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

Студент гр. 8383 _____ Шишкин И.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

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
- Написать функцию, позволяющая загружать изображение пользователя и классифицировать его

Выполнение работы.

Был загружен датасет из keras. Так как изображение хранится в виде матрицы 28x28, в которой каждый элемент – это цвет, то, чтобы значения распознавались нейронной сетью, нужно сделать, чтобы эти значения были не от 0 до 255, а от 0 до 1, поэтому эти значения делятся на 255. Затем, train_labels и test_labels преобразуются к массиву (если цифра равна 7, то массив будет выглядеть следующим образом: [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]).

Была создана модель №1 (листинг 1), состоящая из одного скрытого слоя, на котором 256 нейронов и функция активации ReLu. На выходном слое 10 нейронов, так как на выходе получаются цифры от 0 до 9. Используется функция активации softmax, чтобы на выходе получились значения в виде вероятностей. Оптимизатор – adam, функция потерь – categorical_crossentropy. Количество эпох равно 5.

Листинг 1 – Модель №1

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

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

Исследование оптимизаторов

Точность с такой моделью уже больше 95% (в среднем около 98).

Всего доступно 7 оптимизаторов:

- `keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False, clipnorm=1.0, clipvalue=0.5)` — стохастический градиентный спуск.
- `keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0, clipnorm=1., clipvalue=0.5)` — делит скорость обучения для веса на скользящее среднее значение последних градиентов этого веса.
- `keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0, clipnorm=1., clipvalue=0.5)` — скорость обучения параметра (веса) зависит от частоты его обновления: чем чаще обновляется параметр, тем меньше скорость его обучения.
- `keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0, clipnorm=1., clipvalue=0.5)` — расширение Adagrad, в котором скорости обучения изменяются на основе движущегося окна обновлений градиентов, а не на основе значений всех прежних градиенты, как в Adagrad. Таким образом, Adadelta продолжает учиться, даже если было сделано много обновлений. В оригинальной версии Adadelta не нужно устанавливать начальную скорость обучения. В версии Keras начальную скорость обучения можно задать, отличной от 0.01.
- `keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False, clipnorm=1., clipvalue=0.5)` — вариант стохастической оптимизации.
- `keras.optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, clipnorm=1., clipvalue=0.5)` — вариант алгоритма Adam, основанный на бесконечной норме.

- `keras.optimizers.Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, schedule_decay=0.004, clipnorm=1., clipvalue=0.5)` — алгоритм Адама, использующий импульс Нестерова.

Исследуем следующие параметры, использованные в оптимизаторах: lr – learning rate – скорость обучения, epsilon – константа для числовой устойчивости.

В модели №1 был использован оптимизатор Adam со скоростью обучения 0.001. Для модели №2 поменяем скорость обучения на 0.01 (листинг 2).

Листинг 2 – Модель №2

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность в таком случае упала на 1%.

Для модели №3 изменим скорость обучения на 0.1 (листинг 3).

Листинг 3 – Модель №3

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.1), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность теперь в среднем равна 86%, что примерно на 10% меньше предыдущей модели.

Для модели №4 изменим скорость обучения на 0.0001 (листинг 4).

Листинг 4 – Модель №4

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность поднялась до 94%, что все равно меньше, чем в модели №1.

Изначально коэффициент epsilon=1e-7. В документации к оптимизатору Adam написано следующее: «The default value of 1e-7 for epsilon might not be a good default in general. For example, when training an Inception network on ImageNet a current good choice is 1.0 or 0.1.» Для модели №5 поменяем epsilon на 0.1 (листинг 5).

Листинг 5 – Модель №5

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001, epsilon=0.1), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность в данном случае понизилась до 90%.

Для модели №6 поменяем оптимизатор на SGD (листинг 6).

Листинг 6 – Модель №6

```
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность теперь равна в среднем 91%.

Для модели №7 поменяем оптимизатор на RMSprop (листинг 7).

Листинг 7 – Модель №7

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Данная модель себя показывает так же, как и №1.

Для модели №8 поменяем оптимизатор на Adagrad (листинг 8).

Листинг 8 – Модель №8

```
model.compile(optimizer=tf.keras.optimizers.Adagrad(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность у данной модели равна 93%.

Для модели №9 поменяем оптимизатор на Adadelta (листинг 9).

Листинг 9 – Модель №9

```
model.compile(optimizer=tf.keras.optimizers.Adadelta(learning_rate=1.0), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность данной модели примерное такая же, как у моделей №1 и №7.

Для модели №10 поменяем оптимизатор на Adamax (листинг 10).

Листинг 10 – Модель №10

```
model.compile(optimizer=tf.keras.optimizers.Adamax(learning_rate=0.002), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность с погрешностью около 0.005% такая же, как у моделей №1, 7, 9.

Для модели №11 поменяем оптимизатор на Nadam (листинг 11).

Листинг 11 – Модель №11

```
model.compile(optimizer=tf.keras.optimizers.Nadam(learning_rate=0.002), loss='categorical_crossentropy', metrics=['accuracy'])
```

Точность такая же хорошая, как у моделей №1, 7, 9 и 11.

Таким образом, лучшими оптимизаторами для данной модели являются Adam, RMSprop, Adadelta, Nadam.

Чтобы пользователь мог загружать свое изображение с цифрой, была реализована функция load_image (листинг 12).

Листинг 12 – Функция load_image

```
def load_image(filename):
    # load the image
    loaded_img = load_img(filename, color_mode="grayscale", target_size=(28, 28))
    # convert to array
    loaded_img = img_to_array(loaded_img)
    # reshape into a single sample with 1 channel
    loaded_img = loaded_img.reshape(1, 28, 28, 1)
    # prepare pixel data
    loaded_img = loaded_img.astype('float32')
    loaded_img = loaded_img / 255.0
    return loaded_img
```

Проверка со своими картинками цифр:

На первом рисунке (рис. 1) изображено число 7.

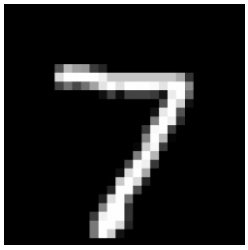

Рисунок 1 – Первая цифра

На втором рисунке (рис. 2) изображено число 1.

Рисунок 2 – Вторая цифра

На третьем рисунке (рис. 3) изображено число 5.

Рисунок 3 – Третья цифра

Вывод программы:

First sample image:

```
[[2.4982822e-07    2.2049104e-08    1.2523947e-05    2.4567065e-05
7.0802947e-10
  1.3009436e-07    5.6552246e-11    9.9996138e-01    4.3558020e-07
6.1815393e-07]]
Your num is 7
Second sample image:
[[1.52429158e-04 8.87716949e-01 1.16709946e-02 1.54441055e-02
  2.23367333e-05 4.39279480e-04 3.36485420e-04 1.58287045e-02
  6.55303597e-02 2.85845180e-03]]
Your num is 1
Third sample image:
[[1.5443545e-06    2.2868151e-03    5.5655015e-05    5.8180187e-03
1.7381423e-06
  9.9097508e-01    2.9329698e-05    1.7667857e-08    7.5914379e-04
7.2752977e-05]]
Your num is 5
```

Видно, что программа почти со 100% вероятностью определила числа 7 и 5. В цифре 1 она была уверена на 89%.

Выводы.

Была создана классификация черно-белых изображений рукописных цифр. Найдена архитектура сети, при которой точность не менее 95%. Исследовано влияние различных оптимизаторов, а также их параметров, на процесс обучения. Написана функция, которая позволяет загружать пользовательское изображение.