МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 1

«Введение в нейронные сети»

по дисциплине «Обработка данных для построения систем искусственного интеллекта»

Выполнил студент группы
ПИЖ-б-о-21-1
Зиберов Александр
« » марта 2024 г.
Подпись студента_____
Работа защищена
« »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь, 2024

Цель работы:

Изучить основные понятия искусственного интеллекта на основе нейронных сетей, научиться создавать простейшие нейросети.

Выполнение работы:

Задание 1.

Создайте систему компьютерного зрения, которая будет определять тип геометрической фигуры. Используя подготовленную базу и шаблон ноутбука проведите серию экспериментов по перебору гиперпараметров нейронной сети, распознающей три категории изображений (треугольник, круг, квадрат).

1. Поменяйте количество нейронов в сети, используя следующие значения:

- один слой 10 нейронов
- один слой 100 нейронов
- один слой 5000 нейронов.

2. Поменяйте активационную функцию в скрытых слоях с `relu` на `linear`.

3. Поменяйте размеры batch_size:

- 10
- 100
- 1000

4. Выведите на экран получившиеся точности.

Всего должно получиться 18 комбинаций указанных параметров.

Создайте сравнительную таблицу по результатам проведенных тестов.

Задание

Создайте систему компьютерного зрения, которая будет определять тип геометрической фигуры. Используя подготовленную базу и шаблон ноутбука проведите серию экспериментов по перебору гиперпараметров нейронной сети, распознающей три категории изображений (треугольник, круг, квадрат).

1. Поменяйте количество нейронов в сети, используя следующие значения:

- один слой 10 нейронов
- один слой 100 нейронов
- один слой 5000 нейронов.

2. Поменяйте активационную функцию в скрытых слоях с `relu` на `linear`.
3. Поменяйте размеры batch_size:

- 10
- 100
- 1000

4. Выведите на экран получившиеся точности.

Всего должно получиться 18 комбинаций указанных параметров.

Создайте сравнительную таблицу по результатам проведенных тестов.

Комбинации параметров:

Количество нейров в сети	Batch Size	Активационная функция
10	10	relu
10	100	relu
10	1000	relu
100	10	relu
100	100	relu
100	1000	relu
5000	10	relu
5000	100	relu
5000	1000	relu
10	10	linear
10	100	linear
10	1000	linear
100	10	linear
100	100	linear
100	1000	linear
5000	10	linear
5000	100	linear
5000	1000	linear

Перед началом экспериментов, создадим нейронную сеть для распознавания и убедимся в ее работоспособности.

Рисунок 1 – Задание 1 (1)

Импорт библиотек:

```
[ ] # Подключение модуля для работы с файлами
    import os
    # Подключение библиотеки для работы с массивами
    import numpy as np
    # Подключение библиотеки для отрисовки изображений
    import matplotlib.pyplot as plt
    # Подключение функции для разделения данных на обучающий и тестовый наборы
    from sklearn.model_selection import train_test_split
    # Подключение утилит для to_categorical
    from tensorflow.keras import utils
    # Подключение класса для создания нейронной сети прямого распространения
    from tensorflow.keras.models import Sequential
    # Подключение класса для создания полносвязного слоя
    from tensorflow.keras.layers import Dense
    # Подключение оптимизатора
    from tensorflow.keras.optimizers import Adam
    # Подключение библиотеки для загрузки изображений
    from tensorflow.keras.preprocessing import image
    # Подключение функции для преобразования меток классов в категориальное представление
    from tensorflow.keras.utils import to_categorical

    WARNING:tensorflow:From c:\dev\ii\venv\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.s
```

Загрузка данных для Google Colab

Закомментировано, так как необходимые файлы были загружены локально.

```
[ ] # Загрузка датасета из облака
    # import gdown
    # gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l3/hw_light.zip', None, quiet=True)

    # Распаковываем архив hw_light.zip в папку hw_light
    # !unzip -q hw_light.zip
```

Рисунок 2 – Задание 1 (2)

Обработка изображения для последующей работы с ними

На данном этапе происходит добавление изображений в массив x_train и меток к ним в y_train.
Предварительно, из скачанного набора данных, больше половины было отведено для тренировки, а оставшиеся картинки - как тестовый набор.

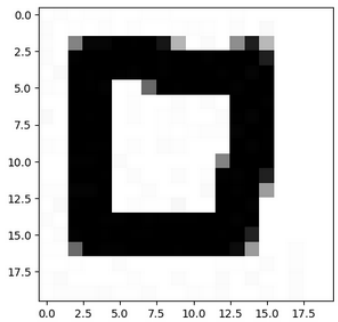
```
[ ]  # Обучающие данные
     base_dir = 'hw_light'
     x_train = []
     y_train = []
     img_height = 20
     img_width = 20

     # Перебор папок в директории базы
     for patch in os.listdir(base_dir):
         for img in os.listdir(os.path.join(base_dir, patch)):
             x_train.append(image.img_to_array(image.load_img(base_dir + '/' + patch + '/' + img,
                                                              target_size=(img_height, img_width),
                                                              color_mode='grayscale')))
             if patch == '0':
                 y_train.append(0)
             elif patch == '3':
                 y_train.append(1)
             else:
                 y_train.append(2)

     # Преобразование в numpy-массив загруженных изображений и меток классов
     x_train_org = np.array(x_train)
     y_train_org = np.array(y_train)

     print('Размер массива x_train:', x_train_org.shape)
     print('Размер массива y_train:', y_train_org.shape)

     plt.show()

     # Разделение данных на обучающий и тестовый наборы
     x_train_org, x_test_org, y_train_org, y_test_org = train_test_split(x_train_org, y_train_org, test_size=0.2,
                                                                         random_state=42)
```

```
Размер массива x_train: (302, 20, 20, 1)
Размер массива y_train: (302,)
```

Рисунок 3 – Задание 1 (3)

Проверка добавленных изображений и меток на примере одного из них:

```
[ ]  # Номер картинки
     n = 50

     # Отрисовка картинки
     plt.imshow(x_train_org[n], cmap='gray')

     # Вывод n-й картинки
     plt.show()

     # Вывод метки класса для n-го изображения
     print(y_train_org[n])
```




```
2
```

Изменение формы входных картинок в одномерную последовательность чисел (избавляемся от сложности данных):

```
[ ]  # Изменение формы входных картинок с 28x28 на 302
     # Первая ось остается без изменения, остальные складываются в вектор
     x_train = x_train_org.reshape(x_train_org.shape[0], -1)
     x_test = x_test_org.reshape(x_test_org.shape[0], -1)

     # Проверка результата
     print(f'Форма обучающих данных: {x_train_org.shape} -> {x_train.shape}')
     print(f'Форма тестовых данных: {x_test_org.shape} -> {x_test.shape}')
```

```
Форма обучающих данных: (241, 20, 20, 1) -> (241, 400)
Форма тестовых данных: (61, 20, 20, 1) -> (61, 400)
```

Нормализуем данные (0-255 -> 0-1)

```
[ ]  # Нормализация входных картинок
     # Преобразование x_train в тип float32 (числа с плавающей точкой) и нормализация
     x_train = x_train.astype('float32') / 255.

     # Преобразование x_test в тип float32 (числа с плавающей точкой) и нормализация
     x_test = x_test.astype('float32') / 255.
```

Рисунок 4 – Задание 1 (4)

Задание константы количества распознаваемых классов

В нашем случае, классов будет три: круг, треугольник и квадрат.

```
[ ]  # Задание константы количества распознаваемых классов
     CLASS_COUNT = 3
```

Преобразование ответов (меток) в формат one_hot_encoding

Это значит, что каждое число будет представлять собой последовательность (вектор) значений **0** или **1**. Последовательность будет длиной **3**, потому что всего существует **3** фигуры на распознавание. В векторе one hot encoding везде стоят нули, кроме позиции самой метки.

```
[ ]  # Преобразование ответов в формат one_hot_encoding
     y_train = utils.to_categorical(y_train_org, CLASS_COUNT)
     y_test = utils.to_categorical(y_test_org, CLASS_COUNT)

     # Вывод формы y_train
     print(y_train.shape)
     # Вывод примера одного выходного вектора
     print(y_train[0])
```

```
(241, 3)
[0. 1. 0.]
```

Просмотр меток элементов

```
[ ]  # Вывод формы массива меток
     print(y_train_org.shape)
     # Вывод метки, соответствующей 36-му элементу
     print(y_train_org[36])
```

```
(241,)
0
```

Рисунок 5 – Задание 1 (5)

Создание нейронной сети модели

Для примера, создадим модель с двумя полносвязными слоями.
В качестве параметра input_dim задаем число 400, просмотрев вывод формы данных после решейпинга.

> Форма обучающих данных: (302, 20, 20, 1) -> (302, **400**)

```
[ ]  # Создание последовательной модели
     model = Sequential()

     # Добавление полносвязного слоя на 1000 нейронов с relu-активацией
     model.add(Dense(1000, input_dim=400, activation='relu'))

     # Добавление полносвязного слоя с количеством нейронов по числу классов с softmax-активацией
     model.add(Dense(CLASS_COUNT, activation='softmax'))
```

```
WARNING:tensorflow:From c:\dev\ii\venv\lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecat
```

Компиляция модели, просмотр её структуры

```
[ ]  # Компиляция модели
     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

     # Вывод структуры модели
     print(model.summary())
```

```
WARNING:tensorflow:From c:\dev\ii\venv\lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1000)              401000

 dense_1 (Dense)             (None, 3)                 3003

=================================================================
Total params: 404003 (1.54 MB)
Trainable params: 404003 (1.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Рисунок 6 – Задание 1 (6)

Обучение созданной модели на данных

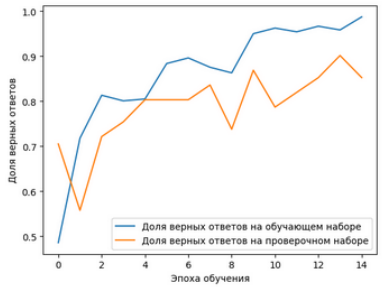
```
history = model.fit(x_train,        # обучающая выборка, входные данные
          y_train,        # обучающая выборка, выходные данные
          batch_size=10, # кол-во примеров, которое обрабатывает нейронка перед одним изменением весов
          epochs=15,      # количество эпох, когда нейронка обучается на всех примерах выборки
          validation_data=(x_test, y_test), # разделение для набора валидации
          verbose=1)      # 0 - не визуализировать ход обучения, 1 - визуализировать
```

```
Epoch 1/15
WARNING:tensorflow:From c:\dev\ii\venv\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is depre

WARNING:tensorflow:From c:\dev\ii\venv\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outsid

25/25 [==============================] - 1s 9ms/step - loss: 1.3305 - accuracy: 0.4855 - val_loss: 0.7979 - val_accuracy: 0.7049
Epoch 2/15
25/25 [==============================] - 0s 4ms/step - loss: 0.7013 - accuracy: 0.7178 - val_loss: 0.9603 - val_accuracy: 0.5574
Epoch 3/15
25/25 [==============================] - 0s 4ms/step - loss: 0.5157 - accuracy: 0.8133 - val_loss: 0.7088 - val_accuracy: 0.7213
Epoch 4/15
25/25 [==============================] - 0s 4ms/step - loss: 0.4589 - accuracy: 0.8008 - val_loss: 0.6210 - val_accuracy: 0.7541
Epoch 5/15
25/25 [==============================] - 0s 4ms/step - loss: 0.4349 - accuracy: 0.8050 - val_loss: 0.5825 - val_accuracy: 0.8033
Epoch 6/15
25/25 [==============================] - 0s 4ms/step - loss: 0.3388 - accuracy: 0.8838 - val_loss: 0.6423 - val_accuracy: 0.8033
Epoch 7/15
25/25 [==============================] - 0s 4ms/step - loss: 0.2910 - accuracy: 0.8963 - val_loss: 0.6012 - val_accuracy: 0.8033
Epoch 8/15
25/25 [==============================] - 0s 4ms/step - loss: 0.3099 - accuracy: 0.8755 - val_loss: 0.5854 - val_accuracy: 0.8361
Epoch 9/15
25/25 [==============================] - 0s 4ms/step - loss: 0.3270 - accuracy: 0.8631 - val_loss: 0.6661 - val_accuracy: 0.7377
Epoch 10/15
25/25 [==============================] - 0s 4ms/step - loss: 0.2010 - accuracy: 0.9502 - val_loss: 0.5037 - val_accuracy: 0.8689
Epoch 11/15
25/25 [==============================] - 0s 4ms/step - loss: 0.1583 - accuracy: 0.9627 - val_loss: 0.6882 - val_accuracy: 0.7869
Epoch 12/15
25/25 [==============================] - 0s 4ms/step - loss: 0.1727 - accuracy: 0.9544 - val_loss: 0.5119 - val_accuracy: 0.8197
Epoch 13/15
25/25 [==============================] - 0s 4ms/step - loss: 0.1318 - accuracy: 0.9668 - val_loss: 0.5190 - val_accuracy: 0.8525
Epoch 14/15
25/25 [==============================] - 0s 4ms/step - loss: 0.1230 - accuracy: 0.9585 - val_loss: 0.4709 - val_accuracy: 0.9016
Epoch 15/15
25/25 [==============================] - 0s 4ms/step - loss: 0.0716 - accuracy: 0.9876 - val_loss: 0.5449 - val_accuracy: 0.8525
```

Рисунок 7 – Задание 1 (7)

Вывод графика точности на обучающей и проверочной выборках

```
# Визуализация точности на обучающей выборке
plt.plot(history.history['accuracy'],
         label='Доля верных ответов на обучающем наборе')

# Визуализация точности на проверочной выборке
plt.plot(history.history['val_accuracy'],
         label='Доля верных ответов на проверочном наборе')

# Отрисовка подписей осей
plt.xlabel('Эпоха обучения')
plt.ylabel('Доля верных ответов')

# Отрисовка легенды
plt.legend()

# Вывод графика
plt.show()
```



Вывод графика ошибки на обучающей и проверочной выборках

```
# Вывод графика ошибки
plt.plot(history.history['loss'],
         label='Ошибка на обучающем наборе')
plt.plot(history.history['val_loss'],
         label='Ошибка на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Ошибка')
plt.legend()
plt.show()
```

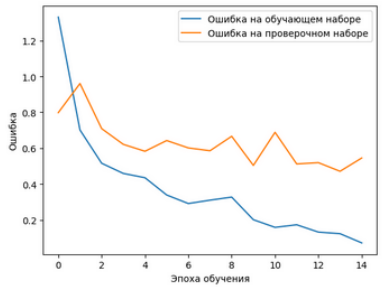
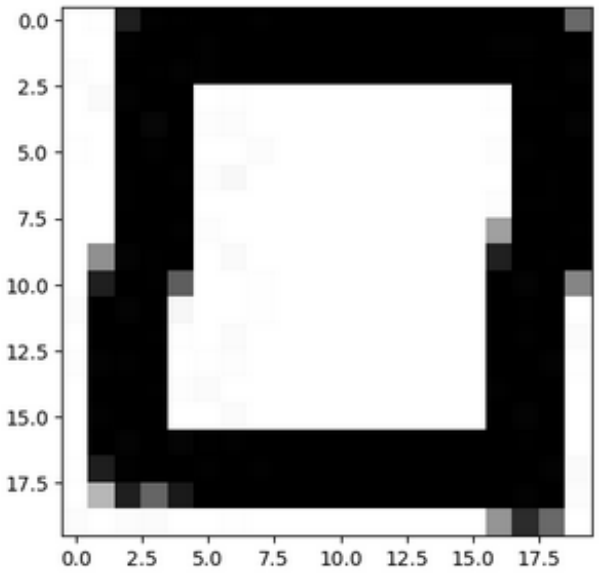


Рисунок 8 – Задание 1 (8)

Тестируем модель!

Выберем рандомную картинку из набора тестовхых данных для распознавания.

```
[ ] # Номер тестовой цифры, которую будем распознавать
    n_rec = 50

    # Отображение картинки из тестового набора под номером n_rec
    plt.imshow(x_test_org[n_rec], cmap='gray')
    plt.show()

    # Выбор нужной картинки из тестовой выборки
    x = x_test[n_rec]

    # Проверка формы данных
    print(x.shape)

    # Добавление одной оси в начале, чтобы нейронка могла распознать пример
    # Массив из одного примера, так как нейронка принимает именно массивы примеров (батчи) для распознавания
    x = np.expand_dims(x, axis=0)

    # Проверка формы данных
    print(x.shape)

    # Распознавание примера
    prediction = model.predict(x)

    # Вывод результата - вектор из 10 чисел
    print(prediction)
    sum(prediction[0])
```




```
(400,)
(1, 400)
1/1 [==============================] - 0s 56ms/step
[[1.5443405e-06 8.0459969e-08 9.9999833e-01]]
0.9999999558704502
```

Рисунок 9 – Задание 1 (9)

Результат:

```
[ ] # Получение и вывод индекса самого большого элемента (это значение цифры, которую распознала сеть)
    pred = np.argmax(prediction)

    match pred:
        case 0:
            print(f'Распознан круг!')

        case 1:
            print(f'Распознан треугольник!')

        case 2:
            print(f'Распознан квадрат!')

        case _:
            print("Нет совпадений!")

    # Вывод правильного ответа для сравнения
    match y_test_org[n_rec]:
        case 0:
            print(f'Правильный ответ: круг!')

        case 1:
            print(f'Правильный ответ: треугольник!')

        case 2:
            print(f'Правильный ответ: квадрат!')

        case _:
            print("Нет совпадений!")
```

```
Распознан квадрат!
Правильный ответ: квадрат!
```

Рисунок 10 – Задание 1 (10)

Теперь эксперименты!

Для перебора параметров тренировки создадим функцию, на входе которой будет:

- Количество нейронов
- Активационная функция
- Batch Size (Размер пакета)

На выходе же получаем аккуратность натренированной модели.

А для создания таблицы воспользуемся библиотекой pandas, взяв список словарей результатов как DataFrame.

```
[ ] import pandas as pd

    epoch_count = 15

    # Функция тренировки модели с разными параметрами
    def train_model(neurons, activation_func, batch_size):
        model = Sequential([
            Dense(neurons, input_dim=400, activation=activation_func),
            Dense(3, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        history = model.fit(x_train, y_train, epochs=epoch_count, batch_size=batch_size, verbose=0, validation_data=(x_test, y_test))


        # Создаем холст для двух графиков
        fig, axs = plt.subplots(1, 2, figsize=(15, 5))

        # График точности
        axs[0].plot(history.history['accuracy'], label='Доля верных ответов на обучающем наборе')
        axs[0].plot(history.history['val_accuracy'], label='Доля верных ответов на проверочном наборе')
        axs[0].set_xlabel('Эпоха обучения')
        axs[0].set_ylabel('Доля верных ответов')
        axs[0].legend()
        axs[0].set_title(f"Количество нейронов в сети: {neurons},\nBatch Size: {batch_size},\nАктивационная функция: {activation_func}")

        # График ошибки
        axs[1].plot(history.history['loss'], label='Ошибка на обучающем наборе')
        axs[1].plot(history.history['val_loss'], label='Ошибка на проверочном наборе')
        axs[1].set_xlabel('Эпоха обучения')
        axs[1].set_ylabel('Ошибка')
        axs[1].legend()
        axs[1].set_title(f"Количество нейронов в сети: {neurons},\nBatch Size: {batch_size},\nАктивационная функция: {activation_func}")

        plt.show()

        _, accuracy = model.evaluate(x_test, y_test)
        return accuracy

    # Параметры для перебора из условия задания
    neurons_list = [10, 100, 5000]
    activation_funcs = ['relu', 'linear']
    batch_sizes = [10, 100, 1000]

    results = []

    # Перебор комбинаций параметров и обучение модели
    for batch_size in batch_sizes:
        for neurons in neurons_list:
            for activation_func in activation_funcs:
                accuracy = train_model(neurons, activation_func, batch_size)
                results.append({'Нейроны': neurons,
                                'Активационная функция': activation_func,
                                'Batch Size': batch_size,
                                'Точность': accuracy})
```

Рисунок 11 – Задание 1 (11)

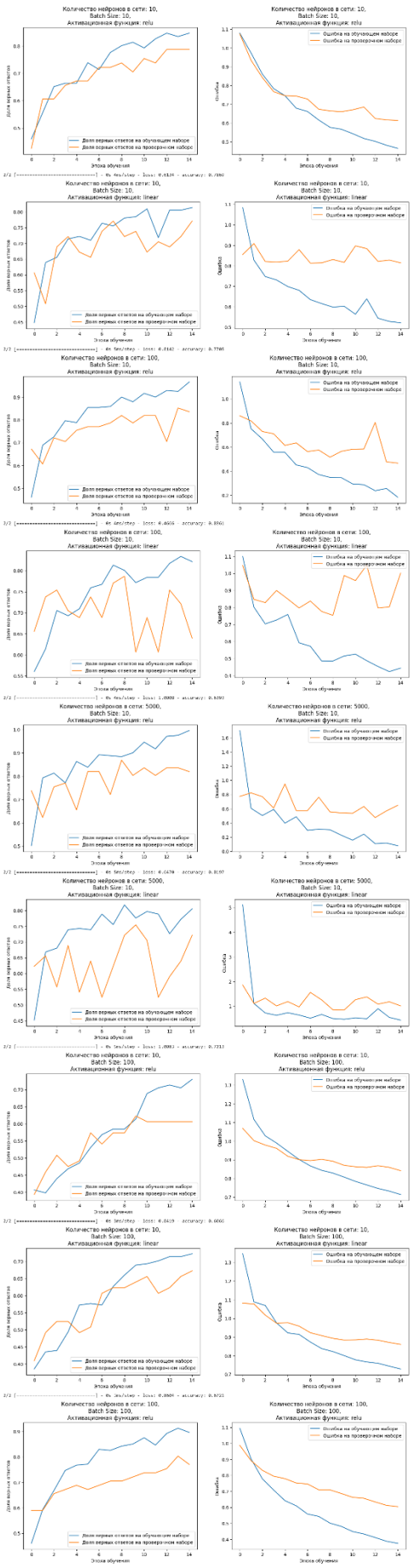


Рисунок 12 – Задание 1 (12)

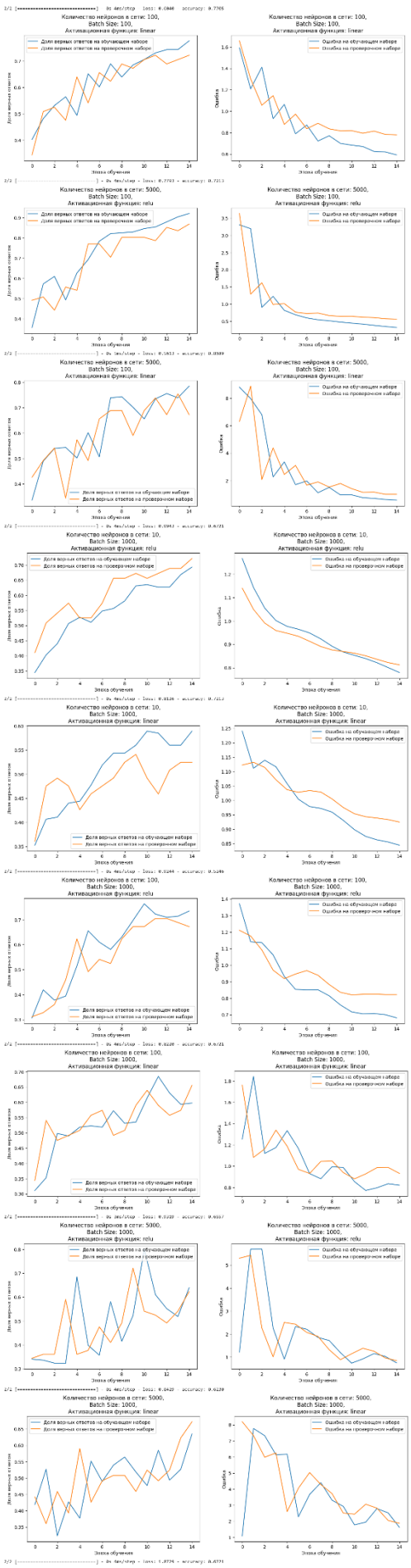


Рисунок 13 – Задание 1 (13)

Сравнительная таблица комбинаций параметров, полученные в процессе тренировки:

```
[ ] # Вывод результатов в виде таблицы
    results_df = pd.DataFrame(results)
    results_df.index = results_df.index + 1
    print(results_df)
```

	Нейроны	Активационная функция	Batch Size	Точность
1	10	relu	10	0.786885
2	10	linear	10	0.770492
3	100	relu	10	0.836066
4	100	linear	10	0.639344
5	5000	relu	10	0.819672
6	5000	linear	10	0.721311
7	10	relu	100	0.606557
8	10	linear	100	0.672131
9	100	relu	100	0.770492
10	100	linear	100	0.721311
11	5000	relu	100	0.868852
12	5000	linear	100	0.672131
13	10	relu	1000	0.721311
14	10	linear	1000	0.524590
15	100	relu	1000	0.672131
16	100	linear	1000	0.655738
17	5000	relu	1000	0.622951
18	5000	linear	1000	0.672131

Отсортируем по точности в порядке убывания:

```
[ ] # Сортировка столбцу 'Точность'
    results_df_sorted = results_df.sort_values(by='Точность', ascending=False)
    print(results_df_sorted)
```

	Нейроны	Активационная функция	Batch Size	Точность
11	5000	relu	100	0.868852
3	100	relu	10	0.836066
5	5000	relu	10	0.819672
1	10	relu	10	0.786885
9	100	relu	100	0.770492
2	10	linear	10	0.770492
13	10	relu	1000	0.721311
10	100	linear	100	0.721311
6	5000	linear	10	0.721311
8	10	linear	100	0.672131
12	5000	linear	100	0.672131
15	100	relu	1000	0.672131
18	5000	linear	1000	0.672131
16	100	linear	1000	0.655738
4	100	linear	10	0.639344
17	5000	relu	1000	0.622951
7	10	relu	100	0.606557
14	10	linear	1000	0.524590

Из таблицы видно:

- Для обоих активационных функций максимальная точность в данном случае достигается при использовании 5000 нейронов, следовательно увеличение количества нейронов в сети приводит к увеличению точности моделей.
- Увеличение значения Batch Size для некоторых комбинаций параметров приводит к увеличению точности, в то время как для других комбинаций результат может быть обратным.

Рисунок 14 – Задание 1 (14)

Задание 2.

Самостоятельно напишите нейронную сеть, которая может стать составной частью системы бота для игры в "Крестики-нолики". Используя подготовленную базу изображений, создайте и обучите нейронную сеть, распознающую две категории изображений: крестики и нолики. Добейтесь точности распознавания более 95% (accuracy).

Задание

Самостоятельно напишите нейронную сеть, которая может стать составной частью системы бота для игры в "Крестики-нолики". Используя подготовленную базу изображений, создайте и обучите нейронную сеть, распознающую две категории изображений: крестики и нолики. Добейтесь точности распознавания более 95% (accuracy)

```
[ ] # Вывод изображения в ноутбуке, а не в консоли или файле
    %matplotlib inline
```

Импорт библиотек:

```
[ ] # Подключение модуля для работы с файлами
    import os
    # Подключение библиотеки для работы с массивами
    import numpy as np
    # Подключение библиотеки для отрисовки изображений
    import matplotlib.pyplot as plt
    # Подключение функции для разделения данных на обучающий и тестовый наборы
    from sklearn.model_selection import train_test_split
    # Подключение утилит для to_categorical
    from tensorflow.keras import utils
    # Подключение класса для создания нейронной сети прямого распространения
    from tensorflow.keras.models import Sequential
    # Подключение класса для создания полносвязного слоя
    from tensorflow.keras.layers import Dense
    # Подключение оптимизатора
    from tensorflow.keras.optimizers import Adam
    # Подключение библиотеки для загрузки изображений
    from tensorflow.keras.preprocessing import image
    # Подключение функции для преобразования меток классов в категориальное представление
    from tensorflow.keras.utils import to_categorical
```

Загрузка данных для Google Colab

Закоментировано, так как необходимые файлы были загружены локально.

```
[ ] # Загрузка датасета из облака
    # import gdown
    # gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l3/hw_pro.zip', None, quiet=True)
```

```
[ ] # Распаковываем архив hw_light.zip в папку hw_light
    # !unzip -q hw_pro.zip
```

Рисунок 15 – Задание 2 (1)

Обработка изображения для последующей работы с ними

Как и в примере из практики, на данном этапе происходит добавление изображений в массив x_train и меток к ним в y_train. Предварительно, из скачанного набора данных, больше половины было отведено для тренировки, а оставшиеся картинки - как тестовый набор.

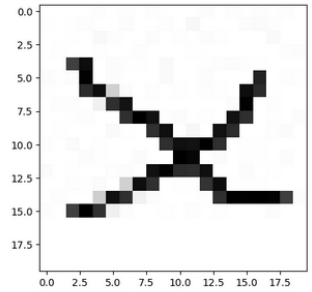
```
[ ] # Путь к директории с базой
    base_dir = 'hw_pro'
    # Создание пустого списка для загрузки изображений обучающей выборки
    x_train = []
    # Создание списка для меток классов
    y_train = []
    # Задание высоты и ширины загружаемых изображений
    img_height = 20
    img_width = 20
    # Перебор папок в директории базы
    for patch in os.listdir(base_dir):
        # Перебор файлов в папках
        for img in os.listdir(base_dir + '/' + patch):
            # Добавление в список изображений текущей картинки
            x_train.append(image.img_to_array(image.load_img(base_dir + '/' + patch + '/' + img,
                                                             target_size=(img_height, img_width),
                                                             color_mode='grayscale')))
            # Добавление в массив меток, соответствующих классам
            if patch == '0':
                y_train.append(0)
            else:
                y_train.append(1)

    # Преобразование в numpy-массив загруженных изображений и меток классов
    x_train_org = np.array(x_train)
    y_train_org = np.array(y_train)

    # Вывод размерностей
    print('Размер массива x_train:', x_train_org.shape)
    print('Размер массива y_train:', y_train_org.shape)

    # Разделение данных на обучающий и тестовый наборы
    x_train_org, x_test_org, y_train_org, y_test_org = train_test_split(x_train_org, y_train_org, test_size=0.2,
                                                                        random_state=42)
```

```
Размер массива x_train: (102, 20, 20, 1)
Размер массива y_train: (102,)
```

Рисунок 16 – Задание 2 (2)

Проверка добавленных изображений и меток на примере одного из них:

```
# Номер картинки
n = 20

# Отрисовка картинки
plt.imshow(x_train_org[n], cmap='gray')

# Вывод n-й картинки
plt.show()

# Вывод метки класса для n-го изображения
print(y_train_org[n])
```




```
1
```

Изменение формы входных картинок в одномерную последовательность чисел (избавляемся от сложности данных):

```
# Изменение формы входных картинок с 28x28 на 784
# первая ось остается без изменения, остальные складываются в вектор

x_train = x_train_org.reshape(x_train_org.shape[0], -1)
x_test = x_test_org.reshape(x_test_org.shape[0], -1)

# Проверка результата
print(f"Форма обучающих данных: {x_train_org.shape} -> {x_train.shape}")
print(f"Форма тестовых данных: {x_test_org.shape} -> {x_test.shape}")
```

```
Форма обучающих данных: (81, 20, 20, 1) -> (81, 400)
Форма тестовых данных: (21, 20, 20, 1) -> (21, 400)
```

Нормализуем данные (0-255 -> 0-1)

```
# Нормализация входных картинок
# Преобразование x_train в тип float32 (числа с плавающей точкой) и нормализация
x_train = x_train.astype('float32') / 255.

# Преобразование x_test в тип float32 (числа с плавающей точкой) и нормализация
x_test = x_test.astype('float32') / 255.
```

Рисунок 17 – Задание 2 (3)

Задание константы количества распознаваемых классов

В этом случае, классов будет два: крестик и нолик.

```
# Задание константы количества распознаваемых классов
CLASS_COUNT = 2
```

Преобразование ответов (меток) в формат one_hot_encoding

Это значит, что каждое число будет представлять собой последовательность (вектор) значений **0** или **1**. Последовательность будет длиной **3**, потому что всего существует **3** фигуры на распознавание. В векторе one hot encoding везде стоят нули, кроме позиции самой метки.

```
# Преобразование ответов в формат one_hot_encoding
y_train = utils.to_categorical(y_train_org, CLASS_COUNT)
y_test = utils.to_categorical(y_test_org, CLASS_COUNT)

# Вывод формы y_train
# 60 тысяч примеров, каждый длины 10 по числу классов
print(y_train.shape)
```

```
(81, 2)
```

Просмотр меток элементов

```
# Вывод примера одного выходного вектора
print(y_train[0])

# Вывод формы массива меток
print(y_train_org.shape)
```

```
[0. 1.]
(81,)
```

Создание нейронной сети модели

Создадим модель с полносвязным слоем с активационной функцией 'relu'.
В качестве параметра input_dim задаем число 400, просмотрев вывод формы данных после решейпинга.

```
Форма обучающих данных: (83, 20, 20, 1) -> (83, 400)
```

```
# Создание последовательной модели
model = Sequential()

# Добавление полносвязных слоев с активацией relu
model.add(Dense(100, input_dim=400, activation='relu'))
model.add(Dense(50, input_dim=400, activation='relu'))
model.add(Dense(10, input_dim=400, activation='relu'))

# Добавление полносвязного слоя с количеством нейронов по числу классов с sigmoid-активацией
model.add(Dense(CLASS_COUNT, activation='softmax'))
```

Рисунок 18 – Задание 2 (4)

Компиляция модели, просмотр её структуры

```
[ ] # Компиляция модели
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    # Вывод структуры модели
    print(model.summary())
```

```
Model: "sequential_25"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_80 (Dense)            (None, 100)               40100

 dense_81 (Dense)            (None, 50)                5050

 dense_82 (Dense)            (None, 10)                510

 dense_83 (Dense)            (None, 2)                 22

=================================================================
Total params: 45682 (178.45 KB)
Trainable params: 45682 (178.45 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Рисунок 19 – Задание 2 (5)

Обучение созданной модели на данных

Аккуратность на последних эпохах больше 95%.

```
history = model.fit(x_train,       # обучающая выборка, входные данные
          y_train,       # обучающая выборка, выходные данные
          batch_size=10, # кол-во примеров, которое обрабатывает нейронка перед одним изменением весов
          epochs=15,     # количество эпох, когда нейронка обучается на всех примерах выборки
          verbose=1, # 0 - не визуализировать ход обучения, 1 - визуализировать
          validation_data=(x_test, y_test)) # выбор валидационно

# Создаем холст для двух графиков
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# График точности
axs[0].plot(history.history['accuracy'], label='Доля верных ответов на обучающем наборе')
axs[0].plot(history.history['val_accuracy'], label='Доля верных ответов на проверочном наборе')
axs[0].set_xlabel('Эпоха обучения')
axs[0].set_ylabel('Доля верных ответов')
axs[0].legend()

# График ошибки
axs[1].plot(history.history['loss'], label='Ошибка на обучающем наборе')
axs[1].plot(history.history['val_loss'], label='Ошибка на проверочном наборе')
axs[1].set_xlabel('Эпоха обучения')
axs[1].set_ylabel('Ошибка')
axs[1].legend()

plt.show()
```

```
Epoch 1/15
9/9 [==============================] - 1s 21ms/step - loss: 0.7836 - accuracy: 0.4691 - val_loss: 0.6755 - val_accuracy: 0.6190
Epoch 2/15
9/9 [==============================] - 0s 5ms/step - loss: 0.7287 - accuracy: 0.4568 - val_loss: 0.6726 - val_accuracy: 0.6190
Epoch 3/15
9/9 [==============================] - 0s 5ms/step - loss: 0.6471 - accuracy: 0.7407 - val_loss: 0.7187 - val_accuracy: 0.3810
Epoch 4/15
9/9 [==============================] - 0s 6ms/step - loss: 0.6132 - accuracy: 0.5926 - val_loss: 0.6589 - val_accuracy: 0.3810
Epoch 5/15
9/9 [==============================] - 0s 5ms/step - loss: 0.5664 - accuracy: 0.7160 - val_loss: 0.5710 - val_accuracy: 0.9524
Epoch 6/15
9/9 [==============================] - 0s 5ms/step - loss: 0.5639 - accuracy: 0.6914 - val_loss: 0.5994 - val_accuracy: 0.5714
Epoch 7/15
9/9 [==============================] - 0s 5ms/step - loss: 0.4795 - accuracy: 0.9383 - val_loss: 0.4882 - val_accuracy: 0.9048
Epoch 8/15
9/9 [==============================] - 0s 5ms/step - loss: 0.4256 - accuracy: 0.9383 - val_loss: 0.4524 - val_accuracy: 0.9524
Epoch 9/15
9/9 [==============================] - 0s 7ms/step - loss: 0.3975 - accuracy: 0.9630 - val_loss: 0.4541 - val_accuracy: 0.9048
Epoch 10/15
9/9 [==============================] - 0s 8ms/step - loss: 0.3883 - accuracy: 0.9877 - val_loss: 0.3625 - val_accuracy: 0.9524
Epoch 11/15
9/9 [==============================] - 0s 6ms/step - loss: 0.2727 - accuracy: 0.9877 - val_loss: 0.3617 - val_accuracy: 0.9048
Epoch 12/15
9/9 [==============================] - 0s 5ms/step - loss: 0.2108 - accuracy: 1.0000 - val_loss: 0.3027 - val_accuracy: 0.9524
Epoch 13/15
9/9 [==============================] - 0s 5ms/step - loss: 0.1739 - accuracy: 0.9877 - val_loss: 0.2437 - val_accuracy: 0.9524
Epoch 14/15
9/9 [==============================] - 0s 6ms/step - loss: 0.1352 - accuracy: 1.0000 - val_loss: 0.2116 - val_accuracy: 0.9524
Epoch 15/15
9/9 [==============================] - 0s 5ms/step - loss: 0.1200 - accuracy: 1.0000 - val_loss: 0.1948 - val_accuracy: 1.0000
```

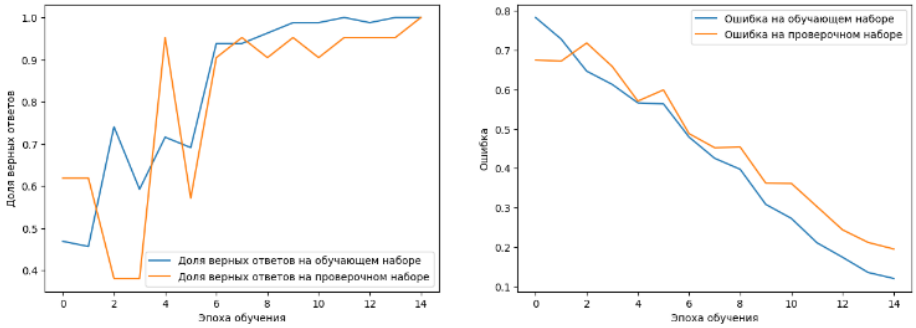
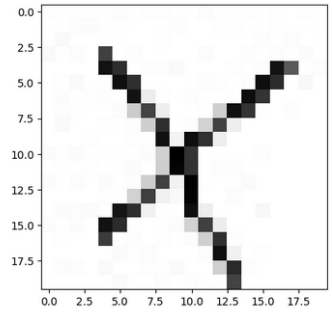


Рисунок 20 – Задание 2 (6)

Тестируем модель!

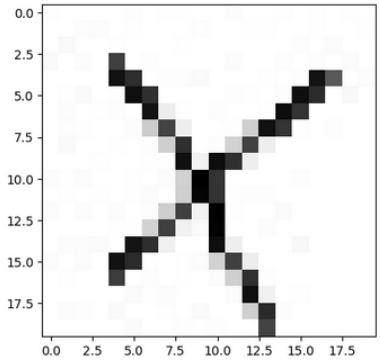
Выберем рандомную картинку из набора тестовых данных для распознавания.

```
[ ]  # Номер тестовой цифры, которую будем распознавать
     n_rec = 20

     # Отображение картинки из тестового набора под номером n_rec
     plt.imshow(x_test_org[n_rec], cmap='gray')
     plt.show()

     # Выбор нужной картинки из тестовой выборки
     x = x_test[n_rec]

     # Проверка формы данных
     print(x.shape)

     # Добавление одной оси в начале, чтобы нейронка могла распознать пример
     # Массив из одного примера, так как нейронка принимает именно массивы примеров (батчи) для распознавания
     x = np.expand_dims(x, axis=0)

     # Проверка формы данных
     print(x.shape)
```




```
(400,)
(1, 400)
```

Рисунок 21 – Задание 2 (7)




```
(400,)
(1, 400)
```

```
[ ]  # Распознавание примера
     prediction = model.predict(x)
     # Вывод результата - вектор из 10 чисел
     print(prediction)
     sum(prediction[0])

     # Получение и вывод индекса самого большого элемента (это значение цифры, которую распознала сеть)
     pred = np.argmax(prediction)


     match pred:
         case 0:
             print(f'Распознан нолик!')

         case 1:
             print(f'Распознан крестик!')

         case _:
             print("Нет совпадений!")

     # Вывод правильного ответа для сравнения
     match y_test_org[n_rec]:
         case 0:
             print(f'Правильный ответ: нолик!')

         case 1:
             print(f'Правильный ответ: крестик!')

         case _:
             print("Нет совпадений!")
```

```
1/1 [==============================] - 0s 51ms/step
[[0.04172273 0.9582773 ]]
Распознан крестик!
Правильный ответ: крестик!
```

Рисунок 22 – Задание 2 (8)

Задание 3.

Распознайте рукописную цифру, написанную на листе от руки. Последовательность шагов, следующая:

- На бумаге рисуем произвольную цифру (желательно нарисовать цифру размером не более 5 * 5 мм и без наклона. В занятии нейронка обучалась на цифрах американских студентов. Эти цифры были написаны на тетрадных листах в клетку и имели схожий размер).
- Фотографируем. Загружаем фото в Collaboratory.
- С помощью функции image.load_img(path, target_size=(28, 28), color_mode = ‘grayscale’) загружаем картинку в переменную.
- С помощью функции image.img_to_array(img) преобразуем изображение в numpy-массив.
- Выполняем инверсию цветов, нормирование и решейп массива.
- Выполняем распознавание собственной рукописной цифры.

Примечание: точность распознавания рукописных цифр может быть достаточно низкой, т.к. рукописные цифры после преобразований хоть и похожи на содержащиеся в базе, но могут отличаться по конфигурации, толщине линий и т.д.

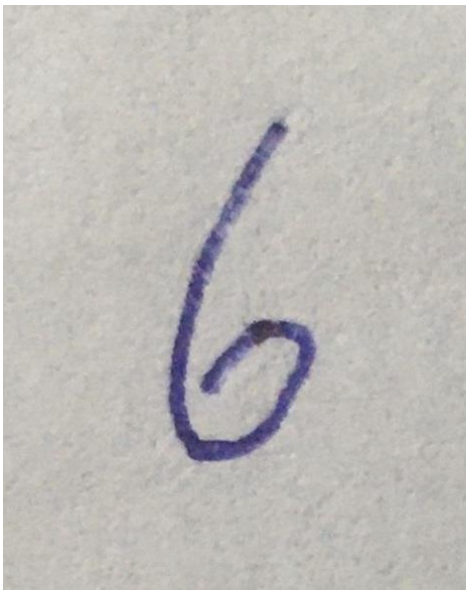


Рисунок 23 – Нарисованная цифра для распознавания

Задание

Распознайте рукописную цифру, написанную на листе от руки. Последовательность шагов следующая:

- На бумаге рисуем произвольную цифру (желательно нарисовать цифру размером не более 5 * 5 мм и без наклона. В занятии нейронка обучалась на цифрах американских студентов. Эти цифры были написаны на тетрадных листах в клетку и имели схожий размер).
- Фотографируем. Загружаем фото в Collaboratory.
- С помощью функции image.load_img(path, target_size=(28, 28), color_mode = 'grayscale') загружаем картинку в переменную.
- С помощью функции image.img_to_array(img) преобразуем изображение в numpy-массив.
- Выполняем инверсию цветов, нормирование и решейп массива.
- Выполняем распознавание собственной рукописной цифры.

Примечание: точность распознавания рукописных цифр может быть достаточно низкой, т.к. рукописные цифры после преобразований хоть и похожи на содержащиеся в базе, но могут отличаться по конфигурации, толщине линий и т.д.

```
[ ] # Вывод изображения в ноутбуке, а не в консоли или файле
    %matplotlib inline
```

Подключим библиотеки:

```
[ ] # Подключение библиотеки для работы с массивами
    import numpy as np
    # Подключение библиотек для отрисовки изображений
    import matplotlib.pyplot as plt

    # Подключение библиотеки для загрузки изображений
    from tensorflow.keras.preprocessing import image
    # Подключение библиотеки для загрузки готовых моделей
    from tensorflow.keras.models import load_model
```

Загрузим ранее полученную в практическом ноутбуке натренированную модель:

```
[ ] savedModel=load_model('model_full.h5')
    savedModel.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_25 (Dense)            (None, 800)               628000

 dense_26 (Dense)            (None, 400)               320400

 dense_27 (Dense)            (None, 10)                4010

=================================================================
Total params: 952410 (3.63 MB)
Trainable params: 952410 (3.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Рисунок 24 – Задание 3 (1)

Загрузим картинку, преобразуем её в массив и выведим для проверки:

```
[ ] # Функцией image загружаем изображение
    temp = image.load_img('cifra.jpg', target_size=(28, 28), color_mode = 'grayscale')

    # Переводим картинку в массив
    img_array = image.img_to_array(temp)

    # Выводим картинку для проверки
    plt.imshow(img_array, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1d4b4b08670>
```

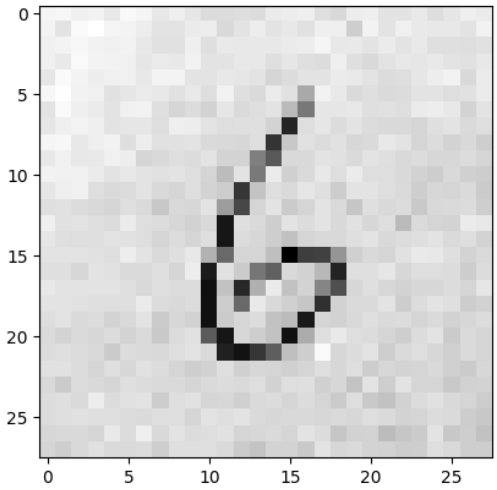


Рисунок 25 – Задание 3 (2)

Предобработка изображения и его распознавание

Проведем операции по инвертированию и нормализации изображения, запустим распознавание

```
[ ] # Инвертируем картинку
    inverted_img_array = 255 - img_array

    # Выводим картинку для проверки
    plt.imshow(inverted_img_array, cmap='gray')

    # # Добавление одной оси в начале
    inverted_img_array = np.expand_dims(inverted_img_array, axis=0)

    # Изменение формы входных картинок с 28x28 на 784
    img_array_reshaped = img_array.reshape(inverted_img_array.shape[0], -1)

    # Нормализация картинки
    # Преобразование x_train в тип float32 (числа с плавающей точкой) и нормализация
    img_array_reshaped = img_array_reshaped.astype('float32') / 255.
    print(img_array_reshaped.shape)

    # Распознавание картинки
    prediction = savedModel.predict(img_array_reshaped)

    # Вывод результата - вектор из 10 чисел
    print(prediction)

    # Получение и вывод индекса самого большого элемента
    pred = np.argmax(prediction)
    print(f'Распознана цифра: {pred}')
```

```
(1, 784)
1/1 [==============================] - 0s 45ms/step
[[3.4045431e-01 3.6438900e-07 1.2898462e-06 1.6242706e-05 3.6449865e-08
  1.8661158e-02 6.4059156e-01 2.7418172e-04 5.3755502e-07 3.5627394e-07]]
Распознана цифра: 6
```

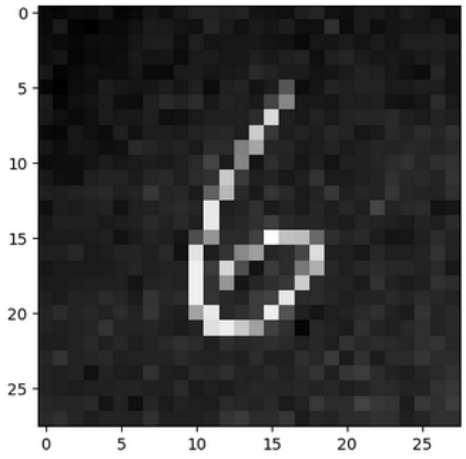


Рисунок 26 – Задание 3 (3)

Вывод: Изучены основные принципы и понятия искусственного интеллекта на основе нейронных сетей, включая возможные слои и активационные функции, подготовка данных и последующая тренировка модели на них, вывод и использование полученных результатов.