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 _____ Киреев К.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы

Распознавание рукописных символов.

Набор данных содержит 60,000 изображений для обучения и 10,000 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

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

Ход работы

Исследование архитектуры сети

Была реализована архитектура нейронной сети, указанная в методических материалах.

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

При данной архитектуре нейронной сети точность результатов достигла 97.76%, что удовлетворяет требованиям лабораторной работы. Архитектура была оставлена без изменений.

```
test_loss:  0.07469
test_acc: 97.76%
```

Исследование влияния оптимизаторов и их параметров

Далее были рассмотрены различные оптимизаторы с разными параметрами и их влияние на точность предсказаний нейронной сети.

- ***SGD***

  Стохастический оптимизатор градиентного спуска. Включает в себя поддержку импульса, затухания скорости обучения и импульса Нестерова.

  - *learning_rate*: скорость обучения
  - *momentum*: параметр, ускоряющий SGD в соответствующем направлении и гасящий колебания
  - *nesterov*: применять ли импульс Нестерова

`optimus = opti.SGD(learning_rate=0.01, nesterov=False)`	`test_loss:  0.32525` `test_acc: 91.07%`
`optimus = opti.SGD(learning_rate=0.1, nesterov=False)`	`test_loss:  0.13589` `test_acc: 96.06%`
`optimus = opti.SGD(learning_rate=0.1, nesterov=True)`	**`test_loss:  0.136`** **`test_acc: 96.12%`**

- ***RMSProp***

  RMSProp оптимизатор. Рекомендуется оставить параметры этого оптимизатора на значениях по умолчанию (за исключением скорости обучения, которая может быть свободно настроена).

  - *learning_rate*: скорость обучения
  - *rho*: коэффициент затухания скользящего среднего

optimus = opti.RMSprop(lr=0.001, rho=0.9)	**test_loss: 0.0714** **test_acc: 97.96%**
optimus = opti.RMSprop(lr=0.01, rho=0.9)	test_loss: 0.21163 test_acc: 96.18%
optimus = opti.RMSprop(lr=0.001, rho=0.6)	test_loss: 0.08186 test_acc: 97.70%
optimus = opti.RMSprop(lr=0.01, rho=0.6)	test_loss: 0.20385 test_acc: 97.40%

- ***Adagrad***

  Адаград — это оптимизатор, в котором скорость обучения зависит от конкретных параметров, которые адаптированы к тому, как часто параметр обновляется во время обучения. Чем больше обновлений получает параметр, тем меньше скорость обучения.

  - *learning_rate*: уровень начального обучения

optimus = opti.Adagrad(learning_rate=0.01)	test_loss: 0.24117 test_acc: 93.31%
optimus = opti.Adagrad(learning_rate=0.1)	**test_loss: 0.0921** **test_acc: 97.03%**

- ***Adadelta***

  Adadelta — более надежное расширение Adagrad, которое адаптирует скорость обучения на основе скользящего окна обновления градиентов вместо того, чтобы накапливать все градиенты прошлых лет. Таким образом, Adadelta продолжает обучение даже тогда, когда сделано много обновлений. По сравнению с Adagrad, в оригинальной версии Adadelta нет необходимости устанавливать начальную скорость обучения. В этой версии, как и в большинстве других оптимизаторов Keras, можно установить начальную скорость обучения и коэффициент распада.

  - *learning_rate*: начальная скорость обучения, по умолчанию 1. Рекомендуется оставить значение по умолчанию

- *rho*: коэффициент распада ададелты, соответствующий доле градиента, который необходимо сохранять на каждом шаге

optimus = opti.Adadelta(learning_rate=1.0, rho=0.95)	**test_loss: 0.08035 test_acc: 97.54%**
optimus = opti.Adadelta(learning_rate=0.1, rho=0.95)	test_loss: 0.21815 test_acc: 93.76%
optimus = opti.Adadelta(learning_rate=1.0, rho=0.6)	test_loss: 0.09865 test_acc: 97.19%

- ***Adam***

  RMSProp с моментами, получаемыми из предыдущих эпох.

  - *learning_rate*: скорость обучения
  - *beta_1*: обычно близко к 1
  - *beta_2*: обычно близко к 1

optimus = opti.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999)	**test_loss: 0.07005 test_acc: 97.79%**
optimus = opti.Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999)	test_loss: 0.15958 test_acc: 96.63%
optimus = opti.Adam(learning_rate=0.001, beta_1=0.99, beta_2=0.999)	test_loss: 0.07533 test_acc: 97.59%
optimus = opti.Adam(learning_rate=0.001, beta_1=0.999, beta_2=0.999)	test_loss: 0.09871 test_acc: 97.07%

- ***Adamax***

  Это вариант Adam, основанный на норме бесконечности.

  - *learning_rate*: скорость обучения
  - *beta_1*: обычно близко к 1
  - *beta_2*: обычно близко к 1

optimus = opti.Adamax(learning_rate=0.002, beta_1=0.9, beta_2=0.999)	**test_loss: 0.08871 test_acc: 97.40%**

`optimus = opti.Adamax(learning_rate=0.001, beta_1=0.9, beta_2=0.999)`	`test_loss: 0.11963` `test_acc: 96.58%`

- ***Nadam***

  Nesterov Adam оптимизатор. Так же, как Adam, по сути, является RMSprop с импульсом, так и Nadam это RMSprop с импульсом Нестерова.

  - *learning_rate*: скорость обучения
  - *beta_1*: обычно близко к 1
  - *beta_2*: обычно близко к 1

`optimus = opti.Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999)`	`test_loss: 0.07329` `test_acc: 97.72%`
`optimus = opti.Nadam(learning_rate=0.01, beta_1=0.999, beta_2=0.999)`	`test_loss: 0.13571` `test_acc: 97.75%`
`optimus = opti.Nadam(learning_rate=0.01, beta_1=0.9999, beta_2=0.9999)`	**`test_loss: 0.07958`** **`test_acc: 98.01%`**

Лучшую точность показала нейронная сеть с оптимизатором ***Nadam***, достигая точности 98.01%.

Конечные параметры нейронной сети:

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

optimus = opti.Nadam(learning_rate=0.01, beta_1=0.9999, beta_2=0.9999)
model.compile(optimizer=optimus, loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

Функция для загрузки пользовательских изображений

Была написана функция для загрузки изображений пользователя.

```
def load_image(filename):
    img = load_img(filename, color_mode='grayscale', target_size=(28, 28))
    img = img_to_array(img)
    img = img.reshape(1, 28, 28, 1)
    img = img.astype('float32') / 255.0
    return img

def pred_img(model, filename):
    img = load_image(filename)
    digit = np.argmax(model.predict(img), axis=-1)
    print(digit[0])

while(True):
    print('type img name to pred; 0 to exit')
    filename = input()
    if filename != '0':
        pred_img(model, filename)
    else:
        break
```

Пример работы нейронной сети

Изображения:

Вывод:

```
test_loss:  0.07961
test_acc: 98.07%
type img name to pred; 0 to exit
four.png
4
type img name to pred; 0 to exit
two.png
2
```

Выводы

В ходе выполнения лабораторной работы была создана нейронная сеть, которая классифицирует черно-белые изображения рукописных цифр по 10 категориям. Была найдена оптимальная конфигурация обучения сети, при которой сеть показывала максимальную точность. Была написана функция, позволяющая пользователю загрузить собственное изображение.