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

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студент гр. 7383 _____ Зуев Д.В.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург

2020

Цели.

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
- Написать функцию, позволяющая загружать изображение пользователи и классифицировать его

Ход работы.

1. Для определения оптимальной архитектуры искусственной нейронной сети с точностью более 95% была взята к сведению статья [1]. Получившаяся архитектура представлена на рис. 1.

```
model.add(Flatten())
model.add(Dense(800, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Рисунок 1 - Архитектура сети

Подтверждение того, что точность данной архитектуры более 95% подтверждена ниже при исследовании результатов обучения на различных оптимизаторах.

2. Для нахождения наиболее точной модель нейронной сети были исследованы различные оптимизаторы с различными параметрами. Исследуемые оптимизаторы представлены на рис. 2.

```
optimizers_list = [optimizers.SGD(learning_rate=0.01, momentum=0.0, nesterov=False),
                   optimizers.SGD(learning_rate=0.1, momentum=0.0, nesterov=False),
                   optimizers.SGD(learning_rate=0.01, momentum=0.1, nesterov=True),
                   optimizers.SGD(learning_rate=0.1, momentum=0.1, nesterov=True),
                   optimizers.RMSprop(learning_rate=0.001, rho=0.9),
                   optimizers.RMSprop(learning_rate=0.1, rho=0.9),
                   optimizers.Adagrad(learning_rate=0.01), optimizers.Adagrad(learning_rate=0.1),
                   optimizers.Adadelta(learning_rate=1.0, rho=0.95),
                   optimizers.Adadelta(learning_rate=0.5, rho=0.5),
                   optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False),
                   optimizers.Adam(learning_rate=0.01, beta_1=0.99, beta_2=0.99, amsgrad=False),
                   optimizers.Adam(learning_rate=0.1, beta_1=0.999, beta_2=0.9, amsgrad=True),
                   optimizers.Adam(learning_rate=0.001, beta_1=0.999, beta_2=0.999, amsgrad=True),
                   optimizers.Adamax(learning_rate=0.002, beta_1=0.9, beta_2=0.999),
                   optimizers.Adamax(learning_rate=0.1, beta_1=0.999, beta_2=0.999),
                   optimizers.Adamax(learning_rate=0.002, beta_1=0.999, beta_2=0.999),
                   optimizers.Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999),
                   optimizers.Nadam(learning_rate=0.1, beta_1=0.999, beta_2=0.999),
                   optimizers.Nadam(learning_rate=0.002, beta_1=0.999, beta_2=0.999)]
```

Рисунок 2 - Исследуемые оптимизаторы

Сравнение потерь и точности моделей, построенных при использовании представленных оптимизаторов, представлено на рис. 3 и рис. 4 соответственно.

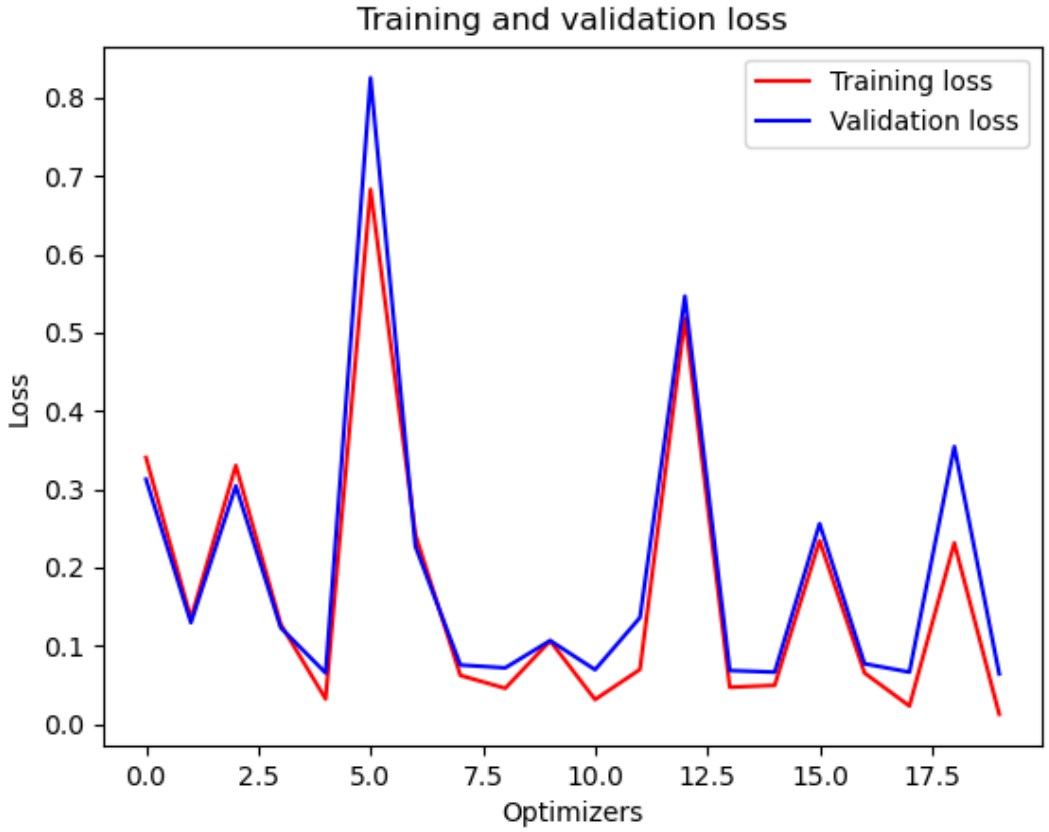

Рисунок 3 - Потери оптимизаторов

Рисунок 4 - Точности оптимизаторов

По рис. 3 и рис. 4, а так же по выводу программы `<tensorflow.python.keras.optimizer_v2.nadam.Nadam object at 0x000001D4A4613CF8> 19` (оптимизатор и его номер в массиве оптимизаторов) видно, что наибольшей точностью обладают следующие оптимизаторы:

- RMSprop(learning_rate=0.001, rho=0.9)
- Adagrad(learning_rate=0.01)
- Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
- Adamax(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
- Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
- Nadam(learning_rate=0.002, beta_1=0.999, beta_2=0.999)

Среди них оптимизатором, с помощью которого была построена самая точная модель является последний. Графики точности и потерь при

обучении с использованием последнего оптимизатора представлены на рис. 5 и рис. 6. соответственно.

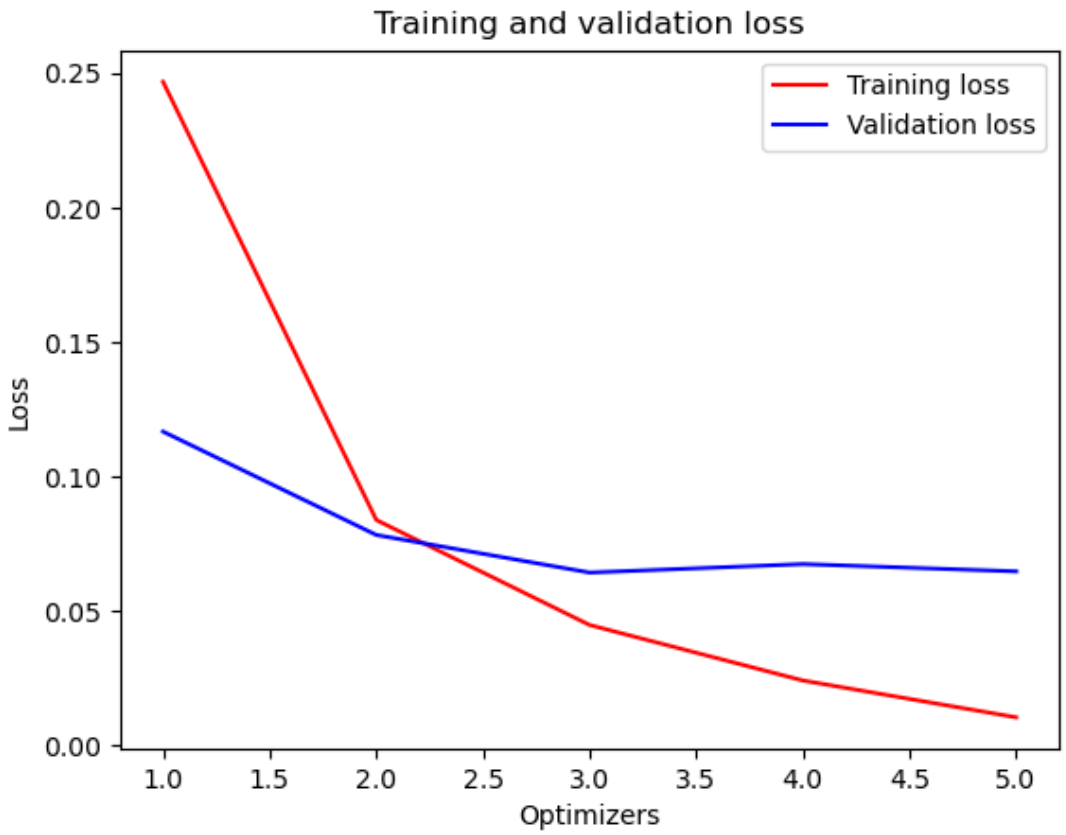

Рисунок 5 - Потери при использовании Nadam

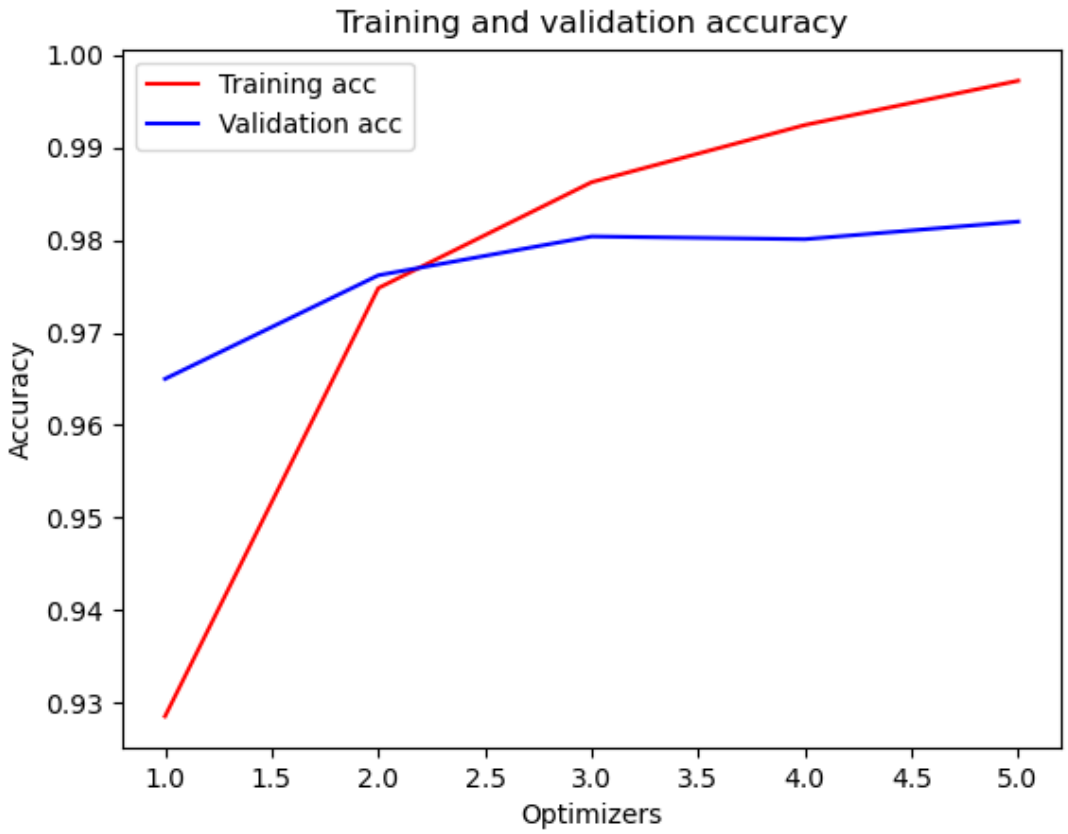

Рисунок 6 - Точность при использовании Nadam

3. Была написана функция для считывания картинки, содержащей цифру. Код представлен в приложении А. Тестовые рисунки представлены на рис. 7.

Рисунок 7 - Тестовые рисунки

Вывод результата работы нейросети на этих картинках представлен на рис. 8.

```
1: [1]
7: [6]
6: [6]
0: [0]
```

Рисунок 8 - Вывод программы

По выводу видно, что несмотря на достаточно большую точность, нейросеть может ошибиться при определении цифры на картинке. Это может быть связано с тем что стиль написания цифры на картинке может сильно отличаться от стиля написания цифр на тренировочных картинках.

Вывод.

В ходе выполнения данной лабораторной работы было изучено представление и обработка графических данных, был выявлен лучший оптимизатор для построения модели искусственной нейронной сети, распознающей рукописные цифры, была построена и протестирована на пользовательских изображениях модель.

Список использованных источников

1. Patrice Y. Simard. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis / Dave Steinkraus, John C. Platt // Seventh International Conference on Document Analysis and Recognition. – 2003.

ПРИЛОЖЕНИЕ А

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

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
optimizers_list = [optimizers.SGD(learning_rate=0.01,
momentum=0.0, nesterov=False),
                   optimizers.SGD(learning_rate=0.1,
momentum=0.0, nesterov=False),
                   optimizers.SGD(learning_rate=0.01,
momentum=0.1, nesterov=True),
                   optimizers.SGD(learning_rate=0.1,
momentum=0.1, nesterov=True),
                   optimizers.RMSprop(learning_rate=0.001,
rho=0.9),
                   optimizers.RMSprop(learning_rate=0.1,
rho=0.9),
                   optimizers.Adagrad(learning_rate=0.01),
optimizers.Adagrad(learning_rate=0.1),
                   optimizers.Adadelta(learning_rate=1.0,
rho=0.95),
                   optimizers.Adadelta(learning_rate=0.5,
rho=0.5),
                   optimizers.Adam(learning_rate=0.001,
beta_1=0.9, beta_2=0.999, amsgrad=False),
                   optimizers.Adam(learning_rate=0.01,
beta_1=0.99, beta_2=0.99, amsgrad=False),
                   optimizers.Adam(learning_rate=0.1,
beta_1=0.999, beta_2=0.9, amsgrad=True),
                   optimizers.Adam(learning_rate=0.001,
beta_1=0.999, beta_2=0.999, amsgrad=True),
                   optimizers.Adamax(learning_rate=0.002,
beta_1=0.9, beta_2=0.999),
                   optimizers.Adamax(learning_rate=0.1,
beta_1=0.999, beta_2=0.999),
                   optimizers.Adamax(learning_rate=0.002,
beta_1=0.999, beta_2=0.999),
```

```
                    optimizers.Nadam(learning_rate=0.002,
beta_1=0.9, beta_2=0.999),
                    optimizers.Nadam(learning_rate=0.1,
beta_1=0.999, beta_2=0.999),
                    optimizers.Nadam(learning_rate=0.002,
beta_1=0.999, beta_2=0.999)]


def load_image(path):
    image = Image.open(path).convert('L')
    image = 255 - np.array(image) # т.к. черная цифра на
белом фоне
    image = image/255
    return np.expand_dims(image, axis=0)


def build_model(optimizer):
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(800, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])
    print(model.optimizer)
    h = model.fit(train_images, train_labels, epochs=5,
batch_size=128)
    test_loss, test_acc = model.evaluate(test_images,
test_labels)
    train_acc = h.history['accuracy'][-1]
    train_loss = h.history['loss'][-1]
    return train_acc, train_loss, test_acc, test_loss


def test_optimizers():
    best_optimizer = optimizers.SGD(learning_rate=0.01,
momentum=0.0, nesterov=False)
    index = 0
    min_test_accuracy = 0
    train_acc_list = []
    test_acc_list = []
    train_loss_list = []
    test_loss_list = []
    for optimizer in optimizers_list:
        train_acc, train_loss, test_acc, test_loss =
build_model(optimizer)
        train_acc_list.append(train_acc)
        train_loss_list.append(train_loss)
        test_acc_list.append(test_acc)
        test_loss_list.append(test_loss)
        if test_acc > min_test_accuracy:
            best_optimizer = optimizer
            index = optimizers_list.index(optimizer)
```

```
    plt.plot(range(len(optimizers_list)), train_loss_list,
'r', label='Training loss')
    plt.plot(range(len(optimizers_list)), test_loss_list,
'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Optimizers')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()
    plt.plot(range(len(optimizers_list)), train_acc_list,
'r', label='Training acc')
    plt.plot(range(len(optimizers_list)), test_acc_list,
'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Optimizers')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
    return best_optimizer, index


optimizer, index = test_optimizers()
print(optimizer, index)
model = Sequential()
model.add(Flatten())
model.add(Dense(800, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])
h = model.fit(train_images, train_labels, epochs=5,
batch_size=128, validation_data=(test_images,
test_labels))

loss = h.history['loss']
val_loss = h.history['val_loss']
acc = h.history['accuracy']
val_acc = h.history['val_accuracy']
print(val_acc[-1])

image = load_image('1.png')
print('1:', model.predict_classes(image))
image = load_image('7.png')
print('7:', model.predict_classes(image))
image = load_image('6.png')
print('6:', model.predict_classes(image))
image = load_image('0.png')
print('0:', model.predict_classes(image))

epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
```

```
plt.xlabel('Optimizers')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.clf()
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Optimizers')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```