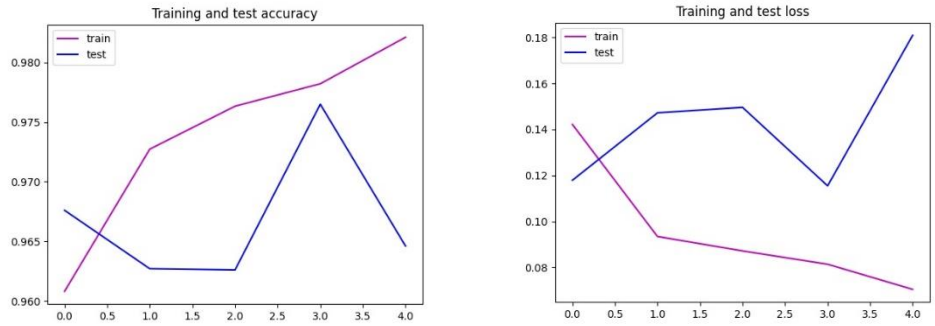

Рисунок 3 – Adam, learning_rate = 0.01

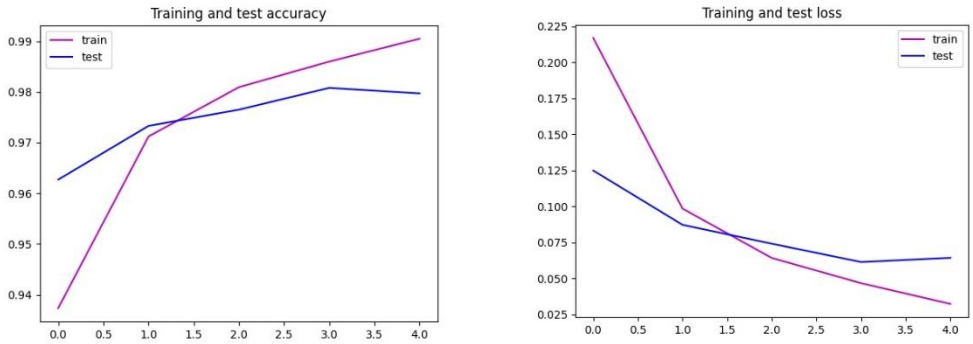

Рисунок 4 – Adam, learning_rate = 0.001

Из результатов видно, что результаты работы сети ухудшаются с 3 увеличением скорости обучения. Наилучшая точность, равная 0.9797, для оптимизатора Adam была достигнута при скорости обучения, равной 0.001

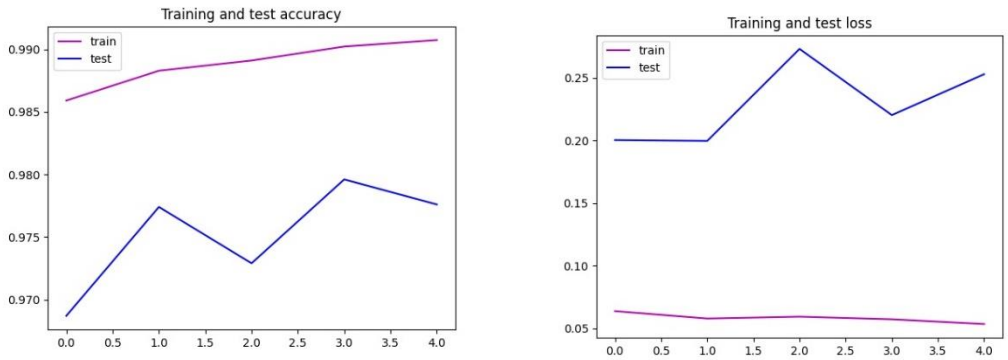

Рисунок 5 – RMSprop, learning_rate = 0.01

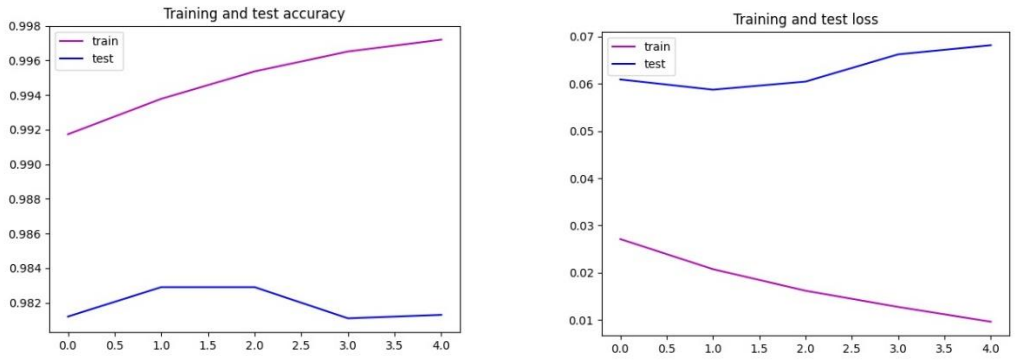

Рисунок 6 – RMSprop, learning_rate = 0.001

Из результатов видно, что результаты работы сети ухудшаются с увеличением скорости обучения. Наилучшая точность, равная 0.9813, для

оптимизатора RMSprop была достигнута при скорости обучения, равной 0.001.

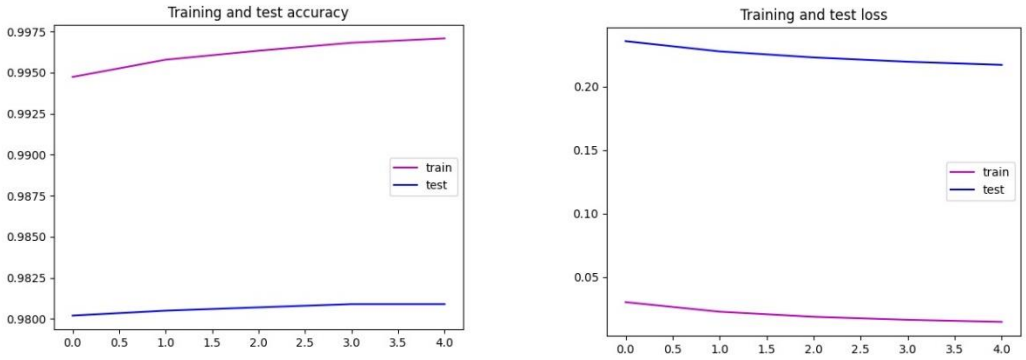

Рисунок 7 – SGD, learning_rate = 0.01

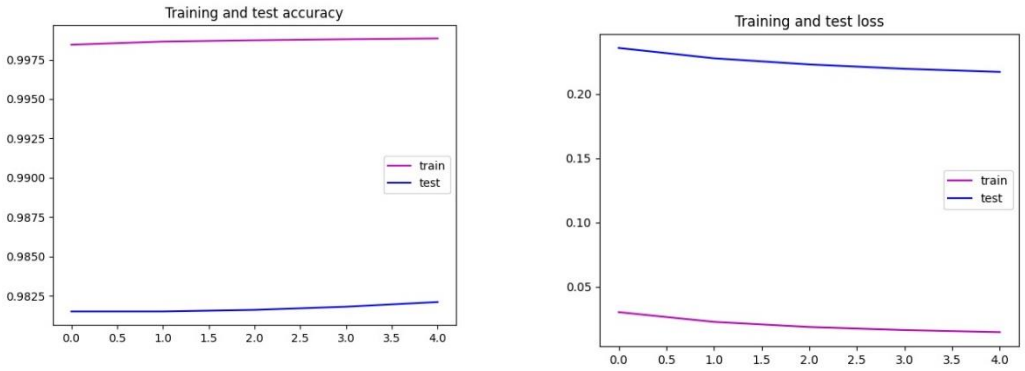

Рисунок 8 – SGD, learning_rate = 0.001

Из результатов видно, что результаты работы сети ухудшается с увеличением скорости обучения. Наилучшая точность, равная 0.9821, для оптимизатора SGD была достигнута при скорости обучения, равной 0.001.

Сравнение точности оптимизаторов приведено в табл.1.

Таблица 1 – Сравнение точности оптимизаторов

Optimizer	Adagrad	Adam	RMSprop	SGD
Accuracy	0.9838	0.9797	0.9813	0.9821

По результатам исследования можно заметить, что показатели сетей с оптимизатором Adagrad улучшились, а показатели сетей с оптимизаторами Adam, RMSprop и SGD ухудшаются при увеличении скорости обучения. Из графиков и значений следует, что наилучшая работа сети наблюдается при заданном оптимизаторе Adagrad с параметром learning_rate = 0.01.

Вывод.

В ходе выполнения лабораторной работы было изучено представление и обработка графических данных, простейших черно-белых изображений из базы данных MNIST, была также определена архитектура сети, при которой точность классификации будет не менее 95%. Было исследовано влияние различных оптимизаторов, а также их параметров на процесс обучения. Лучший результат был достигнут с оптимизатором Adagrad со скоростью обучения 0.01.

ПРИЛОЖЕНИЕ А

```
     import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from tensorflow.keras import optimizers
from numpy import asarray

EPOCHS = 5
result = dict()

def get_img(filename):
img = Image.open(filename).convert('RGB')
img = img.resize((28,28))
return np.expand_dims((1.0 - (asarray(img) / 255)), axis=0)

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images / 255.0

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images / 255.0

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation='relu', input_shape=(28 * 28,)))
model.add(Dense(10, activation='softmax'))

def train_model(optimizer, epochs):
optimizerConf = optimizer.get_config()
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=epochs,
batch_size=128, validation_data=(test_images, test_labels))

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)

plt.title('Training and test accuracy')
plt.plot(history.history['accuracy'], 'm', label='train')
plt.plot(history.history['val_accuracy'], 'b', label='test')
```

```
plt.legend()
plt.savefig("./%s%s_acc.png" % (optimizerConf["name"],
optimizerConf["learning_rate"]), format='png')
plt.show()
plt.clf()

plt.title('Training and test loss')
plt.plot(history.history['loss'], 'm', label='train')
plt.plot(history.history['val_loss'], 'b', label='test')
plt.legend()
plt.savefig("./%s%s_loss.png" % (optimizerConf["name"],
optimizerConf["learning_rate"]), format='png')
plt.show()
plt.clf()

result["%s%s" % (optimizerConf["name"], optimizerConf["learning_rate"])] =
test_acc

for learning_rate in [0.001, 0.01]:
train_model(optimizers.Adagrad(learning_rate=learning_rate), EPOCHS)
train_model(optimizers.Adam(learning_rate=learning_rate), EPOCHS)
train_model(optimizers.RMSprop(learning_rate=learning_rate), EPOCHS)
train_model(optimizers.SGD(learning_rate=learning_rate), EPOCHS)

for res in result:
print("%s: %s" % (res, result[res]))
```