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. 8383 _____ Ларин А.

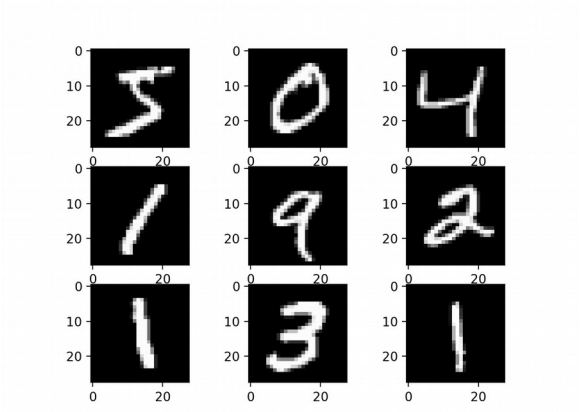
Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

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

- 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

Выполнение

Необходимо решить задачу классификации. Она заключается в распознавании рукописных символов, т. е. отнесению их к одному из конечного набора классов `0`-`9`

1) Набор данных для обучения входит в состав Keras. Он был загружен

Сначала был загружен набор данных «Boston housing» из Keras.

Листинг 1:
```
import tensorflow as tf

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

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].

Затем он был нормализован. Для этого из каждого значения вычтено среднее, а результат поделен на стандартное отклонение, что дает признаки центрированные по нулю и имеющие стандартное отклонение равное единице.

Листинг 2:
```
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.

Листинг 3:
```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Далее была создана базовая модель для обучения
Листинг 4:
```
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'))
```

Модель скомпилирована с начальными параметрами

Листинг 5:
```
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])
```

А затем обучена на загруженных данных

Листинг 6:
```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

Средняя(по 10 прогонам) точность составила 0.977

У оптимизатора Adam по-умолчанию задан скорость обучения (learning rate) 0.001.

Модель была обучена со скоростью обучения 0.01.
Средняя(по 5) составила 0.968
Для скорости обучения 0.1 — 0.862.

Т.е. при увеличении скорости обучения точность падает

Модель была обучена со скоростью 0.0001. Точность составила 0.943.
Оптимальный результат получен при скорости равной 0.001

Особенность оптимизатора Adam в том, что он вычисляет индивидуальные адаптивные скорости обучения для различных параметров из оценок первого и второго моментов градиентов. За показатели экспоненциальной скорости затухания моментов отвечают коэффициенты beta_1 и beta_2. Во всех популярных реализациях они равны 0.9 и 0.999 по-умолчанию

Модель обучена с beta_1 = 0.89 и 0.85.
В обох случаюх точность составила примерно 0.977, т. е. почти не изменилась
Для beta_1 = 0.999 точность 0.970

Зафиксирована beta_1 = 0.9 и поварьирована beta_2.
beta_2 = 0.9, 0.8 ; acc = 0.978

От малых изменений точность практически не меняется. От больших меняется незначительно либо в худшую сторону. Параметры по-умолчанию оптимальны

Исследуем оптимизатор SGD — стохастический градиентный спуск.

Точность составила 0.911
По умолчанию learning_rate=0.01.
При 0.001 точность 0.370
При 0.1 точность 0.961
При 0.2 точность 0.971
При 0.5 точность 0.975
При повышении скорости обучения точность растет. Оптимальны значеня 0.5-0.9 при которых точность ~0.975

Исследуем оптимизатор RMSprop
При скорости обучения 0.001(по-умолчанию) точность составила 0.978
При 0.01 точность 0.968
При 0.1 точность 0.878
При 0.001 точность 0.945

Значение по-умолчанию оптимально

Реализована загрузка пользовательских изображений.

При проверке разных изображений каждый раз переобучать модель было бы неэффективно. По этому модель сохраняется в файл и генерируется с нуля только если файла с готовой еще нет. Сохранение и загрузка модели реализована средствами Keras.

Для управления написан интерфейс командной строки. Для него использована библиотека argparse. Парсятся два параметра — путь до модели(по-умолчанию midel.tf) и путь до файла с изображением.

Для подачи а вход изображения img.png и прогон через модель my_model.tf команда выглядит следующим образом:

```
lab4 -i img.png -m my_model.tf
```

При этом если возможно будет загружена модель my_model.tf, если нет — создана и сохранена, после чего на ее вход подастся изображение, результат будет выведен в стандартный поток вывода.
Параметр «-m» можно опустить, тогда в качестве имени модели автоматически будет использовано «model.tf»
Если изображение не указано или не удается его загрузить происходит только генерация модели(при необходимости).

Для загрузки изображения используется OpenCV. Изображение загружается, масштабируется до размера 28*28 и обесцвечивается(gray-scale).

Результаты тестирования(входные изображения нарисованы в paint):

1. Входное изображение:

   Результат:
   5 : 1.0
   6 : 1.3085669e-22
   0 : 0.0
   1 : 0.0
   2 : 0.0
   3 : 0.0
   4 : 0.0
   7 : 0.0
   8 : 0.0
   9 : 0.0
2. Входное изображение:

Результат:
2 : 1.0
0 : 0.0
1 : 0.0
3 : 0.0
4 : 0.0
5 : 0.0
6 : 0.0
7 : 0.0
8 : 0.0
9 : 0.0

3. Входное изображение:

Результат:
4 : 1.0
0 : 0.0
1 : 0.0
2 : 0.0
3 : 0.0
5 : 0.0
6 : 0.0
7 : 0.0
8 : 0.0
9 : 0.0

Выводы.

Была обучена модель для распознавания рукописных цифр.
Были проведены эксперименты с различными оптимизаторами и их параметрами. По результатам выяснено, что значения по-умолчанию зачастую дают лучший результат. Был написан консольный интерфейс для прогона любых изображений, а также механизм «ленивого» обучения, т. е. только если этого не было сделано ранее.