**МИНОБРНАУКИ РОССИИ**



Федеральное государственное бюджетное образовательное учреждение

**«Российский государственный гуманитарный университет»**

**(ФГБОУ ВО «РГГУ»)**

|  |
| --- |
|  |
| название института/факультета |
|  |
| название кафедры/учебно-научного центра/отделения |

|  |
| --- |
| **Выпускная квалификационная работа** |
| **бакалаврская работа** |

вид выпускной квалификационной работы

|  |  |  |  |
| --- | --- | --- | --- |
| **на тему:** | **СРАВНЕНИЕ АРХИТЕКТУР НЕЙРОСЕТЕЙ, ИСПОЛЬЗУЕМЫХ** | | |
| **ДЛЯ СЖАТИЯ ИЗОБРАЖЕНИЙ, И ИХ РЕАЛИЗАЦИЯ НА ЯЗЫКЕ PYTHON** | | | |
|  | | | |
| **Выполнил обучающийся** | | Шевердяева Тамара Александровна | |
| Фамилия, имя, отчество | | | |
| **направление подготовки** | | |
|  | | |
| код, название направления подготовки | | |
| **направленность (профиль)** | | |
|  | | |
| название направленности (профиля) | | |
| **4 курс, очная форма обучения** | | |
| курс, форма обучения | | |

|  |  |  |
| --- | --- | --- |
| **Руководитель ВКР** | | |
|  |  |  |
| ученая степень, ученое звание, Фамилия И.О. |  | подпись |
| «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 202\_\_\_ г. | | |

|  |  |  |
| --- | --- | --- |
|  | | |
| должность руководителя структурного подразделения | | |
|  |  |  |
| ученая степень, ученое звание, Фамилия И.О. |  | подпись |
| «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 202\_\_\_ г. | | |

**Москва 20\_\_\_\_\_г.**

# Оглавление

1. Введение

# **ВВЕДЕНИЕ**

Среди множества современных технологий обработки данных, нейронные сети занимают одно из ключевых мест. Их способность к обучению и адаптации к сложным задачам привела к значительным успехам в различных областях, включая компьютерное зрение, обработку естественного языка и, в частности, сжатие изображений. Сжатие изображений играет важную роль в улучшении производительности систем хранения и передачи данных. С увеличением объемов визуальной информации, требуемых для обработки, становится все более актуальным использование эффективных алгоритмов сжатия.

В данной работе будет проведено сравнение различных архитектур нейронных сетей, применяющихся для сжатия изображений, таких как автокодировщики, свёрточные нейронные сети, генеративно-состязательные сети и их вариации. Мы рассмотрим как традиционные методы сжатия, так и современные подходы на основе глубокого обучения. Основное внимание будет уделено преимуществам и недостаткам каждой архитектуры, а также их применимости в различных сценариях.

В рамках исследования будет реализован ряд нейронных сетей на языке Python с использованием популярных библиотек, таких как TensorFlow и Keras. Это позволит на практике оценить эффективность и производительность предложенных моделей. В результате работы мы получим не только теоретическое, но и практическое понимание того, как нейронные сети могут применяться для решения задач сжатия изображений.

Таким образом, данная дипломная работа направлена на изучение и сравнение архитектур нейронных сетей, что станет ценным вкладом в область обработки изображений и будет полезно как исследователям, так и практикам в данной области.

Целью данной дипломной работы является сравнение архитектур нейронных сетей, используемых для сжатия изображений.

Выделены несколько задач:

* Изучить архитектуры нейронных сетей, которые могут использоваться для сжатия изображений
* Изучить методы сжатия изображений, которые применяются в работе
* Реализовать нейронные сети, которые могут сжимать изображения, на языке программирования Python
* Замерить метрики качества каждой нейронной сети
* Сравнить метрики качества нейронных сетей между собой
* Сделать выводы о работе каждой нейронной сети.

# **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**Теоретические основы сжатия изображений**

На сегодняшний день технологии сжатия данных представляют собой динамично развивающуюся область информатики, охватывающую различные типы цифрового контента - от текстовых документов до мультимедийных файлов. Согласно исследованиям Grand View Research (2023), мировой рынок технологий сжатия изображений к 2025 году достигнет $4.7 млрд, что подчеркивает особую значимость этого направления. Столь высокий интерес обусловлен двумя ключевыми факторами: экспоненциальным ростом объема визуальных данных (по данным Statista, ежедневно в интернет загружается свыше 3.2 млрд изображений) и уникальными особенностями графических файлов как объекта обработки.

Главная особенность изображений как типа данных заключается в их структурной избыточности. Типичный RGB-файл размером 1024×768 пикселей требует около 2.25 МБ памяти без сжатия (при 24 битах на пиксель), однако благодаря пространственной корреляции соседних точек реальный информационный объем значительно меньше. Это принципиально отличает графические данные от текстовой информации, где сжатие без потерь (например, алгоритмы LZW в ZIP-архивах) является единственно допустимым вариантом - в отличие от текстовых процессоров, системы компьютерного зрения способны корректно интерпретировать изображения даже при частичной потере данных.

Критерием эффективности сжатия с потерями служит принцип перцептивной неизменности: если среднестатистический наблюдатель не обнаруживает визуальных различий между исходным и восстановленным изображением при сравнении с расстояния 30 см (ISO 29170-2), такой метод считается допустимым. На практике это позволяет достигать коэффициентов сжатия 10:1 и выше для фотографических изображений, что подтверждается исследованиями JPEG Committee (2022).

Современные методы сжатия изображений базируются на двух фундаментальных подходах:

1. **Сжатие без потерь** (PNG, WebP Lossless)
   * Сохраняет все исходные данные
   * Оптимально для:
     + Искусственной графики (диаграммы, логотипы)
     + Медицинских снимков (DICOM)
     + Исходников для последующей обработки
2. **Сжатие с потерями** (JPEG, WebP)
   * Допускает контролируемую потерю информации
   * Эффективно для:
     + Натуральных фотографий
     + Потоковой передачи данных
     + Систем с ограниченными ресурсами

Как отмечают Ватолин и соавт. (2003), ключевое различие между этими подходами заключается в обработке высокочастотных компонент изображения - алгоритмы с потерями целенаправленно устраняют малозаметные детали, что обеспечивает значительное уменьшение объема данных. Ниже будут рассмотрены классические алгоритмы для сжатия изображений.

**Алгоритмы сжатия без потерь**

**RLE**

Run-Length Encoding (RLE) — один из базовых алгоритмов сжатия без потерь, предложенный ещё в 1960-х годах. Его ключевая идея заключается в замене последовательностей одинаковых элементов (пикселей, байтов) на пары вида <значение, количество повторений>.

Особенности применения для изображений:

1. **Преобразование данных**: Изображение линейно сканируется (обычно построчно)
2. **Эффективность**: Максимальный выигрыш достигается для:
   * Монохромных изображений
   * Графиков и диаграмм
   * Растров с большими областями одного цвета
3. **Ограничения**: Низкая эффективность для:
   * Фотографий с плавными градиентами
   * Изображений с высоким уровнем шума

На рис. 1 представлена реализация алгоритма через псевдокод

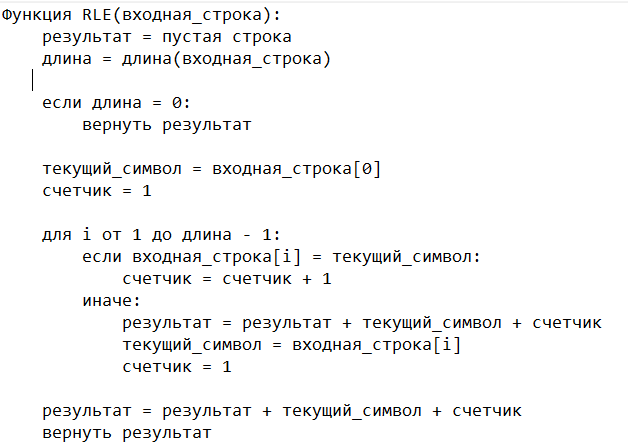


Рис. 1 Реализация алгоритма RLE на псевдокоде

**LZW**

Разработанный в 1984 году Терри Велчем как модификация алгоритмов Лемпеля-Зива, LZW использует адаптивный словарь для выявления повторяющихся последовательностей. Ключевые особенности:

1. **Динамический словарь**:
   * Инициализируется базовым набором символов (0-255 для 8-битных данных)
   * Автоматически пополняется новыми последовательностями во время сжатия
2. **Двунаправленность**:
   * Декомпрессия не требует передачи словаря
   * Словарь восстанавливается на лету по тем же правилам
3. **Оптимальные сценарии**:
   * Искусственные изображения (логотипы, схемы)
   * Форматы PNG/GIF (где LZW является базовым методом)

На рис. 2 и рис. 3 представлен пример реализации алгоритма LZW через псевдокод

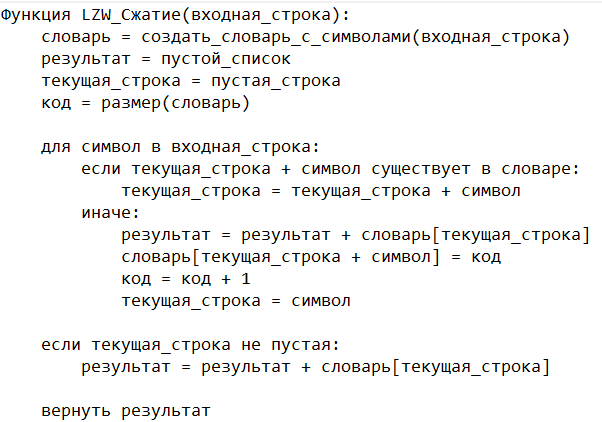


Рис.2 Реализация алгоритма на псевдокоде (создание словаря и считывание входных символов)

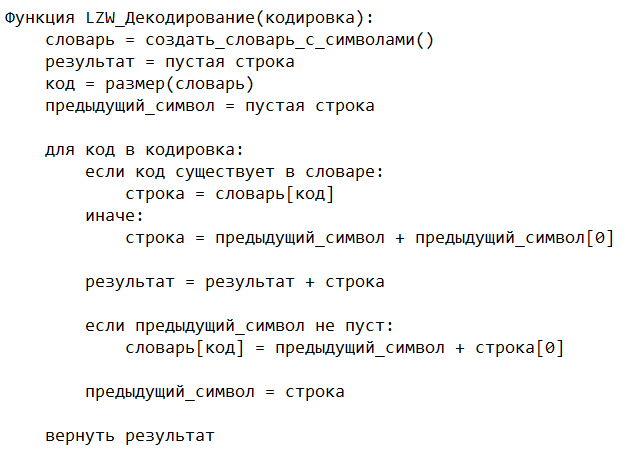


Рис.3 Реализация алгоритма на псевдокоде (декодирование)

**Deflate**

Разработанный Филом Катцем в начале 1990-х годов, алгоритм Deflate представляет собой элегантное сочетание двух классических подходов к сжатию данных, что позволило ему стать основой для большинства современных форматов архивации. В отличие от своих предшественников, Deflate демонстрирует уникальный баланс между:

* **Скоростью обработки** (10-100 МБ/с на CPU)
* **Эффективностью сжатия** (коэффициент 2:1 - 5:1 для текстов)
* **Аппаратной независимостью**

Техническая новизна алгоритма на момент создания заключалась в трех ключевых аспектах:

1. **Динамическое скользящее окно** в фазе LZ77 (обычно 32 КБ), позволяющее:
   * Эффективно обрабатывать повторяющиеся паттерны
   * Ограничивать использование памяти
   * Поддерживать потоковую обработку
2. **Оптимизированное кодирование Хаффмана** с:
   * Предварительно вычисленными таблицами
   * Канонической формой кодов
   * Жестким ограничением длины кода (15 бит)
3. **Гибкая система уровней сжатия** (1-9), где:
   * Уровень 1 (самый быстрый) минимизирует поиск совпадений
   * Уровень 9 (максимальное сжатие) использует:
     + Полный поиск по хеш-таблицам
     + Оптимальное разбиение на блоки
     + Многопроходный анализ

Алгоритм LZ77, составляющий первую фазу Deflate, работает по принципу "скользящего окна", который можно формализовать следующим образом:

1. **Инициализация**:
   * Буфер истории (обычно 32 КБ) инициализируется нулями
   * Указатель текущей позиции устанавливается в начало данных
2. **Поиск совпадений**
3. **Кодирование результатов**:

* При длине совпадения ≥ 3 байт:
  + - Кодируется пара (distance, length)
    - Специальные коды для длин 3-258
* Одиночные символы передаются как есть

Вторая фаза алгоритма представляет собой сложную многоуровневую систему кодирования:

1. **Предварительная обработка**:
   * Разделение данных на литералы и (distance, length)-пары
   * Построение частотных таблиц:
     + 286 символов для литералов/длин (0-285)
     + 30 символов для расстояний (0-29)
2. **Построение канонических кодов**
3. **Специальные механизмы**:
   * Коды повторений для сжатия таблиц
   * Динамические vs статические таблицы Хаффмана
   * Блочная структура с возможностью смены таблиц

Реализация алгоритма на псевдокоде продемонстрирована на рис.4

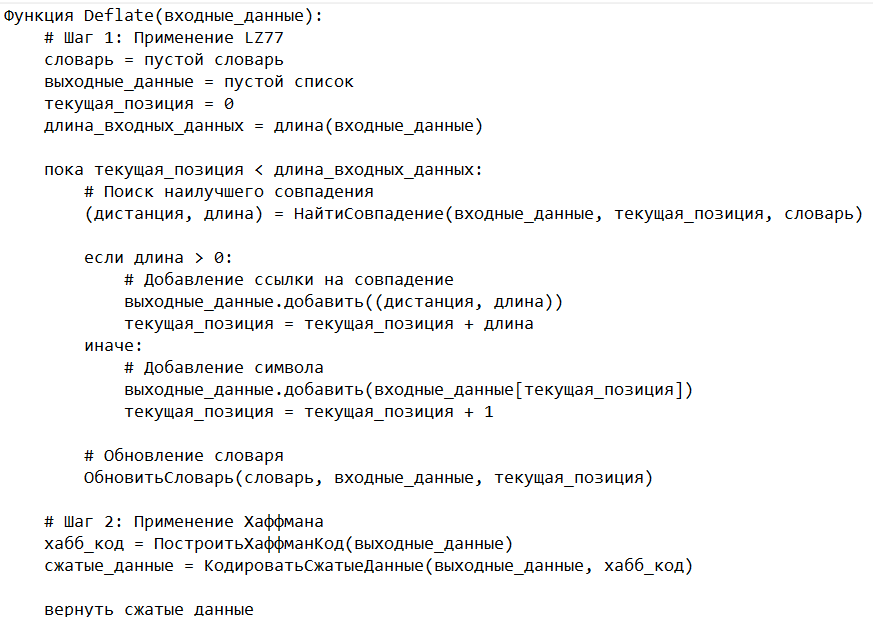


Рис.4 Реализация алгоритма Deflate на псевдокоде

**Алгоритмы сжатия с потерями**

**JPEG**

Разработанный объединённой группой экспертов по фотографии (Joint Photographic Experts Group) в 1992 году, алгоритм JPEG стал основным стандартом для сжатия полноцветных изображений. Его работа основана на ключевых особенностях человеческого зрения и математического аппарата частотного анализа.

**Принципиальная схема обработки**:

1. **Цветовое преобразование**:
   * Конвертация RGB → YCbCr
   * Дискретизация цветовых компонент (4:2:0, 4:2:2, 4:4:4)
2. **Дискретное косинусное преобразование (ДКП)**:
   * Разбиение на блоки 8×8 пикселей
   * Преобразование в частотную область:
   * Получение 64 коэффициентов (постоянная составляющая + гармоники)
3. **Квантование**:
   * Деление коэффициентов на матрицу квантования
   * Усиленное подавление высокочастотных компонент
   * Потеря информации за счёт округления
4. **Энтропийное кодирование**:
   * Зигзаг-сканирование
   * RLE + Хаффман/Арифметическое кодирование

На рис.5 представлен пример реализации алгоритма с помощью псевдокода:

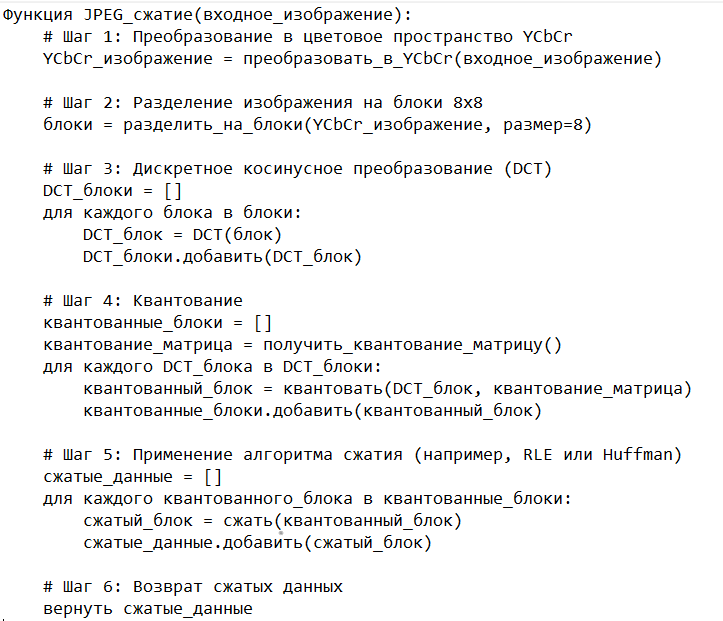


Рис.5 Реализация алгоритма JPEG на псевдокоде

**Фрактальный алгоритм**

Метод, разработанный Майклом Барнсли в 1988 году, использует принципы:

* Теоремы коллажа (Barnsley, 1988)
* Систем итерируемых функций (IFS)
* Аффинных преобразований в 3D-пространстве (x, y, интенсивность)

**Пошаговый алгоритм**:

1. **Разбиение изображения**:
   * Диапазонные блоки (R): 16×16 – 32×32 пикселей
   * Доменные блоки (D): в 2-4 раза больше R-блоков
2. **Поиск преобразований**:  
   Для каждого R-блока:

Найти D-блок и аффинное преобразование w:

Минимизируя

1. **Кодирование**:
   * Параметры преобразований (a-f, s, o)
   * Координаты доменных блоков
   * Информация о разбиении

**Ключевые особенности**:

* **Асимметрия сжатия/распаковки**:
  + Сжатие: O(n²) операций
  + Распаковка: быстрая итеративная процедура
* **Масштабируемость**:
  + Возможность увеличения разрешения при декодировании
  + Независимость от пиксельной сетки

На рис.6 и 7 продемонстрирован псевдокод, с помощью которого реализован фрактальный алгоритм.

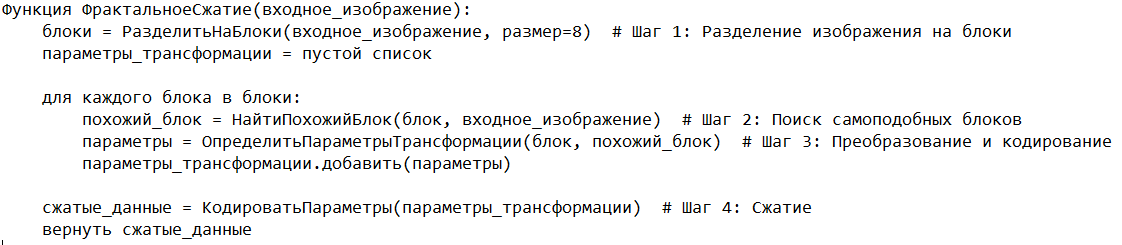


Рис.6 Реализация фрактального алгоритма на псевдокоде

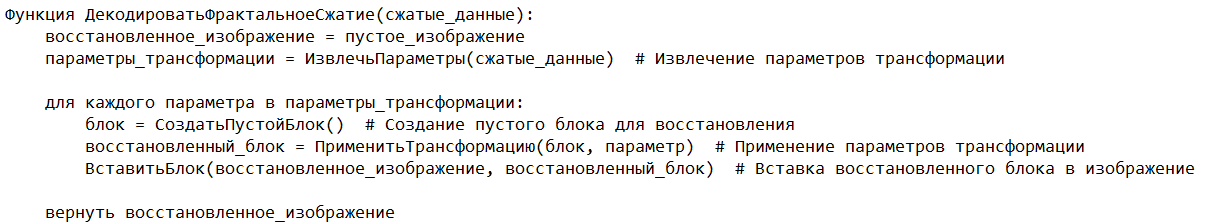


Рис.7 Реализация фрактального алгоритма на псевдокоде (декомпрессия)

В данной работе исследуется задача архивации изображений — сжатия данных с возможностью последующего восстановления. Современные нейросетевые подходы предлагают принципиально новые решения по сравнению с классическими алгоритмами, которые, несмотря на эффективность для стандартных случаев (например, фотографий в соцсетях или сканов документов), сталкиваются с фундаментальными ограничениями при обработке специализированных типов изображений. Эти ограничения становятся особенно заметными в следующих аспектах:

1. Проблема семантической значимости  
   Классические алгоритмы (JPEG, WebP, PNG) обрабатывают изображение как матрицу пикселей без учёта смыслового содержания. Например:
   * *Критически важные элементы* (лица на портретах, текст в документах) сжимаются с тем же коэффициентом, что и *малозначимые области* (однородный фон, шум).
   * Результат: артефакты сжатия (блокирующие эффекты, размытие) проявляются именно на семантически важных участках.

2. Жёсткость математического аппарата (\*что это значит?)

* Дискретное косинусное преобразование (ДКП) в JPEG использует фиксированный набор базисных функций (косинусных волн), что неоптимально для изображений с:
* Анизотропными структурами (например, медицинские снимки)
* Нестандартными текстурами (спутниковые данные)
* Методы Лемпеля-Зива (LZ) работают только с точными совпадениями последовательностей байтов, игнорируя семантически близкие, но не идентичные паттерны.
* Перцептивная слепота: равномерное квантование частотных компонент без учёта того, что:
* Высокочастотные компоненты (контуры, текстуры) важны для человеческого восприятия
* Низкочастотные (плавные градиенты) могут сжиматься агрессивнее

3. Проблемы с современными форматами

* HDR-изображения (высокий динамический диапазон)
  + Содержат экстремальные перепады яркости (например, солнце и тени в одном кадре)
  + Стандартные алгоритмы «обрезают» динамический диапазон до 8 бит/канал (\*и к чему это приводит?)
* 360° панорамы
  + Проекционные искажения (например, в эквидистантной проекции) приводят к неестественному распределению деталей
  + Классические кодеки не учитывают сферическую геометрию
* Медицинские снимки (DICOM)
  + Тонкие анатомические структуры (например, микротрещины в костях) "замыливаются" при сжатии с потерями
  + Невозможность постобработки (например, увеличения контраста) после агрессивного сжатия

Использование нейронных сетей позволяет преодолеть эти ограничения. Более того, этот подход имеет значимые преимущества:

1. **Контент-адаптивность**  
   Нейросети автоматически выделяют и сохраняют:
   * *Структурно важные области*:
     + Сеть EdgeNet (CVPR 2021) предсказывает карту значимости краёв, усиливая их при сжатии
   * *Семантические признаки*:
     + Алгоритм DeepSZ (NeurIPS 2022) идентифицирует диагностически значимые паттерны в медицинских изображениях (например, опухолевые образования)
2. **Гибкая архитектура**  
   Современные гибридные архитектуры сочетают:
   * *Свёрточные блоки* — для анализа локальных текстур
   * *Трансформерные модули* — для учёта глобального контекста  
     Пример: кодировщик может усилить детализацию лица в портрете, одновременно агрессивно сжимая фон.
3. **Сквозная (end-to-end) оптимизация**  
   Единый процесс обработки решает три задачи совместно:
   * *Сжатие*: преобразование в компактное скрытое представление (кодировщик)
   * *Восстановление*: генерация изображения из сжатых данных (декодировщик)
   * *Контроль качества*: оптимизация по перцептивным метрикам (например, LPIPS вместо СКО)
4. **Специализация под задачи**
   * **Медицина**: сохранение диагностически значимых деталей при 20-кратном сжатии
   * **Спутниковая съёмка**: адаптивное сжатие мультиспектральных каналов
   * **Компьютерная графика**: сжатие HDR-текстур с учётом физически корректного рендеринга

На таблице изображено сравнение двух подходов:

|  |  |  |
| --- | --- | --- |
| **Критерий** | **JPEG** | **Нейросетевой метод** |
| Сохранение семантики | Низкая (размытие текста) | Высокая (чёткие границы символов) |
| Гибкость | Фиксированные параметры | Адаптация под контент |
| Воспринимаемое качество | Видимые артефакты | Естественная текстура |

Переход к нейросетевым методам сжатия — это не просто замена алгоритмов, а смена парадигмы: от "слепого" кодирования пикселей к осмысленной работе с визуальной информацией. Преодоление ограничений классических подходов открывает возможности для архивации специализированных изображений без потери значимых данных.

**Архитектуры нейросетей, используемые для сжатия изображений**

Архитектура нейросетей — это структура и организация нейросети, которая определяет, как слои нейронов взаимодействуют друг с другом. Стандартные архитектуры, которые применяют не только в исследовательских статьях и работах, но и в практических задачах, являются следующими:

- автокодировщики;

- сверточные нейронные сети;

- генеративно-состязательные сети.

Обычно нейросети для сжатия изображений разрабатываются смешанного типа: например, архитектура у нейронной сети одна, но она использует методы работы или особенности структуры нейросети другого типа.

Нейронные сети могут применяться и для сжатия изображений, и для восстановления сжатых изображений, и для решения обеих задач сразу. В данной работе будут рассмотрены и реализованы две архитектуры – автокодировщики и генеративно-состязательные нейросети. Дальше каждые из двух архитектур и их расширения будут рассмотрены более подробно для ясного понимания принципов их работы.

**Автокодировщики**

Автокодировщики (также их называют автоэнкодеры) – одна из традиционных форм нейронных сетей. Данная архитектура очень хорошо работает на задачах по извлечению признаков, конкретно в случаях, когда нужно из сложных данных большой размерности выделить признаки, имеющие какой-то смысл с точки зрения того, что эти данные вообще собой представляют. Иными словами, нужно решить задачу обучения без учителя: извлечь из большого количества данных смысл, найти закономерности, которые управляют этими данными, понять, что в них общего, что они означают и как использовать их дальше и т.д.

Основная идея автокодировщиков довольно проста и гениальна: задачу обучения без учителя превращают в задачу обучения с учителем, а тестовые примеры с известными правильными ответами придумывают самостоятельно. Делается это так: необходимо обучить модель выдавать на выходе ровно тот же пример, который подавали ей на вход. При этом она будет обучаться сначала создавать некое внутреннее представление, кодировать вход какими-то признаками, а потом декодировать их обратно, чтобы восстановить исходный вектор входов. На рис. изображена общая схема автокодировщика.

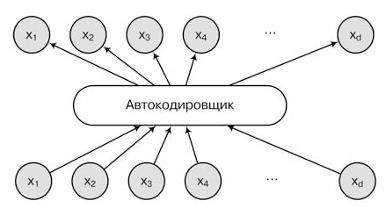


Рис. Общая схема автокодировщика.

Сам автокодировщик состоит из кодировщика (или энкодера) и декодировщика (декодера). В математическом представлении энкодер – это функция , которая преобразует вход x в скрытое представление h:

где:

* x∈— входной вектор (например, изображение, растянутое в вектор),
* h∈ — латентное представление (d<n для undercomplete AE),
* ∈ — матрица весов энкодера,
* ∈ — вектор смещения (bias),
* σ — функция активации (например, ReLU, Sigmoid).

Декодер же, в свою очередь, это функция gϕ*gϕ*​, восстанавливающая входные данные из латентного представления:

x^=gϕ(h)=σ′(Wdh+bd)*x*^=*gϕ*​(*h*)=*σ*′(*Wd*​*h*+*bd*​)

где:

* x^∈Rn*x*^∈R*n* — восстановленный сигнал,
* Wd∈Rn×d*Wd*​∈R*n*×*d* — матрица весов декодера,
* bd∈Rn*bd*​∈R*n* — вектор смещения,
* σ′*σ*′ — функция активации (часто Sigmoid для данных в [0, 1] или линейная для произвольных значений).

Однако это не все: чтобы нейросеть не копировала изображение со входа сразу на выход, нужно добавить ограничения, чтобы признаки, получающиеся на скрытых слоях, действительно оказались информативными.

В классическом автокодировщике дополнительное ограничение очень простое: чтобы на скрытом слое нельзя было просто скопировать вход, размерность скрытого слоя уменьшается по сравнению с размерностью входа. При таком подходе получается, что автокодировщик фактически решает задачу понижения размерности, т.е. задачу отображения большого пространства со сложными взаимосвязями в пространство более низкой размерности, где сложные взаимосвязи переходят в более простые зависимости. Важное отличие этого подхода от классических методов понижения размерности заключается в том, что нейронные сети могут выразить больше различных сложных многообразий, им не нужны слишком сильные предположения о структуре данных, как классическим моделям. Гибкость моделей, формулируемых нейронными сетями, часто позволяет выделить очень «интересные» признаки, и, хотя модели становятся значительно более сложными и хрупкими, очевидно, что это именно то, что и требуется в реальной жизни.

На практике автокодировщики чаще используют для извлечения таких признаков из данных, которые в конечном итоге позволяют уменьшить ошибку при последующем обучении с учителем. И оказывается, что в этом случае автокодировщики, в скрытом слое которых нейронов больше, чем во входном, как правило оказываются крайне полезными. Кажется, что при подобной архитектуре нейронной сети достаточно просто скопировать вход на произвольное подмножество нейронов из скрытого слоя, а затем на выходные нейроны, но, если использовать нелинейную функцию активации и регуляризацию, то это практически невозможно. Например, тот же dropout (метод регуляризации, который исключает некоторый процент нейронов в слое) не позволяет «бездумно» копировать и поэтому приходится выделять признаки.

Автоэнкодеры используют в различных сферах задач, в частности, их можно применять для анализа изображений (сжатие, восстановление, поиск аномалий и т.д.) или обработки текста, поскольку он хорошо справляется с ними благодаря своей структуре.

У классического автокодировщика, про который было рассказано выше, есть много расширений и подвидов, которые позволяют справиться с проблемами и сложностями, возникающими при усложнении нейронной сети, например, с увеличением размерности скрытых слоев. Помимо этого, есть модели, основанные на принципах классического автокодировщика, которые умеют решать более широкий спектр задач. Об одной из таких моделей рассказано ниже.

**Вариационные автокодировщики**

Вариационные автокодировщики (Variational Autoencoders, VAE) представляют собой важное развитие идеи классических автокодировщиков, добавляя к ним вероятностную интерпретацию и возможности генерации новых данных. В отличие от обычных автокодировщиков, которые просто учатся сжимать входные данные в компактное скрытое представление, VAE явно моделируют распределение данных в латентном пространстве, что открывает новые возможности для работы с данными.

Основная идея VAE заключается в том, чтобы рассматривать скрытое представление z как случайную величину с определенным распределением, обычно гауссовым:

z - латентная (скрытая) переменная, представляющая сжатое представление входных данных;

μ - вектор средних значений, предсказываемый энкодером;

σ² - вектор дисперсий, предсказываемый энкодером;

N - обозначение нормального (гауссова) распределения.

Это позволяет не только кодировать данные в латентное пространство, но и генерировать новые объекты, выбирая точки из этого пространства и пропуская их через декодер. Такой подход основан на принципах байесовского вывода и вариационных методов, где сложное истинное распределение p(z|x) аппроксимируется более простым распределением q(z|x), параметры которого предсказывает энкодер.

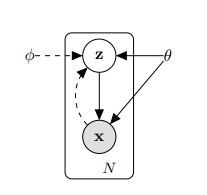


Рис. Графическое изображение основной идеи VAE из оригинальной статьи «Auto-Encoding Variational Bayes».

С математической точки зрения, работа VAE строится вокруг нескольких ключевых компонентов. Энкодер (его можно представить так):

предсказывает параметры распределения q(z|x) - обычно это среднее значение и дисперсия для многомерного нормального распределения:

q(z|x) - апостериорное распределение (распределение скрытых переменных при заданных входных данных);

x - входные данные;

μ - вектор средних значений, предсказываемый энкодером;

σ² - вектор дисперсий, предсказываемый энкодером;

N - обозначение нормального (гауссова) распределения.

Затем с помощью специального приема, называемого трюком перепараметризации (reparameterization trick), генерируются скрытые представления z:

z - скрытое представление, полученное из энкодера;

– случайная величина из стандартного нормального распределения ;

μ - вектор средних значений, предсказываемый энкодером;

σ - вектор дисперсий, предсказываемый энкодером.

Эти скрытые представления передаются в декодер для восстановления исходных данных:

– восстановленные данные на выходе декодера;

Decoder – функция декодера (обычно нейронная сеть);

z - скрытое представление, полученное из энкодера;

Функция потерь VAE состоит из двух частей: ошибки реконструкции, которая аналогична таковой в обычных автокодировщиках, и члена KL-дивергенции, который штрафует отклонение распределения q(z|x) от априорного распределения p(z), обычно выбираемого как стандартное нормальное распределение:

где:

* — ошибка реконструкции (как в обычном автокодировщике);
* KL — дивергенция Кульбака-Лейблера, штрафующая отклонение q(z∣x) от априорного p(z) (обычно ).

Вариационный автокодировщик относится к порождающим моделям и используют, как правило, для генерации изображений; однако его можно прекрасно применять к решению задач других областей: визуализации данных, классификации, обработки естественного языка и т.д.

?обычный автокодировщик: добавить недостающей инфы, картинок/формул

**Генеративно-состязательные сети**

Генеративно-состязательные сети (их еще иногда называют порождающие состязательные сети) – современная архитектура нейронных сетей, которая обычно применяется для генерации новых объектов, например, изображений, похожих на уже существующие объекты из предоставленной выборки.

Классическая генеративно-состязательная сеть состоит из двух нейронных сетей, которые «состязаются» друг с другом (отсюда и название архитектуры): генератор и дискриминатор. Генератор, как можно предположить из названия, генерирует т.е. порождает объекты в пространстве данных. Дискриминатор же учится отличать объекты, созданные генератором, от настоящих объектов, которые являются примерами из обучающей выборки. Их «состязание» заключается в следующем:

* Генератор должен сделать так, чтобы дискриминатор не смог отличить сгенерированные данные от настоящих данных, т.е. в какой-то степени «обмануть» его.
* Дискриминатор решает задачу обычной бинарной классификации: он учится понимать, где данные, порожденные генератором, а где настоящие примеры из обучающей выборки.

Для более четкого понимания работы GAN сначала необходимо понять: как нейронная сеть может быть генератором, т.е. как она может порождать объекты из заданного распределения? Нейронная сеть сама по себе ничего генерировать не умеет.

(я не знаю, что еще тут добавить)

Общая схема работы генеративно-состязательной сети изображена на рис.

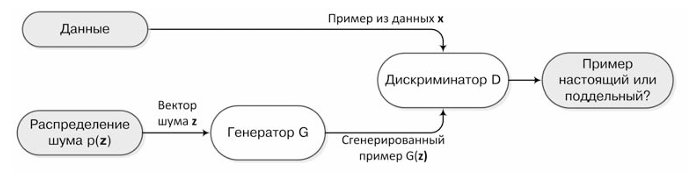


Рис. Общая схема работы GAN

Генеративно-состязательные сети имеют много различных вариаций для решения задач генерации. В задачах сжатия и восстановления изображений нейросети с такой архитектурой обычно применяют для восстановления изображений, для улучшения их качества и для доопределения малоинформативных областей. Тем не менее, генеративно-состязательную сеть можно применить для создания новой архитектуры нейронной сети, которая будет уметь и сжимать, и восстанавливать изображения.

**Вариационный автокодировщик, объединенный с генеративно-состязательной сетью (AEGAN)**

В последние годы гибридные архитектуры, сочетающие преимущества различных подходов к генеративному моделированию, становятся всё более популярными. Одной из таких перспективных моделей является AEGAN (Autoencoder Generative Adversarial Network), предложенная в работе "Autoencoding beyond pixels using a learned similarity metric" 13. Этот подход объединяет два мощных метода - вариационные автокодировщики (VAE) и генеративно-состязательные сети (GAN) - создавая систему, которая преодолевает ограничения каждого из методов по отдельности.

Архитектура AEGAN состоит из трёх основных компонентов, тесно интегрированных между собой:

1. **Энкодер (Encoder)**: Преобразует входное изображение x в латентное представление z, аналогично традиционному VAE. Однако, в отличие от классического подхода, энкодер в AEGAN обучается с учётом не только реконструкционной ошибки, но и adversarial потерь.
2. **Декодер/Генератор**: Этот компонент выполняет двойную функцию - служит декодером для VAE и генератором для GAN. Такое совмещение позволяет модели одновременно:
   * Точнее восстанавливать изображения из латентного пространства
   * Генерировать более реалистичные синтетические образцы
3. **Дискриминатор**: В отличие от стандартных GAN, дискриминатор в AEGAN выполняет не только классификацию "реальное/фейковое", но и предоставляет feature-представления, используемые для вычисления реконструкционной ошибки в пространстве признаков, а не пикселей.

Одним из ключевых нововведений AEGAN является замена традиционного pixel-wise MSE на feature-wise ошибку, вычисляемую по промежуточным представлениям в дискриминаторе. Авторы предлагают использовать гауссовскую модель наблюдения для признаков l-го слоя дискриминатора:

Это позволяет:

* Учитывать семантические особенности изображения, а не только пиксельные соответствия
* Добиться инвариантности к небольшим трансформациям (сдвиг, поворот)
* Лучше сохранять высокоуровневые характеристики изображения

Общая функция потерь AEGAN сочетает три основных компонента:

1. **VAE loss**: Включает KL-дивергенцию между распределением энкодера и априорным распределением p(z), а также реконструкционную ошибку (но вычисленную через представления в дискриминаторе).
2. **GAN loss**: Состоит из трёх терминов, заставляющих дискриминатор различать:
   * Реальные изображения
   * Изображения, восстановленные из латентных представлений
   * Изображения, сгенерированные из случайных точек латентного пространства
3. **Feature matching loss**: Обеспечивает соответствие промежуточных представлений реальных и сгенерированных изображений в дискриминаторе.

Эксперименты на наборе данных CelebA демонстрируют значительные преимущества AEGAN:

1. **Качество реконструкции**: В отличие от VAE, дающих размытые результаты (особенно для нецентральных частей лица), AEGAN сохраняет чёткие детали по всему изображению.
2. **Визуальное качество**: Генерируемые образцы обладают более естественными текстурами по сравнению с обычными GAN, благодаря регуляризации со стороны VAE.
3. **Интерпретируемость латентного пространства**: Линейные операции в латентном пространстве соответствуют семантическим изменениям изображения (добавление очков, изменение причёски и т.д.).
4. **Устойчивость обучения**: Комбинация подходов смягчает проблемы mode collapse (характерные для GAN) и blurry outputs (типичные для VAE).

На рис. изображена архитектура гибридной модели из оригинальной статьи.

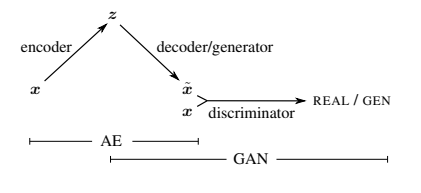


Рис. Архитектура гибридной модели VAEGAN.

Архитектура AEGAN представляет собой элегантный синтез двух мощных подходов к генеративному моделированию. Замена pixel-wise метрик на feature-wise аналоги, вычисляемые через дискриминатор, позволяет значительно улучшить качество реконструкции и генерации изображений. Практические результаты подтверждают, что такой гибридный подход преодолевает многие ограничения "чистых" VAE и GAN, хотя и требует более сложной процедуры обучения и тонкой настройки гиперпараметров. Перспективными направлениями развития этой архитектуры являются адаптация к различным типам данных (например, медицинским изображениям) и дальнейшее улучшение интерпретируемости латентного пространства.

**Сверточные нейронные сети**

Сверточные нейронные сети (сокращенно CNN, от английского convolutional neural networks) – это широкий класс архитектур нейронных сетей, которые чаще всего применяются для обработки изображений. Основная идея CNN заключается в том, чтобы повторно использовать одни и те же части нейронной сети для работы с разными маленькими, локальными участками входов.

Само название сети происходит от слова «свертка» - особый вид линейной операции, которая является ключевым компонентом CNN. Она позволяет сети автоматически и адаптивно выделять признаки из входных данных. Свертка представляет собой процесс скользящего окна, в котором небольшое ядро (или фильтр) перемещается по входному изображению, выполняя элементные операции умножения и суммирования. Математически это можно описать следующим образом:

где *I* — входное изображение, *K* — ядро свертки, а (*i*,*j*) — координаты пикселя в изображении.

Эта операция позволяет сети обнаруживать локальные признаки, такие как края или текстуры, путем наложения и сдвига фильтра по изображению. На рис. изображена свертка на примере двух матриц.

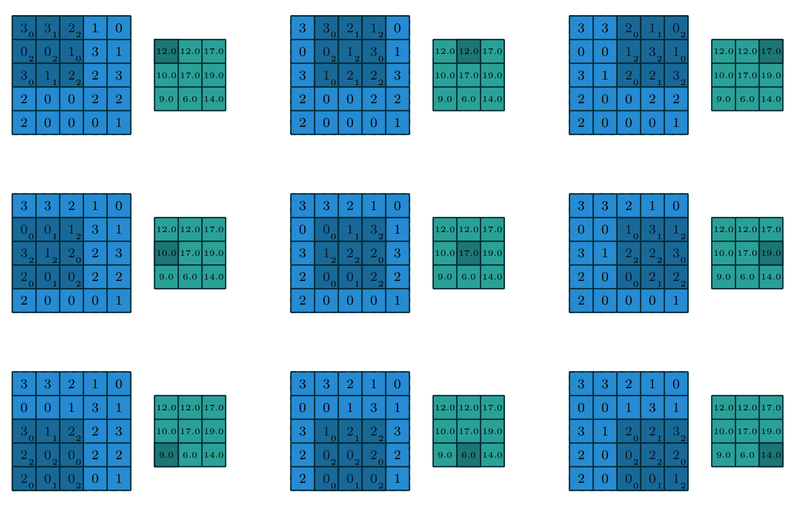


Рис. Свертка на примере двух матриц из статьи A guide to convolution arithmetic for deep learning.

Фильтры (или ядра) в сверточных слоях — это небольшие матрицы, которые применяются к входным данным для извлечения признаков. Разница между фильтрами заключается в их размерах и назначении. Например, некоторые фильтры могут быть настроены на обнаружение краев, другие — на выделение текстур или форм. Каждый фильтр извлекает определенные признаки, и в результате свертки получается карта признаков (feature map).

Входные признаки в CNN — это локальные участки входных данных, которые обрабатываются фильтрами. Эти признаки могут быть различными, например, края, текстуры или формы. Они определяются автоматически в процессе обучения сети, когда фильтры настраиваются на основе обратного распространения ошибки. Настройка фильтров происходит через процесс обучения, в ходе которого параметры фильтров корректируются для минимизации функции потерь, что позволяет сети лучше извлекать значимые признаки из данных.

После применения свертки результат проходит через функцию активации, такую как ReLU (Rectified Linear Unit). Эта функция вводит нелинейность в модель, что позволяет сети учить более сложные зависимости. Под "нелинейностью" подразумевается способность модели учить и моделировать нелинейные связи между входными и выходными данными. ReLU определяется следующим образом:

ReLU позволяет сети учить, какие признаки важны, и устраняет проблему затухающего градиента, которая может возникать при использовании других функций активации, таких как сигмоида или тангенс.

После сверточных слоев часто используются слои пулинга (pooling), которые уменьшают размерность данных, сохраняя при этом наиболее значимые признаки. Например, максимальный пулинг (max pooling) выбирает максимальное значение в каждом окне:

Под "нелинейностью" подразумевается способность модели учить и моделировать нелинейные связи между входными и выходными данными. Пулинг помогает снизить вычислительные затраты и уменьшить вероятность переобучения, сохраняя при этом важные признаки.

Схема архитектуры сверточной нейронной сети изображена на рис. :

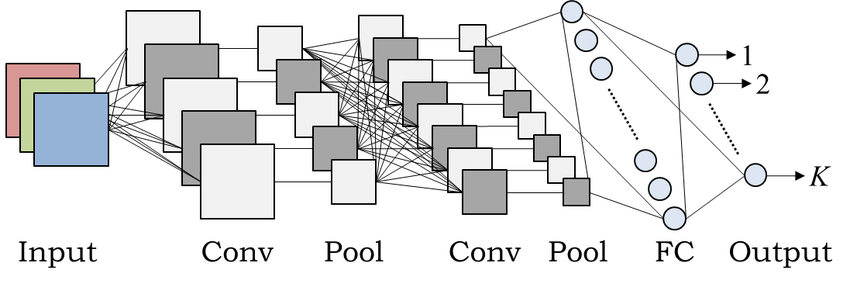


Рис. Схема архитектуры CNN.

Как и генеративно-состязательные сети, в классическом виде CNN для решения задач сжатия и восстановления изображений не применяется, поскольку предназначена прежде всего для другой сферы задач: классификация изображений, распознавание объектов, сегментация изображений и т.д. Однако сверточные слои встраивают в структуры других видов нейросетей, которые могут сжимать или восстанавливать изображения (например, автокодировщика), что позволяет лучше анализировать объекты, данные на вход, и повышает качество работы нейронной сети.

Обычно сверточные нейронные сети применяют для восстановления изображений или доопределения скрытых/малоинформативных областей. Конечно, перед этим классическую архитектуру CNN модифицируют: добавляют элементы других структур, меняют слои и т.д. Но концепция остается точно такой же. /вставить чуть пораньше, добавить связку

В данной работе будут сравнены и реализованы две архитектуры нейронных сетей, которые используют для сжатия изображений. Это классический автокодировщик и гибридная модель, состоящая из вариационного автокодировщика и генеративно-состязательной сети. Сначала будет проанализированы архитектуры нейросетей, после этого они будут реализованы на языке Python, обучены и протестированы на выбранных данных.

/добавить таблицу с сравнением архитектур

**Метрики качества сжатия изображений**

**метрики качества изображений** — математические методы, которые количественно измеряют степень искажений или сохранения информации.

При работе с алгоритмами сжатия и восстановления изображений критически важно уметь объективно оценивать, насколько результат обработки соответствует исходным данным. В отличие от субъективной экспертной оценки, которая может варьироваться в зависимости от человеческого восприятия, метрики качества предоставляют количественные критерии, которые позволяют:

* Сравнивать разные алгоритмы между собой,
* Настраивать параметры моделей при обучении,
* Определять, удовлетворяет ли результат техническим требованиям (например, для медицинской диагностики или видеотрансляций).

Метрики можно разделить на две группы:

1. **Пиксельные (MSE, PSNR)** — оценивают "поточечные" различия, но не всегда соответствуют человеческому восприятию.
2. **Перцептивные (SSIM, LPIPS)** — учитывают структуру, контраст и семантику изображения, приближаясь к тому, как видит глаз.

В данном разделе рассматриваются наиболее распространённые метрики, их преимущества, ограничения и области применения.

**1. Пиковое отношение сигнала к шуму (PSNR)**

Одной из самых простых и широко используемых метрик является **пиковое отношение сигнала к шуму (Peak Signal-to-Noise Ratio, PSNR)**. Она вычисляется через **среднеквадратичную ошибку (MSE)** между исходным и восстановленным изображением:

где ​ — максимально возможное значение пикселя (например, 255 для 8-битного изображения), а MSE определяется как:

где I — исходное изображение, K — восстановленное, а M и N — его размеры.

**Преимущества PSNR:**

* Простота вычисления.
* Чёткая интерпретация: чем выше значение (измеряется в децибелах, дБ), тем лучше качество.

**Недостатки:**

* Слабо коррелирует с человеческим восприятием, особенно при высоких уровнях сжатия.
* Не учитывает структурные искажения — например, размытие или артефакты блочности.

Несмотря на ограничения, PSNR остаётся популярной метрикой благодаря своей прозрачности и лёгкости реализации.

**2. Индекс структурного сходства (SSIM)**

Для более точной оценки визуального качества был предложен **индекс структурного сходства (Structural Similarity Index, SSIM)**. В отличие от PSNR, SSIM учитывает три компонента изображения:

1. **Яркость** (luminance).
2. **Контраст** (contrast).
3. **Структуру** (structure).

Формула SSIM для двух изображений x и y имеет вид:

где:

* , — средние значения яркости,
* , — стандартные отклонения (контраст),
* — ковариация между x и y (структурная схожесть),
* ,​ — константы для стабилизации.

**Преимущества SSIM:**

* Лучше соответствует субъективному восприятию, чем PSNR.
* Учитывает не только яркостные, но и структурные искажения.

**Недостатки:**

* Вычисляется локально (по блокам), что может приводить к неточностям на границах объектов.
* Менее интерпретируем, чем PSNR (значения в диапазоне [-1, 1], где 1 — идеальное совпадение).

SSIM часто используется в сочетании с PSNR для комплексной оценки алгоритмов сжатия, таких как JPEG или нейросетевые методы.

**3. Метрики на основе машинного обучения (LPIPS, VIF)**

С развитием глубокого обучения появились более сложные метрики, учитывающие **перцептивные особенности** человеческого зрения:

**а) Learned Perceptual Image Patch Similarity (LPIPS)**

* Основана на сравнении активаций свёрточных нейросетей (например, VGG или AlexNet).
* Хорошо выявляет артефакты, незаметные для PSNR/SSIM (например, искажения текстур).

**б) Visual Information Fidelity (VIF)**

* Оценивает потерю информации в частотной области.
* Особенно полезна для анализа сжатия с потерями (JPEG, WebP).

**Преимущества:**

* Высокая корреляция с субъективными оценками.
* Учитывают семантику изображения (например, важность сохранения лиц в портретах).

**Недостатки:**

* Требуют предобученных моделей.
* Вычислительно сложнее, чем PSNR/SSIM.

**4. Сравнительный анализ метрик**

На практике выбор метрики зависит от задачи:

* **Для быстрой оценки** подойдут PSNR и SSIM.
* **Для анализа перцептивного качества** — LPIPS или VIF.
* **В научных публикациях** часто приводят несколько метрик для полноты.

Например, при сжатии медицинских изображений критично сохранять мелкие детали, поэтому SSIM и VIF предпочтительнее PSNR. В то же время для стриминга видео важнее скорость вычислений, и PSNR остаётся стандартом.

# **ПРАКТИЧЕСКАЯ ЧАСТЬ**

Данная работа рассматривает задачу архивации изображений, т.е. сжатия с последующим восстановлением до исходного качества. Нейронные сети реализованы в среде разработки Visual Studio Code на языке программирования Python. Устройство, на котором выполнялась работа, имеет следующие технические характеристики:

- процессор: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz;

- оперативная память: 16 ГБ;

- тип системы: 64-разрядная операционная система.

\*технические характеристики компьютера\*

/картинки с файлами

Для работы каждой из программ, реализующих нейросеть, необходим импорт набора библиотек:

- numpy: понадобится для вычислений и проведения математических операций;

- tensorflow: фреймворк для машинного обучения, с его помощью будут реализованы нейронные сети;

- matplotlib: будет использована для визуализации данных, в частности изображений и вывода метрик качества;

- os, sys, signal, time, timedelta: системные библиотеки для работы с файловой системой, интерпретатором Python, обработки сигналов, замеров времени и логирования.

Для оптимизации работы программы и упрощения процесса обучения нейронных сетей реализованы специальные функции, которые позволяют прерывать обучение нейросетей до его завершения без потери прогресса. Для этого в начале программы инициализируется путь к файлу для хранения весов; на рис. продемонстрирован путь к файлу для хранения весов автокодировщика, а также путь к нему. Для гибридной модели реализована такая же строка, но с соответствующим названием пути и именем файла.



Рис. Путь к файлу с весами автокодировщика и его название.

Прерывание запускает функцию сохранения весов в файл, что позволяет продолжить обучение позже без потери прогресса. При новом запуске программы веса загружаются из файла, перед этим происходит проверка на наличие данных в файле. На рис. показана функция сохранения весов, на рис. – функция загрузки весов.

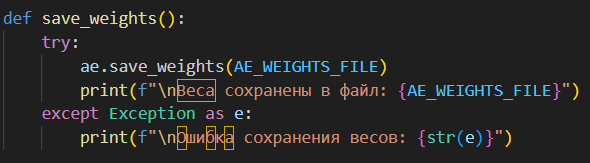


Рис. Функция сохранения весов в файл.

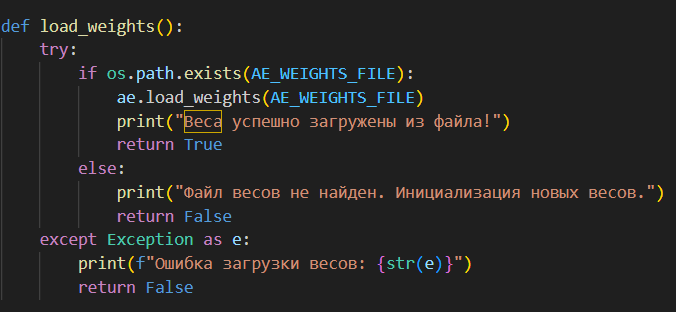


Рис. Функция загрузки весов из файла.

Само прерывание программой реализуется комбинацией Ctrl+C в терминале VSCode. Эту комбинацию обрабатывает специальная функция, которая завершает работу программы. Код функции представлен на рис.

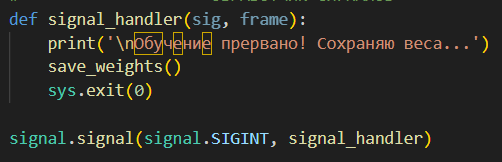


Рис. Функция для завершения программы.

Кроме этого, из базовых функций, которые применяются в обеих нейронных сетях, реализованы функция и класс для отслеживания времени обучения по каждой из эпох. Для удобства восприятия секунды конвертируются в формат ЧЧ:ММ:СС. Класс TimeTracker состоит из нескольких функций: одна инициализирует сам класс, вторая засекает старт текущей эпохи, третья вычисляет конец обучения, четвертая предсказывает время до конца обучения. На рис. изображен блок кода с описанными выше функциями.

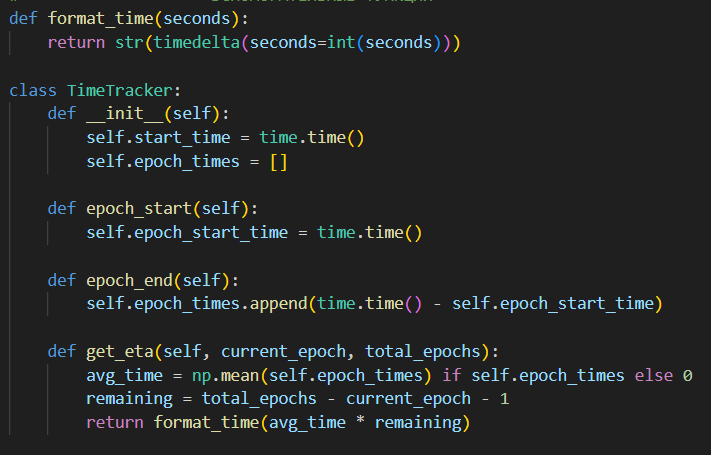


Рис. Функции и класс для работы со временем.

Чтобы отслеживать визуальный прогресс обучения, был реализован класс с визуализацией.

**Нейронная сеть 1: классический автокодировщик**