|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт Информационных Технологий

Кафедра Вычислительной Техники (ВТ)

**ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ**

по дисциплине

«Проектирование и обучение нейронных сетей»

|  |  |
| --- | --- |
| Выполнил студент группы:  ИКБО-15-22 | Оганнисян Г.А. |
| Принял старший преподаватель кафедры ВТ | Семенов Р.Э. |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2025 г. |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2025 г. |

Москва 2025 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc193367360)

[1 ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ ПО МЕТОДУ «ТРАНСФОРМЕР» С НАБОРОМ ДАННЫХ 5](#_Toc193367361)

[1.1 Описание алгоритма 5](#_Toc193367362)

[1.2 Программная реализация обучения нейронной сети по методу «Трансформер» 6](#_Toc193367363)

[2 ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ МЕТОДОМ «ГЕНЕРАТИВНАЯ СЕТЬ (DCGAN)» 8](#_Toc193367364)

[2.1 Описание алгоритма 8](#_Toc193367365)

[2.2 Программная реализация обучения нейронной сети методом «Генеративная сеть (DCGAN)» 8](#_Toc193367366)

[3 ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ С ИСПОЛЬЗОВАНИЕМ МЕТОДА ГРАФОВЫХ НЕЙРОННЫХ СЕТЕЙ (GCN) 11](#_Toc193367367)

[3.1 Описание алгоритма 11](#_Toc193367368)

[3.2 Программная реализация обучения нейронной сети с использованием алгоритма обратного распространения ошибки 11](#_Toc193367369)

[4 ПРИМЕНИМОСТЬ СВЕРТОЧНЫХ НЕЙРОСЕТЕЙ И ТРАНСФОРМЕРОВ К ЗАДАЧАМ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ 13](#_Toc193367370)

[4.1 Введение и цель работы 13](#_Toc193367371)

[4.2 Методика реализации 13](#_Toc193367372)

[4.2.1 Реализация сверточной нейросети 13](#_Toc193367373)

[4.2.2 Реализация трансформерной модели 14](#_Toc193367374)

[4.3 Эксперементальная часть 15](#_Toc193367375)

[4.3.1 Датасет 15](#_Toc193367376)

[4.3.2 Параметры моделей 15](#_Toc193367377)

[4.3.3 Результаты обучения 16](#_Toc193367378)

[4.4 Сравнительный анализ 19](#_Toc193367379)

[4.5 Примеры результатов предсказания 19](#_Toc193367380)

[4.6 Вывод и заключение 22](#_Toc193367381)

[ЗАКЛЮЧЕНИЕ 24](#_Toc193367382)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 25](#_Toc193367383)

[ПРИЛОЖЕНИЯ 26](#_Toc193367384)

**ВВЕДЕНИЕ**

В последние годы нейронные сети стали основой многих современных решений в области искусственного интеллекта, позволяя эффективно решать задачи обработки изображений, генерации контента, анализа графовых структур и многое другое. Развитие аппаратных ресурсов и появление новых архитектур нейронных сетей привели к появлению специализированных методов обучения, ориентированных на различные типы данных и прикладные задачи. В рамках данного отчёта рассматриваются четыре подхода к обучению нейронных сетей: метод «Трансформер», который использует механизмы внимания для глобального анализа входных данных; метод «Генеративная сеть (DCGAN)», применяемый для генерации новых образцов данных с помощью состязательной игры генератора и дискриминатора; метод «Графовые нейронные сети (GCN)», позволяющий эффективно работать с данными в виде графов; а также сравнительный анализ сверточных нейросетей и трансформеров в задаче классификации изображений.

Изучение различных алгоритмов обучения позволяет выбрать наиболее подходящий метод для конкретной задачи, учитывая особенности данных и требования к модели. Кроме того, понимание принципов работы различных методов обучения способствует разработке более эффективных и устойчивых нейронных сетей, способных справляться с разнообразными вызовами в реальных приложениях.

Таким образом, исследование и разработка методов обучения нейронных сетей продолжают оставаться важным направлением в области искусственного интеллекта, открывая новые возможности для создания интеллектуальных систем, способных решать сложные задачи и адаптироваться к изменяющимся условиям.

1. **ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ ПО МЕТОДУ «ТРАНСФОРМЕР» С НАБОРОМ ДАННЫХ**
   1. **Описание алгоритма**

Алгоритм машинного перевода, основанный на трансформерной архитектуре, эффективно сопоставляет представления исходного и целевого языков для генерации корректного перевода. Основная идея заключается в следующем:

Представление входных данных. Исходное предложение на русском языке и целевое предложение (английский перевод) сначала преобразуются в эмбеддинги с использованием специального слоя, после чего к ним добавляются позиционные кодировки. Это обеспечивает учет порядка токенов в предложении.

Обработка с помощью энкодера. Исходное предложение обрабатывается через несколько слоев энкодера, каждый из которых включает механизм многошагового внимания (MultiHeadAttention), слой нормализации и позиционно-зависимую полносвязную нейронную сеть. Благодаря механизму self-attention модель выделяет важные зависимости в предложении.

Генерация представления для перевода. Целевое предложение формируется с помощью декодера, построенного на базе self-attention. При этом применяется cross-attention, позволяющий учитывать закодированное представление исходного текста для выбора наиболее релевантных токенов на английском языке.

Формирование перевода. После объединения представлений декодера и информации об исходном предложении на каждом шаге генерируется логит, определяющий вероятность появления того или иного токена. На этапе инференса используется жадный алгоритм (greedy decoding), который последовательно выбирает токены с наибольшей вероятностью, что обеспечивает формирование связного перевода..

В итоге алгоритм позволяет с высокой точностью преобразовать текст с одного языка на другой, используя мощь механизмов внимания для учета семантических и синтаксических особенностей исходного текста.

* 1. **Программная реализация обучения нейронной сети по методу «Трансформер»**

Выполним программную реализацию модели для машинного перевода на языке Python (Приложения А – Е). Программная реализация организована модульно с использованием библиотеки PyTorch, что обеспечивает гибкость и удобство отладки. Основные компоненты реализации включают:

1. Базовые блоки трансформера.

* Позиционное кодирование. Реализовано синусоидальное позиционное кодирование, позволяющее модели учитывать относительный порядок токенов.
* MultiHeadAttention – механизм многошагового внимания, где входное представление разбивается на несколько «голов», каждая из которых обрабатывается независимо, а затем результаты объединяются.

1. Энкодер и декодер.

* EncoderLayer и TransformerEncoder – состоят из блоков self-attention, полносвязных сетей, слоев нормализации и dropout. Эти компоненты кодируют исходное предложение, выделяя его внутренние зависимости.
* DecoderLayer и TransformerDecoder – аналогичны энкодеру, но дополнительно используют cross-attention для интеграции информации из исходного предложения при формировании перевода.

1. Интеграция и декодирование.

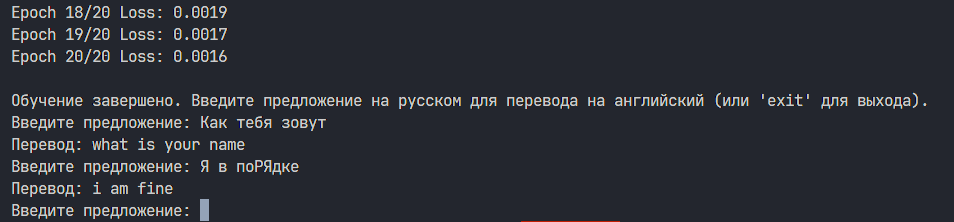
* Класс TransformerModel объединяет работу энкодера и декодера, а также включает финальный линейный слой, преобразующий декодированные представления в распределение вероятностей по словарю целевого языка. При обучении используется механизм teacher forcing, а при тестировании – жадное декодирование.

1. Инференс и обучение.

* Функции подготовки данных, формирования батчей с применением паддинга и маскирования, а также процедуры обучения и тестового перевода реализуют полный цикл работы модели. При обучении вычисляется кросс-энтропия для прогнозируемых токенов, а оптимизация проводится с использованием Adam-оптимизатора.

1. Работа с датасетами.

* В качестве обучающих данных используются пары предложений на русском и английском языках. Каждый пример включает исходное предложение, его перевод и соответствующую токенизацию, что позволяет обеспечить корректное сопоставление и обучение модели. Результат работы модели продемонстрирован на Рисунке 1.1



**Рисунок 1.1 – Пример перевода предложения с русского на английский, выполненного обученной моделью**

1. **ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ МЕТОДОМ «ГЕНЕРАТИВНАЯ СЕТЬ (DCGAN)»**
   1. **Описание алгоритма**

Проект реализует генеративно-состязательную сеть (GAN), состоящую из двух основных компонентов: генератора и дискриминатора.

Генератор принимает на вход случайный латентный вектор (шум) и последовательно проходит через серию линейных и (в данном случае) транспонированных слоёв, с последующим применением BatchNorm и ReLU, генерируя изображение фиксированного размера с пикселями, нормированными в диапазоне [-1, 1].

Дискриминатор – сверточная (или полностью связная, в зависимости от архитектуры) нейронная сеть, которая принимает изображение (реальное или сгенерированное) и, применяя серию слоёв с LeakyReLU и BatchNorm, оценивает вероятность того, что изображение является подлинным.

Во время обучения обе модели взаимодействуют в состязательном режиме: дискриминатор обучается различать реальные и сгенерированные изображения, а генератор – создавать изображения, способные обмануть дискриминатор. Обе сети оптимизируются с использованием функции потерь, основанной на бинарной кросс-энтропии, а также алгоритма Adam.

* 1. **Программная реализация обучения нейронной сети методом «Генеративная сеть (DCGAN)»**

Выполним программную реализацию генеративной сети на языке Python (Приложение Ж – И). Реализация дискриминатора.

Модель определена с использованием нескольких линейных слоёв, каждый из которых преобразует входное изображение, а на выходном слое применяется функция, позволяющая получить логит вероятности подлинности изображения.

Применяемые операции: свёртки, нормализация (BatchNorm), нелинейность (LeakyReLU).

Реализация генератора.

Генератор принимает случайный латентный вектор и с помощью последовательных слоёв, включающих линейные преобразования с ReLU и BatchNorm, поэтапно увеличивает размерность данных до требуемой размерности изображения. На последнем слое применяется Tanh для нормировки значений пикселей.

Основной цикл обучения GAN.

В тренировочном скрипте:

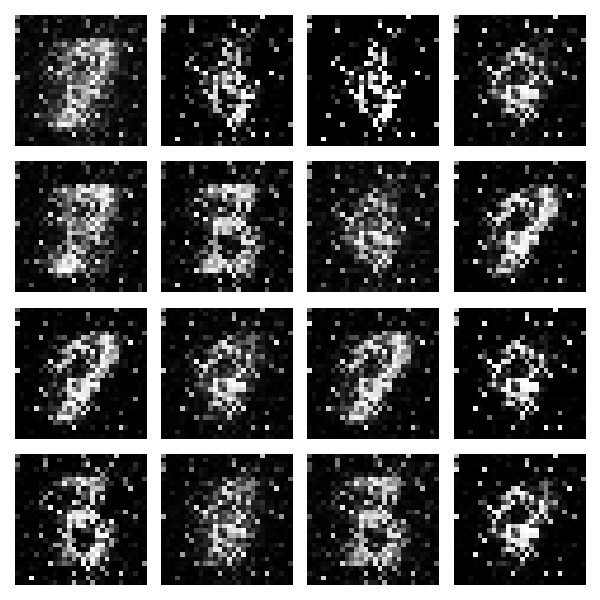
Загружается датасет изображений (например, MNIST или логотипы футбольных команд) с использованием необходимых преобразований (изменение размера, нормализация и т.д.).

Инициализируются модели генератора и дискриминатора, а также их веса с помощью специальной функции инициализации.

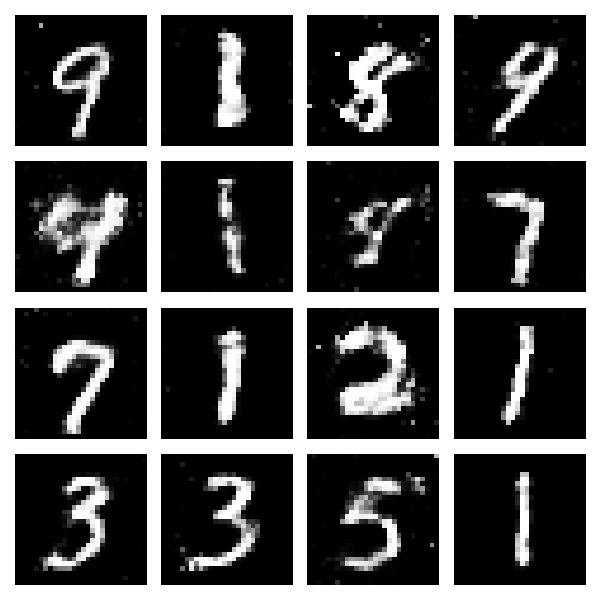
Настраиваются оптимизаторы и функция потерь (BCEWithLogitsLoss).

Обучение проводится в цикле, где для каждого батча сначала обновляется дискриминатор (с отдельными шагами для реальных и сгенерированных изображений), затем – генератор, стремящийся обмануть дискриминатор.

Периодически генерируются и сохраняются примеры изображений, полученные генератором, что позволяет визуально оценить динамику качества обучения. Результат работы модели продемонстрирован на Рисунках 2.1 и 2.2



**Рисунок 2.1 – Примеры сгенерированных изображений на начальных этапах обучения нейронной сети методом «Генеративная сеть»**



**Рисунок 2.2 – Примеры сгенерированных изображений по окончании обучения нейронной сети методом «Генеративная сеть»**

1. **ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ С ИСПОЛЬЗОВАНИЕМ МЕТОДА ГРАФОВЫХ НЕЙРОННЫХ СЕТЕЙ (GCN)** 
   1. **Описание алгоритма**

Проект направлен на анализ ассоциаций между лекарствами с использованием графовых нейронных сетей. Основные этапы алгоритма:

Генерация синтетического графа: На основе реального датасета, например, CORA, создается граф, где узлы представляют лекарства, а ребра – их взаимосвязи (например, совпадение фармакологических профилей или совместное назначение). Каждому лекарству назначается вектор признаков, а также метка, указывающая на наличие атипичного паттерна ассоциаций.

Классификация с помощью GCN: На основе созданного графа обучается графовая сверточная сеть (GCN), состоящая из двух слоев. Первый слой преобразует входные признаки в скрытое представление с применением функции ReLU и dropout для регуляризации, а второй слой генерирует логарифмы вероятностей для каждого класса, что позволяет классифицировать лекарства по паттерну ассоциаций.

Интерпретация результатов: После обучения вычисляется точность классификации, а также производится анализ степени узлов (числа ассоциаций). Статистический анализ (среднее значение и стандартное отклонение) используется для формирования объяснения, позволяющего интерпретировать результаты работы модели.

* 1. **Программная реализация обучения нейронной сети с использованием алгоритма обратного распространения ошибки**

Выполним программную реализацию модели с использованием библиотеки Torch Geometric на языке Python (Приложение К – М).

1. Генерация датасета:

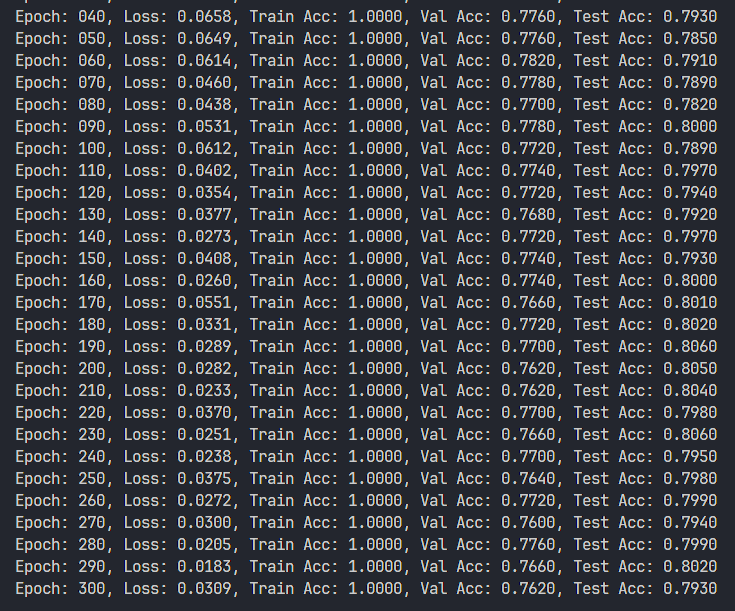
* Датасет CORA используется для представления графа, где узлы содержат признаки и метки классов.

1. Модель GCN:

* Класс GCN реализует графовую сверточную сеть с двумя слоями GCNConv. На первом слое применяется функция ReLU, после чего используется dropout для предотвращения переобучения. Второй слой выводит логарифмы вероятностей по классам, что позволяет проводить классификацию узлов.

1. Обучение и интерпретация:

* Основной скрипт включает функции train и test для обучения модели с использованием оптимизатора Adam и функции потерь Negative Log-Likelihood. Данные делятся на тренировочную, валидационную и тестовую выборки, а итоговые метрики точности выводятся для анализа работы модели. Результат работы программы продемонстрирован на Рисунке 3.1.



**Рисунок 3.1 – Результат работы программы обучения нейронной сети с использованием метода графовых нейронных сетей (GCN)**

1. **ПРИМЕНИМОСТЬ СВЕРТОЧНЫХ НЕЙРОСЕТЕЙ И ТРАНСФОРМЕРОВ К ЗАДАЧАМ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ**
   1. **Введение и цель работы**

В последние годы задачи классификации изображений играют ключевую роль в области компьютерного зрения, находя применение в самых разных сферах – от автономного вождения до медицинской диагностики и систем видеонаблюдения. С развитием глубокого обучения всё активнее применяются различные архитектуры нейросетей для решения данной задачи. Было проведено исследование эффективности двух подходов: сверточных нейросетей (CNN) и трансформеров, адаптированных для задач классификации изображений.

Цель работы – сравнить работоспособность и применимость сверточных нейросетей и трансформеров в условиях ограниченного объёма данных. Для сравнения использовался датасет, состоящий из 25705 изображений для обучения и 6820 изображений для тестирования.

* 1. **Методика реализации**
     1. **Реализация сверточной нейросети**

Для выполнения задачи была реализована простая сверточная нейросеть. В архитектуре (приложение Н) использованы три сверточных блока, каждый из которых состоит из сверточного слоя с ядром 3×3, функции активации ReLU и max-pooling, что приводит к уменьшению пространственных размеров изображения. После сверточных слоёв идет полносвязная часть, которая преобразует извлечённые признаки в логиты семи классов. Процесс обучения модели представлен в приложении О: данные загружаются с использованием безопасного класса загрузки изображений, применяются стандартные аугментации, а обучение происходит с использованием оптимизатора Adam. Также реализован инференс (приложение П), позволяющий проводить предсказания на новых изображениях.

* + 1. **Реализация трансформерной модели**

В качестве альтернативного подхода реализована трансформерная модель с архитектурой «энкодер – декодер», которые описаны в приложениях Р и С соотсветственно. В этой модели для обработки изображений применяется механизм self-attention, позволяющий захватывать глобальные зависимости между частями изображения.

Основные компоненты:

* Patch Embedding: изображение размером 224×224 разбивается на патчи. Размер патча 14, что даёт 256 патчей, что представляет компромисс между детальностью и вычислительной нагрузкой.
* Энкодер: реализован с использованием стандартного TransformerEncoder с позиционным кодированием, которое добавляет информацию о положении патчей.
* Декодер: использует learnable query для получения представления, которое затем преобразуется в логиты классов с помощью полносвязного слоя.

В приложении Т представлен код полного цикла обучения трансформерной модели, включая настройку гиперпараметров (patch\_size=14, embed\_dim=640, 7 слоёв энкодера и декодера, nhead=8) и применение дополнительных аугментаций (RandomRotation, ColorJitter) для улучшения обобщающей способности модели. Программный код для инференса (приложение У), загружая сохранённые веса и выдает предсказания для заданного набора изображений.

* 1. **Эксперементальная часть**
     1. **Датасет**

Для экспериментов использовался собственный датасет, содержащий 25705 изображений для обучения и 6820 изображений для тестирования. Датасет охватывает 7 классов: bike, cars, cats, chair, dogs, fridge, motorbike. Такое распределение позволяет оценить, как способность моделей работать с большим объёмом данных, так и их устойчивость к различным типам объектов.

* + 1. **Параметры моделей**

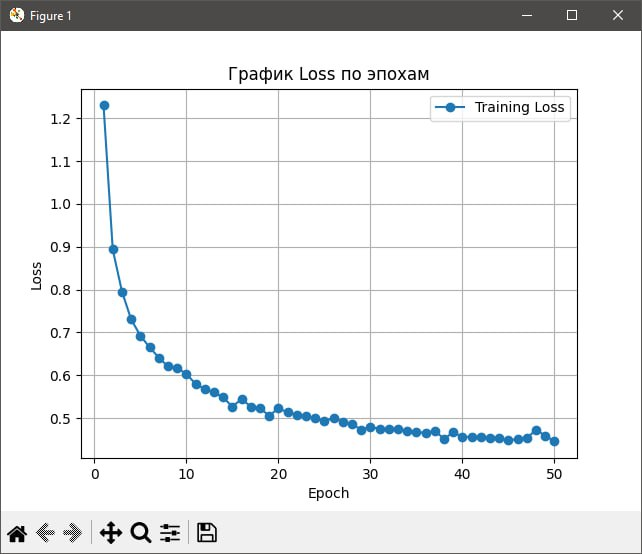
Для сверточной модели использовались базовые архитектурные параметры, позволяющие модели достаточно быстро сходиться и давать удовлетворительные результаты. В случае трансформера проводились эксперименты с различными настройками гиперпараметров. Первоначально трансформер имел более тяжелую конфигурацию (patch\_size=8, embed\_dim=768, 8 слоёв), что приводило к высоким вычислительным затратам и длительному времени обучения. В результате, для достижения баланса между качеством классификации и вычислительной эффективностью были выбраны следующие параметры для трансформера:

* patch\_size: 14 – размер патча (14×14 пикселей), даёт 256 патчей из изображения 224×224.
* embed\_dim: 640 – размерность вектора признаков для каждого патча.
* Число слоёв энкодера/декодера: 7 – количество повторяющихся блоков в энкодере и декодере.
* nhead: 8 – число голов в механизме многоголового внимания, позволяющих параллельно обрабатывать различные зависимости.
* learning rate: 1e-4 – шаг обновления весов, обеспечивающий стабильное и постепенное обучение.
* Оптимизатор: AdamW – оптимизатор, который корректно учитывает весовой распад (weight decay).
* Дополнительная аугментация: RandomRotation и ColorJitter – методы изменения ориентации и цветовых характеристик изображений для повышения обобщающей способности модели.
  + 1. **Результаты обучения**

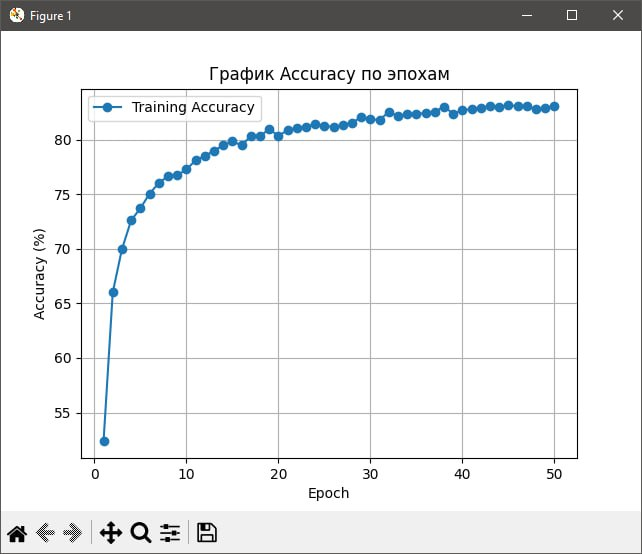
При обучении сверточной сети наблюдалось значительное улучшение показателей с каждой эпохой, а точность на тестовой выборке достигала приемлемых значений. Трансформерная модель с улучшенными параметрами показала лучшее извлечение глобальных зависимостей, однако имела проблемы с тонкой классификацией похожих классов (например, различение собак и кошек).

Графики обучения (loss и accuracy) для обеих моделей были построены и показаны на рисунках 4.1-4.4. Сравнительный анализ показал, что:

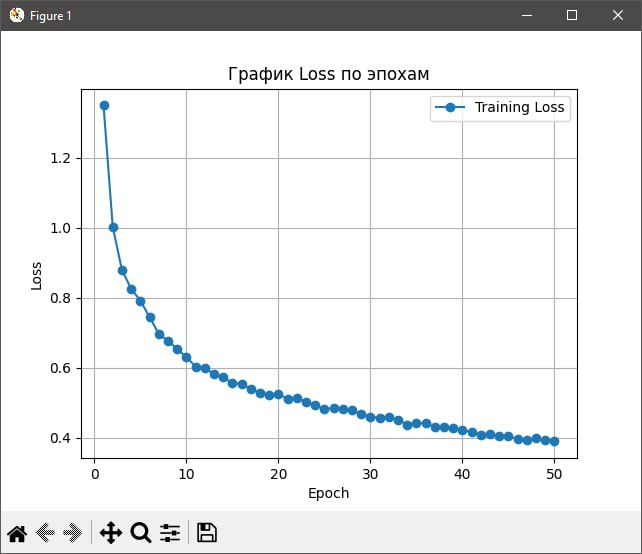
* Сверточная модель быстрее сходится и лучше различает мелкие детали благодаря локальным фильтрам.
* Трансформер, несмотря на более высокую вычислительную сложность, способен моделировать глобальные взаимосвязи, но требует более тщательной настройки гиперпараметров для тонкой классификации.



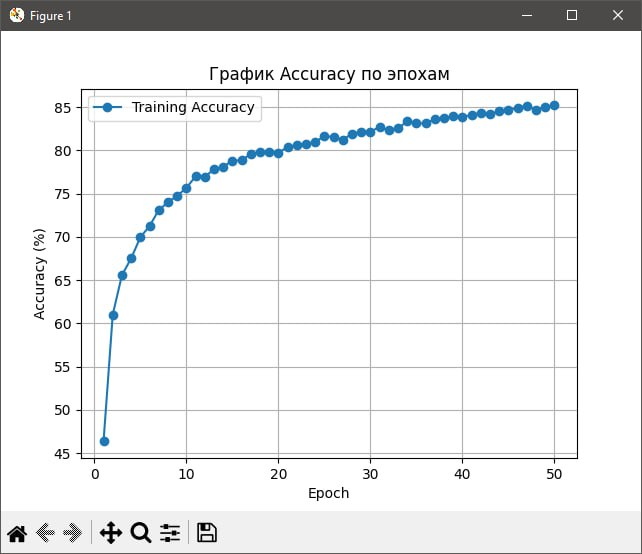
**Рисунок 4.1 – График потерь сверточной нейросети**



**Рисунок 4.2 – График точности сверточной нейросети**



**Рисунок 4.3 – График потерь трансферной модели**



**Рисунок 4.4 – График точности трансферной модели**

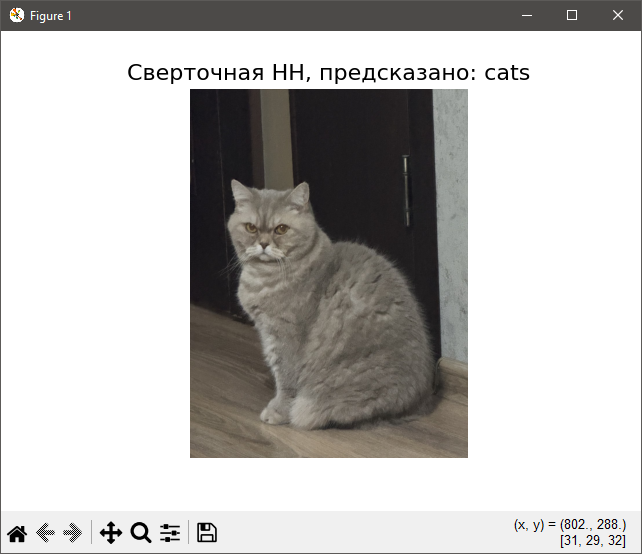
* 1. **Сравнительный анализ**

Сравнение результатов сверточной нейросети и трансформера выявило следующие ключевые моменты:

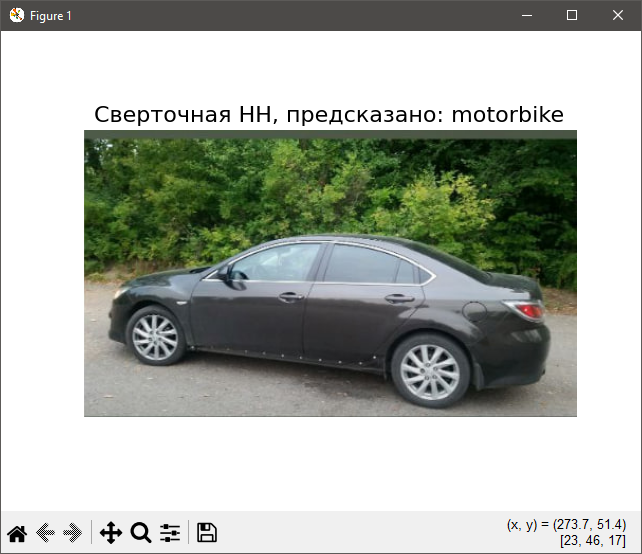
* Извлечение признаков: Сверточная сеть эффективно извлекает локальные признаки, что особенно важно для различения схожих объектов, таких как собаки и кошки. Трансформер же благодаря механизму self-attention способен учитывать глобальные зависимости, но при этом его эффективность зависит от точной настройки размеров патчей и размерности embedding.
* Скорость обучения и вычислительная сложность: Сверточная модель обучается быстрее и требует меньше вычислительных ресурсов. Трансформер при тяжелых настройках (маленький patch\_size, большая размерность embedding) может значительно замедлить процесс, что потребовало оптимизации гиперпараметров.
* Качество классификации: В эксперименте сверточная сеть давала лучшие результаты для тонкой классификации, в то время как трансформер настраивался для извлечения более глобальных зависимостей. Однако трансформер имеет потенциал для дальнейшего улучшения при оптимизации и увеличении объёма данных.
  1. **Примеры результатов предсказания**

Для наглядной демонстрации работы обученных моделей были выбраны несколько тестовых изображений, не входивших в обучающую выборку. Каждое изображение прошло через процедуру инференса – предварительную обработку (преобразования размера и нормализация) и дальнейшую подачу в модель для получения предсказанного класса.

На представленных ниже рисунках можно увидеть, как обе модели обрабатывают тестовые изображения и какой класс объекта в итоге предсказывают. Над каждым изображением указывается предсказанный класс, а также можно визуально сравнить правильность этого предсказания с реальным содержимым изображения.

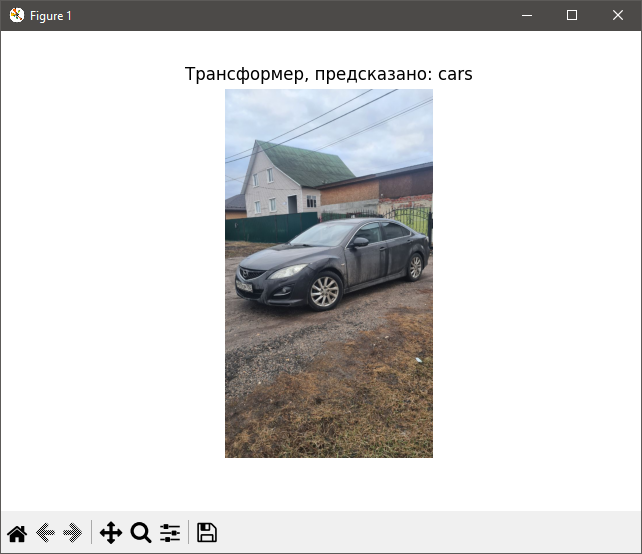


**Рисунок 4.5 – Результат правильного предсказания класса cats сверточной нейросети**

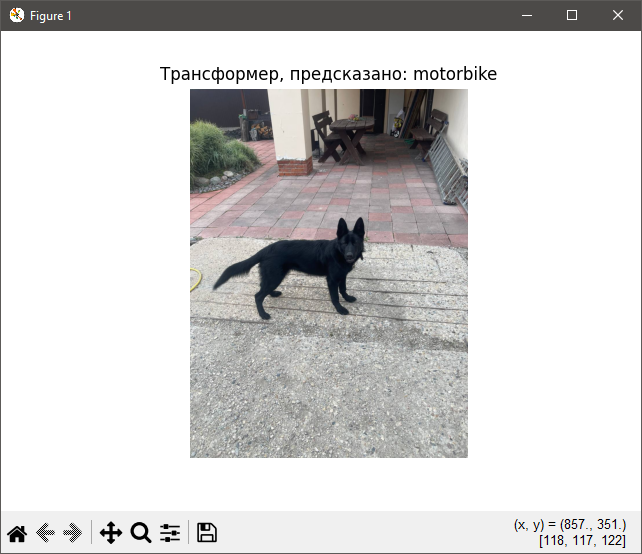


**Рисунок 4.6 – Результат не правильного предсказания класса сверточной нейросети**

На рисунке 4.6, изображена машина, а сверточная нейросесть ошибочно предсказала класс указав, что изображен мотоцикл.



**Рисунок 4.7 – Результат правильного предсказания класса cars трансформерной модели**



**Рисунок 4.8 – Результат не правильного предсказания класса трансформерной модели**

На рисунке 4.8, изображена собака, а модель ошибочно предсказала класс указав, что изображен мотоцикл.

* 1. **Вывод и заключение**

В ходе экспериментов было выявлено, что сверточная нейросеть демонстрирует быструю сходимость и эффективное извлечение локальных признаков, что особенно важно для тонкой классификации изображений при ограниченном объёме данных. Благодаря встроенным свёрточным фильтрам и операциям пулинга, CNN способна хорошо различать мелкие детали, что подтверждается высокими показателями точности для таких классов, как dogs и cats.

С другой стороны, трансформерная модель, реализованная в данной работе, основывается на механизме self-attention и использует patch embedding для разбиения изображения на небольшие патчи. Несмотря на то, что patch embedding может напоминать свёрточное преобразование, трансформер не является сверточной нейросетью в классическом понимании. Он ориентирован на моделирование глобальных зависимостей между патчами, что открывает новые возможности для извлечения сложных взаимосвязей. Однако эффективность трансформера сильно зависит от тонкой настройки гиперпараметров (размер патча, размерность embedding, глубина энкодера/декодера, число голов), и его обучение занимает значительно больше времени. Особое внимание было уделено проблеме различения похожих классов, таких как dogs и cats. Несмотря на достаточное количество обучающих примеров (около 4000 изображений на класс в тренировочном наборе и 1000 в тестовом), трансформерная модель время от времени путала эти классы. Кроме того, обсуждались вопросы вычислительной сложности. Конфигурации трансформера с меньшим размером патча (например, 8) и большей размерностью embedding (например, 768) приводили к квадратичному росту количества патчей и, как следствие, к существенному увеличению вычислительной нагрузки, что значительно замедляло процесс обучения. В результате был выбран компромиссный вариант, эти настройки позволили снизить нагрузку на GPU и сократить время обработки батчей, оставаясь при этом достаточно выразительными для решения задачи классификации. Тем не менее, в некоторых случаях трансформер всё ещё сталкивался с проблемами тонкого различения схожих классов.

Таким образом, можно сделать следующие выводы:

* Сверточная нейросеть обладает сильными индуктивными предпосылками для обработки изображений, быстро обучается и эффективно извлекает локальные признаки, что делает её предпочтительной для задач, где требуется высокая точность при ограниченных вычислительных ресурсах.
* Трансформерная модель имеет потенциал для моделирования глобальных зависимостей и может быть особенно полезной в задачах, где контекстуальные взаимосвязи между элементами изображения играют важную роль. Однако для достижения конкурентоспособных результатов требуется более тщательная настройка гиперпараметров и, возможно, больший объём данных.

Выбор архитектуры зависит от конкретных условий задачи, доступных вычислительных ресурсов и требуемой точности. При ограниченном объёме данных и необходимости быстрого обучения сверточные сети представляют собой оптимальное решение, в то время как трансформеры открывают перспективы для дальнейших исследований и могут стать эффективным инструментом при оптимизации параметров и использовании transfer learning.

**ЗАКЛЮЧЕНИЕ**

В ходе работы были рассмотрены и проанализированы различные методы обучения нейронных сетей, ориентированные на разные типы данных и задачи.

Трансформер продемонстрировал потенциал для моделирования глобальных взаимосвязей, что полезно в задачах обработки последовательностей, табличных данных и изображений.

Генеративная сеть (DCGAN) показала, как состязательный подход позволяет моделям обучаться генерировать реалистичные данные, что востребовано в задачах синтеза изображений и дополнения датасета, когда исходных данных недостаточно.

Графовые нейронные сети (GCN) оказались эффективным инструментом для анализа структурированных данных в форме графов, применяясь там, где важны взаимосвязи между объектами.

Сравнительный анализ сверточных нейросетей и трансформеров в задаче классификации изображений выявил преимущества и ограничения каждого подхода. Сверточные сети оказались более устойчивыми при тонком различии классов и быстрее обучаются при ограниченных данных, в то время как трансформеры, при правильной настройке, могут учитывать глобальные взаимосвязи и достигать высокой точности, но требуют больших ресурсов.

Таким образом, выбор метода обучения зависит от природы данных, доступных вычислительных ресурсов и особенностей решаемой задачи. Методы на базе трансформеров и GCN могут дать выигрыш в случаях, где важен глобальный контекст или графовая структура, а классические свёрточные сети (CNN) продолжают оставаться надёжным решением для большинства задач классификации изображений при умеренных ресурсах.

Дальнейшие исследования могут быть направлены на интеграцию нескольких методов (например, трансформеров и свёрточных слоёв) в единую архитектуру, а также на расширение подходов генерации и анализа данных с помощью GAN и GCN.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Широков, И.Б. Анализ технологий глубокого обучения с подкреплением для систем машинного зрения / И.Б. Широков, С.В. Колесова, В.А. Кучеренко, М.Ю. Серебряков / Известия ТулГУ 2022.
2. Сущеня, Р.В. Нейронные сети и их классификация. Основные виды нейронных сетей / Р.В. Сущеня, А.Э. Кокаев / Международный научный журнал «Вестник науки» 2023.
3. Паршин, А.И. Случайное мультимодальное глубокое обучение в задаче распознавания изображений / А.И. Паршин, М.Н. Аралов, В.Ф. Барабанов, Н.И. Гребенникова / Вестник Воронежского государственного университета 2021.

**ПРИЛОЖЕНИЯ**

Приложение А – Код реализации декодера для обучения нейронной сети методом «Трансформер».

Приложение Б – Код реализации энкодера для обучения нейронной сети методом «Трансформер».

Приложение В – Код реализации датасета для экстрактивного QA по теме футбола.

Приложение Г – Код реализации инференсного модуля для извлечения ответа из контекста.

Приложение Д – Код реализации процесса обучения модели экстрактивного QA методом «Трансформер».

Приложение Е – Код реализации архитектуры модели экстрактивного QA с улучшенной интеграцией вопроса и контекста.

Приложение Ж – Код реализации дискриминатора для генеративной сети (DCGAN).

Приложение З – Код реализации генератора для генеративной сети (DCGAN).

Приложение И – Код основного цикла обучения DCGAN с загрузкой данных, оптимизацией моделей и сохранением сгенерированных изображений.

Приложение К – Код генерации синтетического датасета для анализа ассоциаций между лекарствами.

Приложение Л – Код реализации графовой сверточной сети (GCN) для классификации лекарств по паттерну ассоциаций.

Приложение М – Код обучения, тестирования и интерпретации результатов модели для анализа атипичных паттернов ассоциаций лекарств.

Приложение Н – Код реализации архитектуры простой модели сверточной нейросети.

Приложение О – Код обучения сверточной нейросети.

Приложение П – Код инференса сверточной нейросети для предсказания классов.

Приложение Р – Код реализации энкодера с позиционным кодированием для трансформерной модели.

Приложение С – Код реализации декодера с декодера с learnable query для классификации.

Приложение Т – Код обучения трансформерной модели.

Приложение У – Код инференса трансформерной модели для предсказания классов.

**Приложение А**

Код реализации декодера для обучения нейронной сети методом «Трансформер».

*Листинг А – Код реализации декодера для обучения нейронной сети методом «Трансформер»*

|  |
| --- |
| import torch  import torch.nn as nn  from encoder import MultiHeadAttention  class DecoderLayer(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, d\_ff, dropout=0.1):  super().\_\_init\_\_()  self.self\_attn = MultiHeadAttention(d\_model, num\_heads, dropout)  self.cross\_attn = MultiHeadAttention(d\_model, num\_heads, dropout)  self.norm1 = nn.LayerNorm(d\_model)  self.norm2 = nn.LayerNorm(d\_model)  self.ffn = nn.Sequential(  nn.Linear(d\_model, d\_ff),  nn.ReLU(),  nn.Dropout(dropout),  nn.Linear(d\_ff, d\_model),  )  self.norm3 = nn.LayerNorm(d\_model)  self.dropout = nn.Dropout(dropout)  def forward(self, x, enc\_output, self\_mask=None, enc\_mask=None):  self\_attn\_output, \_ = self.self\_attn(x, x, x, self\_mask)  x = self.norm1(x + self\_attn\_output)  cross\_attn\_output, \_ = self.cross\_attn(x, enc\_output, enc\_output, enc\_mask)  x = self.norm2(x + cross\_attn\_output)  ffn\_output = self.ffn(x)  x = self.norm3(x + self.dropout(ffn\_output))  return x  class TransformerDecoder(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, d\_ff, num\_layers, vocab\_size, max\_seq\_len, dropout=0.1):  super().\_\_init\_\_()  self.embedding = nn.Embedding(vocab\_size, d\_model)  self.positional\_encoding = nn.Parameter(torch.zeros(1, max\_seq\_len, d\_model))  self.layers = nn.ModuleList([  DecoderLayer(d\_model, num\_heads, d\_ff, dropout)  for \_ in range(num\_layers)  ])  def forward(self, x, enc\_output, self\_mask=None, enc\_mask=None):  batch\_size, seq\_len = x.shape  x = self.embedding(x)  x = x + self.positional\_encoding[:, :seq\_len, :]  if self\_mask is not None:  self\_mask = self\_mask.unsqueeze(1).unsqueeze(2)  if enc\_mask is not None:  enc\_mask = enc\_mask.unsqueeze(1).unsqueeze(2)  for layer in self.layers:  x = layer(x, enc\_output, self\_mask, enc\_mask)  return x |

**Приложение Б**

Код реализации энкодера для обучения нейронной сети методом «Трансформер».

*Листинг Б – Код реализации энкодера для обучения нейронной сети методом «Трансформер»*

|  |
| --- |
| import torch  import torch.nn as nn  import torch.nn.functional as F  class ScaledDotProductAttention(nn.Module):  def \_\_init\_\_(self):  super().\_\_init\_\_()  def forward(self, Q, K, V, mask=None):  d\_k = Q.size(-1)  scores = torch.matmul(Q, K.transpose(-2, -1)) / torch.sqrt(torch.tensor(d\_k, dtype=torch.float32))  if mask is not None:  scores = scores.masked\_fill(mask == 0, -1e9)  attn\_weights = F.softmax(scores, dim=-1)  output = torch.matmul(attn\_weights, V)  return output, attn\_weights  class MultiHeadAttention(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, dropout=0.1):  super().\_\_init\_\_()  assert d\_model % num\_heads == 0, "d\_model must be divisible by num\_heads"  self.d\_model = d\_model  self.num\_heads = num\_heads  self.d\_k = d\_model // num\_heads  self.W\_q = nn.Linear(d\_model, d\_model)  self.W\_k = nn.Linear(d\_model, d\_model)  self.W\_v = nn.Linear(d\_model, d\_model)  self.W\_o = nn.Linear(d\_model, d\_model)  self.attention = ScaledDotProductAttention()  self.dropout = nn.Dropout(dropout)  def forward(self, x\_q, x\_k, x\_v, mask=None):  batch\_size = x\_q.size(0)  Q = self.W\_q(x\_q).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)  K = self.W\_k(x\_k).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)  V = self.W\_v(x\_v).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)  attn\_output, attn\_weights = self.attention(Q, K, V, mask)  attn\_output = attn\_output.transpose(1, 2).contiguous().view(batch\_size, -1, self.d\_model)  output = self.W\_o(attn\_output)  return self.dropout(output), attn\_weights |

*Окончание листинга Б*

|  |
| --- |
| class EncoderLayer(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, d\_ff, dropout=0.1):  super().\_\_init\_\_()  self.self\_attn = MultiHeadAttention(d\_model, num\_heads, dropout)  self.norm1 = nn.LayerNorm(d\_model)  self.ffn = nn.Sequential(  nn.Linear(d\_model, d\_ff),  nn.ReLU(),  nn.Dropout(dropout),  nn.Linear(d\_ff, d\_model),  )  self.norm2 = nn.LayerNorm(d\_model)  self.dropout = nn.Dropout(dropout)  def forward(self, x, mask=None):  attn\_output, \_ = self.self\_attn(x, x, x, mask)  x = self.norm1(x + attn\_output)  ffn\_output = self.ffn(x)  x = self.norm2(x + self.dropout(ffn\_output))  return x  class TransformerEncoder(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, d\_ff, num\_layers, vocab\_size, max\_seq\_len, dropout=0.1):  super().\_\_init\_\_()  self.embedding = nn.Embedding(vocab\_size, d\_model)  self.positional\_encoding = nn.Parameter(torch.zeros(1, max\_seq\_len, d\_model))  self.layers = nn.ModuleList([  EncoderLayer(d\_model, num\_heads, d\_ff, dropout)  for \_ in range(num\_layers)  ])  def forward(self, x, mask=None):  batch\_size, seq\_len = x.shape  x = self.embedding(x)  x = x + self.positional\_encoding[:, :seq\_len, :]  if mask is not None:  mask = mask.unsqueeze(1).unsqueeze(2)  for layer in self.layers:  x = layer(x, mask)  return x |

**Приложение В**

Код реализации датасета для экстрактивного QA по теме футбола.

*Листинг В – Код реализации датасета для экстрактивного QA по теме футбола*

|  |
| --- |
| import torch  from torch.utils.data import Dataset  from transformers import AutoTokenizer  class FootballQADataset(Dataset):  """  Датасет для экстрактивного QA по теме футбола.  Содержит (context, question, answer), а также вычисляет start/end позиции ответа в токенизированном context.  """  def \_\_init\_\_(self, model\_name="DeepPavlov/rubert-base-cased", max\_length=256):  super().\_\_init\_\_()  # 30 QA-примеров (русский текст) на тему футбола с акцентом на Real Madrid.  self.data = [  # 1  (  "В финале Лиги Чемпионов 2017 года Real Madrid победил Juventus со счетом 4:1, подтвердив свой статус лидера европейского футбола.",  "Какая команда выиграла финал Лиги Чемпионов 2017 года?",  "Real Madrid"  ),  # 2  (  "В период своего пребывания в Real Madrid Криштиану Роналду установил множество рекордов, став лучшим бомбардиром клуба.",  "Кто стал лучшим бомбардиром Real Madrid благодаря рекордам?",  "Криштиану Роналду"  ),  # 3  (  "Стадион Сантьяго Бернабеу является домашней ареной Real Madrid, где проходят матчи высочайшего уровня.",  "Как называется домашний стадион Real Madrid?",  "Сантьяго Бернабеу"  ),  # 4  (  "Зинедин Зидан, известный своим тактическим мышлением, несколько раз возглавлял тренерский штаб Real Madrid, приводя команду к победам.",  "Кто из тренеров руководил Real Madrid, демонстрируя тактическое мастерство?",  "Зинедин Зидан"  ),  # 5  (  "В испанском чемпионате Ла Лига Real Madrid неоднократно завоевывал титул чемпиона, подтверждая свое доминирование в стране.",  "В каком чемпионате Real Madrid неоднократно завоевывал титул чемпиона?", |

*Продолжение листинга В*

|  |
| --- |
| "Ла Лига"  ),  # 6  (  "Real Madrid известен не только атакующими успехами, но и надежной обороной, которая играет ключевую роль в победах команды.",  "За что, кроме атакующих успехов, ценят Real Madrid?",  "надежной обороной"  ),  # 7  (  "Перед началом нового сезона Real Madrid подписал несколько ключевых игроков, усилив состав команды.",  "Что сделал Real Madrid перед началом нового сезона?",  "подписал несколько ключевых игроков"  ),  # 8  (  "В матче против Атлетико Мадрид Real Madrid одержал победу со счетом 2:0, показав высокий уровень игры.",  "Какой результат был достигнут Real Madrid в матче против Атлетико Мадрид?",  "победу со счетом 2:0"  ),  # 9  (  "Real Madrid завоевал многочисленные трофеи, включая Лигу Чемпионов, Кубок Испании и Суперкубок.",  "Какие трофеи завоевал Real Madrid?",  "Лигу Чемпионов, Кубок Испании и Суперкубок"  ),  # 10  (  "В составе Real Madrid выступает талантливый полузащитник, известный своей универсальностью и точными передачами.",  "Какие качества выделяют полузащитника Real Madrid?",  "универсальностью и точными передачами"  ),  # 11  (  "В сезоне 2019-2020 Real Madrid продемонстрировал стабильно высокую результативность, забив более 70 голов.",  "Сколько голов забил Real Madrid в сезоне 2019-2020?",  "более 70 голов"  ),  # 12  (  "Звездный защитник Real Madrid Рафаэль Варан стал ключевым игроком, демонстрируя выдающиеся оборонительные качества.",  "Какой игрок Real Madrid стал ключевым защитником благодаря своим оборонительным качествам?",  "Рафаэль Варан"  ),  # 13  (  "В последнем трансферном окне Real Madrid сделал ставку на молодых талантов, подписав нескольких перспективных игроков.",  "На что сделала ставку Real Madrid в последнем трансферном окне?",  "подписав нескольких перспективных игроков"  ),  # 14 |

*Продолжение листинга В*

|  |
| --- |
| (  "Real Madrid известен своей атакующей стратегией, которая основывается на быстром переходе от защиты к атаке.",  "На чем основывается атакующая стратегия Real Madrid?",  "быстром переходе от защиты к атаке"  ),  # 15  (  "В рамках международных турниров Real Madrid неоднократно сталкивался с сильнейшими командами Европы.",  "С кем неоднократно сталкивался Real Madrid в международных турнирах?",  "сильнейшими командами Европы"  ),  # 16  (  "Real Madrid использует передовые аналитические методы для оценки эффективности игроков и оптимизации тактики.",  "Что использует Real Madrid для оценки эффективности игроков?",  "передовые аналитические методы"  ),  # 17  (  "Успехи Real Madrid на европейской арене сделали его одним из самых титулованных клубов в истории футбола.",  "Чем известен Real Madrid в истории футбола?",  "одним из самых титулованных клубов"  ),  # 18  (  "В своей истории Real Madrid не раз становился чемпионом как на национальном, так и на международном уровнях.",  "На каких уровнях становился чемпионом Real Madrid?",  "национальном, так и на международном уровнях"  ),  # 19  (  "Real Madrid часто организует благотворительные матчи, собирая средства для поддержки различных социальных проектов.",  "Для чего Real Madrid организует благотворительные матчи?",  "собирая средства для поддержки различных социальных проектов"  ),  # 20  (  "Футбол в Испании, где базируется Real Madrid, отличается высоким уровнем организации и страстной поддержкой болельщиков.",  "Что характеризует футбол в Испании, где базируется Real Madrid?",  "высоким уровнем организации и страстной поддержкой болельщиков"  ),  # 21  (  "В тренерском штабе Real Madrid работают специалисты с мировым опытом, что помогает команде достигать новых вершин.",  "Что помогает Real Madrid достигать новых вершин?",  "специалисты с мировым опытом"  ),  # 22  (  "За последние годы Real Madrid продемонстрировал значительный |

*Продолжение листинга В*

|  |
| --- |
| прогресс, улучшив как атакующую, так и оборонительную линии.",  "Что улучшил Real Madrid за последние годы?",  "атакующую, так и оборонительную линии"  ),  # 23  (  "Real Madrid активно внедряет новые технологии, такие как системы отслеживания игроков и видеоповторы для анализа матчей.",  "Какие новые технологии внедряет Real Madrid?",  "системы отслеживания игроков и видеоповторы"  ),  # 24  (  "Один из самых запоминающихся матчей в истории Real Madrid состоялся в 2002 году, когда команда одержала убедительную победу благодаря контратакам.",  "В каком году состоялся один из самых запоминающихся матчей Real Madrid?",  "2002 году"  ),  # 25  (  "Футболисты Real Madrid известны техникой, быстротой и способностью принимать решения в экстремальных ситуациях.",  "Какими качествами известны футболисты Real Madrid?",  "техникой, быстротой и способностью принимать решения"  ),  # 26  (  "История Real Madrid полна ярких моментов, каждый из которых вдохновляет болельщиков на новые достижения.",  "Что вдохновляет болельщиков Real Madrid на новые достижения?",  "ярких моментов"  ),  # 27  (  "Real Madrid известен не только успехами на поле, но и активной социальной позицией, участвуя в различных благотворительных инициативах.",  "Чем, помимо успехов на поле, известен Real Madrid?",  "активной социальной позицией"  ),  # 28  (  "В матчах против внутренних соперников, таких как Атлетико Мадрид, Real Madrid демонстрирует высокую психологическую устойчивость.",  "Какую характеристику демонстрирует Real Madrid в матчах против Атлетико Мадрид?",  "высокую психологическую устойчивость"  ),  # 29  (  "Футбольный клуб Real Madrid был основан в начале XX века и с тех пор завоевал признание во всем мире.",  "Когда был основан футбольный клуб Real Madrid?",  "в начале XX века"  ),  # 30  (  "Real Madrid продолжает традиции великих футбольных клубов, постоянно стремясь к совершенству и новым победам.", |

*Продолжение листинга В*

|  |
| --- |
| "Какие традиции продолжает Real Madrid?", "В последнем трансферном окне Real Madrid сделал ставку на молодых талантов, подписав нескольких перспективных игроков.",  "На что сделала ставку Real Madrid в последнем трансферном окне?",  "подписав нескольких перспективных игроков"  ),  # 14  "традиции великих футбольных клубов"  )  ]  self.max\_length = max\_length  self.tokenizer = AutoTokenizer.from\_pretrained(model\_name)  self.examples = []  for (context, question, answer) in self.data:  context\_encoded = self.tokenizer(  context,  max\_length=self.max\_length,  truncation=True,  padding="max\_length",  return\_offsets\_mapping=True,  return\_tensors="pt"  )  question\_encoded = self.tokenizer(  question,  max\_length=self.max\_length,  truncation=True,  padding="max\_length",  return\_offsets\_mapping=False,  return\_tensors="pt"  )  context\_ids = context\_encoded["input\_ids"].squeeze(0)  context\_mask = context\_encoded["attention\_mask"].squeeze(0)  offsets = context\_encoded["offset\_mapping"].squeeze(0)  question\_ids = question\_encoded["input\_ids"].squeeze(0)  question\_mask = question\_encoded["attention\_mask"].squeeze(0)  idx = context.find(answer)  if idx == -1:  # Ответ не найден: предупреждение  print(f"WARNING: answer not found in context:\nContext: {context}\nAnswer: {answer}\n")  start\_char, end\_char = 0, 0  else:  start\_char = idx  end\_char = idx + len(answer)  # Определяем start/end токены  start\_token\_idx = 0  end\_token\_idx = 0  for i, (off\_s, off\_e) in enumerate(offsets.tolist()):  if off\_s <= start\_char < off\_e:  start\_token\_idx = i  if off\_s < end\_char <= off\_e:  end\_token\_idx = i  break  if start\_token\_idx >= self.max\_length:  start\_token\_idx = self.max\_length - 1 |

*Окончание листинга В*

|  |
| --- |
| if end\_token\_idx >= self.max\_length:  end\_token\_idx = self.max\_length - 1  self.examples.append({  "context": context,  "question": question,  "answer": answer,  "context\_ids": context\_ids,  "context\_mask": context\_mask,  "question\_ids": question\_ids,  "question\_mask": question\_mask,  "start\_positions": torch.tensor(start\_token\_idx, dtype=torch.long),  "end\_positions": torch.tensor(end\_token\_idx, dtype=torch.long),  })  def \_\_len\_\_(self):  return len(self.examples)  def \_\_getitem\_\_(self, idx):  return self.examples[idx]  def get\_question(self, idx):  """Вернём вопрос для отладки"""  return self.examples[idx]["question"] |

**Приложение Г**

Код реализации инференсного модуля для извлечения ответа из контекста.

*Листинг Г – Код реализации обучения нейронной сети с гауссовой радиально-базисной функцией*

|  |
| --- |
| import re  import torch  import torch.nn.functional as F  from Football\_Dataset import FootballQADataset  # from dima\_dataset import FoodDeliveryQADataset  from transformeer import TransformerEncoder, TransformerDecoder, TransformerForExtractiveQA  def extract\_answer(context\_ids, start\_idx, end\_idx, tokenizer):  """  Извлекает строку ответа из токенов с учетом:  - Если токен перед start\_idx равен "-" (дефис), включаем предыдущие два токена.  - Если токен начинается с "##", расширяем span назад.  - Если следующие токены начинаются с "##", расширяем span вперед.  - Если в контексте присутствует двоеточие, начинаем ответ сразу после него.  После декодирования удаляем лишние пробелы вокруг дефиса.  """  tokens = tokenizer.convert\_ids\_to\_tokens(context\_ids.tolist())  # Если обнаружено двоеточие в контексте, считаем, что ответ начинается сразу после него.  colon\_idx = None  for i in range(start\_idx):  if tokens[i] == ":":  colon\_idx = i  if colon\_idx is not None and start\_idx > colon\_idx + 1:  start\_idx = colon\_idx + 1  # Если непосредственно перед начальным токеном стоит дефис, включаем предыдущие два токена  if start\_idx >= 2 and tokens[start\_idx - 1] == "-":  start\_idx = start\_idx - 2  # Расширяем назад, если начальный токен является subword (начинается с "##")  while start\_idx > 0 and tokens[start\_idx].startswith("##"):  start\_idx -= 1  # Расширяем вперед, если следующие токены начинаются с "##"  while end\_idx + 1 < len(tokens) and tokens[end\_idx + 1].startswith("##"):  end\_idx += 1  ans\_ids = context\_ids[start\_idx:end\_idx + 1]  answer = tokenizer.decode(ans\_ids, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True)  answer = answer.strip()  # Убираем пробелы вокруг дефиса (например, превращаем "чат - боты" в "чат-боты")  answer = re.sub(r'\s\*-\s\*', '-', answer)  return answer |

*Продолжение листинга Г*

|  |
| --- |
| def find\_best\_span\_exhaustive(start\_logits, end\_logits, max\_answer\_len=15, length\_bonus=0.5):  """  Перебирает все допустимые пары (s, e) с условием:  - e >= s  - (e - s + 1) <= max\_answer\_len  При расчете итогового score к сумме логитов добавляется бонус, пропорциональный длине спана.  Это помогает отдавать предпочтение более длинным спанам.  """  best\_score = float("-inf")  best\_s, best\_e = 0, 0  n = start\_logits.size(0)  for s in range(n):  for e in range(s, min(s + max\_answer\_len, n)):  span\_length = e - s + 1  bonus = length\_bonus \* (span\_length - 1) # бонус 0 для span длиной 1  score = start\_logits[s].item() + end\_logits[e].item() + bonus  if score > best\_score:  best\_score = score  best\_s, best\_e = s, e  return best\_s, best\_e, best\_score  def run\_inference\_single\_context(model, context\_ids, context\_mask, question\_ids, question\_mask, tokenizer,  max\_answer\_len=15):  with torch.no\_grad():  start\_logits, end\_logits = model(  context\_ids.unsqueeze(0),  question\_ids,  context\_mask.unsqueeze(0),  question\_mask  )  start\_logits = start\_logits[0]  end\_logits = end\_logits[0]  s\_idx, e\_idx, score\_val = find\_best\_span\_exhaustive(start\_logits, end\_logits, max\_answer\_len)  extracted = extract\_answer(context\_ids, s\_idx, e\_idx, tokenizer)  return extracted, score\_val  def run\_inference\_strict\_question(model, question, dataset\_obj, device="cpu", max\_answer\_len=15):  model.eval()  model.to(device)  q\_enc = dataset\_obj.tokenizer(  question,  max\_length=dataset\_obj.max\_length,  truncation=True,  padding="max\_length",  return\_tensors="pt"  )  q\_ids = q\_enc["input\_ids"].to(device)  q\_mask = q\_enc["attention\_mask"].to(device)  matched\_index = None  for i in range(len(dataset\_obj)):  ds\_question = dataset\_obj[i]["question"] |

*Продолжение листинга Г*

|  |
| --- |
| if ds\_question.strip() == question.strip():  matched\_index = i  print(f"(SpA) Найдено точное совпадение вопроса в датасете: index={i}")  break  if matched\_index is not None:  sample = dataset\_obj[matched\_index]  c\_ids = sample["context\_ids"].to(device)  c\_mask = sample["context\_mask"].to(device)  answer, score\_val = run\_inference\_single\_context(  model, c\_ids, c\_mask,  q\_ids, q\_mask,  dataset\_obj.tokenizer,  max\_answer\_len=max\_answer\_len  )  print(f"(SpA) Используем контекст #{matched\_index}, score={score\_val:.2f}, answer='{answer}'")  return answer  else:  print("(SpA) Вопрос не найден в датасете, fallback -> перебор всех контекстов")  return run\_inference\_all\_contexts(model, q\_ids, q\_mask, dataset\_obj, device=device,  max\_answer\_len=max\_answer\_len)  def run\_inference\_all\_contexts(model, question\_ids, question\_mask, dataset\_obj, device="cpu", max\_answer\_len=15):  best\_global\_score = float("-inf")  best\_answer = ""  with torch.no\_grad():  for i in range(len(dataset\_obj)):  sample = dataset\_obj[i]  c\_ids = sample["context\_ids"].to(device)  c\_mask = sample["context\_mask"].to(device)  extracted, score\_val = run\_inference\_single\_context(  model, c\_ids, c\_mask,  question\_ids, question\_mask,  dataset\_obj.tokenizer,  max\_answer\_len=max\_answer\_len  )  print(f"[Context {i}] score={score\_val:.2f}, answer='{extracted}'")  if score\_val > best\_global\_score:  best\_global\_score = score\_val  best\_answer = extracted  return best\_answer  def main():  dataset = FootballQADataset(model\_name="DeepPavlov/rubert-base-cased", max\_length=256)  real\_vocab\_size = len(dataset.tokenizer)  encoder = TransformerEncoder(  d\_model=128,  num\_heads=4, |

*Окончание листинга Г*

|  |
| --- |
| d\_ff=256,  num\_layers=2,  vocab\_size=real\_vocab\_size,  max\_seq\_len=256  )  decoder = TransformerDecoder(  d\_model=128,  num\_heads=4,  d\_ff=256,  num\_layers=2,  vocab\_size=real\_vocab\_size,  max\_seq\_len=256  )  model = TransformerForExtractiveQA(encoder, decoder, d\_model=128, num\_heads=4, dropout=0.1)  try:  model.load\_state\_dict(torch.load("football\_qa.pt", map\_location="cpu"))  print("Загружены веса из 'transformer\_qa.pt'")  except FileNotFoundError:  print("Внимание: нет файла 'transformer\_qa.pt'. Модель не обучена...")  question\_text = input("\nВведите вопрос: ")  best\_answer = run\_inference\_strict\_question(  model,  question\_text,  dataset,  device="cpu",  max\_answer\_len=15  )  print("\n=== Результат ===")  print("Вопрос:", question\_text)  print("Найденный ответ:", best\_answer)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**Приложение Д**

Код реализации процесса обучения модели экстрактивного QA методом «Трансформер».

*Листинг Д – Код реализации процесса обучения модели экстрактивного QA методом «Трансформер»*

|  |
| --- |
| import torch  import torch.nn.functional as F  import torch.optim as optim  from torch.utils.data import DataLoader  from dima\_dataset import FoodDeliveryQADataset  from transformeer import TransformerEncoder, TransformerDecoder, TransformerForExtractiveQA  def extract\_answer\_from\_ids(context\_ids, start\_idx, end\_idx, tokenizer):  """  Извлекает ответ из контекста с учетом субсловной токенизации:  - Если начальный токен начинается с "##", расширяет span назад.  - Если после end\_idx идут токены, начинающиеся с "##", расширяет span вперед.  """  tokens = tokenizer.convert\_ids\_to\_tokens(context\_ids.tolist())  # Расширяем назад, если токен начинается с "##"  while start\_idx > 0 and tokens[start\_idx].startswith("##"):  start\_idx -= 1  # Расширяем вперед, если следующие токены начинаются с "##"  while end\_idx + 1 < len(tokens) and tokens[end\_idx + 1].startswith("##"):  end\_idx += 1  ans\_ids = context\_ids[start\_idx:end\_idx + 1]  answer = tokenizer.decode(ans\_ids, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True)  return answer.strip()  def train\_qa\_model(model, dataloader, epochs=50, lr=1e-4, device="cpu"):  model.to(device)  optimizer = optim.Adam(model.parameters(), lr=lr)  for epoch in range(epochs):  model.train()  total\_loss = 0.0  for batch\_idx, batch in enumerate(dataloader):  context\_ids = batch["context\_ids"].to(device)  context\_mask = batch["context\_mask"].to(device)  question\_ids = batch["question\_ids"].to(device)  question\_mask = batch["question\_mask"].to(device)  start\_positions = batch["start\_positions"].to(device)  end\_positions = batch["end\_positions"].to(device)  start\_logits, end\_logits = model(context\_ids, question\_ids, context\_mask, question\_mask)  loss\_start = F.cross\_entropy(start\_logits, start\_positions)  loss\_end = F.cross\_entropy(end\_logits, end\_positions) input\_vector = fixed\_colors[np.random.choice(len(fixed\_colors))]  loss = (loss\_start + loss\_end) / 2 |

*Продолжение листинга Д*

|  |
| --- |
| optimizer.zero\_grad()  loss.backward()  optimizer.step()  total\_loss += loss.item()  avg\_loss = total\_loss / len(dataloader)  print(f"Epoch [{epoch + 1}/{epochs}] - Loss: {avg\_loss:.4f}")  def debug\_check(model, dataset, device="cpu"):  """  Проверяем несколько примеров из датасета после обучения.  """  model.eval()  print("\n=== Debug check on dataset samples ===")  with torch.no\_grad():  for i in range(min(3, len(dataset))):  sample = dataset[i]  context\_ids = sample["context\_ids"].unsqueeze(0).to(device)  context\_mask = sample["context\_mask"].unsqueeze(0).to(device)  question\_ids = sample["question\_ids"].unsqueeze(0).to(device)  question\_mask = sample["question\_mask"].unsqueeze(0).to(device)  start\_logits, end\_logits = model(context\_ids, question\_ids, context\_mask, question\_mask)  start\_idx = torch.argmax(start\_logits[0]).item()  end\_idx = torch.argmax(end\_logits[0]).item()  answer = extract\_answer\_from\_ids(context\_ids[0], start\_idx, end\_idx, dataset.tokenizer)  print(f"\nSample #{i}: question = {sample['question']}")  print(f"True answer = {sample['answer']}")  print(f"Predicted = {answer}")  def main():  dataset = FoodDeliveryQADataset(model\_name="DeepPavlov/rubert-base-cased", max\_length=256)  dataloader = DataLoader(dataset, batch\_size=2, shuffle=True)  real\_vocab\_size = len(dataset.tokenizer)  encoder = TransformerEncoder(  d\_model=128,  num\_heads=4,  d\_ff=256,  num\_layers=2,  vocab\_size=real\_vocab\_size,  max\_seq\_len=256  )  decoder = TransformerDecoder(  d\_model=128,  num\_heads=4,  d\_ff=256,  num\_layers=2,  vocab\_size=real\_vocab\_size,  max\_seq\_len=256  )  model = TransformerForExtractiveQA(encoder, decoder, d\_model=128, num\_heads=4, dropout=0.1) |

*Окончание листинга Д*

|  |
| --- |
| train\_qa\_model(model, dataloader, epochs=50, lr=1e-4, device="cpu")  debug\_check(model, dataset, device="cpu")  torch.save(model.state\_dict(), "dima.pt")  print("\nModel saved to transformer\_qa.pt")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**Приложение Е**

Код реализации архитектуры модели экстрактивного QA с улучшенной интеграцией вопроса и контекста.

*Листинг Е – Код реализации архитектуры модели экстрактивного QA с улучшенной интеграцией вопроса и контекста*

|  |
| --- |
| import torch  import torch.nn as nn  import math  import torch.nn.functional as F  # --------------------------  # Базовые блоки трансформера  # --------------------------  class ScaledDotProductAttention(nn.Module):  def \_\_init\_\_(self):  super().\_\_init\_\_()  def forward(self, Q, K, V, mask=None):  d\_k = Q.size(-1)  scores = torch.matmul(Q, K.transpose(-2, -1)) / math.sqrt(d\_k)  if mask is not None:  scores = scores.masked\_fill(mask == 0, float('-inf'))  attn\_weights = torch.softmax(scores, dim=-1)  output = torch.matmul(attn\_weights, V)  return output, attn\_weights  class MultiHeadAttention(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, dropout=0.1):  super().\_\_init\_\_()  assert d\_model % num\_heads == 0, "d\_model must be divisible by num\_heads"  self.d\_model = d\_model  self.num\_heads = num\_heads  self.d\_k = d\_model // num\_heads  self.W\_q = nn.Linear(d\_model, d\_model)  self.W\_k = nn.Linear(d\_model, d\_model)  self.W\_v = nn.Linear(d\_model, d\_model)  self.W\_o = nn.Linear(d\_model, d\_model)  self.attention = ScaledDotProductAttention()  self.dropout = nn.Dropout(dropout)  def forward(self, x\_q, x\_k, x\_v, mask=None):  batch\_size = x\_q.size(0)  Q = self.W\_q(x\_q).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)  K = self.W\_k(x\_k).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)  V = self.W\_v(x\_v).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)  attn\_output, attn\_weights = self.attention(Q, K, V, mask) |

*Продолжение листинга Е*

|  |
| --- |
| attn\_output = attn\_output.transpose(1, 2).contiguous().view(batch\_size, -1, self.d\_model)  output = self.W\_o(attn\_output)  return self.dropout(output), attn\_weights  class EncoderLayer(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, d\_ff, dropout=0.1):  super().\_\_init\_\_()  self.self\_attn = MultiHeadAttention(d\_model, num\_heads, dropout)  self.norm1 = nn.LayerNorm(d\_model)  self.ffn = nn.Sequential(  nn.Linear(d\_model, d\_ff),  nn.ReLU(),  nn.Dropout(dropout),  nn.Linear(d\_ff, d\_model)  )  self.norm2 = nn.LayerNorm(d\_model)  self.dropout = nn.Dropout(dropout)  def forward(self, x, mask=None):  attn\_output, \_ = self.self\_attn(x, x, x, mask)  x = self.norm1(x + attn\_output)  ffn\_out = self.ffn(x)  x = self.norm2(x + self.dropout(ffn\_out))  return x  class TransformerEncoder(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, d\_ff, num\_layers, vocab\_size, max\_seq\_len, dropout=0.1):  super().\_\_init\_\_()  self.embedding = nn.Embedding(vocab\_size, d\_model)  self.positional\_encoding = nn.Parameter(torch.zeros(1, max\_seq\_len, d\_model))  self.layers = nn.ModuleList([  EncoderLayer(d\_model, num\_heads, d\_ff, dropout)  for \_ in range(num\_layers)  ])  def forward(self, x, mask=None):  batch\_size, seq\_len = x.shape  x = self.embedding(x)  x = x + self.positional\_encoding[:, :seq\_len, :]  if mask is not None:  mask = mask.unsqueeze(1).unsqueeze(2)  for layer in self.layers:  x = layer(x, mask)  return x  class DecoderLayer(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, d\_ff, dropout=0.1):  super().\_\_init\_\_()  self.self\_attn = MultiHeadAttention(d\_model, num\_heads, dropout)  self.cross\_attn = MultiHeadAttention(d\_model, num\_heads, dropout)  self.norm1 = nn.LayerNorm(d\_model)  self.norm2 = nn.LayerNorm(d\_model) |

*Продолжение листинга Е*

|  |
| --- |
| self.ffn = nn.Sequential(  nn.Linear(d\_model, d\_ff),  nn.ReLU(),  nn.Dropout(dropout),  nn.Linear(d\_ff, d\_model),  )  self.norm3 = nn.LayerNorm(d\_model)  self.dropout = nn.Dropout(dropout)  def forward(self, x, enc\_output, self\_mask=None, enc\_mask=None):  self\_attn\_out, \_ = self.self\_attn(x, x, x, self\_mask)  x = self.norm1(x + self\_attn\_out)  cross\_attn\_out, \_ = self.cross\_attn(x, enc\_output, enc\_output, enc\_mask)  x = self.norm2(x + cross\_attn\_out)  ffn\_out = self.ffn(x)  x = self.norm3(x + self.dropout(ffn\_out))  return x  class TransformerDecoder(nn.Module):  def \_\_init\_\_(self, d\_model, num\_heads, d\_ff, num\_layers, vocab\_size, max\_seq\_len, dropout=0.1):  super().\_\_init\_\_()  self.embedding = nn.Embedding(vocab\_size, d\_model)  self.positional\_encoding = nn.Parameter(torch.zeros(1, max\_seq\_len, d\_model))  self.layers = nn.ModuleList([  DecoderLayer(d\_model, num\_heads, d\_ff, dropout)  for \_ in range(num\_layers)  ])  def forward(self, x, enc\_output, self\_mask=None, enc\_mask=None):  batch\_size, seq\_len = x.shape  x = self.embedding(x)  x = x + self.positional\_encoding[:, :seq\_len, :]  if self\_mask is not None:  self\_mask = self\_mask.unsqueeze(1).unsqueeze(2)  if enc\_mask is not None:  enc\_mask = enc\_mask.unsqueeze(1).unsqueeze(2)  for layer in self.layers:  x = layer(x, enc\_output, self\_mask, enc\_mask)  return x  class Transformer(nn.Module):  def \_\_init\_\_(self, encoder, decoder):  super().\_\_init\_\_()  self.encoder = encoder  self.decoder = decoder  def forward(self, src, tgt, src\_mask=None, tgt\_mask=None):  enc\_out = self.encoder(src, src\_mask)  dec\_out = self.decoder(tgt, enc\_out, tgt\_mask, src\_mask)  return enc\_out, dec\_out |

*Окончание листинга Е*

|  |
| --- |
| # ----------------------------------------------------  # Изменённая QA-модель с cross-attention для фьюжна  # ----------------------------------------------------  class TransformerForExtractiveQA(nn.Module):  """  Модель экстрактивного QA с улучшенной интеграцией вопроса и контекста.  Вместо простого mean pooling применяется cross-attention для слияния представлений.  """  def \_\_init\_\_(self, encoder, decoder, d\_model, num\_heads=4, dropout=0.1):  super().\_\_init\_\_()  self.transformer = Transformer(encoder, decoder)  # Новый блок cross-attention: для каждого токена контекста вычисляем attended-представление вопроса  self.cross\_attn = MultiHeadAttention(d\_model, num\_heads, dropout)  self.qa\_outputs = nn.Linear(d\_model, 2)  def forward(self, context\_input, question\_input, context\_mask=None, question\_mask=None):  # Получаем представления контекста и вопроса  enc\_output, dec\_output = self.transformer(context\_input, question\_input, context\_mask, question\_mask)  # Применяем cross-attention: для каждого токена контекста вычисляем представление, сфокусированное на вопросе  attended\_question, \_ = self.cross\_attn(enc\_output, dec\_output, dec\_output)  # Объединяем исходное представление контекста с attended представлением  fused\_context = enc\_output + attended\_question  logits = self.qa\_outputs(fused\_context) # [B, C, 2]  start\_logits = logits[:, :, 0]  end\_logits = logits[:, :, 1]  return start\_logits, end\_logits |

**Приложение Ж**

Код реализации дискриминатора для генеративной сети (DCGAN).

*Листинг Ж – Код реализации дискриминатора для генеративной сети (DCGAN)*

|  |
| --- |
| import torch  import torch.nn as nn  class Discriminator(nn.Module):  def \_\_init\_\_(self, nc, ndf):  super(Discriminator, self).\_\_init\_\_()  self.main = nn.Sequential(  # Вход: (nc) x 32 x 32  nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),  nn.LeakyReLU(0.2, inplace=True),  # Размер: (ndf) x 16 x 16  nn.Conv2d(ndf, ndf \* 2, 4, 2, 1, bias=False),  nn.BatchNorm2d(ndf \* 2),  nn.LeakyReLU(0.2, inplace=True),  # Размер: (ndf\*2) x 8 x 8  nn.Conv2d(ndf \* 2, ndf \* 4, 4, 2, 1, bias=False),  nn.BatchNorm2d(ndf \* 4),  nn.LeakyReLU(0.2, inplace=True),  # Размер: (ndf\*4) x 4 x 4  nn.Conv2d(ndf \* 4, 1, 4, 1, 0, bias=False),  nn.Sigmoid() # Выход: вероятность реальности изображения  )  def forward(self, input):  return self.main(input).view(-1) |

**Приложение З**

Код реализации генератора для генеративной сети (DCGAN).

*Листинг З – Код реализации генератора для генеративной сети (DCGAN)*

|  |
| --- |
| import torch  import torch.nn as nn  class Generator(nn.Module):  def \_\_init\_\_(self, nz, ngf, nc):  super(Generator, self).\_\_init\_\_()  self.main = nn.Sequential(  # Вход: латентный вектор (nz x 1 x 1)  nn.ConvTranspose2d(nz, ngf \* 4, 4, 1, 0, bias=False),  nn.BatchNorm2d(ngf \* 4),  nn.ReLU(True),  # Размер: (ngf\*4) x 4 x 4  nn.ConvTranspose2d(ngf \* 4, ngf \* 2, 4, 2, 1, bias=False),  nn.BatchNorm2d(ngf \* 2),  nn.ReLU(True),  # Размер: (ngf\*2) x 8 x 8  nn.ConvTranspose2d(ngf \* 2, ngf, 4, 2, 1, bias=False),  nn.BatchNorm2d(ngf),  nn.ReLU(True),  # Размер: (ngf) x 16 x 16  nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),  nn.Tanh() # Выходные значения в диапазоне [-1, 1]  )  def forward(self, input):  return self.main(input) |

**Приложение И**

Код реализации функций для подготовки временных рядов на языке Python.

*Листинг И – Код реализации функций для подготовки временных рядов*

|  |
| --- |
| import os  import torch  import torch.nn as nn  import torch.optim as optim  import torchvision.datasets as dset  import torchvision.transforms as transforms  import torchvision.utils as vutils  from generator import Generator  from discriminator import Discriminator  # ------------------------  # Гиперпараметры  # ------------------------  dataroot = "./data/football" # Папка с изображениями (структура: data/football/<class\_name>/...)  batch\_size = 64 # Уменьшенный размер батча  image\_size = 32 # Размер изображений (32x32)  nc = 3 # Число каналов (3 для RGB)  nz = 100 # Размер латентного вектора (шум)  ngf = 64 # Число feature maps для генератора  ndf = 64 # Число feature maps для дискриминатора  num\_epochs = 500 # Количество эпох обучения (можете увеличить, если нужно)  lr = 0.0001 # Скорость обучения снижена  beta1 = 0.5 # Параметры Adam  beta2 = 0.9  device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  print("Используемое устройство:", device)  # ------------------------  # Трансформации и загрузка данных  # ------------------------  transform = transforms.Compose([  transforms.Resize(image\_size),  transforms.CenterCrop(image\_size),  transforms.RandomHorizontalFlip(p=0.5), # Аугментация для повышения разнообразия  transforms.ToTensor(),  transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  ])  dataset = dset.ImageFolder(root=dataroot, transform=transform)  dataloader = torch.utils.data.DataLoader(dataset, batch\_size=batch\_size, shuffle=True)  netG = Generator(nz, ngf, nc).to(device)  netD = Discriminator(nc, ndf).to(device)  def weights\_init(m):  classname = m.\_\_class\_\_.\_\_name\_\_  if "Conv" in classname:  nn.init.normal\_(m.weight.data, 0.0, 0.02)  elif "BatchNorm" in classname:  nn.init.normal\_(m.weight.data, 1.0, 0.02) |

*Продолжение листинга И*

|  |
| --- |
| nn.init.constant\_(m.bias.data, 0)  netG.apply(weights\_init)  netD.apply(weights\_init)  criterion = nn.BCELoss()  optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, beta2))  optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, beta2))  os.makedirs("output", exist\_ok=True)  print("Начало обучения...")  for epoch in range(num\_epochs):  for i, data in enumerate(dataloader):  # ---- (1) Обновляем дискриминатор (2 шага)  for \_ in range(2):  netD.zero\_grad()  real\_images = data[0].to(device)  b\_size = real\_images.size(0)  # Метки для реальных изображений  label = torch.full((b\_size,), 1.0, device=device)  output = netD(real\_images)  errD\_real = criterion(output, label)  errD\_real.backward()  # Генерация фейковых изображений  noise = torch.randn(b\_size, nz, 1, 1, device=device)  fake = netG(noise)  label.fill\_(0.0)  output = netD(fake.detach())  errD\_fake = criterion(output, label)  errD\_fake.backward()  errD = errD\_real + errD\_fake  optimizerD.step()  # ---- (2) Обновляем генератор (1 шаг)  netG.zero\_grad()  label.fill\_(1.0) # Цель: заставить дискриминатор принять фейковые изображения за реальные  output = netD(fake)  errG = criterion(output, label)  errG.backward()  optimizerG.step()  # Печать логов каждые 50 батчей  if i % 50 == 0:  print(f"[Epoch {epoch + 1}/{num\_epochs}][Batch {i}/{len(dataloader)}] "  f"Loss\_D: {errD.item():.4f} | Loss\_G: {errG.item():.4f}")  # ---- (3) Каждые 100 эпох генерируем 2 изображения и сохраняем их бок о бок  if (epoch + 1) % 1 == 0:  print(f"Эпоха {epoch + 1} завершена. Генерируем 2 изображения...")  noise\_for\_two = torch.randn(2, nz, 1, 1, device=device)  with torch.no\_grad():  generated = netG(noise\_for\_two).detach().cpu() # (2, nc, 32, 32)  if generated.size(0) == 2:  img1 = generated[0] # Первое сгенерированное изображение  img2 = generated[1] # Второе сгенерированное изображение |

*Окончание листинга И*

|  |
| --- |
| # Склеиваем изображения по ширине (получаем одну картинку, где слева img1, справа img2)  combined = torch.cat([img1, img2], dim=2)  out\_path = f"output/two\_generated\_epoch\_{epoch + 1}.png"  vutils.save\_image(combined, out\_path, normalize=True)  print(f"Сохранено: {out\_path}")  else:  print("Ошибка: не удалось сгенерировать 2 изображения.")  print("Обучение завершено.") |

**Приложение К**

Код генерации синтетического датасета для анализа ассоциаций между лекарствами.

*Листинг К – Код генерации синтетического датасета для анализа ассоциаций между лекарствами*

|  |
| --- |
| import networkx as nx  import numpy as np  import torch  from torch\_geometric.data import Data  def generate\_drug\_association\_dataset(num\_drugs, atypical\_ratio, num\_features, p\_edge):  """  Генерирует синтетический датасет для анализа ассоциаций между лекарствами.  :param num\_drugs: Количество лекарств.  :param atypical\_ratio: Доля лекарств с атипичным паттерном ассоциаций.  :param num\_features: Размерность признакового пространства для лекарств.  :param p\_edge: Вероятность наличия ассоциации между двумя лекарствами.  :return: Объект Data из torch\_geometric.  """  # Создаем случайный граф, где узлы – лекарства, а ребра – ассоциации между препаратами,  # отражающие, например, совпадения в фармакологическом профиле или совместное назначение.  G = nx.erdos\_renyi\_graph(n=num\_drugs, p=p\_edge)  edge\_index = torch.tensor(list(G.edges), dtype=torch.long).t().contiguous()  # Добавляем обратные ребра для неориентированного графа  if edge\_index.numel() > 0:  edge\_index = torch.cat([edge\_index, edge\_index[[1, 0]]], dim=1)  else:  edge\_index = torch.empty((2, 0), dtype=torch.long)  # Генерация случайных признаков для лекарств  x = torch.randn((num\_drugs, num\_features), dtype=torch.float)  # Создаем метки:  # 0 – лекарство с обычными паттернами ассоциаций,  # 1 – лекарство с атипичным паттерном ассоциаций.  labels = np.zeros(num\_drugs, dtype=np.int64)  num\_atypical = int(num\_drugs \* atypical\_ratio)  atypical\_indices = np.random.choice(num\_drugs, num\_atypical, replace=False)  labels[atypical\_indices] = 1  y = torch.tensor(labels, dtype=torch.long)  # Формируем объект Data для PyTorch Geometric  data = Data(x=x, edge\_index=edge\_index, y=y)  return data  if \_\_name\_\_ == '\_\_main\_\_':  # Пример генерации и вывода информации о датасете  data = generate\_drug\_association\_dataset(num\_drugs=1000, atypical\_ratio=0.3, num\_features=16, p\_edge=0.01)  print(f"Количество лекарств: {data.num\_nodes}")  print(f"Размерность признаков: {data.num\_node\_features}")  print(f"Количество ассоциаций: {data.num\_edges}") # Определение модели |

**Приложение Л**

Код реализации графовой сверточной сети (GCN) для классификации лекарств по паттерну ассоциаций.

*Листинг Л – Код реализации графовой сверточной сети (GCN) для классификации лекарств по паттерну ассоциаций*

|  |
| --- |
| import torch  import torch.nn.functional as F  from torch\_geometric.nn import GCNConv  class GCN(torch.nn.Module):  def \_\_init\_\_(self, num\_features, hidden\_channels, num\_classes):  """  Модель GCN для анализа взаимодействий между лекарствами.  :param num\_features: Размерность входных признаков лекарств.  :param hidden\_channels: Количество нейронов в скрытом слое.  :param num\_classes: Количество классов (низкий риск / высокий риск взаимодействий).  """  super(GCN, self).\_\_init\_\_()  self.conv1 = GCNConv(num\_features, hidden\_channels)  self.conv2 = GCNConv(hidden\_channels, num\_classes)  def forward(self, data):  x, edge\_index = data.x, data.edge\_index  x = self.conv1(x, edge\_index)  x = F.relu(x)  x = F.dropout(x, p=0.5, training=self.training)  x = self.conv2(x, edge\_index)  return x |

**Приложение М**

Код обучения, тестирования и интерпретации результатов модели для анализа атипичных паттернов ассоциаций лекарств.

*Листинг М – Код обучения, тестирования и интерпретации результатов модели для анализа атипичных паттернов ассоциаций лекарств*

|  |
| --- |
| import torch  import torch.nn.functional as F  from data.dataset import generate\_drug\_association\_dataset  from model.hetero\_gcn import GCN  # Генерация датасета с 30% лекарств с атипичным паттерном ассоциаций  data = generate\_drug\_association\_dataset(num\_drugs=1000, atypical\_ratio=0.3, num\_features=16, p\_edge=0.01)  # Разделение на обучающую и тестовую выборки (80/20)  num\_drugs = data.num\_nodes  num\_train = int(num\_drugs \* 0.8)  train\_mask = torch.zeros(num\_drugs, dtype=torch.bool)  train\_mask[:num\_train] = True  test\_mask = ~train\_mask  # Инициализация модели, оптимизатора и функции потерь  model = GCN(num\_features=data.num\_node\_features, hidden\_channels=32, num\_classes=2)  optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight\_decay=5e-4)  criterion = torch.nn.CrossEntropyLoss()  def train():  model.train()  optimizer.zero\_grad()  out = model(data)  loss = criterion(out[train\_mask], data.y[train\_mask])  loss.backward()  optimizer.step()  return loss.item()  def test():  model.eval()  logits = model(data)  pred = logits.argmax(dim=1)  train\_acc = (pred[train\_mask] == data.y[train\_mask]).sum().item() / train\_mask.sum().item()  test\_acc = (pred[test\_mask] == data.y[test\_mask]).sum().item() / test\_mask.sum().item()  return pred, train\_acc, test\_acc  def compute\_degrees(data):  # Вычисляем число ассоциаций для каждого лекарства (граф неориентированный)  degrees = torch.bincount(data.edge\_index[0], minlength=data.num\_nodes)  return degrees  def explain\_drug(drug\_id, degree, mean\_degree, std\_degree):  """  Формирует объяснение для лекарства на основе числа его ассоциаций. |

*Окончание листинга М*

|  |
| --- |
| Если число ассоциаций значительно выше среднего (mean + std), то препарат имеет  аномально большое число ассоциаций, что может свидетельствовать о потенциальной синергии.  Если число ассоциаций значительно ниже среднего (mean - std), то препарат имеет  аномально низкое число ассоциаций, что может указывать на недостаточную интеграцию в фармакологический профиль.  В противном случае признаки препарата не отклоняются существенно, однако модель выявила атипичный паттерн ассоциаций.  """  if degree > mean\_degree + std\_degree:  explanation = (f"препарат имеет аномально большое число ассоциаций: {degree} ассоциаций, "  f"среднее значение составляет {mean\_degree:.2f} (± {std\_degree:.2f}).")  elif degree < mean\_degree - std\_degree:  explanation = (f"препарат имеет аномально низкое число ассоциаций: {degree} ассоциаций, "  f"среднее значение составляет {mean\_degree:.2f} (± {std\_degree:.2f}).")  else:  explanation = ("структурные признаки препарата не отклоняются существенно от нормы, "  "но модель выявила атипичный паттерн ассоциаций.")  return explanation  if \_\_name\_\_ == '\_\_main\_\_':  num\_epochs = 5000  for epoch in range(1, num\_epochs + 1):  loss = train()  pred, train\_acc, test\_acc = test()  if epoch % 20 == 0:  print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}, Train Acc: {train\_acc:.4f}, Test Acc: {test\_acc:.4f}')  # Анализ результатов после обучения  pred = pred.cpu()  degrees = compute\_degrees(data).cpu()  mean\_degree = degrees.float().mean().item()  std\_degree = degrees.float().std().item()  print("\nАнализ результатов: классификация лекарств с атипичным паттерном ассоциаций")  atypical\_count = 0  for drug\_id in range(num\_drugs):  if pred[drug\_id].item() == 1:  atypical\_count += 1  degree = degrees[drug\_id].item()  reason = explain\_drug(drug\_id, degree, mean\_degree, std\_degree)  print(f"Препарат с id {drug\_id} классифицирован как имеющий атипичный паттерн ассоциаций. {reason}")  print(f"\nОбщее количество препаратов с атипичным паттерном ассоциаций: {atypical\_count}") |

**Приложение Н**

Код реализации архитектуры простой модели сверточной нейросети.

*Листинг Н – Код реализации архитектуры простой модели сверточной нейросети*

|  |
| --- |
| import torch  import torch.nn as nn  class SimpleCNN(nn.Module):  def \_\_init\_\_(self, num\_classes=7):  super(SimpleCNN, self).\_\_init\_\_()  self.features = nn.Sequential(  nn.Conv2d(3, 16, kernel\_size=3, padding=1),  nn.ReLU(),  nn.MaxPool2d(kernel\_size=2, stride=2),  nn.Conv2d(16, 32, kernel\_size=3, padding=1),  nn.ReLU(),  nn.MaxPool2d(kernel\_size=2, stride=2),  nn.Conv2d(32, 64, kernel\_size=3, padding=1),  nn.ReLU(),  nn.MaxPool2d(kernel\_size=2, stride=2),  )  # После трёх блоков max-pooling (224 -> 28) получается 64 \* 28 \* 28  self.classifier = nn.Sequential(  nn.Linear(64 \* 28 \* 28, 256),  nn.ReLU(),  nn.Dropout(0.5),  nn.Linear(256, num\_classes)  )  def forward(self, x):  x = self.features(x)  x = x.view(x.size(0), -1)  x = self.classifier(x)  return x загружена из '{model\_path}'") |

**Приложение О**

Код обучения сверточной нейросети.

*Листинг О – Код обучения сверточной нейросети*

|  |
| --- |
| import os  import time  import torch  import torch.nn as nn  import torch.optim as optim  import torchvision.transforms as transforms  from torch.utils.data import DataLoader  from PIL import Image, ImageFile  from torchvision.datasets import ImageFolder  import matplotlib.pyplot as plt  import warnings  from model import SimpleCNN # Файл с архитектурой модели  # Подавляем предупреждения PIL о палитровых изображениях с прозрачностью  warnings.filterwarnings("ignore", message="Palette images with Transparency expressed in bytes should be converted to RGBA images")  # Разрешаем загрузку обрезанных (truncated) изображений  ImageFile.LOAD\_TRUNCATED\_IMAGES = True  # Функция для безопасной загрузки изображения  def safe\_pil\_loader(path):  try:  with open(path, "rb") as f:  img = Image.open(f)  return img.convert("RGB")  except Exception as e:  print(f"Ошибка при загрузке файла {path}: {e}")  return None  # Подкласс ImageFolder, который фильтрует отсутствующие файлы  class SafeImageFolder(ImageFolder):  def \_\_init\_\_(self, root, transform=None, target\_transform=None, loader=safe\_pil\_loader):  super().\_\_init\_\_(root, transform=transform, target\_transform=target\_transform)  self.loader = loader  self.samples = [s for s in self.samples if os.path.exists(s[0])]  self.imgs = self.samples  def \_\_getitem\_\_(self, index):  path, target = self.samples[index]  sample = self.loader(path)  if sample is None:  raise FileNotFoundError(f"Не удалось загрузить изображение по пути: {path}")  if self.transform is not None:  sample = self.transform(sample)  if self.target\_transform is not None:  target = self.target\_transform(target)  return sample, target  def main(): |

*Продолжение листинга О*

|  |
| --- |
| # Пути к датасету  train\_dir = r"C:\Users\Litvein\Desktop\datas\train"  test\_dir = r"C:\Users\Litvein\Desktop\datas\test"  # Гиперпараметры  batch\_size = 32  num\_epochs = 50  learning\_rate = 0.001  num\_classes = 7 # bike, cars, cats, chair, dogs, fridge, motorbike  # Используем GPU, если доступно  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  print("Using device:", device)  # Трансформации для обучающей выборки  train\_transforms = transforms.Compose([  transforms.RandomResizedCrop(224),  transforms.RandomHorizontalFlip(),  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225])  ])  # Трансформации для тестовой выборки и инференса  test\_transforms = transforms.Compose([  transforms.Resize((224, 224)),  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225])  ])  # Загрузка датасетов с использованием SafeImageFolder  train\_dataset = SafeImageFolder(root=train\_dir, transform=train\_transforms)  test\_dataset = SafeImageFolder(root=test\_dir, transform=test\_transforms)  print("Train set size:", len(train\_dataset))  print("Test set size:", len(test\_dataset))  train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True,  num\_workers=4, pin\_memory=True)  test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False, num\_workers=2)  # Инициализация модели  model = SimpleCNN(num\_classes=num\_classes).to(device)  criterion = nn.CrossEntropyLoss()  optimizer = optim.Adam(model.parameters(), lr=learning\_rate)  # Списки для сохранения значений loss и accuracy по эпохам  epoch\_losses = []  epoch\_accuracies = []  epochs = []  # Обучение модели с таймером для каждой эпохи  for epoch in range(num\_epochs):  start\_time = time.time()  model.train()  running\_loss = 0.0  correct = 0  total = 0 |

*Окончание листинга О*

|  |
| --- |
| for images, labels in train\_loader:  images = images.to(device)  labels = labels.to(device)  optimizer.zero\_grad()  outputs = model(images)  loss = criterion(outputs, labels)  loss.backward()  optimizer.step()  running\_loss += loss.item() \* images.size(0)  \_, predicted = torch.max(outputs, 1)  correct += (predicted == labels).sum().item()  total += labels.size(0)  epoch\_loss = running\_loss / total  epoch\_acc = 100.0 \* correct / total  epoch\_time = time.time() - start\_time  print(  f"Epoch [{epoch + 1}/{num\_epochs}] | Loss: {epoch\_loss:.4f} | Accuracy: {epoch\_acc:.2f}% | Time: {epoch\_time:.2f} sec")  # Сохраняем метрики по эпохам  epoch\_losses.append(epoch\_loss)  epoch\_accuracies.append(epoch\_acc)  epochs.append(epoch + 1)  torch.save(model.state\_dict(), "model2.pth")  print("Model weights saved to model2.pth")  # Оценка модели на тестовой выборке  model.eval()  test\_correct = 0  test\_total = 0  with torch.no\_grad():  for images, labels in test\_loader:  images = images.to(device)  labels = labels.to(device)  outputs = model(images)  \_, predicted = torch.max(outputs, 1)  test\_correct += (predicted == labels).sum().item()  test\_total += labels.size(0)  test\_accuracy = 100.0 \* test\_correct / test\_total  print(f"Test Accuracy: {test\_accuracy:.2f}%")  # Построение графиков обучения: loss и accuracy по эпохам  plt.figure()  plt.plot(epochs, epoch\_losses, marker='o', label='Training Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.title('График Loss по эпохам')  plt.legend()  plt.grid(True)  plt.show()  plt.figure()  plt.plot(epochs, epoch\_accuracies, marker='o', label='Training Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy (%)')  plt.title('График Accuracy по эпохам')  plt.legend()  plt.grid(True)  plt.show()  if \_\_name\_\_ == '\_\_main\_\_':  main() |

**Приложение П**

Код инференса сверточной нейросети для предсказания классов.

*Листинг П – Код инференса сверточной нейросети для предсказания классов*

|  |
| --- |
| import torch  import torch.nn as nn  import torchvision.transforms as transforms  import matplotlib.pyplot as plt  from PIL import Image  from model import SimpleCNN  # Список классов (в том же порядке, что и при обучении)  classes = ["bike", "cars", "cats", "chair", "dogs", "fridge", "motorbike"]  def main():  # Укажите пути к трём изображениям  image\_paths = [  r"C:\Users\Litvein\Desktop\images\audi.jpg",  r"C:\Users\Litvein\Desktop\images\kreslo.jpg",  r"C:\Users\Litvein\Desktop\images\matya\_bmw4.jpg",  r"C:\Users\Litvein\Desktop\images\matya\_bmw.jpg",  r"C:\Users\Litvein\Desktop\images\ruvi3.jpg",  r"C:\Users\Litvein\Desktop\images\ruvi5.jpg",  r"C:\Users\Litvein\Desktop\images\chief.jpg",  r"C:\Users\Litvein\Desktop\images\ruvi.jpg",  r"C:\Users\Litvein\Desktop\images\fridgee.jpg",  r"C:\Users\Litvein\Desktop\images\motorcycle.jpg",  r"C:\Users\Litvein\Desktop\images\chair.jpg",  r"C:\Users\Litvein\Desktop\images\tosya.jpg",  r"C:\Users\Litvein\Desktop\images\bike.jpg"  ]  # Используем CUDA, если доступно  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  print("Using device:", device)  # Создаём ту же модель, что и в train.py  model = SimpleCNN(num\_classes=7).to(device)  # Загружаем сохранённые веса  model.load\_state\_dict(torch.load("model.pth", map\_location=device))  model.eval()  # Трансформации, аналогичные тестовым  transform = transforms.Compose([  transforms.Resize((224, 224)),  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225])  ])  for path in image\_paths:  # Открываем изображение  try:  image = Image.open(path).convert('RGB')  except Exception as e:  print(f"Ошибка при открытии файла {path}: {e}")  continue |

*Окончание листинга П*

|  |
| --- |
| # Применяем трансформации  input\_tensor = transform(image).unsqueeze(0).to(device)  # Предсказание  with torch.no\_grad():  outputs = model(input\_tensor)  \_, predicted = torch.max(outputs, 1)  predicted\_class = classes[predicted.item()]  # Выводим картинку и результат  plt.figure()  plt.imshow(image)  plt.title(f"Сверточная НН, предсказано: {predicted\_class}", fontsize=16)  plt.axis("off")  plt.show()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**Приложение Р**

Код реализации энкодера с позиционным кодированием для трансформерной модели.

*Листинг Р – Код реализации энкодера с позиционным кодированием для трансформерной модели*

|  |
| --- |
| import torch  import torch.nn as nn  import math  class PositionalEncoding(nn.Module):  def \_\_init\_\_(self, d\_model, max\_len=5000):  super(PositionalEncoding, self).\_\_init\_\_()  pe = torch.zeros(max\_len, d\_model)  position = torch.arange(0, max\_len).unsqueeze(1).float()  div\_term = torch.exp(torch.arange(0, d\_model, 2).float() \* (-math.log(10000.0) / d\_model))  pe[:, 0::2] = torch.sin(position \* div\_term)  pe[:, 1::2] = torch.cos(position \* div\_term)  pe = pe.unsqueeze(0) # (1, max\_len, d\_model)  self.register\_buffer('pe', pe)  def forward(self, x):  # x: (batch\_size, seq\_len, d\_model)  x = x + self.pe[:, :x.size(1)]  return x  class TransformerEncoder(nn.Module):  def \_\_init\_\_(self, d\_model=512, nhead=8, num\_layers=6, dim\_feedforward=2048, dropout=0.1):  super(TransformerEncoder, self).\_\_init\_\_()  encoder\_layer = nn.TransformerEncoderLayer(d\_model=d\_model,  nhead=nhead,  dim\_feedforward=dim\_feedforward,  dropout=dropout)  self.encoder = nn.TransformerEncoder(encoder\_layer, num\_layers=num\_layers)  self.pos\_encoder = PositionalEncoding(d\_model)  self.d\_model = d\_model  def forward(self, src):  # src: (batch\_size, seq\_len, d\_model)  src = self.pos\_encoder(src)  # TransformerEncoder принимает вход (seq\_len, batch\_size, d\_model)  src = src.transpose(0,1)  output = self.encoder(src)  output = output.transpose(0,1) # (batch\_size, seq\_len, d\_model)  return output |

**Приложение С**

Код реализации декодера с learnable query для классификации.

*Листинг С – Код реализации декодера с learnable query для классификации*

|  |
| --- |
| import torch  import torch.nn as nn  class TransformerDecoder(nn.Module):  def \_\_init\_\_(self, d\_model=512, nhead=8, num\_layers=6, dim\_feedforward=2048, dropout=0.1, num\_classes=7):  super(TransformerDecoder, self).\_\_init\_\_()  decoder\_layer = nn.TransformerDecoderLayer(d\_model=d\_model,  nhead=nhead,  dim\_feedforward=dim\_feedforward,  dropout=dropout)  self.decoder = nn.TransformerDecoder(decoder\_layer, num\_layers=num\_layers)  # Learnable query для классификации (одна позиция)  self.query = nn.Parameter(torch.randn(1, 1, d\_model))  self.fc = nn.Linear(d\_model, num\_classes)  self.d\_model = d\_model  def forward(self, memory):  # memory: (batch\_size, seq\_len, d\_model)  batch\_size = memory.size(0)  # расширяем query до размера батча: (1, batch\_size, d\_model)  query = self.query.expand(-1, batch\_size, -1)  # TransformerDecoder ожидает (target\_seq\_len, batch\_size, d\_model)  out = self.decoder(query, memory.transpose(0,1))  out = out.squeeze(0) # (batch\_size, d\_model)  logits = self.fc(out) # (batch\_size, num\_classes)  return logits |

**Приложение Т**

Код обучения трансформерной модели.

*Листинг Т – Код обучения трансформерной модели*

|  |
| --- |
| import os  import time  import math  import warnings  import torch  import torch.nn as nn  import torch.optim as optim  import torchvision.transforms as transforms  from torch.utils.data import DataLoader  from torchvision.datasets import ImageFolder  from torchvision.datasets.folder import default\_loader  from PIL import Image, ImageFile  from tqdm import tqdm  import matplotlib.pyplot as plt  # Подавляем предупреждения PIL о палитровых изображениях с прозрачностью  warnings.filterwarnings("ignore", message="Palette images with Transparency expressed in bytes should be converted to RGBA images")  # Разрешаем загрузку обрезанных изображений  ImageFile.LOAD\_TRUNCATED\_IMAGES = True  # Подкласс ImageFolder, использующий стандартный загрузчик изображений  class SafeImageFolder(ImageFolder):  def \_\_init\_\_(self, root, transform=None, target\_transform=None):  super().\_\_init\_\_(root, transform=transform, target\_transform=target\_transform)  self.loader = default\_loader  self.samples = [s for s in self.samples if os.path.exists(s[0])]  self.imgs = self.samples  def \_\_getitem\_\_(self, index):  path, target = self.samples[index]  sample = self.loader(path)  if sample.mode != "RGB":  sample = sample.convert("RGB")  if self.transform is not None:  sample = self.transform(sample)  if self.target\_transform is not None:  target = self.target\_transform(target)  return sample, target  # Импорт энкодера и декодера из отдельных файлов (убедитесь, что encoder.py и decoder.py находятся в той же папке)  from encoder import TransformerEncoder  from decoder import TransformerDecoder  # PatchEmbedding: разбивает изображение на патчи и проецирует их в пространство embedding  class PatchEmbedding(nn.Module):  def \_\_init\_\_(self, img\_size=224, patch\_size=14, in\_channels=3, embed\_dim=640):  super(PatchEmbedding, self).\_\_init\_\_()  self.num\_patches = (img\_size // patch\_size) \*\* 2 # (224/14)^2 = 16^2 = 256 патчей  self.proj = nn.Conv2d(in\_channels, embed\_dim, kernel\_size=patch\_size, |

*Продолжение листинга Т*

|  |
| --- |
| stride=patch\_size)  def forward(self, x):  # x: (batch\_size, in\_channels, img\_size, img\_size)  x = self.proj(x) # (batch\_size, embed\_dim, H, W) где H = W = 224/14 = 16  x = x.flatten(2) # (batch\_size, embed\_dim, num\_patches)  x = x.transpose(1, 2) # (batch\_size, num\_patches, embed\_dim)  return x  # TransformerClassifier: объединяет patch embedding, энкодер и декодер для классификации  class TransformerClassifier(nn.Module):  def \_\_init\_\_(self, img\_size=224, patch\_size=14, in\_channels=3, embed\_dim=640,  encoder\_layers=7, decoder\_layers=7, nhead=8, num\_classes=7):  super(TransformerClassifier, self).\_\_init\_\_()  self.patch\_embed = PatchEmbedding(img\_size, patch\_size, in\_channels, embed\_dim)  self.encoder = TransformerEncoder(d\_model=embed\_dim, nhead=nhead, num\_layers=encoder\_layers)  self.decoder = TransformerDecoder(d\_model=embed\_dim, nhead=nhead, num\_layers=decoder\_layers, num\_classes=num\_classes)  def forward(self, x):  # x: (batch\_size, 3, img\_size, img\_size)  x = self.patch\_embed(x)  memory = self.encoder(x)  logits = self.decoder(memory)  return logits  def main():  # Пути к датасету  train\_dir = r"C:\Users\Litvein\Desktop\datas\train"  test\_dir = r"C:\Users\Litvein\Desktop\datas\test"  # Гиперпараметры  batch\_size = 32  num\_epochs = 50  learning\_rate = 1e-4  num\_classes = 7  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  print("Using device:", device)  print("Train set size:", len(SafeImageFolder(train\_dir)))  print("Test set size:", len(SafeImageFolder(test\_dir)))  # Трансформации для обучающей выборки с аугментацией  train\_transforms = transforms.Compose([  transforms.RandomResizedCrop(224),  transforms.RandomHorizontalFlip(),  transforms.RandomRotation(10),  transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1),  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225])  ])  # Трансформации для тестовой выборки  test\_transforms = transforms.Compose([  transforms.Resize((224, 224)),  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406], |

*Продолжение листинга Т*

|  |
| --- |
| std=[0.229, 0.224, 0.225])  ])  train\_dataset = SafeImageFolder(root=train\_dir, transform=train\_transforms)  test\_dataset = SafeImageFolder(root=test\_dir, transform=test\_transforms)  train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True,  num\_workers=8, pin\_memory=True)  test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False,  num\_workers=8)  # Инициализация модели  model = TransformerClassifier(img\_size=224, patch\_size=14, in\_channels=3, embed\_dim=640,  encoder\_layers=7, decoder\_layers=7, nhead=8, num\_classes=num\_classes)  model = model.to(device)  criterion = nn.CrossEntropyLoss()  optimizer = optim.AdamW(model.parameters(), lr=learning\_rate)  # Scheduler с warm-up: первые 5 эпох линейно увеличивают LR, затем фиксированный LR  warmup\_epochs = 5  scheduler = optim.lr\_scheduler.LambdaLR(optimizer,  lr\_lambda=lambda epoch: (epoch+1)/warmup\_epochs if epoch < warmup\_epochs else 1.0)  # Списки для хранения метрик по эпохам  epoch\_losses = []  epoch\_accuracies = []  epochs = []  # Обучение модели с tqdm для отслеживания ETA  for epoch in range(num\_epochs):  start\_time = time.time()  model.train()  running\_loss = 0.0  correct = 0  total = 0  progress\_bar = tqdm(enumerate(train\_loader), total=len(train\_loader), desc=f"Epoch [{epoch+1}/{num\_epochs}]")  for batch\_idx, (images, labels) in progress\_bar:  images = images.to(device)  labels = labels.to(device)  optimizer.zero\_grad()  outputs = model(images)  loss = criterion(outputs, labels)  loss.backward()  optimizer.step()  running\_loss += loss.item() \* images.size(0)  \_, predicted = torch.max(outputs, 1)  correct += (predicted == labels).sum().item()  total += labels.size(0)  elapsed = time.time() - start\_time  batches\_done = batch\_idx + 1 |

*Окончание листинга Т*

|  |
| --- |
| batches\_left = len(train\_loader) - batches\_done  avg\_batch\_time = elapsed / batches\_done  eta = batches\_left \* avg\_batch\_time  progress\_bar.set\_postfix(loss=f"{loss.item():.4f}", ETA=f"{eta:.2f} sec")  scheduler.step()  epoch\_loss = running\_loss / total  epoch\_acc = 100.0 \* correct / total  epoch\_time = time.time() - start\_time  print(f"Epoch [{epoch+1}/{num\_epochs}] | Loss: {epoch\_loss:.4f} | Accuracy: {epoch\_acc:.2f}% | Time: {epoch\_time:.2f} sec | LR: {optimizer.param\_groups[0]['lr']:.6f}")  torch.cuda.empty\_cache()  # Сохраняем метрики для графиков  epoch\_losses.append(epoch\_loss)  epoch\_accuracies.append(epoch\_acc)  epochs.append(epoch+1)  torch.save(model.state\_dict(), "transformer\_model3.pth")  print("Model weights saved to transformer\_model3.pth")  # Оценка модели на тестовой выборке  model.eval()  test\_correct = 0  test\_total = 0  with torch.no\_grad():  for images, labels in test\_loader:  images = images.to(device)  labels = labels.to(device)  outputs = model(images)  \_, predicted = torch.max(outputs, 1)  test\_correct += (predicted == labels).sum().item()  test\_total += labels.size(0)  test\_accuracy = 100.0 \* test\_correct / test\_total  print(f"Test Accuracy: {test\_accuracy:.2f}%")  # Построение графиков обучения  plt.figure()  plt.plot(epochs, epoch\_losses, marker='o', label='Training Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.title('График Loss по эпохам')  plt.legend()  plt.grid(True)  plt.show()  plt.figure()  plt.plot(epochs, epoch\_accuracies, marker='o', label='Training Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy (%)')  plt.title('График Accuracy по эпохам')  plt.legend()  plt.grid(True)  plt.show()  if \_\_name\_\_ == '\_\_main\_\_':  main() |

**Приложение У**

Код инференса трансформерной модели для предсказания классов.

*Листинг У – Код инференса трансформерной модели для предсказания классов*

|  |
| --- |
| import os  import torch  import torch.nn as nn  import torchvision.transforms as transforms  from PIL import Image  import matplotlib.pyplot as plt  # Импорт энкодера и декодера из отдельных файлов (убедитесь, что encoder.py и decoder.py находятся в той же папке)  from encoder import TransformerEncoder  from decoder import TransformerDecoder  # Определяем PatchEmbedding, адаптированный под новые параметры  class PatchEmbedding(nn.Module):  def \_\_init\_\_(self, img\_size=224, patch\_size=14, in\_channels=3, embed\_dim=640):  super(PatchEmbedding, self).\_\_init\_\_()  self.num\_patches = (img\_size // patch\_size) \*\* 2 # (224/14)^2 = 16^2 = 256 патчей  self.proj = nn.Conv2d(in\_channels, embed\_dim, kernel\_size=patch\_size, stride=patch\_size)  def forward(self, x):  # x: (batch\_size, in\_channels, img\_size, img\_size)  x = self.proj(x) # (batch\_size, embed\_dim, H, W)  x = x.flatten(2) # (batch\_size, embed\_dim, num\_patches)  x = x.transpose(1, 2) # (batch\_size, num\_patches, embed\_dim)  return x  # Определяем TransformerClassifier с новыми гиперпараметрами  class TransformerClassifier(nn.Module):  def \_\_init\_\_(self, img\_size=224, patch\_size=14, in\_channels=3, embed\_dim=640,  encoder\_layers=7, decoder\_layers=7, nhead=8, num\_classes=7):  super(TransformerClassifier, self).\_\_init\_\_()  self.patch\_embed = PatchEmbedding(img\_size, patch\_size, in\_channels, embed\_dim)  self.encoder = TransformerEncoder(d\_model=embed\_dim, nhead=nhead, num\_layers=encoder\_layers)  self.decoder = TransformerDecoder(d\_model=embed\_dim, nhead=nhead, num\_layers=decoder\_layers,  num\_classes=num\_classes)  def forward(self, x):  # x: (batch\_size, 3, img\_size, img\_size)  x = self.patch\_embed(x)  memory = self.encoder(x)  logits = self.decoder(memory)  return logits  def main():  # Устройство: GPU или CPU  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  print("Using device:", device) |

*Продолжение листинга У*

|  |
| --- |
| # Инициализируем модель с такими же параметрами, как при обучении  model = TransformerClassifier(img\_size=224, patch\_size=14, in\_channels=3, embed\_dim=640,  encoder\_layers=7, decoder\_layers=7, nhead=8, num\_classes=7)  model = model.to(device)  # Загружаем сохранённые веса  model.load\_state\_dict(torch.load("transformer\_model2.pth", map\_location=device))  model.eval()  # Определяем преобразования, как в тестовой выборке  transform = transforms.Compose([  transforms.Resize((224, 224)),  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225])  ])  # Список путей к изображениям для инференса (замените на свои пути)  image\_paths = [  r"C:\Users\Litvein\Desktop\images\audi.jpg",  r"C:\Users\Litvein\Desktop\images\dog.jpg",  r"C:\Users\Litvein\Desktop\images\cat.jpg",  r"C:\Users\Litvein\Desktop\images\ruvi4.jpg",  r"C:\Users\Litvein\Desktop\images\kreslo.jpg",  r"C:\Users\Litvein\Desktop\images\mazda5.jpg",  r"C:\Users\Litvein\Desktop\images\mazda6.jpg",  r"C:\Users\Litvein\Desktop\images\matya\_bmw4.jpg",  r"C:\Users\Litvein\Desktop\images\matya\_bmw.jpg",  r"C:\Users\Litvein\Desktop\images\ruvi3.jpg",  r"C:\Users\Litvein\Desktop\images\ruvi5.jpg",  r"C:\Users\Litvein\Desktop\images\chief.jpg",  r"C:\Users\Litvein\Desktop\images\ruvi.jpg",  r"C:\Users\Litvein\Desktop\images\fridgee.jpg",  r"C:\Users\Litvein\Desktop\images\motorcycle.jpg",  r"C:\Users\Litvein\Desktop\images\chair.jpg",  r"C:\Users\Litvein\Desktop\images\tosya.jpg",  r"C:\Users\Litvein\Desktop\images\bike.jpg"  ]  # Список имен классов (порядок должен соответствовать обучению)  classes = ["bike", "cars", "cats", "chair", "dogs", "fridge", "motorbike"]  for img\_path in image\_paths:  if not os.path.exists(img\_path):  print(f"Image not found: {img\_path}")  continue  # Загружаем изображение и конвертируем в RGB (если необходимо)  image = Image.open(img\_path).convert("RGB")  input\_tensor = transform(image).unsqueeze(0).to(device)  with torch.no\_grad():  outputs = model(input\_tensor)  \_, predicted = torch.max(outputs, 1)  pred\_idx = predicted.item()  pred\_class = classes[pred\_idx] if pred\_idx < len(classes) else str(pred\_idx)  print(f"Image: {img\_path}") |

*Окончание листинга У*

|  |
| --- |
| print(f"Predicted class index: {pred\_idx}")  print(f"Predicted class: {pred\_class}")  print("-" \* 40)  # Отображаем изображение с предсказанным классом  plt.figure()  plt.imshow(image)  plt.title(f"Трансформер, предсказано: {pred\_class}")  plt.axis("off")  plt.show()  if \_\_name\_\_ == "\_\_main\_\_":  main() |