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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Пинский Д.А.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

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

**Тема:** Введение в нейронные сети.

**Цель работы:** в рамках данной лабораторной работы будет рассматриваться решение классификации изображений.

Задание 1 (самостоятельно). Построить многослойный персептрон средствами библиотеки Keras языка Python (или соответствующей библиотеки на другом языке), позволяющий распознавать цифры. Для обучения и тестирования использовать базу MNIST. Возможно использовать любую функции активации, любую структуру сети и любой алгоритм обучения.

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

Рисунок 1 – Импортируемые модули

1. Импортируем библиотеку TensorFlow.

TensorFlow используется для построения и обучения нейронных сетей.

1. Импорт модели Sequential из Keras.

Sequential — это линейная модель, где слои расположены последовательно.

1. Импортируем слои:

Dense — обычный плотный слой, где каждый нейрон соединён со всеми нейронами предыдущего слоя.

Flatten — преобразует многомерный вход (например, 28x28 изображение) в одномерный вектор (например, размером 784).

1. Загружаем датасет MNIST.
2. Импортируем функцию to\_categorical.

to\_categorical преобразует метки в формат «one-hot encoding».

1. Импортируем модуль времени для измерения скорости обучения и выполнения.

Следующим шагом следует загрузка и предобработка данных:

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

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

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

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

Рисунок 2 – Загрузка и предобработка данных.

1. mnist.load\_data() — загрузка MNIST датасета. Возвращает обучающую выборку (x\_train, y\_train) и тестовую выборку (x\_test, y\_test):

x\_train и x\_test содержат данные изображений (28x28 пикселей в оттенках серого). y\_train и y\_test содержат метки классов (от 0 до 9).

2. x\_train.astype('float32') / 255.0 — преобразуем данные изображений в тип float32, а затем нормализуем их, разделив на 255. Значения пикселей, изначально лежащие в диапазоне 0, 255, таким образом преобразуются в диапазон 0, 1, что улучшает стабильность обучения.

3. to\_categorical(y\_train, 10) — преобразуем метки (y\_train и y\_test) в формат «one-hot encoding».

Далее идет создание модели:

model = Sequential([

    Flatten(input\_shape=(28, 28)),

    Dense(128, activation='relu'),

    Dense(64, activation='relu'),

    Dense(10, activation='softmax')

])

model.compile(

    optimizer='adam',

    loss='categorical\_crossentropy',

    metrics=['accuracy']

)

Рисунок 3 – Создание модели.

Sequential — создаёт линейную модель.

Flatten(input\_shape=(28, 28)) — принимает двумерный вход (28x28) и преобразует его в одномерный вектор.

Dense(128, activation='relu') — добавляет плотный слой с 128 нейронами, функция активации relu (Rectified Linear Unit).

Dense(64, activation='relu') — ещё один плотный слой, но уже с 64 нейронами.

Dense(10, activation='softmax') — выходной слой с 10 нейронами (по числу классов в MNIST), использует функцию активации softmax, которая преобразует выходы в вероятности.

model.compile() — собираем модель для обучения:

optimizer='adam' — алгоритм оптимизации. Adam автоматически подстраивает скорость обучения, выполняя его одним из лучших выборов для большинства задач.

loss='categorical\_crossentropy' — функция ошибки для задач многоклассовой классификации. Сравнивает предсказанные вероятности (`softmax`) с истинными метками (one-hot-код).

metrics=['accuracy']`— отслеживаем дополнительную метрику: точность.

Следующий шаг — это обучение модели:

model.fit(

    x\_train, y\_train,

    epochs=10,

    batch\_size=32,

    validation\_split=0.2

)

Рисунок 4 – Обучение модели.

x\_train, y\_train — тренировочные данные (входы и метки).

epochs=10 — Обучение разбито на эпохи. Эпоха это одна итерация по всем входным данным.

batch\_size=32 —  модель разбивает данные на меньшие блоки (batches) и итерирует по этим блокам во время обучения. Это число указывает размер каждого блока данных. Мы делим данные на пакеты по 32 образца для прохождения через модель за раз.

validation\_split=0.2 — 20% тренировочных данных используются для проверки во время обучения.

Оценка модели:

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test, verbose=2)

print(f'Точность модели на тестовых данных: {test\_accuracy:.2f}')

Рисунок 5 – Оценка модели.

evaluate проверяет модель на тестовом наборе данных и возвращает:

test\_loss — средняя потеря на тестовых данных;

test\_acc — точность модели на тестовых данных.

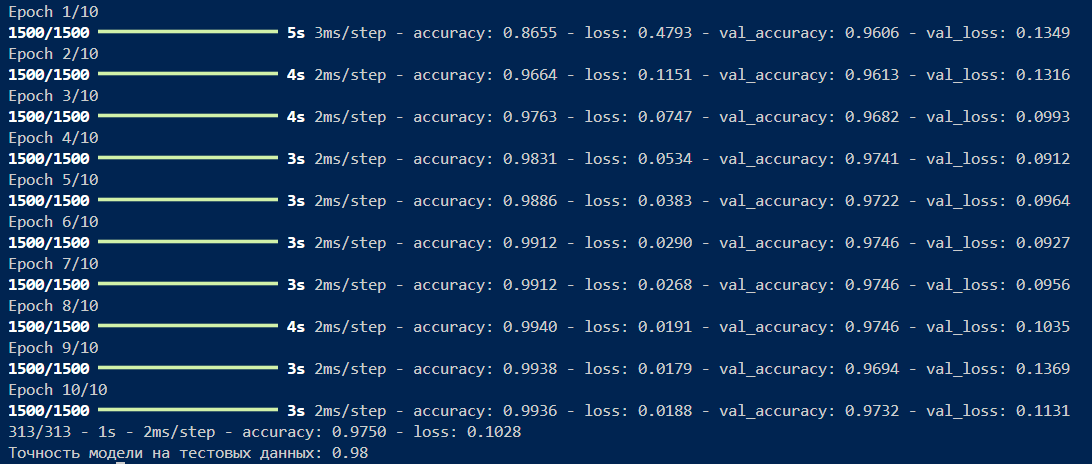


Рисунок 6 – Вывод данных

Задание 2 (самостоятельно). Исследовать нейронную сеть, построенную в задании 1 Подобрать количество эпох обучения, необходимое для выбранного вами предела корректной работы на тестовой базе. Сравнить несколько значений количества эпох по параметрам скорость обучения, скорость работы сети, процент корректной работы на тестовой базе.

Проводить сравнение будем на 5, 10, 20 и 50 эпохах:

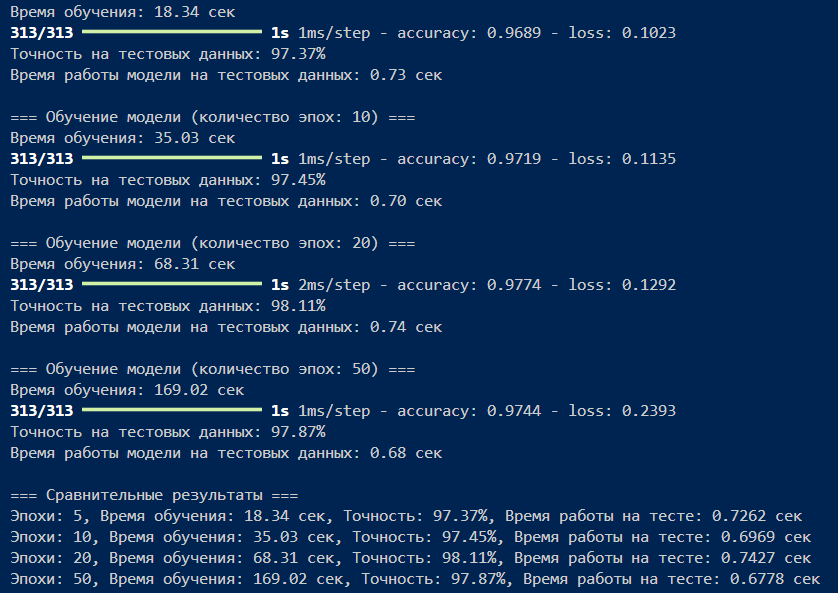


Рисунок 7 – Полученные результаты

В сравнительном результате можем выделить увеличение времени обучения от количества эпох (вот это да, правда?). Точность повышается при увеличении эпох, однако в нашем случае при 20 эпохах точность наибольшая.

Задание 3 (самостоятельно). Построить сверточную нейронную сеть,

решающую ту же задачу классификации. Рассмотреть несколько архитектур, подобрать оптимальные параметры. Сравнить результаты с предыдущей архитектурой.

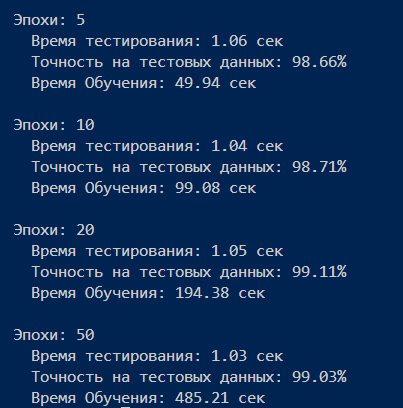


Рисунок 8 – Результаты CNN

По полученным данным можно сделать вывод: сверточная нейронная сеть работает дольше, но намного точнее.

**Листинг программы**

**LR6.py**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

import time

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

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

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

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

def create\_model():

    model = Sequential([

        Flatten(input\_shape=(28, 28)),

        Dense(128, activation='relu'),

        Dense(64, activation='relu'),

        Dense(10, activation='softmax')

    ])

    model.compile(optimizer='adam',

                  loss='categorical\_crossentropy',

                  metrics=['accuracy'])

    return model

results = []

for epochs in [5, 10, 20, 50]:

    print(f'\n=== Обучение модели (количество эпох: {epochs}) ===')

    model = create\_model()

    start\_time = time.time()

    model.fit(x\_train, y\_train, epochs=epochs, batch\_size=32, validation\_split=0.5, verbose=0)

    training\_time = time.time() - start\_time

    print(f'Время обучения: {training\_time:.2f} сек')

    start\_time = time.time()

    test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

    inference\_time = time.time() - start\_time

    print(f'Точность на тестовых данных: {test\_accuracy:.2%}')

    print(f'Время работы модели на тестовых данных: {inference\_time:.2f} сек')

    results.append({

        'epochs': epochs,

        'training\_time': training\_time,

        'test\_accuracy': test\_accuracy,

        'inference\_time': inference\_time

    })

print("\n=== Сравнительные результаты ===")

for res in results:

    print(f"Эпохи: {res['epochs']}, "

          f"Время обучения: {res['training\_time']:.2f} сек, "

          f"Точность: {res['test\_accuracy']:.2%}, "

          f"Время работы на тесте: {res['inference\_time']:.4f} сек")

**LR6\_CNN.py**

import tensorflow as tf

from tensorflow.keras import Sequential

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

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

import time

(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') / 255.0

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1).astype('float32') / 255.0

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

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

def build\_cnn():

    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'))

    return model

epochs\_list = [5, 10, 20, 50]

results = []

for epochs in epochs\_list:

    print(f"\nТренировка модели с {epochs} эпохами...\n")

    cnn\_model = build\_cnn()

    cnn\_model.compile(

        optimizer='adam',

        loss='categorical\_crossentropy',

        metrics=['accuracy']

    )

    train\_start\_time = time.time()

    cnn\_model.fit(x\_train, y\_train, epochs=epochs, batch\_size=32, validation\_split=0.2, verbose=1)

    train\_end\_time = time.time()

    train\_duration = train\_end\_time - train\_start\_time

    test\_start\_time = time.time()

    test\_loss, test\_acc = cnn\_model.evaluate(x\_test, y\_test, verbose=0)

    test\_end\_time = time.time()

    test\_duration = test\_end\_time - test\_start\_time

    results.append({

        'epochs': epochs,

        'train\_time': train\_duration,

        'test\_accuracy': test\_acc,

        'test\_time': test\_duration

    })

for res in results:

    print(f"\nЭпохи: {res['epochs']}")

    print(f"  Время тестирования: {res['test\_time']:.2f} сек")

    print(f"  Точность на тестовых данных: {res['test\_accuracy']:.2%}")

    print(f"  Время Обучения: {res['train\_time']:.2f} сек")