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

Студентка гр. 7383 _____ Тян Е.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9). Набор данных содержит 60,000 изображений для обучения и 10,000 изображений для тестирования.

Задание.

1. Ознакомиться с представлением графических данных.
2. Ознакомиться с простейшим способ передачи графических данных нейронной сети.
3. Создать модель.
4. Настроить параметры обучения.
5. Написать функцию, позволяющую загружать изображение пользователя и классифицировать его.

Ход работы.

Набор данных MNIST уже входит в состав Keras в форме набора из четырех массивов Numpy:

```
import tensorflow as tf

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

Здесь train_images и train_labels — это тренировочный набор, то есть данные, необходимые для обучения. После обучения модель будет проверяться тестовым (или контрольным) набором, test_images и test_labels. Изображения хранятся в массивах Numpy, а метки — в массиве цифр от 0 до 9. Изображения и метки находятся в прямом соответствии, один к одному.

Для проверки корректности загрузки достаточно сравнить тестовое изображение с его меткой:

```
import matplotlib.pyplot as plt

plt.imshow(train_images[0],cmap=plt.cm.binary)
plt.show()
print(train_labels[0])
```

Исходные изображения представлены в виде массивов чисел в интервале [0, 255]. Перед обучением их необходимо преобразовать так, чтобы все значения оказались в интервале [0, 1]:

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Также необходимо закодировать метки категорий. В данном случае прямое кодирование меток заключается в конструировании вектора с нулевыми элементами со значением 1 в элементе, индекс которого соответствует индексу метки:

```
from keras.utils import to_categorical

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

Теперь можно задать базовую архитектуру сети:

```
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.models import Sequential

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

Чтобы подготовить сеть к обучению, нужно настроить еще три параметра для этапа компиляции:

- функцию потерь, которая определяет, как сеть должна оценивать качество своей работы на обучающих данных и, соответственно, как корректировать ее в правильном направлении;
- оптимизатор — механизм, с помощью которого сеть будет обновлять себя, опираясь на наблюдаемые данные и функцию потерь;
- метрики для мониторинга на этапах обучения и тестирования — здесь нас будет интересовать только точность (доля правильно классифицированных изображений).

```
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])
```

Теперь можно начинать обучение сети, для чего в случае использования библиотеки Keras достаточно вызвать метод fit сети — он пытается адаптировать (fit) модель под обучающие данные:

```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

В процессе обучения отображаются две величины: потери сети на обучающих данных и точность сети на обучающих данных.

Теперь проверим, как модель распознает контрольный набор:

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

Была получена точность классификации: 0.9786.

Исследуем влияние различных оптимизаторов (‘Adam’, ‘Adagrad’, ‘RMSprop’), а также их параметров на процесс обучения.

На рис. 1 – 6 изображены графики ошибки и точности для оптимизаторов с параметром learning_rate = 0.01. Для оптимизатора Adagrad были получены результы: loss=0.24, accuracy=0.93. Для оптимизатора Adam были получены результы: loss=0.15, accuracy=0.96. Для оптимизатора SGD были получены результы: loss=0.32, accuracy=0.91.

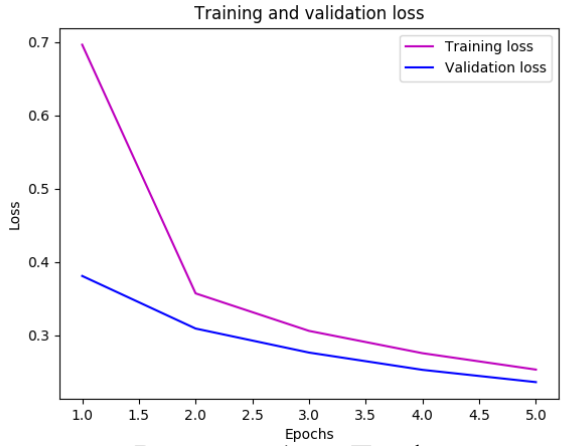

Рисунок 1 — График ошибки(оптимизатор Adagrad)

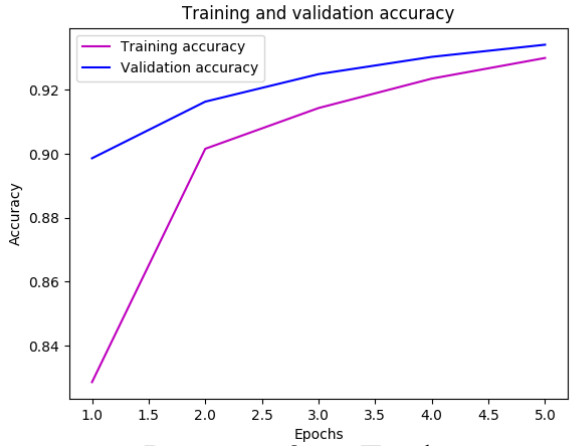

Рисунок 2 — График точности(оптимизатор Adagrad)

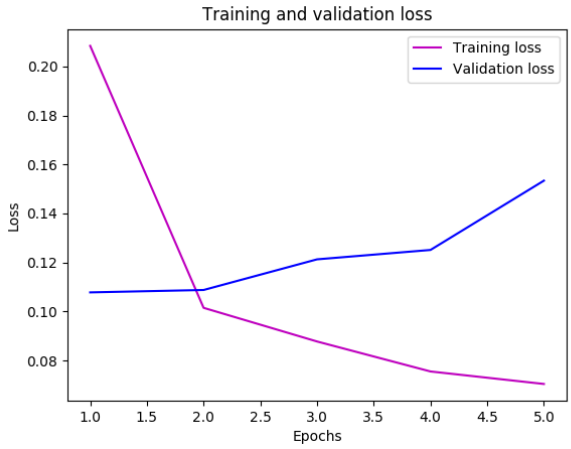

Рисунок 3 — График ошибки(оптимизатор Adam)

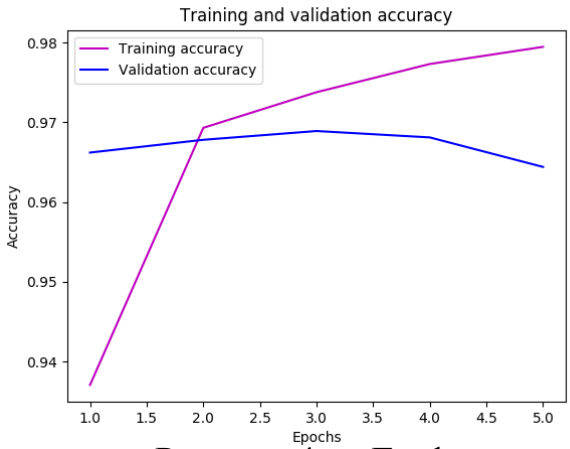

Рисунок 4 — График точности(оптимизатор Adam)

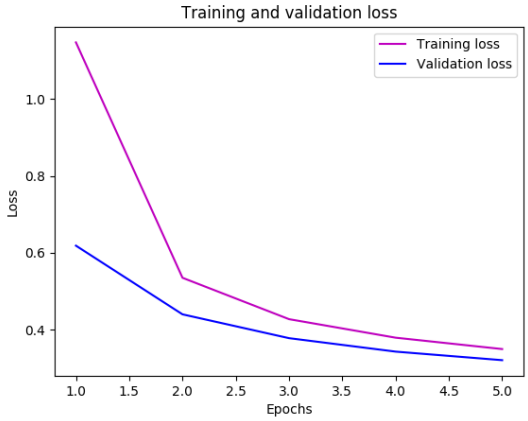

Рисунок 5 — График ошибки(оптимизатор SGD)

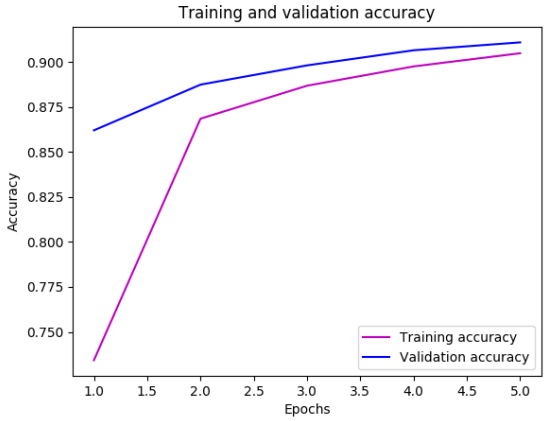

Рисунок 6 — График точности(оптимизатор SGD)

На рис. 7 – 12 изображены графики ошибки и точности для оптимизаторов с параметром learning_rate = 0.001. Для оптимизатора Adagrad были получены результы: loss=0.98, accuracy=0.81. Для оптимизатора Adam были получены результы: loss=0.08, accuracy=0.97. Для оптимизатора SGD были получены результы: loss=0.55, accuracy=0.87.

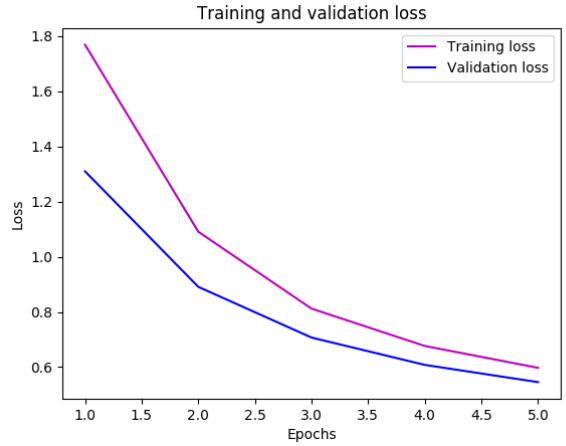

Рисунок 7 — График ошибки(оптимизатор Adagrad)

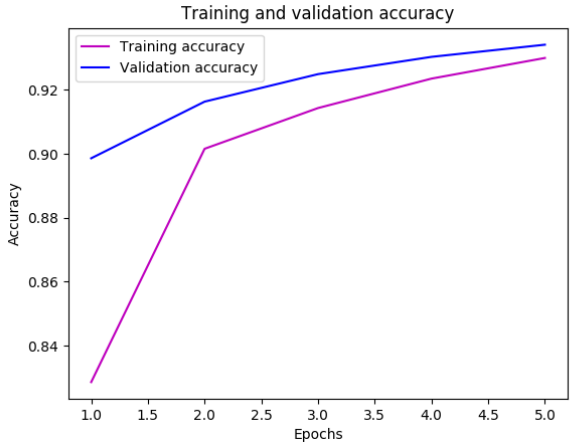

Рисунок 8 — График точности(оптимизатор Adagrad)

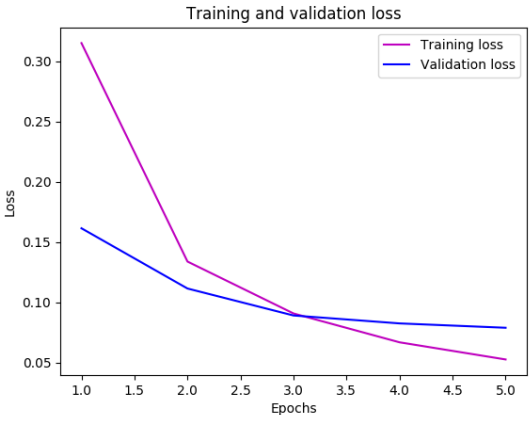

Рисунок 9 — График ошибки(оптимизатор Adam)

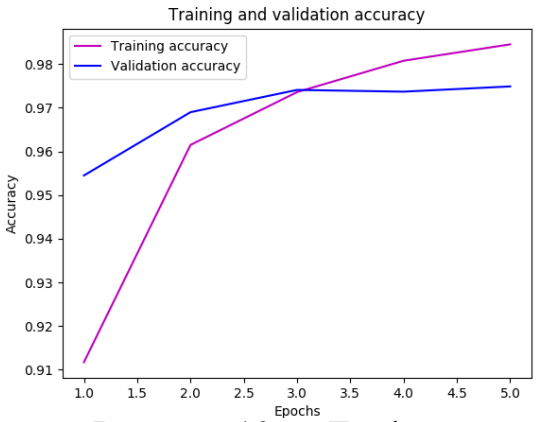

Рисунок 10 — График точности(оптимизатор Adam)

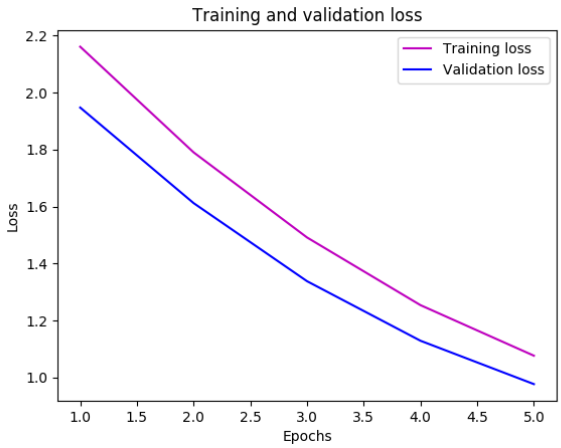

Рисунок 11 — График ошибки(оптимизатор SGD)

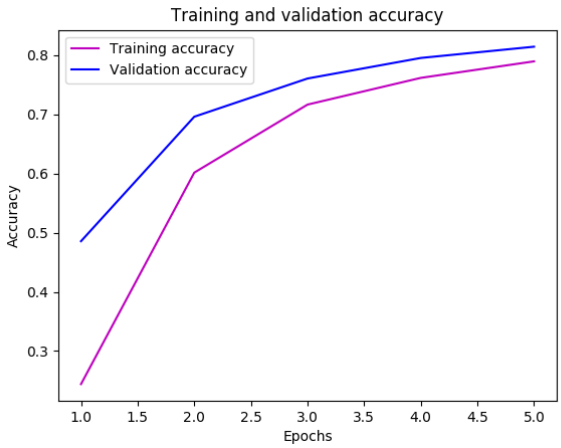

Рисунок 12 — График точности(оптимизатор SGD)

Можно заметить, что показатели ухудшились, при уменьшении скорости обучения для Adagrad и SGD. Для оптимизатора Adam, при уменьшении скорости обучения, точность увеличилась, а ошибка уменьшилась.

Выводы.

В ходе выполнения лабораторной работы была определена архитектура сети, при которой точность классификации будет не менее 95%. Было исследовано влияние различных оптимизаторов, а также их параметров, на процесс обучения. Написана функция, которая позволит загружать пользовательское изображение не из датасета.