МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

**ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ИНСТИТУТ МАТЕМАТИКИ, ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК**

|  |  |
| --- | --- |
| «Рекомендовать к защите»  Заведующий кафедрой информационных технологий и прикладной математики, доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Данилова С.В.  протокол заседания кафедры № \_\_\_\_  от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |  |

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(БАКАЛАВРСКАЯ РАБОТА)

**Тест производительности нейронных сетей на примере CIFAR–10**

|  |  |
| --- | --- |
| Направление подготовки: | 02.03.02 Фундаментальная информатика и информационные технологии |
| Направленность образовательной программы: | Фундаментальная информатика и информационные технологии |
| Выпускную квалификационную работу выполнил: | студент 4 курса очной формы обучения \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Калабин Александр Николаевич |
| Руководитель выпускной квалификационной работы: | доцент кафедры информационных технологий и прикладной математики,  канд. физ.–мат. наук, \_\_\_\_\_\_\_\_Хашин Сергей Иванович  доцент |

Иваново, 2024

Министерство образования и науки Российской Федерации

ФГБОУ ВО «Ивановский государственный университет»

Институт математики, информационных технологий и естественных наук

Кафедра информационных технологий

и прикладной математики

|  |
| --- |
| **Утверждаю:**  Зав. кафедрой ИТ иПМ  канд.экон.наук, доцент\_\_\_\_\_\_\_\_\_Данилова С.В. |
| «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024год |

**ЗАДАНИЕ**

по подготовке выпускной квалификационной работы

студента 4 курса бакалавриата

института математики, информационных технологий и естественных наук

Калабина Александра Николаевича

(фамилия, имя и отчество студента)

Тема работы: Тест производительности нейронных сетей на примере CIFAR–10

(утверждена приказом ИвГУ № 2п/283 от «29» декабря 2023 г.)

Срок сдачи студентом законченной работы: «16» июня 2024 г.

Исходные данные к работе:

методические указания по выполнению выпускной квалификационной работы для бакалавров по направлению 02.03.02 Фундаментальная информатика и информационные технологии;

Краткое содержание работы:

В работе описывается процесс создания теста производительности нейронных сетей на примере CIFAR–10. Реализация данной программы позволит пользователям быстро и эффективно оценивать производительность нейронных сетей на своем ПК без необходимости установки и настройки сложных зависимостей, что сделает процесс тестирования доступным и удобным, в свою очередь это поможет привлечь больше людей к изучению и созданию нейронных сетей.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата выдачи задания: | «3» мая 2024 г. | | |
| Научный руководитель,  канд. физ.–мат. наук, доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |  | Хашин С.И. |
| Задание принял к исполнению,  студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |  | Калабин А.Н. |

Министерство образования и науки Российской Федерации

ФГБОУ ВО «Ивановский государственный университет»

Институт математики, информационных технологий и естественных наук

Кафедра информационных технологий

и прикладной математики

|  |
| --- |
| **Утверждаю:**  Зав. кафедрой ИТ и ПМ  канд.экон.наук, доцент\_\_\_\_\_\_\_\_\_Данилова С.В. |
| «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024год |

**КАЛЕНДАРНЫЙ ГРАФИК**

выполнения выпускной квалификационной работы

(бакалаврской работы)

студента института математики, информационных технологий и естественных наук Калабина Александра Николаевича

(фамилия, имя и отчество студента)

Тема работы: Тест производительности нейронных сетей на примере CIFAR–10

(утверждена приказом ИвГУ № 2п/283 от «29» декабря 2023г.)

|  |  |  |
| --- | --- | --- |
| Характер и объем работ | Плановый срок выполнения | Отметки  о выполнении |
| 1. Сбор информации и изучение  литературы | 15.05.2024 г. |  |
| 2. Обработка и анализ материалов  для написания ВКР | 20.05.2024 г. |  |
| 3. Написание практического раздела | 25.05.2024 г. |  |
| 4. Написание аналитического раздела | 31.05.2024 г. |  |
| 6. Прохождение нормоконтроля | 6.06.2024 г. |  |
| 7. Предоставление ВКР на рецензию | 8.06.2024 г. |  |
| 8. Подготовка презентационного материала | 12.06.2024 г. |  |
| 9. Предоставление ВКР на кафедру  для допуска к защите | 16.06.2024 г. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Дата выдачи календарного графика: | «3» мая 2024 г. | | |
| Научный руководитель,  канд. физ.–мат. наук, доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |  | Хашин С.И. |
| Календарный график принял к исполнению,  студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |  | Калабин А.Н. |

**CОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 7](#_Toc168699705)

[**1. Создание модели нейронной сети на примере CIFAR–10 и тестирование полученной модели.** 8](#_Toc168699706)

[1.1 Нейронные сети, TensorFlow и CIFAR–10 8](#_Toc168699707)

[1.2 Создание модели нейронной сети 11](#_Toc168699708)

[**1.2.1 Подготовительный этап создания** 11](#_Toc168699709)

[**1.2.2 Определение параметров** 11](#_Toc168699710)

[**1.2.4 Создание модели** 14](#_Toc168699711)

[**1.2.5 Проверка модели** 17](#_Toc168699712)

[1.3 Поиск пределов 18](#_Toc168699713)

[1.4 Вывод по главе 1 20](#_Toc168699714)

[**2. Создание программы–теста производительности нейронных сетей** 21](#_Toc168699715)

[2.1 Предпосылки для создания программы и требования к программе 21](#_Toc168699716)

[**2.1.1 Сохранение логов тестирования** 23](#_Toc168699717)

[**2.1.2 Улучшение читаемости лога, звуковое оповещение** 25](#_Toc168699718)

[**2.1.3 Информация о системе, балловая система** 28](#_Toc168699719)

[2.2 Flet Framework 31](#_Toc168699720)

[**2.2.1 Создание окна приложения** 33](#_Toc168699721)

[**2.2.2 Создание навигационной панели для приложения** 34](#_Toc168699722)

[**2.2.3 Создание диалоговых окон и визуальной системы оповещения** 37](#_Toc168699723)

[**2.2.4 Создание текстовых редакторов для отображения информации** 41](#_Toc168699724)

[**2.2.5 Заполнение вкладок программы** 44](#_Toc168699725)

[**2.2.6 Сборка настольного приложения** 47](#_Toc168699726)

[**2.2.7 Создание установщика для программы** 48](#_Toc168699727)

[2.3 Выводы по главе 2 49](#_Toc168699728)

[**3. Тестирование зависимостей и различных ПК с помощью программы** 50](#_Toc168699729)

[3.1 Тестирование зависимостей 51](#_Toc168699730)

[**3.1.1 Тестирование зависимости от частоты ядер, количества ядер и многопоточности** 52](#_Toc168699731)

[**3.1.2 Тестирование зависимости от количества планок ОЗУ и частоты** 53](#_Toc168699732)

[**3.2.1 AMD Ryzen 5 5600X, 32 Gb DDR4 3800 MHz** 54](#_Toc168699733)

[**3.2.2 AMD Ryzen 5 3600, 16 Gb DDR4 3200 MHz** 55](#_Toc168699734)

[**3.2.3 AMD FX 6300, 16 Gb DDR3 2000 MHz** 56](#_Toc168699735)

[**3.2.4 AMD Ryzen 7 5800H, 32 Gb DDR4 3200 MHz** 57](#_Toc168699736)

[**3.2.5 AMD Ryzen 3 3200U, 12 Gb DDR4 2400 MHz** 58](#_Toc168699737)

[**3.2.6 AMD Athlon Gold 3150U, 16 Gb DDR4 2133 MHz** 59](#_Toc168699738)

[**3.2.7 Intel Core i3 3120m, 6Gb DDR3 1600 MHz** 60](#_Toc168699739)

[**3.2.8 AMD Ryzen 5 5500, 16 Gb DDR4 3200 MHz** 61](#_Toc168699740)

[**3.2.9 AMD Ryzen 7 5800X3D, 32 Gb DDR4 3600 MHz** 62](#_Toc168699741)

[**3.2.10 AMD Ryzen AMD Ryzen 7 7800X3D, 32 Gb DDR5 6000 MHz** 63](#_Toc168699742)

[**3.2.11 Intel Core i3 10105F, 16 Gb DDR4 3200 MHz** 64](#_Toc168699743)

[**3.2.12 Intel Core i5 10600K, 16 Gb DDR4 3600 MHz** 65](#_Toc168699744)

[**3.2.13 Intel Xeon CPU E5 2696 v3, 32 Gb DDR3 2133 MHz** 66](#_Toc168699745)

[**3.2.14 Intel Core i7 14700K, 64 Gb DDR5 6400 MHz** 67](#_Toc168699746)

[3.3 Выводы по итогам тестирования 68](#_Toc168699747)

[**Заключение** 69](#_Toc168699748)

[**Список литературы** 70](#_Toc168699749)

[**СПИСОК СОКРАЩЕНИЙ** 72](#_Toc168699750)

[**Приложение 1** 73](#_Toc168699751)

[**Приложение 2** 75](#_Toc168699752)

[**Приложение 3** 77](#_Toc168699753)

[**Приложение 4** 94](#_Toc168699754)

[**Приложение 5** 95](#_Toc168699755)

# **ВВЕДЕНИЕ**

В последние время нейронные сети стали неотъемлемой частью современных технологий, они находят применение в самых различных областях, от распознавания образов до автономных транспортных средств. С развитием глубокого обучения и доступности вычислительных ресурсов возникла потребность в оценке производительности нейронных сетей и их эффективности в различных задачах.

Целью данной работы является создания теста производительности нейронных сетей на примере CIFAR–10, который сможет использовать даже пользователь не знакомый с программированием.

Основные задачи работы включают в себя:

1. Создание модели нейронной сети на примере CIFAR–10;
2. Интеграция системы оценки производительности нейронной сети;
3. Создание пользовательского интерфейса программы;
4. Создание простого и понятного установщика для программы;
5. Тестирование готовой программы;
6. Определение минимальных системных требований.

Реализация данной программы позволит пользователям быстро и эффективно оценивать производительность нейронных сетей на своем ПК без необходимости установки и настройки сложных зависимостей, что сделает процесс тестирования доступным и удобным. Это поможет привлечь больше людей к изучению и созданию нейронных сетей. На момент начала написания работы программа не имеет аналогов.

# **1. Создание модели нейронной сети на примере CIFAR–10 и тестирование полученной модели.**

## 1.1 Нейронные сети, TensorFlow и CIFAR–10

Для достижения цели будет использован язык программирования **python**, библиотека **TensorFlow**, набор данных **CIFAR–10**. В своей работе я буду использовать сверточную нейронную сеть. Для создания своей модели нейронный сети будут использованы **python** версии 3.11.9, **TensorFlow** версии 2.16 и **Keras** версии 3.0, а для загрузки **CIFAR–10** будет использован **TDFS Nightly** версии 4.9.4.

Нейронные сети представляют собой математические модели, вдохновленные биологическими нейронами. Они являются ключевым компонентом в области искусственного интеллекта и машинного обучения, способными решать сложные задачи распознавания образов, классификации, прогнозирования и многого другого.

Нейронная сеть состоит из множества искусственных нейронов, организованных в слои: входной слой, скрытые слои и выходной слой:

* **Входной слой** принимает исходные данные.
* **Скрытые слои** осуществляют нелинейное преобразование входных данных, чтобы выделить важные признаки и паттерны.
* **Выходной слой** формирует конечный результат, который интерпретируется как ответ сети.

Каждый нейрон получает входные данные, применяет к ним весовые коэффициенты, суммирует и передает результат через нелинейную активационную функцию, что позволяет модели обучаться сложным нелинейным зависимостям.

Нейронные сети делятся на типы:

* **Полносвязные нейронные сети (Fully Connected Neural Networks, FCNN)**: Каждый нейрон одного слоя соединен с каждым нейроном следующего слоя.
* **Сверточные нейронные сети (Convolutional Neural Networks, CNN)**: специализируются на обработке изображений, используя сверточные слои для выделения пространственных признаков.
* **Рекуррентные нейронные сети (Recurrent Neural Networks, RNN)**: применяются для обработки последовательных данных, таких как временные ряды и текст, учитывая зависимость предыдущих состояний.

**TensorFlow** – это открытая библиотека машинного обучения, разработанная компанией Google для создания и обучения нейронных сетей и других моделей машинного обучения. Она предоставляет широкий спектр инструментов и возможностей для работы с данными, построения моделей, их обучения, оценки и использования в продуктивной среде. Больше информации об **TensorFlow** вы можете узнать на официальном сайте [9]**.**

**Keras** – это высокоуровневый интерфейс для создания и обучения нейронных сетей, который работает поверх различных библиотек машинного обучения, таких как TensorFlow, Theano и Microsoft Cognitive Toolkit (CNTK). Он предоставляет простой и интуитивно понятный API для построения и обучения различных типов нейронных сетей, включая сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и их комбинации. Больше информации об **Keras** вы можете узнать на официальном сайте [10].

**TensorFlow Datasets (TFDS)** – это библиотека, предназначенная для загрузки и использования общедоступных наборов данных для обучения моделей машинного обучения. TDFS Nightly – это версии программного обеспечения, которые создаются автоматически каждую ночь из последних изменений в репозитории.

Набор данных **CIFAR–10** (Canadian Institute for Advanced Research) — это набор изображений, который широко используется в машинном обучении и компьютерном зрении для обучения и тестирования алгоритмов классификации изображений. Набор состоит из 60000 цветных изображений размером 32x32 пикселя, разделенных на 10 классов: самолеты, автомобили, птицы, кошки, олени, собаки, лягушки, лошади, корабли и грузовики. Каждый класс содержит по 6000 изображений. Больше информации вы можете найти на сайте университета компьютерных исследований в Торонто [6] (рис. 1).

CIFAR–10 часто используется для демонстрации и оценки алгоритмов машинного обучения и глубокого обучения, таких как сверточные нейронные сети (CNN). В силу своей относительной небольшой размерности и разнообразия классов, CIFAR–10 хорошо подходит для исследований и экспериментов в области компьютерного зрения.

В данном разделе будет описан процесс создания модели нейронный сети с использованием набора данных CIFAR–10 для последующего использования в создании теста производительности нейронных сетей.

Рисунок 1. CIFAR–10 визуализация

## 1.2 Создание модели нейронной сети

### **1.2.1 Подготовительный этап создания**

Для начала выбираем удобную вам среду разработки, я рекомендую использовать Visual Studio Code. Информацию о вы можете найти на официальном сайте Visual Studio Code [12]. В первую очередь необходимо загрузить модули для этого в терминале вам следует вписать вот эти вот команды:

pip install tensorflow

pip install keras

pip install tdfs

Для начала нужно загрузить, а затем импортировать TensorFlow, Keras и TDFS:

import tensorflow as tf

import tensorflow\_datasets as tfds from tensorflow.keras

### **1.2.2 Определение параметров**

Для удобства использования поместим модель в функцию neuro. Параметр B будет определять количество эпох обучения модели, а параметр A количество нейронов. Начнем с определения параметров:

def neuro(A, B):

batch\_size = 64

num\_classes = 10

epochs = B

Теперь разберем, что делает каждая из строк:

* **batch\_size** – определяет размер пакета данных, используемого для обновления весов модели за одну итерацию. Определяет количество образцов, обрабатываемых перед тем, как обновятся параметры модели.

Размер пакета (**batch size**) в машинном обучении влияет на скорость обучения, стабильность и точность градиентов, потребление памяти, генерализацию и регулировку гиперпараметров. Малый размер пакета приводит к более частому обновлению весов с большей стохастичностью, что может улучшить генерализацию, но требует больше времени и меньше памяти. Большой размер пакета делает градиенты более стабильными и точными, ускоряет процесс обучения в терминах времени, но требует больше памяти и может привести к переобучению. Оптимальный размер пакета зависит от конкретной задачи и требует подбора. Эта информация подробно описана в книге Майкла Нильсена "Neural Networks and Deep Learning"[7].

* **num\_classes** – определяет количество классов для классификации. В нашем случае их 10. Подробное объяснение классификации и обработки многоклассовых задач можно найти в книге Яна Гудфеллоу "Deep Learning" [5].
* **epochs** – определяет количество эпох обучения модели, передаваемое как параметр B.

Эпоха (**epoch**) – это один проход через все обучающие данные в обучающем наборе. Во время каждой эпохи модель проходит через все обучающие примеры, вычисляя потери и корректируя веса с помощью алгоритма оптимизации, такого как градиентный спуск. Увеличение числа эпох может помочь модели лучше адаптироваться к обучающим данным и улучшить ее производительность. В моём случае обучение продолжатся до тех пор, пока не будет достигнуто заданное в параметре B количество эпох. Этот процесс подробно описан в книге Кристофера Бишопа "Pattern Recognition and Machine Learning"[1]

#### **1.2.3 Загрузка и предобработка данных**

После того как все необходимые импорты были добавлены, а классы определены необходимо загрузить и предобработать данные набора CIFAR–10. Для этого я сделал функцию **load\_data**:

def load\_data():

def preprocess\_image(image, label):

image = tf.image.convert\_image\_dtype(image, tf.float32)

return image, label

ds\_train, info = tfds.load("cifar10", with\_info=True, split="train", as\_supervised=True)

ds\_test = tfds.load("cifar10", split="test", as\_supervised=True)

ds\_train = ds\_train.repeat().shuffle(A).map(preprocess\_image).batch(batch\_size)

ds\_test = ds\_test.repeat().shuffle(A).map(preprocess\_image).batch(batch\_size)

return ds\_train, ds\_test, info

В данном случае **preprocess\_image** является вспомогательной функцией для предобработки изображений, преобразующая изображения в тип данных float32. Это нормализует значения пикселей изображения, чтобы они были в диапазоне [0, 1].

Для того чтобы загрузить CIFAR–10 нужно использовать функцию **tfds.load** для загрузки датасетов из TensorFlow Datasets и распределить на обучающие (**ds\_train)** и **(ds\_test)** тестовые выборки:

* **split="train" и split="test"** – они указывают, что мы загружаем тренировочную и тестовую части набора данных соответственно.
* **with\_info=True** – возвращает дополнительную информацию о наборе данных, такую как количество образцов и структура данных.
* **as\_supervised=True** – возвращает данные в формате пар (изображение, метка).
* **repeat и shuffle** – это методы для повторения и перемешивания данных перед обучением. **repeat** позволяет использовать данные бесконечно, что полезно при определении количества шагов на эпоху. **shuffle** перемешивает данные для того, чтобы модель не училась на данных в определенном порядке.
* **map** – это метод для применения предобработки к данным. Здесь он используется для преобразования изображений в формат float32.
* **batch** – это метод для разбиения данных на батчи (пакеты). Размер батча определен параметром batch\_size. В нашем случае равен 64.

Этот процесс загрузки и предобработки данных подробно объясняется в книге Франсуа Шолле "Deep Learning with Python"[2], где приводятся примеры и рекомендации по работе с датасетами и их подготовке для обучения моделей нейронных сетей. Также, базовые концепции и применение методов обработки данных освещены в книге Яна Гудфеллоу "Deep Learning"[5].

### **1.2.4 Создание модели**

Я буду использовать тип модели **Sequential,** который представляет собой стек слоев, через который данные проходят последовательно и слои Conv2D, MaxPooling2D, Dense, Dropout, Activation, Flatten. Сначала добавим в импорты все необходимое:

import Input from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Activation, Flatten

from keras.models import Sequential

Теперь создадим модель нейронной сети с использованием Keras Sequential API и для удобства поместим её в функцию **create\_model**:

def create\_model(input\_shape):

1. Сначала создадим пустую модель нейронной сети **Sequential**, которую далее мы будем заполнять различными слоями:

model = Sequential()

1. Первым шагом добавим входной слой, который далее будем заполнять данными, получаемыми во время обучения на других слоях:

model.add(Input(shape=input\_shape))

1. Далее добавляем сверточные слои, которые применяют фильтры к входным данным для извлечения пространственных признаков. Каждый фильтр скользит по данным, вычисляя скалярное произведение и формируя карты признаков. Они помогают сетям эффективно обрабатывать изображения, сохраняя их пространственную структуру. Сначала добавляем сверточный слой с 32 фильтрами размера (3, 3) и заполнением "**same**", а затем слой с активационной функцией **ReLU** после каждого сверточного слоя:

model.add(Conv2D(filters=32,kernel\_size=(3,3),

padding="same"))

model.add(Activation("relu"))

model.add(Conv2D(filters=32,kernel\_size=(3,3),

padding="same"))

model.add(Activation("relu"))

1. Добавляем слои пулинга и регуляризации. Слои пулинга упрощают представление данных, сокращая размерность, и помогают снизить вычислительную сложность модели. Слои регуляризации предотвращают переобучение, добавляя ограничения на параметры сети, такие как случайное обнуление выходных данных (**Dropout**), что способствует обобщению модели:

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

model.add(Dropout(0.25))

1. Повторяем шаги 3–4 для фильтров 64 и 128, что позволит извлечь больше различных признаков модели и в нашем случае улучшить ее способность к классификации. Полный код модели вы можете найти в **Приложении 1**.
2. Добавляем слой Flatten, который преобразует выходные данные предыдущего слоя в одномерный вектор:

model.add(Flatten())

1. Добавляем полносвязный слой с параметром **A,** который задает количество нейронов, а так же слой с активационной функцией **ReLU** и регуляризации:

model.add(Dense(A))

model.add(Activation("relu"))

model.add(Dropout(0.5))

1. Добавляем полносвязный слой вывода с количеством нейронов, равным числу классов, и активационной функцией softmax для прогнозирования вероятностей принадлежности к каждому классу, так же добавляем метод, который выводит сводку (**summary**) структуры модели, включая информацию о каждом слое, количество параметров и форму вывода каждого слоя:

model.add(Dense(num\_classes, activation="softmax"))

model.summary()

1. И наконец компилируем полученную модель с функцией потерь sparse\_categorical\_crossentropy, оптимизатором adam и метрикой accuracy:

model.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

return model

Оптимизатор Adam (Adaptive Moment Estimation) – это алгоритм оптимизации, который адаптивно регулирует скорость обучения для каждого параметра на основе первого и второго моментов градиента. Это позволяет ему эффективно обучаться на данных с разной структурой и распределением градиентов.

### **1.2.5 Проверка модели**

Для того чтобы проверить модель нам нудно сначала создать функцию вызова **load\_data()**, которая загружает и предварительно обрабатывает данные для обучения и тестирования модели.

ds\_train, ds\_test, info = load\_data()

Далее создаем нейронную сеть используя функцию **create\_model()**, которая определяет архитектуру модели:

model = create\_model(input\_shape=info.features["image"].shape)

Используем метод **fit()** для обучения модели. **epochs –** этоколичество эпох обучения, которые задаются параметром **B**. Используем данные для обучения (ds\_train) и проверки (ds\_test). Параметр verbose управляет уровнем вывода информации о прогрессе обучения, для проверки модели активируем его, установив на 1. Параметры steps\_per\_epoch и validation\_steps указывают количество шагов обучения и проверки соответственно:

model.fit(ds\_train, epochs=epochs, validation\_data=ds\_test, verbose=1,

steps\_per\_epoch=info.splits["train"].num\_examples // batch\_size,

validation\_steps=info.splits["test"].num\_examples // batch\_size,)

Вызовем полученную функцию neuro с параметрами (2048,48).

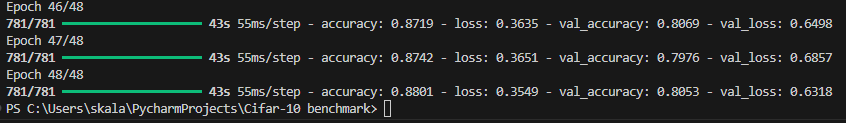


Рисунок 2. Результаты обучения

Полученная модель способна показывать точность свыше 87% на данных валидации, модель хорошо обучена и демонстрирует способность делать точные предсказания на новых данных, которые она ранее не видела, модель успешно обобщает знания, полученные в процессе обучения, на новые ситуации (рис. 2).

Эти шаги по созданию и обучению модели нейронной сети описаны в книге Франсуа Шолле "Deep Learning with Python"[2] и Яна Гудфеллоу "Deep Learning"[5], где рассматриваются ключевые аспекты и методы работы с нейронными сетями, включая архитектуру, оптимизацию и оценку моделей.

## 1.3 Поиск пределов

В этом разделе мы будем искать количество эпох, при котором перестает улучшаться точность полученной модели, потому что это необходимо для того чтобы позже определить уровни сложности для теста производительности. Для достижения данной цели будет установлен **tensorflow–gpu** –модуль позволяющий использовать видеокарту при обучении модели нейронной сети. Это необходимо так как сильно ускорит процесс поиска, потому что даже среднеуровневые видеокарты быстрее флагманских процессоров в несколько десятков раз. Так происходит, потому что видеокарты используют гораздо более быструю память нежели оперативную память которую используют процессоры. Но, к сожалению, пока интегрировать в программу его поддержку не выйдет, слишком много зависимостей, которые не получится решить в рамках одной программы и работает данный модуль только с видеокартами NVIDIA. Существует ещё модуль **tensorflow–directml**, но в данный момент его разработка приостановлена.

Характеристики тестового стенда:

* Видеокарта: NVIDIA GeForce RTX 3060Ti
* Накопитель: SSD M.2 Nvme 500Gb
* Операционная система: Windows 11 23H2

В качестве результата будет браться среднее значение, полученное по результатам 10 тестов для каждого значения нейронов округленное в большую сторону.

Таблица 1.

Тестирование NVIDIA GeForce RTX 3060Ti

|  |  |  |  |
| --- | --- | --- | --- |
| Нейроны | Время  (в секундах) | Номер эпохи при которой точность перестает улучшаться | Точность  (в процентах) |
| 64 | 43 | 7 | 71 |
| 128 | 45 | 11 | 74 |
| 256 | 41 | 17 | 76 |
| 512 | 50 | 25 | 80 |
| 1024 | 47 | 33 | 86 |
| 2048 | 89 | 49 | 88 |

Время я указал только в целях демонстрации того насколько быстро видеокарта способна обучить нейронную сеть.

## 1.4 Вывод по главе 1

Для создания модели нейронной сети на примере CIFAR–10 был использован Sequential API из Keras. Модель содержит несколько слоев Conv2D, MaxPooling2D, Dense и Dropout, которые обеспечивают сверточную часть архитектуры нейронной сети.

Проведено тестирование полученной модели и найдены номера эпох при которых её точность перестает улучшаться при заданном количестве нейронов. Исходя из полученных данных я решил, что оптимальным решением будет реализовать в программе 5 уровней сложности, от уровня сложности будет зависеть передаваемое количество нейронов и эпох:

* Тестовая (1 эпоха 1 нейрон);
* Низкая (8 эпох 64 нейронов);
* Средняя (16 эпох 256 нейронов);
* Высокая (32 эпох 1024 нейронов);
* Очень высокая (48 эпох 2048 нейронов).

# **2. Создание программы–теста производительности нейронных сетей**

## 2.1 Предпосылки для создания программы и требования к программе

Идея о создании теста производительности нейронных сетей появилась во время написания одной из научно–исследовательских работ. От меня требовалось проверить производительность нейронных сетей на примере CIFAR–10 на различных компьютерах. Устанавливать среду разработки и устранять зависимости на каждом компьютере было долго и муторно, я пытался найти готовое решение, которое позволяло бы решить эту проблему, но как оказалось такого еще нет. Тогда я начал изучать различные материалы на тему разработки графических интерфейсов и решил, чего бы мне хотелось видеть в ней и сделал её первый концепт арт (рис. 3).

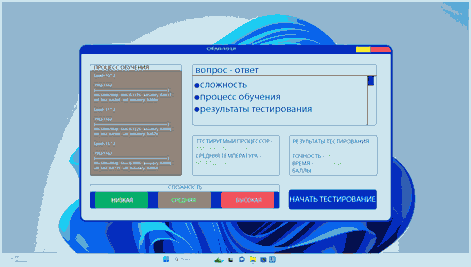


Рисунок 3. Первый концепт арт

Рассмотрев множество способов реализации графического интерфейса, был выбран Flet Framework. Для достижения моей цели это оптимальный выбор. Flet Framework прост в освоении и имеет все необходимое для создания графического интерфейса и сборки готовой программы. В первую очередь я изучил краткую документацию от пользователя Solter21 [8].

В первой главе была подготовлена нейронная сеть, которую можно использовать в программе. Теперь опишем требования к программе:

* Разработать механизм для записи логов тестирования в файл для последующего анализа.
* Внедрить форматирование логов для улучшения читаемости.
* Добавить звуковые оповещения о завершении этапов обучения.
* Реализовать отображение информации о системе, на которой проводится обучение.
* Разработать балловую систему для оценки производительности различных конфигураций.
* Рассмотреть преимущества использования Flet Framework для разработки пользовательских интерфейсов.
* Создать основное окно приложения для отображения результатов обучения и информации о системе.
* Разработать навигационную панель для удобства перемещения между различными разделами приложения.
* Реализовать диалоговые окна и систему оповещения для интерактивного взаимодействия с пользователем.
* Разработать средства для отображения логов и другой важной информации.
* Написать теоретическую справку с помощью, которой пользователь сможет написать свою модель нейронной сети.
* Написать информацию о программе в самой программе.
* Разработать установщик для программы.

В третьей описан процесс тестирования программы и по его итогам сформированы минимальные системные требования.

### **2.1.1 Сохранение логов тестирования**

Лог в соответствии со сложностью и временем начала обучения будет сохраняться в папке **CIFAR–10 benchmark** ввиде текстового файлакоторая будет создаваться в папке пользователя на основном накопителе данных. Для перенаправления вывода в файл будем использовать модуль **sys**. Чтобы работать с папками в **python** необходим модуль **os**, а чтобы папка называлась в соответствии с актуальным временем нужен модуль **datetime** который создаёт объекты, представляющие текущее время или определенную дату и время. Добавим соответствующие импорты:

import os

import sys

import datetime

Сначала создадим переменную **user\_path** в которой будем хранить информацию о расположении папки пользователя. Далее создадим переменную **cifar\_benchmark\_path** в которой будем хранить информацию о расположении папки CIFAR–10 benchmark:

user\_path = os.path.join(os.path.expanduser("~"))

cifar\_benchmark\_path = os.path.join(user\_path, "CIFAR–10 benchmark")

Теперь создадим переменную, которая будет сохранять информацию о базовой директории программы:

base\_result\_dir = os.path.join(cifar\_benchmark\_path, "Результаты")

Далее работаем с сделанной в первой главе функцией neuro. Сначала добавим строку, которая получает текущий рабочую директорию:

current\_dir = os.getcwd()

Создадим переменные **current\_time** и **folder\_name**, чтобы позже использовать для присвоение имени папки в соответствии с актуальным на компьютере времени:

current\_time = datetime.datetime.now()

folder\_name = current\_time.strftime("%Y–%m–%d\_%H–%M–%S")

Далее создадим цикл который определяет сложность задачи на основе количества эпох, которые вы передаете в функцию **neuro(A, B)**. Переменная **complexity** принимает одно из определенных значений, указывающих на уровень сложности задачи.

if epochs == 1:

complexity = "Тестoвая сложность"

elif epochs == 8:

complexity = "Низкая сложность"

elif epochs == 16:

complexity = "Средняя сложность"

elif epochs == 32:

complexity = "Высокая сложность"

elif epochs == 48:

complexity = "Очень высокая сложность"

Теперь создадим переменную **result\_dir**, котораясоздает путь к директории результатов, объединяя базовый путь к результатам, уровень сложности и текущее время. Так же создадим директорию для результатов по указанному пути, аргумент **exist\_ok=True** гарантирует, что если директория уже существует, она не будет перезаписана.

result\_dir = os.path.join(base\_result\_dir, complexity, folder\_name)

os.makedirs(result\_dir, exist\_ok=True)

И в теперь изменим текущий рабочий каталог на только что созданную директорию результатов.

os.chdir(result\_dir)

Обновим функцию **load\_data()**, которая находится в **neuro(A,B)** так чтобы вывод перенаправлялся в текстовый файл **log.txt**:

sys.stdout = open('log.txt', 'w', encoding='utf–8')

print('Тестирование начато:', current\_time)

Запустим нашу функцию neuro с параметрами 1,1:

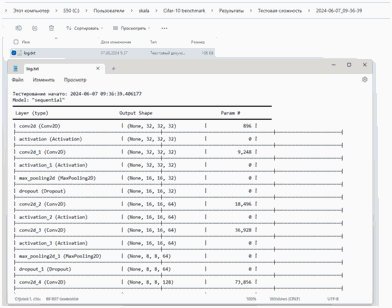


Рисунок 4. Результат запуска neuro(1,1)

Как мы видим на рисунке 4 файл log.txt содержащий информацию из вывода создался по пути C:\Users\skala\Cifar–10 benchmark\Результаты\Тестoвая сложность\2024–06–07\_09–36–39 .

### **2.1.2 Улучшение читаемости лога, звуковое оповещение**

Рассмотрев файл log.txt модно заметить непонятные квадраты, сложно читаемые строки (рис. 5).

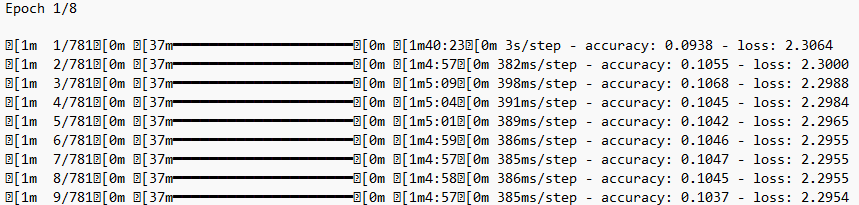


Рисунок 5. log.txt

Для того чтобы это исправить необходимо кастомизировать обратный вызов и для этого мы используем **Keras CustomCallback.**

**CustomCallback в Keras** − это пользовательский обратный вызов, который позволяет определить и настроить собственную логику во время обучения модели. Обратные вызовы представляют собой объекты, которые передаются методу **fit()** модели и выполняются в различных точках во время обучения (например, в начале эпохи, в конце эпохи, в начале обучения и т. д.). Код класса **CustomCallback** вы можете найти в **Приложении 2**, а тут мы рассмотрим, что делают использованные его методы:

1. **on\_train\_batch\_end(self, batch, logs=None)** − этот метод вызывается в конце каждого батча в процессе обучения модели. Он записывает информацию о текущем шаге обучения, такую как время выполнения шага, точность на текущем батче и количество шагов в секунду, в файл логов **log.txt**. Затем он вычисляет оставшееся время обучения и также записывает его в файл логов.
2. **on\_epoch\_end(self, epoch, logs=None)** − этот метод вызывается в конце каждой эпохи в процессе обучения модели. Он записывает информацию о текущей эпохе, такую как время выполнения эпохи, точность на текущей эпохе и количество эпох в секунду, в файл логов **log.txt**. Затем он вычисляет оставшееся время обучения (время на эпоху, умноженное на количество оставшихся эпох) и также записывает его в файл логов.

Теперь обновим функцию **neuro**, чтобы файл **log.txt** был более информативен, для этого добавим модуль **time** с помощью которого сделаем таймер. Добавим импорт класса History() и Callback для Tensorflow, которые позволит работать с полученными входе обучения данными. Так же добавим звуковое оповещение с помощью модуля **beepy**, которое будет появляться в момент завершения обучения:

from keras.callbacks import History, Callback

import beepy

import time

Перед **model.fit** в нашей функции neuro добавим активацию таймера и присвоим класс History() переменной history:

history = History()

tic = time.perf\_counter()

Обновляем **model.fit** добавляя функцию обратного вызова callbacks присваивая ей значение history и CustomCallback():

model.fit(ds\_train, epochs=epochs, validation\_data=ds\_test, verbose=0,

steps\_per\_epoch=info.splits["train"].num\_examples // batch\_size,

validation\_steps=info.splits["test"].num\_examples // batch\_size, callbacks=[history, CustomCallback()])

После **model.fit** останавливаем таймер в переменным **scores** и **scores2** присваиваем точность полученной модели, перед тем как вывести краткую информацию о результатах завершим, а потом опять перенаправлять вывод в файл **log.txt.** Вконце добавим звуковой сигнал и завершим вывод:

toc = time.perf\_counter()

scores = history.history['accuracy']

scores2 = float(scores[–1])

sys.stdout.close()

sys.stdout = open('log.txt', 'a', encoding='utf–8')

print(' ')

print(f'Обучение заняло {toc – tic:0.2f} секунд')

print(f'Точность модели: %.2f%%' % (scores2 \* 100))

beepy.beep(sound=1)

sys.stdout.close()

Запустим neuro с параметрами (1,1), чтобы посмотреть, как выглядит обновленная версия файла log.txt (рис. 6):

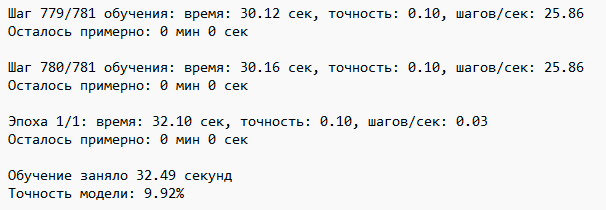


Рисунок 6. Обновленный log.txt

### **2.1.3 Информация о системе, балловая система**

Сначала создадим балловую систему, которая в зависимости от количества эпох (**epochs**), вычисляет коэффициент сложности задачи (**complexity\_coefficient**). Затем вычисляет максимальное количество баллов (**max\_points**), учитывая точность модели, коэффициент сложности и время обучения. Кроме того, вычисляется коэффициент времени обучения (**training\_time\_coefficient**), который зависит от того, насколько быстро завершилось обучение по сравнению с 1 часом. Затем общее количество баллов (**total\_points**) округляется.

if epochs == 1:

complexity\_coefficient = 1.0

elif epochs == 8:

complexity\_coefficient = 1.2

elif epochs == 16:

complexity\_coefficient = 1.5

elif epochs == 32:

complexity\_coefficient = 2.0

elif epochs == 48:

complexity\_coefficient = 2.5

accuracy = scores2 \* 100

max\_points = accuracy \* complexity\_coefficient \* 100

training\_time\_coefficient = max(0.1, 1.0 – (toc – tic) / 3600)

total\_points = round(max\_points \* training\_time\_coefficient)

Чтобы получить информацию о версии операционной системы, модели процессора и объём озу мною были использованы модули psutil и winreg. Добавим необходимые импорты:

import psutil

import winreg

Теперь создадим функцию get\_windows\_version с помощью которой получим информацию о версии используемой пользователем операционной системы. Эта функция использует модуль **winreg**, чтобы получить информацию о версии операционной системы Windows, включая название продукта, издание и номер сборки. Если номер сборки больше или равен 22000, то считается, что это Windows 11:

def get\_windows\_version():

key = r"SOFTWARE\Microsoft\Windows NT\CurrentVersion"

with winreg.OpenKey(winreg.HKEY\_LOCAL\_MACHINE, key) as key\_handle:

product\_name = winreg.QueryValueEx(key\_handle, "ProductName")[0]

edition = winreg.QueryValueEx(key\_handle, "EditionID")[0]

build\_number = int(winreg.QueryValueEx(key\_handle, "CurrentBuildNumber")[0])

if build\_number >= 22000:

return f"Windows 11 {edition} Edition Build {build\_number}"

else:

return f"{product\_name} {edition} Edition Build {build\_number}"

windows\_version = get\_windows\_version()

Cоздадим функцию **get\_cpu\_name** чтобы получить информацию о процессоре компьютера:

def get\_cpu\_name():

key = r"HARDWARE\DESCRIPTION\System\CentralProcessor\0"

with winreg.OpenKey(winreg.HKEY\_LOCAL\_MACHINE, key) as key\_handle:

processor\_name = winreg.QueryValueEx(key\_handle, "ProcessorNameString")[0]

return processor\_name

processor\_name = get\_cpu\_name()

Создадим переменную **ram** в которой будем хранить информацию об ОЗУ:

ram = round(psutil.virtual\_memory().total / (1024 \*\* 3) , 2)

Полученные данные мы будем записывать в файл **stats.txt**:

sys.stdout = open('stats.txt', 'w', encoding='utf–8')

print("Модель процессора:", processor\_name)

print("Объем оперативной памяти (RAM):", ram, "GB")

print("Версия операционной системы:", windows\_version)

print('Получено баллов:', total\_points)

print(f'Обучение заняло {toc – tic:0.2f} секунд')

print(f'Точность модели: %.2f%%' % (scores2 \* 100))

now = datetime.datetime.now()

print('Тестирование завершено:', now)

beepy.beep(sound=1)

sys.stdout.close()

return 0

Рассмотрим полученный файл stats.txt(рис. 7):

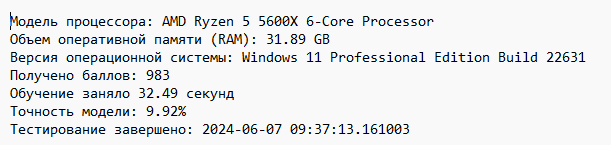


Рисунок 7. Файл stats.txt

**2.2 Flet Framework**

Flet Framework – это инструмент для разработки кроссплатформенных программ, созданный на основе языка программирования Dart и использующийся в комбинации с Flutter. Flet Framework облегчает процесс разработки программы, предоставляя набор готовых компонентов, библиотек и инструментов для создания интерфейсов. Он помогает разработчикам создавать красивые и функциональные программы для различных платформ и операционных таких как **Linux** и **Windows**, с минимальными затратами времени и ресурсов, благодаря этой особенности я планирую портировать свою программу на отечественные операционные системы на основе **Linux.**

В данном разделе будет описан процесс создания интерфейса моей программы. **Flet Framework** достаточно молодой и информации по нему мало в интернете, а книг и вовсе нет, поэтому информация из моей работы будет полезна тем кто хочет начать работать с Flet. Больше информации о нем вы можете найти на официальном сайте [4].

Рассмотрены плюсы, из–за которых был выбран данный фреймворк.

Основные плюсы Flet Framework:

1. **Удобство разработки** − предоставляет интуитивно понятные инструменты и компоненты, что делает процесс разработки простым и эффективным.
2. **Кроссплатформенность** − с помощью Flet Framework вы можете создавать программы, которые могут работать как на Linux, так и на Windows, используя общий код и ресурсы.
3. **Высокая производительность** − благодаря компиляции в нативный код и эффективной работе с графическим интерфейсом, программы, разработанные с использованием Flet Framework, обладают высокой производительностью и отзывчивостью. Например, моя графическая оболочка потребляет около 250 мегабайт оперативной памяти.
4. **Поддержка Material Design и Cupertino Design** − Flet Framework позволяет создавать приложения с использованием стандартных дизайн–языков от Google (Material Design) и Apple (Cupertino Design), что помогает им интегрироваться с экосистемой каждой платформы.

### **2.2.1 Создание окна приложения**

Сначала создадим пустое окно, в котором будет отображаться программа определив функцию **main**, которая принимает объект **page** типа **ft.Page** и не возвращает ничего (**None**) и зададим название программы **page.title**:

def main(page: ft.Page) –> None:

page.title = 'CIFAR–10 benchmark'

Зададим мааксимальные и минимальные размеры окна по высоте **page.window\_max\_height, page.window\_height** и ширине **page.window\_max\_width, page.window\_min\_height**:

page.window\_max\_height = 1440

page.window\_max\_width = 2560

page.window\_width = 1366

page.window\_height = 768

page.window\_min\_height = 600

page.window\_min\_width = 800

Установим возможность изменения размеров окна программы пользователем и после этого обновим страницу.

page.window\_resizable = True

page.update()

И запускаем нашу программу используя main в качестве цели для создания интерфейса (рис. 8):

if \_\_name\_\_ == '\_\_main\_\_':

ft.app(target=main)

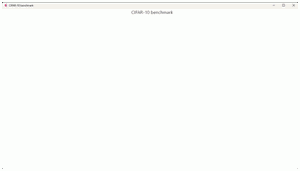
****

Рисунок 8. Пустое окно для будущей программы

### **2.2.2 Создание навигационной панели для приложения**

Создадим навигационную используя метод **ft.NavigationRailDestination** для простого перехода пользователя по функционалу приложения для удобства поместим его в переменную **rail**. Выбрать данный метод навигации меня вдохновил пользователь Gece Software который подробно рассказал о нем в своем ролике [13]. Укажем начальный индекс **selected\_index** (рис. 9):

    rail = ft.NavigationRail(

        selected\_index=0,

Укажем, что надписи будут отображаться для всех пунктов назначения с помощью **ft.NavigationRailLabelType.ALL**

        label\_type=ft.NavigationRailLabelType.ALL,

Зададим минимальную ширину панели и создадим кнопку тест которая всегда будет сверху и будет выполнять запуск neuro с параметрами (1,1):

        min\_width=100,

        leading=ft.FloatingActionButton(icon=ft.icons.PLAY\_ARROW, text='Тест', on\_click=fab\_pressed),

Установим выравнивание группы пунктов **group\_alignment**, укажем то, что панель не должна расширятся **extended** и установим функцию **on\_change**, которая вызывается при изменении пункта назначения:

        group\_alignment=–0.9,

extended=False,

        on\_change=set\_screen,

И последним шагом создадим список пунктов назначения. Каждый пункт представлен объектом **ft.NavigationRailDestination**, который содержит иконку, надпись и контент для отображения. В данном случае созданы четыре пункта: "Главная", "Лог", "Теория" и "О программе". Каждый пункт имеет свою иконку и надпись. С помощью **label\_content** указывается название пункта назначения, с помощью label указывается название, которое будет отображаться в интерфейсе. С помощью **icon** можно задать иконку пункта назначения использовав библиотеку иконок Flet [3]:

        destinations=[

            ft.NavigationRailDestination(

                icon=ft.icons.DOMAIN\_VERIFICATION\_SHARP,

                label\_content=ft.Text('Главная'),

                label='Главная'

            ),

            ft.NavigationRailDestination(

                icon=ft.icons.PLAYLIST\_ADD\_CHECK\_CIRCLE\_ROUNDED,

                selected\_icon=ft.icons.PLAYLIST\_ADD\_CHECK\_CIRCLE\_OUTLINED,

                label\_content=ft.Text('Лог'),

                label='Лог'

            ),

            ft.NavigationRailDestination(

                icon=ft.icons.QUESTION\_ANSWER,

                selected\_icon=ft.icons.QUESTION\_ANSWER\_OUTLINED,

                label\_content=ft.Text('Теория'),

                label='Теория'

            ),

            ft.NavigationRailDestination(

                icon=ft.icons.QUESTION\_MARK,),],)

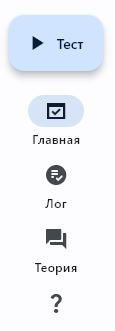


Рисунок 9. Создание навигационной панели

**2.2.3 Функция для смены цветовой темы приложения**

Визуальная составляющая является неотъемлемой частью приложения. Некоторые из пользователей предпочитают использовать приложение только темную тему (рис. 10). Для того чтобы сменить цветовую тему приложения нужно написать функцию **theme\_changed:**

def theme\_changed(e):

        page. theme\_changed theme\_mode = (ft.ThemeMode.DARK if page.theme\_mode == ft.ThemeMode.LIGHT else ft.ThemeMode.LIGHT)

        c.label = 'Светлая тема' if page.theme\_mode == ft.ThemeMode.LIGHT else 'Тёмная тема'

        page.update()

Теперь для удобства в переменную **с** поместим элемент управления **CupertinoSwitch**:

    c = ft.CupertinoSwitch(label='Светлая тема', on\_change=theme\_changed)

****

Рисунок 10. Функция для смены цветовой темы приложения

Установим стандартную тему приложения в светлый режим:

page.theme\_mode = ft.ThemeMode.LIGHT

### **2.2.3 Создание диалоговых окон и визуальной системы оповещения**

Сначала импортируем модуль threading, в дальнейшем он пригодится для обновления с полем оповещения в программе:

import threading

C помощью модуля **threading** можно использовать многопоточность в Python. Многопоточность позволяет выполнять несколько задач одновременно.

Так же импортируем модуль **subprocess**, в дальнейшем он пригодится для создания кнопки запуска диспетчера задач непосредственно из программы:

import subprocess

def open\_task\_manager():

    subprocess.Popen(['taskmgr'])

Создадим заготовку для кнопки открытия папки пользователя,откуда можно будет перейти в папку где сохраняются результаты тестирования:

def open\_user\_folder():

        user\_folder\_path = os.path.expanduser("~")

        os.startfile(user\_folder\_path)

Создадим параметры, которые будут передавать функции neuro:

    def get\_test\_parameters(complexity):

        if complexity == "тестовая":

            return 1, 1

        elif complexity == "низкая":

            return 64, 8

        elif complexity == "средняя":

            return 256, 16

        elif complexity == "высокая":

            return 1024, 32

        elif complexity == "очень высокая":

            return 2048, 48

Создадим функцию для кнопки “Тест”, созданной в навигационной панели:

    def fab\_pressed(e):

        complexity = "тестовая"

        start\_test(complexity)

Создадим функцию **test\_dialog** для диалоговых окон **ft.AlertDialog** ), в котором пользователю предлагается начать тестирование с определенной сложностью (**complexity**). Диалоговое окно содержит две кнопки действия: "Да" и "Нет". При нажатии на "Да" вызывается функция **handle\_yes**, а при нажатии на "Нет" вызывается функция **handle\_no**. Функция **handle\_yes** вызывает функцию **start\_test** для начала тестирования с указанной сложностью, а затем вызывает функцию **close\_dialog\_func**, чтобы закрыть диалоговое окно (рис. 11):

def test\_dialog(e, complexity):

        def close\_dialog():

            dialog.open = False

            page.update()

        def handle\_no(e):

            close\_dialog()

        dialog = ft.AlertDialog(

            modal=True,

            content=ft.Text(f'Вы хотите начать тест на сложности ({complexity})?'),

            actions=[

                ft.TextButton('Да', on\_click=lambda e: handle\_yes(complexity, close\_dialog)),

                ft.TextButton('Нет', on\_click=handle\_no),

            ],

            actions\_alignment=ft.MainAxisAlignment.END,

        )

        page.dialog = dialog

        dialog.open = True

        page.update()

    def handle\_yes(complexity, close\_dialog\_func):

        start\_test(complexity)

        close\_dialog\_func()

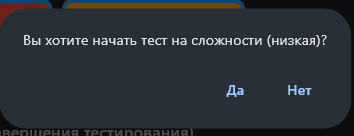


Рисунок 11. Диалоговое окно подтверждения

Создадим визуальную систему оповещения о прогрессе тестирования. Для этого используем внутреннюю функцию **inner\_start\_test()**, которая будет запускаться в отдельном потоке:

def start\_test(complexity):

Cначала визуализируем прогресс тестирования, отображая текст “Идет тестирование на сложности (“Сложность на которой проходит тестирование”)” и прогресс–бар (рис. 12).

        def inner\_start\_test():

            page.splash = ft.Row(

                [ft.Text(f'Идет тестирование на сложности ({complexity})', size=20),

                 ft.ProgressBar(width=400, bar\_height=40)],

                alignment=ft.MainAxisAlignment.CENTER, expand=True)

            page.update()

            neuro(\*get\_test\_parameters(complexity))

            page.splash = None

            page.update()



Рисунок 12. Визуализация прогресса тестирования

После завершения тестирования прогресс–бар скрывается, и выводится сообщение о завершении тестирования (рис. 13):

page.splash = ft.Row([ft.Text(' '),

                                  ft.Text(

                                      f'Тестирование на сложности ({complexity}) завершено, обновите вкладку <Главная>'чтобы увидеть результат', color='RED', size=20)],

                                 alignment=ft.MainAxisAlignment.CENTER, expand=True)

            page.update()



Рисунок 13. Сообщение о завершении тестирования

Все эти операции выполняются в отдельном потоке, чтобы не блокировать основной поток выполнения и обеспечить отзывчивость интерфейса. Для этого используется модуль **threading**, и функция **inner\_start\_test** запускается в новом потоке с помощью **threading.Thread(target=inner\_start\_test).start()**:

        threading.Thread(target=inner\_start\_test).start()

### **2.2.4 Создание текстовых редакторов для отображения информации**

Для отображения данных из создаваемых программой текстовых файлов нужно создать классы текстового редактора, которые открывают и выводят данные. Единственный недостаток данного метода вывода информации – это невозможность пользователю видеть информацию в реальном времени, но зато это позволяет пользователю копировать или редактировать лог и результат. То, как реализовать текстовый редактор подробно рассказал в своем ролике пользователь Indently подробно рассказал о нем в своем ролике [14], у него была взята основа и модифицирована под нужды моей программы.

Сначала создадим класс **TextEditorBase** который будет базовым классом для редакторов текста. Он наследует от **ft.UserControl** из встроенного класса в Flet Framework. Конструктор класса **\_\_init\_\_** в **TextEditorBase** инициализирует путь к файлу **file\_path** и создает текстовое поле **textfield**, которое представляет собой многострочное текстовое поле для редактирования текста, некоторые из его параметров настраиваются в зависимости от условия, в частности, количество минимальных строк **min\_lines**:

class TextEditorBase(ft.UserControl):

    def \_\_init\_\_(self, file\_path: str) –> None:

        super().\_\_init\_\_()

        self.file\_path = file\_path

        self.textfield = ft.TextField(

            adaptive=True,

            multiline=True,

            autofocus=True,

            border=ft.colors.RED,

            min\_lines=400 if 'stats' in file\_path else 40,

            on\_change=self.save\_text,

            content\_padding=30,

            cursor\_color='blue'

        )

Создадим методы сохранения и чтения текста из файла:

    def save\_text(self, e: ft.ControlEvent) –> None:

        with open(self.file\_path, 'w', encoding='utf–8') as f:

            f.write(self.textfield.value)

    def read\_text(self) –> str:

        try:

            with open(self.file\_path, 'r', encoding='utf–8') as f:

                return f.read()

        except FileNotFoundError:

            self.textfield.hint\_text = 'Результаты тестирования(Появятся после завершения тестирования)' \

                if 'stats' in self.file\_path else 'Тут отобразится лог'

            return ''

Создадим метод для построения пользовательского интерфейса. Здесь текстовому полю **textfield** устанавливается значение, прочитанное из файла, и оно возвращается:

    def build(self) –> ft.TextField:

        self.textfield.value = self.read\_text()

        return self.textfield

**TextEditor1** и **TextEditor** являются наследниками класса **TextEditorBase**. В программе они определяют файлы для сохранения и чтения текста **stats.txt** и **log.txt**:

class TextEditor1(TextEditorBase):

    def \_\_init\_\_(self) –> None:

        super().\_\_init\_\_('stats.txt')

class TextEditor(TextEditorBase):

    def \_\_init\_\_(self) –> None:

        super().\_\_init\_\_('log.txt')

И посмотрим, как выглядит данный метод вывода (рис. 14):



Рисунок 14. Текстовый редактор отображающий log.txt

### **2.2.5 Заполнение вкладок программы**

Создадим структуру **screen\_list** в которой будет находится содержание всех разделов. Код, использованный в структуре **screen\_list**, находится в **Приложении 3** так как он слишком объемен.

1. **Главная** (рис. 15):

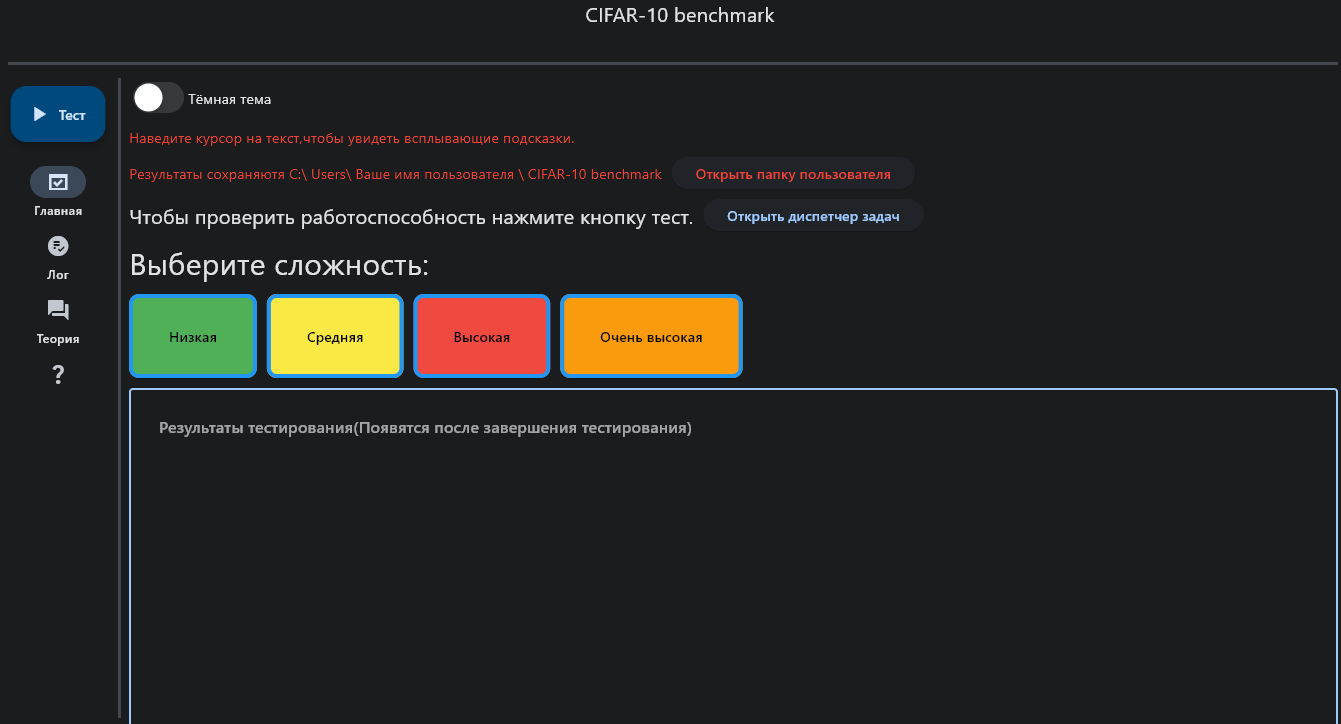


Рисунок 15. Главная

* + Есть возможность навести курсор на текст, чтобы увидеть всплывающие подсказки (рис. 16).

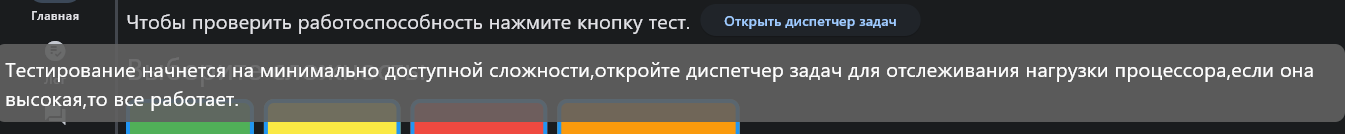


Рисунок 16. Всплывающие подсказки

* + Пользователю предоставляется информация о том, где сохраняются результаты тестирования и возможность быстро перейти в нужный каталог (рис. 17).



Рисунок 17. Сообщение и кнопка перехода в каталог

* + Кнопка открытия диспетчера задач (рис. 18).



Рисунок 16. Сообщение и кнопка запуска диспетчера задач

* + Пользователь может выбрать сложность тестирования (рис. 18).

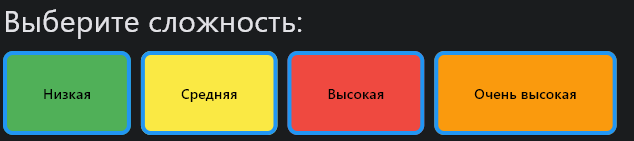


Рисунок 18. Выбор сложности

* + Текстовый редактор с результатами тестирования (рис. 19).

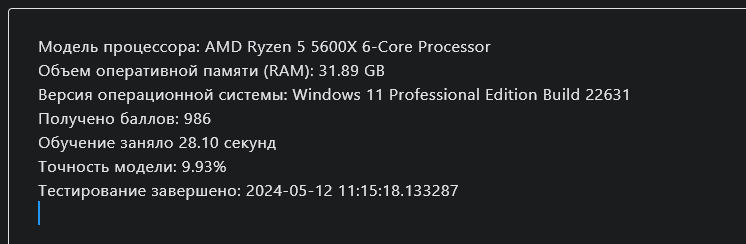


Рисунок 19. Вывод результатов

1. **Лог**:

* Текстовый редактор (рис. 20):



Рисунок 20. Вывод лога

1. **Теория** (рис. 21):

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 21. Теоретическая справка

1. **"О программе"**(рис. 22):

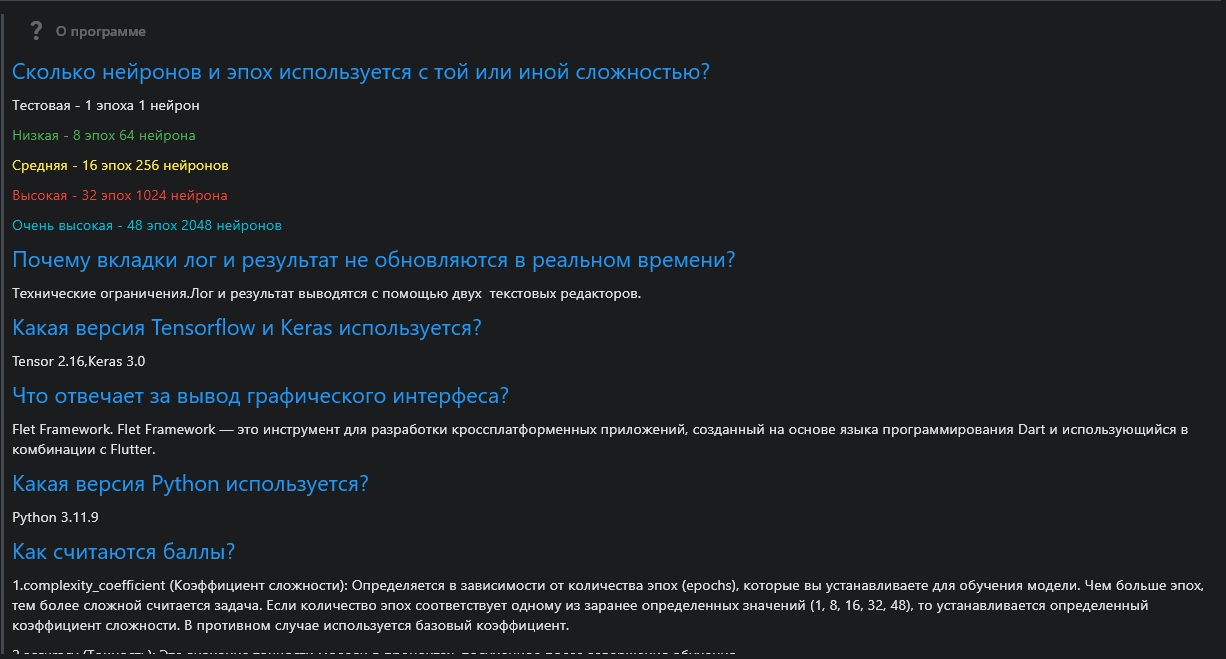


Рисунок 22. О программе

### **2.2.6 Сборка настольного приложения**

Программа Flet Python и все его зависимости могут быть собраны в исполняемый файл, и пользователь может запустить его на своем компьютере без установки интерпретатора Python или каких–либо модулей.

Flet использует Flutter SDK для упаковки программы Flet Python для создания быстрого, автономного, полностью настраиваемого исполняемого файла для Windows, Linux и macOS с Python runtime, встроенного в исполняемый файл и работающего в процессе.

Сначала заполняем файл с зависимостями с помощью команды:

pip freeze requirements.txt

Полный список зависимостей для моей программы вы можете найти в **Приложении 5**.

Затем выполняем сборку с помощью команды:

flet build windows

И получаем готовую программу (рис. 23):

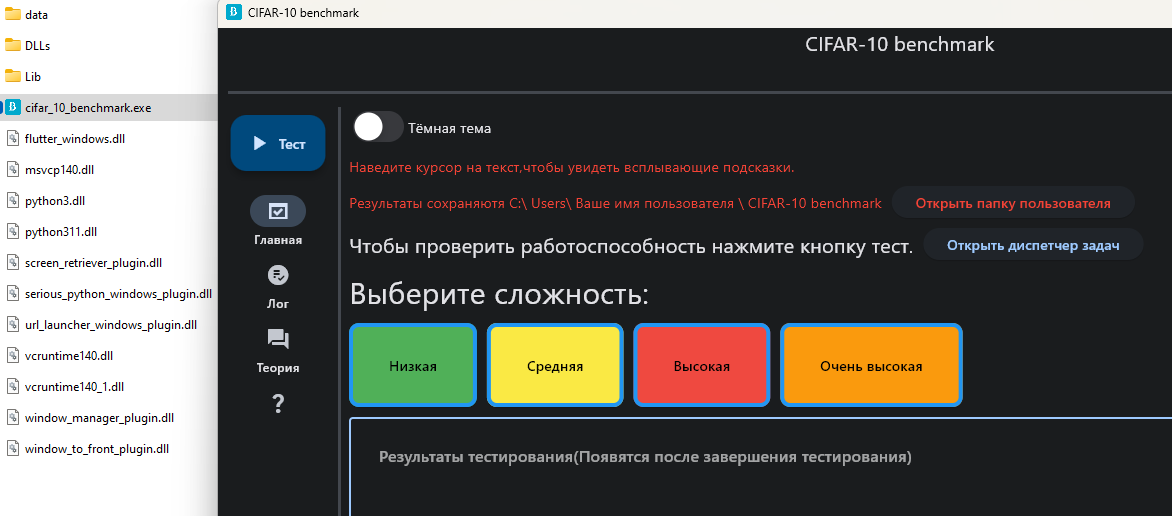


Рисунок 23. Полученная программа

### **2.2.7 Создание установщика для программы**

Даже после сборки остается 2 зависимости:

1. Нужно поместить набор данных в папку пользователя, чтобы сделать программу независимой от наличия интернет–соединения, но если по какой–то причине набор не будет помещен в папку пользователя, то модуль tdfs загрузит необходимый набор;
2. Программе нужен VC\_redist.x64.

VC\_redist.x64 − это пакет компонентов Microsoft Visual C++ Redistributable для 64–разрядных операционных систем Windows. Он содержит необходимые библиотеки для запуска приложений, разработанных с использованием Microsoft Visual C++ в среде выполнения на 64–битных версиях Windows.

Этот пакет устанавливает ряд компонентов, таких как библиотеки DLL (динамически подключаемые библиотеки), необходимые для выполнения программ, которые были скомпилированы с использованием соответствующих версий Microsoft Visual C++.

Для создания установщика будет использован Inno Setup. Код использованный в Inno Setup, находится в **Приложении 4**. **Inno Setup** − это бесплатный инструмент для создания программ установки для Windows. создавать установочные программы для своих приложений с помощью удобного интерфейса (рис. 24). Изучил его с помощью статьи пользователя Anton Titovets на сайте infostart[11].

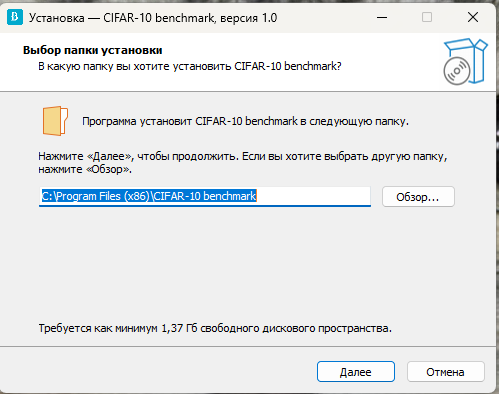


Рисунок 24. Установщик

## 2.3 Выводы по главе 2

Flet предоставляет удобные инструменты для создания пользовательского интерфейса на основе Flutter. Обладает широким набором виджетов и компонентов, что облегчает разработку интерактивных программ. Поддерживает красивый дизайн и анимации, что способствует созданию красивого пользовательского интерфейса.

Реализована навигационная панель, обеспечивающая удобную навигацию между различными разделами программы.

В программе реализовано создание главного раздела программы с использованием виджетов Flet, обеспечивающих необходимый интерфейс для работы с программой, использован текстовый редактор для отображения информации, что позволяет пользователям комфортно просматривать полученные в ходе обучения данные. Так же у пользователя есть возможность изучить лог обучения модели перейдя в соответствующий раздел. Так же реализован раздел с теоретической справкой на тему создания нейронной сети.

Использованы диалоговые окна и визуальные системы оповещения для информирования пользователя о различных событиях и действиях в программе.

Создан установщик для программы с помощью Inno Setup, обеспечивающий удобную установку программы на компьютеры пользователей.

**3. Тестирование зависимостей и различных ПК с помощью программы**

Целью тестирования является проверка работоспособности программы на различных конфигурациях компьютеров и версиях операционных систем, а также выявление возможных проблем с зависимостями и установка минимальных системных требований.

Нужно убедиться, что программа работает эффективно на различных конфигурациях ПК. Оценить производительность программы на разных компьютерах и в различных условиях нагрузки, чтобы убедиться, что она эффективно использует ресурсы и обеспечивает хорошую производительность.

Программа была распространена среди моих родных и друзей для того, чтобы проверить как можно больше различных компьютеров, чтобы убедится, что процесс установки и использования прост и понятен, что программа не конфликтует с различными антивирусами или другим программным обеспечением.

## 3.1 Тестирование зависимостей

В этом разделе приведены результаты тестирования программы с использованием разного количества ядер, разной частоты ЦПУ и ОЗУ, с мульти поточностью и без, с разным количеством планок ОЗУ на примере AMD Ryzen 5 5600X. Будет оцениваться только время, затраченное на обучение.

Мульти поточность (МТ) — это способность программы или операционной системы выполнять несколько потоков выполнения (threads) параллельно. Поток представляет собой отдельный путь выполнения внутри процесса, который может быть запущен и завершен независимо от других потоков.

В таблицах используется сокращения, их вы можете найти в списке сокращений, который находится перед приложениями.

Характеристики тестового стенда:

* Процессор: AMD Ryzen 5 5600X
* Оперативная память: 32 Gb DDR4 3800 MHz,4 на 8 гигабайт
* Накопитель: SSD M.2 Nvme 500Gb
* Операционная система: Windows 11 23H2

### **3.1.1 Тестирование зависимости** **от частоты ядер, количества ядер и многопоточности**

Таблица 2.

Тестирование зависимостей 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Частота и мт | Сложность | | | |
| Низкая | Средняя | Высокая | Очень высокая |
| 6/12,  Сток | 216 | 451 | 1061 | 2073 |
| 6/6,  Сток | 252 | 517 | 1193 | 2162 |
| 6/12,  2000 МГц | 444 | 932 | 2026 | 3533 |
| 6/6,  2000 МГц | 510 | 1052 | 2370 | 3972 |
| 6/12,  3000 МГц | 298 | 609 | 1395 | 2642 |
| 6/6,  3000 МГц | 342 | 697 | 1599 | 2841 |
| 6/12,  4000 МГц | 228 | 465 | 1128 | 2108 |
| 6/6,  4000 МГц | 257 | 524 | 1219 | 2307 |
| 6/12,  ОС | 197 | 404 | 951 | 1824 |
| 2/2,  4000 МГц | 566 | 1152 | 2512 | 4196 |
| 2/4,  4000 МГц | 580 | 1189 | 2560 | 4321 |
| 4/4,  4000 МГц | 359 | 722 | 1645 | 2828 |
| 4/8,  4000 МГц | 321 | 669 | 1484 | 2697 |

По итогам тестирования можно понять, что каждый параметр сказывается на затраченном на обучение времени. Наибольшее влияние оказало отключение части ядер и мульти поточности. Программа помогла выявить нестабильность при разгоне благодаря высокой нагрузке на ЦПУ.

### **3.1.2 Тестирование зависимости от количества планок ОЗУ и частоты**

Таблица 3.

Тестирование зависимостей 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Частота и мт | Сложность | | | |
| Низкая | Средняя | Высокая | Очень высокая |
| 1/2666 | 266 | 546 | 1528 | 3307 |
| 2/2666 | 236 | 483 | 1208 | 2352 |
| 4/2666 | 233 | 483 | 1153 | 2226 |
| 1/3800 | 255 | 537 | 1492 | 3345 |
| 2/3800 | 234 | 479 | 1187 | 2415 |
| 4/3800 | 228 | 465 | 1128 | 2108 |

По итогам тестирования выявлено, что 8 гигабайтов ОЗУ вне зависимости от частоты недостаточно для обучения нейронной сети на очень высокой сложности. Программа хорошо реагирует как на увеличение частоты, так и на увеличение количества планок ОЗУ.

### **3.2.1 AMD Ryzen 5 5600X, 32 Gb DDR4 3800 MHz**

Процессор: AMD Ryzen 5 5600X

Оперативная память: 32 Gb DDR4 3800 MHz

Накопитель: SSD M.2 Nvme 500Gb

Операционная система: Windows 11 23H2

Таблица 4.

Тестирование AMD Ryzen 5 5600X

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 216 | 7718 | 68.42 |
| Средняя | 451 | 10538 | 80.33 |
| Высокая | 1061 | 12090 | 85.73 |
| Очень высокая | 2073 | 9282 | 87.58 |

**AMD Ryzen 5 5600X** – это процессор среднего уровня, который был представлен в 2020 году. Он является частью линейки Vermin и использует архитектуру Zen 3, работает на сокете AM4. Имеет 6 ядер и 12 потоков, базовая частота составляет 3,7 ГГц, а максимальная частота в режиме Turbo – до 4,6 ГГц. TDP процессора составляет 65 Вт.

### **3.2.2 AMD Ryzen 5 3600, 16 Gb DDR4 3200 MHz**

Процессор: AMD Ryzen 5 3600

Оперативная память: 16 Gb DDR4 3200 MHz

Накопитель: SSD M.2 Nvme 250Gb

Операционная система: Windows 11 23H2

Таблица 5.

Тестирование AMD Ryzen 5 3600

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 228 | 7923 | 70.51 |
| Средняя | 478 | 10350 | 79.58 |
| Высокая | 1187 | 11487 | 85.72 |
| Очень высокая | 2377 | 7407 | 87.26 |

**AMD Ryzen 5 3600** – это процессор среднего уровня, который был представлен в 2019 году. Он является частью линейки Matisse и использует архитектуру Zen 2,работает на сокете AM4. Имеет 6 ядер и 12 потоков, базовая частота составляет 3,6 ГГц, а максимальная частота в режиме Turbo – до 4.2 ГГц. TDP процессора составляет 65 Вт.

В моём случае частота процессора была зафиксирована на 4,4 ГГц по всем ядрам при напряжении 1,26 для лучшей стабильности во время тестирования. При стандартных настройках напряжение сильно завышено.

### **3.2.3 AMD FX 6300, 16 Gb DDR3 2000 MHz**

Процессор: AMD FX 6300

Оперативная память: 16 Gb DDR3 2000 MHz

Накопитель: SSD 120Gb

Операционная система: Windows 10 22H2

Таблица 6.

Тестирование AMD FX 6300

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 2863 | 1731 | 70.54 |
| Средняя | 5827 | 1204 | 80.28 |
| Высокая | 12256 | 1719 | 85.93 |
| Очень высокая | 20065 | 2202 | 88.10 |

**AMD FX 6300** – это процессор среднего уровня, который был представлен в 2012 году. Он является частью линейки Vishera и использует архитектуру Piledriver, работает на сокете AM3+. Имеет 6 ядер и 6 потоков, базовая частота составляет 3,5 ГГц, а максимальная частота в режиме Turbo – до 4,1 ГГц. TDP процессора составляет 95 Вт.

На операционной системе windows 7 приложение отказывалось запускаться, поэтому операционная система была обновлена до Windows 10 22H2.

### **3.2.4 AMD Ryzen 7 5800H, 32 Gb DDR4 3200 MHz**

Процессор: AMD Ryzen 7 5800H

Оперативная память: 32 Gb DDR4 3200 MHz

Накопитель: SSD M.2 Nvme 500Gb

Операционная система: Windows 11 23H2

Таблица 7.

Тестирование AMD Ryzen 7 5800H

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 241 | 7798 | 69.66 |
| Средняя | 509 | 10332 | 80.25 |
| Высокая | 1274 | 10965 | 84.88 |
| Очень высокая | 2454 | 6996 | 87.96 |

**AMD Ryzen 7 5800H** – это мобильный процессор высокого уровня, который был представлен в 2021 году. Он является частью линейки Cezanne и использует архитектуру Zen 3. Имеет 8 ядер и 16 потоков, базовая частота составляет 3,2 ГГц, а максимальная частота в режиме Turbo – до 4,4 ГГц. TDP процессора составляет 45 Вт.

### **3.2.5 AMD Ryzen 3 3200U, 12 Gb DDR4 2400 MHz**

Процессор: AMD Ryzen 3 3200U

Оперативная память: 12 Gb DDR4 2400 MHz

Накопитель: SSD M.2 Nvme 500Gb

Операционная система: Windows 11 23H2

Таблица 8.

Тестирование AMD Ryzen 3 3200U

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 1792 | 4264 | 70.78 |
| Средняя | 3663 | 1208 | 80.53 |
| Высокая | 7955 | 1701 | 85.07 |

AMD Ryzen 3 3200U – это мобильный процессор начального уровня, который был представлен в 2021 году. Он является частью линейки “Raven Ridge” и использует архитектуру Zen. Имеет 2 ядра и 4 потока, базовая частота составляет 2,6 ГГц, а максимальная частота в режиме Turbo – до 3,5 ГГц. TDP процессора составляет 15 Вт.

На высокой сложности была выявлена нехватка оперативной памяти, активно использовался файл подкачки. Скорее всего это связано с тем, что для встроенной графики резервируется 2 гигабайта памяти.

### **3.2.6 AMD Athlon Gold 3150U, 16 Gb DDR4 2133 MHz**

Процессор: AMD Athlon Gold 3150U

Оперативная память: 16 Gb DDR4 2133 MHz

Накопитель: SSD M.2 Nvme 500Gb

Операционная система: Windows 11 23H2

Таблица 9.

Тестирование AMD Athlon Gold 3150U

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 1758 | 4309 | 70.21 |
| Средняя | 3617 | 1206 | 80.39 |
| Высокая | 7826 | 1711 | 85.53 |

AMD Athlon Gold 3150U – это мобильный процессор начального уровня, который был представлен в 2020 году. Он является частью линейки Dali и использует архитектуру Zen. Имеет 2 ядра и 4 потока, базовая частота составляет 2,6 ГГц, а максимальная частота в режиме Turbo – до 3,3 ГГц. TDP процессора составляет 15 Вт.

### **3.2.7 Intel Core i3 3120m, 6Gb DDR3 1600 MHz**

Процессор: Intel Core i3 3120m

Оперативная память: 6Gb DDR3 1600 MHz

Накопитель: SSD M.2 Nvme 500Gb

Операционная система: Windows 10 22H2

Таблица 10.

Тестирование Intel Core i3 3120m

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 4680 | 843 | 70.22 |
| Средняя | 9481 | 1210 | 80.64 |
| Высокая | 21598 | 1711 | 85.55 |

**Intel Core i3 3120M** – это мобильный процессор начального уровня, который был представлен в 2020 году. Он использует архитектуру Ivy Bridge. Имеет 2 ядра и 4 потока, базовая частота составляет 2,5 ГГц, а режим Turbo отсутствует. TDP процессора составляет 35 Вт.

Уже на средней сложности была выявлена нехватка оперативной памяти, активно использовался файл подкачки. Чтобы пройти ноутбук прошел тест на высокой сложности 2 гигабайтная плашка ОЗУ была заменена на 4 гигабайтную.

### **3.2.8 AMD Ryzen 5 5500, 16 Gb DDR4 3200 MHz**

Процессор: AMD Ryzen 5 5500

Оперативная память: 16 Gb DDR4 3200 MHz

Накопитель: SSD M.2 Nvme 500Gb

Операционная система: Windows 11 23H2

Таблица 11.

Тестирование AMD Ryzen 5 5500

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 245 | 7747 | 69.28 |
| Средняя | 478 | 10291 | 79.78 |
| Высокая | 1225 | 11368 | 86.17 |
| Очень высокая | 2332 | 7723 | 87.75 |

**AMD Ryzen 5 5500** – это процессор среднего уровня, который был представлен в 2022 году. Он является частью линейки Cezanne и использует архитектуру Zen 3, работает на сокете AM4. Имеет 6 ядер и 12 потоков, базовая частота составляет 3,6 ГГц, а максимальная частота в режиме Turbo – до 4,2 ГГц. TDP процессора составляет 65 Вт.

### **3.2.9 AMD Ryzen 7 5800X3D, 32 Gb DDR4 3600 MHz**

Процессор: AMD Ryzen 7 5800X3D

Оперативная память: 32 Gb DDR4 3600 MHz

Накопитель: SSD M.2 Nvme 1000Gb

Операционная система: Windows 11 23H2

Таблица 12.

Тестирование AMD Ryzen 7 5800X3D

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 136 | 8129 | 70.42 |
| Средняя | 279 | 11117 | 80.35 |
| Высокая | 646 | 14151 | 86.24 |
| Очень высокая | 1208 | 14736 | 88.73 |

**AMD Ryzen 7 5800X3D** – это процессор высокого уровня, который был представлен в 2022 году. Он является частью линейки Vermeer и использует архитектуру Zen 3, работает на сокете AM4. Имеет 8 ядер и 16 потоков, базовая частота составляет 3,4 ГГц, а максимальная частота в режиме Turbo – до 4,5 ГГц. TDP процессора составляет 105 Вт.

### **3.2.10 AMD Ryzen AMD Ryzen 7 7800X3D, 32 Gb DDR5 6000 MHz**

Процессор: AMD Ryzen AMD Ryzen 7 7800X3D

Оперативная память: 32 Gb DDR5 6000 MHz

Накопитель: SSD M.2 Nvme 1000Gb

Операционная система: Windows 11 23H2

Таблица 13.

Тестирование AMD Ryzen AMD Ryzen 7 7800X3D

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 113 | 8126 | 69.93 |
| Средняя | 234 | 11255 | 80.26 |
| Высокая | 580 | 14185 | 84.56 |
| Очень высокая | 1060 | 15417 | 87.42 |

**AMD Ryzen 7 7800X3D** – это процессор высокого уровня, который был представлен в 2023 году. Он является частью линейки Raphael и использует архитектуру Zen 4, работает на сокете AM5. Имеет 8 ядер и 16 потоков, базовая частота составляет 4,2 ГГц, а максимальная частота в режиме Turbo – до 5,0 ГГц. TDP процессора составляет 120 Вт.

### **3.2.11 Intel Core i3 10105F, 16 Gb DDR4 3200 MHz**

Процессор: Intel Core i3 10105F

Оперативная память: 16 Gb DDR4 3200 MHz

Накопитель: SSD M.2 Nvme 256Gb

Операционная система: Windows 10 22H2

Таблица 14.

Тестирование Intel Core i3 10105F

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 341 | 7573 | 69.72 |
| Средняя | 710 | 9577 | 79.54 |
| Высокая | 1664 | 9127 | 84.89 |
| Очень высокая | 2970 | 3851 | 88.05 |

**Intel Core i3 10105F** – это процессор начального уровня, который был представлен в 2021 году. Он является частью линейки Comet Lake и использует архитектуру Comet Lake–S, работает на сокете LGA1200. Имеет 4 ядра и 8 потоков, базовая частота составляет 3,7 ГГц, а максимальная частота в режиме Turbo – до 4,4 ГГц. TDP процессора составляет 65 Вт.

### **3.2.12 Intel Core i5 10600K, 16 Gb DDR4 3600 MHz**

Процессор: Intel Core i5 10600K

Оперативная память: 16 Gb DDR4 3600 MHz

Накопитель: SSD M.2 Nvme 512Gb

Операционная система: Windows 10 22H2

Таблица 15.

Тестирование Intel Core i5 10600K

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 235 | 7963 | 71.01 |
| Средняя | 489 | 10396 | 80.21 |
| Высокая | 1218 | 11410 | 86.25 |
| Очень высокая | 2325 | 7803 | 88.13 |

**Intel Core i5 10600K** – это процессор среднего уровня, который был представлен в 2020 году. Он является частью линейки Comet Lake и использует архитектуру Comet Lake–S, работает на сокете LGA1200. Имеет 6 ядер и 12 потоков, базовая частота составляет 4,1 ГГц, а максимальная частота в режиме Turbo – до 4,8 ГГц. TDP процессора составляет 125 Вт.

### **3.2.13 Intel Xeon CPU E5 2696 v3, 32 Gb DDR3 2133 MHz**

Процессор: Intel Xeon CPU E5–2696 v3

Оперативная память: 32 Gb DDR3 2133 MHz

Накопитель: SSD M.2 Nvme 512Gb

Операционная система: Windows 11 23H2

Таблица 16.

Тестирование Intel Xeon CPU E5 2696 v3

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 190 | 8031 | 70.66 |
| Средняя | 342 | 10861 | 80.02 |
| Высокая | 777 | 13394 | 85.43 |
| Очень высокая | 1420 | 13328 | 88.07 |

**Intel Xeon CPU E5 2696 v3** – это серверный процессор, который был представлен в 2014 году. Он является частью линейки Haswell и использует архитектуру Haswell–EP, работает на сокете LGA2011–3. Имеет 18 ядер и 36 потоков, базовая частота составляет 2,3 ГГц, а максимальная частота в режиме Turbo – до 3,8 ГГц. TDP процессора составляет 145 Вт.

### **3.2.14 Intel Core i7 14700K, 64 Gb DDR5 6400 MHz**

Процессор: Intel Core i7 14700K

Оперативная память: 64 Gb DDR5 6400 MHz

Накопитель: SSD M.2 Nvme 1024Gb

Операционная система: Windows 11 23H2

Таблица 17.

Тестирование Intel Core i7 14700K

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность | Время  (в секундах) | Баллы | Точность  (в процентах) |
| Низкая | 87 | 8226 | 70.25 |
| Средняя | 163 | 11462 | 80.04 |
| Высокая | 342 | 15615 | 86.27 |
| Очень высокая | 724 | 17427 | 87.28 |

**Intel Core i7 14700K** – это процессор высокого уровня, который был представлен в 2023 году. Он является частью линейки Raptor Lake–S и использует архитектуры Raptor Cove для производительных ядер и Gracemont для энергоэффективных ядер, работает на сокете LGA1700. Имеет 8 производительных ядер,12 энергоэффективных ядер и 12 потоков. Для производительных ядер базовая частота составляет 3,4 ГГц, а максимальная частота в режиме Turbo – до 5,6 ГГц. Для энергоэффективных ядер базовая частота составляет 2,5 ГГц, а максимальная частота в режиме Turbo – до 4,3 ГГц. TDP процессора составляет 125 Вт.

## 3.3 Выводы по итогам тестирования

По итогам тестирования можно прийти к выводу, что оптимальным решением для обучения нейронной сети является использование конфигурации с среднебюджетным 6 ядерным процессором 2019 года и 16 гигабайтами оперативной памяти, желательно наличие в системе твердотельного накопителя.

Обучить нейронную сеть можно и на двухъядерном современном процессоре, но тогда желательно иметь 8 гигабайтов оперативной памяти.

Так же было выявлено, что TensorFlow 2 требует от процессора наличие с поддержкой SSE4.1, AVX инструкций. Так же в системе должен быть установлен компонент Visual C++ для Visual Studio 2015, 2017, 2019 и 2022. На системах с менее чем 6 гигабайтами оперативной памяти приложение запускается, но работает гораздо медленнее и может прекратить работу. Приложение не работает на операционной системе Windows 7 и старее.

По итогам тестирования можно прийти к выводам, что минимальные системные требования для запуска приложения:

1. Процессор 2 ядра x64 с наличием с поддержкой AVX инструкций.
2. Оперативная память 6 гигабайт
3. Операционная система Windows 10.

# **Заключение**

В ходе работы была разработана программа на основе Flet Framework, позволяющая проверить производительность нейронных сетей на персональном компьютере без решения зависимостей. Программа обеспечивает пользовательский интерфейс для удобного взаимодействия с пользователем, реализует алгоритмы проверки производительности нейронных сетей на примере CIFAR–10 и интегрирует инструменты оценки производительности и метрик качества.

Эта работа имеет большое практическое значение, так как она упрощает процесс проверки производительности нейронных сетей для пользователей, позволяя им быстро и эффективно оценить производительность на своем компьютере. Кроме того, программа делает этот процесс более доступным, так как не требует установки и настройки сложных зависимостей. На момент завершения написания работы программа не имеет аналогов.

Выводы по результатам работы:

1. Разработанная программа успешно реализует поставленную цель и задачи, обеспечивая удобный и эффективный инструмент для проверки производительности нейронных сетей на примере CIFAR–10.
2. Программа может быть использована как специалистами в области машинного обучения и исследований, так и обычными пользователями, желающими оценить производительность своего компьютера для запуска нейронных сетей и это может сподвигнуть их начать изучать нейронные сети.

Дальнейшее развитие может включать расширение функциональности программы, портирование на отечественные операционные системы, добавление новых алгоритмов проверки производительности.

# **Список литературы**

1. Bishop C. M. Pattern Recognition and Machine Learning [Электронный ресурс]. Черновики главы можно найти на официальной странице автора. URL: https://users.aston.ac.uk/~mansd/socprml/ (дата обращения: 10.03.2024).
2. Chollet F. Deep Learning with Python [Электронный ресурс]. Первая глава книги доступна бесплатно на GitHub. URL: https://github.com/fchollet/deep–learning–with–python–notebooks (дата обращения: 10.03.2024).
3. Flet. Иконки. Материал из галереи Flet. URL: https://gallery.flet.dev/icons–browser/ (дата обращения: 10.03.2024).
4. Flet. Материал из документации – Flet Framework для Python. URL: https://flet.dev/docs/ (дата обращения: 10.03.2024).
5. Goodfellow I., Bengio Y., Courville A. Deep Learning [Электронный ресурс]. Доступно бесплатно на сайте Университета Монреаля. URL: http://www.deeplearningbook.org/ (дата обращения: 10.03.2024).
6. Krizhevsky A. Learning Multiple Layers of Features from Tiny Images [Электронный ресурс]. Доступно бесплатно на официальном сайте автора. URL: https://www.cs.toronto.edu/~kriz/cifar.html (дата обращения: 10.03.2024).
7. Nielsen M. Neural Networks and Deep Learning [Электронный ресурс]. Доступно бесплатно на официальном сайте автора. URL: http://neuralnetworksanddeeplearning.com/ (дата обращения: 10.03.2024).
8. Solter21 Flet Framework: документация. URL: https://dasolter21.github.io/flet–docs/ (дата обращения: 10.03.2024).
9. TensorFlow. Материал из документации – TensorFlow для Python. URL: https://www.tensorflow.org/learn?hl=ru (дата обращения: 10.03.2024).
10. TensorFlow. Руководство по Keras [Электронный ресурс]. URL: <https://www.tensorflow.org/guide/keras?hl=ru> (дата обращения: 07.06.2024).
11. Titovets A. Создание установщика из любых файлов в Inno Setup [Электронный ресурс]. URL: https://infostart.ru/1c/articles/2115499/ (дата обращения: 10.03.2024).
12. Visual Studio Code [Электронный ресурс]. URL: <https://code.visualstudio.com/> (дата обращения: 10.03.2024).
13. [YouTube] Gece Software Flet: Navigation Rail {Yan Menü} | Build Flutter Apps in Python | Python ile Flutter Uygulaması Yap [Электронный ресурс]. URL: https://www.youtube.com/watch?v=Xta2geE\_VhM (дата обращения: 10.03.2024).
14. [YouTube] Indently How To Create A Cool Notepad App In Python (Flet Tutorial) [Электронный ресурс]. URL: https://www.youtube.com/watch?v=2o2Gu–QO0b8 (дата обращения: 10.03.2024).

# **СПИСОК СОКРАЩЕНИЙ**

6/12 – где 6 количество ядер, а 12 количество потоков;

МГц – мегагерц;

ГГц – гигагерц;

Сток – стандартные частоты;

OC – разгон;

1/2666 – где 1 это количество планок ОЗУ, а 2666 частота

# **Приложение 1**

Приложение 1.

def create\_model(input\_shape):

model = Sequential()

model.add(Input(shape=input\_shape))

model.add(Conv2D(filters=32, kernel\_size=(3, 3), padding="same"))

model.add(Activation("relu"))

model.add(Conv2D(filters=32, kernel\_size=(3, 3), padding="same"))

model.add(Activation("relu"))

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

model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel\_size=(3, 3), padding="same"))

model.add(Activation("relu"))

model.add(Conv2D(filters=64, kernel\_size=(3, 3), padding="same"))

model.add(Activation("relu"))

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

model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel\_size=(3, 3), padding="same"))

model.add(Activation("relu"))

model.add(Conv2D(filters=128, kernel\_size=(3, 3), padding="same"))

model.add(Activation("relu"))

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

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(A))

model.add(Activation("relu"))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation="softmax"))

model.summary()

model.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

return model

# **Приложение 2**

Приложение 2.

class CustomCallback(Callback):

def on\_train\_batch\_end(self, batch, logs=None):

current\_time = time.time()

elapsed\_time = current\_time – self.model.start\_time

steps\_per\_second = batch / elapsed\_time if elapsed\_time > 0 else 0

with open('log.txt', 'a', encoding='utf–8') as f:

f.write(f"\nШаг {batch}/{self.params['steps']} обучения: "

f"время: {elapsed\_time:.2f} сек, "

f"точность: {logs['accuracy']:.2f}, "

f"шагов/сек: {steps\_per\_second:.2f}\n")

remaining\_steps = self.params['steps'] – batch

if remaining\_steps > 0 and steps\_per\_second > 0:

remaining\_time = remaining\_steps / steps\_per\_second

minutes, seconds = divmod(remaining\_time, 60)

with open('log.txt', 'a', encoding='utf–8') as f:

f.write(f"Осталось примерно: {int(minutes)} мин {int(seconds)} сек\n")

def on\_epoch\_end(self, epoch, logs=None):

current\_time = time.time()

elapsed\_time = current\_time – self.model.start\_time

steps\_per\_second = (epoch + 1) / elapsed\_time if elapsed\_time > 0 else 0

with open('log.txt', 'a', encoding='utf–8') as f:

f.write(f"\nЭпоха {epoch + 1}/{self.params['epochs']}: "

f"время: {elapsed\_time:.2f} сек, "

f"точность: {logs['accuracy']:.2f}, "

f"шагов/сек: {steps\_per\_second:.2f}\n")

remaining\_epochs = self.params['epochs'] – (epoch + 1)

remaining\_time = remaining\_epochs \* elapsed\_time / (epoch + 1) if (epoch + 1) > 0 else 0

minutes, seconds = divmod(remaining\_time, 60)

with open('log.txt', 'a', encoding='utf–8') as f:

f.write(f"Осталось примерно: {int(minutes)} мин {int(seconds)} сек\n")

# **Приложение 3**

Приложение 3.

screen\_list = [

ft.Row([ft.Column([c, ft.Text('Наведите курсор на текст,чтобы увидеть всплывающие подсказки', color='RED'),

ft.Row([ft.Text('Результаты сохраняютя C:\ Users\ Ваше имя пользователя \ CIFAR–10 benchmark', color='RED'),

ft.ElevatedButton("Открыть папку пользователя", on\_click=lambda e: open\_user\_folder(),color='RED')]),

ft.Row([ft.Tooltip(message="Тестирование начнется на минимально доступной сложности,откройте диспетчер задач для отслеживания нагрузки процессора,если она высокая,то все работает.Чтобы открыть диспетчер задач из программы запустите её c правами администратрора.",

content=ft.Text("Чтобы проверить работоспособность нажмите кнопку тест.", size=20),

padding=10,

border\_radius=10,

text\_style=ft.TextStyle(size=20, color=ft.colors.WHITE)),

ft.ElevatedButton("Открыть диспетчер задач", on\_click=lambda e: open\_task\_manager())]),

ft.Row([ft.Tooltip(message="Чтобы увидеть результаты тестирования нужно перейти во вкладку лог и вернуться на главную",content=ft.Text("Выберите сложность:", size=30),

padding=10,

border\_radius=10,

text\_style=ft.TextStyle(size=20, color=ft.colors.WHITE))]),

ft.Row([\*[ft.ElevatedButton(complexity.capitalize(),

style=ft.ButtonStyle(color={ft.MaterialState.HOVERED: ft.colors.WHITE,ft.MaterialState.FOCUSED: ft.colors.BLUE,

ft.MaterialState.DEFAULT: ft.colors.BLACK,},

shape={ft.MaterialState.HOVERED:ft.RoundedRectangleBorder(radius=20),ft.MaterialState.DEFAULT:ft.RoundedRectangleBorder(radius=10)},bgcolor={'': color},padding={ft.MaterialState.HOVERED: 40, ft.MaterialState.DEFAULT: 40},elevation={'pressed': 0, '': 1},

animation\_duration=1000,

side={ft.MaterialState.DEFAULT: ft.BorderSide(4,ft.colors.BLUE),ft.MaterialState.HOVERED: ft.BorderSide(4,ft.colors.BLUE),},),on\_click=open\_dialog(complexity))

for complexity, color in

[("низкая", ft.colors.GREEN),

("средняя", ft.colors.YELLOW),

("высокая", ft.colors.RED),

("очень высокая", ft.colors.ORANGE)]],]),

TextEditor1()],alignment=ft.MainAxisAlignment.START, expand=True)]),ft.Row([ft.Column([ft.Row([ft.Tooltip(

message="Чтобы увидеть актуально состояние лога нужно перейти во вкладку главная и вернуться на лог",

content=ft.Text("Лог", size=30),

padding=10,

border\_radius=10,

text\_style=ft.TextStyle(size=20, color=ft.colors.WHITE))]),TextEditor()], alignment=ft.MainAxisAlignment.START,expand=True, scroll=True)]),

ft.Row([ft.Column([ft.TextButton(

'Теоретическая справка',

icon='FACT\_CHECK', disabled=True),

ft.Text('Набор данных CIFAR–10', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('Набор данных CIFAR–10 (Canadian Institute for Advanced Research) — это набор изображений, который широко используется в машинном обучении и компьютерном зрении для обучения и тестирования алгоритмов классификации изображений. Набор состоит из 60000 цветных изображений размером 32x32 пикселя, разделенных на 10 классов: самолеты, автомобили, птицы, кошки, олени, собаки, лягушки, лошади, корабли и грузовики. Каждый класс содержит по 6000 изображений.'),

ft.Text('CIFAR–10 часто используется для демонстрации и оценки алгоритмов машинного обучения и глубокого обучения, таких как сверточные нейронные сети (CNN). В силу своей относительной небольшой размерности и разнообразия классов, CIFAR–10 хорошо подходит для исследований и экспериментов в области компьютерного зрения.'),

ft.Text('TensorFlow и Keras', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('Рассмотрим пример создания нейронной сети с использованием TensorFlow версии 2.16 и Keras версии 3.0, а для загрузки CIFAR–10 будет использован TDFS Nightly версии 4.9.4'),

ft.Text('TensorFlow — это открытая библиотека машинного обучения, разработанная компанией Google для создания и обучения нейронных сетей и других моделей машинного обучения. Она предоставляет широкий спектр инструментов и возможностей для работы с данными, построения моделей, их обучения, оценки и использования в продуктивной среде.'),

ft.Text('Keras — это высокоуровневый интерфейс для создания и обучения нейронных сетей, который работает поверх различных библиотек машинного обучения, таких как TensorFlow, Theano и Microsoft Cognitive Toolkit (CNTK). Он предоставляет простой и интуитивно понятный API для построения и обучения различных типов нейронных сетей, включая сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и их комбинации.'),

ft.Text('TensorFlow Datasets (TFDS) — это библиотека, предназначенная для загрузки и использования общедоступных наборов данных для обучения моделей машинного обучения. Nightly builds — это версии программного обеспечения, которые создаются автоматически каждую ночь из последних изменений в репозитории.'),

ft.Text('Создание модели нейронной сети', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('Для начала выбираем удобную вам среду разработки,я рекомендую использовать Visual Studio Code.'),

ft.Text('В первую очередь необходимо загрузить модули для этого в терминале вам следует вписать вот эти вот команды:'),

ft.Text('pip install tensorflow', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('pip install keras', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('pip install tdfs', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('Далее работаем с основным телом программы'),

ft.Text('Сначала импортируем все необходимое'),

ft.Text('import tensorflow as tf', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('import tensorflow\_datasets as tfds from tensorflow.keras', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('import Input from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Activation, Flatten', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('from keras.models import Sequential', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('import tensorflow as tf: Этот оператор импортирует библиотеку TensorFlow. Она будет использоваться для создания и обучения моделей машинного обучения.'),

ft.Text('import tensorflow\_datasets as tfds: Этот оператор импортирует библиотеку TensorFlow Datasets (TFDS), которая предоставляет доступ к множеству стандартных наборов данных для машинного обучения.'),

ft.Text('from tensorflow.keras import Input: Этот оператор импортирует класс Input из подмодуля keras библиотеки TensorFlow. Класс Input используется для определения входных данных модели.'),

ft.Text('from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Activation, Flatten: Этот оператор импортирует несколько типов слоев нейронных сетей из подмодуля layers библиотеки Keras, такие как сверточные слои Conv2D, слои пулинга MaxPooling2D, полносвязные слои Dense, слои регуляризации Dropout, активационные функции Activation и слой Flatten.'),

ft.Text('from keras.models import Sequential: Этот оператор импортирует класс Sequential из библиотеки Keras. Класс Sequential используется для создания последовательных моделей нейронных сетей, где слои добавляются последовательно друг за другом.'),

ft.Text('Conv2D, MaxPooling2D, Dense, Dropout, Activation и Flatten — это различные слои нейронных сетей, которые используются при создании моделей глубокого обучения с помощью библиотеки Keras. Вот краткое описание каждого из этих слоев:'),

ft.Text('1. Conv2D: Этот слой представляет собой сверточный слой, который применяет операцию свертки к входным данным. Он используется для обнаружения признаков в изображениях или других многомерных данных. Параметры этого слоя включают количество фильтров (или ядер), размер ядра свертки и способ заполнения.'),

ft.Text('2. MaxPooling2D: Этот слой представляет собой слой пулинга, который уменьшает размерность входных данных, выбирая максимальное значение из набора значений в каждой области. Он используется для уменьшения размера карт признаков и улучшения инвариантности к масштабированию и сдвигам.'),

ft.Text('3. Dense: Этот слой представляет собой полносвязный слой, который соединяет каждый нейрон во входном слое с каждым нейроном в выходном слое. Он применяет линейное преобразование к входным данным. Параметры этого слоя включают количество нейронов и функцию активации.'),

ft.Text('4. Dropout: Этот слой представляет собой слой регуляризации, который случайным образом отключает некоторые нейроны во время обучения для предотвращения переобучения. Он помогает повысить обобщающую способность модели.'),

ft.Text('5. Activation: Этот слой представляет собой слой активации, который применяет нелинейную функцию активации к выходу предыдущего слоя. Популярные функции активации включают ReLU (Rectified Linear Unit), sigmoid и tanh.'),

ft.Text('6. Flatten: Этот слой представляет собой слой сглаживания, который преобразует многомерные данные в одномерный вектор. Он используется для перехода от высокомерных представлений, таких как карты признаков изображений, к векторным представлениям, которые могут быть входом для полно связных слоев.'),

ft.Text('Для удобства поместим модель в функцию под названием neuro, которая будет принимать два аргумента A и B, где A это количество нейронов, а B это количество эпох.'),

ft.Text('Эпохи и нейроны — это два важных концепта в машинном обучении, особенно при обучении нейронных сетей.'),

ft.Text('1. Эпохи (Epochs):'),

ft.Text('– Эпоха представляет собой один проход через все обучающие данные во время обучения модели.'),

ft.Text('– При обучении нейронной сети данные разбиваются на батчи (мини–пакеты), и каждый батч используется для обновления параметров модели в процессе обучения.'),

ft.Text('– После завершения одной эпохи модель обычно оценивается на тестовом наборе данных, чтобы оценить ее производительность и обобщающую способность.'),

ft.Text('– Число эпох — это гиперпараметр, который задается заранее и обычно выбирается путем экспериментов. Увеличение числа эпох может улучшить производительность модели, но может также привести к переобучению.'),

ft.Text('2. Нейроны (Neurons):'),

ft.Text('– Нейрон представляет собой базовую единицу обработки в нейронной сети. Он принимает входные данные, вычисляет взвешенную сумму этих данных и применяет функцию активации к этой сумме.'),

ft.Text('– В скрытых слоях нейронной сети каждый нейрон получает входные данные от предыдущего слоя и передает свой выход следующему слою.'),

ft.Text('– Количество нейронов в слое — это гиперпараметр модели, который также задается заранее и обычно выбирается на основе сложности задачи и доступных ресурсов.'),

ft.Text('– Большее количество нейронов в слое может увеличить сложность модели и ее способность к выявлению сложных зависимостей в данных, но также может потребовать больше вычислительных ресурсов и увеличить риск переобучения.'),

ft.Text('Начнем с определения параметров:'),

ft.Text('def neuro(A, B):', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' batch\_size = 64', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' num\_classes = 10', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' epochs = B', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('– batch\_size: размер пакета данных для обучения модели.'),

ft.Text('– num\_classes: количество классов (в данном случае 10 классов CIFAR–10).'),

ft.Text('– epochs: количество эпох обучения, которое задается в аргументе B.'),

ft.Text('Дальше загрузим и преобразуем набор данных CIFAR–10:'),

ft.Text('def load\_data():', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' def preprocess\_image(image, label):', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' image = tf.image.convert\_image\_dtype(image, tf.float32)', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' return image, label', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' ds\_train, info = tfds.load("cifar10", with\_info=True, split="train", as\_supervised=True)', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' ds\_test = tfds.load("cifar10", split="test", as\_supervised=True) ', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' ds\_train = ds\_train.repeat().shuffle(A).map(preprocess\_image).batch(batch\_size)', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' ds\_test = ds\_test.repeat().shuffle(A).map(preprocess\_image).batch(batch\_size)', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' return ds\_train, ds\_test, info', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('– Эта функция загружает набор данных CIFAR–10 с использованием библиотеки TensorFlow Datasets (TFDS).'),

ft.Text('– Изображения преобразуются к типу float32 и масштабируются на диапазон [0, 1].'),

ft.Text('– Загружаются наборы данных для обучения и тестирования, каждый из которых перемешивается, повторяется и разбивается на пакеты размером batch\_size.'),

ft.Text('– Функция возвращает наборы данных для обучения и тестирования, а также информацию о наборе данных.'),

ft.Text('И наконец–то создадим модель нейронной сети с использованием Keras Sequential API:'),

ft.Text('def create\_model(input\_shape):', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model = Sequential()', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Input(shape=input\_shape))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Conv2D(filters=32, kernel\_size=(3, 3), padding="same"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Activation("relu"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Conv2D(filters=32, kernel\_size=(3, 3), padding="same"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Activation("relu"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(MaxPooling2D(pool\_size=(2, 2)))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Dropout(0.25))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' ', font\_family="Consolas"),

ft.Text(' model.add(Conv2D(filters=64, kernel\_size=(3, 3), padding="same"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Activation("relu"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Conv2D(filters=64, kernel\_size=(3, 3), padding="same"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Activation("relu"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(MaxPooling2D(pool\_size=(2, 2)))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Dropout(0.25))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' ', font\_family="Consolas"),

ft.Text(' model.add(Conv2D(filters=128, kernel\_size=(3, 3), padding="same"))', font\_family="Consolas"),

ft.Text(' model.add(Activation("relu"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Conv2D(filters=128, kernel\_size=(3, 3), padding="same"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Activation("relu"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(MaxPooling2D(pool\_size=(2, 2)))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Dropout(0.25))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' ', font\_family="Consolas"),

ft.Text(' model.add(Dropout(0.25))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Dense(A))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Activation("relu"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Dropout(0.5))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.add(Dense(num\_classes, activation="softmax"))', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.summary()', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' return model', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('Теперь разберем, что делает каждая из строк:'),

ft.Text('1. Создание объекта модели Sequential:'),

ft.Text('– model = Sequential(): Создается пустая модель нейронной сети типа Sequential. Sequential модель представляет собой линейный стек слоев.'),

ft.Text('2. Добавление входного слоя:'),

ft.Text('– model.add(Input(shape=input\_shape)): Добавляется входной слой нейронной сети с указанной формой входных данных (input\_shape).'),

ft.Text('3. Добавление сверточных слоев:'),

ft.Text('– model.add(Conv2D(filters=32, kernel\_size=(3, 3), padding="same")): Добавляется сверточный слой с 32 фильтрами размера (3, 3) и заполнением "same".'),

ft.Text('– model.add(Activation("relu")): Добавляется активационная функция ReLU после каждого сверточного слоя.'),

ft.Text('4. Добавление слоев пулинга и регуляризации:'),

ft.Text('– model.add(MaxPooling2D(pool\_size=(2, 2))): Добавляется слой максимального пулинга с размером пула (2, 2).'),

ft.Text('– model.add(Dropout(0.25)): Добавляется слой регуляризации Dropout с коэффициентом отсева 0.25.'),

ft.Text('5. Повторение шагов 3–4 для добавления нескольких сверточных слоев, слоев пулинга и регуляризации.'),

ft.Text('6. Преобразование многомерных данных в одномерный вектор:'),

ft.Text('– model.add(Flatten()): Добавляется слой Flatten, который преобразует выходные данные предыдущего слоя в одномерный вектор.'),

ft.Text('7. Добавление полносвязных слоев:'),

ft.Text('– model.add(Dense(A)): Добавляется полносвязный слой с A нейронами.'),

ft.Text('– model.add(Activation("relu")): Добавляется активационная функция ReLU после полносвязного слоя.'),

ft.Text('8. Добавление слоя вывода:'),

ft.Text('– model.add(Dense(num\_classes, activation="softmax")): Добавляется полносвязный слой с количеством нейронов, равным числу классов, и активационной функцией softmax для прогнозирования вероятностей принадлежности к каждому классу.'),

ft.Text('9. Компиляция модели:'),

ft.Text('– model.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"]): Модель компилируется с функцией потерь sparse\_categorical\_crossentropy, оптимизатором adam и метрикой accuracy.'),

ft.Text('10. Возврат модели'),

ft.Text('– return model: Функция возвращает созданную и скомпилированную модель нейронной сети.'),

ft.Text('И создаем основное тело программы:'),

ft.Text('if \_\_name\_\_ == "\_\_main\_\_":', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' ds\_train, ds\_test, info = load\_data()', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model = create\_model(input\_shape=info.features["image"].shape)', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.start\_time = time.time()', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' model.fit(ds\_train, epochs=epochs, validation\_data=ds\_test, verbose=0,', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' steps\_per\_epoch=info.splits["train"].num\_examples // batch\_size,', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' validation\_steps=info.splits["test"].num\_examples // batch\_size,', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text(' callbacks=[history])', font\_family="Consolas", theme\_style=ft.TextThemeStyle.BODY\_LARGE),

ft.Text('Теперь разберем, что происходит в основном теле программы из строк:'),

ft.Text('– Вызывается функция load\_data() для загрузки данных.'),

ft.Text('– Вызывается функция create\_model() для создания модели.'),

ft.Text('– Модель обучается на данных с использованием метода fit().'),

ft.Text('– Обучение проводится в течение epochs эпох.'),

ft.Text('– Количество шагов обучения и проверки рассчитывается на основе размера пакета и количества примеров в наборах данных.'),

ft.Text('– Используются обратный вызов history.'),

ft.Text('Полученная модель способна показывать точность свыше 87% на данных валидации, модель хорошо обучена и демонстрирует способность делать точные предсказания на новых данных, которые она ранее не видела, модель успешно обобщает знания, полученные в процессе обучения, на новые ситуации.')

], alignment=ft.MainAxisAlignment.START, expand=True, scroll=True)

]

), ft.Row( [ft.Column([ft.TextButton('О программе',

icon='QUESTION\_MARK', disabled=True),

ft.Text('Сколько нейронов и эпох используется с той или иной сложностью?', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('Тестовая – 1 эпоха 1 нейрон'),

ft.Text('Низкая – 8 эпох 64 нейрона', color='GREEN'),

ft.Text('Средняя – 16 эпох 256 нейронов', color='YELLOW'),

ft.Text('Высокая – 32 эпох 1024 нейрона', color='RED'),

ft.Text('Очень высокая – 48 эпох 2048 нейронов', color='CYAN'),

ft.Text('Почему вкладки лог и результат не обновляются в реальном времени?', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('Технические ограничения.Лог и результат выводятся с помощью двух текстовых редакторов.'),

ft.Text('Какая версия Tensorflow и Keras используется?', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('Tensor 2.16,Keras 3.0'),

ft.Text('Что отвечает за вывод графического интерфеса?', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('Flet Framework. Flet Framework — это инструмент для разработки кроссплатформенных приложений, созданный на основе языка программирования Dart и использующийся в комбинации с Flutter.'),

ft.Text('Какая версия Python используется?', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('Python 3.11.9'),

ft.Text('Как считаются баллы?', theme\_style=ft.TextThemeStyle.TITLE\_LARGE,color='Blue'),

ft.Text('1.complexity\_coefficient (Коэффициент сложности): Определяется в зависимости от количества эпох (epochs), которые вы устанавливаете для обучения модели. Чем больше эпох, тем более сложной считается задача. Если количество эпох соответствует одному из заранее определенных значений (1, 8, 16, 32, 48), то устанавливается определенный коэффициент сложности. В противном случае используется базовый коэффициент.'),

ft.Text('2.accuracy (Точность): Это значение точности модели в процентах, полученное после завершения обучения.'),

ft.Text('3.max\_points (Максимальное количество баллов): Рассчитывается как произведение точности и коэффициента сложности, умноженного на 100. Это максимальное количество баллов, которое модель может получить.'),

ft.Text('4.training\_time\_coefficient (Коэффициент времени обучения): Это значение, которое учитывает время, затраченное на обучение модели. Оно рассчитывается как 1 минус отношение времени обучения к 3600. Если время обучения составляет менее часа, то этот коэффициент устанавливается как 0.1.'),

ft.Text('5.total\_points (Общее количество баллов): Рассчитывается как произведение максимального количества баллов на коэффициент времени обучения. Это округленное до ближайшего целого число.')],alignment=ft.MainAxisAlignment.START, expand=True, scroll=True),

container = ft.Container(content=screen\_list[0], expand=True)

def set\_screen(e):

container.content = screen\_list[e.control.selected\_index]

page.update()

# **Приложение 4**

Приложение 4.

[Setup]

AppName=CIFAR–10 benchmark

AppVersion=1.0

DefaultDirName={pf}\CIFAR–10 benchmark

DefaultGroupName=CIFAR–10 benchmark

OutputDir=Output

Compression=lzma2/ultra64

SetupIconFile="C:\Project\icon\_windows.ico"

SolidCompression=True

InternalCompressLevel=ultra64

[Files]

Source: "C:\Project\CIFAR–10 benchmark\CIFAR–10 benchmark.exe"; DestDir: "{app}"

Source: "C:\Project\CIFAR–10 benchmark\\*"; DestDir: "{app}"; Flags: recursesubdirs

Source: "C:\Project\tensorflow\_datasets\\*"; DestDir: "C:\Users\{username}\tensorflow\_datasets\"; Flags: recursesubdirs

Source: "C:\Project\VC\_redist.x64.exe"; DestDir: "{app}"; Flags: ignoreversion

[Run]

Filename: "{app}\VC\_redist.x64.exe"; Parameters: "/install /norestart"; Flags: postinstall skipifsilent

[Icons]

Name: "{commondesktop}\CIFAR–10 benchmark"; Filename: "{app}\CIFAR–10 benchmark.exe"

[Languages]

Name: "russian"; MessagesFile: "compiler:Languages\Russian.isl"

# **Приложение 5**

Приложение 5.

absl–py==2.1.0

annotated–types==0.6.0

anyio==4.3.0

arrow==1.3.0

astunparse==1.6.3

beepy==1.0.7

binaryornot==0.4.4

certifi==2024.2.2

chardet==5.2.0

charset–normalizer==3.3.2

click==8.1.7

colorama==0.4.6

cookiecutter==2.6.0

dm–tree==0.1.8

etils==1.8.0

fastapi==0.110.3

flatbuffers==24.3.25

flet==0.22.0

flet–core==0.22.0

flet–runtime==0.22.0

fsspec==2024.3.1

gast==0.5.4

google–pasta==0.2.0

googleapis–common–protos==1.63.0

grpcio==1.63.0

h11==0.14.0

h5py==3.11.0

httpcore==1.0.5

httptools==0.6.1

httpx==0.27.0

idna==3.7

immutabledict==4.2.0

importlib\_resources==6.4.0

Jinja2==3.1.3

keras==3.3.3

libclang==18.1.1

Markdown==3.6

markdown–it–py==3.0.0

MarkupSafe==2.1.5

mdurl==0.1.2

ml–dtypes==0.3.2

namex==0.0.8

numpy==1.26.4

oauthlib==3.2.2

opt–einsum==3.3.0

optree==0.11.0

packaging==23.2

promise==2.3

protobuf==4.25.3

psutil==5.9.8

pyarrow==16.0.0

pydantic==2.7.1

pydantic\_core==2.18.2

Pygments==2.17.2

pypng==0.20220715.0

python–dateutil==2.9.0.post0

python–dotenv==1.0.1

python–slugify==8.0.4

PyYAML==6.0.1

qrcode==7.4.2

repath==0.9.0

requests==2.31.0

rich==13.7.1

simpleaudio==1.0.4

six==1.16.0

sniffio==1.3.1

starlette==0.37.2

tensorboard==2.16.2

tensorboard–data–server==0.7.2

tensorflow==2.16.1

tensorflow–intel==2.16.1

tensorflow–io–gcs–filesystem==0.31.0

tensorflow–metadata==1.15.0

termcolor==2.4.0

text–unidecode==1.3

tfds–nightly==4.9.4.dev202405020044

toml==0.10.2

tqdm==4.66.2

types–python–dateutil==2.9.0.20240316

typing\_extensions==4.11.0

urllib3==2.2.1

uvicorn==0.29.0

watchdog==4.0.0

watchfiles==0.21.0

websockets==12.0

Werkzeug==3.0.2

wrapt==1.16.0

zipp==3.18.1