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

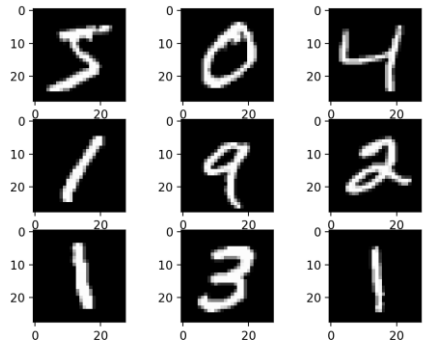
Студентка гр. 8383 _____ Максимова А.А.

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 для тестирования.

Задачи

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения
- Написать функцию, позволяющую загружать изображение пользователю 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

Основные теоретические положения

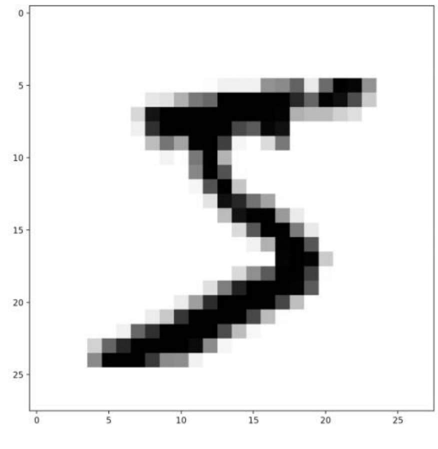
Представление графических данных

Каждое изображение из базы данных MNIST имеет размер 28 на 28 пикселей и представлено в градациях серого, где 0 - черный цвет, 255 - белый, а все, что между, это и есть градации серого. Для более глубокого понимания рассмотрим как хранится первое изображение из этой базы данных.

Одно изображение представляет из себя тензор 2 ранга, форма имеет значение (28, 28). Как видно из рисунка, предоставленного ниже, матрица хранит в себе целочисленные значения (от 0 до 255).

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

Для вывода двумерного массива в виде изображения можно воспользоваться командой imshow(), назначение которой состоит в представлении 2D-растров.



Задача классификации

Задача, в которой имеется множество объектов, где каждый объект, исходя из его свойств и параметров, можно отнести к конкретному классу. Таким образом, нейронная сеть, получая на вход объект, возвращает на выход вероятность (дискретную величину) его принадлежности к каждому из классов. В процессе обучения ИНС стремимся достигнуть результата, когда вероятность принадлежности к правильному классу имеет значение единицы, к другим классам - нуля.

Выполнение работы

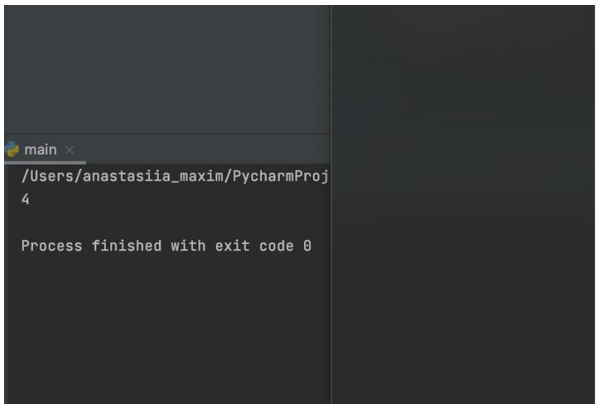
1. Были импортированы все необходимые для работы классы и функции.

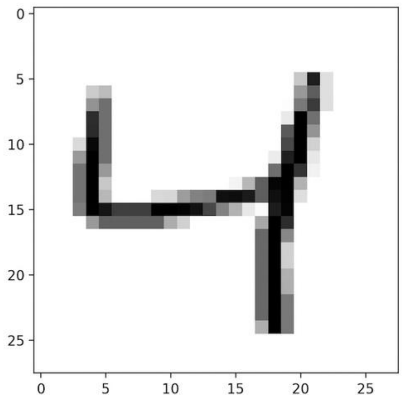
```
import tensorflow as tf
import matplotlib.pyplot as plt
import tensorflow.keras.optimizers as opt

from tensorflow.keras.layers import Dense, Flatten              #полносвязанный слой
from tensorflow.keras.models import Sequential                  #сеть прямого распространения
from tensorflow.keras.utils import to_categorical
```

2. Загрузка данных была выполнена с помощью функции load_data(), которая возвращает кортеж из 4 массивов NumPy: данные для обучения и тестирования.

```
mnist = tf.keras.datasets.mnist
#загрузка обучающих и тестовых данных - 4 массива NumPy
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Для проверки успешности сравнили тестовой изображение с его меткой.





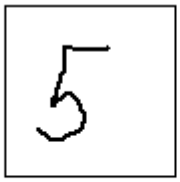
3. Была выполнена нормализация входных данных - массивов изображений, так как предпочтительно подавать нейронной сети стандартизированные данные, изменяющиеся в диапазоне от нуля до единицы.

```
#нормализация данных
train_images = train_images / 255.0
test_images = test_images / 255.0
print(train_images[0])
```

Как видно, нормализация выполнена успешно.

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.01176471 0.07058824 0.07058824 0.07058824 0.49411765 0.53333333
 0.68627451 0.10196078 0.65098039 1.         0.96862745 0.49803922
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.19215686 0.93333333 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.98431373
 0.36470588 0.32156863 0.32156863 0.21960784 0.15294118 0.
 0.         0.         0.         0.        ]
```

4. Кроме того, необходимо было подготовить правильный формат выходных значений. Правильные ответы находятся в диапазоне от 0 до 9, а нейронная сеть будет возвращать вектор из 10 элементов значения которых должны изменяться в диапазоне от 0 до 1. Таким образом, нужно было заменить исходные метки на вектора с нулевыми значениями и 1 в элементе, индекс которого соответствует значению, хранимому прежде в метке.

train_images[0] = 

train_labels[0] = 5 train_labels[0] = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

```
#изменение формата правильных ответов
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
print(train_labels)
```

Как можно заметить, преобразование из вектора в матрицу было выполнено успешно.

```
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]]
```

5. После была определена функция для создания модели ИНС прямого распространения, состоящей из 3 слоев: первый (входной) содержит $28^2 = 784$ (пикселя) нейрона, второй (скрытый) - 256 нейронов, функция активации - Relu: max(0, x); выходной слой - 10 нейронов (10 классов), функция активации - Softmax: $\frac{e^{x_i}}{\sum_{i=0}^{k} e^{x_i}}$, чтобы интерпретировать полученные результаты в терминах вероятности.

6. Передача графических данных нейронной сети

Так как на входной слой необходимо передавать вектор из 784 элементов, а не имеющуюся матрицу 28 на 28, то по этой причине используется специальный слой Flatten, выравнивающий входные данные (преобразующий формат изображения из 2D-массива в 1D-массив).

7. Были определены следующие параметры обучения сети: в качестве функции потерь используется "categorical_crossentropy", которую предпочтительно использовать в задачах классификации, когда количество классов больше двух, метрика - точность, оптимизатор - "adam".

```
def build_model():
    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'])
    return model
```

8. После было запущено обучение сети с помощью метода fit(адаптирует модель под обучающие данные).

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

9. Для проверки работоспособности программы она была запущена. В процессе обучения нейронной сети отображаются две величины: loss - потери сети и accuracy - точность на обучающих данных. Как видно, сеть уже показывает хорошие результаты.

```
Epoch 1/5
469/469 [==============================] - 1s 1ms/step - loss: 0.5243 - accuracy: 0.8555
Epoch 2/5
469/469 [==============================] - 1s 1ms/step - loss: 0.1407 - accuracy: 0.9589
Epoch 3/5
469/469 [==============================] - 1s 1ms/step - loss: 0.0927 - accuracy: 0.9729
Epoch 4/5
469/469 [==============================] - 1s 1ms/step - loss: 0.0680 - accuracy: 0.9813
Epoch 5/5
469/469 [==============================] - 1s 1ms/step - loss: 0.0516 - accuracy: 0.9852
```

10. Была выполнена проверка модели на распознавание контрольного набора. Как и ожидалось, величина точности на тестовых данных оказалась ниже примерно на 1% по сравнению с точностью на обучающих данных. Так как выбранная архитектура сети достигает заданной по условию точности 95%, то не будем вносить в нее изменения.

```
#проверка работы сети на контрольных данных
test_loss, test_acc = model.evaluate(test_images, test_labels)
print("test_loss: ", test_loss, "\n test_acc: ", test_acc)
```

```
test_loss:  0.07180648297071457
 test_acc:  0.9782999753952026
```

Исследование влияния различных оптимизаторов, а также их параметров на процесс обучения

Рассматриваем задачу минимизации целевой функции (функции потерь), имеющей форму суммы: $Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$,

где параметр w, минимизирующий $Q(w)$, следует оценить. Каждый член суммы Q_i обычно ассоциируется с $i-$ым наблюдением (объектом) в обучающем наборе данных.

Когда для минимизации вышеприведенной функции используется стандартный метод градиентного спуска, то итерации можно представить в виде следующей формулы:

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n$$

где $\eta -$ размер шага (скорость обучения).

Эксперимент 1

Название: SDG - Стохастический градиентный метод

Формула: $$w := w - \eta \nabla Q_i(w)$$

Суть: в отличие от классического градиентного спуска, вычисляющего градиент по всей обучающей выборке, берет случайным образом один объект из выборки и вычисляет по нему градиент, что позволяет сократить вычислительные ресурсы, достигая более быстрые итерации в обмен на более низкую сходимость. Кроме того, для повышения точности, можно использовать mini-batch, то есть вычислять градиент не по всему множеству обучающей выборки, но и не по одному объекту, что приводит к более гладкой сходимости.
Недостатки: огромная дисперсия оценки градиента.

Параметры:

learning_rate: float >= 0. Скорость обучения.
momentum: float >= 0. Параметр, ускоряющий SGD в соответствующем направлении и гасящий колебания.
nesterov: boolean. Применять ли импульс Нестерова?

Результаты применения:

№ опыта	learning_rate	momentum	nesterov	Точность
1	0.9	0.0	False	97.9%
2	0.5	0.0	False	97.3%
3	0.1	0.0	False	96.2%
4	0.01	0.0	False	90.9%
5	0.001	0.0	False	81.4%
6	0.9	0.0	True	98.0%
7	0.1	0.0	True	95.9%
8	0.001	0.0	True	91.1%
9	0.9	0.1	False	91.4%
10	0.9	0.5	False	92.4%
11	0.9	0.9	False	95.9%
12	0.9	0.9	True	95.9%

Выводы:

Как видно из первых 5 опытов, точность работу нейронной сети при использовании оптимизатора SDG увеличивается пропорционально увеличению параметра learning_rate, отвечающего за скорость обучения.

Применение импульса Нестерова, как видно из опытов 6-8 в сравнении с опытами 1, 3 и 5, не дало явного увеличения или уменьшения точности сети.

Как можно заметить из значений точности на опытах 9-11, увеличение значение параметра momentum приводит к увеличению точности работы сети, но в сравнение с опытом 1, значение точности стало меньше.

Эксперимент 2

Название: AdaGrad - Адаптивный градиентный алгоритм

Формула:

$$G_t = G_t + g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

где G_t – сумма квадратов обновлений параметра, ϵ – сглаживающий параметр, необходимый чтобы избежать деление на 0.

Суть: является модификацией стохастического алгоритма градиентного спуска с отдельной для каждого параметра скоростью обучения. Чем больше обновлений получает параметр, тем меньше скорость обучения.

Недостатки: G_t может увеличиваться сколько угодно, что через некоторое время приводит к слишком маленьким обновлениям и параличу алгоритма.

Параметры:

learning_rate: float >= 0. Уровень начального обучения.

Результаты применения:

№ опыта	learning_rate	Точность
1	0.9	90.4%
2	0.5	96.8%
3	0.1	97.4%
4	0.01	93.4%
5	0.001	87.5%

Выводы:

Как видно из результатов опытов, наилучшее значение точности достигается при некотором среднем значении скорости обучения.

Эксперимент 3

Название: RMSprop - Среднеквадратичное распространение

Формула:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Суть: скорость обучения настраивается для каждого параметра. Модификация идеи Adagrad: также обновляет меньше веса, которые слишком часто обновляются, но вместо полной суммы обновлений использует усредненный по истории квадрат градиента. Проще говоря, идея заключается в делении скорости обучения для весов на сгруппированные средние значения недавних градиентов этого веса.

Параметры:

learning_rate: float >= 0. Скорость обучения
rho: float >= 0.

Результаты применения:

№ опыта	learning_rate	rho	Точность
1	0.9	0.9	18.2%
2	0.5	0.9	38.5%
3	0.1	0.9	84.7%
4	0.01	0.9	97.6%
5	0.001	0.9	98%
6	0.001	0.5	97.5%
7	0.001	0.1	97.3%

Выводы:

Как видно из опытов 1-5, уменьшение скорости обучения увеличивает точность работы сети. Изменение второго параметра не дало улучшения.

Эксперимент 4

Название: Adam

Формула:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t},\ \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Суть: Представляет из себя объединение преимуществ двух других расширений стохастического градиентного спуска (AdaGrad и RMSProp), то есть сочетает идею накопления движения и идею более слабого обновления весов для типичных признаков.

Параметры:

learning_rate: float >= 0. Скорость обучения.
beta_1: float, 0 < beta < 1. Обычно близко к 1.
beta_2: float, 0 < beta < 1. Обычно близко к 1.
amsgrad: boolean. Применять ли AMSGrad вариант этого алгоритма из статьи «О конвергенции Adam и не только».

Результаты применения:

№ опыта	learning_rate	beta_1	beta_2	amsgrad	Точность
1	0.9	0.9	0.999	False	8.9%
2	0.5	0.9	0.999	False	9.9%
3	0.1	0.9	0.999	False	81.6%
4	0.01	0.9	0.999	False	97.1%
5	0.001	0.9	0.999	False	97.7%
6	0.001	0.9	0.999	True	97.8%
7	0.001	0.5	0.999	True	97.8%

8	0.001	0.9	0.5	True	96.5%
9	0.001	0.5	0.5	True	96.5%
10	0.001	0.1	0.999	True	97.7%
11	0.001	0.9	0.1	True	96.2%
12	0.001	0.1	0.1	True	95.9%

Выводы:

Как видно из опытов 1-5, увеличение скорости обучения влечет к потери точности нейронной сети.

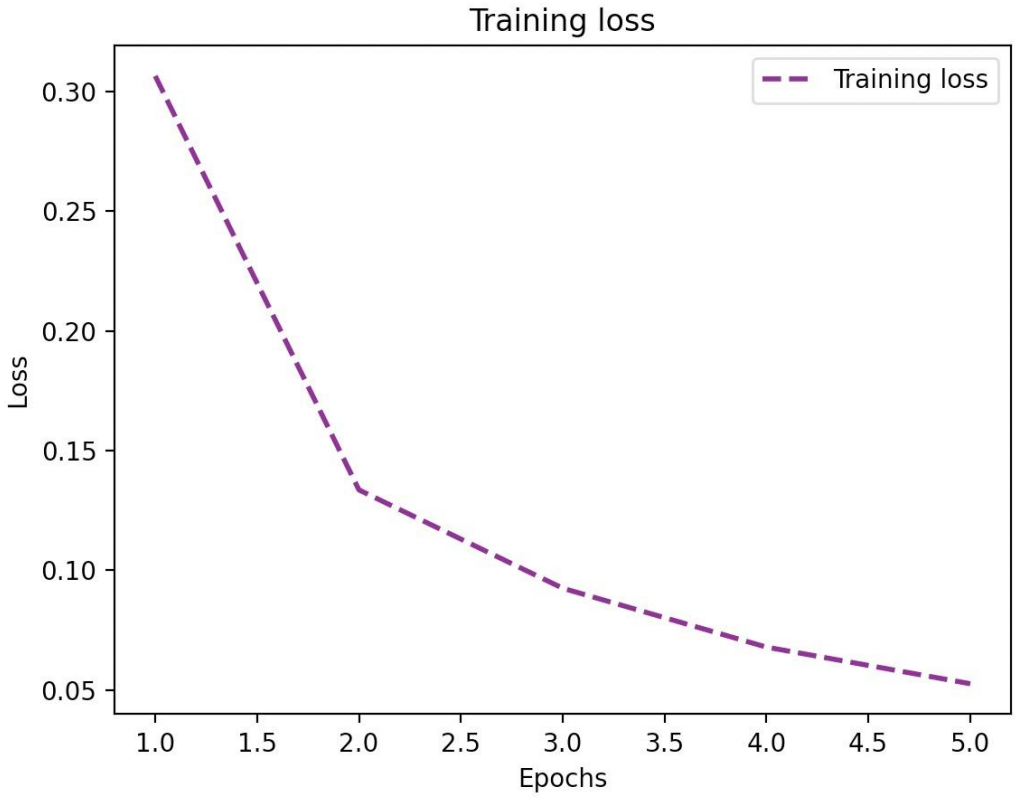
Из опыта 6 видно, что применение AMSgrad незначительно увеличивает точности ИНС.

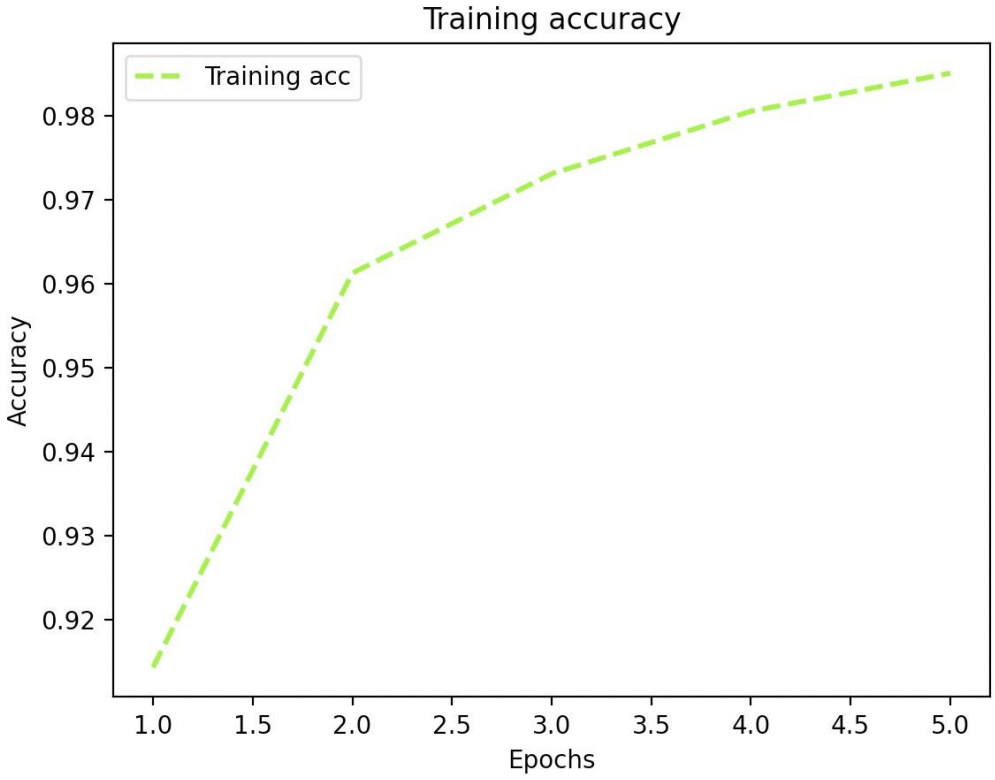
Как видно из опытов 6-8, уменьшение значений beta_1, beta_2 привело к несущественной потери точности сети, еще большее уменьшение данных параметров в опытах 10-12 влечет немного большие потери. Кроме того, видно, что уменьшение параметра beta_2 сильнее сказывается на точности сети.

Сравнение оптимизаторов

Наибольшие значения точности 97.8-98% были достигнуты при использовании оптимизаторов Adam и RMSprop при одинаковой скорости обучения равной 0.001, SDG при скорости обучения равной 0.9. С учетом того, что оптимизатор Adam является расширением стохастического градиентного спуска и RMSprop, то такое результат вполне логичен. Оставим оптимизатор Adam в качестве используемого, в силу его преимуществ - маленькие требования к памяти, вычислительно эффективен, хорошо подходит для задач, которые являются большими с точки зрения данных и параметров.

Ниже представлены графики точности и потерь сети на обучающих данных и значения данных показателей на контрольных данных для выбранной конфигурации сети.






```
test_loss:  0.07213353365659714
 test_acc:  0.9771999716758728
```

Написание программы, позволяющей загружать изображение пользователю и классифицировать его

Выполнение работы

1. Были импортированы все необходимые для работы классы и функции.

```
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image
from tensorflow.keras.models import load_model
```

2. Загрузка изображения была выполнена с помощью функции load_usr_img(usr_path), принимающей путь к изображению, считываемый в функции work_with_usr().

```
def load_usr_img(usr_path):
    img = Image.open(usr_path).convert('L')      #преобразовать в черно-белый
    img = img.resize((28, 28))                   #изменить размер
    img = np.array(img)                          #конвертируем в массив numpy
    img = 1 - img / 255                          #нормализация и инверсия
    return np.expand_dims(img, axis=0)           #predict ожидает трехмерный тензон
```

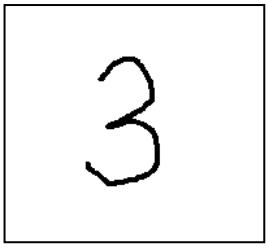
```
def work_with_usr():
    print("Enter the path to the image")
    path = input()
    while not path:
        print("Try it again")
        path = input()
    img = load_usr_img(path)
    model = load_model("ins.h5")                 #загрузка инс из файла
    prediction = model.predict(img)              #применение сети для распознавания

    plt.imshow(img[0], cmap=plt.cm.binary)
    plt.show()
    print("Network prediction: ", np.argmax(prediction))
```

Тестирование

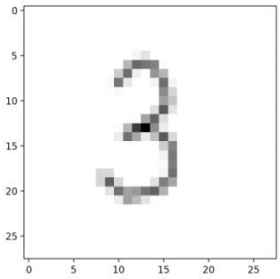
Производилось на картинках разных размеров, в различных цветах и толщине линий.

Тест 1

Входной изображение:



Обработанное изображение:

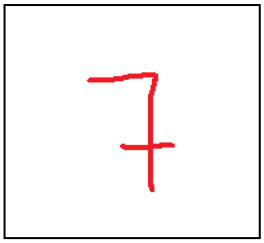


Предсказание сети:

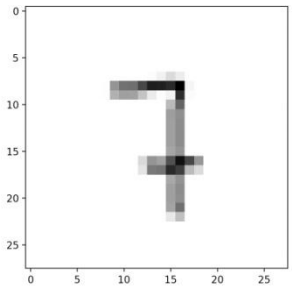
```
Network prediction:  3
Want to upload your own image? Enter: y or n.
```

Тест 2

Входной изображение:



Обработанное изображение:

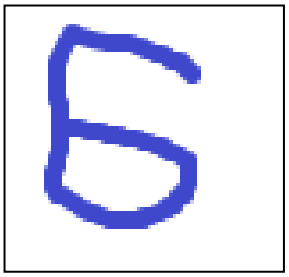


Предсказание сети:

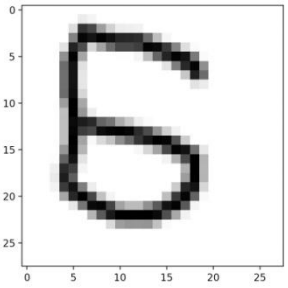
```
tests/img2.png
Network prediction:  7
Want to upload your own image? Enter: y or n.
```

Тест 3

Входной изображение:



Обработанное изображение:

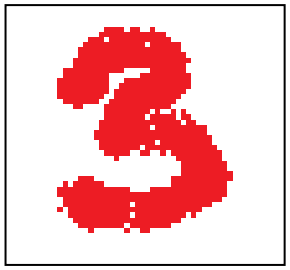


Предсказание сети:

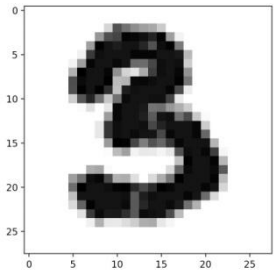
```
Network prediction:  3
```

Тест 4

Входной изображение:



Обработанное изображение:

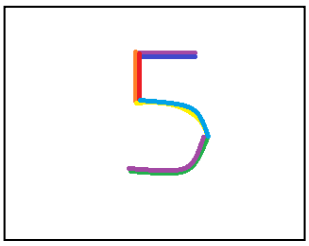


Предсказание сети:

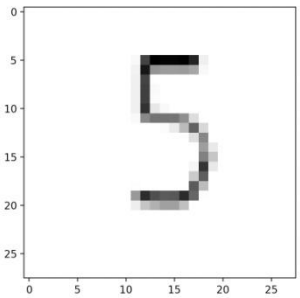
```
Network prediction:  3
```

Тест 5

Входной изображение:



Обработанное изображение:



Предсказание сети:

```
Enter the path to the image
tests/img5.png
Network prediction:  5
Want to upload your own image? Enter: y or n.
```

Выводы

В результате выполнения лабораторной работы был реализован механизм распознавания рукописных цифр из базы данных MNIST. Была построена модель, решающая задачу классификации. Было изучено влияние различных оптимизаторов и их параметров на процесс обучения нейронной сети. Были построены графики точности и потерь сети для выбранной конфигурации. Получилось достигнуть достаточно высокого значения точности сети - 98%.

Была написана и протестирована программа, реализующая загрузку пользовательского изображения и его классификации. В результате тестирования на пользовательских изображения ИНС дала правильный прогноз в 4 из 5 случаях.