**Федеральное государственное автономное образовательное учреждение**

**высшего образования   
«Национальный исследовательский университет   
«Высшая школа экономики»**

**Московский институт электроники и математики им. А.Н.Тихонова**

Направление подготовки

**«10.03.01 Информационная безопасность»**

Образовательная программа **«Информационная безопасность»**

**О Т Ч Е Т**

**о прохождении**

**учебной практики**

**Студент Ярославский А. В. БИБ211**

(Фамилия И.О.) номер группы

**Руководитель практики студента:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Зав. каф. информационной безопасности киберфизических систем департамента электронной инженерии МИЭМ НИУ ВШЭ |  | Евсютин О.О. |  |  |
| должность и место работы |  | Фамилия И.О. |  | подпись |

**Руководитель практики от НИУ ВШЭ:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Зав. каф. информационной безопасности киберфизических систем департамента электронной инженерии МИЭМ НИУ ВШЭ |  | Евсютин О.О. |  |  |
| должность и место работы |  | Фамилия И.О. |  | подпись |

**Практика пройдена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва, 2023**

СОДЕРЖАНИЕ

[1 Введение 3](#_Toc130586174)

[2 Краткая характеристика организации 4](#_Toc130586175)

[3 Описание профессиональных задач 5](#_Toc130586176)

[4 Исполненное индивидуальное задание 6](#_Toc130586177)

[4.1 Принцип работы встраивания информации в контейнеры 6](#_Toc130586178)

[4.2 Используемый алгоритм встраивания 8](#_Toc130586179)

[4.3 Понятие стегоанализа 9](#_Toc130586180)

[4.4 Основной принцип работы нейронных сетей 11](#_Toc130586181)

[4.4.1 Что такое нейросеть 11](#_Toc130586182)

[4.4.2 Классификация нейросетей 13](#_Toc130586183)

[4.4.3 Задачи нейронных сетей 13](#_Toc130586184)

[4.5 Основной принцип работы сверточных нейронных сетей 13](#_Toc130586185)

[4.5.1 Работа сверточных слоев 14](#_Toc130586186)

[4.5.2 Работа субдискретизирующего слоя 15](#_Toc130586187)

[4.6 Обучение нейронной сети 16](#_Toc130586188)

[4.7 Описание нейронной сети из статьи 18](#_Toc130586189)

[4.7.1 Обучение нейронной сети 19](#_Toc130586190)

[4.8 Результаты обучения 19](#_Toc130586191)

[5 Заключение 21](#_Toc130586192)

[6 Список использованных источников 23](#_Toc130586193)

[ПРИЛОЖЕНИЕ А 24](#_Toc130586194)

# 1 Введение

Целью прохождения практики является закрепление, расширение и углубление полученных теоретических знаний и приобретение первоначальных практических навыков в решении конкретных проблем.

Задачи практики:

- закрепление и углубление теоретических знаний по прослушанным за время обучения в университете дисциплинам;

- формирование и совершенствование базовых профессиональных навыков и умений в области информационной безопасности;

- знакомство и отработка навыков работы с реальными исследовательскими, промышленными и образовательными проектами;

- формирование информационной компетентности с целью успешной работы в профессиональной деятельности;

- получение навыков самостоятельной работы, а также работы в составе коллектива;

- обработка полученных материалов и оформление отчета о прохождении практики.

# 2 Краткая характеристика организации

Национальный исследовательский университет «Высшая школа экономики» — исследовательский университет, осуществляющий свою миссию через научно- образовательную, проектную, экспертно-аналитическую и социокультурную деятельности на основе международных научных и организационных стандартов. Университет — это коллектив ученых, сотрудников, аспирантов и студентов, которых отличает внутренняя приверженность к поддержанию высоких академических стандартов своей деятельности. ВУЗ стремимся обеспечить наиболее благоприятные условия для развития каждого члена нашего коллектива [1].

МИЭМ НИУ ВШЭ готовит специалистов, бакалавров и магистров в области электроники, информационных технологий, телекоммуникаций, вычислительной техники, прикладной математики, кибернетики и дизайна. Педагогический коллектив МИЭМ включает 1 академика РАН, 4 члена-корреспондента РАН, 34 лауреата государственных премий РФ. Тесные связи с ведущими отраслевыми институтами, институтами РАН, мировыми компаниями, такими как National Instruments, InfoWatch, Zyxel, QNAP, Altium Limited, а также оснащенные новейшим оборудованием лаборатории: 3D-визуализации; лазерных технологий; телекоммуникации; кибербезопасности — позволяют готовить востребованных специалистов на самом высоком уровне [2].

# 3 Описание профессиональных задач

Задачи практики:

1. Закрепление и углубление теоретических знаний по прослушанным за время обучения в университете дисциплинам.

2. Формирование и совершенствование базовых профессиональных навыков и умений в области информационной безопасности;

3. Знакомство и отработка навыков работы с реальными исследовательскими, промышленными и образовательными проектами;

4. Формирование информационной компетентности с целью успешной работы в профессиональной деятельности

5. Получение навыков самостоятельной работы, а также работы в составе коллектива;

6. обработка полученных материалов и оформление отчета о прохождении практики.

Содержание практики (вопросы, подлежащие изучению):

1. Знакомство с проблемной областью встраивания дополнительной информации в цифровые объекты

2. Изучение и программная реализация алгоритма стегоанализа цифровых изображений, основанного на машинном обучении

3. Оценка эффективности реализованного алгоритма

Планируемые результаты:

1. Программная реализация алгоритма стегоанализа цифровых изображений, основанного на машинном обучении

2. Результаты вычислительных экспериментов по оценке эффективности реализованного алгоритма.

# 4 Исполненное индивидуальное задание

Целью данной работы является рассмотрение работ по теме стегоанализа цифровых изображений с использованием методов глубокого машинного обучения, а также изложение оригинальных результатов в плане использования глубоких нейронных сетей сверточного типа для обработки цветных изображений большого размера и их сравнение с результатами, полученными другими авторами.

Задача данной работы – изучение алгоритмов стегоанализа на основе глубоких сверточных сетей, ориентированных на анализ цветных изображений большого размера. Идея предлагаемого подхода состоит в проведении обучения сетей относительно простой архитектуры на небольших фрагментах размером 32×32, 64×64, 128×128 для реализации последовательной вторичной обработки совокупности результатов классифкации, выполненной с использованием ранее обученного нейросетевого классификатора на блоках целостного изображения большого размера, с целью принятия окончательного решения.

С целью понятного изложения материала был подготовлен план, по которому и будет происходить изучение данной темы:

1. Введение, принцип работы встраивания информации в контейнеры.

2. Используемый алгоритм встраивания.

3. Cтегоанализ определение.

4. Основной принцип работы нейронных сетей.

5. Основной принцип работы сверточных нейронных сетей.

6. Обучение нейронной сети.

7. Описание написанной нейронной сети из статьи.

8. Результаты обучения.

9. Выводы.

Каждый пункт раскрывает определенную тему машинного обучения и стеганографии.

## **4.1 Принцип работы встраивания информации в контейнеры**

Стеганография – метод сокрытия информации, при котором остается в тайне сам факт ее передачи. Как правило, используются различные методы, которые позволяют скрыть сообщения в других объектах, например, в картинках, звуковых файлах, видео. Принцип работы стеганографии заключается в том, что сообщение «встроено» в другой объект таким образом, что его наличие никак не заметно. Чтобы раскрыть стеганографическое сообщение, необходимо знать точный метод, которым оно было закодировано. Существуют два принципа, на которых базируется компьютерная стеганография: первый из них – цифровые носители данных, такие как фотография, звук или видео, могут в некой степени видоизменяться без потери их функциональности, второй – неспособность органов чувств человека распознать незначительные изменения цифровых данных.

Стегосистема – совокупность средств и методов, используемые для формирования скрытого канала передачи информации, а также способы извлечения этих данных из контейнеров. Стегосистемы могут быть использованы для различных целей, например, для защиты конфиденциальной информации, а также для скрытой передачи информации без привлечения внимания сторонних лиц. Существуют различные типы стегосистем, от простых программ, которые могут скрыть текстовое сообщение в картинке, до сложных систем, которые могут скрыть информацию в видеофайлах или потоках данных. Некоторые из них используют множество методов шифрования, чтобы сделать стегосообщение более надежным и труднодоступным для раскрытия. Можно рассмотреть общую модель стегосистемы, в соответствии с рисунком 1:

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1 – Модель стегосистемы.

Контейнер – некий цифровой объект, нужный для встраивания сообщения. В качестве сообщения может использоваться любой вид информации: текст, изображение, аудиосигнал. Стегоканал – канал передачи встроенного секретного сообщения. Ключ – ключ, предназначенный для сокрытия сообщения в контейнере.

Контейнеры бывают трех основных видов: визуальные, цифровые и звуковые. Визуальные контейнеры представляют из себя картинку или фотографию, в которой для встраивания сообщений используются небольшие изменения яркости заранее определённых точек растра изображения. Звуковые контейнеры представляет собой речевой или музыкальный сигнал, в котором для встраивания сообщений используются младшие биты аудиосигнала, что практически не отражается на качестве звука. Текстовый контейнер – текстовый файл, подготовленный к печати на принтере, в котором для встраивания сообщений используются небольшие изменения стандартов печати.

## **4.2 Используемый алгоритм встраивания**

Алгоритм WOW (Wavelet Obtained Weights) является стеганографическим методом для более эффективного скрытия информации в изображениях, который использует дискретное вейвлет-преобразование.

Вейвлет-преобразование – это математический метод, используемый для анализа сигналов и изображений. Он основан на представлении сигнала в виде суммы вейвлет-функций, которые являются короткими, ограниченными по времени волнами. Используя вейвлет-функции, можно анализировать как локальные, так и глобальные свойства сигнала, например, определение границ объектов на изображениях или распознавании паттернов в сигналах.

Каждая вейвлет-функция имеет свой уникальный спектр, который позволяет его использовать для анализа различных свойств сигнала или изображения.

Вейвлет-преобразование может быть использовано для сжатия изображений и сигналов, фильтрации шумов, анализа статистических свойств и определения основных компонент сигнала или изображения.

Для этого алгоритм использует набор направленных высокочастотных фильтров , которые позволяют получить направленные остатки, связанные с предсказуемостью пикселя в определенном направлении.

Затем измеряется влияние внедрения информации на каждую направленную остаточную нагрузку, и эти воздействия агрегируются для вычисления стоимости внедрения. Алгоритм сохраняет высокую стоимость внедрения в гладких областях и вдоль краев, где содержание предсказуемо по крайней мере в одном направлении, и низкую стоимость внедрения в текстурированных или шумных зонах, где содержание непредсказуемо в каждом направлении.

Такой подход делает алгоритм очень адаптивным и лучше сопротивляется стеганализу. Внедрение информации происходит в специально определенные места, а после внедрения происходит небольшое изменение окрестности точки изображения для большей скрытности алгоритма.

Вложение обрабатывается аддитивными искажениями в формуле 1:

(1)

где 𝜌𝑖𝑗 - затраты на изменение пикселя 𝑋𝑖𝑗 на 𝑌𝑖𝑗.

Аддитивность – свойство алгоритма, которое означает, что он не учитывает взаимное влияние отдельных изменений при внедрении информации. Если затраты на каждый пиксель в изображении известны, то внедрение последовательности бит с минимальным искажением эквивалентно исходному кодированию с учетом точности. Это означает, что алгоритм не будет учитывать взаимное влияние каждого бита на другие биты при их внедрении, а только их индивидуальные затраты на каждый пиксель.

Чем более текстурированы или шумны области изображения, тем труднее модифицировать их в любом направлении, и, следовательно, они предпочтительнее для встраивания информации. Для оценки гладкости изображения в нескольких направлениях в алгоритме WOW используется набор фильтров высоких частот, который позволяет получить направленные остаточные нагрузки, связанные с предсказуемостью пикселей в определенном направлении. Эти остаточные нагрузки затем используются для оценки влияния внедрения на каждое направление и соответствующей агрегации воздействий, чтобы определить стоимость внедрения.

Визуализацию работы данного алгоритма можно увидеть на рисунке 2:

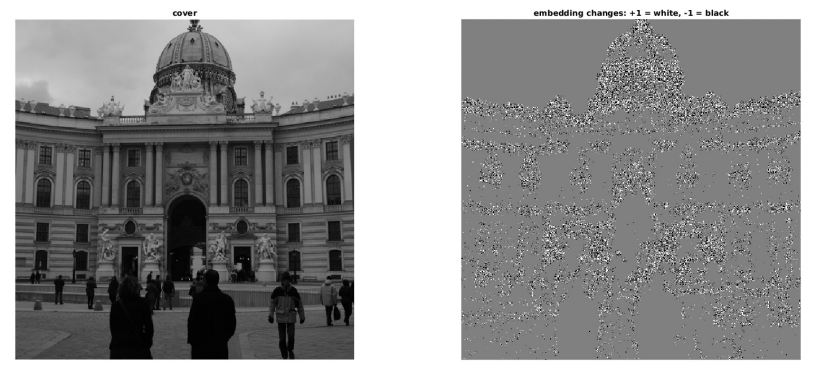


Рисунок 2 – Пример работы алгоритма WOW. Слева – исходное изображение, справа – маска, накладываемая на исходное изображение.

Алгоритм WOW обладает рядом преимуществ, например, он может быть успешно применен к изображениям с высоким разрешением, а также основан на статистическом анализе весов матриц, что делает невозможным обнаружение наличия вставленной информации без знания ключа для ее извлечения. Однако, как и любой метод стеганографии, WOW не является идеальным средством для защиты секретной информации, и может быть обнаружен с использованием специализированных программ и технологий.

## **4.3 Понятие стегоанализа**

Стегоанализ является разделом стеганографии, который направлен на выявление факта встраивания данных в контейнеры. Многие стегосистемы разработаны с целью обойти различные методы стегоанализа, что делает установление наличия стеганографических сообщений более сложным и требующим специальных знаний и компетенций. Задача стегоанализа может решаться в прямой постановке, то есть выявление факта встраивания информации с неизвестным содержимым, так и в обратной постановке, то есть проверка скрытности встраивания данных в контейнеры. Проблема стегоанализа довольно популярна в настоящее время. Об этом свидетельствует огромное количество алгоритмов встраивания. Это связано с широким использованием методов и алгоритмов машинного обучения в последние 15–20 лет, как инструмента для анализа большого объема данных. В основном приводят такую классификацию методов и алгоритмов стегоаналица:

1) По степени универсальности используемых алгоритмов и вариативности контейнера выделяют специальные, которые привязаны к определенному алгоритму или контейнеру и универсальные, предназначенные стегоанализа без привязки к определенным методам или объектам встраивания.

2) По влиянию на контейнер, потенциально содержащий водяной знак, выделяют активные методы анализа изображений, которые определяют параметры внедренного сообщения и используемого алгоритма и обеспечивают, по возможности, его расшифровку или устранение; пассивные, которые направлены только на определение самого факта наличия встроенной информации.

3) По используемым принципам обработки информации методы стегоанализа разделяются на сигнатурные, в ходе которых определяется, используя визуальный и количественный анализ, нехарактерные включения рассматриваемых контейнеров; вероятностные, которые базируются на статистическом анализе, а именно необычные отклонения от статистик для обычных контейнеров; методы машинного обучения, базирующиеся на построении методов, позволяющих отнести объект к определенному классу или категории на основе анализа его признаков, называющиеся классификаторами, с использованием обучающих наборов контейнеров, разделяющиеся на заполненные и незаполненные.

Последний тип стегоанализа испытывает бурный рост в последнее время, именно поэтому мы их рассмотрим поближе. Универсальность – отличительная особенность данного типа стегоанализа. Можно разделить это направление установления факта встраивания на два потока: методы поверхностного обучения и методы, основанные на применении глубоких нейронных сетей. Данная работа направлена на изучение использования глубоких нейронных сетей в стегоанализе цифровых изображений. С этой целью стоит изучить базовое устройство нейронных сетей и сверточных нейронных сетей.

## **4.4 Основной принцип работы нейронных сетей**

### **4.4.1 Что такое нейросеть**

Приведем определение нейросети с точки зрения машинного обучения. Нейросеть – это математическая модель, берущая основную идеологию с нейронной сети живого организма. Нейросети способны заменить или дополнить работу человека в тех случаях, когда решение нужно принимать на основе предыдущего опыта. Основной задачей нейронных сетей является решение задач, не имеющих заранее известного ответа, то есть интеллектуальные. Особенность, которая выделяет эту математическую модель из других – способность к обучению, не завися от человека, а на личном опыте [4].

Минимальной частью нейронной сети является нейрон, представленный в виде числа. Данное число как-то описывает данные, которые мы подаем на вход. Например, если рассматривать изображение, как объект, который идет на вход нейронной сети, то степень освещенности отдельного пикселя, лежащая в пределе от 0 до 1, это и есть математическое представление одного нейрона.

Следующим по масштабу элементом нейронной сети является нейронный слой. По своей сути это вектор, состоящий из весов всех нейронов. В основном сеть состоит из нескольких нейронных слоев. Чем больше слоев, тем сложнее и различнее задачи может решать нейронная сеть. Существует три вида нейронных слоев:

1) Входной слой, состоящий из нейронов, характеризующие входной объект.

2) Скрытые слои, описывающие признаки входного объекта.

3) Выходной слой, дающий ответ на поставленную задачу.

Нейронные слои связаны друг с другом. Места соединения двух нейронов в живой природе называются синапсами. При прохождении через них сигнал может усиливаться, ослабевать или оставаться неизменным, что в итоге влияет на конечный результат. Одна нейронная группа возбуждается, передавая свое возбуждение другим группам нейронов, находящимся в другом нейронном слое. Аналогом природных синапсов является параметр, называемый весом, показывающий меру взаимосвязи между определенными нейронами. Если:

1) вес равен нулю, значит взаимосвязи нет

2) вес больше нуля, значит последующий нейрон усиливает принимаемый сигнал

3) вес меньше нуля, значит последующий нейрон уменьшает принимаемый сигнал

Математическое представление последующего нейрона представляется в виде линейной комбинацией весов i взаимосвязей и значения нейронов i, которые соединены с последующим нейроном в соответствии с формулой (1):

(1)

где – значение последующего нейрона,

– значение нейрона предыдущего слоя,

– вес взаимосвязи последующего нейрона с n-ым нейроном предыдущего слоя.

L может представлять из себя любое число. Однако зачастую значение нейрона в следующем слое нужно представить в определенных пределах. Для этого используется функция активации.

Функция активации – функция, вычисляющая значение нейрона на основе линейной комбинации значений связанных нейронов с их весами, и нормализует это значение в обозначенных пределах. Приведем несколько примеров функции активации:

1. Функция активации ReLU. Эта функция выводит данные напрямую, если входной слоя является положительным числом; в противном случае выводит ноль. Вычисляется по формуле (2):

(2)

где х – значение, передаваемое в функцию.

2. Сигмоида. Данный тип функции активации в настоящее время почти не используется и вычисляется по формуле (3):

(3)

где x – значение, передаваемое в функцию.

Приведем модель нейронной сети в соответствии с рисунком (3):

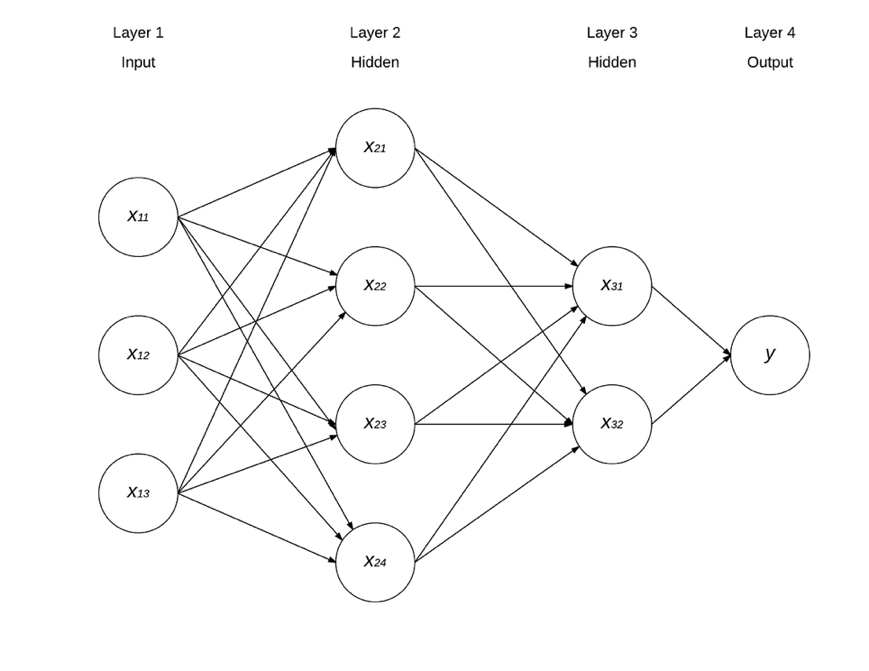


Рисунок 3 – Модель нейронной сети.

где Layer 1 – входной слой, представленный в виде 3 нейронов,

Layer 2, Layer 3 – скрытые слои с 4 и 2 нейронами соответственно,

Layer 3 – выходной слой из 1 нейрона.

После того, как мы привели полную модель нейронной сети, можно переписать определение нейрона [4]. Нейрон – функция, принимающая выходы со всех связанных нейронов и генерирующая новое число, используя функцию активации. По своей сути вся нейронная сеть тоже функция, у которой нейроны – переменные, а веса – параметры.

### **4.4.2 Классификация нейросетей**

Под определенные задачи выбирается определенный тип нейросети, которая лучше справляется с поставленными задачами:

1) По типу структуры:

1) Однослойные, которые не имею скрытых слоев.

2) Многослойные, которые имеют один или несколько скрытых нейронных слоев.

2) По направленности действий:

1) Прямого распространения, в которой сигнал передается от предыдущего к последующему до выходного слоя. Возможность вернуться к предыдущему слою отсутствует.

2) С функцией обратного распространения, что существенно расширяет функционал. Эти нейросети обладают подобием кратковременной памяти.

3) По типу контента:

1) Сверточные нейронные сети, работающие с изображениями

2) Генеративные нейронные сети, создающие изображения и звукозаписи.

### **4.4.3 Задачи нейронных сетей**

Нейронные сети используют во многих сферах жизни человека. Приводят следующую классификацию задач, решаемых нейронными сетями:

1) Классификация. Применяется для группировки объектов по определенным признакам, например, населения города.

2) Предсказание. Применяются для прогнозирования валютного курса, цен акций на бирже, определения погоды, реакции на лечение.

3) Распознавание. Применяется в основном для детекции объектов, расположенных на изображении.

## **4.5 Основной принцип работы сверточных нейронных сетей**

Сверточная нейронная сеть – яркий пример классификатора, упрощенный аналог зрительной коры головного мозга, основанный на идеи приёма изображения и вывода его класса. Сверточные нейронные сети широко применяются в компьтерном зрении, в таких областях, как распознавание лиц. Они позволяют автоматически обрабатывать большие массивы данных и выявлять скрытые связи между признаками, делая их незаменимыми инструментами для решения различных задач. Сверточная нейронная сеть производит группировку изображений посредством поиска характеристик низшего уровня, относящихся только к определенным группам [3]. Суммируя все базовые признаки путем абстрактных концепций через группы сверточных слоев, нейронная сеть классифицирует изображение. В общем случае данная сеть состоит из тех слоев: сверточный, субдискретизирующий и полносвязный. Их порядок может быть различным и выбирается исходя из поставленных задач. Архитектура сверточной нейронной сети представлена на рисунке 4:

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 4 – Архитектура сверточной сети.

### **4.5.1 Работа сверточных слоев**

Свертка всегда является первым нейронным слоем в данном типе нейронных сетей. Свертка – операция вычисления нового значения пикселя, при котором учитываются значения рядом находящихся пикселей. Ядро свертки – центр квадратной матрицы, которая называется фильтром. Фильтр является идентификатором каких-либо свойств изображения, таких как кривых или прямых границ. Якорь свертки – элемент фильтра, который прикладывается к исходному изображению, зачастую им является ядро свертки [3]. При вычислении нового значения пикселя фильтр накладывается якорем свертки на выбранный пиксель. Вычисляется произведение накрытых фильтром пикселей и значения соответствующих пикселей фильтра. Полученные числа суммируются, это и есть новое значение выбранного пикселя изображения. Данный алгоритм проходится по всем пикселям изображения, тем самым создавая карты признаков изображения.



Допустим, у нас есть изображение . Тогда размеры фильтра -- . Якорем свертки является ядро фильтра. После свертки получится новое изображение с новым размером . Полученная матрица является картой признаков. Каждый последующий сверточный слой будет представлять свойства изображения более высокого уровня, чем просто кривые или прямые линии. На рисунке 5 представлена визуализация данного процесса:

Изображение выглядит как текст, электроника

Автоматически созданное описание

Рисунок 5 – Модель сверточного слоя.

### **4.5.2 Работа субдискретизирующего слоя**

Данный слой в основном используется после сверточного для уменьшения размерности карт признаков. Принцип работы данного слоя прост [3]:

1. Карты признаков разделяются на ячейки определенной размерности.

2. В каждой ячейке используется метод pooling, который заключается в том, чтобы заменить целую ячейку одним элементом, который будет подходить под условия метода. Данный метод имеет три режима работы: max-pooling, когда ищется максимальный элемент ячейки, min-pooling, выбирается минимальный элемент ячейки, и avg-pooling, когда целая ячейка заменятся средним значением всех элементов ячейки.

3. Из выбранных или полученных элементов формируется новая карта признаков с уменьшенным размером.

Пример работы субдискретизирующего слоя с методом max-pooling представлен на рисунке 6:

Изображение выглядит как текст, письмо

Автоматически созданное описание

Рисунок 6 – Модель субдискретизирующего слоя.

**5.3 Работа полносвязного слоя**

Данный слой идет в самом конце нейронной сети и представляет из себя вектор, состоящий из классов, по значениям которых и идет классификация. Числа в этом векторе представляют вероятности принадлежности объекта на исходном изображении к определенному классу. Эти числа получаются по формуле (1). Визуализация работы полносвязного слоя представлена на рисунке 7:

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 7 – Модель полносвязного слоя.

Однако создание нейронных слоев и архитектуры нейронной сети не является окончанием работы. Следующий шаг – обучение нейронной сети.

## **4.6 Обучение нейронной сети**

Введем несколько определений, которые сделают дальнейшие объяснения более понятными. Цель обучения нейронной сети – подобрать определенные параметры сети, такие как веса, так, чтобы процент ошибок нейронной сети стремился к нулю. Ошибка нейронной сети – отличие предсказания от правильного значения. Нейронная сеть обучается на размеченных данных, для которых заранее известны правильные значения. Эти данные называются тестовой выборкой.

Вначале обучения весам соединений нейронов задаются случайные величины. Из-за этого сеть будет работать некорректно, так как она выполняет случайные действия. Чтобы такого не происходило определяют функцию ошибки.

Функция ошибки – функция, корректирующая работу нейронной сети. Она задается для оценки работы сети. Зачастую используется квадрат разности между значениями нейронов, которые выводит сеть и реальными значениями тестовой выборки. Сумма всех таких величин – ошибка обучающего примера. Если среднее значение данной величины мало – нейронная сеть работает корректно, если, наоборот, велико – нейронная сеть выполняет случайные действия, тем самым некорректно работая. Пример функции ошибки для задачи регрессии в соответствии с формулой (4):

(4)

где n – число нейронов в выходном слое,

– реальные значения тестовой выборки,

– значение нейрона.

По своей сути, функция ошибки принимает нейроны как переменные, а веса как параметры. На вывод она подает число – насколько плохи выбранные значения нейронов и весов.

Однако определения корректности работы сети недостаточно для обучения нейронной сети. Для этого мы должны минимизировать функцию ошибки. Градиент функции показывает направление максимума в определенной точке, а значит противоположное направление градиента покажет направление минимума функции. Длина градиента указывает насколько сильно функция убывает в указанном направлении.

Алгоритм минимизации функции ошибки выглядит следующим образом:

1) Вычислить значение градиента.

2) Шагнуть в противоположное направление градиента.

3) Повторить, пока вектор градиента не станет стремиться к нулю.

Функция ошибки включает в себя среднее от всей выборки, именно поэтому минимизируя ее, тем самым улучшается работа нейронной сети на всех примерах. Данная функция должна быть гладкой, чтобы существовала возможность вычисления градиента функции. Из-за этого у нейронов тоже гладкая функция активации нейронов. Весь процесс сдвигания на отрицательный градиент – градиентный спуск, способ приблизиться к локальному минимуму. Значения вектора градиента показывают какие смещения весов окажут наибольшее влияние на функцию ошибки. Делая это рекурсивно для каждого предыдущего слоя, появляется возможность настраивать веса всей нейронной сети более гибко. Это принцип называется обратным распространением ошибки. Тогда градиент функции вычисляется по формуле (5):

(5)

где – значение функции ошибки,

– значения весов.

Компьютеру нужно много времени, чтобы учесть все изменения весов для каждого элемента выборки, чтобы потом их усреднить и поменять веса. В данном случае перемешиваются элементы выборки, и выборка делится на части, называющиеся mini-batches. Градиент, вычисленный на mini-batches сильно ускоряет процесс обучения нейронной сети. Градиентный спуск, выполненный с использованием mini-batches, называется стохастическим.

## **4.7 Описание нейронной сети из статьи**

Предложенная сверточная нейросеть содержит три линейных, три и три полносвязных слоя [5]. Первый сверточный слой используется для предобработки входного изображения размерами с помощью пространственного высокочастотного фильтра, который предоставляет возможность гибкой настройки параметров фильтра. Его размеры - . Функция активации этого слоя – гауссовская функция активации с настраиваемым среднеквадратичным отклонением описывается по формуле (6):

(6)

где – значение нейронов по всей batch-выборке,

– среднее значение нейронов по всей batch-выборке,

– среднеквадратичное отклонение.

Значение выбирается случайно по равновероятностному закону в диапазоне от 0.01 до 0.5. В дальнейшем будет везде использоваться функция активации ReLU, кроме последнего волносвязного слоя. Там используется стандартная активация Softmax. Перейдем к описанию архитектуры нейросети, изображенной на рисунке 8:

Изображение выглядит как текст, письмо

Автоматически созданное описание

Рисунок 8 – Архитектура предлагаемой нейронной сети.

На рисунке 8 использованы следующие обозначения:

Image – входной слой, принимающий на вход трехканальное изображение;

Pred\_HF – слой, в котором происходит высокочастотная фильтрация;

Conv\_1 – слой свертки с фильтром размерами , шагом 1 и 32 картами признаков на выходе;

Conv\_2 и Conv\_3 – слой свертки, в котором используется фильтр , шагом 1, 64 и 128 картами признаков соответственно;

ReLu\_2, ReLu\_3 – функция активации ReLu сверточного слоя;

Bn\_1, Bn\_2, Bn\_3 – слои стандартной batch-нормализации;

Gaussian\_1 – функция активации первого слоя свертки;

Av\_1, Av\_2, Av\_3 – слои субдискретизации, используя avg\_pooling и ячейки размером с шагом 2;

Fc\_1, Fc\_2, Fc\_3 – полносвязные слои с 128, 128 и 2 выходами соответственно;

ReLu\_Fc\_1, ReLu\_Fc\_2 – функция активации ReLu для полносвязного слоя;

Softmax – функция активации последнего слоя;

Class – слой классификации, отвечающий за вычисление функции потерь.

Таким образом Сonv\_1, Bn\_1, Gaussian\_1 и Av\_1 – образуют первый сверточный слой, Conv\_2, Bn\_2, ReLu\_2 и Av\_2 – второй сверточный слой, Conv\_3, Bn\_3, ReLu\_3 и Av\_3 – третий слой свертки. Fc\_1, ReLu\_Fc\_1 – первый полносвязный слой, Fc\_2, ReLu\_Fc\_2 – второй полносвязный слой, Fc\_3 и Softmax – третий полносвязный слой.

В связи с используемым алгоритмом встраивания WOW, который заполняет контейнер неравномерно, подбор оптимального минимального значения количества ответов «да» при стегоанализе, как и оценка вероятностей правильных и неправильных решений, приводилась экспериментально.

### **4.7.1 Обучение нейронной сети**

Исходный датасет, содержащий 10000 цветных изображений размером , был взят с интернет-ресурса [6]. Для встраивания псевдослучайной последовательности в контейнер был использован адаптивный алгоритм встраивания WOW с заполнением одного канала цветности.

Для обучения предлагаемой нейронной сети было взято 6000 изображений: 4000 изображений без встраивания и 2000 со встраиванием. Для валидации использовалось 3000 изображений: 2000 со встраиванием и 1000 без встраивания. На тестовую выборку выделялось 1000 изображений: 500 без встраивания и 500 со встраиванием.

В процессе обучения нейронной сети использовались следующие параметры: оптимизатор adamax на 15 эпохах с начальной скоростью 0.01 и параметром L2 – регуляризации 0.0001 и размером mini-batch 64 элемента. Дообучение нейронной сети не использовалось.

Написанная нейронная сеть в целом повторяет предложенную, лишь с одним отличаем. В качестве последней функцией активации была взята logSoftmax вместо Softmax. Данный выбор был сделан на основе личного опыта.

## **4.8 Результаты обучения**

В соответствии с вышеуказанными параметрами проводилось обучение и тестирование нейросети. Внедрение стегосообщений проводилось с использованием алгоритма WOW при разных значениях параметра payload, который отвечает за объем внедрения, выраженный в процентах. Приведем график 1 зависимости точности классификации валидационной и тестировочной выборки от номера эпохи:

**Изображение выглядит как диаграмма

Автоматически созданное описание**

График 1 – Поведение графика точности в процессе работы нейросети.

На графике 1 используются следующие обозначения:

Train\_accuracy – точность классификации тренировочной выборки;

Valid\_accuracy – точность валидационной выборки.

Значение параметра loss: в начале обучения – 1593.5804, в конце обучения – 0.0602. По окончании обучения точность классификации тренировочной выборки равна 0.9120, валидационной – 0.7754, а тестовой – 0.7558.

Следующим этапом анализа является влияние параметра payload и особенностей датасета на итоговую точность классификации. Под особенностями датасета понимается набор изображений, обладающими одинаковыми чертами, такими как количество границ объектов и зашумленность изображения. Такое внимание на именно эти типы изображений связано с тем, что алгоритм встраивания WOW адаптивный. Из этого следует, что он избегает встраивания стегосообщения в границы объектов, расположенных на изображении, и ищет зашумленные зоны, где и происходит встраивание. Для выяснения зависимости точности классификации от особенностей датасета, были сформированы две группы изображений: с большим количеством шума и большим количеством граней. Получили график 2:

Изображение выглядит как диаграмма

Автоматически созданное описание

График 2 – Зависимость графика точности классификации от параметра payload и выбранного датасета.

На графике 2 используются следующие обозначения:

No Border dataset – группа изображений с большим количеством шума;

Border dataset – группа изображений с большим количеством граней.

Из графика видно, что точность классификации выше при всех значениях параметра payload для изображений с большим количеством шума. Поведение графика в каждой точке для двух датасетов.

Можно сделать вывод, что степень зашумленности изображения напрямую влияет на точность классификации. Изображения с большим количеством граней не сильно влияют на точность, это видно из графика, значение точности которого при payload = 0.4 равен точности при тестовой выборке.

# 5 Заключение

Исходя из вышеописанного можно сделать следующий вывод. Нейронные сети используются почти каждой сфере человеческой жизни. Именно поэтому они не обошли тему защиты данных и активно используются во всех сферах криптографии. В изуенной статье упор делался на стегоанализ – раздел стеганографии, и применение глубоких нейронных сетей с этой целью [7].

В данном отчете детально разобран все этапы работы нейронных сетей. Анализ данной работы привел к выводу, что глубокие нейронные сети показывают хороший результат и перспективу развития в стегоанализе как черных, так и цветных изображений. Развитие данного направления напрямую зависит от исследований в сфере сверточных сетей для последовательного анализа блоков изображения. Архитектура предложенной сети довольна проста. Единственным усложнением является дополнительный слой высокочастотной фильтрации для уменьшения шума на изображении и выделении контуров на изображении и гауссовская функция активации нейронов. В ходе обучения было доказано, что алгоритмы, основанные на таком подходе, показывают неплохой результат выявления факта наличия встраивания при использовании алгоритма WOW.

К минусам данной нейронной сети можно отнести зависимость от используемого датасета и коэффициента встраивания. К плюсам – высокий уровень точности при несложной архитектуре нейросети и независимость реализуемой схемы обработки от размера анализируемого изображения в той ее части, где проводится обучение нейронных сетей. Предложенная функция активации logSoftmax позволила добиться результатов, сравнимых в статье, при меньших размерах датасета.

# 6 Список использованных источников

1.Высшая школа экономики - это... // ВШЭ URL: https://www.hse.ru/info/fulltext (дата обращения: 20.03.2023).

2.Московский институт электроники и математики им. А. Н. Тихонова // МИЭМ URL: https://miem.hse.ru/ (дата обращения: 20.03.2023).

3.Сверточные нейронные сети, архитектура сверточной нейронной сети // Studfile URL: https://studfile.net/preview/6871550/page:9/ (дата обращения: 20.03.2023).

4.Что такое нейронная сеть // Хабр URL: https://habr.com/ru/post/309508/ (дата обращения: 20.03.2023).

5.А. А. Сирота, М. А. Дрюченко, А. Ю. Иванков СТЕГОАНАЛИЗ ЦИФРОВЫХ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ ПОВЕРХНОСТНОГО И ГЛУБОКОГО МАШИННОГО ОБУЧЕНИЯ: ИЗВЕСТНЫЕ ПОДХОДЫ И НОВЫЕ РЕШЕНИЯ // Воронеж: 2021 – c. 10 – 27

6.Your Machine Learning and Data Science // Kaggle URL: https://www.kaggle.com/ (дата обращения: 21.03.2023).

7.Ставров Е. А. АВТОМАТИЗИРОВАННАЯ СИСТЕМА АНАЛИЗА ИЗОБРАЖЕНИЙ НА ПРЕДМЕТ СКРЫТОЙ ИНФОРМАЦИИ: дис. канд. Математика и компьютерные науки наук: 02.04.01. - Белгород, 2017. - 83 с.

# ПРИЛОЖЕНИЕ А

Листинг программы, написанной на языке Python:

#!/usr/bin/env python

# coding: utf-8

# # СТЕГОАНАЛИЗ ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ ГЛУБОКОГО МАШИННОГО ОБУЧЕНИЯ

# ## Аннотация

# Рассматривается современное состояние проблемы стегоанализа цифровых изображений, направленной на исследование и разработку эффективных методов выявления стеганографически скрытых (визуально незаметных) сообщений в контейнерах- изображениях.

# ## Содержание

#

# 1. [Импорт необходимых библиотек](#first)

# 2. [Получение и предобработка данных](#second)

# 1. [Алгоритм WOW](#wow\_algos)

# 2. [Встраивание и предобработка данных](#data\_prepare)

# 3. [DataLoader - структурирование данных](#data\_loader)

# 4. [Сверточная нейросеть](#neural\_net)

# 5. [Обучение и валидация модели](#tests)

# 6. [Тесты нейросети с различными параметрами встраивания](#wow\_test)

# 1. [Тесты на датасете с картинками без граней](#wow\_test\_1)

# 2. [Тесты на датасете с картинками с гранями](#wow\_test\_2)

# ## 1. Импорт необходимых библиотек

# <span id="first"></span>

# In[1]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

from scipy.signal import convolve2d

from scipy.interpolate import make\_interp\_spline

from skimage import io, transform

from IPython.display import clear\_output

from sklearn.utils import shuffle

from sklearn.metrics import accuracy\_score, f1\_score

import torch, torchvision

from torchvision import transforms

from torch.utils.data import Dataset, DataLoader

from torch import nn

import torch.nn.functional as F

import os

import string

import random

from tqdm import tqdm

import time

import warnings

# In[2]:

warnings.filterwarnings("ignore")

plt.rcParams["figure.figsize"] = (3,3)

# ## 2. Получение и предобработка данных

# ### 2.1 Алгоритм WOW

# <span id="wow\_algos"></span>

# In[3]:

def WOW(cover, payload:float, params:int):

# Input: cover ... картинка

# payload ..... информация в битах на пиксель

# Output: stego ....... конечная картинка с встроенной информацией

# Получение 2D вейвлет-фильтров

# 1D фильтр высокочастотного разложения

hpdf = np.matrix([[-0.0544158422, 0.3128715909, -0.6756307363, 0.5853546837, 0.0158291053, -0.2840155430,

-0.0004724846, 0.1287474266, 0.0173693010, -0.0440882539, -0.0139810279, 0.0087460940,

0.0048703530, -0.0003917404, -0.0006754494, -0.0001174768]])

# 1D фильтр низкочастотного разложения

lpdf = np.matrix((-1) \*\* np.array(range(hpdf.shape[1])) \* np.fliplr(hpdf).A)

# построение двумерных вейвлет-фильтров

F = []

F.append(lpdf.T \* hpdf)

F.append(hpdf.T \* lpdf)

F.append(hpdf.T \* hpdf)

## Получение затрат на встраивание

cover = cover.astype('float64')

p = params

wetCost = 10 \*\* 10

sizeCover = cover.shape

# добавление padding

padSize = np.max(np.concatenate([F[0].shape, F[1].shape, F[2].shape]))

coverPadded = np.pad(cover, ((padSize, padSize), (padSize, padSize)), 'symmetric')

# вычисление остатка направленности и пригодность xi для каждого фильтра

xi = []

for fIndex in range(0, 3):

# вычисление остатка

R = convolve2d(coverPadded, F[fIndex], 'same')

# вычисление пригодности

xi.append(convolve2d(abs(R), np.rot90(abs(F[fIndex]), 2), 'same'))

# корректировка сдвига пригодности, если размер фильтра четный

if np.mod(np.size(F[fIndex], 0), 2) == 0:

xi[fIndex] = np.roll(xi[fIndex], 1, axis=0)

if np.mod(np.size(F[fIndex], 1), 2) == 0:

xi[fIndex] = np.roll(xi[fIndex], 1, axis=1)

# удаляем padding

a\_idx\_s = int((np.size(xi[fIndex], 0) - sizeCover[0]) / 2)

a\_idx\_e = int(np.size(xi[fIndex], 0) - (np.size(xi[fIndex], 0) - sizeCover[0]) / 2)

b\_idx\_s = int((np.size(xi[fIndex], 1) - sizeCover[1]) / 2)

b\_idx\_e = int(np.size(xi[fIndex], 1) - (np.size(xi[fIndex], 1) - sizeCover[1]) / 2)

xi[fIndex] = xi[fIndex][a\_idx\_s:a\_idx\_e, b\_idx\_s:b\_idx\_e]

# вычисление стоимости встраивания \rho

rho = (xi[0] \*\* p + xi[1] \*\* p + xi[2] \*\* p) \*\* (-1 / p)

# корректировка затрат на встраивание

rho[rho > wetCost] = wetCost # порог для затрат

rho[np.isnan(rho)] = wetCost # если все xi равны нулю, то порог затрат снижается

rhoP1 = rho.copy()

rhoM1 = rho.copy()

rhoP1[cover == 255] = wetCost # не встраивать +1, если пиксель имеет максимальное значение

rhoM1[cover == 0] = wetCost # не встраивать -1, если пиксель имеет минимальное значение

# встраиваем

stego = EmbeddingSimulator(cover, rhoP1, rhoM1, payload \* cover.size, False)

distortion\_local = rho[cover != stego]

distortion = np.sum(distortion\_local)

return stego, distortion

def EmbeddingSimulator(x, rhoP1, rhoM1, m, fixEmbeddingChanges):

n = x.size

m\_lambda = calc\_lambda(rhoP1, rhoM1, m, n)

pChangeP1 = (np.exp(-m\_lambda \* rhoP1)) / (1 + np.exp(-m\_lambda \* rhoP1) + np.exp(-m\_lambda \* rhoM1))

pChangeM1 = (np.exp(-m\_lambda \* rhoM1)) / (1 + np.exp(-m\_lambda \* rhoP1) + np.exp(-m\_lambda \* rhoM1))

if fixEmbeddingChanges:

np.random.seed(100)

else:

np.random.seed(int(time.time()))

randChange = np.random.rand(\*x.shape)

y = x.copy()

y[randChange < pChangeP1] = y[randChange < pChangeP1] + 1

y[np.logical\_and(randChange >= pChangeP1, randChange < (pChangeP1+pChangeM1))] = y[np.logical\_and(randChange >= pChangeP1, randChange < (pChangeP1+pChangeM1))] - 1

return y

def calc\_lambda(rhoP1, rhoM1, message\_length, n):

l3 = 1e+3

m3 = float(message\_length + 1)

iterations = 0

while m3 > message\_length:

l3 = l3 \* 2

pP1 = np.exp(-l3 \* rhoP1) / (1 + np.exp(-l3 \* rhoP1) + np.exp(-l3 \* rhoM1))

pM1 = np.exp(-l3 \* rhoM1) / (1 + np.exp(-l3 \* rhoP1) + np.exp(-l3 \* rhoM1))

m3 = ternary\_entropyf(pP1, pM1)

iterations = iterations + 1

if iterations > 10:

m\_lambda = l3

return

l1 = 0

m1 = float(n)

m\_lambda = 0

alpha = float(message\_length) / n

# ограниваем поиск 30 итерациями

# и смотрим, чтобы относительная полезная нагрузка, встроенная в сообщение, находилась примерно в пределах 1/1000 от требуемой

while float(m1-m3)/n > (alpha/1000.0) and (iterations < 30):

m\_lambda = l1 + (l3 - l1) / 2

pP1 = (np.exp(-l3 \* rhoP1)) / (1 + np.exp(-l3 \* rhoP1) + np.exp(-l3 \* rhoM1))

pM1 = (np.exp(-l3 \* rhoM1)) / (1 + np.exp(-l3 \* rhoP1) + np.exp(-l3 \* rhoM1))

m2 = ternary\_entropyf(pP1, pM1)

if m2 < message\_length:

l3 = m\_lambda

m3 = m2

else:

l1 = m\_lambda

m1 = m2

iterations = iterations + 1

return m\_lambda

def ternary\_entropyf(pP1, pM1):

eps = 3e-16

p0 = 1 - pP1 - pM1

P = np.concatenate([p0.flatten(order='F'), pP1.flatten(order='F'), pM1.flatten(order='F')])

P[P == 0] = 1e-16

H = - (P \* np.log2(P))

H[np.logical\_or(P < eps, P > (1 - eps))] = 0

Ht = sum(H)

return Ht

# ### 2.2 Предобработка данных

# <span id="data\_prepare"></span>

# In[4]:

clear\_data\_path = "data/clear\_data"

stego\_data\_path = "data/stego\_data"

# In[ ]:

pictures\_names = os.listdir('data/clear\_data')

for name in tqdm(pictures\_names):

if not name.startswith('.'):

payload = 0.4

params = -1

filename = os.path.join(clear\_data\_path, name)

cover = Image.open(filename)

cover = cover.resize((256,256))

cover = cover.convert('L')

cover.save(filename)

# cover = np.array(cover)

# stego, distortion = WOW(cover, payload, params)

# Image.fromarray(np.uint8(stego)).save(os.path.join(stego\_data\_path, name))

# In[5]:

clear\_paths = list(map(lambda x: os.path.join(clear\_data\_path, x), os.listdir(clear\_data\_path)))

pictures\_clear\_df = pd.DataFrame({

"picture\_link": clear\_paths,

"is\_changed": np.zeros(len(clear\_paths), dtype=int)

})

stego\_paths = list(map(lambda x: os.path.join(stego\_data\_path, x), os.listdir(stego\_data\_path)))

pictures\_graphed\_df = pd.DataFrame({

"picture\_link": stego\_paths,

"is\_changed": np.ones(len(stego\_paths), dtype=int)

})

data = shuffle(pd.concat([pictures\_clear\_df, pictures\_graphed\_df]))

data.reset\_index(drop=True, inplace=True)

# In[6]:

data.drop(index=data[data["is\_changed"] == 1].sample(n=2300)['is\_changed'].index, inplace=True)

data.info()

data.head()

# In[7]:

plt.hist(data.is\_changed)

plt.xlabel("Класс объекта")

plt.ylabel("Кол-во объектов данного класса")

plt.title("Гистограмма отношения классов");

# In[8]:

data = data[data['picture\_link'] != "data/clear\_data/.DS\_Store"]

data = data[data['picture\_link'] != "data/stego\_data/.DS\_Store"]

data.to\_csv("data.csv", index=False)

# ## 3. DataLoader для удобного структурирования данных

# <span id="data\_loader"></span>

# In[76]:

class Dataset(Dataset):

"""Описантельный класс датасета для удобной работы с ним"""

def \_\_init\_\_(self, csv\_file=None, transform=None):

"""

Args:

csv\_file (string): Путь к csv файлу с разметкой

transform (callable, optional): Опционально, трансформации применяемые к картинкам

"""

self.annotations = pd.read\_csv(csv\_file)

self.transform = transform

def \_\_len\_\_(self):

return self.annotations.shape[0]

def \_\_getitem\_\_(self, idx):

if torch.is\_tensor(idx):

idx = idx.tolist()

img\_info = self.annotations.iloc[idx]

image = io.imread(img\_info[0])

label = img\_info[1]

if self.transform:

image = self.transform(image)

sample = torch.tensor([image], dtype=torch.float32), label

return sample

# In[10]:

transform = transforms.Compose([transforms.ToPILImage(), transforms.Resize((256, 256)), transforms.ToTensor()])

train\_dataset = Dataset(csv\_file='data.csv')

train\_part = np.arange(0, int(len(train\_dataset) \* 0.7))

val\_part = np.arange(int(len(train\_dataset) \* 0.7), int(len(train\_dataset) \* 0.9))

test\_part = np.arange(int(len(train\_dataset) \* 0.9), len(train\_dataset))

sampler\_to\_train = torch.utils.data.SubsetRandomSampler(train\_part)

sampler\_to\_val = torch.utils.data.SubsetRandomSampler(val\_part)

sampler\_to\_test = torch.utils.data.SubsetRandomSampler(test\_part)

# In[11]:

train\_loader = DataLoader(train\_dataset, batch\_size=64, sampler=sampler\_to\_train)

val\_loader = DataLoader(train\_dataset, batch\_size=64, sampler=sampler\_to\_val)

test\_loader = DataLoader(train\_dataset, batch\_size=64, sampler=sampler\_to\_test)

# In[12]:

for image, label in train\_loader:

print(torch.tensor(image).shape, label.sum())

plt.imshow(image[0].transpose(0, -1).transpose(1, 0))

plt.title(label[0])

plt.show();

break

# ## 4. Нейросеть на torch

# <span id="neural\_net"></span>

# In[13]:

def gaussian(inp: torch.Tensor) -> torch.Tensor:

"""Считает гауссову функцию для каждого входного тензора

Args:

inp (torch.Tensor): входной тензор

Returns:

torch.Tensor: тензор после применения Гаусса

"""

return torch.exp(-((inp - torch.mean(inp)) \*\* 2) / (torch.std(inp)) \*\* 2)

# In[14]:

class ImageProcessing(nn.Module):

"""Считает свёртку с помощью использования пространственного фильтра высоких частот с фиксированным ядром"""

def \_\_init\_\_(self) -> None:

super().\_\_init\_\_()

self.kv\_filter = (

torch.tensor(

[

[-1.0, 2.0, -2.0, 2.0, -1.0],

[2.0, -6.0, 8.0, -6.0, 2.0],

[-2.0, 8.0, -12.0, 8.0, -2.0],

[2.0, -6.0, 8.0, -6.0, 2.0],

[-1.0, 2.0, -2.0, 2.0, -1.0],

],

).view(1, 1, 5, 5) / 12.0

)

def forward(self, inp: torch.Tensor) -> torch.Tensor:

"""Возвращает тензор, над которым была произведена свертка"""

return F.conv2d(inp, self.kv\_filter)

# In[15]:

class ConvPool(nn.Module):

"""

Данный класс возвращает экземпляр свертки, выстроенной по необходимым параметрам,

то есть схема данного блока это conv -> batch norm -> gaussian -> average pooling

"""

def \_\_init\_\_(

self,

in\_channels: int = 16,

out\_channels: int = 32,

conv\_kernel\_size: tuple[int, int] = (3, 3),

conv\_stride: int = 1,

pool\_stride: int = 2,

pool\_kernel\_size: tuple[int, int] = (3, 3),

pool\_padding: int = 0,

) -> None:

super().\_\_init\_\_()

self.conv = nn.Conv2d(

in\_channels=in\_channels,

out\_channels=out\_channels,

kernel\_size=conv\_kernel\_size,

stride=conv\_stride,

padding=0,

bias=True,

)

self.pool = nn.AvgPool2d(kernel\_size=pool\_kernel\_size, stride=pool\_stride, padding=pool\_padding)

self.bn = nn.BatchNorm2d(out\_channels)

def forward(self, inp: torch.Tensor) -> torch.Tensor:

"""Соответсвенно conv->batch norm->activation->average pooling."""

return self.pool(gaussian(self.bn(self.conv(inp))))

# In[16]:

class CNN(nn.Module):

"""

Реализация сверточной нейронной сети

"""

def \_\_init\_\_(self) -> None:

super().\_\_init\_\_()

self.layer1 = ConvPool(

in\_channels=1,

out\_channels=16,

conv\_kernel\_size=(5, 5),

pool\_kernel\_size=(3,3)

)

self.layer2 = ConvPool(

in\_channels=16,

out\_channels=64,

conv\_kernel\_size=(3, 3),

)

self.layer3 = ConvPool(

in\_channels=64,

out\_channels=128,

conv\_kernel\_size=(3, 3),

)

self.fully\_connected = nn.Sequential(

nn.Linear(in\_features=100352, out\_features=128),

nn.ReLU(inplace=True),

nn.Linear(in\_features=128, out\_features=128),

nn.ReLU(inplace=True),

nn.Linear(in\_features=128, out\_features=2),

nn.LogSoftmax(dim=1),

)

def forward(self, image: torch.Tensor) -> torch.Tensor:

"""Вернет логиты для данного изображения"""

with torch.no\_grad():

out = ImageProcessing()(image)

out = self.layer1(out)

out = self.layer2(out)

out = self.layer3(out)

out = out.view(out.size(0), -1)

out = self.fully\_connected(out)

return out

# In[17]:

class Evaluation:

def \_\_init\_\_(self, train\_loader, val\_loader, test\_loader, model, lr=0.02, epoch=15):

self.model = model

self.train\_loader = train\_loader

self.val\_loader = val\_loader

self.test\_loader = test\_loader

self.epoch = epoch

self.optimizer = torch.optim.Adamax(

self.model.parameters(),

lr=lr,

betas=(0.9, 0.999),

eps=1e-8,

weight\_decay=0,

)

self.loss\_fn = nn.NLLLoss()

def validation(self):

accuracy\_list = []

for xb, yb in self.val\_loader:

y\_pred = self.model(xb).data.max(1)[1]

accuracy = accuracy\_score(y\_pred, yb)

accuracy\_list.append(accuracy)

return np.mean(np.array(accuracy\_list))

def test(self):

accuracy\_list = []

for xb, yb in self.test\_loader:

y\_pred = self.model(xb).data.max(1)[1]

accuracy = accuracy\_score(y\_pred, yb)

accuracy\_list.append(accuracy)

return np.mean(np.array(accuracy\_list))

def fit(self):

losses = []

valid\_accuracies = []

train\_accuracies = []

test\_accuracies = []

for i in range(1, self.epoch + 1):

epoch\_train\_accuracies = []

counter = 1

self.model.train()

for X, y in tqdm(self.train\_loader):

print(f'Iteration {counter} started')

counter += 1

clear\_output(wait=True)

self.optimizer.zero\_grad()

y\_pred = self.model(X)

loss = self.loss\_fn(y\_pred, y)

losses.append(loss.item())

print(loss.item())

y\_pred = y\_pred.data.max(1)[1]

accuracy = accuracy\_score(y\_pred, y)

epoch\_train\_accuracies.append(accuracy)

loss.backward()

self.optimizer.step()

print("Starting validation")

self.model.eval()

valid\_accuracy = self.validation()

train\_accuracy = np.mean(np.array(epoch\_train\_accuracies))

valid\_accuracies.append(valid\_accuracy)

train\_accuracies.append(train\_accuracy)

print("Valid\_accuracy in:", valid\_accuracy)

print("Train\_accuracy in:", train\_accuracy)

print(f'Epoch {i} is finished!')

time.sleep(10)

test\_accuracy = self.test()

print(f"Mean loss is: {np.mean(np.array(losses))}")

print(f"Mean valid\_accuracy is: {np.mean(np.array(valid\_accuracies))}")

print(f"Mean train\_accuracy is: {np.mean(np.array(train\_accuracies))}")

return losses, valid\_accuracies, train\_accuracies, test\_accuracy

# In[18]:

net = CNN()

# ## 5. Обучение и валидация модели

# <span id="tests"></span>

# In[19]:

evaluation = Evaluation(train\_loader=train\_loader, val\_loader=val\_loader, test\_loader=test\_loader, model=net)

# In[20]:

losses, valid\_accuracies, train\_accuracies, test\_accuracy = evaluation.fit()

# In[21]:

torch.save(evaluation.model.state\_dict(), "NN\_weights")

# ## 6. Вывод

# В процессе выполнения, были выполнены такие шаги, как получение, обработка и классификация данных. Во-первых, мы использовали алгоритм стеганографии, который называется `WOW`, встраивание производилось с параметрами `payload = 0.4`. После разбиения данных на выборки, мы удобно их структурировали и приступили к обучению модели. По итогу имеем такие результаты, изображенные на графике:

# In[22]:

plt.rcParams["figure.figsize"] = (5,5)

X\_Y\_res\_val = make\_interp\_spline(range(1, 16), valid\_accuracies)

X\_res\_val = np.linspace(1, 15, 500)

Y\_res\_val = X\_Y\_res\_val(X\_res\_val)

plt.plot(X\_res\_val, Y\_res\_val, label="Train\_accuracy")

X\_Y\_res\_train = make\_interp\_spline(range(1, 16), train\_accuracies)

X\_res\_train = np.linspace(1, 15, 500)

Y\_res\_train = X\_Y\_res\_train(X\_res\_train)

plt.plot(X\_res\_train, Y\_res\_train, label="Valid\_accuracy")

plt.legend()

plt.grid()

plt.ylabel("Значение accuracy\_score")

plt.xlabel("Номер эпохи")

plt.show()

# In[23]:

print(f"Значение loss по окончании обучения: {np.min(losses)}")

print(f"Значение loss на начале обучения обучения: {np.max(losses)}")

print(f"Значение accuracy\_score на test: {test\_accuracy}")

print(f"Значение accuracy\_score на train: {np.max(train\_accuracies)}")

print(f"Значение accuracy\_score на valid: {np.max(valid\_accuracies)}")

# Тут мы видим, что `accuracy\_score` для тренировочной выборки скачет, это из-за того, что нейросети постоянно попадались батчи с новыми и новыми признаками. Но тем не менее, валидационный `accuracy\_score` растет постоянно, поэтому можно сказать, что никаких проблем не наблюдается.

#

# ### Итог: payload = 0.4

# In[24]:

pd.DataFrame({

'data\_type':['train', 'valid', 'test'],

'accuracy\_score':[np.max(train\_accuracies), np.max(valid\_accuracies), test\_accuracy],

})

# ## 6. Тесты на устойчивость

# <span id="wow\_test"></span>

# In[25]:

for i in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:

os.system(f"mkdir tests/barrier\_tests/stego\_payload\_{i}")

os.system(f"mkdir tests/barrier\_tests/stego\_payload\_{i}/images")

os.system(f"mkdir tests/no\_barrier\_tests/stego\_payload\_{i}")

os.system(f"mkdir tests/no\_barrier\_tests/stego\_payload\_{i}/images")

# ### 6.1 Без граней (город, машины)

# <span id="wow\_test\_1"></span>

# In[26]:

barriers\_pictures\_dir = list(filter(lambda x: x!= "clear" and not x.startswith("."), os.listdir("tests/no\_barrier\_tests")))

clear\_data = os.listdir("tests/no\_barrier\_tests/clear")

for directory in tqdm(barriers\_pictures\_dir):

payload = float(directory.split("\_")[-1])

params = -1

for img in clear\_data:

filename = os.path.join("tests/no\_barrier\_tests/clear", img)

cover = Image.open(filename)

cover = cover.resize((256,256))

cover = cover.convert('L')

cover.save(filename)

cover = np.array(cover)

stego, distortion = WOW(cover, payload, params)

Image.fromarray(np.uint8(stego)).save(os.path.join(f"tests/no\_barrier\_tests/{directory}/images", img))

# In[27]:

end\_data = {

"payload":[],

"accuracy":[],

}

for directory in tqdm(barriers\_pictures\_dir):

path = f"tests/no\_barrier\_tests/{directory}/"

filename = f"{path}{'+'.join(directory.split('\_')[-1].split('.'))}.csv"

stego\_links = list(map(lambda x: path + "images/" + x, os.listdir(path + "images")))

clear\_links = list(filter(lambda y: not y.split("/")[-1].startswith("."), map(lambda x: "tests/no\_barrier\_tests/clear\_for\_test/" + x, os.listdir("tests/no\_barrier\_tests/clear\_for\_test"))))

df = shuffle(pd.DataFrame({

"picture\_link":stego\_links + clear\_links,

"is\_changed":np.concatenate((np.ones(len(stego\_links), dtype=int), np.zeros(len(clear\_links), dtype=int)), axis=None)

}))

df.to\_csv(filename, index=False)

test\_set = Dataset(csv\_file=filename)

sampler = torch.utils.data.SubsetRandomSampler(np.arange(0, df.picture\_link.shape[0]))

loader = DataLoader(test\_set, batch\_size=32, sampler=sampler)

t\_accuracies = []

for xb, yb in loader:

with torch.no\_grad():

y\_pred = evaluation.model(xb).data.max(1)[1]

accuracy = accuracy\_score(y\_pred, yb)

t\_accuracies.append(accuracy)

end\_data["payload"].append(directory.split('\_')[-1])

end\_data["accuracy"].append(np.mean(np.array(accuracy)))

# In[28]:

d = pd.DataFrame(end\_data).sort\_values(by='payload')

d

# ### 6.2 С гранями (рис)

# <span id="wow\_test\_2"></span>

# In[29]:

barriers\_pictures\_dir = list(filter(lambda x: x!= "clear" and not x.startswith("."), os.listdir("tests/barrier\_tests")))

clear\_data = os.listdir("tests/barrier\_tests/clear")

for directory in tqdm(barriers\_pictures\_dir):

payload = float(directory.split("\_")[-1])

params = -1

for img in clear\_data:

filename = os.path.join("tests/barrier\_tests/clear", img)

cover = Image.open(filename)

cover = cover.resize((256,256))

cover = cover.convert('L')

cover.save(filename)

cover = np.array(cover)

stego, distortion = WOW(cover, payload, params)

Image.fromarray(np.uint8(stego)).save(os.path.join(f"tests/barrier\_tests/{directory}/images", img))

# In[30]:

end\_data\_g = {

"payload":[],

"accuracy":[],

}

for directory in tqdm(barriers\_pictures\_dir):

path = f"tests/barrier\_tests/{directory}/"

filename = f"{path}{'+'.join(directory.split('\_')[-1].split('.'))}.csv"

stego\_links = list(map(lambda x: path + "images/" + x, os.listdir(path + "images")))

clear\_links = list(filter(lambda y: not y.split("/")[-1].startswith("."), map(lambda x: "tests/barrier\_tests/clear\_for\_test/" + x, os.listdir("tests/barrier\_tests/clear\_for\_test"))))

df = shuffle(pd.DataFrame({

"picture\_link":stego\_links + clear\_links,

"is\_changed":np.concatenate((np.ones(len(stego\_links), dtype=int), np.zeros(len(clear\_links), dtype=int)), axis=None)

}))

df.to\_csv(filename, index=False)

test\_set = Dataset(csv\_file=filename)

sampler = torch.utils.data.SubsetRandomSampler(np.arange(0, df.picture\_link.shape[0]))

loader = DataLoader(test\_set, batch\_size=32, sampler=sampler)

t\_accuracies = []

for xb, yb in loader:

with torch.no\_grad():

y\_pred = evaluation.model(xb).data.max(1)[1]

accuracy = accuracy\_score(y\_pred, yb)

t\_accuracies.append(accuracy)

end\_data\_g["payload"].append(directory.split('\_')[-1])

end\_data\_g["accuracy"].append(np.mean(np.array(accuracy)))

# In[31]:

no\_d = pd.DataFrame(end\_data\_g).sort\_values(by='payload')

no\_d

# In[32]:

payloads = np.array(list(map(float, d.payload.values)))

X\_Y\_Spline\_d = make\_interp\_spline(payloads, d.accuracy.values)

X\_d = np.linspace(payloads.min(), payloads.max(), 500)

Y\_d = X\_Y\_Spline\_d(X\_d)

X\_Y\_Spline\_nod = make\_interp\_spline(payloads, no\_d.accuracy.values)

X\_nod = np.linspace(payloads.min(), payloads.max(), 500)

Y\_nod = X\_Y\_Spline\_nod(X\_nod)

plt.plot(X\_d, Y\_d, label="No Border dataset")

plt.plot(X\_nod, Y\_nod, label="Border dataset")

plt.xlabel("Значение payload")

plt.ylabel("Значение accuracy")

plt.legend()

plt.grid()

plt.title("Зависимость accuracy от payload\n на разных типах картинок")

plt.show()