Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Нагалевский

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Крамаренко

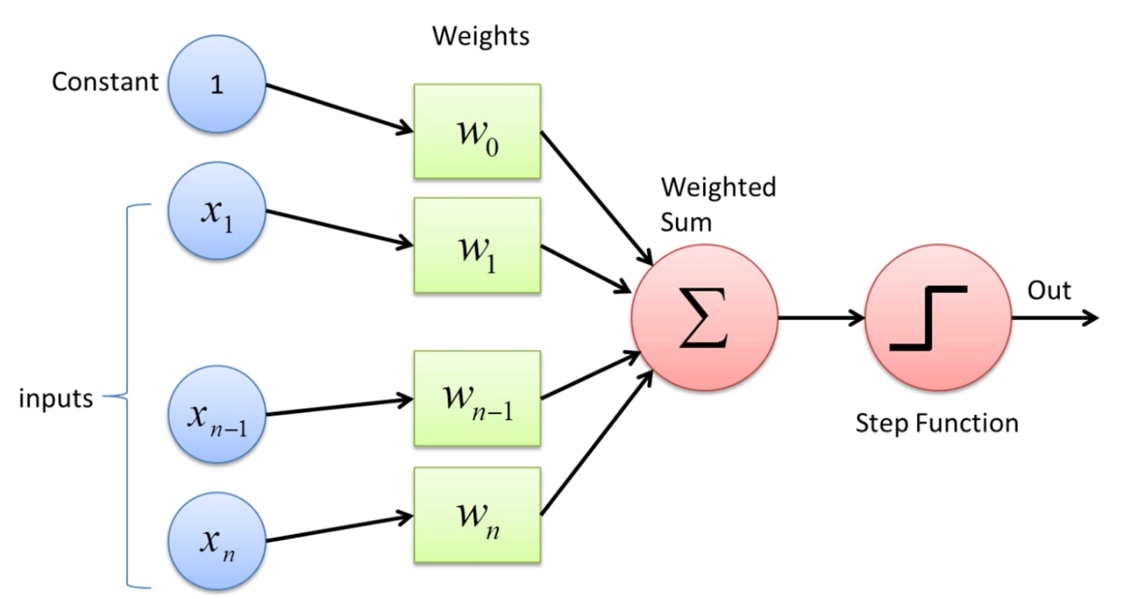
**Тема работы:** Решение классификации изображений с помощью нейронных сетей.

**Ход работы:**

Обычная нейронная сеть и персептрон — это два основных элемента в мире искусственных нейронных сетей, которые играют важную роль в машинном обучении и глубоком обучении. Рассмотрим каждый из них подробно.

**Персептрон** — это простейшая форма искусственного нейрона, предложенная Фрэнком Розенблаттом в 1957 году. Он служил вдохновением для развития более сложных нейронных сетей. Персептрон принимает несколько входов, применяет к ним взвешенные коэффициенты (веса) и суммирует результаты. Затем, к полученной сумме применяется функция активации, которая определяет, будет ли нейрон активирован и передаст ли сигнал на выход.

В формальной записи, работа персептрона может быть представлена следующим образом:



Где:

- входные значения.

- веса, которые назначены каждому входу.

- функция активации (может быть другая)

Персептрон прекрасно подходит для задач, в которых можно провести линейную границу между классами. Он используется в бинарной классификации, но не способен решать сложные задачи, такие как обработка изображений или текста.

**Обычная нейронная сеть** (или искусственная нейронная сеть) — это более сложная и многослойная версия персептрона. Она состоит из нескольких слоев нейронов, включая входной слой, скрытые слои и выходной слой. Каждый нейрон в сети связан с нейронами в соседних слоях через веса.

Работа нейронной сети заключается в передаче информации от входного слоя через скрытые слои и, наконец, к выходному слою. Каждый нейрон в сети применяет взвешенные суммы к данным, а затем применяет функцию активации для определения своего выхода. В процессе обучения веса в нейронной сети подстраиваются так, чтобы минимизировать ошибку предсказания.

Преимущество нейронных сетей состоит в их способности решать сложные задачи, такие как распознавание образов, обработка естественного языка, синтез речи и другие. Они являются ключевой технологией в области глубокого обучения.

Примеры типовых архитектур нейронных сетей включают в себя многослойные перцептроны (MLP), сверточные нейронные сети (CNN) для обработки изображений и рекуррентные нейронные сети (RNN) для обработки последовательных данных.

Рассмотрим реализацию такой сети.

Установим библиотеку TensorFlow и фреймворк Keras.

Загрузим данные MNIST – датасет рукописных цифр. x\_train и x\_test содержат изображения рукописных цифр, а y\_train и y\_test - соответствующие метки классов (цифры).

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

Масштабируем, чтобы значения пикселей находились в диапазоне от 0 до 1. Это ускоряет процесс обучения нейронной сети.

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

Далее изображения преобразуются из 2D-массивов в 1D-массивы, чтобы быть совместимыми с входным слоем нейронной сети и метки классов преобразуются в форму one-hot encoding для обучения нейронной сети в задаче многоклассовой классификации.

x\_train = x\_train.reshape(x\_train.shape[0], 784)

x\_test = x\_test.reshape(x\_test.shape[0], 784)

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

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

Создается пустая последовательная модель нейронной сети, к которой будут поочередно добавляться слои. Добавляются два скрытых слоя с разным количеством нейронов и активацией ReLU (Rectified Linear Unit). Эти слои обеспечивают способность сети извлекать признаки из данных и добавляется выходной слой с 10 нейронами (по числу классов в задаче) и функцией активации softmax для определения вероятности принадлежности каждого класса.

model = Sequential()

model.add(Dense(512, input\_shape=(784, ), activation='relu'))

model.add(Dense(768, activation='relu'))

model.add(Dense(10, activation='softmax'))

Модель компилируется с использованием оптимизатора Adam и категориальной функции потери для обучения многоклассовой классификации. Также определяются метрики для оценки производительности модели.

optimizer = Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

Далее обучаем модель течение 3 эпох (полных проходов через данные). Обновление весов происходит после каждых 128 образцов (minibatch), оцениваем точность модели и сохраняем.

model.fit(x\_train, y\_train, epochs=3, batch\_size=128)

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

print(f'Точность {accuracy[1]\*100:.2f}%')

model.save("my\_model.keras")

Теперь проверим на примерах работу нейронной сети. Данную задачу выполняет следующий код.

model = load\_model("my\_model.keras")

image\_path ="img/7.jpg"

img\_cv = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

img\_cv = cv2.resize(img\_cv, (28, 28))

image = img\_cv / 255.0

image = image.reshape(1, 784)

predictions = model.predict(image)

predicted\_class = np.argmax(predictions)

print("Предсказанный класс:", predicted\_class)

Результат работы представлен на рисунке 1.

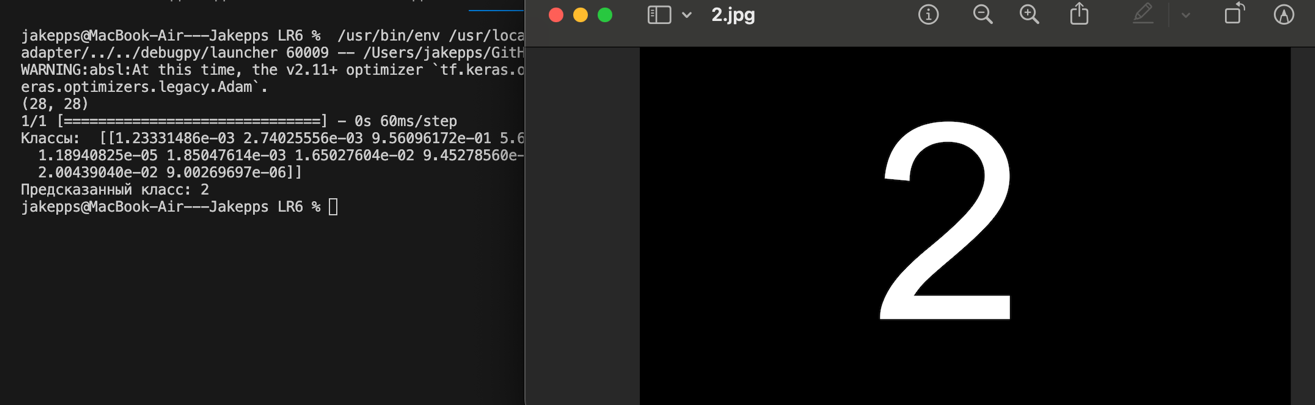


Рисунок 1 – результаты работы нейронной сети.

Сравним несколько значений количества эпох по параметрам скорость обучения, скорость работы сети, процент корректной работы на тестовой базе. В итоге получились результаты, изображенные на рисунке 2.

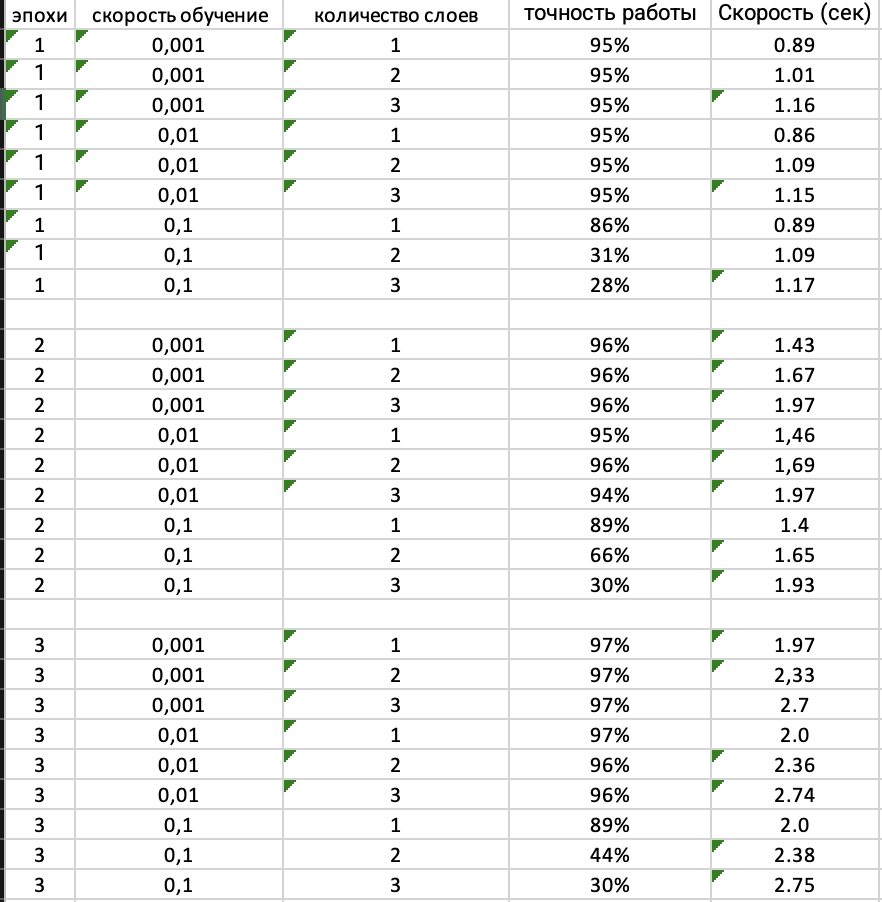


Рисунок 2 – результаты сравнения параметров.

**Сверточная нейронная сеть** (Convolutional Neural Network, CNN) - это тип искусственной нейронной сети, разработанный специально для обработки и анализа структурированных данных, таких как изображения и видео. Она имеет революционное значение в области компьютерного зрения, и она может успешно выполнять задачи, связанные с распознаванием образов, классификацией объектов, сегментацией изображений, детекцией объектов и многими другими задачами, требующими анализа визуальных данных.

Рассмотрим подробно описание ключевых аспектов сверточной нейронной сети:

1. Сверточные слои (Convolutional Layers): Основная идея CNN заключается в использовании сверточных слоев для извлечения признаков из изображений. Сверточные фильтры перемещаются по входным данным и выполняют операцию свертки, что позволяет выявить локальные паттерны, такие как грани, углы, текстуры и другие характеристики. Каждый сверточный слой может иметь несколько фильтров.
2. Пулинг слои (Pooling Layers): Пулинг используется для уменьшения размерности данных и выделения наиболее важных признаков. Популярные методы пулинга включают в себя среднее пулинг (average pooling) и максимальное пулинг (max pooling), которые уменьшают размерность данных, сохраняя наиболее существенные детали.
3. Полносвязные слои (Fully Connected Layers): после сверточных и пулинговых слоев идут один или несколько полносвязных слоев. Они выполняют классификацию или регрессию на основе извлеченных признаков. Полносвязные слои соединяют каждый нейрон с каждым нейроном предыдущего слоя.
4. Регуляризация: Чтобы предотвратить переобучение, CNN может использовать методы регуляризации, такие как dropout или L2 регуляризацию.
5. Архитектуры CNN: Существует несколько известных архитектур сверточных нейронных сетей, включая LeNet, AlexNet, VGG, GoogLeNet (Inception), ResNet и другие. Каждая из них имеет свою уникальную структуру и обеспечивает выдающиеся результаты в различных задачах.

Преимущества сверточных нейронных сетей включают способность автоматически извлекать признаки из данных, инвариантность к масштабированию и сдвигам, а также способность работать с изображениями разного размера. Они широко применяются в машинном обучении, компьютерном зрении, автономных транспортных средствах, медицинской диагностике и других областях, где необходим анализ визуальных данных.

Рассмотрим реализацию такой сети.

Как и в прошлый раз, загружаем данные, обрабатываем данные и кодируем метки классов.

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

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1).astype('float32') / 255

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1).astype('float32') / 255

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

Создаем модель сверточной нейронной сети с несколькими сверточными слоями (Conv2D), слоями пулинга (MaxPooling2D), слоем выравнивания (Flatten) и полносвязными слоями (Dense). Эти слои позволяют извлекать признаки из изображений и выполнять классификацию.

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))

Модель компилируется с использованием оптимизатора Adam и категориальной функции потери для обучения многоклассовой классификации. Также определяются метрики для оценки производительности модели.

optimizer = Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

Модель обучается на обучающих данных в течение 3 эпох. Обновление весов происходит после каждых 128 образцов (minibatch). Валидационные данные используются для оценки производительности модели во время обучения. Модель оценивается на тестовых данных, и результаты сохраняются в переменной score. Сохраняем модель и выводим результаты.

model.fit(x\_train, y\_train, epochs=3, batch\_size=128, validation\_data=(x\_test, y\_test))

score = model.evaluate(x\_test, y\_test, verbose=0)

model.save("my\_nerone\_set.keras")

print('Потеря тестовых данных: ', score[0])

print('Точность на тестовых данных: ', round(score[1], 3))

Теперь проверим на примерах работу сверточной нейронной сети. Данную задачу выполняет следующий код.

model = load\_model('my\_nerone\_set.keras')

img\_path = 'img/4.jpg'

img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

img = cv2.resize(img, (28, 28))

img = img / 255.0

img = img.reshape(1, 28, 28, 1)

predictions = model.predict(img)

predicted\_digit = np.argmax(predictions)

print(f"Предсказанная цифра: {predicted\_digit}")

Результат работы представлен на рисунке 3.

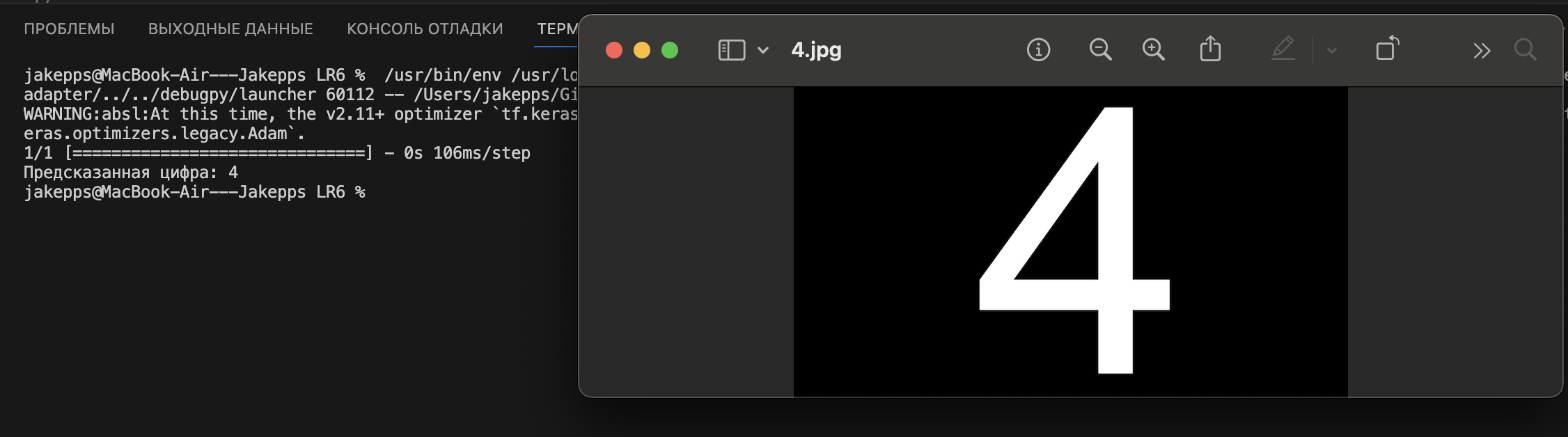


Рисунок 3 – результаты работы сверточной нейронной сети.

**Листинг программ**

Файл lab6\_1.py

import tensorflow as tf

import numpy as np

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Activation

from keras.utils import to\_categorical

from keras.optimizers import Adam

import ssl

ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context

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

x\_train = x\_train.reshape(x\_train.shape[0], 784)

x\_test = x\_test.reshape(x\_test.shape[0], 784)

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

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

model = Sequential()

model.add(Dense(512, input\_shape=(784, ), activation='relu'))

model.add(Dense(768, activation='relu'))

model.add(Dense(10, activation='softmax'))

optimizer = Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=3, batch\_size=128)

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

print(f'Точность {accuracy[1]\*100:.2f}%')

model.save("my\_model.keras")

Файл lab6\_2.py

from keras.models import load\_model

import cv2

import numpy as np

model = load\_model("my\_model.keras")

image\_path ="img/7.jpg"

img\_cv = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

img\_cv = cv2.resize(img\_cv, (28, 28))

print(img\_cv.shape)

image = img\_cv / 255.0

image = image.reshape(1, 784)

predictions = model.predict(image)

print("Классы: ", predictions)

predicted\_class = np.argmax(predictions)

print("Предсказанный класс:", predicted\_class)

Файл lab6\_2\_compare.py

import tensorflow as tf

from keras.models import load\_model

from keras.datasets import mnist

from keras.utils import to\_categorical

import time

model = load\_model("my\_model.keras")

epochs\_list = [1, 2, 3]

learning\_rates = [0.001, 0.01, 0.1]

num\_layers\_list = [1, 2, 3]

accuracies = []

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

train\_images = train\_images.reshape((60000, 28, 28, 1)).astype('float32') / 255

test\_images = test\_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

train\_images\_flat = train\_images.reshape((60000, 784))

test\_images\_flat = test\_images.reshape((10000, 784))

train\_labels = to\_categorical(train\_labels, 10)

test\_labels = to\_categorical(test\_labels, 10)

for epochs in epochs\_list:

for lr in learning\_rates:

for num\_layers in num\_layers\_list:

layered\_model = tf.keras.Sequential()

for \_ in range(num\_layers):

layered\_model.add(tf.keras.layers.Dense(128, activation='relu'))

layered\_model.add(tf.keras.layers.Dense(10, activation='softmax'))

layered\_model.compile(optimizer=tf.keras.optimizers.legacy.RMSprop(learning\_rate=lr),

loss='categorical\_crossentropy',

metrics=['accuracy'])

start\_time = time.time()

layered\_model.fit(train\_images\_flat, train\_labels, epochs=epochs, batch\_size=128, verbose=0)

end\_time = time.time()

test\_time = end\_time - start\_time

test\_loss, test\_accuracy = layered\_model.evaluate(test\_images\_flat, test\_labels, verbose=0)

accuracies.append((epochs, lr, num\_layers, test\_accuracy, test\_time))

for epochs, lr, num\_layers, accuracy, t\_time in accuracies:

print(f'Эпохи: {epochs}, Скорость обучения: {lr}, Количество слоев: {num\_layers}, Точность на тесте: {round(accuracy, 2)\*100}%, Затраченное время: {round(t\_time, 2)} сек.')

Файл lab6\_3.py

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

from keras.utils import to\_categorical

from keras.datasets import mnist

from keras.models import Sequential

from keras.optimizers import Adam

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

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1).astype('float32') / 255

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1).astype('float32') / 255

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))

optimizer = Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=3, batch\_size=128, validation\_data=(x\_test, y\_test))

score = model.evaluate(x\_test, y\_test, verbose=0)

model.save("my\_nerone\_set.keras")

print('Потеря тестовых данных: ', score[0])

print('Точность на тестовых данных: ', round(score[1], 3))

Файл lab6\_4.py

import cv2

import numpy as np

from keras.models import load\_model

model = load\_model('my\_nerone\_set.keras')

img\_path = 'img/4.jpg'

img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

img = cv2.resize(img, (28, 28))

img = img / 255.0

img = img.reshape(1, 28, 28, 1)

predictions = model.predict(img)

predicted\_digit = np.argmax(predictions)

print(f"Предсказанная цифра: {predicted\_digit}")