МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

Факультет «Математика и естественные науки»  
Кафедра «Прикладная математика и информационные технологии»

**ПРОГРАММНАЯ РЕАЛИЗАЦИЯ НЕЙРОСЕТЕВЫХ АЛГОРИТМОВ ДЛЯ ПЕРЕНОСА ХУДОЖЕСТВЕННЫХ СТИЛЕЙ В ИЗОБРАЖЕНИЯХ**(расчетно-пояснительная записка к выпускной квалификационной работе)

Направление подготовки 01.03.04 «Прикладная математика»

Профиль – «Математическое и программное обеспечение систем обработки информации и управления»

Утверждаю  
зав.кафедрой МОИС,  
д.т.н., профессор И.Г. Русяк

Руководитель работы  
к.т.н., доцент С.А. Бендер

Выполнил  
студент группы Б17-182-1 Г.В. Горбов

Ижевск 2021

**Техническое задание**

на выпускную квалификационную работу

студента группы Б17-182-1 Горбов Г.В.

1. **Тема**: “Программная реализация нейросетевых алгоритмов для переноса художественных стилей в изображениях”.
2. **Объект исследования**: художественные стили в изображениях, их математическая интерпретация и преобразование.
3. **Предмет исследования**: архитектуры сверточных нейронных сетей для решения задач классификации, совокупности их гиперпараметров.
4. **Цель работы**: программная реализация нейросетевых алгоритмов для переноса художественных стилей в изображениях с использованием методов машинного обучения и облачных средств.
5. **Состав задач**:
   1. Постановка задачи семантической сегментации изображения, переноса художественного стиля по двум исходным изображениям.
   2. Реализация алгоритма семантической сегментации.
   3. Реализация алгоритма переноса стиля.
6. **Программное обеспечение**: Windows, Google Сolab, Python 3.3.
7. **Критерий окончания работы**: Разработанные методы и алгоритмы решения задачи должны быть реализованы в виде расчетных программ. Результаты работы должны быть оформлены в виде отчета.

УТВЕРЖДАЮ

зав. кафедрой МОИС

д.т.н., профессор И.Г. Русяк

РУКОВОДИТЕЛЬ РАБОТЫ

к.ф.-м.н., доцент С.А. Бендер

ИСПОЛНИТЕЛЬ

студент гр. Б17-182-1 Г.В. Горбов

**РЕФЕРАТ**

Объем записки: 51 страниц, 29 рисунков, 2 табл., 20 библ. наим.

Ключевые слова: нейросеть, сверточные слои, перенос стиля, сегментация, архитектура нейронной сети, машинное обучение.

Работа посвящена программной реализации нейросетевых алгоритмов для переноса стиля. Объектом изучения и анализа являются художественные стили в изображениях, их математическая интерпретация и преобразование. Для решения задачи как переноса стиля так и семантической сегментации применяется технология нейронных сетей.

В работе изложена теоретическая основа построения модели и ее решения: проведены эксперементы для получения и выбора наилучшей модели, приведены результаты работы разработанного алгоритма.

В работе содержатся программная реализация алгоритма в среде Google colab, получены графики ошибок. А так же описаны ситуации и целесообразность вариации параметров алгоритма в зависимости от этих ситуаций.

**ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ**

Объектом исследования являются художественные стили в изображениях, их математическая интерпретация и преобразование.

Актуальность темы обусловлена рядом причин.

1. Все более активно развивающийся область IT требующая решения более интеллектуальных задач.
2. Необходимость решения чисто визуальных задач, которые решались до этого момента лишь человеком.

Цель работы программная реализация нейросетевых алгоритмов для переноса художественных стилей в изображениях с использованием методов машинного обучения и облачных средств.

Методика работы основана на использовании численных методов оптимизации, нейросетевых технологиях.

На защиту выносятся

Математическая модель решения задачи. Готовая программная реализация алгоритма по переносу стиля написанная на языке Python в среде Google Colab.

Научная новизна. Предложенный алгоритм позволяет решать множество задач в развлекательной сфере.

Практическая ценность. Разработанный программный комплекс осуществляет решение визуальных, художественных задач.

Публикации. По результатам работы написаны отчёт и статья для ‘Липановские чтения’, а так же представлена презентация на ‘Выставке инноваций’ по направлению Цифровые технологии.

Структура и объем работы. Работа содержит введение, три главы и заключение, изложенные на 51 страницах. В работу включены 29 рисунок, 2 таблицы и список литературы из 20 наименований.

ОГЛАВЛЕНИЕ

[ОСНОВНЫЕ ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ](#_Toc43825454) 6

[ВВЕДЕНИЕ](#_Toc43825454) 7

[1. Математическая постановка задачи.](#_Toc43825455) 9

[1.1. Математическая постановка задачи для сегментации](#_Toc43825456) 9

[1.2. Математическая постановка для задачи переноса стиля. 10](#_Toc43825457)

[2. Сгементация изображения.](#_Toc43825458) 11

[2.1. Сверточные нейронные сети.](#_Toc43825459) 12

[2.2. Выбор модели для решения задачи сегментации.](#_Toc43825460) 18

[3. Нейронный перенос стиля](#_Toc43825461) 21

[3.1. Особенность сверточных нейронных сетей 21](#_Toc43825462)

[3.2. Выбор архитектуры для решения задачи переноса стиля 2](#_Toc43825463)4

[3.3. Определение функции потерь модели 2](#_Toc43825464)5

[4. Определение гиперпараметров модели](#_Toc43825465) 29

[4.1. Описание алгоритма обучения модели 3](#_Toc43825466)2

[5. Интеграция сегментации в алгоритм переноса стиля 3](#_Toc43825466)3

[6. Анализ результатов исследования. Отображение результатов](#_Toc43825466) 35

[ВЫВОДЫ 4](#_Toc43825467)1

[СПИСОК ЛИТЕРАТУРЫ 4](#_Toc43825468)2

[ПРИЛОЖЕНИЕ 4](#_Toc43825468)4

# ОСНОВНЫЕ ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

;

;

;

;

;

;

;

MASK – результат преобразованного выхода сегментации , двумерная матрица из 0 и 1, размера входного изображения;

Module – класс pytorch предоставляющий удобный интерфейс для формирования архитектуры нейронных сетей средствам наследования от него;

optimizer – обьект класса pytorch optim, реализующий конкретный численный метод оптимизации;

AdamW – алгоритм численной оптимизации (модификация градиентного спуска);

LBFGS – алгоритм численной оптимизации (стохастический);

# ВВЕДЕНИЕ

Задачи и предложенные к ним алгоритмы CV(computer vision) области машинного обучения имеют в нынешнее время огромную популярность. Такие алгоритмы очень часто применяются в различных междисциплинарных областях в которых предполагается, что алгоритмы должны обладать человеческой интеллектуальностью и творчеством. В данном проекте будет рассмотрена одна из таких задач – перенос стиля на изображениях на заранее распознанные обьекты.

Человеку и высшим животным буквально на каждом шагу приходится распознавать, принимать решения и обучаться. Нейросетевой подход возник из стремления понять, каким образом мозг решает столь сложные задачи, и реализовать эти принципы в автоматических устройствах. Пока искусственные нейронные сети (artificial neural networks, ANN) являются лишь предельно упрощёнными аналогами естественных нейронных сетей. Нервные системы животных и человека гораздо сложнее тех устройств, которые можно создать с помощью современных технологий. Однако для успешного решения многих практических задач оказалось вполне достаточно «подсмотреть» лишь общие принципы функционирования нервной системы. Некоторые разновидности ANN представляют собой математические модели, имеющие лишь отдалённое сходство с нейрофизиологией, что отнюдь не препятствует их практическому применению. Кроме задач экстраполяции многомерных функций искусственные нейронные сети применяются во многих задачах распознавания образов и прогнозирования. С помощью нейронных сетей распознаются изображения, печатные и рукописные тексты; другие сети позволяют сжимать данные и управлять сложными техническими системами; в экономике нейронные сети используются для прогнозирования поведения финансовых рынков, в медицине в задачах классификации болезней и т.д. Сегодня созданием и обучением искусственных нейронных сетей занимаются крупные корпорации и научные центры. Они разрабатывают специализированные нейросетевые программные средства. Методы обработки данных с помощью нейронных сетей также включены в математические программные комплексы и в прикладные статистические пакеты.

В ходе работы были предложены математические определения таких интуитивных понятий как ‘стиль изображения’, вычисление расстояний между стилями изображений.

В ходе данной работы поставлена задача сегментации, переноса художественного стиля, реализован алгоритм семантической сегментации, а так реализован алгоритм, позволяющий численными методами оптимизации, разрешать достаточно интуитивную задачу человеческого творчества. Были исследование работы различных моделей, которые могли бы решать выше описанную задачу и так же был произведен выбор среди них той, что имеет наилучший результат. Произведено обучение выбранных моделей, при помощи средств электронного облака, представлен результат по работе конечного алгоритма, и оформление его в удобный интерфейс для упрощения использования.

# Математическая постановка задачи

# Математическая постановка работы состоит из двух частей. Первая часть – это математическая постановка задачи сегментации. Под сегментацией подразумевается задача семантической сегментации машинного обучения, в данном случае ожидается по окончанию этой части получить матрицу(изображение) с такими же размерами как и исходное изображение, но имеющее 2 цвета (2 значения элементов) 0 и 1, где 1 будет отвечать за ту область изображения в которой содержится предполагаемый обьект, тогда как значение 0 будет отвечать за фон.

# Вторая часть – это математическая постановка задачи переноса стиля.

* 1. **Математическая постановка для задачи сегментации**

Пусть имеется множество изображений каждый элемент которого является (RGB изображением), а так же множество ответов *Y* где *,* каждый ответ является тензором, каждый канал которого отвечает за свой класс. Пусть так же имеется функция

Требуется по заранее заданной выборке определить функцию

Которая бы наилучшим образом аппроксимировала функцию *f* при помощи рассмотрения семейства функций *,* где  *,*  (m,n – размер входных изображений RGB) и нахождения *,* которая бы разделяла пиксили входного изображения x в соответствии с заранее заданными классами. Где поиск точки осуществляется при помощи определения функции ошибок , от параметров и матриц и решения задачи где Q – эмпирический риск.

* 1. **Математическая постановка для задачи переноса стиля**

Пусть имеется множество изображений каждый элемент которой является (RGB изображением), а так же множество ответов *Y* где . Пусть так же имеется функция

Требуется по заданному изображению составить алгоритм определения из множества параметрических моделей:

где (*m,n* – размер входных изображений RGB)

Функции *,* которая бы наилучшим образом аппроксимировала функцию *S*. Где поиск точки осуществляется при помощи определения функции ошибок , от параметров и матриц , а так же алгоритма P численных методов оптимизации решается задача

где *Q* – эмпирический риск.

# Сегментация изображения

В соответствии с постановкой задачи сегментации, требуется определить функцию

которая бы наилучшим способом аппроксимировала имеющуюся зависимость

Где является (RGB изображением), а так же множество ответов *Y* где .

Так как мы работаем с изображениями, каждому обьекту-изображению мы будет сопоставлять тензор, либо матрицу в зависимость от цвета изображения.

Тоесть имеем .

Таким образом мы свели рассмотрение изображений к рассмотрению тензоров.

Определимся теперь с тем, что подразумевать под “наилучшей” аппроксимацией. В данном случае под функцией ошибок *L* будет пониматься так называемая Dice coefficient, формула для которой

Где *X*  является предсказанием, а *Y* правильно размеченной маской на текущем объекте. Минимизация функции *L = Dice* будет отвечать за ‘наилучшесть’ аппроксимации.

## . Сверточные нейронные сети

В нынешнее время имеет место быть ситуация, что лучшие наблюдаемые результаты как в области сегментации, как и в области детекции наилучшие результаты имеет тип нейросети - Convolutional Neural или сверточная нейронная сеть, которая вытекает в достаточно логичной мере из таких идейных архитектур как когнитрона и неокогнитрона. Дело в том, что такие результаты являются в достаточной мере обусловлены тем, что учитывается двумерная топология изображения, тогда как в других алгоритмах используется многослойный персептрон.

Слои нейронной сверточной сети имеют большую устойчивость к размерам изображения, к различным выбросам которые в них встречаются такие как расплывчивость обьектов, смещенная ориентация. Так же они представляют собой результат комозиции различных архитектурных подходов, которые в совокупности образуют сеть, устойчивую к выше описанным помехам на изображении. Эти подходы описаны как:

* Синаптические коэффициенты (обеспечивают детектирование некоторых черт в любом месте изображения и уменьшают общее число весовых коэффициентов);
* Организация с пространственными подвыборками иерархически организованная.
* Рецепторные локальные поля (обеспечивают локальную двумерную связность нейронов);

Можно утверждать, что в нынешнее время CNN и результаты тех различных преобразований, что были применены к ней являются наиболее быстрыми среди своих аналогов в решении задачи распознавание образов, а так же опережают их по качеству определения обьектов.

С 2012 года, на мероприятиях известной платформы ImageNet данные алгоритмы занимают лидирующие позиции. Приведем иллюстрацию.



Рисунок 2.1. Результаты соревнований на сайте ImageNet во времени

Convolutional Neural Network состоит из разных видов слоев: сверточные (convolutional) слои, субдискретизирующие (subsampling, подвыборка), а так же те слои, что используются в полносвязной нейронной сети – персептроны.

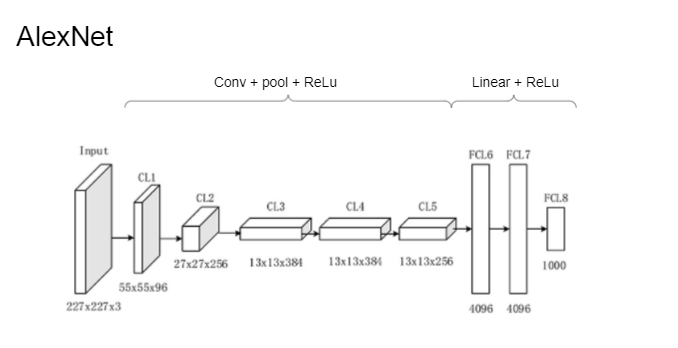


Рисунок 2.2. Архитектура сверточной сети AlexNet

Начальные слои Conv+pool+Relu, чередуясь между собой, формируют входной вектор признаков для многослойного персептрона.  
Свое название сверточная сеть получила по названию операции – свертка, суть которой будет описана дальше.

Сверточные сети являются удачной серединой между биологически правдоподобными сетями и обычным многослойным персептроном. На сегодняшний день лучшие результаты в распознавании изображений получают с их помощью. В среднем точность распознавания таких сетей превосходит обычные ИНС на 10-15%. СНС – это ключевая технология Deep Learning.   
 Основной причиной успеха СНС стало концепция общих весов. Несмотря на большой размер, эти сети имеют небольшое количество настраиваемых параметров по сравнению с их предком – неокогнитроном. Имеются варианты СНС (Tiled Convolutional Neural Network), похожие на неокогнитрон, в таких сетях происходит, частичный отказ от связанных весов, но алгоритм обучения остается тем же и основывается на обратном распространении ошибки. СНС могут быстро работать на последовательной машине и быстро обучаться за счет чистого распараллеливания процесса свертки по каждой карте, а также обратной свертки при распространении ошибки по сети.

На рисунке ниже продемонстрирована визуализация свертки и подвыборки:

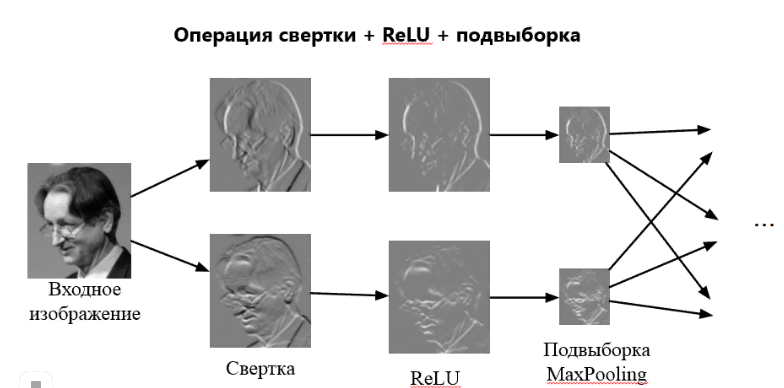


Рисунок 2.3. Визуализация свертки и подвыборки

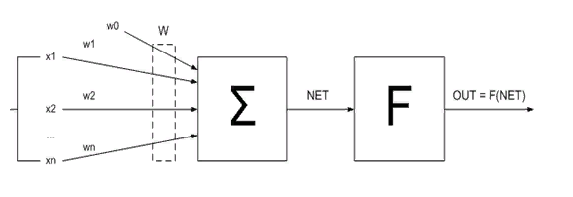


Рисунок 2.4. Модель нейрона

где *wi* – вес *i*  нейрона;

*xi* – выход *i* нейрона;

*w0* – вспомогательный параметр, смещение;

*n* – количество синаптических связей, входящих в нейрон.

Принцип работы сверточного слоя состоит в следующем. Изначально значения каждой карты сверточного слоя равны 0. Значения весов ядер задаются случайным образом в области от -0.5 до 0.5. Ядро скользит по предыдущей карте и производит операцию свертка, которая часто используется для обработки изображений, формула:

где *f* – исходная матрица изображения;

*g* – ядро свертки.

Неформально эту операцию можно описать следующим образом — окном размера ядра g проходим с заданным шагом (обычно 1) все изображение *f*, на каждом шаге поэлементно умножаем содержимое окна на ядро *g*, результат суммируется и записывается в матрицу результата, как на рисунке

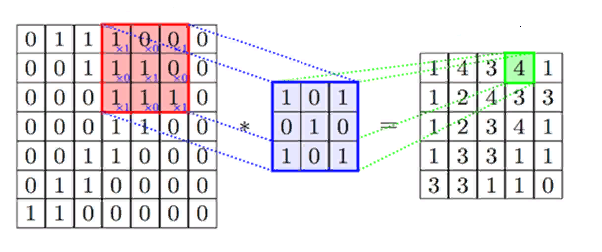


Рисунок 2.5. Операция свертки

Подвыборочный слой также, как и сверточный имеет карты, но их количество совпадает с предыдущим (сверточным) слоем, их 6. Смысл слоя состоит в том, чтобы уменьшить размер выхода предыдущего слоя. Если на предыдущей операции свертки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. В процессе сканирования ядром подвыборочного слоя (фильтром) карты предыдущего слоя, сканирующее ядро не пересекается в отличие от сверточного слоя. Обычно, каждая карта имеет ядро размером 2x2, что позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Вся карта признаков разделяется на ячейки 2х2 элемента, из которых выбираются максимальные по значению.

Обычно в подвыборочном слое применяется функция активации RelU. Операция подвыборки (или MaxPooling – выбор максимального) в соответствии с рисунком.

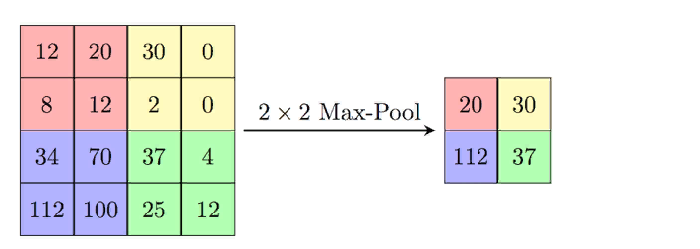


Рисунок 2.6. Операция подвыборки(Max Pooling)

## 2.2. Выбор модели для решения задачи сегментации

В качестве модели для решения задачи сегментации были рассмотрены уже известные алгоритмы, которые приведены в следующей таблице.

Таблица 1. Модели семантической сегментации

|  |  |  |
| --- | --- | --- |
| Network | mean IoU | global pixelwise acc |
| FCN ResNet50 | 60.5 | 91.4 |
| FCN ResNet101 | 63.7 | 91.9 |
| DeepLabV3 ResNet50 | 66.4 | 92.4 |
| DeepLabV3 ResNet101 | 67.4 | 92.4 |
| DeepLabV3 MobileNetV3-Large | 60.3 | 91.2 |
| LR-ASPP MobileNetV3-Large | 57.9 | 91.2 |

Для сегментации изображения мы будем использовать нейросеть ResNet101. Предлагается выбранную нейронную сеть загружать с предобученными весами, на 21 класс.

Таблица 2. Каналы на выходе ResNet101

|  |  |
| --- | --- |
| Индекс класса | Имя класса |
| 0. | фон |
|  | самолет |
|  | велосипед |
|  | птица |
|  | лодка |
|  | бутылка |
|  | автобус |
|  | машина |
|  | кошка |
|  | стул |
|  | корова |
|  | обеденный стол |
|  | собака |
|  | лошадь |
|  | мотоцикл |
|  | человек |
|  | растение в горшке |
|  | овца |
|  | диван |
|  | поезд |
|  | телевизор / монитор |

Данная таблица предоставлена в целях определения тех обьектов, которые алгоритм переноса стиля будет способен сегментировать. Так же предлагается рассмотреть более простую задачу бинарной сегментации. Модель рекомендуется загружать с параметром pretrained=True, то есть с обученными параметрами на датасетах ImageNet.

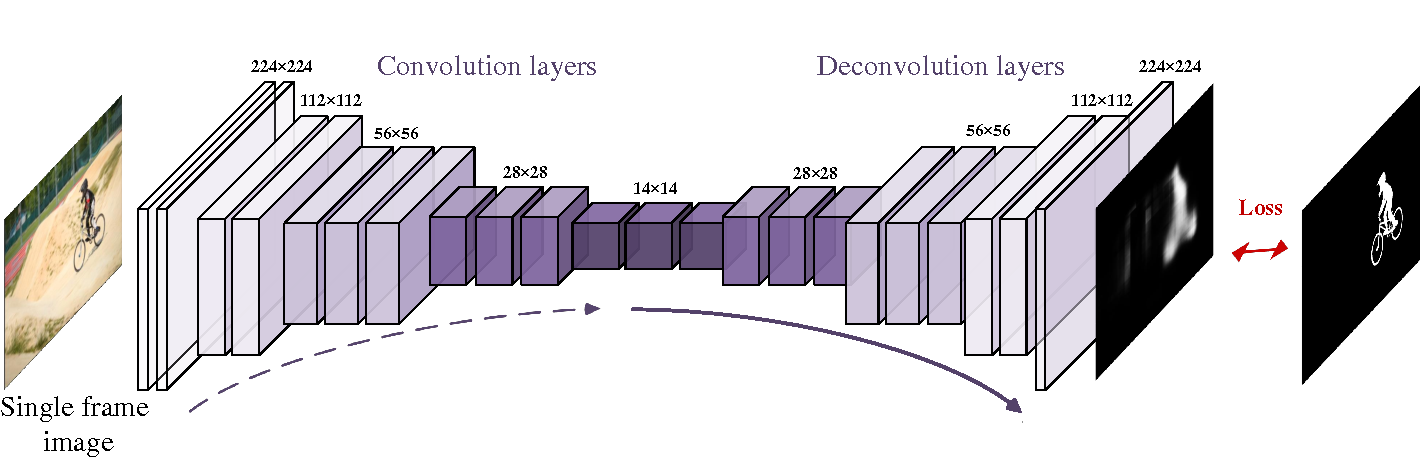
****

Рисунок 2.7. Архитектура ResNet101

Выбор модели основан на том, что такие популярные архитектуры как Unet и Segnet предназначены для распознавания более мелких обьектов в композиции с другими и более громоздких содержащихся в единственном экземпляре на изображении соответственно. По этой причине, так как заведомо не известно, что будет содержаться на изображении в силу общности задачи рекомендуется рассматривать архитектуру ResNet101, как то, что является чем то средним между двумя описанными.

# Нейронный перенос стиля

Для реализации алгоритма переноса стиля, нам потребуется для заранее подобранной архитектуре сети реализовывать функцию ошибки таким образом, чтобы потери во время обучения имели связь как с изображением content так и с изображением style.

В случае нейронного переноса стиля алгоритм будет строится следующим образом. Вначале требуется рассмотреть различные модели классификаторов , основанных на применении сверточных слоев нейросетей, а затем доизменять выбранную модель классификатора, что будет представлять из себя первый этап работы.

Второй этап работы – это внесение изменений в уже выбранную архитектуру классификатора и его адаптация для решения поставленной задачи. На этом этапе будет описан алгоритм получения значения функции потерь на конкретной итерации обучения, а так же использование этой информации для преобразования исходного изображения в целях снижения значения потери.

## 3.1. Особенность сверточных нейронных сетей

Следуя выше описанному алгоритму проектировки модели для решения задачи переноса стиля, начнем разработку первого этапа. По окончании этого этапа мы должны получить модель, которая наилучшим образом удовлетворяла нашим потребностям в дальнейшем наложении стиля. Итак этот этап основан на известном принципе работы сверточных слоев нейронной сети, а конкретно на том, что если мы имеем подряд некоторое количество слоев Где на первый слой будет подаваться изображение RGB являющееся тензором размерности 3, тоесть , то к примеру если *L1* задает переход с количества каналов 3(RGB) в количество каналов 64, затем *L2*: 64 в 128, *L3*: 128 в 256 и тд. То в такой конфигурации мы будем иметь выход после первого преобразования *L1* в виде тензора, с количеством каналов равным 64, тогда как высота и ширина каждого канала будет определяться в зависимости от таких параметров слоя *L1* как upsampling, размер ядра(фильтра) – kernel size, padding, dilation, groups и тд. На данный момент размерами высоты и ширины можно пренебречь. Принцип будет представлять из себя следующее: каждый канал после выхода конкретного слоя *Li* будет отвечать за распознавание каких либо структур в исходном изображении, которое было подано на вход нейронной сети. Рассмотрим рисунок

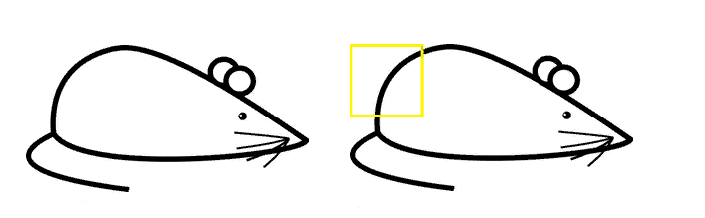


Рисунок 3.1. Оригинально изображение, визуализация фильтра на изображении.

Фильтры нейронной сети будут реагировать на эти структуры в исходном изображении и в соответствующих так называемых картах активации значения элементов будут большими по модулю, тогда как в остальных картах, которые отвечают за те признаки, которые не содержались в исходном изображении либо были в малом количестве будут содержать по модулю малые значения. Затем, так же стоит упомянуть то, что чем дальше ‘продвигается’ исходное изображение по сети тем более сложные структуры в этом изображении будут распознаваться, что означает чем глубже лежит карта активации, то тот признак, за который она отвечает будет более сложен. Пример значений фильтра на начальных слоях приведет на рисунке

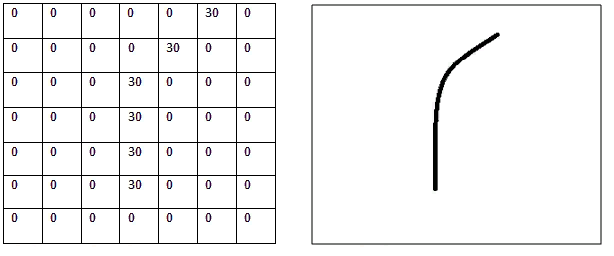


Рисунок 3.2. Визуализация детекции фильтром признака отвечающего за дугу.

Например: если мы будем запускать в нейронную сеть, содержащую свертку, изображение лица человека, то в самых первых слоях мы будем наблюдать, что карты активации, тоесть признаки отвечают за распознавание прямых горизонтальных, прямых вертикальных линий. Далее в иерархии признаков идут дуги, окружности и тд. Тогда как в конечных слоях, если нейросеть является например классификатором, уже обученым, карты активации будут отвечать за распознавание всего лица учитывая то, какие признаки были идентифицированы на предыдущих картах. Что можно иллюстрировать следующим изображением.

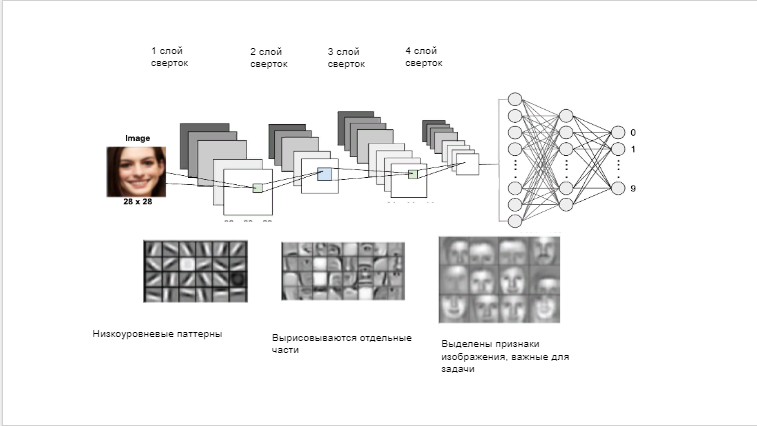


Рисунок 3.3. Пример иерархичности в сверточных слоях нейронной сети

Таким образом, на основе описанного принципа выведенного скорее эмпирическим путем нежели математическим(как и большинство различных методик машинного обучения) будет выстраиваться модель по переносу стиля.

## Выбор архитектуры для решения задачи переноса стиля

В качестве архитектуры нейронной сети былаи рассмотрены архитектуры VGG-19. Которые были разработаны для распознавания объектов на изображениях. В 2014-м году сеть VGG-16 на соревнованиях по распознаванию изображений базы ImageNet достигла небывалой на тот момент точности в 92,7%. И по этому показателю почти сравнялась с человеком. Теперь она выложена в открытом доступе (причем, с обученными весовыми коэффициентами), чтобы каждый желающий мог проводить с ней любые эксперименты.

Итак, общая структура сети VGG-16, следующая:

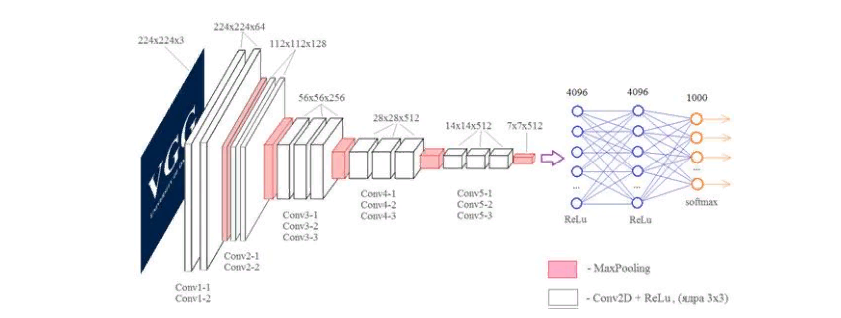


Рисунок 3.4. Архитектура VGG-19

На вход подает полноцветное изображение с тремя каналами: RGB. Затем, оно последовательно проходит через сверточные слои Conv1-1 и Conv1-2, каждый имеет по 64 фильтра с размером ядер 3х3. Далее, операция MaxPooling уменьшает вдвое линейные размеры карт признаков и результат обрабатывается двумя последовательными слоями Conv2-1 и Conv2-2 со 128 фильтрами с тем же размером ядер 3х3. И так далее. Потом через три сверточных слоя все проходит, потом еще через три.  Результирующий тензор размером 7х7х512 подается на полносвязную НС с 4096 нейронами двух скрытых слоев и 1000 нейронами выходного слоя. Число 1000 соответствует количеству классов, на которые эта сеть была обучена.

Преимуществом этой сети является ее относительная простота. А недостатком – медленная скорость обучения и большое количество весовых коэффициентов. Если их все сохранить на диск, то получится объем примерно 533 МБ.

## 3.3. Определение функции потерь модели

Функция ошибки L будет состоять из двух отдельных компонент, а более конкретно представляться следующей формулой.

Так, мы должны будем следить за тем, чтобы наше изображение не отставало от исходного стилевого изображения по стилю и от исходного контент изображения по контенту.

Теперь дадим математическое определение понятия стиля изображения, чтобы можно было находить схожесть между двумя изображениями по их стилю, в соответствии с интуитивным пониманием этого понятия. Согласно описанному выше принципу распознавания структур в изображениях, которые производят Convolutions neural layers, мы имеем, что при рассмотрении произвольного слоя сети, предположим это будет слой переводящий 64 карт активации в 128 карт. Тогда каждая из 128 карт – это будет матрица, которая реагирует на проявление какого – либо конкретного признака в изображении. Таким образом для удобства преобразуем тензор с количеством каналов 128 в матрицу, путем развертывания каждой матрицы в вектор, а затем составлением из них двумерной матрицы. Проиллюстрируем эту операцию рисунком.

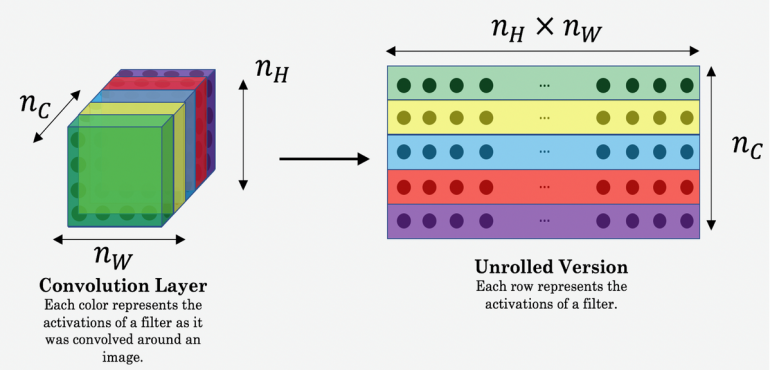


Рисунок 3.5. Преобразование карт активаций с произвольного слоя

Затем при помощи стандартного скалярного произведения между векторами в Евклидовом пространстве, будем находить корреляцию между каждой парой признаков, для обнаружения того, насколько эти признаки одновременно часто встречаются на нашем изображении. По итогу этой операции будет получена так называемая Грамм матрица(скалярных произведений).

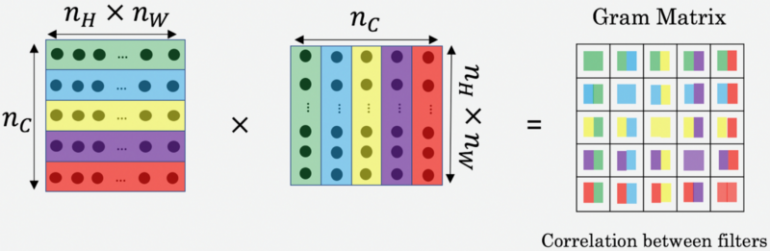


Рисунок 3.6. Получение Грамм матрицы

Собственно именно Грамм матрица и будем отвечать за стиль произвольного изображения – совокупность корреляции признаков, которые встречаются в этом изображении. Теперь остается лишь указать то как будет определяться схожесть между стилями изображений. Это будет происходить при помощи рассмотрения Евклидового расстояния между полученными Грамм матрицами двух изображений. Например имеются изображения из множества . Для определения того, насколько схожи два этих изображения по стилю, потребуется используя уже описанную измененную модель нейронной сети VGG19, прогнать эти два изображения по ней, а затем на последнем слое(который отвечает за распознавание сложных структур) по соответствующим картам активации изображения составить грамм матрицу , аналогично поступив и для изображения , получая . Затем находя Евклидовое расстояние между ними получаем число. И это число и будет отвечать за схожесть изображений по стилю. Так образом получаем конечную формулу.

Где

Определив то, что мы понимаем под стилем изображений и как находить схожие в смысле стиля изображения, перейдем к рассмотрению изображений схожих по контенту. Расстояние между контентом двух изображений , будем определять как расстояние между тензорами состоящими из карт активаций нейронной сети на конкретно выбранном слое. Тоесть если – i-й слой сверточной нейронной сети, где те карты активации, что подаются в нее, а те, что получаются на выходе, то если для , а для имеем , тогда Евклидово расстояние будет отвечать за схожесть по контенту между изображениями. Получаем расстояние по контенту

Где a(S), a(G) – результаты преобразования в двумерные матрицы соответственно, тоесть

И того добавляя параметры , которые выступают в качестве гиперпараметров модели и которые можно изменять, чтобы определять вклад каждой из компонент в общее значение функции потери.

После задания гиперпараметров модели переноса стиля, функции потери по стилю и по контенту, опишем принцип работы алгоритма обучения. Обучение происходит по количеству итераций – 500. На протяжении одного цикла, начинает по новому выстраивать архитектуру нейронной сети, по очередно добавляя слои исходной архитектуры с учетом того, что на конкретно выбранных из них будет реализовано взятие ошибки по стилю и контенту. По окончании формирования сети, начинается этап forward propogation, в котором исходное изображение по очередно проходит по каждому слою, который служит в роли операции отображающей исходный тензор в конечный. Таким образом получая вначале 64 затем 128, 256 , 512 карт активаций , происходит высчитывание ошибки на слоях с номерами, по следующим формулам. В рамках языка питон для определения ошибки используются классы ContentLoss и StyleLoss.

## 4. Определение гиперпараметров модели

В этой части работы опишем то, как именно мы будем изменять выбранный нами алгоритм классификатора, который нам необходимо адаптировать под нашу задачу. Начнем с указания гиперпараметров модели, которые были получены как результат многократных обучений модели с различными вариациями этих параметров. Первый предложенный гиперпараметр это произведение над архитектурой классификатора VGG19 следующих изменений: вначале избавимся от той части нейронной сети которая содержит перцептроны аналогичные простой полносвязной сети. Визуально теперь модель можно представить следующим образом.

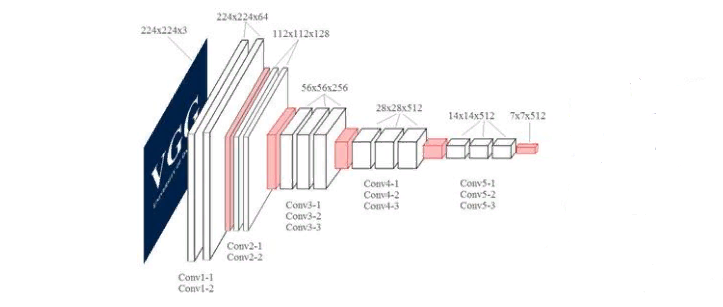


Рисунок 4.1. Измененная модель на основе архитектуры VGG19

Далее будем выбирать те слои нейронной сети с которых будет высчитываться ошибка по стилю и по контенту () conv1, conv2, conv3, conv4, conv5. Слой для потери по контенту – conv4.

Следующая группа гиперпараметров – это те которые будут задействованы непосредственно во время обучения. Рассмотрим такие параметры как алгоритм оптимизации, а так же количество итераций обучения алгоритма. Среди огромного количества вариаций алгоритмов численной оптимизации были выделены несколько, которые представляют наилучший результат. Визуализируем графически их работу.

Алгоритм оптимизации – AdamW(Метод стохастической оптимизации с регуляризацией распада несвязного веса) обучающая скорость(lr) = 0.001, количество итераций полагаем равным 750.

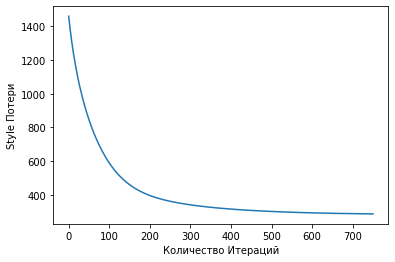


Рисунок 4.2. optimizer = AdamW, num\_iterations = 750

Выбранный алгоритм достаточно гладко минимизирует целевую функцию

Но при увеличении количества итераций происходит проигрыш по времени обучения около 20 секунд за каждые 200 итераций. Суммарное время обучения алгоритма 2 минуты. Наименьшее значение ошибки около 287.400146 по стилю.

Следующая вариация гиперпараметров, которая реализует хороший результат следующая – в качестве оптимизатора использовать алгоритм LBFGS(Алгоритм – Бройдена – Флетчера – Гольдфарба - Шанно), количество итераций обучения 300.

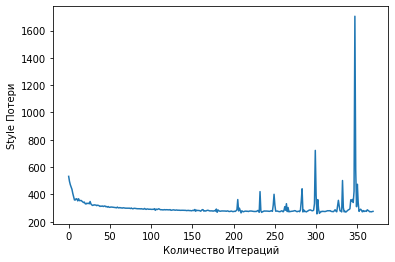


Рисунок 4.3. optimizer = LBFGA, num\_iterations = 400

В силу особенностей Алгоритма – Бройдена – Флетчера – Гольдфарба – Шанно, целевая функция имеет более резкий характер, что означает мы не можем, используя этот алгоритм, гарантировать гладкое понижение ошибки, но довольно малое ее значение будет достигаться на гораздо меньшем числе итераций относительно оптимизатора AdamW, как видно по рисунку для приемлимого значения ошибки количество итераций можно было поставить равным 150-300. Суммарное время обучения – 40 секунд. Наименьшее значение ошибки около 290 по стилю.

Таким образом исходя из приведенных опытов из которых выделена пара наиболее значимых предлагается данный алгоритм использовать со следующими гиперпараметрами обучения. Параметры , которые отвечают за вклад ошибки по стилю и по контенту будут равны 1000000 и 100. Так как исходный холст инициализируется контент изображением, то параметр отвечающий за потерю по контенту будет принимать довольно малое значение, так как его вклад в итоговую ошибку не существенен. Количество итераций обучения num\_steps полагаем равным 250. В качестве метода оптимизации целевой функции предлагается использовать алгоритм LBFGS(Алгоритм – Бройдена – Флетчера – Гольдфарба - Шанно).

## 4.1. Описание алгоритма обучения модели

Перед началом обучения модели, происходит этап подготовки исходного изображения. Три соответствующих изображения Input,Style,Content, загружаются в среду Google Colab, а затем, используя инструменты библиотеки PIL, а так же инструменты блиблиотеки torchvision.transforms определяется функция image\_loader, которая принмает на вход имя загруженного изображения в колаб. Данная процедура преобразует изображение в обьект PiL.Image , затем используя заранее заданный pileline имеющий название loader, сжимает изображение к размеру *Imsize=500* по ширине и высоте, и преобразует его в обьект torch.Tensor(матрицу). Далее по входному изображению загружается маска, которая будет отвечать за отделение фона от обьектов, так же определяется модель в переменную model. На этом оканчивается этап первичной обработки.

Опишем то, что происходит во время одной итерации цикла обучения. На поверхности он происходит в два этапа – определение функции closure, а затем получение оптимизатором(optimizer) этой функции в качестве аргумента, для реализации итерации алгоритма Бройдена – Флетчера – Гольдфарба - Шанно. Внутри функции closure, реализуется процедура forward propogation для высчитывания суммарной ошибки , которая накапливается с каждого слоя указанного гиперпараметром. Затем командой loss.backword() происходит получение градиентов. Таким образом по окончании каждой итерации происходит изменение параметров, которые являются пикселями исходного изображения так, чтобы суммарная ошибка становилась меньше.

Получаем график значений функции потерь от количества итераций . А так же график значений функции потерь от количества итераций .

## 5. Интеграция сегментации в алгоритм переноса стиля

Интегрировать предлагается следующим образом: вначале задается маска для заданного изображения-контента. При помощи алгоритма сегментации получаем тензор глубины 21 с матрицами размера , который затем преобразуем в 2D изображение, путем изменения каждого элемента заранее взятой матрицы на соответствующий наибольший элемент по каждому из каналов. . Затем заменяя каждый не нулевой элемент матрицы MASK на 1, получаем решение задачи бинарной классификации, то есть матрицу, которая будет отвечать за разделение обьекта изображение от фона.

Далее, так как в исходном алгоритме переноса стиля в сверточных слоях нейросети vgg19 три входных изображения style, input и content раскладываются на то количество карт активаций, которое соответственно указано в этих слоях, а затем в заранее подобранных слоях высчитываются ошибки по контенту и стилю, предлагается на этом этапе внести изменения в подсчет значений функции потери.

Так как в общем случае функция потерь имеет вид (1.2) Соответствующее вычисление следует изменить так, чтобы стиль на исходное изображение input, инициализируемое изображением content, накладывался только на его часть, а именно та матрица MASK, которую получается по итогу сегментации будет участвовать в роли шаблона. Таким образом после получения шаблона маски(MASK) рассматривая произвольный слой нейронной сети на котором будет подсчитываться ошибка по стилю, следует предпринять следующие действия. Шаблон маски первоначально масштабируется до размера матриц в канале тензора карт активаций. Далее производится поэлементное умножение каждого канала на полученный шаблон, что будет означать – распознавание структур изображения происходит только на пикселях вне обьекта выделенного сегментацией. Следующим этапом происходит вычисление ошибки, тоесть разложение тензора карт активаций на вектора и обьединение их в двумерную матрицу, для дальнейшего нахождения матрицы корреляций этих векторов(Грамм матрица). Визуально эту процедуру можно представить следующим образом:

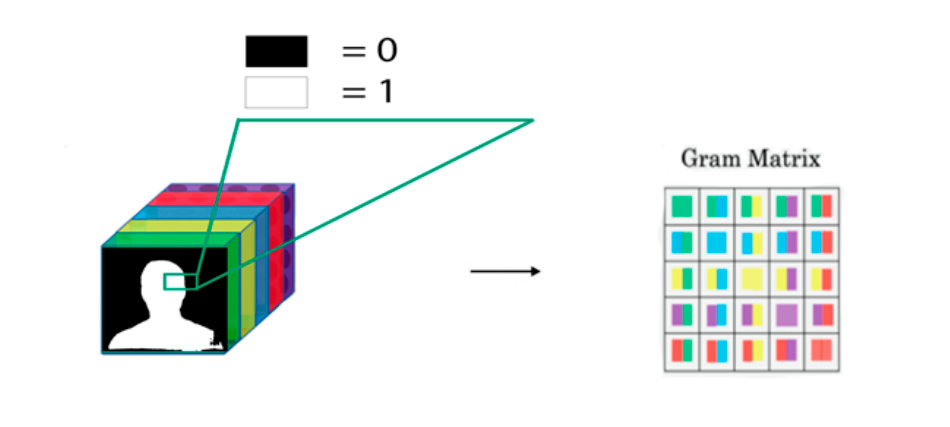


Рисунок 5.1. Преобразование карт активаций с произвольного слоя

В более формальной интерпретации получаем, что

Далее как и в исходном алгоритме происходит вычисление функции потерь, но уже на новой матрице Грамма, и решается задача оптимизации

Где параметры функции служат пиксели изображения input.

По полученной точке минимума строится изображение, которое и будет соответствовать переносу стиля на фон либо на обьект изображения таким образом, что стиль полученной картинки наилучшим образом схож со стилевым изображением в смысле матрицы Грамма.

## 6. Анализ результатов исследования. Отображение результатов

Реализуем алгоритм с описанным набором гиперпараметров, который представляет наилучший его результат. Рассмотрим работу алгоритма с входными данными в виде следующих двух изображений.



Рисунок 6.1. Выходные контент-изображение и изображение – стиль.

Первым этапом алгоритм разделяет фон контент изображения от человека на этом фоне, тоесть производит семантическую сегментацию, получен результат.

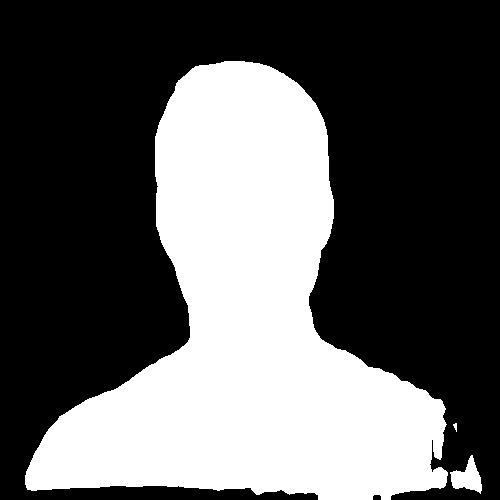


Рисунок 6.2. Сегментация мужчины

Далее следующим этапом происходит наложение на фон, который был распознан на изображении(черным цветом).



Рисунок 6.3. Выходное изображение алгоритма

На протяжении обучения модели, был получен график значений функции потерь от количества итераций, как по стилю так и по контенту, а так же время работы самого алгоритма.

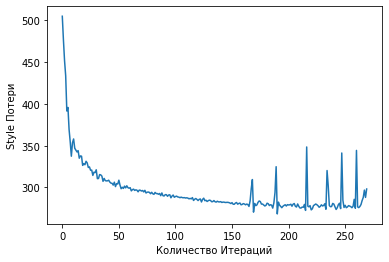


Рисунок 6.4. Функция потерь на протяжении обучения

Время работы алгоритма - 52.6 с.

Рассмотрим работу алгоритма с входными данными в виде следующих двух изображений.



Рисунок 6.5. Выходные контент-изображение и изображение – стиль.

Семантическая сегментация контент изображения представлена на рисунке

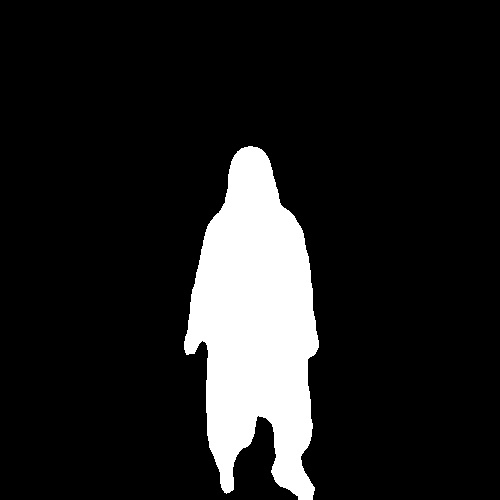


Рисунок 6.6. Сегментация изображения.

Наложение стиля на фон изображения.



Рисунок 6.7. Выходное изображение алгоритма

График функции потерь во время обучения.

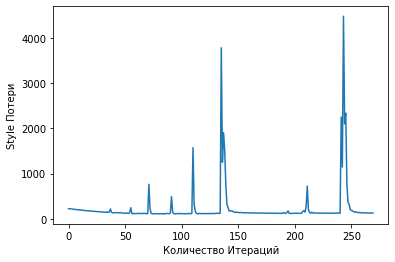


Рисунок 6.8. Функция потерь на протяжении обучения

Время работы алгоритма - 52.3 с.

Рассмотрим работу алгоритма с входными данными в виде следующих двух изображений.



Рисунок 6.9. Выходные контент-изображение и изображение – стиль.

Семантическая сегментация контент изображения представлена на рисунке.

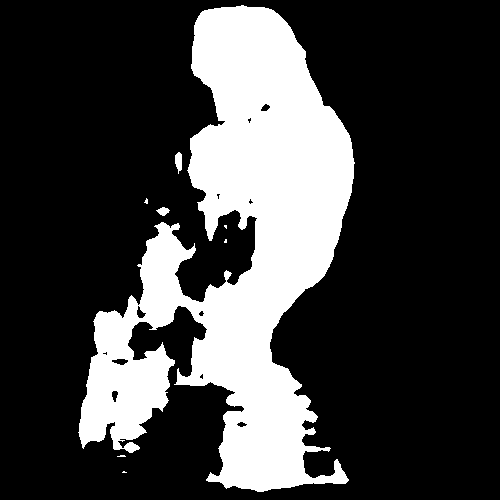
****

Рисунок 6.10. Сегментация изображения

****

Рисунок 6.11. Выходное изображение алгоритма

График функции потерь во время обучения.

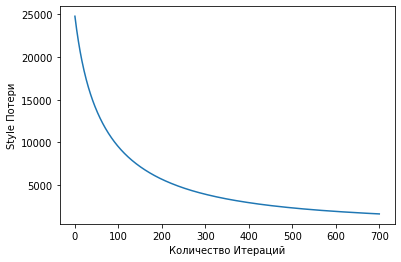


Рисунок 6.12. Функция потерь на протяжении обучения.

Время работы алгоритма - 52.4 с.

(Обучение происходило с помощью оптимизатора – AdamW, количество итераций = 700)

# 

# ВЫВОДЫ

В ходе данной работы определена постановка задачи нейросетевого переноса художественного стиля в изображениях. Описана методика решения задачи, а так же дана математическая интерпретация существенных для этой задачи понятий таких как стиль изображения, схожесть по стилю. Был предложен алгоритм решающий как задачу семантической сегментации изображения так и задачу переноса стиля. В качестве конечной конфигурации алгоритма были выбраны модели ResNet101,Vgg19, метод оптимизации – Алгоритма – Бройдена – Флетчера – Гольдфарба – Шанно, число итераций обучения –300.

Алгоритм предполагается применять в бесплатной облачной среде Google Colab, на языке Python 3.3 с использованием инструментов, которые предоставляют библиотеки Pytorch, Numpy, Pandas, PIL, torchvision. Так же были предложены различные совокупности параметров, которые целесообразно применять в зависимости от поставленной задачи перед пользователем описанного алгоритма.

Время работы итогового алгоритма по двум входным изображениям в среднем составляет 52 с.

# СПИСОК ЛИТЕРАТУРЫ

1. Курс продвинутого потока Deep learning School 1 часть.
2. Курс базового потока Deep learning School 1 часть.
3. Машинное обучение (курс лекций, К.В.Воронцов).
4. Course machine learning by Andrew Ng URL: https://www.coursera.org/learn/machine-learning.
5. Neural Style Transfer with TensorFlow URL: <https://towardsdatascience.com/introduction-to-neural-style-transfer-with-tensorflow-99915a5d624a>
6. Neural Network in Market Segmentation: A Descriptive study of Market Segmentation using Artificial Neural Networks Paperback – June 2, 2017
7. NEURAL NETWORKS A Comprehensive Foundation Second Edition. Simon Haykin
8. Соснин А.С., Суслова И.А. ФУНКЦИИ АКТИВАЦИИ НЕЙРОСЕТИ: СИГМОИДА, ЛИНЕЙНАЯ, СТУПЕНЧАТАЯ, RELU, TAHN. // Материалы XII международной научно-практической конференции Наука. Информатизация. Технологии. Образование Екатеринбург, 25 февраля–1 марта 2019 г.
9. A Neural Algorithm of Artistic Style URL: <https://arxiv.org/pdf/1508.06576.pdf>
10. Beyond BatchNorm: Towards a General Understanding of Normalization in Deep Learning URL: <https://arxiv.org/pdf/2106.05956v1.pdf>
11. «Искусственный интеллект: современный подход (AIMA-2). 2-е издание», Стюарт Рассел, Питер Норвиг.
12. «Python и машинное обучение. Машинное и глубокое обучение с использованием Python, Scikit-Learn и TensorFlow. 2-е издание», Себастьян Рашка, Вахид Мирджалил
13. СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ Романов А.А URL: <https://scientificresearch.ru/images/PDF/2018/21/svertochnye.pdf>
14. ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ И ИХ ПРИЛОЖЕНИЯ Ф.М. ГАФАРОВ, А.Ф. ГАЛИМЯНОВ URL: <https://kpfu.ru/staff_files/F1493580427/NejronGafGal.pdf>
15. Convolutional neural networks for classification Le Manh Ha URL: <https://mipt.ru/upload/medialibrary/659/91_97.pdf>
16. Сверточные нейронные сети. Глубокие остаточные сети Кустикова Валентина, к.т.н., ст.преп. каф. МОСТ ИИТММ, ННГУ им. Н.И. Лобачевского URL: <http://hpc-education.unn.ru/files/courses/intel-neon-course/Rus/Lectures/Presentations/4_CNN.pdf>
17. Semantic Segmentation in the era of Neural Networks Sergios Karagiannakos URL: <https://theaisummer.com/Semantic_Segmentation>
18. ImageNet Classification with Deep Convolutional Neural Networks Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton URL: <https://www.researchgate.net/publication/267960550_ImageNet_Classification_with_Deep_Convolutional_Neural_Networks>
19. Conditional Generative Adversarial Nets Mehdi Mirza, Simon Osindero URL: <https://arxiv.org/pdf/1411.1784.pdf>
20. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks by Jason Brownlee URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

# ПРИЛОЖЕНИЕ

# Листинг программы решающую задачу нейронного переноса стиля с первичной сегментацией.

## Листинг 1.1.

#Загружаем требуемые нам библиотеки

%matplotlib inline

from PIL import Image

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

import matplotlib.pyplot as plt

import torchvision.transforms as transforms

import torchvision.models as models

import copy

#Задаем размеры по высоте и ширине для наших картинок и приводим к ним их

imsize = 500

loader = transforms.Compose([

transforms.Resize(imsize), # нормируем размер изображения

transforms.CenterCrop(imsize),

transforms.ToTensor()]) # превращаем в удобный формат

#Функция для загрузки изображений

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

def image\_loader(image\_name):

image = Image.open(image\_name)

image = loader(image).unsqueeze(0)

return image.to(device, torch.float)

#Загружаем 2 стиля и контент

style\_img = image\_loader("Style.jpg")

content\_img = image\_loader("Man.jpg")

#Смотрим на загруженные нами картинки

unloader = transforms.ToPILImage() # тензор в кратинку

plt.ion()

def imshow(tensor, title=None):

image = tensor.cpu().clone()

image = image.squeeze(0) # функция для отрисовки изображения

image = unloader(image)

plt.imshow(image)

if title is not None:

plt.title(title)

plt.pause(0.001)

# отрисовка

plt.figure()

imshow(style\_img, title='Style Image')

plt.figure()

imshow(content\_img, title='Content Image')

## Листинг 1.2.

#ЧАСТЬ С СЕГМЕНТАЦИЕЙ

from torchvision import models

import numpy as np

model = models.segmentation.fcn\_resnet101(pretrained=True).to('cuda')

y\_pred = model(content\_img)['out']

om=torch.argmax(y\_pred.squeeze(),dim=0).detach().to("cpu")

om[om>0]=1

unloader(om.type(torch.FloatTensor))

new = unloader(om.type(torch.FloatTensor))

new = new.resize((800,800))

om = transforms.ToTensor()(new)

## Листинг 1.3.

#Задаем класс новых словев в нашей нейросети, которые высчитывают лосс по контенту

from PIL import Image

import time

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

import matplotlib.pyplot as plt

import torchvision.transforms as transforms

import torchvision.models as models

import copy

class ContentLoss(nn.Module):

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

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

self.target = target.detach()

self.loss = F.mse\_loss(self.target, self.target )

def forward(self, input):

self.loss = F.mse\_loss(input, self.target)

return input

def gram\_matrix(input):

batch\_size , h, w, map\_num = input.size()

features = input.view(batch\_size \* h, w \* map\_num)

G = torch.mm(features, features.t())

return G.div(batch\_size \* h \* w \* map\_num)

#Задаем класс новых словев в нашей нейросети, которе высчитывают лосс по стилю

class StyleLoss(nn.Module):

def \_\_init\_\_(self, target\_feature, mask):

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

self.target = gram\_matrix(target\_feature).detach()

self.loss = F.mse\_loss(self.target, self.target)

self.mask=mask

def forward(self, input):

#Будем высчитывать Loss только по нашей маске

\_, \_, h, w = input.size()

mask\_2d = unloader(self.mask.type(torch.FloatTensor))

mask\_2d = mask\_2d.resize((h,w))

mask\_2d = transforms.ToTensor()(mask\_2d)

mask = torch.zeros\_like(input)

mask[:, :] = mask\_2d.clone()

mask = (1-mask).abs()

G = gram\_matrix(input\*mask)

self.loss = F.mse\_loss(G, self.target)

return input

cnn\_normalization\_mean = torch.tensor([0.485, 0.456, 0.406]).to(device)

cnn\_normalization\_std = torch.tensor([0.229, 0.224, 0.225]).to(device)

class Normalization(nn.Module):

def \_\_init\_\_(self, mean, std):

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

self.mean = torch.tensor(mean).view(-1, 1, 1)

self.std = torch.tensor(std).view(-1, 1, 1)

def forward(self, img):

return (img - self.mean) / self.std

content\_layers\_default = ['conv\_4']

style\_layers\_default = ['conv\_1', 'conv\_2', 'conv\_3', 'conv\_4', 'conv\_5']

cnn = models.vgg19(pretrained=True).features.to(device).eval()

def get\_style\_model\_and\_losses(cnn, mask, normalization\_mean, normalization\_std,

style\_img, content\_img,

content\_layers=content\_layers\_default,

style\_layers=style\_layers\_default):

cnn = copy.deepcopy(cnn)

normalization = Normalization(normalization\_mean, normalization\_std).to(device)

content\_losses = []

style\_losses\_1 = []

model = nn.Sequential(normalization)

i = 0 # i увеличиваем когда видим сверточный

for layer in cnn.children():

if isinstance(layer, nn.Conv2d):

i += 1

name = 'conv\_{}'.format(i)

elif isinstance(layer, nn.ReLU):

name = 'relu\_{}'.format(i)

layer = nn.ReLU(inplace=False)

elif isinstance(layer, nn.MaxPool2d):

name = 'pool\_{}'.format(i)

elif isinstance(layer, nn.BatchNorm2d):

name = 'bn\_{}'.format(i)

else:

raise RuntimeError('Unrecognized layer: {}'.format(layer.\_\_class\_\_.\_\_name\_\_))

model.add\_module(name, layer)

if name in content\_layers:

# добавляем по контенту потерю

target = model(content\_img).detach()

content\_loss = ContentLoss(target)

#Добавление к нашей строющейся сети нового слоя потерь

model.add\_module("content\_loss\_{}".format(i), content\_loss)

content\_losses.append(content\_loss)

if name in style\_layers:

# add style loss:

target\_feature = model(style\_img).detach()

style\_loss\_1 = StyleLoss(target\_feature,mask)

#Добавление к нашей строющейся сети нового слоя потерь

model.add\_module("style\_loss\_1{}".format(i), style\_loss\_1)

style\_losses\_1.append(style\_loss\_1)

for i in range(len(model) - 1, -1, -1):

if isinstance(model[i], ContentLoss) or isinstance(model[i], StyleLoss):

break

model = model[:(i + 1)]

return model, style\_losses\_1, content\_losses

def get\_input\_optimizer(input\_img):

optimizer = optim.LBFGS([input\_img.requires\_grad\_()])

return optimizer

losses\_st = []

losses\_cont = []

params=[]

def run\_style\_transfer(cnn, segment,normalization\_mean, normalization\_std,

content\_img, style\_img, input\_img, num\_steps=2000,

style\_1\_weight=1000000, content\_weight=100):

#Задание маски для нашего алгоритма

y\_pred = segment(content\_img)['out']

om=torch.argmax(y\_pred.squeeze(),dim=0).detach().to("cpu")

om[om>0]=1

mask = om

model, style\_losses\_1, content\_losses = get\_style\_model\_and\_losses(cnn,mask,

normalization\_mean, normalization\_std, style\_img, content\_img)

optimizer = get\_input\_optimizer(input\_img)

epoches = [0]

while epoches[0] <= num\_steps:

def closure():

input\_img.data.clamp\_(0, 1)

optimizer.zero\_grad()

model(input\_img)

style\_score\_1 = 0

style\_score\_2 = 0

content\_score = 0

for sl1 in style\_losses\_1:

style\_score\_1 += sl1.loss

for cl in content\_losses:

content\_score += cl.loss

#взвешивание ошибки

style\_score\_1 \*= style\_1\_weight

content\_score \*= content\_weight

#сохранение ошибок для графика и получение конечной ошибки

losses\_st.append(style\_score\_1.item())

losses\_cont.append(content\_score.item())

params.append(input\_img.data.clamp\_(0, 1))

loss = style\_score\_1 + content\_score

#Высчитывание градиентов

loss.backward()

epoches[0] +=1

if epoches[0] % 50 == 0:

print('number of epoches {}'.format(epoches[0]))

print('Style Loss : {:4f} Content Loss: {:4f}'.format(

style\_score\_1.item(), content\_score.item()))

return style\_score\_1 + content\_score

optimizer.step(closure)

input\_img.data.clamp\_(0, 1)

return input\_img

input\_img = content\_img.clone()

plt.figure()

num\_steps = 400

output = run\_style\_transfer(cnn, model,cnn\_normalization\_mean, cnn\_normalization\_std,

content\_img, style\_img, input\_img, num\_steps = num\_steps)

#Получим индекс минимального элемента в массиву losses\_style

min\_st=losses\_st[0]

ind\_min=0

for ind,el in enumerate(losses\_st):

if (el<min\_st):

min\_st=el

ind\_min=ind

output=params[3].data.clamp\_(0, 1)

#Фотография с наименьшей потерей по стилю

plt.figure()

imshow(output)

plt.ioff()

plt.show()

fig = plt.figure()

ax\_1= fig.add\_subplot(1,1,1)

ax\_1.plot([i for i in range(0, len(losses\_st[50:]))], losses\_st[50:])

ax\_1.set(xlabel="Количество Итераций", ylabel="Style Потери")