**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_Toc179878421)

[1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ 8](#_Toc179878422)

[1.1 Понятие нейросети и нейрона 8](#_Toc179878423)

[1.2 Функция активации 8](#_Toc179878424)

[1.3 Архитектуры нейронных сетей 9](#_Toc179878425)

[1.3.1 Полносвязные нейронные сети 9](#_Toc179878426)

[1.3.2 Сверточные нейронные сети 9](#_Toc179878427)

[1.3.3 Рекуррентные нейронные сети 9](#_Toc179878428)

[1.4 Плюсы и минусы нейросетей 10](#_Toc179878429)

[1.5 Выбор архитектуры нейросети 11](#_Toc179878430)

[1.6 Пользовательский интерфейс 11](#_Toc179878431)

[1.7 Использованное ПО 11](#_Toc179878432)

[1.8 Вывод 12](#_Toc179878433)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 13](#_Toc179878434)

[2.1 Общие задачи и результаты 13](#_Toc179878435)

[2.2 Построение модели нейронной сети 14](#_Toc179878436)

[2.2.1 Библиотека Keras 14](#_Toc179878437)

[2.2.2 Сбор данных для обучения 15](#_Toc179878438)

[2.2.3 Формат цифр MNIST 15](#_Toc179878439)

[2.2.4 Подготовка данных для обучения 16](#_Toc179878440)

[2.2.5 Построение модели нейросети 16](#_Toc179878441)

[2.3 Обучение и тестирование 18](#_Toc179878442)

[2.4 Обработка входных данных 20](#_Toc179878443)

[2.4.1 Правила построение формата MNIST 20](#_Toc179878444)

* + 1. [Правила построение формата MNIST 19](#_bookmark23)
    2. [Алгоритмы обработки изображения 19](#_bookmark24)

[ЗАКЛЮЧЕНИЕ 22](#_bookmark25)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 23](#_bookmark26)

# ВВЕДЕНИЕ

Мир технологий и гаджетов уже давно заменил многие привычные процессы. Теперь мы почти всегда можем обойтись без написания от руки, прогулок или чтения энциклопедий — достаточно просто погуглить информацию. Однако это не означает, что эти навыки стали ненужными. Напротив, современные технологические помощники значительно упрощают нашу жизнь, но если мы начинаем злоупотреблять ими, то теряем важное — перестаем тренировать наш главный инструмент — мозг. А без регулярных тренировок он начинает слабеть.

Одним из наиболее полезных навыков является устный счет, который прекрасно развивает наше мышление. Кажется, зачем учиться быстро считать в уме, если под рукой всегда есть калькулятор — на смартфоне, компьютере или даже в некоторых часах?

Тем не менее, мы недооцениваем этот навык. Умение складывать, умножать, вычитать и делить числа — это лишь часть его ценности. Устный счет помогает планировать семейный бюджет, делать более выгодные покупки, рассчитывать проценты по кредитам и переплаты. И это лишь небольшая часть повседневных ситуаций, где калькулятор не всегда поможет, ведь он не примет решение за вас.

Целью данного проекта является разработка консольного приложения, способного помочь решить данную проблему.

При разработке проекта были поставлены следующие задачи: разработка интуитивно понятного интерфейса, гибких настроек, позволяющих подстроить программу под текущий уровень, и систему логирования для отслеживания своего прогресса.

**1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ**

**Консольное приложение**

Консольное приложение — это программа, которая выполняется в текстовом режиме, используя командную строку или терминал для взаимодействия с пользователем. В таких приложениях ввод и вывод данных осуществляется через текстовые команды, а не через графический интерфейс.

**Интерфейс**

Для удобного использования программы необходимо было разработать удобный интерфейс, подходящий для консольного приложения. Он был создан на основе ветвления и управления при помощи пользовательского ввода.

Настройки программы

ффвфв

Использованное ПО

Разработка всех компонентов программы производилась на языке программирования Python как на одном из самых популярных и высокоуровневых. Для разработки интерфейса использовалась стандартная библиотека Tkinter. Для построения, обучения, тестирования нейросети использовалась высокоуровневая библиотека Keras. Также в ходе работы были задействованы следующие

дополнительные библиотеки для работы с изображениями и другими данными: Pillow, NumPy, OpenCV, SciPy.

Вывод

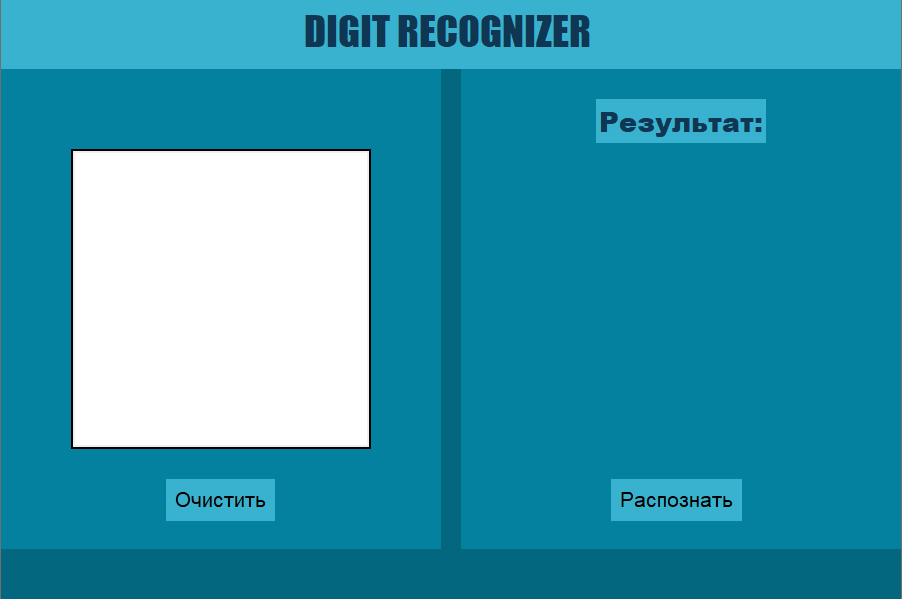
На исследовательском этапе проекта были получены знания о основных принципах работы, видах, шагах разработки и параметрах нейросетей. Были изучены библиотеки для реализации поставленной задачи. В качестве основного инструмента для разработки нейросети была выбрана библиотека Keras. Приобретенные знания помогли выбрать подходящую архитектуру нейросети, подобрать верные параметры, спроектировать модель и реализовать её в практической части проекта.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Общие задачи и результаты

В практической части проекта стояло несколько основных задач: построение эффективной модели нейросети, создание дружественного интерфейса и интеграция нейросети в интерфейс.

Приведем общие результаты, полученные при решении вышеупомянутых задач. На Рисунке 2.1 показан разработанный интерфейс приложения.

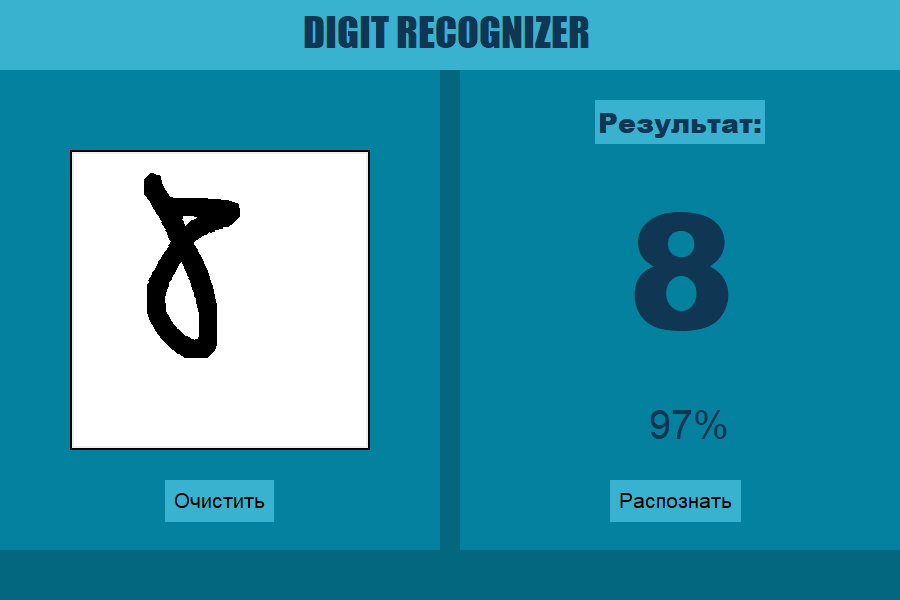


**Рисунок 2.1 — Интерфейс приложения**

Интерфейс приложения условно разделен на три части. В верхней части окна интерфейса написано название приложения. Левая часть интерфейса предназначена для ввода пользователем рукописной цифры от 0 до 9. Под холстом ввода расположена кнопка «Очистить», предназначенная для очисти области ввода. Правая часть интерфейса отвечает за вывод результата работы нейросети. Выходные данные представляют из себя саму цифру, которую распознала нейронная сеть и вероятность в процентах, что это именно эта цифра.

Результат появляется после того, как пользователь введет цифру и нажмет на кнопку «Распознать».

Продемонстрируем процесс работы приложения. Результат работы приложения при введённой цифре 8 изображен на Рисунке 2.2.



**Рисунок 2.2 — Результат работы приложения**

## Построение модели нейронной сети

В рамках выполнения проектной работы была поставлена задача создать эффективную модель нейронной сети, которая позволит с высокой точностью определять рукописные цифры.

### Библиотека Keras

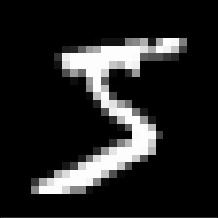
В качестве основы для решения этой задачи была выбрана высокоуровневая библиотека Keras. Данная библиотека является эффективным и доступным инструментом для работы с искусственными нейронными сетями, так как она содержит реализации широко используемых алгоритмов и функций.

### Сбор данных для обучения

Сбор данных для обучения традиционно является первым шагом для разработки нейронной сети. Данный шаг является одним из важнейших при проектировании нейросети и может быть достаточно затратным по времени и ресурсам. Но для задачи распознавания цифр существует огромная база данных образцов рукописных цифр MNIST (Modified National Institute of Standards and Technology). База данных MNIST состоит из 70 тыс. картинок в специальном формате, которые могут быть использованы для формирования обучающей и тестовой выборки.

### Формат цифр MNIST

При обучении нейросети важно понимать в каком формате представлены обучающие данные. Формат цифр MNIST представляет из себя изображение 28x28 пикселей, на котором изображена белая цифру на черном фоне. Сама же цифра помещается в окно 20x20 пикселей. Пример цифры формата MNIST показан на Рисунке 2.3.



**Рисунок 2.3 — Формат цифры MNIST**

Знание устройства формата данных, на котором будет обучаться нейросеть, поможет в дальнейшем преобразовывать цифру, введенную пользователем к виду, который «понимает» нейросеть.

### Подготовка данных для обучения

Перейдем к построению нейросети. Для начала импортируем библиотеки, которые будем использовать. Импорт необходимых библиотек представлен в Листинге 2.1.

*Листинг 2.1* — *Импорт необходимых библиотек*

from tensorflow import keras from keras import Sequential

from tensorflow.keras.datasets import mnist

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

Выгрузим обучающие и тестовые данные из базы данных MNIST и нормализуем их (Листинг 2.2).

*Листинг 2.2* — *Выгрузка данных и нормализация*

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train / 255 x\_test = x\_test / 255

y\_train = keras.utils.to\_categorical(y\_train, 10) y\_test = keras.utils.to\_categorical(y\_test, 10)

Данные x\_train и x\_test представляют из себя массив значений с от 0 до 255, где число определяет цвет этого пикселя, то есть 0 – черных, а 255 – полностью белый. Нормализованные данные заключены в диапазон от 0 до 1, что помогает нейросети более удобно работать с данными. Данные y\_train и y\_test также нормализуются и приводятся к формату вероятности.

### Построение модели нейросети

Перейдем к построению модели нейросети. Как говорилось ранее для решения задачи распознавания цифр была выбрана сверточная архитектура нейронной сети, которая предназначена для работы с изображениями.

Реализация сверточной модели нейронной сети представлена в Листинге 2.3.

*Листинг 2.3* — *Сверточная модель нейросети*

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)), Flatten(),

Dense(128, activation='relu'), Dense(10, activation='softmax')

])

Опишем составляющие построенной архитектуры. Первый слой Conv2D является сверточным слоем с 32 фильтрами (картами-признаков), которые постепенно накладываются на кусочки изображения и определяют наличие того или иного признака на нём. Сами фильтры имеют размер 3x3 и функцию активации ReLU. На вход слоя подается изображение 28x28 пикселей с одном цветовым каналом (черно-былым).

Следующий слой MaxPooling2D или слой пулинга с размером 2x2. Он выполняет функцию уменьшения размерности данных и помогает извлечь наиболее значимые признаки, полученные из предыдущего слоя.

Далее используется сверточный слой с 64 фильтрами размером 3x3 и функцией активации ReLU. По логике работы аналогичен первому. После расположен еще один слой пулинга с размером 2x2.

Flatten это слой, преобразующий данные из двумерного формата в одномерный формат, то есть в одномерный вектор. Это используется для подачи данных на слои, работающие только с одномерными векторами, как например полносвязный слой.

Слой Dense представляет из себя полносвязные слой с 128 нейронами и функцией активации ReLU.

Наконец, последний полносвязный слой Dense с 10 выходами (равными количеству цифр) и функцией активации softmax. Данные слой подает на выход вероятность, того, что была распознана так или иная цифра от 0 до 9. Максимальное значения среди всех выходных вероятностей является лучшим «предсказанием»

нейросети. Цифра с наивысшей вероятностью и сама вероятность поступают в интерфейс как конечный «ответ» нейросети.

## Обучение и тестирование

Перед началом обучения модели необходимо задать параметры обучения. В Листинге 2.4 приведена часть кода, на которой настраиваются параметры и запускается процесс обучения.

*Листинг 2.4* — *Обучение нейросети*

model.compile( optimizer="adam",

loss='categorical\_crossentropy', metrics=['accuracy']

)

model.fit(x\_train, y\_train, batch\_size=32, epochs=5, validation\_split=0.2)

Сначала устанавливаются следующие параметры обучения: оптимизатор, функция потерь, метрика оценки точности.

Оптимизатор определяет способ обновления весов и минимизации функции потерь. Оптимизатор Adam является одним из самых широко используемых и эффективных алгоритмов обучения. Данный оптимизатор автоматические адаптирует скорость обучения и параметры весов. Это позволяет оптимизатору эффективно и быстро настроить веса модели.

Функция потерь используется для оценки отклонения между предсказанным значением и истинным результатом. При обучении данной модели используется категориальная кросс-энтропия функция потерь как одна из популярных функций в задачах многоклассовой классификации.

В качестве метрики оценки производительности модели используется метрика точности (accuracy), которая измеряет процент правильных предсказаний модели относительно общего числа примеров.

Далее передаются входные данные для обучения и устанавливаются следующие параметры: размер батча, количество эпох и параметр валидации.

Размер батча определяет количество примеров, которые будут одновременно обрабатываться моделью перед обновлением весов в процессе обучения.

Количество эпох отвечает за то, сколько раз обучающий набор данных будет проходить через модель нейросети при обучении.

Наконец, параметр валидации определяет сколько примеров из обучающей выборки будет использовано для тестирования модели в каждой эпохе. В данном случае это 20% обучающих данных.

Обученная модель должна быть протестирована данными, которые не учувствовали в обучении. Тестирование в библиотеке Keras происходит с помощью метода evaluate. Если модель показывает высокую точность, то её можно сохранить и использовать дальше для других данных. Процесс тестирования и сохранения модели показан в Листинге 2.5

*Листинг 2.5* — *Тестирование и сохранение модели*

model.evaluate(x\_test, y\_test) model.save('model.h5')

Полный код программы отражающий процесс построения, обучения и тестирования модели показан в Листинге 2.6.

*Листинг 2.6* — *Полный код программы*

from tensorflow import keras from keras import Sequential

from tensorflow.keras.datasets import mnist

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

(x\_train, y\_train), (

x\_test, y\_test) = mnist.load\_data() # Выгружаем данные из mnist (x -

входные данные y - ожидаемый результат)

x\_train = x\_train / 255 x\_test = x\_test / 255

y\_train = keras.utils.to\_categorical(y\_train, 10) y\_test = keras.utils.to\_categorical(y\_test, 10)

model = Sequential([ # Сверточная архитектура нейросети

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)), Flatten(),

Dense(128, activation='relu'), Dense(10, activation='softmax')

])

*Продолжение Листинга 2.6*

# Определение метрик, функции ошибки и оптимизации для обучения model.compile(optimizer="adam", loss='categorical\_crossentropy', metrics=['accuracy'])

# Запуск обучения сети, установка кол-ва эпох и батча

# (эпоха - кол-ва итераций прогона обучающих данных, батч - часть данных использующаяся в эпохе)

model.fit(x\_train, y\_train, batch\_size=32, epochs=5, validation\_split=0.2) #

Для небольшой сверточной модели

model.evaluate(x\_test, y\_test)

# Сохранения модели в файл

model.save('model.h5')

## Обработка входных данных

### Правила построение формата MNIST

Для того чтобы обученная нейронная сеть могла работать с произвольными данными требуется специальная обработка входных изображений. Другими словами, нужно преобразовать входные данные, полученные от пользователя, к формату на котором обучалась нейронная сеть и только потом их передавать.

Как было описано ранее нейросеть обучалась на формате, использованном в базе рукописных цифр MNIST. Сам формат строился по определенным правилам. Рисовалась белая цифра на черном фоне, которая умещалась в бокс 20x20 пикселей. Далее, вычислялся центр масс изображения, и оно помещалось на поле 28x28, так чтобы центр масс совпадал с центром поля.

* + 1. **Алгоритмы обработки изображения**

Для преобразования входного изображения к формату MNIST использовались библиотеки OpenCV, NumPy и SciPy, содержащие алгоритмы для работы с изображениями и другими данными.

На Листинге 2.7 представлен полный код программы, отвечающий за преобразование изображения к формату MNIST.

*Листинг 2.7* — *Обработка изображения*

from scipy.ndimage import center\_of\_mass import math

import cv2

import numpy as np

from keras.models import load\_model

# Загрузка обученной модели нейронной сети

model = load\_model('model.h5')

# Вычисление центра масс и направления сдвига

def getBestShift(img):

cy, cx = center\_of\_mass(img)

rows, cols = img.shape

shiftx = np.round(cols / 2.0 - cx).astype(int) shifty = np.round(rows / 2.0 - cy).astype(int)

return shiftx, shifty

def shift(img, sx, sy): rows, cols = img.shape

M = np.float32([[1, 0, sx], [0, 1, sy]]) shifted = cv2.warpAffine(img, M, (cols, rows)) return shifted

def recognize():

img = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE) gray = 255 - img

# применяем пороговую обработку

(thresh, gray) = cv2.threshold(gray, 128, 255, cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)

# удаляем нулевые строки и столбцы

while np.sum(gray[0]) == 0: gray = gray[1:]

while np.sum(gray[:, 0]) == 0:

gray = np.delete(gray, 0, 1) while np.sum(gray[-1]) == 0:

gray = gray[:-1]

while np.sum(gray[:, -1]) == 0:

gray = np.delete(gray, -1, 1) rows, cols = gray.shape

# изменяем размер, чтобы помещалось в box 20x20 пикселей

if rows > cols:

factor = 20.0 / rows rows = 20

cols = int(round(cols \* factor))

gray = cv2.resize(gray, (cols, rows)) else:

factor = 20.0 / cols cols = 20

rows = int(round(rows \* factor))

gray = cv2.resize(gray, (cols, rows)) # расширяем до размера 28x28

colsPadding = (int(math.ceil((28 - cols) / 2.0)), int(math.floor((28 - cols) / 2.0)))

*Продолжение Листинга 2.7*

rowsPadding = (int(math.ceil((28 - rows) / 2.0)), int(math.floor((28 - rows) / 2.0)))

gray = np.lib.pad(gray, (rowsPadding, colsPadding), 'constant')

# сдвигаем центр масс

shiftx, shifty = getBestShift(gray) shifted = shift(gray, shiftx, shifty) gray = shifted

cv2.imwrite('image\_mnist.jpg', gray) img = gray / 255.0

img = np.array(img).reshape(-1, 28, 28, 1) # конвертируем в картинку 28

на 28 пикселей

prediction = model.predict(img)

percent = prediction[0][np.argmax(prediction)] # Результаты работы сети

percent = int(percent \* 100) digit = np.argmax(prediction) print("Predicted digit:", digit) print(percent)

return digit, percent

Алгоритм обработки изображения и выдачи результата состоит из трех основных функций: getBestShift, shift и recognize.

Функция getBestShift вычисляет сдвиг изображения, чтобы его центр масс совпадал с центром изображения. Это позволяет корректно выравнивать и центрировать цифру перед распознаванием.

Функция выполняет сдвиг изображения на указанные значения sx и sy. Это используется для фактического сдвига цифры в соответствии с вычисленными сдвигами.

Функция recognize загружает изображение, преобразует его в оттенки серого и применяет пороговую обработку для получения бинарного изображения. Затем она обрезает нулевые строки и столбцы, чтобы удалить пустые области вокруг цифры. Затем изображение изменяется до размера 20x20 пикселей и затем до размера 28x28 пикселей. Затем происходит сдвиг центра масс изображения и сохранение результата в файл. Затем изображение нормализуется и преобразуется в массив, который передается в модель для предсказания. Далее возвращаются результаты работы нейросети для обработанного изображения. Эти данные подаются на интерфейс для отображения в приложении.

**ЗАКЛЮЧЕНИЕ**

В ходе данного проекта была успешно разработана нейросеть для распознавания цифр с дружественным интерфейсом. Основной целью проекта было создание эффективной модели, взаимодействие с которой происходило через пользовательский графический интерфейс.

В рамках работы были достигнуты все поставленные задачи. Был проведен обширный анализ данных, предварительную обработку изображений и подготовку набора данных для обучения. На этапе обучения модели была применена нейронная сеть с сверточной архитектурой, которая показала отличные результаты в распознавании цифр. Был спроектирован простой и понятный интерфейс для взаимодействия с пользователем.

Однако, в рамках дальнейших исследований и разработки проекта, есть несколько направлений для улучшения. Во-первых, можно провести дополнительные эксперименты с различными архитектурами нейронных сетей и параметрами обучения, чтобы улучшить точность распознавания.

Кроме того, стоит обратить внимание на расширение функциональности интерфейса. В дальнейшем можно добавить возможность распознавания и классификации не только цифр, но и других символов или объектов. Также можно улучшить пользовательский опыт, добавив функции для обработки и улучшения входных изображений, чтобы повысить точность распознавания в условиях с плохим качеством фотографий.

Предложенный проект может быть доработан и улучшен в будущем, открывая возможности для применения в различных сферах, где требуется автоматическое распознавание цифровых данных.

Таким образом, цель данного проекта была успешно достигнута, и разработанная нейросеть продемонстрировала высокую точность и эффективность в распознавании цифр.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Введение в нейросети / Ившин Вадим. URL: <https://habr.com/ru/articles/342334/> (Дата обращения: 22.03.23)
2. Траск Эндрю Грокаем глубокое обучение. — СПб.: Питер, 2019. — 352

с.

1. Keras layers / Документация библиотеки Keras. URL:

<https://keras.io/api/layers/> (Дата обращения: 25.03.23)

1. MNIST digits classification dataset / Документация библиотеки Keras. URL: <https://keras.io/api/datasets/mnist/> (Дата обращения: 25.03.23)
2. Нейросети и глубокое обучение, глава 1: использование нейросетей для распознавания рукописных цифр / Голованов Вячеслав. URL: <https://habr.com/ru/articles/456738/> (Дата обращения: 03.04.23)
3. Гасников А. В. Современные численные методы оптимизации. Метод универсального градиентного спуска: учебное пособие. — М.: МФТИ, 2018. — 291 с.
4. Оптимизаторы в ML / Тяпкин Д. / URL: <https://academy.yandex.ru/handbook/ml/article/optimizaciya-v-ml> (Дата обращения: 06.04.23)