ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ
им. А.Н. ТИХОНОВА

Бычкова Евгения Денисовна, группа БИБ192

ВСТРАИВАНИЕ ЦИФРОВЫХ ВОДЯНЫХ ЗНАКОВ В ИЗОБРАЖЕНИЯ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ

Выпускная квалификационная работа
по направлению 10.03.01 Информационная безопасность
студента образовательной программы бакалавриата
«Информационная безопасность»

Студент

_____ Е.Д. Бычкова
подпись

Руководитель
Зав. каф. информационной безопасности
киберфизических систем департамента
электронной инженерии МИЭМ НИУ ВШЭ
О.О. Евсютин

Москва 2023г.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ
им. А.Н. ТИХОНОВА

ЗАДАНИЕ
на выпускную квалификационную работу бакалавра

студенту группы БИБ192_Бычковой Евгении Денисовне

1. Тема работы
   «Встраивание цифровых водяных знаков в изображения на основе нейронных сетей»

2. Требования к работе

2.1. Цель работы

- Разработка и исследование эффективного алгоритма встраивания цифровых водяных знаков в цифровые изображения на основе нейронных сетей.

2.2. Требования к результатам работы

- Должен быть получен оригинальный алгоритм встраивания цифровых водяных знаков в цифровые изображения, основанный на аппарате нейронных сетей.
- Эффективность полученного алгоритма должна быть подтверждена с помощью вычислительных экспериментов.

2.3. Требования к документации

- Текст ВКР должен быть оформлен в соответствии с ГОСТ.

3. Содержание работы

3.1. Обзор литературы по теме работы

3.2. Выявление актуальных направлений исследований, связанных с применением машинного обучения и нейронных сетей для встраивания цифровых водяных знаков в цифровые изображения.

3.3. Сравнительное исследование выбранных алгоритмических решений по встраиванию цифровых водяных знаков в цифровые изображения.

3.4. Реализация алгоритма встраивания цифровых водяных знаков в

цифровые изображения, основанного на аппарате нейронных сетей
3.5 Модификация алгоритма встраивания цифровых водяных знаков для достижения большей эффективности
3.6. Проведение вычислительных экспериментов, направленных на оценку эффективности разработанного алгоритма
3.7. Подготовка пояснительной записки к ВКР

4. Сроки выполнения этапов работы

Проект ВКР представляется студентом в срок до	«12» февраля 2023г.
Первый вариант ВКР представляется студентом в срок до	«25» апреля 2023г.
Итоговый вариант ВКР представляется студентом руководителю до загрузки работы в систему «Антиплагиат» в срок до	«10» мая 2023г.

Задание выдано	«25» декабря 2023 г.	_____ подпись руководителя	О.О. Евсютин
Задание принято к исполнению	«25» декабря 2023г.	_____ подпись студента	Е.Д. Бычкова

Оглавление

Аннотация 5
Введение 6
Цель и задачи работы 8
1. Общие сведения о встраивании цифровых водяных знаков в изображения. 9
1.1 Показатели эффективности схем 9
1.2 Классификация водяных знаков 9
1.2.1 Классификация по области встраивания 10
1.2.2 Классификация по надежности водяного знака 10
1.2.3 Классификация по процедуре извлечения 10
1.3 Атаки на изображения 11
1.4 Использование нейросетей 12
2. Обзор литературы 14
3. Схема встраивания водяного знака 18
3.1 Описание схемы 18
3.1.1 Извлечение данных об изображении 19
3.1.2 Кодировщик 20
3.1.3 Симулятор атаки 22
3.1.4 Декодер 23
3.1.5 Дискриминатор 24
3.2 Обучение нейросетей 26
3.2.1 Функция потери 26
3.2.2 Наборы данных 26
4. Результаты 28
4.1 Первичные результаты 28
4.2 Описание модификаций 29
4.3 Конечные результаты 31
4.4 Оценка устойчивости 32
Вывод 36
Список литературы 37
Приложение 1 40

Аннотация

Цифровые водяные знаки используются для защиты авторских прав и подтверждения подлинности медиаданных. Одним из направлений развития данной области является построение схем с использованием нейросетей. Они достигают большой производительности и гибкости в решении задач. Это позволяет создавать новые эффективные схемы или увеличивать качество уже существующих.

В данной работе исследуются статьи, совмещающие обе темы и показывающие, каким образом нейросети применяются в защите авторских прав. Подробно изучается схема, встраивающая цифровые водяные знаки в изображения и состоящая из нескольких нейросетей, а также проводятся эксперименты по увеличению ее показателей эффективности.

Введение

В современном мире появляется все больше и больше мультимедийных данных. Люди имеют свободный доступ к огромному количеству видео, аудио файлов, изображений и прочего. Они могут копировать, хранить и передавать информацию. Сложности не представляет даже ее изменение из-за распространения программ редакторов. Это быстро привело к незаконной деятельности, нарушающей авторские права владельцев мультимедийных данных, из-за чего возникает необходимость в механизмах, позволяющих проверить подлинность информации и определить ее авторство.

Одним из таких механизмов стали цифровые водяные знаки. Это дополнительная информация о цифровом объекте и его владельце, которая встраивается в сам объект по заданной схеме [1]. Таким образом, он может незаметно содержать в себе данные о праве собственности, получателе, метки времени и прочие необходимые для конкретной задачи данные.

В схемах цифровых водяных знаков определяется алгоритм для встраивания и извлечения информации. Она может быть представлена как битовая строка, цветное изображение, двоичное изображение, текст и другие типы данных. Объект, в который она встраивается, называется контейнером. В данной работе в качестве контейнеров рассматриваются только изображения, так как они являются одним из самых частых типов мультимедийных данных.

Для встраивания водяных знаков часто используют дискретное вейвлет-преобразование, дискретное косинусное преобразование, дискретное преобразование Фурье и прочее. Они обеспечивают хорошую устойчивость. Однако разрабатываются “вручную” для разных типов данных и требую достаточного количества знаний и внимания ко множеству факторов. Это замедляет процесс развития цифровых водяных знаков.

Возможным решением данных проблем является использование нейронных сетей в схемах цифровых водяных знаков. Они дают возможность формулировать задачу как минимизацию функции потерь, что упрощает

задачу. При этом нейросети помогают оптимизировать параметры эффективности и работают с разными типами данных.

Цель и задачи работы

Целью работы является реализация схемы встраивания цифровых водяных знаков в изображения, использующей нейросети.

Для достижения цели во время работы нужно выполнить задачи:

– изучить понятие цифровых водяных знаков,

– изучить уже существующие схемы встраивания цифровых водяных знаков в изображения, использующие нейросети,

– выбрать схему и реализовать ее,

– провести анализ показателей эффективности реализованной схемы,

– провести эксперименты по улучшению показателей эффективности.

1. Общие сведения о встраивании цифровых водяных знаков в изображения

1.1 Показатели эффективности схем

Эффективность цифровых водяных знаков оценивается с помощью нескольких параметров [2]:

1. Устойчивость. Изображения распространяются по различным каналам, в том числе по Интернету. Данные могут исказиться во время передачи из-за шумов или могут быть подвержены атакам. Схеме необходимо уметь извлекать и проверять водяной знак даже после таких воздействий.

2. Невидимость. Встраивание водяных знаков ведет к искажению контейнера. Это может быть заметно в изменении деталей изображения. Из-за этого необходимо обеспечить сходство изображения с оригиналом и незаметность искажений.

3. Емкость. Водяной знак содержит информацию о владельце. Ее должно быть достаточно, чтобы подтвердить авторские права. Таким образом, необходимо обеспечить достаточную вместимость контейнера.

4. Вычислительная сложность. Сложность алгоритма влияет на время его выполнения. Также, устройства могут иметь ограничения, которые необходимо учитывать при проектировании схемы.

Оптимизировать схему по всем параметрам - задача, практически не выполнимая, так как все они зависят друг от друга. При попытке увеличить емкость контейнера, может пострадать невидимость водяного знака. Увеличение невидимости приведет к снижению прочности, и тому подобное. Поэтому во время проектирования схемы выбирают приоритеты в зависимости от задачи.

1.2 Классификация водяных знаков

Цифровые водяные знаки можно классифицировать по разным признакам, связанным с процессами встраивания и извлечения или параметрами водяных знаков. Общая информация показана на рисунке 1.

1.2.1 Классификация по области встраивания

1. Пространственная область. Такие водяные знаки являются самыми простыми и встраиваются непосредственно в пиксели изображения. Они имеют низкую сложность и высокую емкость, однако они также легко ломаются при изменении изображения [3].

2. Область преобразования. Для встраивания таких водяных знаков используются коэффициенты частотной области. Это делает их более прочными и незаметными. Однако в коэффициенты нельзя встроить большое количество информации. Также схемы встраивания в область преобразования имеют более высокую сложность.

3. Нулевой водяной знак. В отличие от остальных вариантов, такие схемы не встраивают информацию в изображение. Вместо этого они создают отдельный ключ, используя водяной знак и важные характеристики основного изображения. Таким образом, нулевые водяные знаки обладают идеальной невидимостью [4].

1.2.2 Классификация по надежности водяного знака

1. Хрупкие. Эти водяные знаки разрушаются от любого искажения контейнера. Они могут использоваться для контроля целостности изображения и определения области искажения.

2. Устойчивые водяные знаки можно извлечь после сильных искажений контейнера. Они используются для подтверждения авторства и подлинности изображения.

3. Полухрупкие. Такие водяные знаки, в отличие от хрупких, способны выдержать некоторые типы атак. Они в какой-то степени решают обе задачи предыдущих водяных знаков.

1.2.3 Классификация по процедуре извлечения

1. Слепые. При извлечении водяного знака не требуется оригинальное

изображение или какая-то дополнительная информация. Такие схемы часто имеют высокую сложность [5].

2. Полуслепые. Здесь, для извлечения нужна дополнительная информация: ключ или изначальный водяной знак. При этом схеме возможно нужно выделить дополнительное пространство для этой информации.

3. Неслепые. Для извлечения требуются оригинальное изображение и водяной знак.

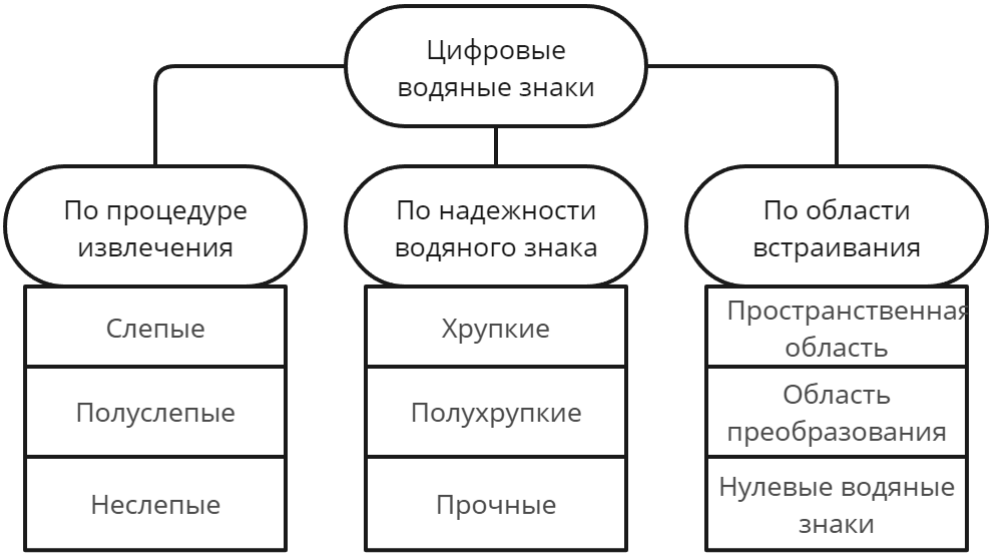

Рисунок 1 - Классификация цифровых водяных знаков

1.3 Атаки на изображения

Изображения могут быть подвержены различным атакам. Среди распространенных атак можно выделить [6]:

1. Атаки обработки сигналов. Чаще всего это незлонамеренные и широко распространенные действия: сжатие, печать, сканирование. Такие действия обычно предпринимаются для соблюдения требований формата или скорости передачи. К атакам обработки сигналов также относятся и преднамеренные действия. Например, добавление шумов к изображению.

2. Геометрические атаки. Они могут быть разделены на общие геометрические атаки и локальные геометрические атаки. К общим

геометрическим атакам относятся вращение, масштабирование, сдвиг и другие преобразования. В отличие от общих геометрических атак, локальные относят к себе преобразования в отдельных местах изображения: обрезка, удаление строк или столбцов изображения и так далее.

1.4 Использование нейросетей

Уже долгое время активно развивается направление нейросетей. Они достигают большой производительности и гибкости в решении различных задач. Будучи универсальным инструментом, они нашли применение в самых разных областях: от медицины [7] до экономики [8]. В сфере информационной безопасности, в частности, в построении цифровых водяных знаков, они также активно используются.

Так как данная работа фокусируется на встраивании водяных знаков в изображения, то стоит отметить успех нейросетей в их обработке.

Одним из лучших инструментов в работе с изображениями являются сверточные нейросети [9]. Операция свертки, служащая их основой, помогает извлекать полезные функции из локальных областей изображения. Это позволяет нейросетям получать ключевые признаки и хорошее внутреннее представление об изображениях без их предварительной обработки.

Таким образом, нейросети способны решать ряд задач, связанных с изображениями. Например, их классификация и сегментация. Извлеченные признаки используются для обнаружения границ объектов на изображении и их оценку.

Также нейросети хорошо показали себя в создании и модификации изображений. Одним из практических применений здесь является улучшение качества и повышение расширения. С точки зрения передачи данных, это позволяет исправлять дефекты, возникшие при транспортировке изображения.

Помимо вышеперечисленных возможностей, нейросети могут обеспечивать большую автоматизацию системы за счет замены долгосрочной деятельности автоматическими процедурами, тем самым увеличивая скорость

работы [10]. Например, в [11] рассматривается метод оптимизации качества и надежности водяного знака с использованием машинного обучения, который не только решает свою непосредственную задачу, но и учитывает затраты по времени.

Нейросети применяются для встраивания и извлечения цифровых водяных знаков, что более подробно рассматривается в секции 2.

2. Обзор литературы

В данной секции рассматриваются недавние статьи, посвященные цифровым водяным знакам. Все они описывают схемы, использующие нейросети на разных этапах работы. Обобщенные данные находятся в таблице 1.

В работе [12] объединено дискретное вейвлет преобразование и сверточные нейронные сети. Оба преобразования используются и для встраивания, и для извлечения. В нейросеть для встраивания водяного знака подается один из слоев после вейвлет-преобразования, после чего результат проходит через обратные преобразования. Такая комбинация методов повышает точность и производительность.

Статья [13] тоже описывает схему водяных знаков, в которой совмещены два преобразования. Изображение обрабатывается нейросетью. Однако некоторые ее слои заменены на блоки вейвлет-преобразования.

Работа [14] описывает схему, основанную на вейвлет преобразовании. С его помощью цифровой водяной знак встраивается в контейнер. На этапе извлечения вейвлет преобразование используется, чтобы получить набор признаков, которые затем обрабатываются нейросетью.

Алгоритм, предложенный в статье [15] по отдельности преобразует RGB каналы изображения, вставляя в них водяной знак и дополнительную информацию. Однако нейросеть здесь используется только для увеличения качества и надежности извлеченного водяного знака. Так же, как и в схемах [16] и [17].

В статье [18] все преобразования производятся сверточными нейросетями. Для достижения незаметности и надежности используется предобработка контейнера для выделения областей, изменение которых будет менее заметно для человеческого глаза.

Похожую схему описывают в работе [19]. Встраивание и извлечение происходя с помощью нейросетей. Отличительной особенностью этой работы является внедрение вспомогательной информации. Она позволяет помимо

получения водяного знака восстановить и исходное изображение.

Алгоритм [20] встраивает водяные знаки по той же общей схеме, однако здесь основное внимание уделяется защите от съемки экрана, что редко можно увидеть в других статьях. Кроме того, работа рассматривает не любые изображения, а изображения документов.

Статья [21] описывает схему цифрового водяного знака для медицинских изображений. Чтобы не искажать их, создается нулевой водяной знак. Для этого обучается нейросеть, извлекающая из изображения его признаки. Они складываются по модулю 2 с водяным знаком. Извлечение водяного знака происходит по тому же алгоритму, что и его создание.

В работе [22] описывается схема нулевого водяного знака. Для этого обучаются две нейросети. Одна из них создает изображение, содержащее цветовой стиль основного изображения и содержание логотипа с отметкой времени. Это изображение и является нулевым водяным знаком. Вторая нейросеть обучается для извлечения из него логотипа.

Как видно, нейросети применяются в области цифровых водяных знаков достаточно часто и для разных задач, что говорит об актуальности данного направления.

Все схемы имеют хорошие показатели эффективности. Однако с точки зрения использования нейросетей наибольший интерес представляют схемы, использующие их и для встраивания, и для извлечения водяных знаков.

Схемы с нулевыми водяными знаками используют способность нейросетей к обработке изображений, используя преобразование стилей и извлечение признаков. Однако с точки зрения практического применения схемы, встраивающие водяной знак в контейнер, кажутся более интересными, потому что их задача состоит в обеспечении не только прочности водяного знака, но и его невидимости. Здесь используются нейросети разных архитектур и обеспечивается их совместная работа для выполнения задачи.

Для реализации выбрана схема [18], так как из рассмотренных схем, использующих только нейросети, она имеет наибольший показатель PNSR.

Таблица 1 - Рассматриваемые схемы

№	Какая нейросеть используется	Для чего используется нейросеть	Основное преобразование схемы	Контейнер	Водяной знак	PNSR	Рассматриваемые атаки
[12]	CNN	Встраивание извлечение, предварительная обработка контейнера и водяного знака	Нейросети, DWT	Цветное изображение 256x256	Строка 256 бит	50,9	Гауссовский шум, шум соли и перца, сжатие jpeg, выбивание пикселей
[13]	U-net, CNN	Встраивание, извлечение	DWT, нейросети	Цветное изображение 400x400	Битовая строка	32,35*	Сжатие jpeg, шум, размытие
[14]	FCNN	Извлечение	DWT	Серое изображение 512x512	Двоичное изображение 32x32	44,11	Сжатие jpeg, размытие по Гауссу, сжатие
[15]	DnCNN	Повышение качества и надежности извлеченного водяного знака		Цветные изображения 512x512	Серое изображение 128x128	57,7	Гауссовский шум, шум соли и перца, медианный фильтр, сжатие jpeg, масштабирование, обрезание
[16]	ResNet	Шумоподавление и реконструкция супер-разрешения	DCT	Цветные изображения 512x512	цветные изображения 64x64	37,25	Сжатие jpeg, гауссовский шум, шум соли и перца, фильтрация: медианная, нижних частот: масштабирование, выравнивание гистограммы, вращение, осветление, затемнение
[17]	SRCNN	Повышение качества и надежности извлеченного водяного знака	QR разложение	Цветное изображение 128x128	Два цветных изображения 64x64 и 64x32	38,02	Шум соли и перца, фильтр Гаусса, вращение, масштабирование, обрезка, осветление, затемнение, сжатие jpeg
[18]	CNN	Встраивание, извлечение	Нейросети	Цветное изображение 128x128	Серое изображение 128x128	40,69	Сжатие jpeg, обрезка, выбивание пикселей, размытие по Гауссу
[19]	CNN	Встраивание, извлечение	Нейросети	Цветные изображения 128x128	Битовая строка 50 бит	38,00*	Сжатие jpeg, гауссовский шум, вращение, кадрирование
[20]	U-net, CNN	Встраивание, извлечение	Нейросети	Изображение документа 400x400	100 бит	28.44	Съемка экрана, сжатие jpeg, масштабирование, размытие по Гауссу

[21]	VGG	Создание нулевого водяного знака, извлечение	Нейросети	Изображения 128x125	Изображение 64x64	-	Гауссовский шум, сжатие jpeg, медианный фильтр
[22]	VGG19, VCNN	Создание нулевого водяного знака, извлечение	Нейросети	Изображение 128x128	Изображение 128x128	-	Сжатие jpeg, гауссовский шум, масштабирование, обрезка, вращение

3. Схема встраивания водяного знака

3.1 Описание схемы

Схема цифрового водяного знака имеет структуру автокодировщика.

Автокодировщики обычно состоят из двух частей: кодировщика и декодера. Кодировщик приводит изображение к определенному виду, из которого декодер восстанавливает новое изображение согласно задаче. При этом для обучения не требуются специально размеченные данные.

Данная схема, показанная на рисунках 2 и 3, состоит из нескольких блоков:

1. Извлечение данных об изображении
2. Кодировщик
3. Симулятор атаки
4. Декодер
5. Дискриминатор

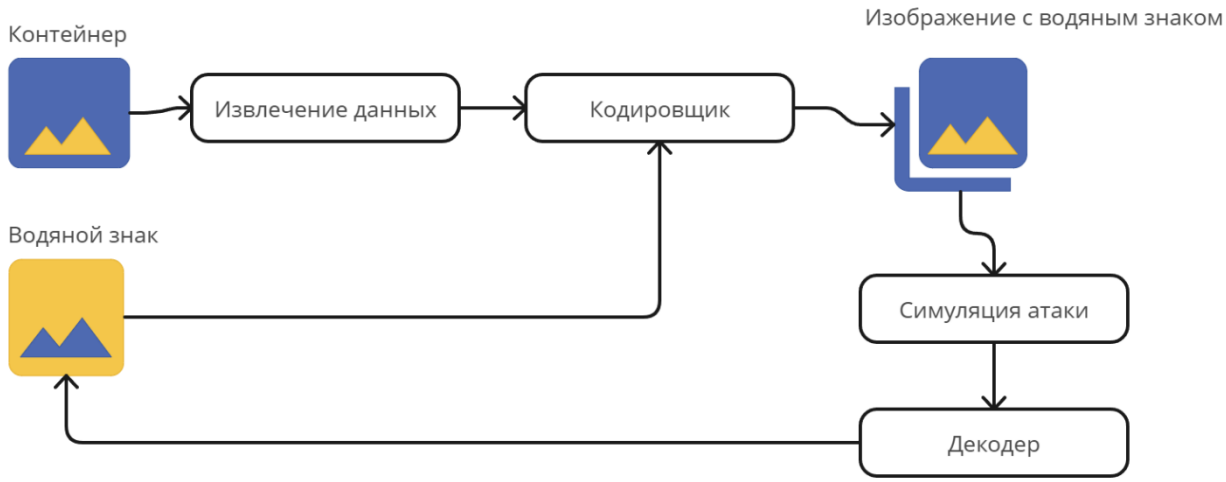

Рисунок 2 - Схема цифрового водяного знака

Рисунок 3 - Схема работы дискриминатора

3.1.1 Извлечение данных об изображении

Задача данного блока состоит в выделении областей, подходящих для встраивания водяного знака. Человеческий глаз менее чувствителен к изменениям некоторых областей изображения. Следовательно, встраивание в них водяного знака будет менее заметно.

Для преобразования необходимо найти области изображения с границами и высокой цветностью. Границы находятся с помощью оператора Кэнни по формуле:

$$S_E(x) = \exp\left(-\frac{Canny(x) + 1}{\tau}\right)$$

где $\tau = 2$.

Для вычисления цветности сначала надо конвертировать изображение формат YCbCr. После этого применяется формула:

$$S_C(x) = 1 - \exp\left(-\frac{f_b^2(x) + f_c^2(x)}{\delta^2}\right)$$

где $\delta = 0{,}25$.

Окончательное изображение вычисляется по формуле:

$$I_{in} = I_{co} - \left(1 - \frac{S_C(x) + S_E(x)}{2}\right)$$

где I_{co} – исходное изображение после нормализации.

Результаты работы данного блока показаны на рисунке 4.

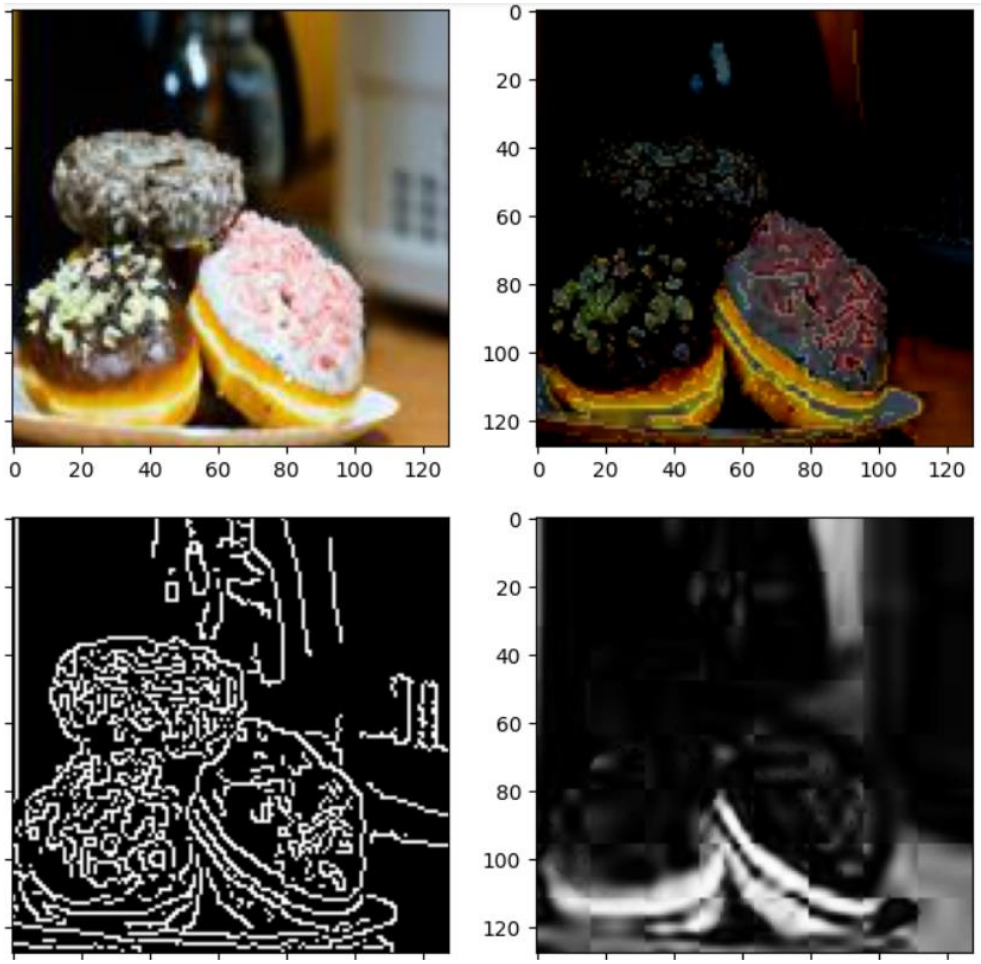

Рисунок 4 – Предобработка изображения: оригинальное изображение, обработанное изображение, выделенные границы, выделенные области с повышенной цветностью

3.1.2 Кодировщик

Задача кодировщика - встраивание цифрового водяного знака. Он представляет собой сверточную нейросеть, архитектура которой показана на рисунке 5.

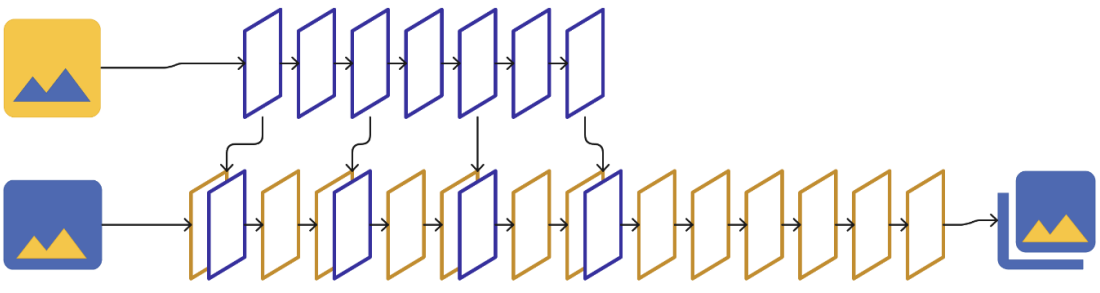

Рисунок 5 – Архитектура кодировщика

Вся нейросеть состоит из сверточных слоев. Они нужны для выделения отличительных признаков изображения [25]. Это происходит с помощью операции свертки. На исходное изображение накладывается матрица – детектор признаков. Результат перемножения сохраняется в карте признаков, после чего матрица сдвигается, и операция повторяется. На рисунке 6 показана операция свертки.

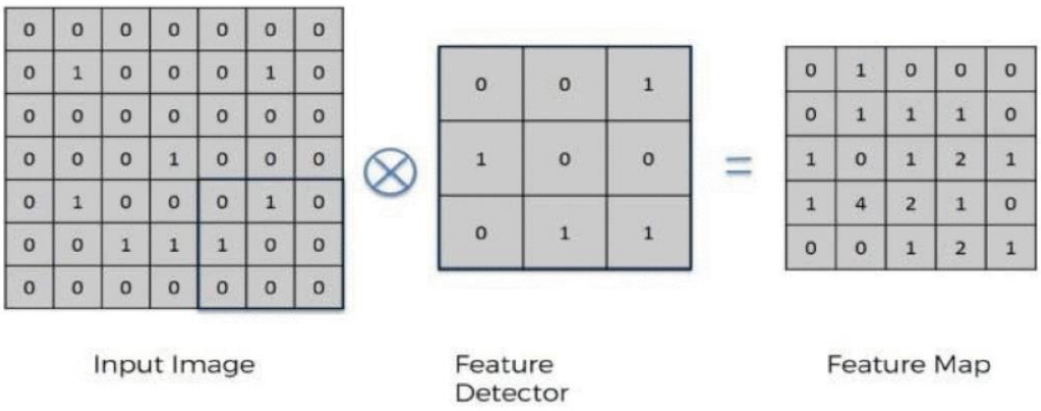

Рисунок 6 – Операция свертки

Таким образом каждый элемент карты показывает, есть ли в данной области изображения определенный матрицей признак. Причем, в одном слое может использоваться несколько матриц. Их количество определяется количеством выходных каналов и задается программой. Так же, как и размер матрицы, называемый размером ядра. Отдельный параметр сверточного слоя – это паддинг. Он определяет ширину рамки из нулей вокруг изображения. Его добавляют для того, чтобы исходное изображение не уменьшилось. Параметры слоев данной нейросети указаны в таблице 2.

Нейросеть состоит из двух параллельных веток. Одна из них обрабатывает изображение водяного знака, а вторая работает с предобработанным контейнером. На каждом слое из изображений извлекаются определенные признаки. Несколько раз они объединяются друг с другом. Таким образом, в контейнере появляется информация о водяном знаке.

Таблица 2 – Параметры слоев кодировщика

Первая ветка (для водяного знака)				
№ слоя	Входные каналы	Выходные каналы	Размер ядра	Паддинг
1	3	16	3	1
2	16	16	3	1
3	16	16	3	1
4	16	16	3	1
5	16	16	3	1
6	16	16	3	1
Вторая ветка (для контейнера)				
1	3	16	3	1
2	32	16	3	1
3	16	16	3	1
4	32	16	3	1
5	16	16	3	1
6	32	16	3	1
7	16	16	3	1
8	32	64	3	1
9	64	128	3	1
10	128	64	3	1
11	64	32	3	1
12	32	16	3	1
13	16	3	3	1

3.1.3 Симулятор атаки

Задача этого блока - имитировать атаки на изображение, чтобы нейросеть научилась извлекать водяные знаки даже из искаженных контейнеров. Он находится между кодировщиком и декодером, таким образом изображение с водяным знаком сначала подвергается атаке, а затем поступает в декодер.

Каждую итерацию обучения случайным образом выбирается одна из атак. Они показаны на рисунке 7.

1. Размытие по Гауссу,
2. Выбивание пикселей,
3. Обрезка,
4. Сжатие JPEG.

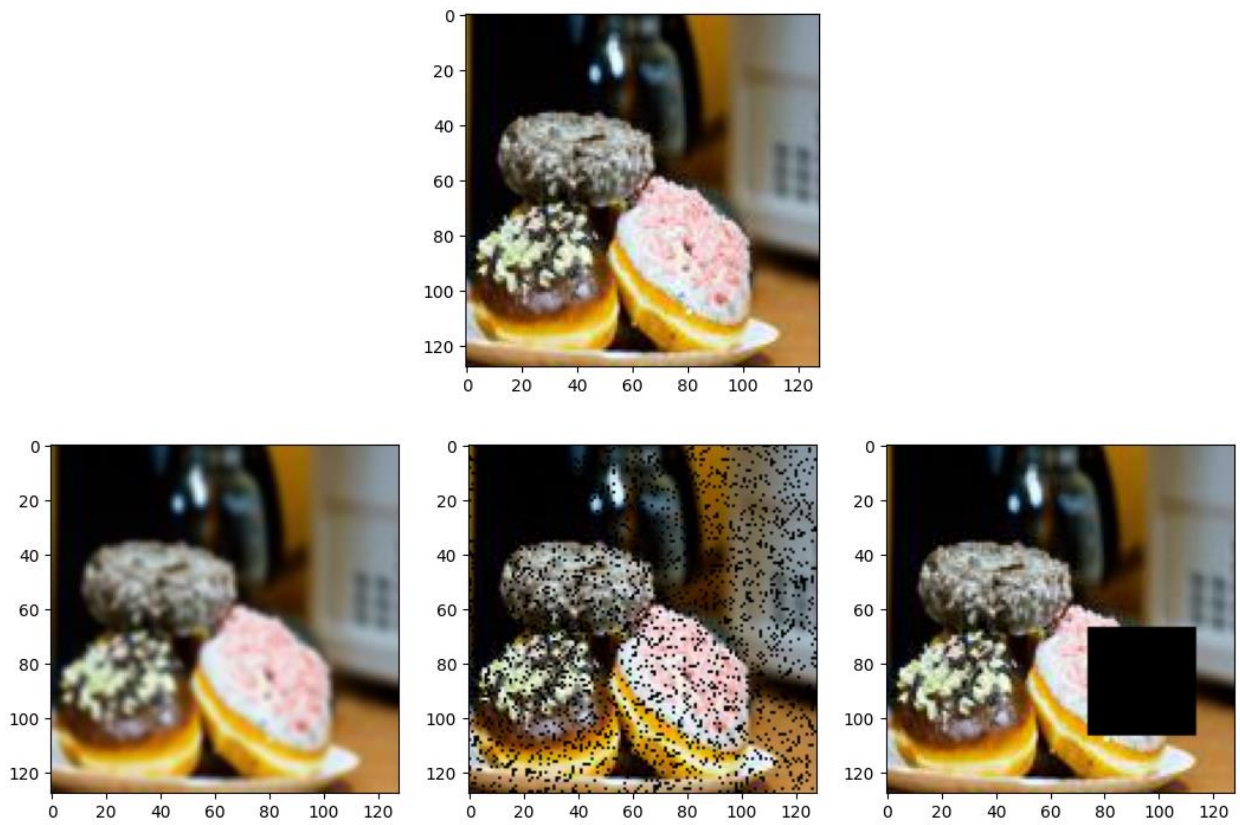

Рисунок 7 – Атаки на изображение: оригинальное изображение, размытие по Гауссу, выбивание пикселей, обрезка

3.1.4 Декодер

Задача декодера - извлечение водяного знака из изображения. При этом изображение может быть искажено на предыдущем этапе.

Декодер представляет собой сверточную нейросеть, архитектура которой показана на рисунке 8. Здесь чередуются сверточные слои, нормализация пакетов и функции активации.

Нормализация пакетов позволяет стабилизировать работу нейросети и повысить ее производительность. Во время обучения нейросеть обрабатывает пакет данных, однако они могут различаться по значениям, поэтому их необходимо преодобработать, чтобы они имели нулевое математическое ожидание и единичную дисперсию.

Функция активации – это нелинейное преобразование. Ее используют между сверточными слоями, чтобы внести изменения в линейные операции

свертки. Таким образом нейросеть может извлекать более информативные признаки.

Нейросеть сначала извлекает признаки из изображений, с каждым слоем все больше, после чего объединяет их, уменьшая количество выходных каналов.

Параметры сверточных слоев указаны в таблице 3.

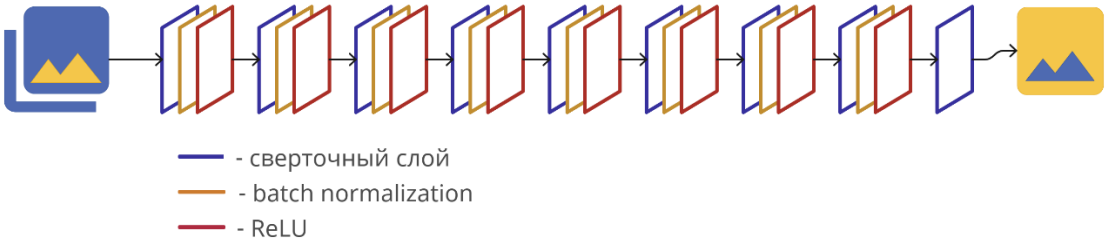

Рисунок 8 – Архитектура декодера

Таблица 3 – Параметры сверточных слоев декодера

№ слоя	Входные каналы	Выходные каналы	Размер ядра	Паддинг
1	3	16	3	1
2	16	32	3	1
3	32	64	3	1
4	64	128	3	1
5	128	64	3	1
6	64	32	3	1
7	32	16	3	1
8	16	1	3	1

3.1.5 Дискриминатор

Основная роль дискриминатора заключается в улучшении визуального сходства между оригинальным изображением и изображением со встроенным водяным знаком путем состязательного обучения.

Архитектура дискриминатора показана на рисунке 9. Она практически аналогична архитектуре декодера. Разница в том, что дискриминатор должен вывести только одно число, показывающее наличие или отсутствие в изображении водяного знака. Таки образом, нейросеть завершается линейным

слоем.

В данной сети используется слой пулинга. Его задача – уменьшение размера входных данных. Здесь входные данные делятся на блоки, к которым применяется определенная функция. В данной нейросети – берется среднее значение.

Линейный слой производит линейное преобразование входящих данных, перемножая их на определенные веса. При этом для него определяется размер входящих данных и размер выходящих данных.

Нейросеть извлекает признаки из изображений с помощью сверточных слоев, после чего на их основе делает предсказание.

Параметры слоев указаны в таблице 4.

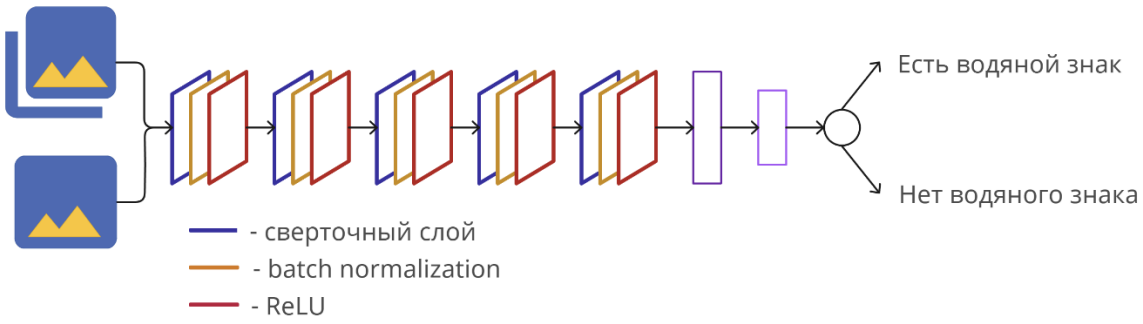

Рисунок 9 – Архитектура дискриминатора

Таблица 4 – Параметры слоев дискриминатора

№ слоя	Входные каналы	Выходные каналы	Размер ядра	Паддинг
1	3	16	3	1
2	16	32	3	1
3	32	64	3	1
4	64	128	3	1
5	128	256	3	1
6	AvgPool: размер ядра 2			
7	Линейный слой: вход 256*64*64, выход 1			
8	Сигмоида			

3.2 Обучение нейросетей

3.2.1 Функция потери

Все три нейросети обучаются одновременно и имеют общую функцию потери, которая складывается из трех составляющих.

Кодировщик стремится незаметно вставить водяной знак в изображение. Поэтому потери будут считаться из сравнения оригинального и закодированного изображения.

$$L_D(I_{co}, I_{en}) = \|I_{co} - I_{en}\|_2^2$$

где I_{co} и I_{en} – оригинальное и закодированное изображения соответственно.

Декодер стремиться извлечь водяной знак так, чтобы он был максимально похожим на изначальный. Таким образом, функция потери для декодера аналогична функции потери кодировщика.

$$L_R(M, M_{out}) = \|M - M_{out}\|_2^2$$

где M и M_{out} – изначальный и извлеченный водяные знаки соответственно.

Цель дискриминатора – определить, содержит ли изображение водяной знак. Его функция потерь показывает, правильно ли отвечает дискриминатор, и состоит из двух частей: проверка ответов для изображений с водяным знаком и проверка ответов для обычных изображений.

$$L_A = \log\big(1 - D(I_{co})\big) + \log\left(D(I_{en})\right)$$

где D – это дискриминатор.

Таким образом общая функция потерь имеет вид:

$$L = \alpha L_D + \beta L_R + \gamma L_A$$

где α, β и γ – гиперпараметры, определяющие, на что надо больше обращать внимание во время обучения.

3.2.2 Наборы данных

В качестве контейнеров используются цветные изображения размером

128×128. Водяными знаками являются серые изображения того же размера. Выборка формируется из пар этих изображений.

5000 цветных изображений взято из набора данных COCO [23]. Серые изображения взяты из logo-2k+ [24]. Все они были масштабированы согласно заданным размерам.

Имеющиеся 5000 пар разделены в соотношении 8:1:1 на обучающую, тестовую и валидационную выборки.

4. Результаты

4.1 Первичные результаты

Написанная модель использовала данные, предоставленные в оригинальной работе. На рисунке 10 показаны результаты ее работы, а в таблице 5 – сравнение показателей с оригинальной работой.

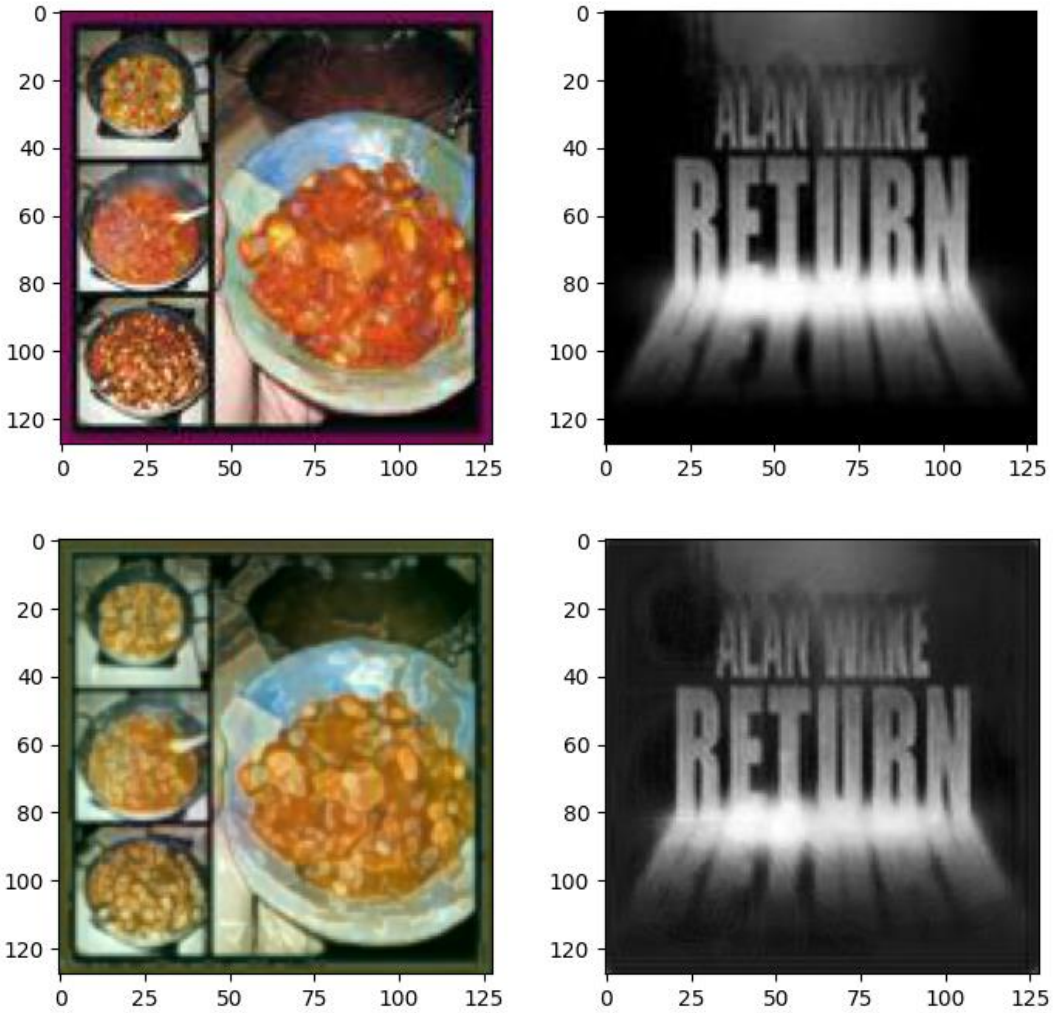

Рисунок 10 – Результаты работы схемы: контейнер, водяной знак, закодированное изображение и извлеченный водяной знак

Таблица 5 – Сравнение PNSR

	Контейнер	Водяной знак
Реализация	38.45	38.60
[18]	40.69	38.19

Как видно, водяной знак извлекается с достаточным качеством и даже немного превосходит в этом плане оригинальную работу. Однако незаметность его встраивания требуется повысить.

Вследствие этого были произведены попытки модифицировать схему для повышения ее показателей качества.

4.2 Описание модификаций

Конечный результат имеет несколько изменений относительно оригинальной схемы. Так как основной целью было повышение незаметности встраивания, большинство из них затрагивает кодировщик.

1. Добавление функции активации. В оригинальной схеме кодировщик не использует функции активации, однако они помогают находить более информативные признаки изображений, увеличивая тем самым качество обучения.

2. Добавление в кодировщик оригинального изображения. По рисунку видно, что закодированное изображение имеет менее интенсивные цвета, чем оригинальное. Поэтому в кодировщик была добавлена информация о цвете в виде оригинального изображения. При этом для слияния признаков с водяным знаком используется только предобработанное изображение.

3. Добавление слоев. Так как на предыдущем шаге в нейросеть была добавлена новая информация, потребовалось добавить сверточные слои, для более качественной ее обработки.

4. Добавление видов атак. В оригинальной работе в симуляторе атак реализовано 4 атаки: сжатие jpeg, размытие по Гауссу, выбивание пикселей, обрезка. Для данной реализации добавлены шум соли и перца и медианный фильтр. Таким образом, планируется повысить устойчивость водяного знака. Новые атаки показаны на рисунке 11.

5. Гиперпараметры функции потерь. В оригинальной работе гиперпараметры $\alpha = 0{,}3$, $\beta = 0{,}7$, $\gamma = 0{,}001$. Они изменены на $\alpha = \beta = 0{,}5$, $\gamma = 0{,}001$.

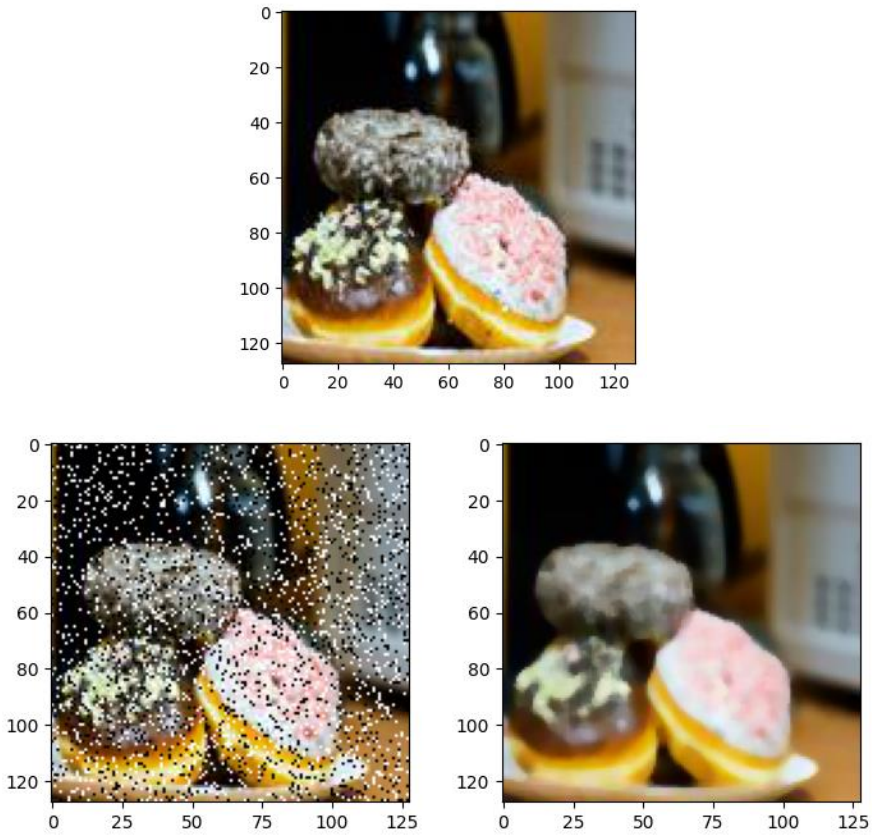

Рисунок 11 – Атаки на изображение: оригинальное изображение, шум соли и перца, медианный фильтр

Архитектура кодировщика была изменена, и она показана на рисунке 12. Параметры слоев указаны в таблице 6.

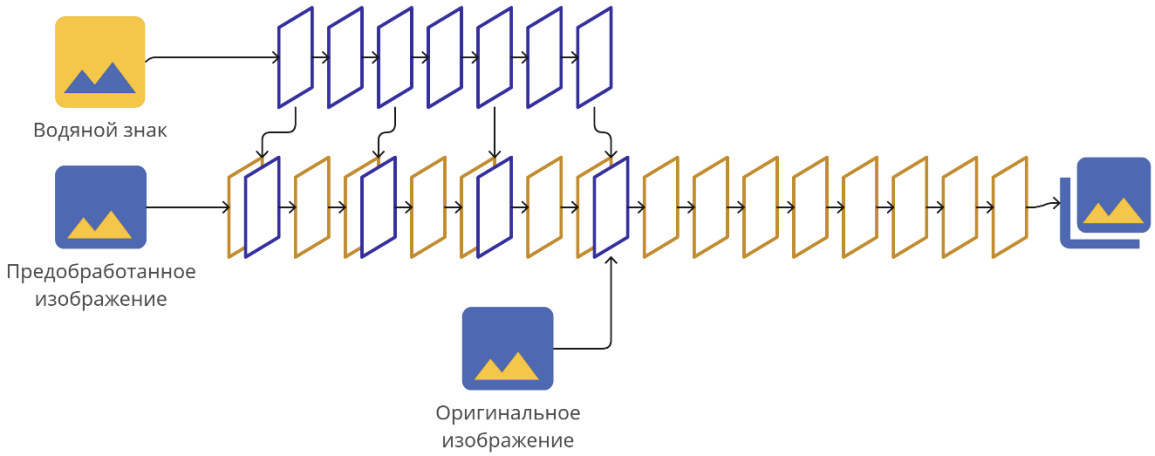

Рисунок 12 – Новая архитектура кодировщика

Таблица 6 – Параметры слоев нового кодировщика

Первая ветка (для водяного знака)					
№ слоя	Входные каналы	Выходные каналы	Размер ядра	Паддинг	Функция активации
1	3	16	3	1	
2	16	16	3	1	
3	16	16	3	1	
4	16	16	3	1	
5	16	16	3	1	
6	16	16	3	1	
Вторая ветка (для контейнера)					
1	3	16	3	1	
2	32	16	3	1	
3	16	16	3	1	
4	32	16	3	1	
5	16	16	3	1	
6	32	16	3	1	
7	16	16	3	1	
8	35	64	3	1	
9	64	128	3	1	ReLU
10	128	256	3	1	ReLU
11	256	128	3	1	ReLU
12	128	64	3	1	ReLU
13	64	32	3	1	ReLU
14	32	16	3	1	ReLU
15	16	3	3	1	

4.3 Конечные результаты

После модификации модель выдает результаты, показанные на рисунке 13, а в таблице 7 сравнение показателей.

Таблица 7 – Сравнение PNSR разных реализаций

	Контейнер	Водяной знак
Первая реализация	38.45	38.60
[18]	40.69	38.19
Конечная реализация	42.28	41.81

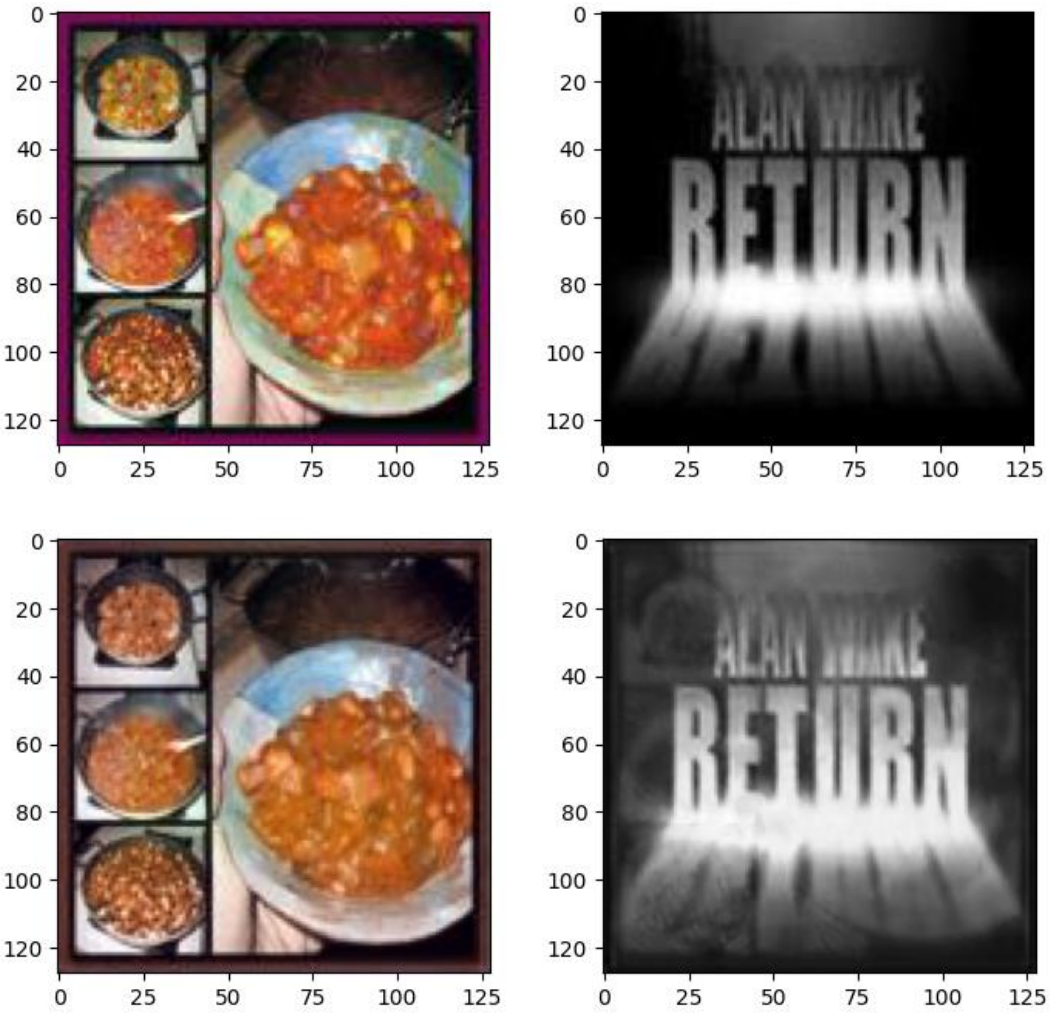

Рисунок 13 – Результаты работы новой схемы: оригинальное изображение, водяной знак, закодированное изображение, извлеченный водяной знак

Как видно, показатели эффективности новой схемы улучшились, а закодированное изображение визуально намного более похоже на оригинал, чем в первой версии реализации.

4.4 Оценка устойчивости

Для оценивания устойчивости водяного знака необходимо атаковать закодированное изображение. На рисунках 14-20 показаны атакованные изображения и извлеченные из них водяные знаки.

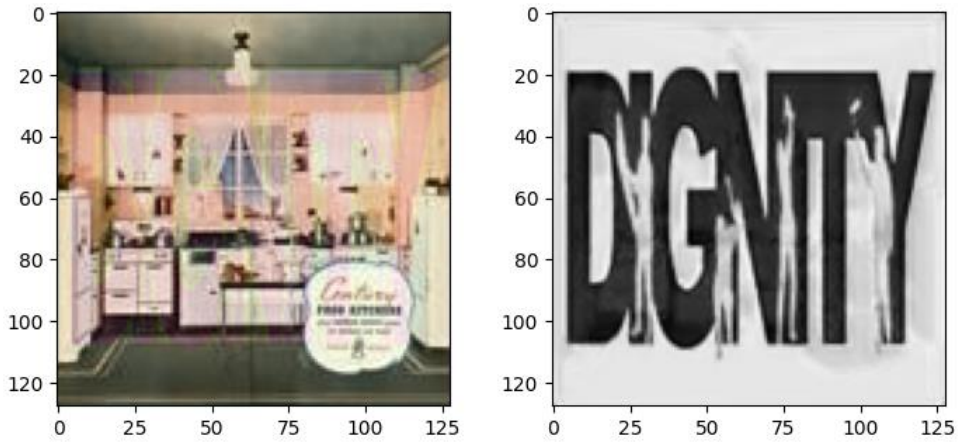

Рисунок 14 – Извлечение водяного знака из неатакованного изображения

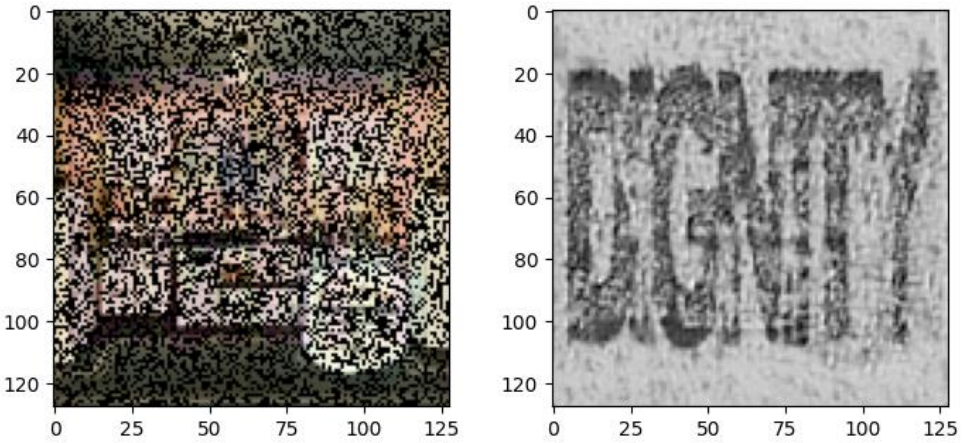

Рисунок 15 – Извлечение водяного знака после выпадения пикселей

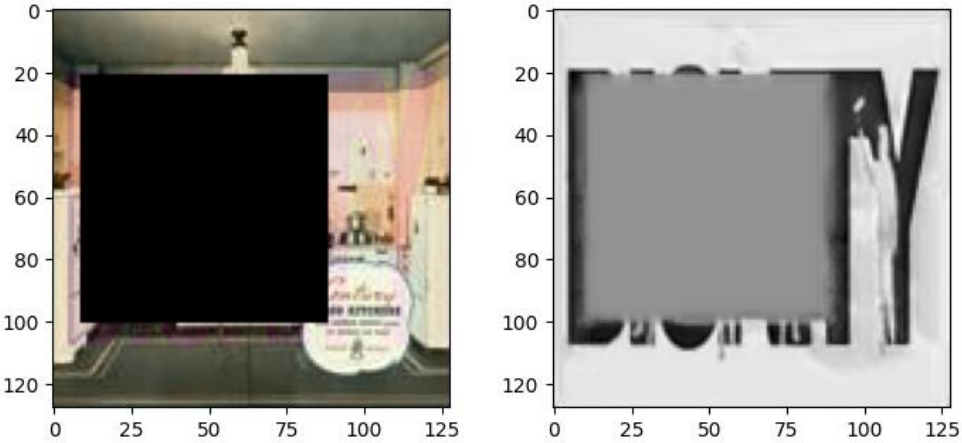

Рисунок 16 – Извлечение водяного знака после обрезки

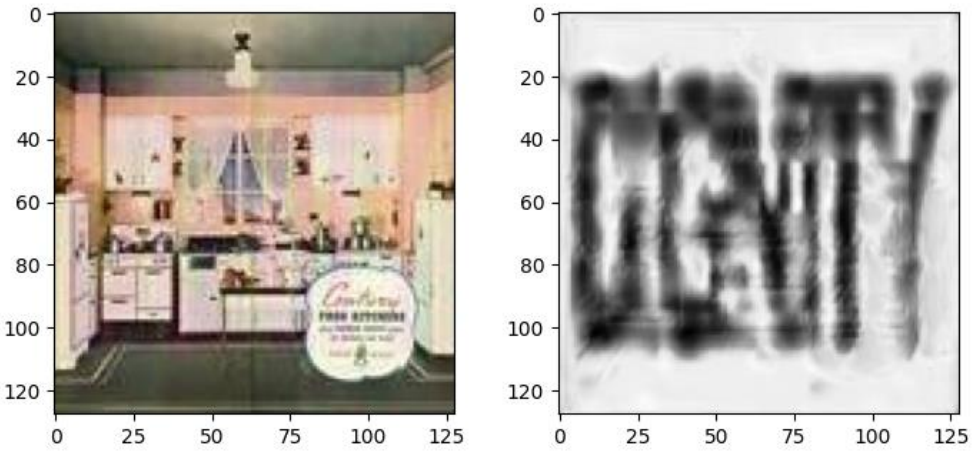

Рисунок 17 – Извлечение водяного знака после сжатия jpeg

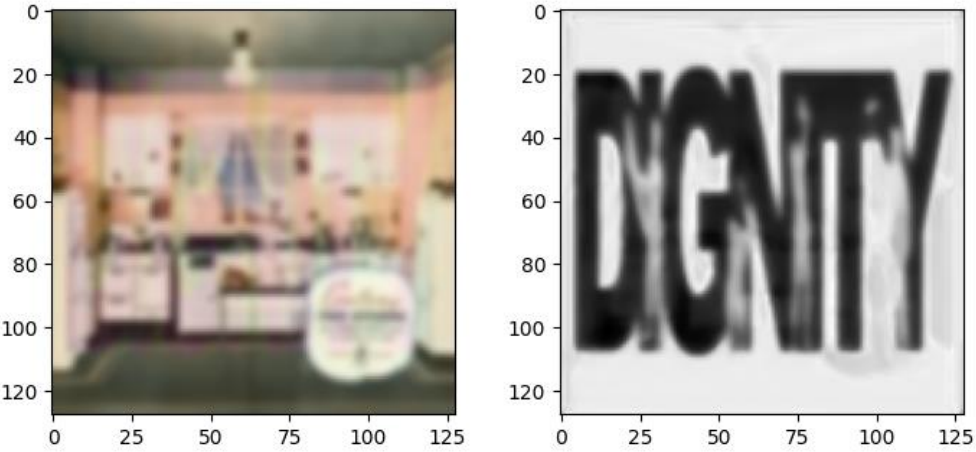

Рисунок 18 – Извлечение водяного знака после размытия по Гауссу

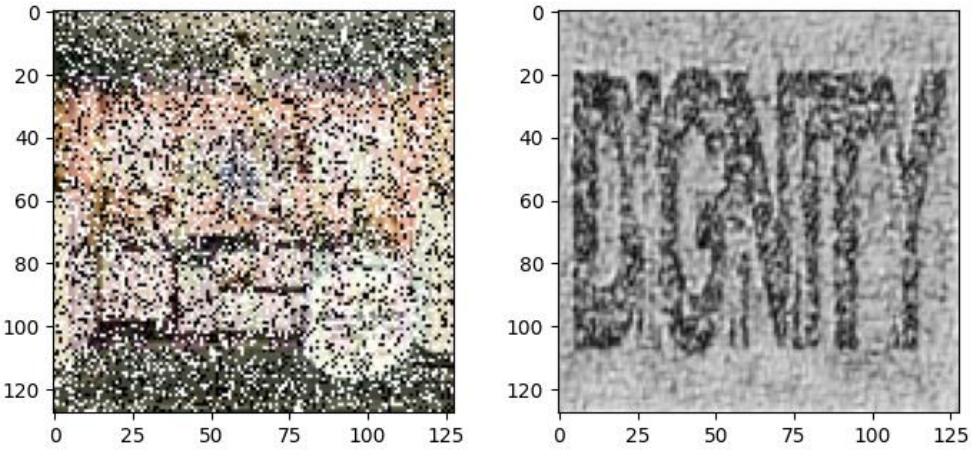

Рисунок 19 – Извлечение водяного знака после нанесения шума соли и перца

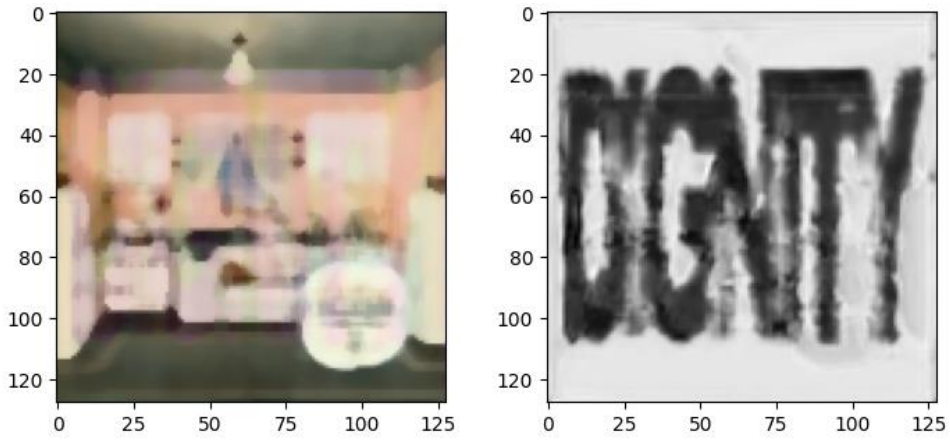

Рисунок 20 – Извлечение водяного знака после применения медианного фильтра

Для проверки параметры искажений были увеличены относительно тех, что использовались при обучении. Как видно, даже при потере практически половины изображения, еще можно различить водяной знак. Показатели для сравнения показаны в таблице 8.

	Размытие по Гауссу (9)	Кадрирование (0,4)	Выпадение пикселей (0,45)	Сжатие jpeg (60)	Шум соли и перца (0,45)	Медианный фильтр (5)
Первая реализация	38,41	38,39	37,98	–	–	–
[18]	36,24	37,06	35,64	34,17	–	–
Конечная реализация	41,83	41,57	41,49	41,69	41,43	41,77

Таким образом, полученная схема имеет хорошие показатели устойчивости. Однако здесь не рассматриваются некоторые распространенные атаки, такие как масштабирование и вращение.

Вывод

На данный момент защита авторских прав и подлинности изображений имеет высокую значимость из-за развития Интернета и легкости распространения медиаданных. Это подчеркивается многочисленными исследованиями по теме цифровых водяных знаков. За последнее время многие из них были связаны с нейросетями. Это универсальный инструмент, который можно использовать для решения разных задач, в том числе и в рассматриваемой области. Изученные статьи показывают, что использование нейросетей для создания цифровых водяных знаков – актуальная тема, которая будет продвигаться дальше.

Так как область защиты информации все время развивается, она требует постоянного обновления уже существующих методов или создания новых. Данная работа вносит вклад в развитие защиты авторских прав модернизацией схемы встраивания цифровых водяных знаков.

Список литературы

1. Evsutin, O. Digital Steganography and Watermarking for Digital Images: A Review of Current Research Directions / O. Evsutin, A. Melman, R. Meshcheryakov // IEEE Access. — 2020. — Vol. 8. — P. 166589-166611.
2. A comprehensive survey of state-of-art techniques in digital watermarking / S. Allwadhi [et al.] // 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N). — 2022.
3. A comprehensive survey on robust image watermarking / W. Wan [et al.] // Neurocomputing. — 2022. — Vol. 488. — P. 226-247.
4. Color image zero-watermarking based on fast quaternion generic polar complex exponential transform / H. Yang [et al.] // Signal Processing: Image Communication. — 2020.
5. Mahto, D. K. A survey of color image watermarking: State-of-the-art and research directions / D. K. Mahto, A. K. Singh // Computers & Electrical Engineering. — 2021. — Vol. 93. — P. 107255.
6. A comprehensive survey on robust image watermarking / W. Wan [et al.] // Neurocomputing. — 2022. — Vol. 488. — P. 226-247.
7. Sarvamangala, D. R. Convolutional neural networks in medical image understanding: a survey / D. R. Sarvamangala, R. V. Kulkarni // Evolutionary Intelligence. — 2021. — Vol. 15, iss. 1. — P. 1-22.
8. Ghoddusi, H. Machine learning in energy economics and finance: A review / H. Ghoddusi, G. G. Creamer, N. Rafizadeh // Energy Economics. — 2019. — Vol. 81. — P. 709-727.
9. A survey of the recent architectures of deep convolutional neural networks / A. Khan [et al.] // Artificial Intelligence Review. — 2020. — Vol. 53, iss. 8. — P. 5455-5516.
10. Das, S. N. Digital Image Watermarking Techniques Using Machine Learning—A Comprehensive Survey / S. N. Das, M. Panda // Lecture Notes in Networks and Systems. — 2022. — P. 455-467.
11. Abdelhakim, A. M. A time-efficient optimization for robust image

watermarking using machine learning / A. M. Abdelhakim, M. Abdelhakim // Expert Systems with Applications. — 2018. — Vol. 100. — P. 197-210.

12. Tavakoli, A. Convolutional neural network-based image watermarking using discrete wavelet transform / A. Tavakoli, Z. Honjani, H. Sajedi // International Journal of Information Technology. — 2023.
13. Wavelet-Based CNN for Robust and High-Capacity Image Watermarking / J. Lu [et al.] // 2022 IEEE International Conference on Multimedia and Expo (ICME). — 2022.
14. A fast and efficient image watermarking scheme based on Deep Neural Network / S. Mellimi [et al.] // Pattern Recognition Letters. — 2021. — Vol. 151. — P. 222-228.
15. Mahto, D. K. Hybrid optimisation-based robust watermarking using denoising convolutional neural network / D. K. Mahto, A. Anand, A. K. Singh // Soft Computing. — 2022. — Vol. 26, iss. 16. — P. 8105-8116.
16. Hu, H. Blind color image watermarking incorporating a residual network for watermark denoising and super-resolution reconstruction / H. Hu, L. Hsu // Soft Computing. — 2022. — Vol. 27, iss. 2. — P. 917-934.
17. Hsu, L. A high-capacity QRD-based blind color image watermarking algorithm incorporated with AI technologies / L. Hsu, H. Hu, H. Chou // Expert Systems with Applications. — 2022. — Vol. 199. — P. 117134.
18. Zhang, B. Embedding Guided End-to-End Framework for Robust Image Watermarking / B. Zhang, Y. Wu, B. Chen // Security and Communication Networks. — 2022. — Vol. 2022. — P. 1-11.
19. Two-Stage Robust Reversible Image Watermarking Based on Deep Neural Network / J. Huang [et al.] // Advances in Artificial Intelligence and Security. — 2022. — P. 325-335.
20. A screen-shooting resilient document image watermarking scheme using deep neural network / S. Ge [et al.] // IET Image Processing. — 2022. — Vol. 17, iss. 2. — P. 323-336.

21. Robust zero-watermarking scheme based on a depthwise overparameterized VGG network in healthcare information security / T. Huang [et al.] // Biomedical Signal Processing and Control. — 2023. — Vol. 81. — P. 104478.
22. An invisible and robust watermarking scheme using convolutional neural networks / G. Liu [et al.] // Expert Systems with Applications. — 2022. — Vol. 210. — P. 118529.
23. Microsoft COCO: Common Objects in Context / T. Lin [et al.] // Computer Vision – ECCV 2014. — 2014. — P. 740-755.
24. Logo-2K+: A Large-Scale Logo Dataset for Scalable Logo Classification / J. Wang [et al.] // Proceedings of the AAAI Conference on Artificial Intelligence. — 2020. — Vol. 34, iss. 04. — P. 6194-6201.
25. Ajit, A. A Review of Convolutional Neural Networks / A. Ajit, K. Acharya, A. Samanta // 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE). — 2020.

Приложение 1

Исходный код программы

```
# Подключение библиотек
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from PIL import Image
import torch
from torch import nn, optim
from torch.utils.data import Dataset
from torchvision import datasets
import torchvision.transforms as transforms
from typing import Type, Union
from IPython.display import clear_output, display
from ipywidgets import Output
from tqdm.auto import trange
from numpy.random import randint
import os
import zipfile
from torch.utils.data import Dataset
from torchvision import datasets

%matplotlib inline

# Инициализация значений генератора случайных чисел
np.random.seed(14)
torch.manual_seed(14)

device = "cuda" if torch.cuda.is_available() else "cpu"

# Скопировать датасет с гугл-диска
!cp /content/drive/MyDrive/dataset.zip /content

# Разархиваровать zip архив
fantasy_zip = zipfile.ZipFile('dataset.zip')
fantasy_zip.extractall('dataset')
fantasy_zip.close()

# Класс преобразования контейнера
class PreprocessImage:
    # Получение информации о границах изображения
    def edge_information(self, image):
        img_np = np.array(image*255).transpose(1, 2, 0).astype(np.uint8)
        canny = cv.Canny(img_np,100,200)
        tau = 2
        edge = (canny + 1) / tau
        edge = np.exp(edge * (-1))
        return torch.from_numpy(edge)
```

```
    # Получение информации о цветности изображения
    def chrominance_information(self, image):
        new_img = image #* 255
        y = 0.299 * new_img[0] + 0.587 * new_img[1] + 0.114 * new_img[2]
        cb = 0.564*(new_img[2] - y)
        cr = 0.713*(new_img[0] - y)
        teta = 0.25
        cb_norm = torch.square(cb)
        cr_norm = torch.square(cr)
        chrominance = (cb_norm + cr_norm) / (teta ** 2) * (-1)
        chrominance = torch.exp(chrominance) * (-1) + 1
        return chrominance

    # Преобразование изображения
    def preprocess_cover(self, image):
        img_norm = torch.zeros(image.size())
        chrominance = self.chrominance_information(image)
        edge = self.edge_information(image)
        know = (chrominance + edge) / 2
        img_norm[0] = image[0] + know - 1
        img_norm[1] = image[1] + know - 1
        img_norm[2] = image[2] + know - 1
        return img_norm

# Класс датасета
class ImageDataset(Dataset):
    # Инициализация переменных
    def __init__(self, path):
        self.path = path
        self.cover_files = os.listdir(f'{self.path}/covers')
        self.logo_files = os.listdir(f'{self.path}/logo')
        self.transform =  transforms.Compose([transforms.ToTensor()])
        self.preprocess = PreprocessImage()

    # Длина датасета
    def __len__(self):
        return len(self.cover_files)

    # Получение элемента датасета
    def __getitem__(self, idx):
        cover_path = self.cover_files[idx]
        logo_path = self.logo_files[idx]
        cover =
Image.open(f'{self.path}/covers/{cover_path}').convert('RGB')
        logo = Image.open(f'{self.path}/logo/{logo_path}')
        cover = self.transform(cover)
        logo = self.transform(logo)
        cover_norm = self.preprocess.preprocess_cover(cover)
        return cover, logo, cover_norm
```

```
# Инициализация тестовой и обучающей выборки
train_dataset = ImageDataset('dataset/train')
test_dataset = ImageDataset('dataset/test')

train_dataloader = torch.utils.data.DataLoader(
    train_dataset, batch_size=16, shuffle=True, num_workers=0
)
test_dataloader = torch.utils.data.DataLoader(
    test_dataset, batch_size=16, shuffle=False, num_workers=0
)

# Класс кодировщика
class Encoder(nn.Module):
    # Инициализация слоев нейросети
    def __init__(self):
        super(Encoder, self).__init__()

        self.conv1_watermark = nn.Conv2d(in_channels=1, out_channels=16,
kernel_size=3, padding=1)
        self.conv2_watermark = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)
        self.conv3_watermark = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)
        self.conv4_watermark = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)
        self.conv5_watermark = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)
        self.conv6_watermark = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)
        self.conv7_watermark = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)

        self.conv1_cover = nn.Conv2d(in_channels=3, out_channels=16,
kernel_size=3, padding=1)
        self.conv2_cover = nn.Conv2d(in_channels=32, out_channels=16,
kernel_size=3, padding=1)
        self.conv3_cover = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)
        self.conv4_cover = nn.Conv2d(in_channels=32, out_channels=16,
kernel_size=3, padding=1)
        self.conv5_cover = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)
        self.conv6_cover = nn.Conv2d(in_channels=32, out_channels=16,
kernel_size=3, padding=1)
        self.conv7_cover = nn.Conv2d(in_channels=16, out_channels=16,
kernel_size=3, padding=1)
        self.conv8_cover = nn.Conv2d(in_channels=35, out_channels=64,
kernel_size=3, padding=1)
        self.conv9_cover = nn.Conv2d(in_channels=64, out_channels=128,
kernel_size=3, padding=1)
```

```
        self.conv9_1_cover = nn.Conv2d(in_channels=128, out_channels=256,
kernel_size=3, padding=1)
        self.conv9_2_cover = nn.Conv2d(in_channels=256, out_channels=128,
kernel_size=3, padding=1)
        self.conv10_cover = nn.Conv2d(in_channels=128, out_channels=64,
kernel_size=3, padding=1)
        self.conv11_cover = nn.Conv2d(in_channels=64, out_channels=32,
kernel_size=3, padding=1)
        self.conv12_cover = nn.Conv2d(in_channels=32, out_channels=16,
kernel_size=3, padding=1)
        self.conv13_cover = nn.Conv2d(in_channels=16, out_channels=3,
kernel_size=3, padding=1)

        self.activator = nn.ReLU()

    # Структура нейросети
    def forward(self, input):
        (cover, watermark, cover_orig) = input

        watermark = self.conv1_watermark(watermark)
        cover = self.conv1_cover(cover)

        cover = torch.cat([cover, watermark], 1)

        watermark = self.conv2_watermark(watermark)
        watermark = self.conv3_watermark(watermark)
        cover = self.conv2_cover(cover)
        cover = self.conv3_cover(cover)

        cover = torch.cat([cover, watermark], 1)

        watermark = self.conv4_watermark(watermark)
        watermark = self.conv5_watermark(watermark)
        cover = self.conv4_cover(cover)
        cover = self.conv5_cover(cover)

        cover = torch.cat([cover, watermark], 1)

        watermark = self.conv6_watermark(watermark)
        watermark = self.conv7_watermark(watermark)
        cover = self.conv6_cover(cover)
        cover = self.conv7_cover(cover)

        cover = torch.cat([cover, watermark, cover_orig], 1)

        cover = self.conv8_cover(cover)
        cover = self.activator(self.conv9_cover(cover))
        cover = self.activator(self.conv9_1_cover(cover))
        cover = self.activator(self.conv9_2_cover(cover))
        cover = self.activator(self.conv10_cover(cover))
```

```
        cover = self.activator(self.conv11_cover(cover))
        cover = self.activator(self.conv12_cover(cover))
        cover = self.conv13_cover(cover)

        return cover

# Класс декодера
class Decoder(nn.Module):
    # Инициализация слоев нейросети
    def __init__(self):
        super(Decoder, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16,
kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32,
kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(in_channels=64, out_channels=128,
kernel_size=3, padding=1)
        self.conv5 = nn.Conv2d(in_channels=128, out_channels=64,
kernel_size=3, padding=1)
        self.conv6 = nn.Conv2d(in_channels=64, out_channels=32,
kernel_size=3, padding=1)
        self.conv7 = nn.Conv2d(in_channels=32, out_channels=16,
kernel_size=3, padding=1)
        self.conv8 = nn.Conv2d(in_channels=16, out_channels=1,
kernel_size=3, padding=1)

        self.bn1 = nn.BatchNorm2d(16)
        self.bn2 = nn.BatchNorm2d(32)
        self.bn3 = nn.BatchNorm2d(64)
        self.bn4 = nn.BatchNorm2d(128)
        self.bn5 = nn.BatchNorm2d(64)
        self.bn6 = nn.BatchNorm2d(32)
        self.bn7 = nn.BatchNorm2d(16)

        self.activator = nn.ReLU()

    # Структура нейросети
    def forward(self, input):
        output = self.activator(self.bn1(self.conv1(input)))
        output = self.activator(self.bn2(self.conv2(output)))
        output =
self.activator(self.bn3(self.conv3(output)))
        output = self.activator(self.bn4(self.conv4(output)))
        output = self.activator(self.bn5(self.conv5(output)))
        output = self.activator(self.bn6(self.conv6(output)))
        output = self.activator(self.bn7(self.conv7(output)))
        output = self.conv8(output)
```

```
        return output

# Класс дискриминатора
class Discriminator(nn.Module):
    # Инициализация слоев нейросети
    def __init__(self):
        super(Discriminator, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16,
kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32,
kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(in_channels=64, out_channels=128,
kernel_size=3, padding=1)
        self.conv5 = nn.Conv2d(in_channels=128, out_channels=256,
kernel_size=3, padding=1)

        self.bn1 = nn.BatchNorm2d(16)
        self.bn2 = nn.BatchNorm2d(32)
        self.bn3 = nn.BatchNorm2d(64)
        self.bn4 = nn.BatchNorm2d(128)
        self.bn5 = nn.BatchNorm2d(256)

        self.activator = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.pool = nn.AvgPool2d(2)
        self.fc = nn.Linear(256 * 64 * 64, 1)

    # Структура нейросети
    def forward(self, input):
        output = self.activator(self.bn1(self.conv1(input)))
        output = self.activator(self.bn2(self.conv2(output)))
        output = self.activator(self.bn3(self.conv3(output)))
        output = self.activator(self.bn4(self.conv4(output)))
        output = self.activator(self.bn5(self.conv5(output)))
        output = self.pool(output)
        output = output.view(-1, 256 * 64 * 64)
        output = self.fc(output)
        output = self.sigmoid(output)

        return output

# Класс симулятора атаки
class Attack:
    # Размытие по Гауссу
    def gaussian(self, image, p=3):
        transform_gaussian = transforms.Compose([
```

```
            transforms.GaussianBlur(p)
        ])
        return transform_gaussian(image)

    # Кадрирование
    def cropping(self, image):
        crop = torch.ones(image.size()).to(device)
        a = randint(0,crop.shape[1]-40)
        c = randint(0,crop.shape[2]-40)
        crop[:,a:a+40,c:c+40] = 0
        return image * crop

    # Выбивание пикселей
    def dropout(self, image, p=0.15):
        mask = np.random.choice([0,1],image.size()[1:],True,[p,1-p])
        mask = torch.from_numpy(mask).to(device)
        return image[:] * mask

        # Выбивание пикселей
    def salt(self, image, p=0.2):
        salt = np.random.choice([0,1],image.size()[1:],True,[p/2,1-p/2])
        peper = np.random.choice([0,1],image.size()[1:],True,[1-p/2,p/2])
        salt = torch.from_numpy(salt).to(device)
        peper = torch.from_numpy(peper).to(device)
        return image[:] * salt + peper

    def medianFilter(self, image, p = 5):
        img_np = np.asarray(image.cpu().detach()).transpose(1,2,0)
        img_bl = cv.medianBlur(img_np, p)
        return transforms.ToTensor()(img_bl)

    def jpg(self, image, p=90):
        img_np = np.asarray(image.cpu().detach()*255).transpose(1,2,0)
        encode_param = [int(cv.IMWRITE_JPEG_QUALITY), p]
        result, encimg = cv.imencode('.jpg', img_np, encode_param)
        decimg = cv.imdecode(encimg, 1)
        return transforms.ToTensor()(decimg)

    # Случайная атака
    def random_attack(self, image):
        attack = randint(0,7)
        if attack == 1:
            return self.gaussian(image)
        elif attack == 2:
            return self.cropping(image)
        elif attack == 3:
            return self.dropout(image)
        elif attack == 4:
            return self.salt(image)
        elif attack == 5:
```

```
            return self.medianFilter(image)
        elif attack == 6:
            return self.jpg(image)
        return image

# Класс автокодировщика
class AutoEncoder(nn.Module):
    # Инициализация переменных
    def __init__(self) -> None:
        super().__init__()

        self.encoder = Encoder()
        self.decoder = Decoder()
        self.discriminator = Discriminator()
        self.attack_class = Attack()
        self.alfa = 0.5
        self.beta = 0.5
        self.sigma = 0.001
        self.criterion = nn.MSELoss()

    # Кодирование
    def encode(self, x, y, z):
        return self.encoder((x,y,z))

    # Декодирование
    def decode(self, x):
        return self.decoder(x)

    # Проверка наличия водяного знака
    def discriminate(self, x):
        return self.discriminator(x)

    # Проведение атаки на изображения
    def attack(self, batch):
        noise_batch = torch.ones(batch.size()).to(device)
        for i in range(batch.size()[0]):
            noise_batch[i] = self.attack_class.random_attack(batch[i])
        return noise_batch

    # Вычмсление ошибки модели
    def compute_loss(
        self,
        cover: torch.Tensor,
        watermark: torch.Tensor,
        cover_norm: torch.Tensor
    ) -> torch.Tensor:

        encode_image = self.encode(cover_norm, watermark, cover)
        is_watermark = self.discriminate(encode_image)
```

```
        encode_loss = self.criterion(cover,encode_image)
        discriminate_loss = - torch.log(is_watermark + 0.0001).mean()

        noise_image = self.attack(encode_image)
        decode_image = self.decode(encode_image)
        not_watermark = self.discriminate(cover)

        decode_loss = self.criterion(watermark,decode_image)
        discriminate_loss = discriminate_loss - torch.log(1 -
not_watermark + 0.0001).mean()
        loss = self.alfa * encode_loss + self.beta * decode_loss +
self.sigma * discriminate_loss

        return loss

# Обучение одной эпохи
def train_epoch(
    model: nn.Module,
    train_dataloader: torch.utils.data.DataLoader,
    optimizer: torch.optim.Optimizer,
    number,
    verbose_num_iters: int = 32,
    device: torch.device = "cuda",
) -> list[float]:
    model.to(device)
    model.train()
    epoch_loss_trace = []

    display()
    out = Output()
    display(out)

    for i, batch in enumerate(train_dataloader):
        cover, logo, cover_norm = batch
        cover = cover.to(device)
        logo = logo.to(device)
        cover_norm = cover_norm.to(device)
        loss = model.compute_loss(cover, logo, cover_norm)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        epoch_loss_trace.append(loss.item())

        if (i + 1) % verbose_num_iters == 0:
            with out:
                clear_output(wait=True)

                plt.figure(figsize=(10, 5))
                plt.subplot(1, 2, 1)
                plt.title(f"Current epoch loss: {number}", fontsize=22)
```

```
                plt.xlabel("Iteration", fontsize=16)
                plt.ylabel("Reconstruction loss", fontsize=16)
                plt.grid()
                plt.plot(epoch_loss_trace)
                plt.show()

    out.clear_output()

    return epoch_loss_trace

# Обучение модели
def train_model(
    model: nn.Module,
    train_dataloader: torch.utils.data.DataLoader,
    optimizer: torch.optim.Optimizer,
    num_epochs: int = 5,
    verbose_num_iters: int = 32,
    device: torch.device = "cuda"
) -> None:
    loss_trace = []
    epoch_number = 1
    for epoch in trange(num_epochs, desc="Epoch: ", leave=True):
        epoch_loss_trace = train_epoch(
            model=model,
            train_dataloader=train_dataloader,
            optimizer=optimizer,
            number = epoch_number,
            verbose_num_iters=verbose_num_iters,
            device=device,
        )

        loss_trace += epoch_loss_trace
        epoch_number += 1

    plt.figure(figsize=(10, 5))
    plt.title("Total training loss", fontsize=22)
    plt.xlabel("Iteration", fontsize=16)
    plt.ylabel("Reconstruction loss", fontsize=16)
    plt.grid()
    plt.plot(loss_trace)
    plt.show()

    model.eval()

model = AutoEncoder()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
train_model(model, train_dataloader, optimizer, 15, device=device)
```