

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
ИМЕНИ АКАДЕМИКА С.П.КОРОЛЕВА»**

**(САМАРСКИЙ УНИВЕРСИТЕТ)**

|  |  |
| --- | --- |
| Институт | информатики, математики и электроники |
| Факультет | информатики |
| Кафедра | геоинформатики и информационной безопасности |

|  |
| --- |
| **ОТЧЕТ ПО ПРАКТИКЕ** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вид практики: | | преддипломная | | | |
| Срок прохождения практики: | | | | 03.02.2022 - 17.05.2022 | |
| по направлению подготовки 10.05.03 информационная безопасность автоматизированных систем (специалитет) | | | | | |
| Студент группы № 6512-100503D | |  | | | А.Н. Коновалов |
|  | | (подпись) | | |  |
| Руководитель практики от | | |  | |  |
| университета, д.ф.-м.н., профессор кафедры ГИиИБ | | |  | | В.В. Мясников |
|  | | (подпись) | | |  |
| Руководитель практики | | |  | |  |
| от организации |  | | | | В.В. Мясников |
|  | | (подпись) | | |  |

|  |  |  |
| --- | --- | --- |
|  | Дата сдачи: | 17.05.2022 г. |
|  | Дата защиты: | 17.05.2022 г. |
|  | Оценка |  |

Самара 2022

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
ИМЕНИ АКАДЕМИКА С.П.КОРОЛЕВА»**

(Самарский университет)

|  |  |
| --- | --- |
| Институт | информатики, математики и электроники |
| Факультет | информатики |
| Кафедра | геоинформатики и информационной безопасности |

**Индивидуальное задание на практику**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Студенту | | | А. Н. Коновалову | группы № 6512-100503D | | |
| Направление на практику оформлено приказом по университету | | | | | | **№ 3**6 **от 20.01.2022** |
| в | Самарский университет, кафедра ГИиИБ, | | | | | |
|  | | (наименование профильной организации или структурного подразделения университета) | | | | |
| в соответствии с договором о направлении на практику от | | | | | -. | |

|  |  |  |
| --- | --- | --- |
| Планируемые результаты освоения образовательной программы | Планируемые результаты практики | Содержание задания |
| ПСК-7.1 | способность разрабатывать и исследовать модели информационно-технологических ресурсов, разрабатывать модели угроз и модели нарушителя информационной безопасности в распределенных информационных системах | Исследовать нейросетевой метод обнаружения следов ретуширования изображений. |
| ПСК-7.2 | способностью проводить анализ рисков информационной безопасности и разрабатывать, руководить разработкой политики безопасности в распределенных информационных системах | Проанализировать современную актуальность проблемы обнаружения следов ретуширований изображений. |
| ПСК-7.5 | способностью координировать деятельность подразделений и специалистов по защите информации в организациях, в том числе на предприятии и в учреждении | Распределить время на выполнение плана по изучению ретуширования изображений, разработку программного обеспечения для подготовки данных для изучения и исследование нейросетевого метода обнаружения следов ретуширования на них. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Дата выдачи задания: | 03 февраля 2022 г. | | | | |
| Срок предоставления на кафедру отчета о практике: | | | | 17 мая 2022 г. | |
| Руководитель практики от | | |  | |  |
| университета, д.ф.-м.н., профессор кафедры ГИиИБ | | |  | | В.В. Мясников |
|  | | | (подпись) | |  |
| Руководитель практики | | |  | |  |
| от профильной организации | |  | | | В.В. Мясников |
|  | | | (подпись) | |  |
| Задание принял к исполнению | | |  | |  |
| студент группы № 6512-100503D | |  | | | А.Н. Коновалов |
|  | | (подпись) | | |  |

**Рабочий план проведения практики**

|  |  |  |
| --- | --- | --- |
| Дата (период) | Содержание задания | Результаты практики |
| 03.02.2022 – 01.03.2022 | Изучить теоретические материалы, касающиеся тематики ретуширования изображений и нейронных сетей. Проанализировать современную актуальность данной темы. Распределить время на выполнение плана по изучению методов восстановления исходных данных в изображениях. | Были изучены соответствующие теоретические материалы, сделан вывод об актуальности проблемы реконструкции изображений. Распределено время на выполнение поставленных задач. |
| 02.03.2022– 01.04.2022 | Поиск готового программного обеспечения для ретуширования изображений, выбор нескольких решений из найденных. Написание собственного программного обеспечения для подготовки базы данных изображений с вырезанными различными способами фрагментами и анализа качества реконструкции. | Было найдено готовое программное обеспечения для ретуширования изображений, выбрано нескольких решений из найденных. Написано собственное программное обеспечение для подготовки базы данных изображений с вырезанными различными способами фрагментами и анализа качества реконструкции. |
| 02.04.2022–  01.05.2022 | Реконструирование изображений из базы данных различными способами. Исследование применимости разных архитектур сверточной нейронной сети для обнаружения следов ретуширования изображений | Разными способами реконструированы изображения из базы данных. Была исследована применимость разных архитектур сверточной нейронной сети для обнаружения следов ретуширования изображений |
| 01.05.2022–  16.05.2022 | Анализ качества результатов. Создание отчета по проделанной работе | Проанализировано качество результатов, создан отчет по проделанной работе |

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель практики от | |  |  |
| университета, д.ф.-м.н., профессор кафедры ГИиИБ | |  | В.В. Мясников |
|  | | (подпись) |  |
| Руководитель практики | |  |  |
| от профильной организации |  | | В.В. Мясников |
|  | | (подпись) |  |

**О Т Ч Е Т**

о выполнении индивидуального задания

|  |  |  |
| --- | --- | --- |
| по | преддипломной | практике |

При прохождении преддипломной практики, руководителем были поставлены следующие задачи:

* Изучить теоретические материалы, касающиеся тематики ретуширования изображений и нейронных сетей;
* Проанализировать современную актуальность проблемы обнаружения следов ретуширования изображений;
* Распределить время на выполнение плана по изучению ретуширования изображений, поиск программного обеспечения по методам ретуширования изображений, разработку программного обеспечения для подготовки данных этими методами, разработку архитектур сверточных нейронных сетей и исследование их эффективности в задаче обнаружения следов ретуширования изображений;
* Найти программное обеспечение, реализующее несколько методов ретуширования изображений и при необходимости адаптировать его;
* Разработать при необходимости программное обеспечение для подготовки данных выбранными программными решениям.
* Разработать различные архитектуры сверточных нейронных сетей;
* Исследовать эффективность архитектур сверточных нейронных сетей обнаружения следов ретуширования на сгенерированных изображениях;
* Собрать результаты и реализовать отчёт.

Задания необходимо было выполнять последовательно в течение всего времени практики, предоставляя руководителю промежуточные отчеты.

ВЫПОЛНЕНИЕ ЗАДАНИЯ

Для выполнения поставленных руководителем задач было распределено время и были изучены теоретические материалы, касающиеся реконструкции изображений, а также нейронных сетей. Актуальность выбранной темы заключается в том, что обнаружение следов ретуширования изображения позволяет выявлять локальные подделки, которые могут косвенно указывать на наличие какой-либо конфиденциальной информации.

В качестве объекта исследования выбраны методы реконструирования исходных частей изображений на основе Contextual Residual Aggregation (CRA), Deep Fusion Network (DFN) и Hypergraph Image Inpainting (HII), а также архитектуры сверточных нейронных сетей для их распознавания.

Код CRA: <https://github.com/Atlas200dk/sample-imageinpainting-HiFill>

Заранее обученная модель для СRA взята отсюда (places2):

<https://github.com/Atlas200dk/sample-imageinpainting-HiFill/tree/master/GPU_CPU/pb>

Код DFN: <https://github.com/hughplay/DFNet>

Заранее обученная модель для DFN взята отсюда (places2):

<https://drive.google.com/drive/folders/1lKJg__prvJTOdgmg9ZDF9II8B1C3YSkN>

Код HII: <https://github.com/GouravWadhwa/Hypergraphs-Image-Inpainting>

Заранее обученная модель для HII взята отсюда (places2):

<https://drive.google.com/drive/folders/1dk1zSm1FxZVaafOtvoud8aAdZ6Ubs4oU>

В качестве исходных изображений для реконструкции взяты изображения из каждой выборки:

<https://www.kaggle.com/datasets/metavision/accurate-wasp-shapessegmentation>

<https://www.kaggle.com/datasets/metavision/accurate-mapple-leaf-shapessegmentation>

<https://www.kaggle.com/datasets/metavision/accurate-vinca-flower-shapessegmentation>

https://www.kaggle.com/datasets/metavision/accurate-mantis-shapessegmentation

Темой работы стало: «Разработка и исследование нейросетевого метода обнаружения следов ретуширования цифровых изображений».

Структура исследования представляет собой следующие этапы:

1. Описание объектов исследования;
2. Реализация программного обеспечения для подготовки данных;
3. Разработка и обучение архитектур сверточных нейронных сетей;
4. Исследование эффективности разработанных архитектур.

В [4] описано, что постановка задачи реконструкции цифрового изображения в наиболее простой постановке может быть представлена следующим образом. Пусть функция яркости изображения

(1)

задана на множестве отсчетов , где пара , определяет вертикальный и горизонтальный размеры изображения, , а множество - множество целых чисел до N. Значениями функции яркости выступают величины из множества V, в качестве которого на практике обычно выступают множества или , где K – число «каналов» изображения (обычное цветное изображения имеет три спектральных канала, соответствующих различным длинам волн), а B – число требуемых для представления яркости отсчета битов. Область задания изображения – множество Ω – в задаче реконструкции изображения (inpainting) предполагается состоящей из двух непересекающихся областей: , где определяет множество отсчетов изображения с известными значениями функции яркости (1), а , напротив, с неизвестными. Таким образом, в простейшем варианте постановки задачи реконструкции без обучения необходимо доопределить значения функции (1) на множестве (- известные величины функции яркости):

В случае реконструкции с обучением, наряду в самим реконструируемым изображением в постановке задачи могут фигурировать другие изображения того же типа (например, изображения лиц или изображения зданий), заданные обычно на всем множестве Ω. При этом в наиболее общей постановке для различных изображений множества Ω задания функции яркости могут быть различными.

Согласно [4] все методы решения задачи реконструкции могут быть разделены на классические, то есть использующие математические модели, и использующие машинное обучение. В [5] описано широкое многообразие методов из второй категории, основанных на различных методах машинного обучения, включая нейронные сети, например, CNN, DCNN, PEN, GAN и т. д.

Для реализации CRA согласно [1] используется нейронная сеть, причем изображение сначала делается более низкого разрешения, потом происходит восстановление фрагмента и затем изображение возвращается в исходное расширение, в результате изображение становится размытым. Далее происходит генерация компонент в области высоких частот специальным образом, формирующиеся исходя из соседних областей изображения, которым задаются определенные веса. Наконец, сгенерированные компоненты добавляются к увеличенному размытому изображению для получения более четкого результата. Поскольку сеть работает только с изображениями с низким разрешением, стоимость памяти и вычислительного времени значительно снижается. Более того, поскольку модель можно обучать с изображениями с низким разрешением, необходимость в наборах данных для обучения с высоким разрешением отпадает. Для улучшения качества восстановления и скорости вычислений в описании в дополнение могут применяться некоторые техники, например, конфигурации в виде тонких и глубоких слоев, распределение оценок внимания, многомерное распределение внимания и легковесные стробированные свертки (LWGC). Метод позволяет восстанавливать изображения размером до 8K с удовлетворительным качеством.

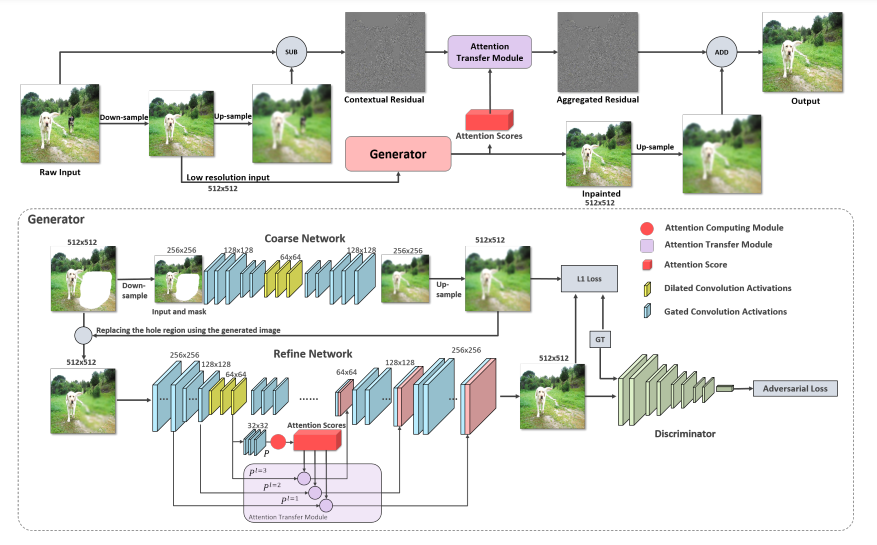


Рисунок 1 – Общий механизм CRA (в верхней части) и генератора (нижняя часть)

На рисунке 1 показан общий конвейер предлагаемого механизма CRA, в котором генератор - единственный обучаемый компонент в схеме. Учитывая высокое разрешение входного изображения, сначала понижается размер изображения до 512 × 512, а затем происходит его повышение до получения размытого изображения того же размера, что и исходное. Высота и ширина изображения не обязательно должны быть равны, но должны быть кратны 512.

В модуле вычисления внимания (Attention Computing Module) оценки внимания рассчитываются на основе близости к региону из высокоуровневой карты признаков (обозначенной буквой *P* на рисунке 1). *P* делится на фрагменты, и ACM вычисляет косинусное сходство между фрагментами внутри и снаружи отсутствующих областей:

где является i-м фрагментом, извлеченным из *P* снаружи маски, является j-м фрагментом, извлеченным из *P* внутри маски. Затем softmax-функция применяется к оценкам сходства, чтобы получить оценки внимания для каждого патча:

где *N* - количество фрагментов за пределами вырезанного региона. Размер каждого региона составляет 3 × 3, а *P* - 32 × 32, таким образом, может быть извлечено всего 1024 фрагмента. По всей матрице 1024 × 1024 сохраняются оценки близости между любой возможной парой фрагментов, хотя только часть из них полезна.

В модуле передачи внимания (Attention Tranfer Module), после получения оценок внимания из *P*, соответствующие вырезанные области в картах признаков нижнего уровня () могут быть заполнены контекстными фрагментами, взвешенными по оценкам внимания:

где l ∈ 1,2,3 - номер слоя, – i-й фрагмент, извлеченный из вне областей, охваченных маской, а – j-й фрагмент, который должен быть заполнен внутри замаскированных областей. N указывает количество контекстных фрагментов (фона). После расчета всех фрагментов внутри вырезанной области получается заполненная карта признаков . Поскольку размер карт признаков зависит от слоя, размер фрагментов должен соответственно меняться. Предполагая, что размер карты признаков равен 1282, а оценка внимания рассчитывается на основе 322 фрагментов, становится ясно что размеры фрагментов должны быть больше или равны (128/32)2 = 42, чтобы можно было охватить все пиксели. Если размеры фрагментов больше 4 × 4, то определенные пиксели перекрываются, что нормально, поскольку следующие уровни сети научатся адаптироваться.

Целью остаточного агрегирования (Residual Aggregation) является вычисление остатков для вырезанной области, чтобы можно было восстановить точные детали отсутствующего содержимого. Остатки для недостающего содержимого можно рассчитать путем агрегирования взвешенных контекстных остатков, полученных на предыдущих шагах:

где *R* - остаточное изображение, - это i-й фрагмент, извлеченный из контекстного остаточного изображения за пределами маски, а - j-й фрагмент, который должен быть заполнен внутри маски. Размеры фрагмента должны быть откалиброваны, чтобы точно покрывать все пиксели без перекрытия и гарантировать, что заполненные остатки согласованы с окружающими областями. После получения агрегированного остаточного изображения происходит его добавление к размытому изображению генератора с повышенной размером и получается более четкий результат.

На рисунке 1 также показана архитектура сети генератора. Используется двухступенчатая сетевая архитектура, где “грубая” (Coarse) сеть заполняет примерным образом отсутствующее содержимое, а “точная” (Refine) сеть генерирует уточнение результатов. Генератор берет изображение и двоичную маску, указывающую области вырезанных фрагментов, в качестве входных данных и строит законченное изображение. Ожидается, что размеры входа и выхода будут 512 × 512. Для увеличения поля восприятия и сокращения вычислений входные данные подвергаются понижению размера до 256 × 256 перед сверткой в ​​грубой сети, в отличие от уточненной сети, которая работает над размером 512 × 512. Прогноз грубой сети наивно смешивается с входным изображением путем замены вырезанной области в последнем на область первого в качестве входных данных для уточняющей сети. Сеть Refine вычисляет оценки контекстного внимания с помощью высокоуровневой карты признаков и выполняет передачу внимания на нескольких картах признаков более низкого уровня, таким образом, удаленная контекстная информация может быть заимствована на нескольких уровнях абстракции. Также применяются расширенные свертки (dilated convolutions) как в “грубой”, так и в “точной” сетях, чтобы еще больше увеличить размер полей восприятия.

В [2] декларируется проблема того, что восстановление изображения с помощью глубокого обучения обычно негармонично соединяет восстановленный фрагмент с существующим контентом, особенно в приграничной зоне. В данной статье эта проблема рассматривается с новой точки зрения создания плавного перехода и предлагается решение DFN. Вводится специальный “блок слияния” (fusion block), который создает гибкую карту альфа-композиции для объединения известных и неизвестных регионов (рисунок 3). Блок слияния не только обеспечивает плавное слияние восстановленного и существующего контента, но также предоставляет карту внимания, чтобы сосредоточить внимание сети на неизвестные пиксели. Таким образом, он создает мост для структурной и текстурной информации, так что информация может естественным образом распространяться из известного региона.

Блоки слияния встраиваются в несколько уровней декодера сети U-Net, что показано на рисунке 2. В сочетании с регулируемым механизмом ограничения потерь на каждом слое, достигается более точная структурная информация. Метод является высокопроизводительным, а также обеспечивает гармоничный фактурный переход, фактурную детализацию и смысловую структурную согласованность.

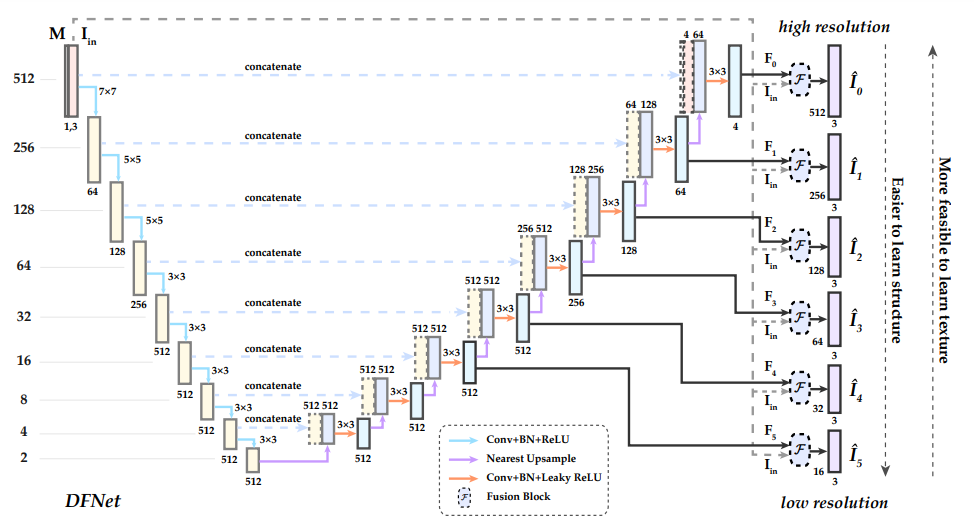


Рисунок 2 – общая структура DFNet.

Сеть U-Net, согласно [6] содержит сверточную и разверточную части. На каждом шаге количество каналов признаков удваивается. Сверточная часть похожа на обычную свёрточную сеть, она содержит два подряд свёрточных слоя 3×3, после которых идет слой ReLU и пулинг с функцией максимума 2×2 с шагом 2. Каждый шаг разверточной части содержит слой, обратный пулингу, который расширяет карту признаков, после которого следует свертка 2×2, которая уменьшает количество каналов признаков. После идет конкатенация с соответствующим образом обрезанной картой признаков из сжимающего пути и две свертки 3×3, после каждой из которой идет ReLU. Обрезка нужна из-за того, что мы теряем пограничные пиксели в каждой свёртке. На последнем слое свертка 1×1 используется для приведения каждого 64-компонентного вектора признаков до требуемого количества классов.

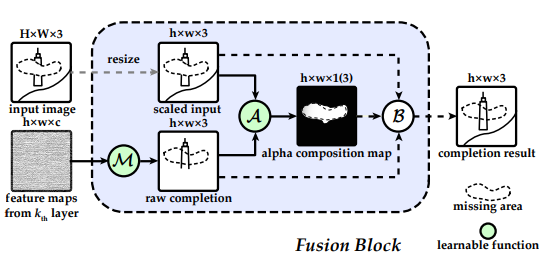


Рисунок 3 – Блок слияния получает сырой вариант заполнения отсутствующего региона из карт признаков на выходе обучаемой функции M, а также строит карту альфа-композиции с помощью функции A. Далее происходит объединение полученного варианта заполнения с масштабированным входным изображением с помощью функции смешивания B.

В [3], описывающем HII, предлагается при разработке метода реконструкции опираться на гиперграфы. Большинство существующих подходов реконструкции используют механизм внимания для изучения глобального контекста изображения. Этот механизм внимания семантически правдоподобный, но генерирует расплывчатые результаты из-за невозможности зафиксировать глобальный контекст. В данной работе описано использование свертки гиперграфов на пространственных объектах для использования в обучении сложных взаимоотношения между данными. Для этого вводится обучаемый механизм для подключения узлов, использующий гиперребра для свертки гиперграфа, показанный на рисунке 4.

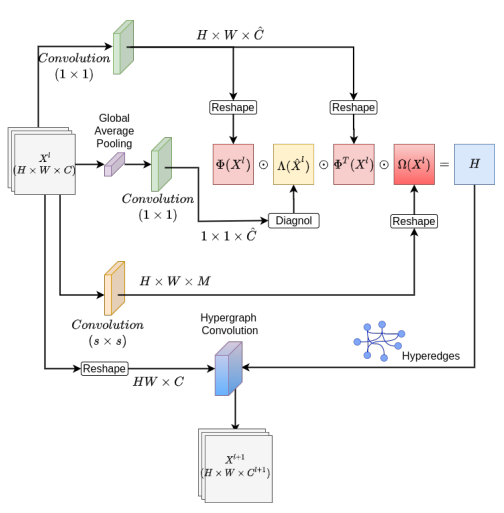


Рисунок 4 – Обучаемый слой гиперграфа. Сначала вычисляется матрица инцидентности H с использованием информации, собранной из входных характеристик, а затем вычисляется свертка гиперграфа с использованием вычисленной матрицы инцидентности.

Структура гиперграфов используется во многих задачах компьютерного зрения для моделирования ограничений высокого порядка на данные, которые не могут быть размещены в традиционной структуре графа. В отличие от парных связей в графах, гиперграф содержит гиперребро, которое может соединять две или более вершины. Гиперграф определяется как , где – набор вершин, – набор гиперребер, . Матрица инцидентов определяется как:

Для данного гиперграфа *G* степень вершины и степень гиперребра также определяются как и .

Матрица инцидентности *H*, степень вершины *D* и степень гиперребра *B* используются для вычисления нормализованной матрицы Лапласа гиперграфа , причем . Это симметричная положительно полуопределенная матрица и ее собственное разложение можно использовать для получения полного набора ортонормированных собственных векторов и диагональной матрицы , содержащая соответствующие неотрицательные собственные значения. Мы можем определить преобразование Фурье гиперграфа , которое преобразует сигнал в спектральную область, определенную на базисе *Φ*, также известный как базис Фурье. С обобщением теоремы о свертке на структурированное пространство гиперграфов, свертка по сигналу может определяться как:

Где является функцией коэффициентов Фурье. Однако для вычисления свертки сигнала *x* потребуется вычислить собственные векторы матрицы Лапласа.

Далее предлагается использовать параметризацию усеченными многочленами Чебышева до K-го порядка, таким образом определяя сверточную операцию над сигналом гиперграфа как:

Где вектор коэффициентов полинома Чебышева, а полином Чебышева. В этой формуле исключена операция вычисления собственных векторов матрицы Лапласа. Она может быть еще больше упрощена ограничением К = 1. Оно аппроксимируется ≈ 2 из-за масштабируемости нейронных сетей, следовательно, сверточная операция над сигналом гиперграфа принимает вид:

где *θ* - единственный коэффициент Чебышева, оставшийся после взятия K = 1 многочленов Чебышева.

Для полученного сигнала в виде гиперграфа , где это размерность вектора признаков на входе слоя под номером *l*, операция свертки в многослойной сверточной сети гиперграфов обобщается как:

Где - обучаемый параметр, а *σ* - нелинейная функция активации. Матрица инцидентности *H* кодирует структуру гиперграфа, которая в дальнейшем используется для распространения информации между узлами гиперграфа. Следовательно, можно легко увидеть, что более качественные соединения гиперребер могут привести к лучшему обмену информацией между узлами, дополнительно улучшая законченное изображение.

Чтобы преодолеть ограниченное восприимчивое поле архитектур CNN, недавние исследования преобразуют карты пространственных характеристик в структуру на основе графа и выполняют свертку графа, чтобы зафиксировать глобальную взаимосвязь между данными. Легко заметить, что простые графы являются частным случаем гиперграфов, где каждое гиперребро соединяет только два узла. Простой граф может легко представить парные отношения между данными, но уже труднее с помощью него представить сложные отношения между пространственными характеристиками изображения, поэтому предлагается использовать гиперграфы вместо графов. Чтобы преобразовать пространственные признаки в графоподобную структуру, каждый пространственный объект рассматривается как узел, имеющий вектор признаков размера *c*, .

В недавних исследованиях для задачи визуальной классификации матрица инцидентности *H* формируется с использованием евклидова расстояния между элементами изображений. Чтобы лучше фиксировать внутрипространственную структуру изображения, предлагается улучшенная матрица инцидентности, которая может научиться фиксировать долгосрочные внутрипространственные зависимости. Вместо евклидова расстояния между пространственными объектами используется взаимная корреляция пространственных объектов для вычисления вклада каждого узла в каждое гиперребро. Матрица инцидентности *H* содержит информацию о вкладе каждого узла в каждое гиперребро и выражается как:

Где - это линейное встраивание входных функций, за которым следует нелинейная функция активации (функция ReLU), – размерность вектора признаков после линейного встраивания, - диагональная матрица, которая помогает лучше изучить метрику расстояния между узлами для матрицы инцидентности *H* и, наконец, – величина, помогающая определить вклад каждого узла для каждого гиперребра, m - количество гиперребер в гиперграфе. , и - матрицы, зависящие от данных. реализуется сверткой 1 × 1 со входными признаками, реализуется путем объединения глобального среднего значения по каналам, за которым следует свертка 1 × 1 и используется Ω (X) для захвата глобальной взаимосвязи функций для создания лучших гиперребер (реализовано с использованием фильтра s × s, s = 7).

Матрица инцидентности вычисляется как показано на рисунке 4, и ее можно сформулировать как:

Где - карта признаков, созданная после глобального объединения входных признаков, , , - обучаемые параметры для линейного вложения. Чтобы избежать отрицательных значений в матрице инцидентности *H*, которые могут привести к мнимым значениям в матрицах степеней, в матрице инцидентности используются модули. Тогда слой свертки гиперграфа на пространственных признаках формулируется как:

Где – обучаемые параметры, – нелинейная функция активации, – входные признаки, – выходные признаки.

На рисунке 5 показано использование в HII двухступенчатой сетевой архитектуры от грубого к точному. “Грубая” сеть примерно заполняет недостающую область, которая наивно смешивается с входным изображением, а затем уточняющая сеть предсказывает более точные результаты с резкими краями. В уточненной сети используется слой гиперграфа с высокоуровневыми картами функций, чтобы увеличить восприимчивое поле сети и получить отдаленную глобальную информацию об изображении. Используются расширенные свертки для грубой и точной сетей, чтобы еще больше увнличить восприимчивое поле. В дополнение использовались стробированные свертки, чтобы улучшить нашу производительность на определенной нерегулярной маске. Также удаляются все слои пакетной нормализации и заменяются все слои свертки закрытой сверткой, с помощью которой обеспечивается локальная согласованность в завершенном изображении.

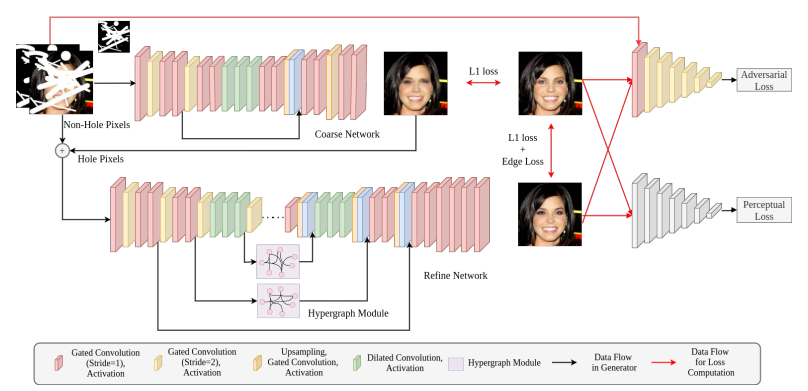


Рисунок 5 – “Грубая” сеть генерирует примерное заполнение недостающих областей. Позже “точная” сеть на основе свертки гиперграфа генерирует окончательно завершенное изображение высокого качества.

Нейронная сеть (также искусственная нейронная сеть, ИНС) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. ИНС представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Каждый нейрон суммирует входные сигналы на синапсах, через которые они перемножаются на соответствующие веса, а затем подает на некую функцию определенного вида, называемую функцией активацией. Выходное значение этой функции подается на выход нейрона в качестве входного сигнала на один из синапсов следующего нейрона, либо в качестве составляющей выходного сигнала всей нейронной сети. Благодаря методу обратного распространения ошибок, который корректирует веса нейронной сети в зависимости от величины ошибки в сравнении с истинным решением, нейронная сеть способна к обучению и адаптации к решениям различных задач.

К задачам обработки изображений лучше всего адаптированы сверточные нейронные сети. Свёрточная нейронная сеть (англ. convolutional neural network, CNN) — специальная архитектура искусственных нейронных сетей, нацеленная на эффективное распознавание образов. Использует некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Таким образом, идея свёрточных нейронных сетей заключается в чередовании свёрточных слоёв (англ. convolution layers) и субдискретизирующих слоёв (англ. subsampling layers или англ. pooling layers, слоёв подвыборки). Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения.

В данной работе было проведено исследование нескольких архитектур сверточной нейронной сети, каждая из которых выделена в программном коде в качестве отдельного класса. В качестве базовой библиотеки для создания нейронных сетей был использован PyTorch. Каждое изображение представляется как трехмерный тензор значений. Две размерности в нем отвечают за высоту и ширину, а третья – за значения трех цветов RGB. Изначально для исследования было взято два блока. Первый блок состоит из одно входного сверточного слоя, который увеличивает количество каналов во входном изображении с 3 (RGB) до 32 и одного скрытого, обрабатывающего 32-канальнй тензор. Второй блок содержит один выходной слой, который выводит приводит все значения к двум каналам, каждый из которых означает класс сегментации изображения: фон и ретушированная область. Значение в 32 канала взято как произвольное. После каждого слоя свертки была применена пакетная нормализация для приведения значений к диапазону, с которым работают нейроны, а после нее функция активация ReLU. Такая архитектура представлена в модуле decoder\_32.py.

Затем в процессе изучения к данной архитектуре было добавлено посередине два новых блока из двух сверточных слоев с такой же последовательностью пакетной нормализации и ReLU. Первый блок приводит 32-канальный тензор к 64-канальному и обрабатывает, а второй наоборот. Данная сеть представлена в файле decoder\_64.py. Симметрично между блоками в архитектуре расположены слои сжатия (субдискретизации) и расширения тензоров. Таким образов изображение, проходящее через каждый блок сверток. сначала становится меньшим по площади, но большим по числу каналов тензоров, а затем наоборот. Блоки, после которых идут слои сжатия можно для удобства назвать “кодирующими”, а те, после которых идут слои “расширения” – декодирующими.

Аналогичным образом нейронная сеть была расширена еще на два блока между предыдущими, но уже обрабатывающими 128 каналов, а затем еще на два с 256 каналами. Они представлены в файлах decoder\_128.py и decoder\_256.py. Далее была предпринята попытка преобразовать последнюю архитектуру в сеть вида U-Net, в котором результаты каждого кодирующего блока конкатенировались с входными тензорами в симметричных им декодирующих блоках. Данное решение представлено в файле UNet256.

Также была попытка изучить нейронную сеть с архитектурой SegNet, которая похожа на предыдущие, но во внутренних слоях имеющая по три слоя свертки. Данные решения представлены в файлах SegNet128.py и SegNet256.py.

Изначально формируется вручную или с помощью скрипта система папок, которая содержится в папке с названием objects. В этой папке может содержаться ряд папок, каждая из которых обозначает определенный вырезанный объект на изображении. В данном случае это 4 объекта: mantis (богомол), mapple (кленвый лист), vinca (цветок барвинка), wasp (оса).

В каждой папке объекта есть папки images, masks, CRA, DFN, HII. images - это изображения с объектами; masks - сегментированные маски объектов; CRA, DFN, HII - изображения с вырезанными и ретушированными разными методами объектами.

Сначала необходимо скопировать изображения и маски объектов в соответствующие папки. Далее применить скрипт prepare\_data\_object.py. Затем запустить DFN и CRA на сгенерированных изображениях. Затем применить скрипт invert.py, который меняет местами черную и белую область в масках. После этого запустить HII. Далее применить скрипт HII\_format\_results.py.

Затем необходимо запустить скрипт augmentation.py, который объединяет полученные изображения и маски в единый датасет с двумя папками - images и masks, а также применяет повороты под разными углами с целью добиться пространственной инвариантности при обучении нейронной сети.

Для обучения нейронной сети на датасете используется скрипт train\_model.py. В нем импортируется один из классов CNNNet, означающий определенную архитектуру сверточной нейросети. Импорт может происходить из decoder\_32.py, decoder\_64.py, decoder\_128.py, decoder\_256.py, UNet256.py. Также применялась попытка обучить архитектуры SegNet256.py и SegNet128.py, но функция потерь в них быстро застывает в определенном значении, возможно из-за умирания ReLU, причем использование LeakyReLU и Dropout не помогает. Видимо слишком глубокие сети тоже снижают эффективность обучения. После обучения нейронная сеть сохраняется в указанный файл.

Для проверки нейронной сети на тестовых данных используется скрипт validate\_model.py. Выводятся значения функции потерь, а затем итоговый процент случаев, в котором функция потерь была ниже заданного порога. Для генерации масок с детектированными случаями ретуширования с помощью предварительно обученной нейронной сети используется скрипт load\_model.py для определенной папки с изображениями. Из всего датасета изображений половина выделена для обучения, половина для тестирования.

Итоговые результаты для каждой архитектуры представлены в таблицах. Построчно расположено указание на какой выборке изображений нейронная сеть обучалась. Это может быть выборка изображений, реутшированных только одним из методов, либо всеми в совокупности. Столбцы означают на какой выборке изображений нейронная сеть тестировалась. Значения в ячейках означают процент случаев, в котором функция потерь, сравнивающая полученную маску изображения с исходной, ниже определенного порога, в данном случае 0.15. В качестве функции потерь взята BCEWithLogitsLoss() из PyTorch. Это функция по существу вычисляет бинарную кросс энтропию, считающейся по формуле:

,

где – это значение пикселя исходной маски, – значение пикселя полученной маски.

Здесь и далее представлены таблицы с результатами и графики с функциями потерь при обучении и тестировании на наборе данных ретушированным всеми тремя методами.

Таблица 1 – Процент успешных случаев распознавания ниже заданного порога при тестировании архитектуры decoder\_32

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CRA | DFN | HII | Все |
| CRA | 0.237 | 0.237 | 0.237 | 0.237 |
| DFN | 0.232 | 0.232 | 0.233 | 0.232 |
| HII | 0.229 | 0.23 | 0.232 | 0.23 |
| Все | 0.2315 | 0.23 | 0.231 | 0.231 |

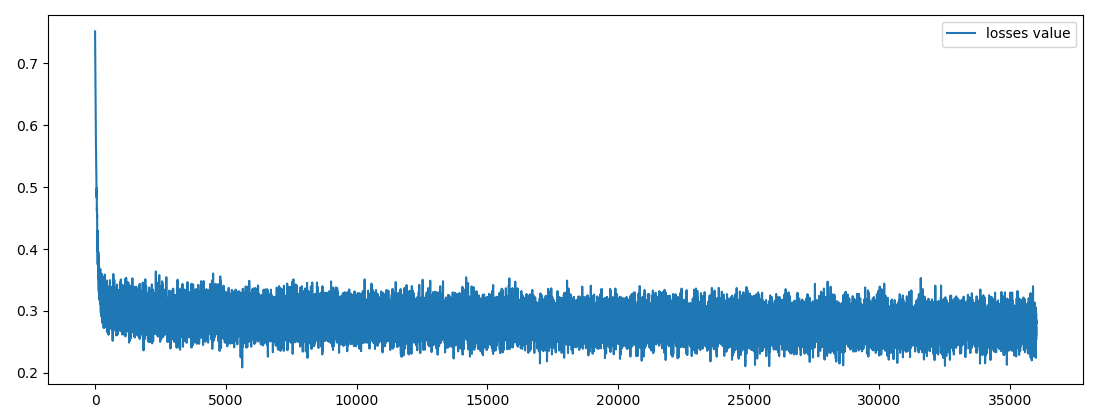


Рисунок 6 – График функции потерь при обучении архитектуры decoder\_32 на всем наборе данных ретушированным всеми тремя методами.

Таблица 2 – Процент успешных случаев распознавания ниже заданного порога при тестировании архитектуры decoder\_64

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CRA | DFN | HII | Все |
| CRA | 0.137 | 0.137 | 0.136 | 0.137 |
| DFN | 0.169 | 0.17 | 0.168 | 0.169 |
| HII | 0.159 | 0.162 | 0.166 | 0.162 |
| Все | 0.169 | 0.174 | 0.178 | 0.173 |

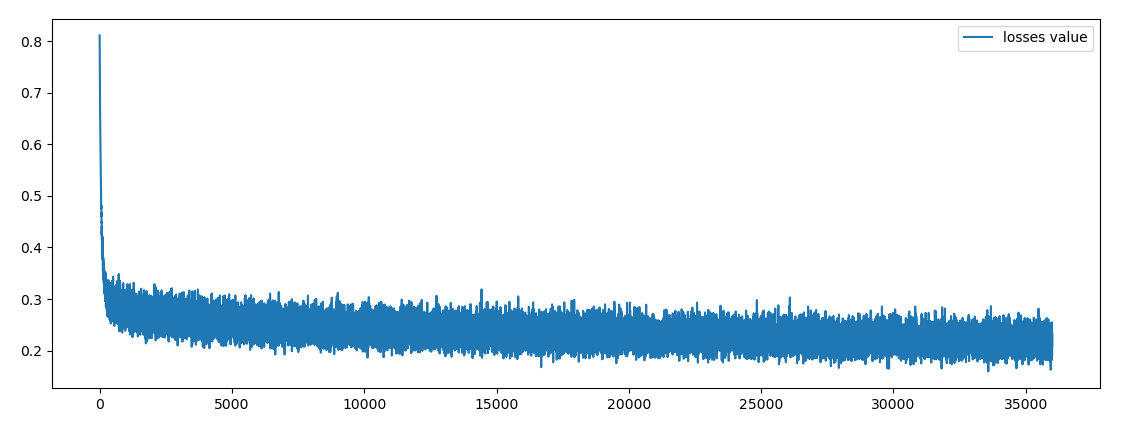


Рисунок 7 – График функции потерь при обучении архитектуры decoder\_64 на всем наборе данных ретушированным всеми тремя методами.

Таблица 3 – Процент успешных случаев распознавания ниже заданного порога при тестировании архитектуры decoder\_128

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CRA | DFN | HII | Все |
| CRA | 0.289 | 0.221 | 0.167 | 0.226 |
| DFN | 0.188 | 0.269 | 0.188 | 0.215 |
| HII | 0.172 | 0.212 | 0.284 | 0.223 |
| Все | 0.367 | 0.387 | 0.372 | 0.376 |

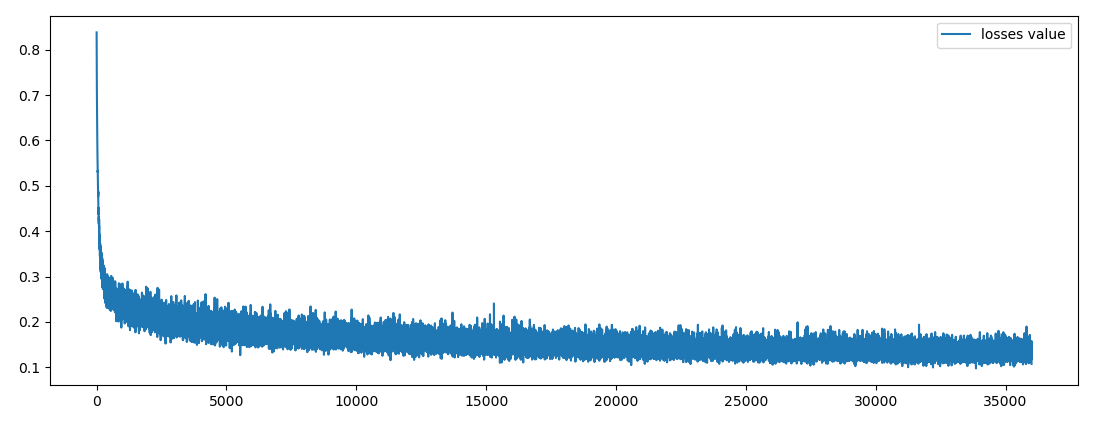


Рисунок 8 – График функции потерь при обучении архитектуры decoder\_128 на всем наборе данных ретушированным всеми тремя методами.

Таблица 4 – Процент успешных случаев распознавания ниже заданного порога при тестировании архитектуры decoder\_256

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CRA | DFN | HII | Все |
| CRA | 0.499 | 0.36 | 0.285 | 0.382 |
| DFN | 0.334 | 0.49 | 0.369 | 0.397 |
| HII | 0.241 | 0.363 | 0.478 | 0.36 |
| Все | 0.547 | 0.558 | 0.543 | 0.549 |

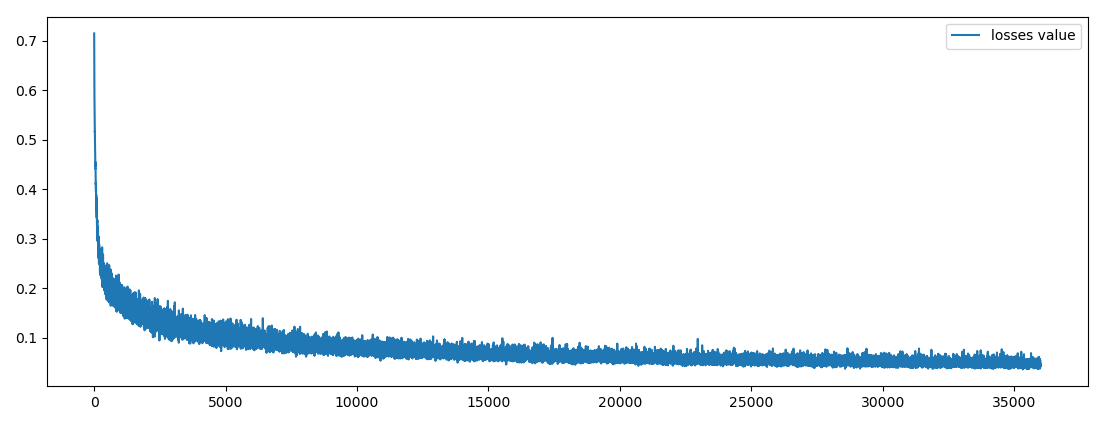


Рисунок 10 – График функции потерь при обучении архитектуры decoder\_256 на всем наборе данных ретушированным всеми тремя методами.

Таблица 5 – Процент успешных случаев распознавания ниже заданного порога при тестировании архитектуры UNet256

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CRA | DFN | HII | Все |
| CRA | 0.513 | 0.383 | 0.299 | 0.398 |
| DFN | 0.338 | 0.496 | 0.36 | 0.398 |
| HII | 0.214 | 0.329 | 0.483 | 0.342 |
| Все | 0.547 | 0.556 | 0.543 | 0.548 |

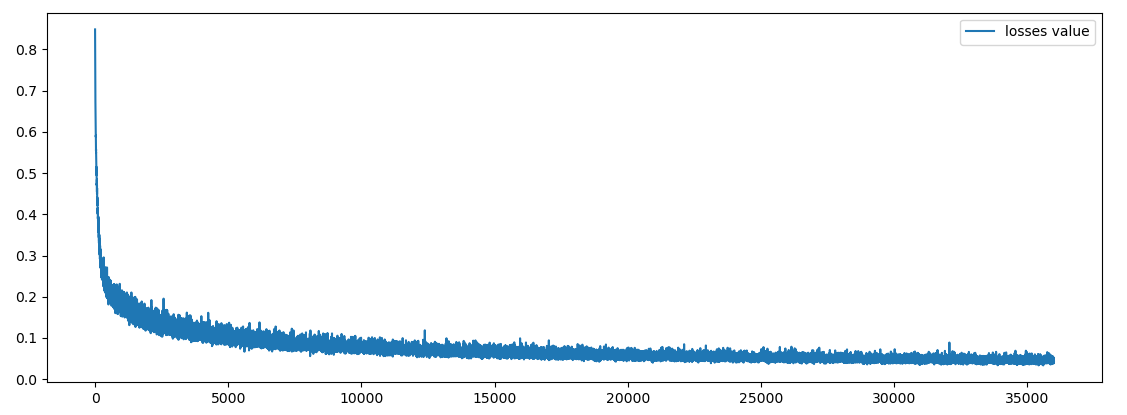


Рисунок 11 – График функции потерь при обучении архитектуры UNet256 на всем наборе данных ретушированным всеми тремя методами.

Для архитектур SegNet128 и SegNet256 таблицы не представлены, так как их функция потерь при обучении быстро застывает на крайне высоком значении и далее не меняется. Исходя из остальных таблиц, можно отметить что в итоге самой оптимальной архитектурой для задачи сегментации ретушированных областей оказались U-Net и decoder256. Видно, что с углублением числа слоев в целом результаты распознавания становятся лучше. При этом заметно различие между случаями, когда нейронная сеть обучается на наборе данных, где ретуширование происходило одним методом, и наборе данных, где применялись все три метода. Исходя из этих данных можно сделать вывод, что лучше обучать сеть на скомбинированном датасете.

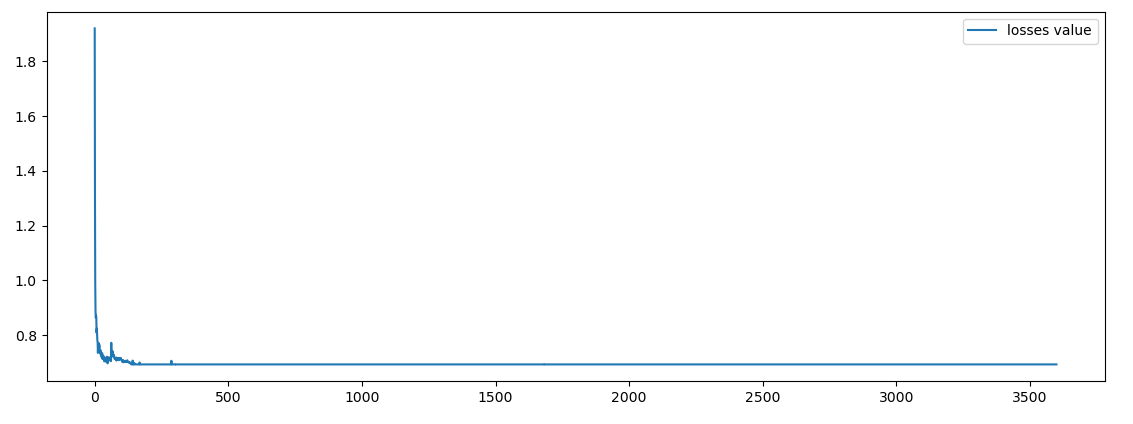


Рисунок 12 – График функции потерь при обучении архитектуры SegNet256 на всем наборе данных ретушированным всеми тремя методами. Ситуация при обучении SegNet128 выглядит также.



Рисунок 13 – Пример исходного изображения и заполнения области вместо объекта тремя способами слева направо: CRA, DFN, HII

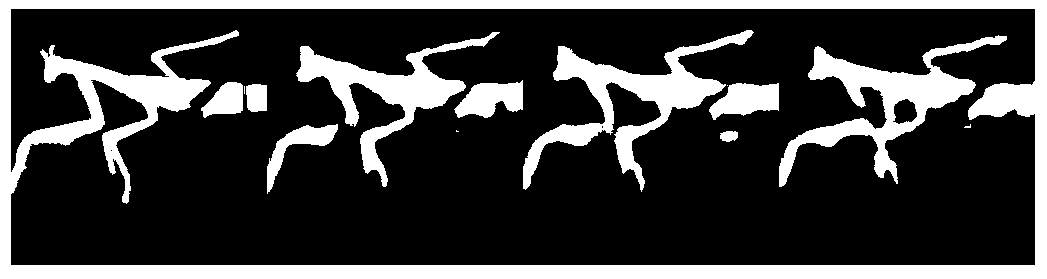


Рисунок 14 – Пример обнаружения зоны заполнения области вместо объекта. Слева находится исходная маска, по которой вырезался объект в изображении, далее слева направо маски, сгенерированные нейронной сетью для методов: CRA, DFN, HII

ЗАКЛЮЧЕНИЕ

В течение преддипломной успешно выполнены поставленные задачи: были изучены темы в области методов ретуширования изображений, а также исследован нейросетевой подход к обнаружению следов этих методов. Для этой задачи было реализовано ПО для генерации подходящих образцов изображений. Далее полученные образы были использованы с целью обучения и проверки архитектур сверточных нейронных сетей. По итогам работы подобрана наиболее оптимальная архитектура из исследованных.

За время прохождения практики освоены необходимые компетенции, в частности овладел знаниями об устройстве методов восстановления исходных данных в изображениях, опытом разработки программного обеспечения, касающейся данной тематики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Yi, Zili and Tang, Qiang and Azizi, Shekoofeh and Jang, Daesik and Xu, Zhan. Contextual Residual Aggregation for Ultra High-Resolution Image Inpainting. – IEEE/CVF, 2020.

2. Hong, Xin and Xiong, Pengfei and Ji, Renhe and Fan, Haoqiang. Deep Fusion Network for Image Completion. – ACM, 2019.

3. Wadhwa, Gourav and Dhall, Abhinav and Murala, Subrahmanyam and Tariq, Usman. Hyperrealistic Image Inpainting With Hypergraphs. – IEEE/CVF, 2021.

4. В.В. Мясников, А.В. Веричев. Реконструкция изображения как задача квадратичного программирования. – ИТНТ-2020, Самара, 2020.

5. David Josue Barrientos Rojas, Bruno Jose Torres Fernandes, Sergio Murilo Maciel Fernandes. A Review on Image Inpainting Techniques and Datasets. – IEEE, 2020.

6. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. – University of Freiburg, Germany, 2015.

ПРИЛОЖЕНИЕ

Код программы

Файл prepare\_data\_objects.py

import os

from PIL import Image

import numpy as np

import cv2

def format\_to\_256x256(path\_old, path\_new):

img = Image.open(path\_old)

width, height = img.size

if width > height:

left = int((width - height)/2)

top = 0

right = width - left

bottom = height

img = img.crop((left, top, right, bottom))

if width < height:

left = 0

top = int((height - width) / 2)

bottom = height - top

right = width

img = img.crop((left, top, right, bottom))

img = img.resize((256, 256), Image.ANTIALIAS)

img.save(path\_new)

path\_old = "C:\\Users\\maste\\Documents\\objects\\"

path\_new = "C:\\Users\\maste\\Documents\\objects\\"

mantis = "mantis\\"

mapple = "mapple\\"

vinca = "vinca\\"

wasp = "wasp\\"

L = [mantis, mapple, vinca, wasp]

images = "images\\"

masks = "masks\\"

DFN = "DFN\\"

CRA = "CRA\\"

HII = "HII\\"

def create\_dir():

os.mkdir(path\_new)

for object in L:

os.mkdir(path\_new + object)

os.mkdir(path\_new + object + images)

os.mkdir(path\_new + object + masks)

os.mkdir(path\_new + object + DFN)

os.mkdir(path\_new + object + CRA)

os.mkdir(path\_new + object + HII)

def create\_db():

for object in L:

image\_names = os.listdir(path\_old + object + images)

for name in image\_names:

print(path\_old + object + images + name)

format\_to\_256x256(path\_old + object + images + name, path\_new + object + images + name)

mask\_names = os.listdir(path\_old + object + masks)

for name in mask\_names:

print(path\_old + object + masks + name)

format\_to\_256x256(path\_old + object + masks + name, path\_new + object + masks + name)

def format\_mask(path\_old, path\_new):

img = Image.open(path\_old)

img = img.convert('RGB')

img = np.asarray(img)

black = [0, 0, 0]

white = [255, 255, 255]

r\_img, g\_img, b\_img = img[:, :, 0].copy(), img[:, :, 1].copy(), img[:, :, 2].copy()

img = r\_img + g\_img + b\_img

backgorund = black[0] + black[1] + black[2]

r\_img[img == backgorund] = white[0]

g\_img[img == backgorund] = white[1]

b\_img[img == backgorund] = white[2]

r\_img[img != backgorund] = black[0]

g\_img[img != backgorund] = black[1]

b\_img[img != backgorund] = black[2]

result = np.dstack([r\_img, g\_img, b\_img])

result = Image.fromarray(result.astype(np.uint8))

result.save(path\_new)

image = cv2.imread(path\_new)

processed\_image = cv2.medianBlur(image, 3)

cv2.imwrite(path\_new, processed\_image)

def apply\_format\_to\_all():

for object in L:

mask\_names = os.listdir(path\_old + object + masks)

for name in mask\_names:

format\_mask(path\_new + object + masks + name, path\_new + object + masks + name)

print(path\_new + object + masks + name)

create\_dir()

create\_db()

apply\_format\_to\_all()

Файл invert.py

import os

from PIL import Image

import numpy as np

def img\_open(path):

img = Image.open(path)

img = img.convert('RGB')

img = np.asarray(img)

return img

def img\_save(img, path):

result = Image.fromarray(img.astype(np.uint8))

result.save(path)

def invert\_img(img):

new\_img = img.copy()

new\_img[img == 255] = 0

new\_img[img == 0] = 255

return new\_img

def norm\_img(img):

new\_img = img.copy()

new\_img[img >= 127] = 255

new\_img[img < 127] = 0

return new\_img

def apply\_invert\_to\_all(path\_mask):

L = os.listdir(path\_mask)

for l in L:

path = path\_mask + l

print(path)

img = img\_open(path)

img = norm\_img(img)

img = invert\_img(img)

img\_save(img, os.path.splitext(path)[0] + '.png')

name\_dir\_data = "objects"

path1 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mantis\\masks\\"

path2 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mapple\\masks\\"

path3 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\vinca\\masks\\"

path4 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\wasp\\masks\\"

apply\_invert\_to\_all(path1)

apply\_invert\_to\_all(path2)

apply\_invert\_to\_all(path3)

apply\_invert\_to\_all(path4)

Файл augmentation.py

import os

import shutil

from PIL import Image

def rot0(path\_read, path\_save):

shutil.copy(path\_read, path\_save)

return

def rot90(path\_read, path\_save):

img = Image.open(path\_read)

img = img.rotate(90)

img.save(path\_save)

return

def rot180(path\_read, path\_save):

img = Image.open(path\_read)

img = img.rotate(180)

img.save(path\_save)

return

def rot270(path\_read, path\_save):

img = Image.open(path\_read)

img = img.rotate(270)

img.save(path\_save)

return

def augmentation(masks, method1, method2, method3, path\_save\_images, path\_save\_masks, funcs):

M = []

M1 = []

M2 = []

M3 = []

for i in range(len(masks)):

M.append(os.listdir(masks[i]))

M1.append(os.listdir(method1[i]))

M2.append(os.listdir(method2[i]))

M3.append(os.listdir(method3[i]))

id = 48001

for i in range(len(M[0])):

print(i)

for j in range(len(M)):

for f in funcs:

f(masks[j] + M[j][i], path\_save\_masks + str(id) + '.png')

f(method1[j] + M1[j][i], path\_save\_images + str(id) + '.png')

id += 1

f(masks[j] + M[j][i], path\_save\_masks + str(id) + '.png')

f(method2[j] + M2[j][i], path\_save\_images + str(id) + '.png')

id += 1

f(masks[j] + M[j][i], path\_save\_masks + str(id) + '.png')

f(method3[j] + M3[j][i], path\_save\_images + str(id) + '.png')

id += 1

return

new\_dataset = "objects\_dataset"

path\_save\_images = "C:\\Users\\maste\\Documents\\" + new\_dataset + "\\images\\"

path\_save\_masks = "C:\\Users\\maste\\Documents\\" + new\_dataset + "\\targets\\"

os.mkdir("C:\\Users\\maste\\Documents\\" + new\_dataset)

os.mkdir(path\_save\_images)

os.mkdir(path\_save\_masks)

name\_dir\_data = "objects"

path\_mask\_1 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mantis\\masks\\"

path\_mask\_2 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mapple\\masks\\"

path\_mask\_3 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\vinca\\masks\\"

path\_mask\_4 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\wasp\\masks\\"

MASKS = [path\_mask\_1, path\_mask\_2, path\_mask\_3, path\_mask\_4]

path\_cra\_1 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mantis\\CRA\\"

path\_cra\_2 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mapple\\CRA\\"

path\_cra\_3 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\vinca\\CRA\\"

path\_cra\_4 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\wasp\\CRA\\"

CRA = [path\_cra\_1, path\_cra\_2, path\_cra\_3, path\_cra\_4]

path\_dfn\_1 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mantis\\DFN\\result\\"

path\_dfn\_2 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mapple\\DFN\\result\\"

path\_dfn\_3 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\vinca\\DFN\\result\\"

path\_dfn\_4 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\wasp\\DFN\\result\\"

DFN = [path\_dfn\_1, path\_dfn\_2, path\_dfn\_3, path\_dfn\_4]

path\_hii\_1 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mantis\\HII\\"

path\_hii\_2 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mapple\\HII\\"

path\_hii\_3 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\vinca\\HII\\"

path\_hii\_4 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\wasp\\HII\\"

HII = [path\_hii\_1, path\_hii\_2, path\_hii\_3, path\_hii\_4]

augmentation(MASKS, CRA, DFN, HII, path\_save\_images, path\_save\_masks, [rot0, rot90, rot180, rot270])

Файл HII\_format\_results.py

import cv2

import os

# отсекает в выводе HII финальное решение (четвертое изображение из склеенных)

def extract\_final\_images\_from\_HII(path):

images = os.listdir(path)

images = [path + file for file in images]

for name in images:

img = cv2.imread(name)

h1, h2 = img.shape[0], img.shape[1]

img = img[:, h2 - int(h2 \* 0.25):h2, :]

cv2.imwrite(name, img)

name\_dir\_data = "objects\_add"

path1 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mantis\\HII\\"

path2 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\mapple\\HII\\"

path3 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\vinca\\HII\\"

path4 = "C:\\Users\\maste\\Documents\\" + name\_dir\_data + "\\wasp\\HII\\"

extract\_final\_images\_from\_HII(path1)

print('mantis formated')

extract\_final\_images\_from\_HII(path2)

print('mapple formated')

extract\_final\_images\_from\_HII(path3)

print('vinca formated')

extract\_final\_images\_from\_HII(path4)

print('wasp formated')

Файл conf\_gen.py

import os

name\_dir = "objects"

path1 = "C:\\Users\\maste\\Documents\\" + name\_dir + "\\mantis\\"

path2 = "C:\\Users\\maste\\Documents\\" + name\_dir + "\\mapple\\"

path3 = "C:\\Users\\maste\\Documents\\" + name\_dir + "\\vinca\\"

path4 = "C:\\Users\\maste\\Documents\\" + name\_dir + "\\wasp\\"

def gen(path, filename):

f = open('samples/' + filename + '.txt', 'w')

images = os.listdir(path + "images\\")

masks = os.listdir(path + "masks\\")

for i in range(len(images)):

s = path + "images\\" + images[i] + ' ' + path + "masks\\" + masks[i] + '\n'

f.write(s)

gen(path1, 'conf\_mantis')

gen(path2, 'conf\_mapple')

gen(path3, 'conf\_vinca')

gen(path4, 'conf\_wasp')

Файл train\_model.py

import os

import torch

from torch.utils.data import Dataset, DataLoader

from torchvision import transforms

from skimage.io import imread

import matplotlib.pyplot as plt

from UNet256 import CNNNet

class SegmentationDataSet(Dataset):

def \_\_init\_\_(self, inputs: list, targets: list, transform=None):

self.inputs = inputs

self.targets = targets

self.transform = transform

self.inputs\_dtype = torch.float32

self.targets\_dtype = torch.float32

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

return len(self.inputs)

def \_\_getitem\_\_(self, index: int):

# Select the sample

input\_ID = self.inputs[index]

target\_ID = self.targets[index]

# Load input and target

x, y = imread(input\_ID), imread(target\_ID)

# Preprocessing

if self.transform is not None:

x, y = self.transform(x, y)

# y = np.expand\_dims(y[:, :, 0], axis=2)

y = y[:, :, 0]

y[y == 255] = 1

# Typecasting

x, y = torch.from\_numpy(x).type(self.inputs\_dtype), torch.from\_numpy(y).type(self.targets\_dtype)

return x, y

losses = []

def train(model, optimizer, loss\_fn, train\_data\_loader, epochs=20):

total\_steps = len(train\_data\_loader)

for epoch in range(epochs):

for i, (images, masks) in enumerate(train\_data\_loader, 1):

images = images.type(torch.FloatTensor)

images = images.reshape(images.shape[0], images.shape[3], images.shape[1], images.shape[2])

masks = masks.type(torch.FloatTensor)

# masks = masks.reshape(masks.shape[0], masks.shape[3], masks.shape[1], masks.shape[2])

images = images.cuda()

masks = masks.cuda()

optimizer.zero\_grad()

outputs = model.forward(images)

loss = loss\_fn(torch.max(outputs, 1)[0].float(), masks)

# Backward and optimize

loss.backward()

optimizer.step()

# if (i) % 100 == 0:

print("Epoch [" + str(epoch + 1) + "/" + str(epochs) + "], Step ["

+ str(i) + "/" + str(total\_steps) + "], Loss: " + str(loss.item()))

losses.append(loss.item())

def sort\_special(method, lst, len):

lst\_method = []

while method < len:

lst\_method.append(lst[method])

method += 3

return lst\_method

torch.cuda.device('cuda')

torch.cuda.empty\_cache()

model = CNNNet()

# model.load\_state\_dict(torch.load("model\_unet\_256")) # дообучение если надо

# model.eval()

model.cuda()

transforms = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

learning\_rate = 1e-4

batch\_size = 50

epochs = 50

N = 72000

path\_to\_images = "C:\\Users\\Admin\\Documents\\dataset\_object\\images\\"

path\_to\_masks = "C:\\Users\\Admin\\Documents\\dataset\_object\\targets\\"

inputs = os.listdir(path\_to\_images)

targets = os.listdir(path\_to\_masks)

inputs = sorted(inputs, key=lambda x: int(os.path.splitext(x)[0]))

targets = sorted(targets, key=lambda x: int(os.path.splitext(x)[0]))

inputs = inputs[:int(N/2)]

targets = targets[:int(N/2)]

inputs = [path\_to\_images + file for file in inputs]

targets = [path\_to\_masks + file for file in targets]

CRA = 0

DFN = 1

HII = 2

inputs\_CRA = sort\_special(CRA, inputs, int(N/2))

targets\_CRA = sort\_special(CRA, targets, int(N/2))

inputs\_DFN = sort\_special(DFN, inputs, int(N/2))

targets\_DFN = sort\_special(DFN, targets, int(N/2))

inputs\_HII = sort\_special(HII, inputs, int(N/2))

targets\_HII = sort\_special(HII, targets, int(N/2))

training\_dataset = SegmentationDataSet(inputs=inputs, targets=targets, transform=None) # менять в зависимости от обучающего датасета

training\_dataloader = DataLoader(dataset=training\_dataset, batch\_size=batch\_size, shuffle=True)

loss\_fn = torch.nn.BCEWithLogitsLoss().cuda()

optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)

print("start learning")

train(model, optimizer, loss\_fn, training\_dataloader, epochs=epochs)

torch.save(model.state\_dict(), "model\_unet\_256")

plt.plot(range(0, len(losses)), losses, label='losses value')

plt.legend()

plt.show()

Файл validate\_model.py

import os

import torch

from torch.utils.data import Dataset, DataLoader

from torchvision import transforms

from skimage.io import imread

import matplotlib.pyplot as plt

from UNet256 import CNNNet

class SegmentationDataSet(Dataset):

def \_\_init\_\_(self, inputs: list, targets: list, transform=None):

self.inputs = inputs

self.targets = targets

self.transform = transform

self.inputs\_dtype = torch.float32

self.targets\_dtype = torch.float32

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

return len(self.inputs)

def \_\_getitem\_\_(self, index: int):

# Select the sample

input\_ID = self.inputs[index]

target\_ID = self.targets[index]

# Load input and target

x, y = imread(input\_ID), imread(target\_ID)

# Preprocessing

if self.transform is not None:

x, y = self.transform(x, y)

# y = np.expand\_dims(y[:, :, 0], axis=2)

y = y[:, :, 0]

y[y == 255] = 1

# Typecasting

x, y = torch.from\_numpy(x).type(self.inputs\_dtype), torch.from\_numpy(y).type(self.targets\_dtype)

return x, y

losses = []

good = []

def validate(model, loss\_fn, train\_data\_loader, min\_fn, epochs=1):

total\_steps = len(train\_data\_loader)

for epoch in range(epochs):

for i, (images, masks) in enumerate(train\_data\_loader, 1):

images = images.type(torch.FloatTensor)

images = images.reshape(images.shape[0], images.shape[3], images.shape[1], images.shape[2])

masks = masks.type(torch.FloatTensor)

images = images.cuda()

masks = masks.cuda()

outputs = model.forward(images)

loss = loss\_fn(torch.max(outputs, 1)[0].float(), masks)

# if (i) % 100 == 0:

print("Epoch [" + str(epoch + 1) + "/" + str(epochs) + "], Step ["

+ str(i) + "/" + str(total\_steps) + "], Loss: " + str(loss.item()))

losses.append(loss.item())

if loss.item() < min\_fn:

good.append(loss.item())

def sort\_special(method, lst, len):

lst\_method = []

while method < len:

lst\_method.append(lst[method])

method += 3

return lst\_method

torch.cuda.device('cuda')

torch.cuda.empty\_cache()

model = CNNNet()

model.load\_state\_dict(torch.load("model\_unet\_256"))

model.eval()

model.cuda()

transforms = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

learning\_rate = 1e-4

batch\_size = 1

epochs = 1

min\_fn = 0.1

N = 72000

path\_to\_images = "C:\\Users\\Admin\\Documents\\dataset\_object\\images\\"

path\_to\_masks = "C:\\Users\\Admin\\Documents\\dataset\_object\\targets\\"

inputs = os.listdir(path\_to\_images)

targets = os.listdir(path\_to\_masks)

inputs = sorted(inputs, key=lambda x: int(os.path.splitext(x)[0]))

targets = sorted(targets, key=lambda x: int(os.path.splitext(x)[0]))

inputs = inputs[int(N/2):]

targets = targets[int(N/2):]

inputs = [path\_to\_images + file for file in inputs]

targets = [path\_to\_masks + file for file in targets]

CRA = 0

DFN = 1

HII = 2

inputs\_CRA = sort\_special(CRA, inputs, int(N/2))

targets\_CRA = sort\_special(CRA, targets, int(N/2))

inputs\_DFN = sort\_special(DFN, inputs, int(N/2))

targets\_DFN = sort\_special(DFN, targets, int(N/2))

inputs\_HII = sort\_special(HII, inputs, int(N/2))

targets\_HII = sort\_special(HII, targets, int(N/2))

training\_dataset = SegmentationDataSet(inputs=inputs, targets=targets, transform=None)

training\_dataloader = DataLoader(dataset=training\_dataset, batch\_size=batch\_size, shuffle=True)

loss\_fn = torch.nn.BCEWithLogitsLoss().cuda()

print("start validating")

validate(model, loss\_fn, training\_dataloader, min\_fn, epochs=epochs)

plt.plot(range(0, len(losses)), losses, label='losses value')

plt.legend()

plt.show()

print(len(good)/len(inputs))

Файл load\_model.py

import os

import torch

import numpy as np

from PIL import Image

from UNet256 import CNNNet

def invert(img):

img[img < 127] = 1

img[img >= 127] = 2

img[img == 1] = 255

img[img == 2] = 0

return img

def detect(model, path\_read, path\_write):

test\_img = Image.open(path\_read)

test\_img = test\_img.convert('RGB')

test\_img = np.asarray(test\_img)

test\_img = np.expand\_dims(test\_img, axis=0)

test\_img = torch.from\_numpy(test\_img)

test\_img = test\_img.type(torch.FloatTensor)

test\_img = test\_img.reshape(test\_img.shape[0], test\_img.shape[3], test\_img.shape[1], test\_img.shape[2])

result\_img = model.forward(test\_img)

result\_img = torch.max(result\_img, 1)[0]

result\_img = result\_img.detach().numpy()

result\_img = np.squeeze(result\_img, axis=0)

result\_img = result\_img.astype(np.uint8)

result\_img = invert(result\_img)

result = Image.fromarray(result\_img)

result.save(path\_write)

model = CNNNet()

model.load\_state\_dict(torch.load("model\_unet\_256\_x5"))

model.eval()

N = 72000

path\_to\_images = "C:\\Users\\Admin\\Documents\\dataset\_object\\images\\"

path\_to\_save = "C:\\Users\\Admin\\Documents\\dataset\_object\\predicted\\"

inputs = os.listdir(path\_to\_images)

inputs = sorted(inputs, key=lambda x: int(os.path.splitext(x)[0]))

inputs\_read = [path\_to\_images + file for file in inputs]

os.mkdir(path\_to\_save)

for i in range(len(inputs)):

name\_read = inputs\_read[i]

name\_write = path\_to\_save + inputs[i]

detect(model, name\_read, name\_write)

Файл decoder\_32.py

import torch.nn as nn

class CNNNet(nn.Module):

def \_\_init\_\_(self):

super(CNNNet, self).\_\_init\_\_()

self.block1 = nn.Sequential(

nn.Conv2d(3, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.final = nn.Conv2d(32, 2, kernel\_size=(3, 3), padding=1)

def forward(self, x):

x = self.block1(x)

x = self.final(x)

return x

Файл decoder\_ 64.py

import torch.nn as nn

class CNNNet(nn.Module):

def \_\_init\_\_(self):

super(CNNNet, self).\_\_init\_\_()

self.block1 = nn.Sequential(

nn.Conv2d(3, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.pool1 = nn.MaxPool2d((2, 2))

self.block2 = nn.Sequential(

nn.Conv2d(32, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

)

self.up1 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block3 = nn.Sequential(

nn.Conv2d(64, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.final = nn.Conv2d(32, 2, kernel\_size=(3, 3), padding=1)

def forward(self, x):

x = self.pool1(self.block1(x))

x = self.up1(self.block2(x))

x = self.block3(x)

x = self.final(x)

return x

Файл decoder\_128.py

import torch.nn as nn

class CNNNet(nn.Module):

def \_\_init\_\_(self):

super(CNNNet, self).\_\_init\_\_()

self.block1 = nn.Sequential(

nn.Conv2d(3, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.pool1 = nn.MaxPool2d((2, 2))

self.block2 = nn.Sequential(

nn.Conv2d(32, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

)

self.pool2 = nn.MaxPool2d((2, 2))

self.block3 = nn.Sequential(

nn.Conv2d(64, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

)

self.up1 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block4 = nn.Sequential(

nn.Conv2d(128, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

)

self.up2 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block5 = nn.Sequential(

nn.Conv2d(64, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.final = nn.Conv2d(32, 2, kernel\_size=(3, 3), padding=1)

def forward(self, x):

x = self.pool1(self.block1(x))

x = self.pool2(self.block2(x))

x = self.up1(self.block3(x))

x = self.up2(self.block4(x))

x = self.block5(x)

x = self.final(x)

return x

Файл decoder\_256.py

import torch.nn as nn

class CNNNet(nn.Module):

def \_\_init\_\_(self):

super(CNNNet, self).\_\_init\_\_()

self.block1 = nn.Sequential(

nn.Conv2d(3, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.pool1 = nn.MaxPool2d((2, 2))

self.block2 = nn.Sequential(

nn.Conv2d(32, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

)

self.pool2 = nn.MaxPool2d((2, 2))

self.block3 = nn.Sequential(

nn.Conv2d(64, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

)

self.pool3 = nn.MaxPool2d((2, 2))

self.block4 = nn.Sequential(

nn.Conv2d(128, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.ReLU(),

nn.Conv2d(256, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.ReLU(),

)

self.up1 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block5 = nn.Sequential(

nn.Conv2d(256, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

)

self.up2 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block6 = nn.Sequential(

nn.Conv2d(128, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

)

self.up3 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block7 = nn.Sequential(

nn.Conv2d(64, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.final = nn.Conv2d(32, 2, kernel\_size=(3, 3), padding=1)

def forward(self, x):

x = self.pool1(self.block1(x))

x = self.pool2(self.block2(x))

x = self.pool3(self.block3(x))

x = self.up1(self.block4(x))

x = self.up2(self.block5(x))

x = self.up3(self.block6(x))

x = self.block7(x)

x = self.final(x)

return x

Файл UNet256.py

import torch.nn as nn

import torch

class CNNNet(nn.Module):

def \_\_init\_\_(self):

super(CNNNet, self).\_\_init\_\_()

self.block1 = nn.Sequential(

nn.Conv2d(3, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.pool1 = nn.MaxPool2d((2, 2))

self.block2 = nn.Sequential(

nn.Conv2d(32, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

)

self.pool2 = nn.MaxPool2d((2, 2))

self.block3 = nn.Sequential(

nn.Conv2d(64, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

)

self.pool3 = nn.MaxPool2d((2, 2))

self.block4 = nn.Sequential(

nn.Conv2d(128, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.ReLU(),

nn.Conv2d(256, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.ReLU(),

)

self.up1 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block5 = nn.Sequential(

nn.Conv2d(384, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

)

self.up2 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block6 = nn.Sequential(

nn.Conv2d(192, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

)

self.up3 = nn.UpsamplingNearest2d(scale\_factor=2)

self.block7 = nn.Sequential(

nn.Conv2d(96, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

)

self.final = nn.Conv2d(32, 2, kernel\_size=(3, 3), padding=1)

def forward(self, x):

out1 = self.block1(x)

out\_pool1 = self.pool1(out1)

out2 = self.block2(out\_pool1)

out\_pool2 = self.pool1(out2)

out3 = self.block3(out\_pool2)

out\_pool3 = self.pool1(out3)

out4 = self.block4(out\_pool3)

out\_up1 = self.up1(out4)

# return out\_up1

out5 = torch.cat((out\_up1, out3), dim=1)

out5 = self.block5(out5)

out\_up2 = self.up2(out5)

out6 = torch.cat((out\_up2, out2), dim=1)

out6 = self.block6(out6)

out\_up3 = self.up3(out6)

out7 = torch.cat((out\_up3, out1), dim=1)

out7 = self.block7(out7)

out = self.final(out7)

return out

Файл SegNet128.py

import torch.nn as nn

import torch

class CNNNet(nn.Module):

def \_\_init\_\_(self):

# p = 0.5

super(CNNNet, self).\_\_init\_\_()

self.down1 = nn.Sequential(

nn.Conv2d(3, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

# nn.Dropout2d(p),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

# nn.Dropout2d(p)

nn.MaxPool2d((2, 2))

)

self.down2 = nn.Sequential(

nn.Conv2d(32, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

# nn.Dropout2d(p),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

# nn.Dropout2d(p)

nn.MaxPool2d((2, 2))

)

self.down3 = nn.Sequential(

nn.Conv2d(64, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

# nn.Dropout2d(p),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

# nn.Dropout2d(p)

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

nn.MaxPool2d((2, 2))

)

self.up1 = nn.Sequential(

nn.UpsamplingNearest2d(scale\_factor=2),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

# nn.Dropout2d(p),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU(),

# nn.Dropout2d(p)

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.ReLU()

)

self.up2 = nn.Sequential(

nn.UpsamplingNearest2d(scale\_factor=2),

nn.Conv2d(128, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

# nn.Dropout2d(p),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.ReLU(),

# nn.Dropout2d(p)

)

self.up3 = nn.Sequential(

nn.UpsamplingNearest2d(scale\_factor=2),

nn.Conv2d(64, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

# nn.Dropout2d(p),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.ReLU(),

# nn.Dropout2d(p)

)

self.final = nn.Conv2d(32, 2, kernel\_size=(3, 3), padding=1)

def forward(self, x):

x = self.down1(x)

x = self.down2(x)

x = self.down3(x)

x = self.up1(x)

x = self.up2(x)

x = self.up3(x)

return x

Файл SegNet256.py

import torch.nn as nn

import torch

class CNNNet(nn.Module):

def \_\_init\_\_(self):

# p = 0.5

super(CNNNet, self).\_\_init\_\_()

self.down1 = nn.Sequential(

nn.Conv2d(3, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.LeakyReLU(),

# nn.Dropout2d(p),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.LeakyReLU(),

# nn.Dropout2d(p)

nn.MaxPool2d((2, 2))

)

self.down2 = nn.Sequential(

nn.Conv2d(32, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.LeakyReLU(),

# nn.Dropout2d(p),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.LeakyReLU(),

# nn.Dropout2d(p)

nn.MaxPool2d((2, 2))

)

self.down3 = nn.Sequential(

nn.Conv2d(64, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.LeakyReLU(),

# nn.Dropout2d(p),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.LeakyReLU(),

# nn.Dropout2d(p)

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.LeakyReLU(),

nn.MaxPool2d((2, 2))

)

self.down4 = nn.Sequential(

nn.Conv2d(128, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.LeakyReLU(),

# nn.Dropout2d(p),

nn.Conv2d(256, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.LeakyReLU(),

# nn.Dropout2d(p)

nn.Conv2d(256, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.LeakyReLU(),

nn.MaxPool2d((2, 2))

)

self.up1 = nn.Sequential(

nn.UpsamplingNearest2d(scale\_factor=2),

nn.Conv2d(256, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.LeakyReLU(),

# nn.Dropout2d(p),

nn.Conv2d(256, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.LeakyReLU(),

# nn.Dropout2d(p)

nn.Conv2d(256, 256, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(256),

nn.LeakyReLU()

)

self.up2 = nn.Sequential(

nn.UpsamplingNearest2d(scale\_factor=2),

nn.Conv2d(256, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.LeakyReLU(),

# nn.Dropout2d(p),

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.LeakyReLU(),

# nn.Dropout2d(p)

nn.Conv2d(128, 128, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(128),

nn.LeakyReLU()

)

self.up3 = nn.Sequential(

nn.UpsamplingNearest2d(scale\_factor=2),

nn.Conv2d(128, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.LeakyReLU(),

# nn.Dropout2d(p),

nn.Conv2d(64, 64, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(64),

nn.LeakyReLU(),

# nn.Dropout2d(p)

)

self.up4 = nn.Sequential(

nn.UpsamplingNearest2d(scale\_factor=2),

nn.Conv2d(64, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.LeakyReLU(),

# nn.Dropout2d(p),

nn.Conv2d(32, 32, kernel\_size=(3, 3), padding=1),

nn.BatchNorm2d(32),

nn.LeakyReLU(),

# nn.Dropout2d(p)

)

self.final = nn.Conv2d(32, 2, kernel\_size=(3, 3), padding=1)

def forward(self, x):

x = self.down1(x)

x = self.down2(x)

x = self.down3(x)

x = self.down4(x)

x = self.up1(x)

x = self.up2(x)

x = self.up3(x)

x = self.up4(x)

return x

**ОТЗЫВ**

**О ПРОХОЖДЕНИИ ПРАКТИКИ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вид практики | преддипломная | | | |
|  | (учебная, производственная, преддипломная) | | | |
| Срок прохождения практики: | | | 03.02.22-17.05.22 | |
| по направлению подготовки 10.05.03 информационная безопасность автоматизированных систем (специалитет) | | | |
| студентом группы № | | 6512-100503D Коноваловым Артёмом Николаевичем | | |
| Фамилия, имя, отчество | | | | |

|  |  |  |
| --- | --- | --- |
| № п/п | Критерии оценки | Оценка (по 5-бальной шкале) |
| 1. | Общая систематичность и ответственность работы в ходе практики |  |
| 2. | Достижения планируемых результатов практики |  |
| 3. | Корректность в сборе, анализе и интерпретации представляемых данных |  |
| 4. | Степень личного участия и самостоятельности практиканта в представляемом отчете о практике |  |
| 5. | Качество оформления отчетной документации |  |
|  | **ИТОГОВАЯ ОЦЕНКА[[1]](#footnote-1)** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель практики | |  |  |
| от профильной организации |  | |  |
| д.ф.-м.н., профессор кафедры ГИиИБ |  | | В.В. Мясников |
|  | | (подпись) |  |

1. Итоговая оценка выставляется как средняя арифметическая оценок по пяти критериям оценки [↑](#footnote-ref-1)