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. 8382 _____ Кузина А.М.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы

Реализовать классификацию черно-белых изображений рукописных цифр (28×28) по 10 категориям (от 0 до 9). Набор данных содержит 60000 изображений для обучения и 10000 изображений для тестирования.

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

Требования:

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

Ход работы

Каждое изображение датасета представлено в виде матрицы интенсивностей пикселей, т.е. двумерного тензора размера 28×28, состоящего из чисел от 0 до 255. Данные масштабируются для упрощения обучения сети так, чтобы значения оказались в интервале [0, 1].

Базовая архитектура сети состоит из трех слоев – входной слой, переводящий матрицу 28 на 28 в вектор длины 784, скрытый слой с 256

нейронами и функцией активации relu и выходной слой с 10ю нейронами и функцией активации softmax.

Далее производится обучение сети в течении 5 эпох с различными оптимизаторами и параметрами их настройки. Ниже приведены рассматриваемые варианты и точности, которые были на них получены:

Adam():

313/313 [==============================] - 0s 593us/step - loss: 0.0733 - accuracy: 0.9784

test_acc: 0.9783999919891357

Adam(learning_rate=0.05):

313/313 [==============================] - 0s 619us/step - loss: 0.3245 - accuracy: 0.9325

test_acc: 0.9325000047683716

Adam(learning_rate=0.0002):

313/313 [==============================] - 0s 606us/step - loss: 0.1403 - accuracy: 0.9587

test_acc: 0.9587000012397766

Adam(learning_rate=0.2):

313/313 [==============================] - 0s 632us/step - loss: 1.9773 - accuracy: 0.2758

test_acc: 0.2757999897003174

Adagrad():

313/313 [==============================] - 0s 605us/step - loss: 0.5387 - accuracy: 0.8784

test_acc: 0.8784000277519226

Adagrad(0.5):

313/313 [==============================] - 0s 600us/step - loss: 0.1097 - accuracy: 0.9660

test_acc: 0.9660000205039978

Adagrad(0.1):

313/313 [==============================] - 0s 587us/step - loss: 0.0919 - accuracy: 0.9717

test_acc: 0.9717000126838684

Adagrad(0.0001):

313/313 [==============================] - 0s 609us/step - loss: 1.7786 - accuracy: 0.6114

test_acc: 0.6114000082015991

Adamax():

313/313 [==============================] - 0s 603us/step - loss: 0.1184 - accuracy: 0.9661

test_acc: 0.9660999774932861

Adamax(learning_rate = 0.5):

313/313 [==============================] - 0s 699us/step - loss: 0.4843 - accuracy: 0.8990

test_acc: 0.8989999890327454

Adamax(learning_rate = 0.05):

313/313 [==============================] - 0s 622us/step - loss: 0.1121 - accuracy: 0.9705

test_acc: 0.9704999923706055

Adamax(learning_rate = 0.0001):

313/313 [==============================] - 0s 680us/step - loss: 0.2925 - accuracy: 0.9195

test_acc: 0.9194999933242798

RMSprop():

313/313 [==============================] - 0s 638us/step - loss: 0.0775 - accuracy: 0.9776

test_acc: 0.9775999784469604

RMSprop(learning_rate = 0.5):

313/313 [==============================] - 0s 622us/step - loss: 2.4099 - accuracy: 0.3445

test_acc: 0.34450000524520874

RMSprop(learning_rate = 0.01):

313/313 [==============================] - 0s 699us/step - loss: 0.1591 - accuracy: 0.9729

test_acc: 0.9728999733924866

RMSprop(rho= 0.5):

313/313 [==============================] - 0s 615us/step - loss: 0.0957 - accuracy: 0.9736

test_acc: 0.9735999703407288

RMSprop(rho= 0.9):

313/313 [==============================] - 0s 612us/step - loss: 0.0758 - accuracy: 0.9778

test_acc: 0.9778000116348267

Adadelta():

313/313 [==============================] - 0s 628us/step - loss: 1.8345 - accuracy: 0.5434

test_acc: 0.54339998960495

Adadelta(learning_rate = 1.5):

313/313 [==============================] - 0s 622us/step - loss: 0.0677 - accuracy: 0.9796

test_acc: 0.9796000123023987

Adadelta(learning_rate = 0.5):

313/313 [==============================] - 0s 593us/step - loss: 0.1029 - accuracy: 0.9709

test_acc: 0.9708999991416931

Adadelta(learning_rate = 0.05):

313/313 [==============================] - 0s 673us/step - loss: 0.2684 - accuracy: 0.9267

test_acc: 0.9266999959945679

Adadelta(rho= 0.9):

313/313 [==============================] - 0s 603us/step - loss: 1.9131 - accuracy: 0.5157

test_acc: 0.5156999826431274

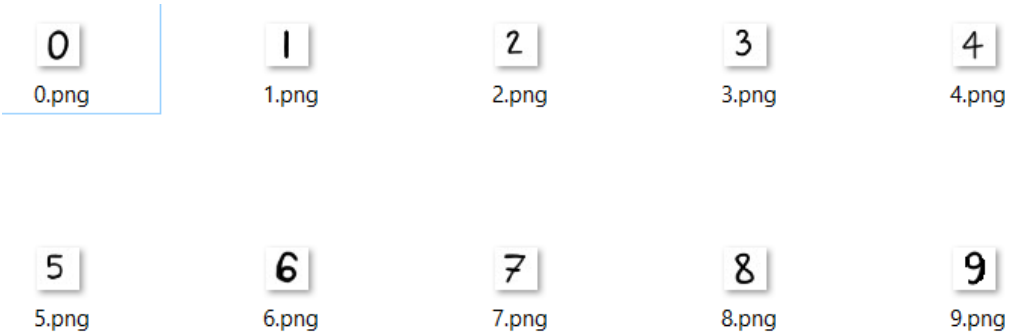
Из полученных данных можно сделать вывод о том, что каждый из рассмотренных оптимизаторов выдаст результат свыше 95% точности при правильно подобранных параметрах. Итоговая конфигурация сети:

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

model.compile(optimizer= Adadelta(learning_rate = 1.5), loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=128, verbose=0)
```

Далее был реализован функционал классификации пользовательских изображений. Отдельно функция загрузки изображения и предсказания на его основе. Также было учтено то, что в датасете цифры написаны на черном фоне белым цветом, что может звучать непривычно и нелогично для пользователя – гораздо чаще люди пишут черным по белому. Поэтому в функции загрузки изображения цвета инвертируются, что проще для восприятия.

Затем загружаются 10 изображений цифр – от 0 до 9 последовательно, и по каждому из них уже обученная сеть делает предсказание. Результаты печатаются на экран в сравнительном виде – что было на изображении и какое предсказание сети.



Все представленные изображения 28 на 28 пикселей. Рассмотрим результаты сети:

```
test_acc: 0.9779999852180481        test_acc: 0.9799000024795532
На картинке - 0  Предсказание - 0   На картинке - 0  Предсказание - 0
На картинке - 1  Предсказание - 1   На картинке - 1  Предсказание - 1
На картинке - 2  Предсказание - 2   На картинке - 2  Предсказание - 2
На картинке - 3  Предсказание - 3   На картинке - 3  Предсказание - 3
На картинке - 4  Предсказание - 4   На картинке - 4  Предсказание - 4
На картинке - 5  Предсказание - 5   На картинке - 5  Предсказание - 8
На картинке - 6  Предсказание - 6   На картинке - 6  Предсказание - 6
На картинке - 7  Предсказание - 2   На картинке - 7  Предсказание - 2
На картинке - 8  Предсказание - 8   На картинке - 8  Предсказание - 8
На картинке - 9  Предсказание - 3   На картинке - 9  Предсказание - 9
```

Можно заметить, что несмотря на столь высокую точность на датасете, на пользовательских изображениях некоторые ошибки – не редкость. Связано это скорее всего со спецификой написания цифр в датасете и пользователем – например наклон цифры, курсивное написание и черточки (например две или одна параллельные черты в 7 и одной чертой или «уголком» написана 1) Но также стоит заметить, что сеть чаще путает цифры похожими. Например 9 и 3 отличаются всего одним соединением, а 5 и 8 всего двумя. В среднем сеть путает 1-2 цифры.

Выводы

В ходе лабораторной работы были изучены возможности работы с изображениями, их использование как датасет нейронной сети, инвертирование цветов изображения, получение предсказания о содержимом изображения с использованием нейронных сетей.

Рассмотрены различные оптимизаторы и их параметры, то как их изменение влияет на точность сети. Было определено, что для данной задачи, каждый из рассмотренных оптимизаторов может предоставить требуемую точность в 95% при подобранных параметрах.