МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Гусева Евгений Алексеевич
(фамилия, имя, отчество)

Институт (факультет) ИРИТ
Кафедра Информатики и систем управления
Группа 16 СБК

Дата защиты
«____»_____

Индекс

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт радиоэлектроники и информационных технологий
Направление подготовки (специальность) 09.03.02 Информационные системы и технологии
Направленность (профиль) образовательной программы Безопасность информационных систем
Кафедра Информатики и систем управления

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра

Студента Гусева Евгения Алексеевича группы 16 СБК на тему:

«Стилистическая обработка изображения. Часть1. Генеративная нейронная сеть.»

СТУДЕНТ:
_____Гусев Е.А.
(подпись) (фамилия, и., о.)

_____
(дата)

РУКОВОДИТЕЛЬ:
_____Тюрин А.И.
(подпись) (фамилия, и., о.)

_____
(дата)

РЕЦЕНЗЕНТ:

_____
(подпись) (фамилия, и., о.)

_____
(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ:
_____Соколова Э. С.
(подпись) (фамилия, и., о.)

_____
(дата)

КОНСУЛЬТАНТЫ:
1. По ____нормоконтролю_____
_____Шагалова П.А.
(подпись) (фамилия, и., о.)

_____
(дата)

2. По _____

_____
(подпись) (фамилия, и., о.)

_____
(дата)

3. По _____

_____
(подпись) (фамилия, и., о.)

_____
(дата)

ВКР защищена _____
(дата)

протокол № _____
с оценкой _____

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Кафедра Информатики и систем управления

УТВЕРЖДАЮ

Зав. кафедрой

_____ Э.С. Соколова

«___» _____ 2020 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

по направлению подготовки (специальности)09.03.02 Информационные системы и технологии студенту Гусеву А. Е. группы 16 СБК

1. Тема «Стилистическая обработка изображения. Часть1. Генеративная нейронная сеть.»

(утверждена приказом по вузу от № 868/5 от 15.04.2020)

2. Срок сдачи студентом законченной работы _____30.06.20_____

3. Исходные данные к работе:

4. Содержание расчетно-пояснительной записки (перечень вопросов, подлежащих разработке)

Обзор предметной области

Алгоритмы нефотореалистичного рендеринга для задачи стилистической обработки изображений

Алгоритмы нейронного переноса стиля

Реализация алгоритма стилистической обработки изображений

Оценка реализованной системы

5. Перечень графического материала (с точным указанием обязательных чертежей)

- Цели работы;
- Процесс развития проекта;
- Обзор предметной области;
- Нейронный перенос стиля;
- Реализация системы;
- Тестирование качества обработки;
- Тест скорости работы алгоритма;
- Выводы.

6. Консультанты по ВКР (с указанием относящихся к ним разделов)

Нормоконтроль Шагалова П.А.

7. Дата выдачи задания 02.03.2020

Код и содержание Компетенции	**Задание**	**Проектируемый результат**	**Отметка о выполнении**
ПК-23: готовность участвовать в постановке и проведении экспериментальных исследований	Изучить методы стилистической обработки изображений	Разработать модель решающую задачу стилизации изображений	Выполнено
ПК-24: способность обосновывать правильность выбранной модели, сопоставляя результаты экспериментальных данных и полученных решений	На основе разработанной модели реализовать систему и провести ряд экспериментов для сбора статистки	Изменять параметры модели до того, как реализованная система будет выдавать оптимальный результат	Выполнено
ПК-25 - способность использовать математические методы обработки, анализа и синтеза результатов профессиональных исследований ПК-30 – способность поддерживать работоспособность информационных систем и технологий в заданных функциональных характеристиках и соответствии критериям качества	Провести исследование алгоритмов стилистической обработки изображений	Теоретические выводы, подкрепленные графиками и таблицами	Выполнено
ПК-32 – способность адаптировать приложения к изменяющимися условиями функционирования	Реализовать систему с возможностью ее работы на разных платформах. А также с возможностью ее расширения	Система способна работать на платформах Windows и Linux, а также есть возможность добавления новых стилей	Выполнено

ПК-26: способность оформлять полученные рабочие результаты в виде презентаций, научно-технических отчетов, статей и докладов на научно-технических конференциях ПК-31 – способность обеспечивать безопасность и целостность данных информационных систем и технологий	Оформить графические материалы и пояснительную записку к ВКР	Графические материалы и пояснительная записка к ВКР	Выполнено
ПК-33 – способность составлять инструкции по эксплуатации информационных систем	Составить инструкцию пользователя для работы с созданной программой	Написание полной и подробной инструкции для пользователя	Выполнено
ОПК-5 – способность использовать современные компьютерные технологии поиска информации для решения поставленной задачи, критического анализа этой информации и обоснования принятых идей и подходов к решению	Использовать компьютерные технологии для ознакомления и прочтения множества статей по схожим темам	Использование электронной базы arxiv.org для изучения научных статей	Выполнено

Руководитель _____ Тюрин А. И.
(подпись)

Задание принял к исполнению_____
(дата)

Студент _____Гусев Е. А.
(подпись)

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

АННОТАЦИЯ

к выпускной квалификационной работе

по направлению подготовки (специальности)09.03.02 Информационные системы и технологии студента Гусева Е.А. группы 16 СБК

по теме «Стилистическая обработка изображения. Часть1. Генеративная нейронная сеть.»

Выпускная квалификационная работа выполнена на ____ страницах, содержит ____ диаграмм, _____ таблиц, библиографический список из _____источников, _____приложений.

Актуальность: Актуальность данной работы обусловлена потребностью людей в стилистической обработке изображений. Тому подтверждение популярность такого графического редакторов как Adobe Photoshop или веб-сервисов как Ostagram. Но процесс использования графических редакторов для стилизации изображений слишком трудоемкий и занимает много времени, а большинство сервисов используют устаревшие алгоритмы, которые обладает большим количеством ограничений и предоставляют неудобный интерфейс. Данная работа является решением, которое поможет сделать процесс стилистической обработки изображений быстрым, простым и удобным.

Объект исследования: Процесс стилистической обработки изображений.

Предмет исследования: Алгоритмы стилизации изображений.

Цель исследования: Разработка системы для стилистической обработки изображений.

Задачи исследования: Анализ алгоритмов переноса стиля, анализ методов для улучшения качества стилизованного изображения, поиск способов для оптимизации скорости работы алгоритма.

Структура работы:

Введение рассказывается о цели работы и ее актуальности

В 1 разделе «ПО для стилистической обработки изображений» рассматриваются существующие сервисы и программы для стилизации изображений

Во 2 разделе «Обзор предметной области» описывается задача стилистической обработки изображений и рассматриваются алгоритмы для ее решения

В 3 разделе «Проектирование системы» рассматриваются инструменты разработки и архитектура системы

В 4 разделе «Реализация» описывается и тестируется реализованная система

Выводы:

Рекомендации:

_____/_____ «____» _____ 20_____ г.

подпись студента /расшифровка подписи

«___»_____ 20__ г.

Содержание

Введение

Искусство является важной сферой человеческой жизни. Работы таких великих художников как Пабло Пикассо и Микеланджело вдохновляли многих людей. В некоторых из них закрадывалось желание научится рисовать также хорошо и создавать подобные шедевры, но лишь у немногих хватало терпения достичь этой цели. Написание действительно хорошей картины, претендующей на звание произведения искусства, занимает большое количество времени. Многие художники посвящают годы жизни на создание чего-то стоящего. У каждого из них вырабатывается свой уникальный стиль, благодаря которому картины автора узнаваемы по всему миру. Например, картины Винсента Ван-Гога или Клода Моне.

Но в век компьютерных технологий человечество не готово ждать столь долго. Опираясь на стремление к созданию картин, схожих с работами художников, перед энтузиастами встала задача стилистической обработки изображений. С развитием технологий компьютерного зрения и компьютерной графики решение задачи стилистической обработки изображений стало осуществимым. На сегодняшний день разработаны алгоритмы и методы, позволяющие создавать изображения, которые порой сложно отличить от реальных работ художников. Существующие решения значительно упрощают и автоматизируют процесс стилизации изображений. Благодаря этому у людей появилась уникальная возможность сделать свои фотографии более красивыми, не прилагая особых усилий, и при этом сэкономив большое количество времени.

Технология стилизации изображений не могла не заинтересовать человека, интересующегося цифровым искусством. Поэтому появилась необходимость в разработке программного обеспечения и сервисов для предоставления возможности широкому кругу пользователей обрабатывать изображения в определенном стиле и получать результат, который бы их удовлетворил.

Актуальность данной работы обусловлена потребностью людей в стилистической обработке изображений. Тому подтверждение популярность такого графического редакторов как Adobe Photoshop или веб-сервисов как Ostagram. Но процесс использования графических редакторов для стилизации изображений слишком трудоемкий и занимает много времени, а большинство сервисов используют устаревшие алгоритмы, которые обладает большим количеством ограничений и предоставляют неудобный интерфейс. Данная работа является решением, которое поможет сделать процесс стилистической обработки изображений быстрым, простым и удобным.

1 ПО для стилистической обработки изображений

Для стилистической обработки изображений существует не так много программ и сервисов, решающих поставленную задачу. Рассмотрим известные решения:

1.1 Adobe Photoshop

Компания Adobe является лидером на рынке предоставления услуг по обработке изображений. Их основным продуктом является графический редактор Adobe Photoshop. Данный редактор является многофункциональным инструментом, который обрел большую популярность как в кругу обычных пользователей, которые используют его для любительской обработки, так и среди профессиональных дизайнеров.

В рамках решения задачи стилистической обработки изображений исследовательским отделом Adobe Research был разработан метод абстракции [1], который позволяет на основе низкоуровневого размытия и изменении резкости изображения создавать чрезвычайно привлекательные мультипликационные эффекты. На основе данного метода был разработан модуль стилизации для графического редактор Adobe Photoshop.

а) б)

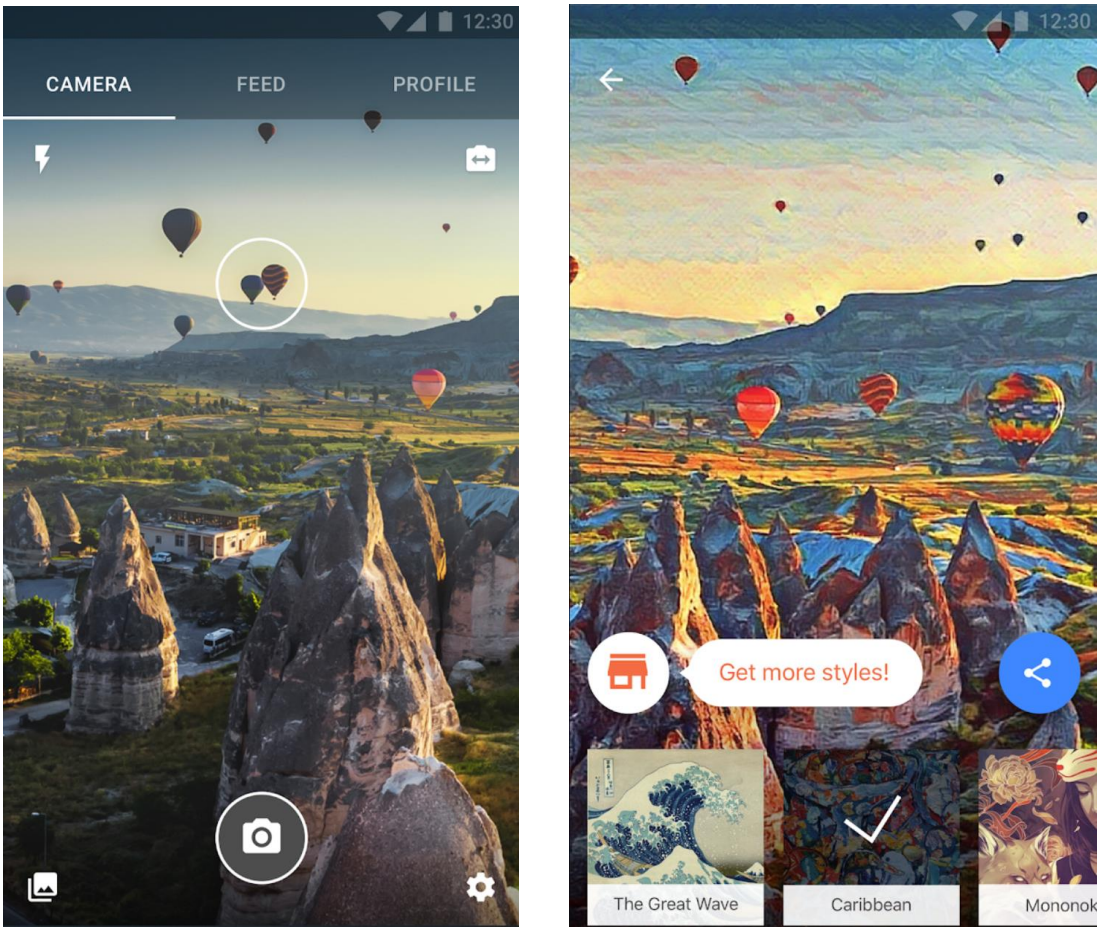
Рисунок 1 – Пример использования метода абстракции.
а) Исходное изображение; б) Стилизованное изображение.

Модуль стилизации графического редактора Adobe Photoshop предоставляет хорошие возможности для стилизации изображения, но для их использования необходимо обладать навыками работы с редактором. Данный графический редактор обладает

огромным количеством других инструментов для обработки изображений и отлично подходит для решения таких задач как: ретушь, цветокоррекция и т.д., но неудобен для быстрой стилизации изображений. Также продукт предоставляется за слишком высокую плату – 1300 рублей в месяц.

1.2 Prisma

Приложение Prisma разработано в 2016 году компанией Prisma Labs и предназначено для стилизации изображений. В качестве ядра приложения лежит алгоритм, основанный на использовании нейросетевых технологий. Главное преимущество данного приложения заключается в простоте его использования, но в то же время большинство стилей для обработки предоставляются за дополнительную плату. Например, чтобы получить доступ ко всем предложенным стилям необходимо заплатить порядка 7000 рублей. Также недостатком данного приложения является то, что оно доступно только на мобильных устройствах.



а) б)

Рисунок 2 – Интерфейс приложения Prisma.

а) Исходное изображение; б) Стилизованное изображение.

1.3 Ostagram

Веб-сервис Ostagram предназначен для стилистической обработки изображений и разработан студентами НГТУ им. Р.Е.Алексеева в конце 2015 года. Особую популярность данный сервис начал набирать в апреле 2016 года. Изначально Ostagram предоставлял бесплатную услугу по стилистической обработке изображений, но в связи с большим спросом сервис был монетизирован и сейчас в бесплатном режиме имеется большое количество ограничений.

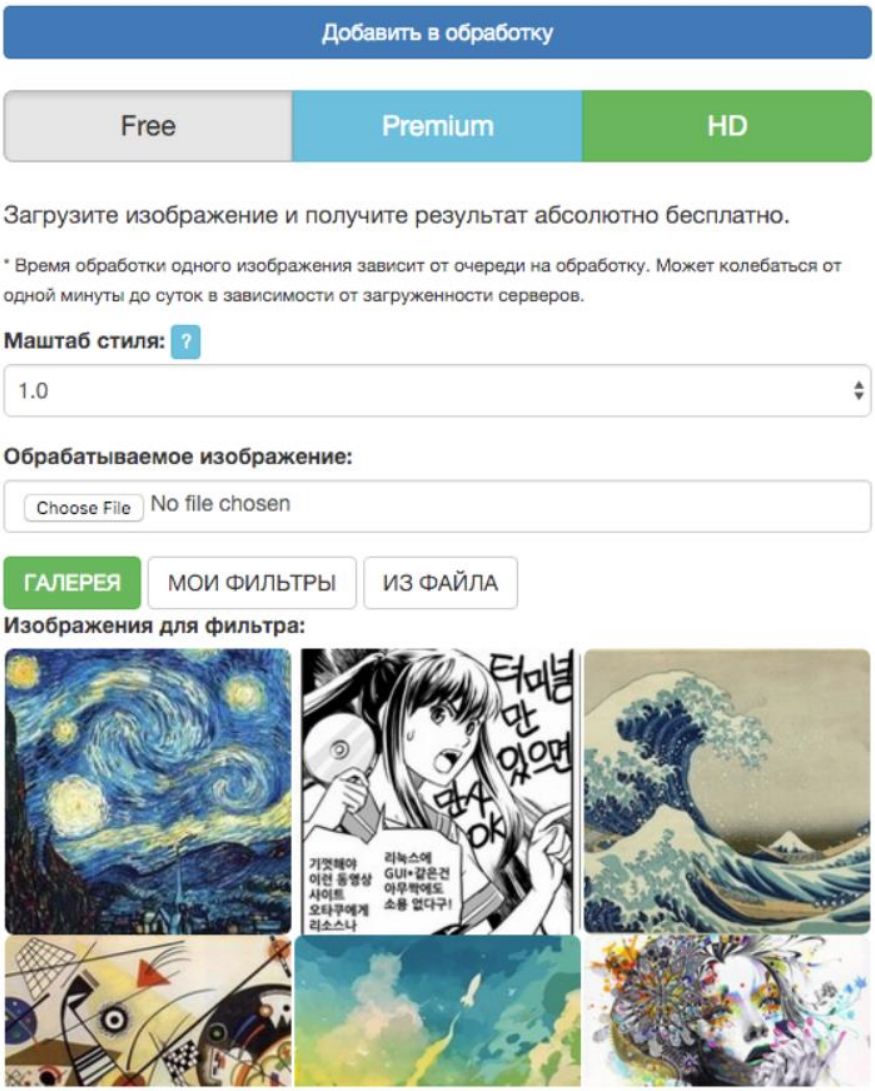


Рисунок 3 – Интерфейс веб-сервиса Ostagram.

Главным недостатком веб-сервиса Ostagram является алгоритм, который был выбран для стилистической обработки изображений. Используемый алгоритм, как и в приложении Prisma, основан на использовании нейросетвых технологий, но его эффективность по времени работы и по используемым ресурсам ограничивается итеративным процессом стилизации изображений. Из-за этого процесс обработки может

занимать от 5 минут до нескольких часов. Еще одним недостатком данного сервиса является ограничение на разрешение для обработанного изображения. В бесплатной версии в результате обработки пользователь получит изображение с разрешением не большее 600 пикселей по длинной стороне, в платной версии максимальное разрешение достигает 1200 пикселей.

Рисунок 4 – Пример стилистической обработки с использованием сервиса Ostagram.

1.4 Задачи для решения

Существующие решения предоставляют отличные возможности для стилизации изображений, но сегодня наблюдается тенденция упрощения доступа к услугам. Большинство компаний стараются объединить все предоставляемые услуги в одном продукте, чтобы у пользователей не было необходимости для каждой отдельной задачи скачивать большое количество приложений. Пользователи не готовы ждать по несколько часов, чтобы получить обработанную фотографию, им хочется получить результат здесь и сейчас. Поэтому отличным решением для представления услуг стилистической обработки изображений будет разработка системы, которая предоставляет простой и удобный интерфейс и осуществляет процесс обработки изображений быстрее чем большинство разработанных программных продуктов.

В результате анализа существующих решений были поставлены следующие задачи:

– Разработка системы для стилистической обработки изображений;
– Выбор и реализация алгоритма для стилистической обработки изображений;
– Разработка пользовательского интерфейса для предоставления услуг стилизации изображений;

Система должна удовлетворять следующим требованиям:

– Разработанный пользовательский интерфейс должен быть доступным простым и удобным;
– Система должна предоставлять возможность обработки изображений с максимальным разрешением 3840x2160;
– Выбранный алгоритм должен обрабатывать изображение в разрешении 3840x2160 меньше чем за 1 минуту.

Так как задача достаточно объемная было принято решение разделить ее на 2 части:

– алгоритм стилистической обработки изображений;
– пользовательский интерфейс.

Перед студентом Гусевым Е.А. была поставлена задача выбора и реализация алгоритма обработки.

Перед студентом Назаренко А.Р. была поставлена задача разработки пользовательского интерфейса.

2 Обзор предметной области

На сегодняшний день разработаны различные методы и алгоритмы для решения задачи стилистической обработки изображений. С их помощью осуществляется перенос стиля с одного изображения на другое. Прежде чем перейти к рассмотрению существующих решений, необходимо рассмотреть такие понятия как: *стиль, перенос стиля, изображение контента и изображение стиля.*

Стиль можно рассматривать с двух позиций. Первая из них связана с процессом написания картин. Каждый художник обладает своим уникальным стилем, который формируется из того, как художник орудует кистью. Каждая картина создается на основании нанесения на полотно мазков кистью. Один художник используют густые мазки, другие более жидкие. Кто-то пишет картины импульсивно, используя размашистые резкие мазки, иные художники предпочитают создавать свои работы в спокойном темпе, нанося точные выверенные мазки кистью. Также стоит учитывать, что при написании картин используются различные техники, такие как: масло, гуашь, пастель, эмаль и др. Все выше сказанное вместе формирует уникальный стиль автора. Второй подход для рассмотрения, что такое стиль, является более формальным и основывается на представление стиля как совокупности цветового наполнения и текстур изображения. Текстуру характеризует форма линий на изображении. Например, текстура дерева состоит из узора, который сформирован из радиальных и тангенциальных разрезов Рисунок 5.

Рисунок 5 – Пример текстуры дерева.

Цветное наполнение изображения характеризуется спектром представленных цветов и степенью их содержания. На Рисунке 6 представлены все цвета радуги и не только, но преобладают желтый и синий.

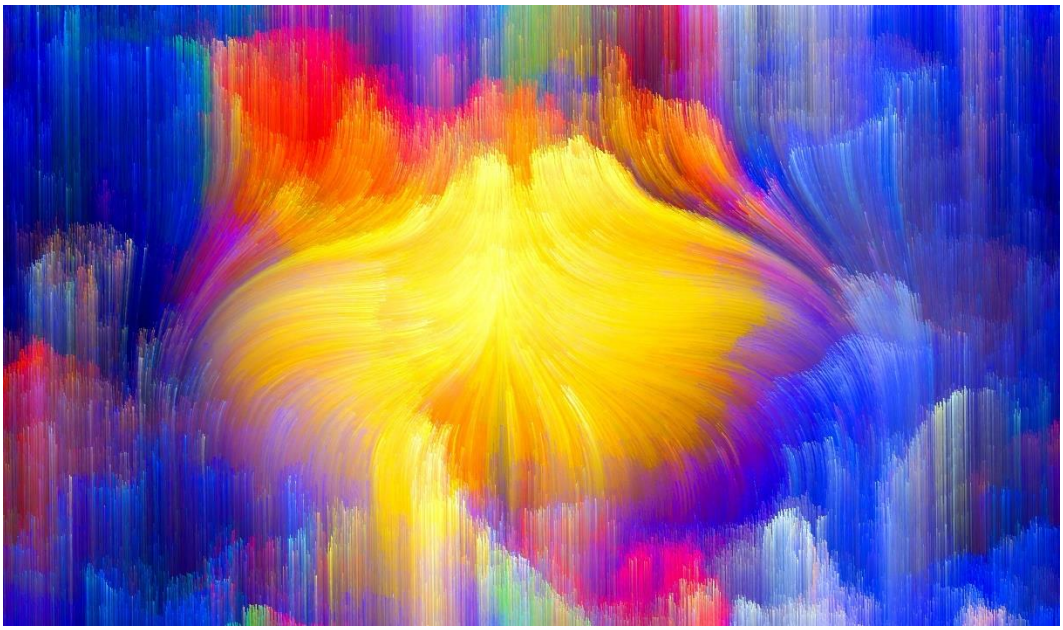


Рисунок 6 – Изображение с ярким цветовым наполнением.

Задача стилистической обработки изображения заключается в том, чтобы перенести цветовое наполнение и текстуру одного изображения на другое. В результате переноса стиля должно получиться стилистически обработанное изображение в том стиле, который использовался. В качестве изображения стиля как правило используют картины художников.

а)

б)

Рисунок 7 – Пример стилистической обработки изображений.

а) Картина Франсиса Пикабиа “Udnie”; б) Фотография, обработанная в стиле картины.

Изображение контента представляет из себя исходное изображение, предназначенное для стилистической обработки. Под контентом понимаются расположенные на изображении объекты, их форма и расположение в пространстве. Рассмотрим понятие контента на примере Рисунка 8. В центре изображения находится озеро, которое имеет определенную форму. На переднем плане ближе к левому краю изображения располагаются два дерева. Справа от озера есть лес. На заднем плане видны горы. Решая задачу переноса стиля, мы должны сохранить как объекты на изображение, так и их взаимное расположение в пространстве. В результате стилистической обработки изображения необходимо получить композицию близкую к исходной, но с цветовым содержанием и текстурой выбранного стиля.

а)

б)

Рисунок 8 – Пример обработки изображения контента.

а) Исходное изображение контента; Обработанное изображение.

Для решения задачи стилистической обработки изображений используются два принципиально разных подхода. Один основан на использовании инструментов нефтореалистичного рендеринга, другой на использовании нейросетвых технологий. Рассмотрим подробнее каждый из них.

2.1 Нефотореалистчный рендринг

Нефотореалистичный рендеринг относится к области компьютерной графики. Под рендерингом понимается процесс отрисовки модели, которая описывает объекты и явления. Изначально нефотореалистичный рендеринг был вдохновлен работами художников, анимационными фильмами и видеоиграми. На основе инструментов нефотореалистичного рендеринга были предложены первые решения задачи стилистической обработки изображений. Разработанные инструменты активно используются в киноиндустрии, видеоиграх и мультфильмах. Ярким примером применения данной технологии является художественный фильм “Куда приводят мечты”.

Рисунок 9 – Кадр из художественного фильма “Куда приводят мечты”.

Основными методами нефотреалистичного рендеринга для стилизации являются:

- Рендеринг на основе штрихов;
- Рендеринг на основе регионов;
- Рендеринг на основе примеров.

2.1.1 Рендеринг на основе штрихов

Рендеринг на основе штрихов является одним из первых алгоритмов в истории компьютерной графики решающий задачу стилистической обработки изображений. Идея алгоритма заключатся в том, чтобы последовательно наносить на изображение цифровые штрихи, которые имитируют мазки кистью. Первым, кто предложил эту идею, был разработчик компании Silicon Graphics Пол Хеберли. В 1990 году он опубликовал в газете SIGGRAPH статью [2], в которой предложил с помощью мышки наносить на изображение штрихи, чтобы из обычной фотографии получить изображение, похожее на картину художника.

Рисунок 10 – Пример стилизации изображения путем нанесения цифровых штрихов

Но для получения стилизованного изображения необходимо использовать программное обеспечение и самостоятельно с помощью мышки обрабатывать изображение. В 1997 году на конференции SIGGRAPH был представлен алгоритм рендеринга на основе нанесения штрихов [3], который был посвящен автоматизации процесса стилизации изображения цифровыми штрихами.

а) б)

Рисунок 11 – Пример рендеринга на основе штрихов.
а) Исходное изображение; б) Обработанное изображение.

Рендеринг на основе штрихов позволяет превратить изображение в картину, выполненную определенной технике, но решить задачу переноса стиля с его помощью не представляется возможным, так как этот подход не позволяет кардинально менять цветовое наполнение изображения.

2.1.2 Рендеринг на основе регионов

Рендеринг на основе регионов решает задачу стилистической обработки изображений следующим образом: изображение контента подвергается сегментации и после перерисовывается в необходимом стиле. Процесс сегментации заключается в выделении регионов, которые представляют из себя области пикселей, обладающих схожими свойствами.

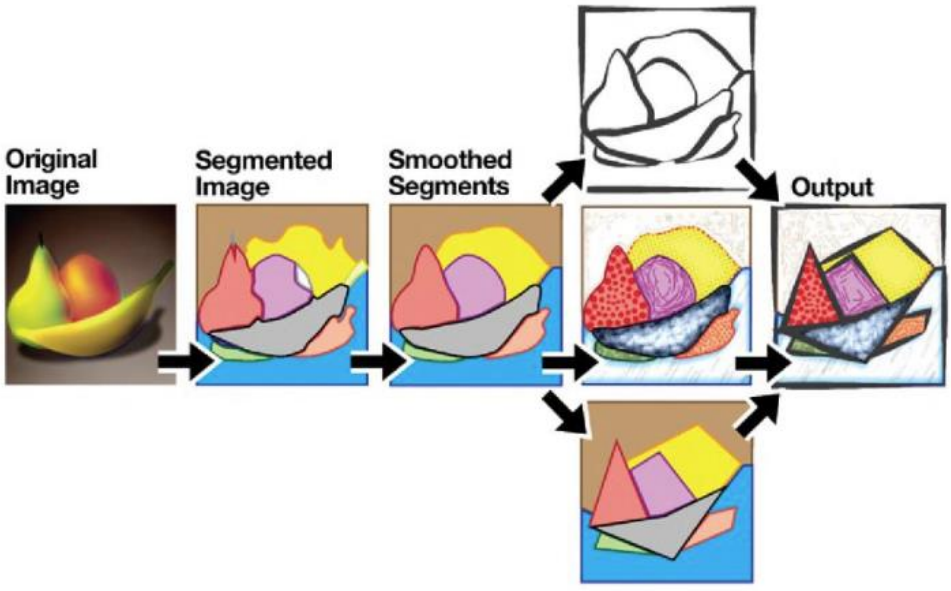


Рисунок 12 – Процесс рендеринга изображения на основе регионов.

Проблема данного метода заключается в том, что для каждого стиля необходимо построить модель и реализовать ее в виде программного модуля. Это задача является очень трудоемкой, так как для отдельного стиля придется писать и оптимизировать отдельную программу. И при всем этом разработанные модули стилизации как правило встраиваются в такие программные продукты как Adobe Photoshop и Adobe Illustrator, для использования которых необходимо обладать навыком работы с ними.

2.1.3 Рендеринг на основе примеров

Рендеринг на основе примеров решает задачу стилистической обработки изображений с использованием технологий машинного обучения. Одним из первых алгоритмов для стилизации изображений на основе примеров является алгоритм аналогий [4]. Его идея заключается в том, чтобы взять пару изображений: изображение контента A и изображение A’, нарисованное художником на основе изображения контента. На основе этой обучающей пары алгоритм аналогий должен изучить преобразование из изображения A в изображение A’. Пример обучающей пары представлен на Рисунке 13.

а) б)

Рисунок 13 – Пример обучающей пары.

а) Изображение контента A; б) Изображение, нарисованное художником A’.

Используя обученный алгоритм, можно воссоздавать художественный стиль для любых изображений.

Рисунок 14 – Пример работы алгоритма аналогий.
а) Исходное изображение; б) Стилизованное изображение.

Только каждый раз искать художника, чтобы он нарисовал на основе изображения контента его стилизованную версию, слишком утомительно. И стоит учитывать, что в наши дни Ван-Гога уже не найти, поэтому подавляющее большинство стилей художников не получится воспроизвести. Из-за этого данный алгоритм слишком ограничен по стилям по причине сложности создания обучающих пар.

2.2 Нейронный перенос стиля

За последнее десятилетие особую популярность обрели нейронные сети, которые позволяют решать задачи классификации, прогнозирования, кластеризации и др. Нейросетевые технологии рассматривались во второй половине 20-го века, но лишь в 2012 году, после победы сверточной нейронной сети Алекса Крижевски [5] на конкурсе по классификации изображений ImageNet, эта технология привлекла всеобщее внимание. После этого были разработаны нейронные сети для увеличения разрешения изображений, наведения резкости на изображении, выделения объектов и в том числе для стилистической обработки изображений.

Первая научная работа по решению задачи переноса стиля с использованием нейронных сетей [6] была написана в 2015 году. Авторы статьи Леон Гатис и Александр Экель предложили для переноса стиля использовать алгоритм, идея которого заключается в том, чтобы использовать сверточную нейронную сеть для построения функции потерь, которая будет характеризовать степень отклонения стилизованного изображения от изображений контента и стиля. Прежде чем перейти к подробному рассмотрению алгоритма, предложенного Гатисом, необходимо рассмотреть принцип построения функции потерь для решения задачи стилистической обработки изображения.

2.2.1 Функция потерь

Сверточная нейронная сеть работает по принципу перехода от конкретных особенностей изображений таких как линии, фигуры и т.д., к более абстрактным деталям, вплоть до понятий высокого уровня. Опираясь на таком представлении изображения сетью, можно построить функцию потерь для решения задачи стилистической обработки изображений. Чтобы посчитать значение функции потерь, необходимо для стилизованного изображения посчитать потерю по стилю и потерю по контенту. Исходя из этого функцию потерь необходимо разделить на две части: функцию потерь контента и функцию потерь стиля. Основываясь на этом, функция потерь для задачи стилистической обработки изображений примет следующий вид:

$$L_{\text{общая}}(S, C, G) = \alpha * L_{\text{контента}}(C, G) + \beta * L_{\text{стиля}}(S, G) \quad (1)$$

В формуле (1), чтобы вычислить $L_{общая}$, необходимо рассчитать потерю контента $L_{контента}$, и потерю стиля $L_{стиля}$. Параметры α и β представляют собой веса каждого из типа потерь, что позволяет контролировать степень переноса стиля. Для вычисления функции потерь используется 3 изображения: стилизованное изображение G, изображение контента C, изображение стиля S. Значения функций потерь стиля и контента рассчитываются на основе значений функций активации нейронов для выбранных сверточных слоев.

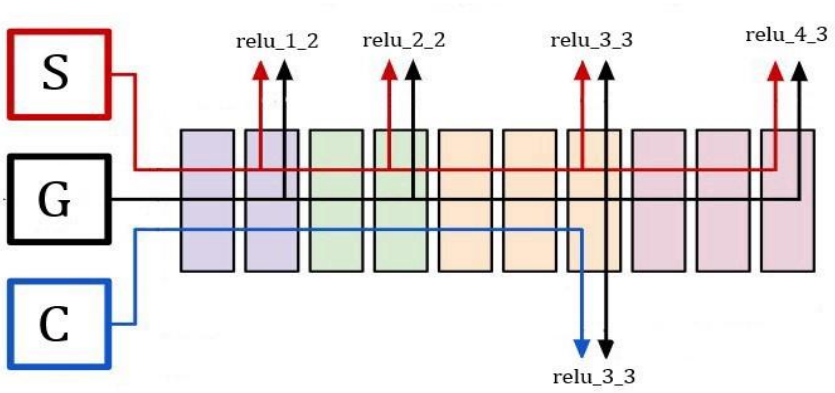


Рисунок 15 – Принцип расчета функции потерь для задачи переноса стиля.

Для того, чтобы рассчитать значение функции потерь контента, достаточно сформировать матрицы значений функций активации нейронов сверточного слоя для стилизованного изображения и изображения контента. И посчитать разницу между полученными матрицами.

$$L_{\text{контент}}(C, G, l) = \frac{1}{2} * \sum_{ij} \left(a[l](C)_{ij} - a[l](G)_{ij}\right)^2 \quad (2)$$

В формуле (2) a – матрица значений функций активации нейронов, l – номер сверточного слоя. Такой способ вычисления функции потерь позволит минимизировать

разницу между представлением признаков сетью для стилизованного изображения и изображения контента.

Для расчета значения функции потерь стиля не получиться использовать такой же метод, что и для потерь функции потерь контента. Чтобы вычислить потерю стиля, необходимо измерить степень корреляции между картами признаков для изображения. То есть необходимо определить для отдельных участков изображения насколько часто различные цвета и текстуры встречаются друг с другом. Для расчета корреляции между картами признаков используется определитель Грама, который представляет из себя квадратную симметричную матрицу, составленную из скалярных произведений значений функций активации нейронов сверточного слоя. Чтобы рассчитать функцию потерь стиля, необходимо построить матрицы Грама для стилизованного изображения и изображения стиля и найти квадрат разности между элементами этих матрица.

$$L_{\text{стиля}}(S, G, l) = \frac{1}{4N_l^2 M_l^2} * \sum_{ij}(GM[l](S)_{ij} - GM[l](G)_{ij})^2 \qquad (3)$$

Используя формулу (3) в качестве функции потери стиля, минимизируется степень отклонения между стилизованным изображением и изображением стиля.

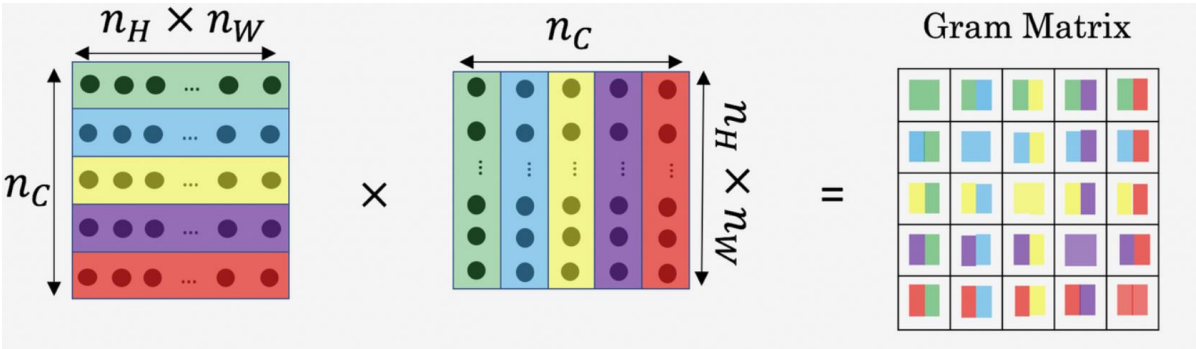


Рисунок 16 – Графическое представление матрицы Грама.

Перейдем к рассмотрению алгоритмов нейронного переноса стиля.

2.2.2 Алгоритм оптимизации изображения

Леон Гатис и Александр в Экель в рамках научной статьи [6] представили алгоритм оптимизации изображения. Идея алгоритма заключается использовании сверточной нейронной сети для извлечения информации о стиле и контенте из входных изображений, и на ее основе сгенерировать стилизованное изображение.

Алгоритм работает по следующем принципу:

– Генерируется произвольное изображение, на основе которого будет генерироваться стилизованное изображение;
– На вход сверточной нейронной сети подается изображение контента;

– Рассчитывается значение функции потерь контента;
– На вход сверточной нейронной сети подается изображение стиля;
– Рассчитывается значение функции потерь стиля;
– На основе значения общей функции потерь меняются значения пикселей обрабатываемого изображения.

Подобный подход называется оптимизацией изображений, так как меняются не весовые коэффициенты нейронной сети, а значения пикселей изображения. Главным недостатком алгоритм является его скорость работы. Связано это с тем, что процесс стилистической обработки изображения происходит итеративно, то есть приходится несколько раз пропускать обрабатываемое изображение через сверточную нейронную сеть, чтобы рассчитать значение функции потерь. Процесс стилистической обработки изображений с разрешением 1920x1080 может занимать порядка 10 минут. Данный недостаток не являлся бы проблемой, если бы оборудование, на котором осуществляется обработка изображения, не подвергалось большим нагрузкам.

2.2.3 Алгоритм Оптимизации модели

В 2016 году был представлен алгоритм [7], который основан не на оптимизации изображения, а на оптимизации нейронной сети. Автор статьи Дмитрий Ульянов представил реализацию алгоритма, основанного на оптимизации модели нейронной сети. Алгоритм также, как и алгоритм оптимизации изображения Гатиса [6], основан на использовании функции потерь, построенной на основе сверточной нейронной сети. Главным отличием алгоритм оптимизации модели [7] является использование генеративной нейронной сети для решения задачи стилистической обработки изображения. Функция потерь используется для обучения этой самой сети, то есть на основании ее значения мы можем определить, насколько хорошо обучаемая генеративная сеть стилизует изображения. Этот алгоритм позволяет обрабатывать изображения с разрешением Full HD меньше чем за 10 секунд, при этом стилизация происходит за один проход изображения через нейронную сеть из-за чего нагрузка на оборудование минимальна.

2.2.4 Генеративно-состязательные нейронные сети

Генеративно-состязательные нейронные сети – это алгоритм машинного обучения без учителя, который был предложен компанией Google в 2014 году. Принцип данного алгоритма основывается на “состязании” сети генератора и сети дискриминатора. Первая сеть пытается сгенерировать образец. Вторая сеть на вход получает результат работы

первой сети и некоторый эталон, после чего она должна определить правдоподобность образца полученного от первой сети.

Используя эту технику, можно генерировать изображения, которые человеческий глаз воспринимает как реальные фотографии. Были созданы модели сетей, которые генерирует изображения кошек. В итоге экспертам было достаточно сложно отличить реальные изображения от сгенерированных.

Сложность данного алгоритма заключается прежде всего в том, что необходимо обучать сразу две модели нейронных сетей. Также, если не соблюдать баланс между их обучением, то нейронные сети в целом не будут работать конкретно. Сеть генератора пытается обучиться создавать более качественный результат, дискримантор обучается распознавать реальные данные от сгенерированных.

Процесс обучения генеративно-состязательной модели основывается на том, что дискриминационная сеть, анализируя сгенерированные образцы и реальные данные, достигает некоторой точности различия. Генератор при этом начинает со случайного шума, а после оценки дискримантора, применяется метод обратного распространения ошибки, который позволяет улучшить качество генерации, подправив весовые коэффициенты нейронной сети. После несколько итераций состязания, качество сгенерированных изображений улучшается. Сеть дискриминатор представляет из себя сверточную нейронную сеть, а генератор, наоборот, разворачивает изображение на базе скрытых параметров.

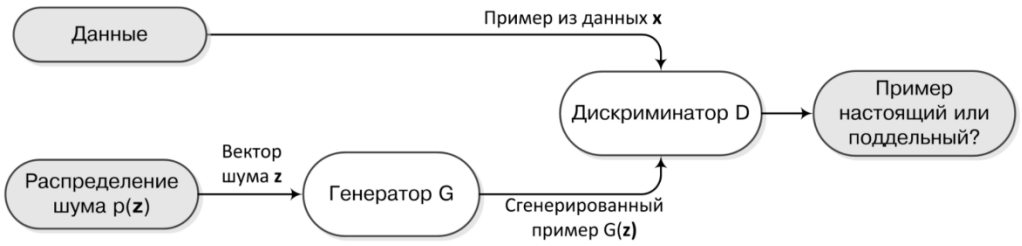


Рисунок 17 – Схема Генеративно-Состязательной нейронной сети.

3 Проектирование системы

3.1 Требования к функциональным возможностям

Проектируя систему, сначала нужно установить требования к функционалу этой системы:

- Возможность сохранения обработанных изображений в форматах jpg и png;
- Обработка изображений с максимальным разрешением 3840x2160;
- Время обработки одного изображения не более 1 минуты;
- Сохранение исходного разрешения для обработанного изображения;
- Возможность добавления новых стилей в систему.

Выбор алгоритма

Алгоритмы нефотореалистичного рендеринга для решения задачи стилистической обработки изображений сложны в реализации. Например, чтобы реализовать алгоритм для наложения одного стиля, потребуется несколько месяцев. Поэтому было решено опираться на решения, основанные на использовании нейросетвых технологий. Алгоритм оптимизации изображения [6] обрабатывает изображение в разрешении 1920x1080 порядка 5 минут, что не соответствует требованиям. Поэтому было принято использовать алгоритм нейронного переноса стиля, основанный на оптимизации модели [7]. Выбранный алгоритм позволяет обрабатывать изображения с любым разрешением. Также есть возможность добавлять новые стили в систему, не затрачивая на это много усилий. Для этого достаточно обучить еще одну модель нейронной сети.

3.2 Выбор средств разработки

На сегодняшний день для решения задач обработки изображений недостаточно использовать язык программирования без библиотек машинного обучения. Поэтому главными критериями выбора стали: наличие библиотек для машинного обучения и инструментов для работы с изображениями.

Следующим критериям удовлетворяют языки программирования – C++, C#, Python и Java. Каждый из перечисленных языков программирования активно развивается, за годы их существования были разработано большое количество инструментов и библиотек, в том числе и для машинного обучения. Существует и другое множество языков программирования, но они представляют недостаточно широкий спектр инструментов для решения поставленной задачи.

Начнем рассмотрение языков программирования C++. Используя C++ в качестве языка программирования, можно разработать систему для стилистической обработки

изображений, но сделать это достаточно сложно. Для C++ представлено очень мало библиотек по машинному обучению. Да, можно использовать OpenCV или портировать Tensorflow на C++, но аналогичная библиотека на Python будет работать в 1,5 раза быстрее и при этом взаимодействовать с ней будет гораздо проще. Также стоит учитывать, что разработка системы может занять порядка 2-3 месяцев, что слишком долго.

Java и C# предлагают примерно одинаковые возможности для разработки. С использованием этих языков программирования можно спроектировать и реализовать нейронную сеть. Для Java разработана библиотека машинного обучения Deeplearning4, а для C# - ML.NET. Но использовать эти языки программирования для решения задач машинного обучения не является целесообразным из-за их скорости работы. Программы, написанные на Java и C#, работают стабильно и надежно, но стоит учитывать, что они запускаются в виртуальной машине, которая работает поверх операционной системы, что значительно замедляет скорость работы.

Python – это интерпретируемый язык программирования, который позволяет разрабатывать программы быстро и без особых сложностей. Большинство библиотек для машинного обучения написаны именно для Python. Также для этого языка программирования представлены удобные библиотеки для работы с изображениями, например Pillow.

Опираясь на вышесказанное, в качестве языка разработки был выбран Python, так как он предоставляет удобные инструменты для разработки, что позволит реализовать систему без каких-либо сложностей. Java и C# не были выбраны из-за их скорости работы, а разработка на C++ слишком сложна и занимает много времени.

3.3 Библиотеки машинного обучения Python

На сегодняшний день представлено большое количество библиотек для машинного обучения на Python. Рассмотрим основные из них:

Scikit-learn

Легковесная библиотека машинного обучения, которая предоставляет простые и эффективные инструменты для решения задач обнаружения и анализа данных. Использовать эту библиотеку для глубокого обучения достаточно сложно, так отсутствуют инструменты для оптимизации этого процесса.

Theano

Theano – библиотека, разработанная группой LISA в Монреальском университет в Канаде. Инструменты данной библиотеки предназначены для осуществления быстрых числовых вычислений. Отлично подходит для решения задач машинного обучения.

Tensorflow

Tensorflow – является самой популярной библиотекой машинного обучения и глубокого обучения. Данная библиотека очень хороша оптимизирована для решения задач, связанных с обучением нейронных сетей, и предоставляет простой API для ее использования. Tensorflow разработана компаний Google, поэтому постоянно поддерживается и оптимизируется из года в год. Главными преимуществом Tensroflow являются: простота, скорость работы и большое комьюнити. Используя данную библиотеку, можно без проблем найти ответы на возникающие вопросы.

В рамках решения задачи стилистической обработки изображений, будет использоваться технология глубокого обучения, поэтому библиотека Scikit-learn не была выбрана в качестве инструмента разработки. Выбирая между Tensorflow и Theano, необходимо учитывать опыт работы с этими библиотеками, размер комьюнити и качество документации.

В качестве инструмента для проектирования и обучения нейронной сети была выбрана библиотека Tensorflow, так за время обучения был получен опыт работы с ней. Также компания Google предоставляет подробную документацию [7] по использованию библиотеки, что позволит сэкономить время на решении возникающих проблем. И немаловажным является то, что количество проектов реализованных на Tensorflow значительно больше, чем на Theano. Опираясь на данные платформы github, на Tensorflow реализовано порядка 82000 проектов, а на Theano лишь около 2000 проектов.

3.4 Дополнительные инструменты

Python Image Library (PIL) – библиотека, предназначенная для работы с растровой графикой. Разработка библиотеки была прекращена в 2011 году, но сообщество на ее базе создала форк Pillow, который поддерживается на всех платформах Windows, Ubuntu и др. Так как мы решаем задачу стилистической обработки изображений, то нам понадобится данная библиотека для преобразования формата изображений, их хранения, чтения и записи.

Для использования возможностей Tensorflow использовать для обучения графический процессор, необходимо использовать CUDA и cuDNN.

CUDA и cuDNN – это программные продукты разработанные компанией Nvidia для использования возможностей графических ускорителей видеокарт их производства. Без использования данных продуктов не представляется возможным обучение нейронной сети на графическом процессоре, что не дает воспользоваться возможностью ускорить процесс обучения нейронной сети на несколько порядков.

3.5 Проектирование модели нейронной сети

Для реализации алгоритма нейронного переноса стиля [6] необходимо использовать сверточную нейронную сеть. Обучение собственной модели для решения задачи классификации изображений может занять порядка месяца. Поэтому было принято решение использовать предобученную сверточную нейронный сеть.

На сегодняшний день существует большое количество архитектур сверточных нейронных сетей, таких как: VGG, ResNet, Inception и др. Опираясь на рекомендации авторов научной статьи [6] и эксперименты, проведенные в рамках статьи [8], было принято решение использовать предобученную сверточную нейронную сеть VGG16.

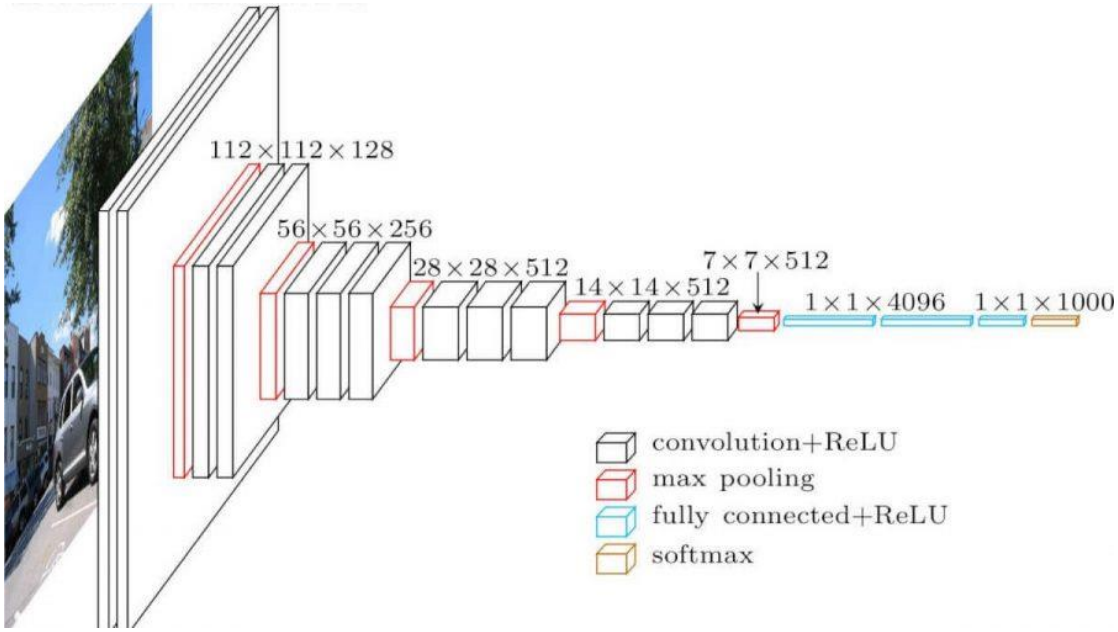
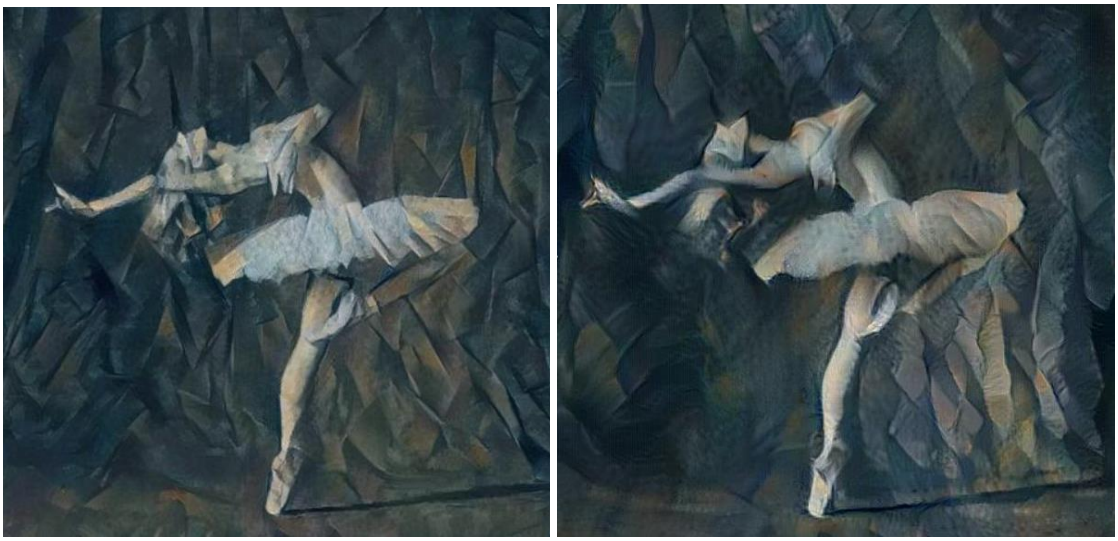


Рисунок 18 - Архитектура сверточной нейронной сети VGG16.

Сверточная нейронная сеть с архитектурой VGG16 решает хуже задачу классификации изображений, чем сети с архитектурами ResNet и Inception, но при решении задачи стилистической обработки изображений получаются более красивые изображения Рисунок 19.

Таблица 1 – Сравнение ошибок распознавания для различных архитектур CNN.

Архитектура сети	Топ 1 ошибка	Топ 5 ошибка
VGG16	70.5%	91.2%
ResNet-50	75.2%	93%
Inseption V1	69.8%	89.3%
Inseption V3	78.8%	94.4%



а) б)

Рисунок 19 – Пример обработанного изображения. а) Стилизация сетью VGG16. б) Стилизация сетью ResNet-50.

В качестве сети генератора была разработана архитектура сети, состоящая из нескольких слоев свертки, остаточных слоев и слоев развертки (деконволюции) Рисунок 20.

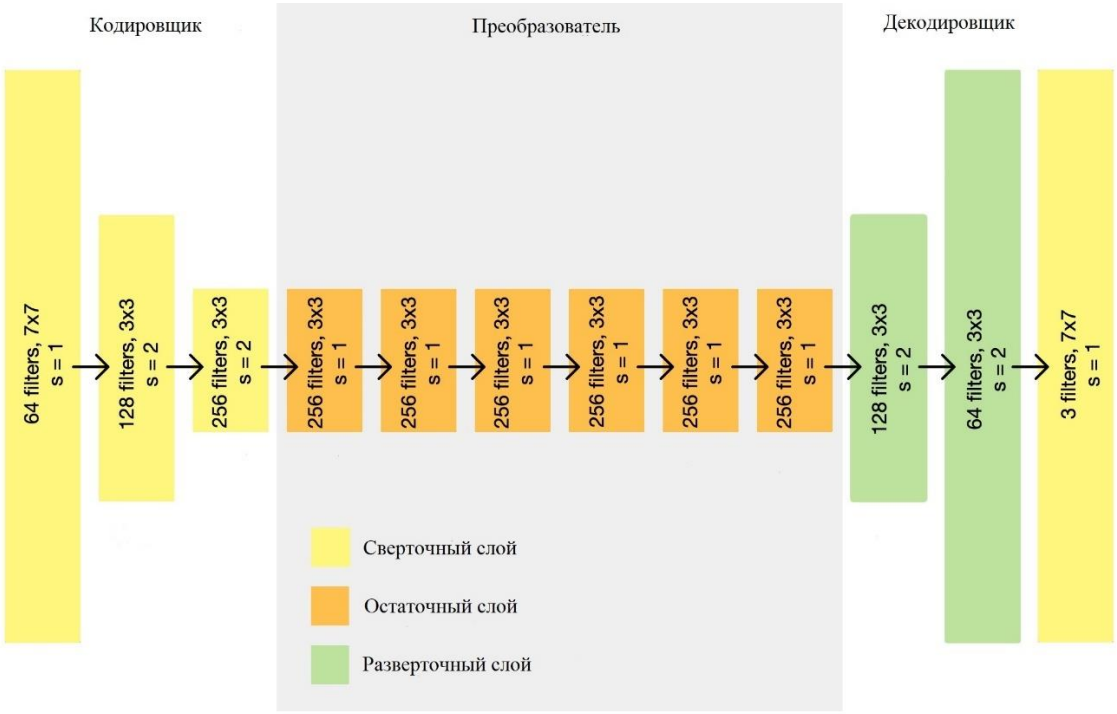


Рисунок 20 – Архитектура сети генератора для стилистической обработки изображений.

Кодировщик основан на использовании сверточных слоев. Слои свертки выделяют поверхностные признаки, к которым также относятся и цвет изображений. Данные слои выполняют функцию частичного отсечения стиля входного изображения от его содержимого. Это необходимо для более качественной стилистической обработки.

Преобразователь выполняет непосредственное наложение стиля на изображение. Данная генеративная сеть обучается на основе метода градиентного спуска и обратного распространения ошибки. При использовании данного класса методов возникает проблема исчезающего градиента, которая заключается в том, что при обратном распространении ошибки в слоях, которые ближе к входному слою, значения весовых коэффициентов почти меняются. Одним из возможных решений может быть разработка многослойной нейронной сети, но при решении задачи стилистической обработки изображения она получится слишком тяжеловесна и с большим количеством параметров, что приведет к ее долгому обучению. Так же более глубокая сеть будет обладать более высокой ошибкой обучения. Поэтому для решения проблемы исчезающего градиента в архитектуре сети генератора используются остаточные слои (Residual layers). Данный подход к обучению нейронных сетей подробнее рассматривается в научной статье [9]. Использование остаточных слоев позволило относительно легко оптимизировать нейронную сеть и также увеличить качество обучения за счет увеличения глубины сети.

Декодировщик выполняет функцию генерации стилизованного изображения. Процесс генерации стилизованного изображения основан на операции развертывания

признаков, выделенных нейронной сетью. Для достижения этого используются слои развертки (деконволюции). Слои развертки и отличают генеративную сеть от обычной сверточной нейронной сети. Слой деконволюции выполняет обратную задачу по сравнению со слоями свертки. Функция слоя основывается на том, что, зная размеры ядра свертки, можно восстановить изображение (расширить карту признаков).

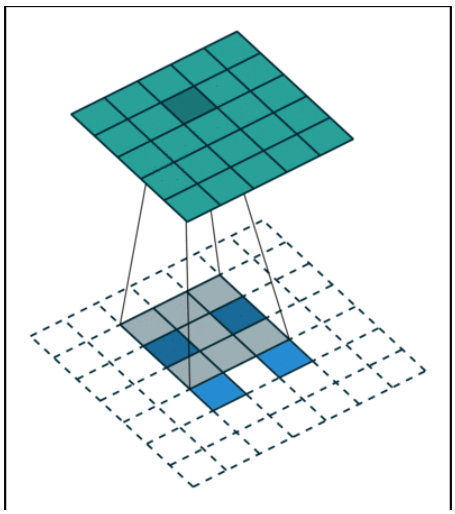


Рисунок 21 – Визуальное представление операции развертывания.

4 Реализация

Обучение сети генератора проводилось на основе стационарного компьютера со следующими характеристиками:

Таблица 2 – Характеристики используемого оборудования.

ОСНОВНЫЕ ХАРАКТЕРИСТИКИ	
Операционная система	Windows 10 Профессиональная
ПРОЦЕССОР	
Производитель процессора	Intel
Модель процессора	Core i7-9700K
Тактовая частота	3.4 ГГц
Максимальная тактовая частота	5 ГГц
Количество ядер	8
Кэш L2	12 МБ
ОПЕРАТИВНАЯ ПАМЯТЬ	
Объем оперативной памяти	32 ГБ
Тип оперативной памяти	DDR4
Частота памяти	2666 МГц
НАКОПИТЕЛЬ	
Тип накопителя	SSD
Объем накопителя	1 ТБ
ГРАФИЧЕСКАЯ СИСТЕМА	
Тип видеокарты	Дискретная
Производитель видеокарты	NVidia
Модель видеокарты	GeForce GTX1070
Объем видеопамяти	8 ГБ
Тип видеопамяти	GDDR5

4.1 Программное обеспечение

Для реализации архитектуры сети и ее дальнейшего обучения использовались следующие нструменты:

- IDE PyCharm 2019;
- Tensorflow-gpu 1.14.0;

– CUDA 10.0;
– cuDNN 11.0.

4.2 Обучающая выборка

Для обучения генеративной модели нейронной сети необходимо собрать базу изображений, содержащую объекты, которые относятся к разным классам. В качестве обучающей выборки был выбран готовый набор данных MS COCO2014. Данный набор состоит из 82763 изображений, которые содержат объекты разных классов.

Рисунок 22 – Пример изображений набора MS COCO.

4.3 Подготовка к обучению

Для обучения нейронной сети было разработано несколько модулей.

Модуль utils.py содержит в себе функции для загрузки, сохранения и подготовки изображений. Код модуль можно посмотреть в приложении 1. Для загрузки фотографий с помощью данного модуля необходимо обучающую выборку поместить в директорию, где находится проект.

Модуль vgg16.py содержит в себе структуру, содержащую программное представление архитектуры сверточной нейронной сети, а также методы для загрузки предобученной модели. Чтобы запустить процесс обучения, для начала необходимо загрузить предобученную vgg16 из интернета.

Модуль transform.py содержи программное представление архитектуры генеративной нейронной сети, методы для формирования сверточных, остаточных и развертоочных слоев.

Модуль run_train.py необходим для ручной настройки параметров обучения. В данном модуле можно настроить количество эпох, размер батчей и другие параметры. Также есть возможность передать изображение для тестирования работы сети во время обучения.

Модуль style_transfer_trainer.py осуществляет непосредственную инициализацию моделей сетей и их последующее обучение. Также в данном модули описаны методы для расчета функций потерь стиля, контента.

Также в модуль обучения можно передать следующие параметры:

- --content_weight – вес потерь контента;
- --style_weight – вес потерь стиля;
- --content_layers – сверточный слой для расчета потерь контента;
- --style_layers – сверточные слои для расчета потерь стиля;
- --max_size – максимальный размер входного изображения;
- --num_epochs – количество эпох;
- --batch_size – размер батча;
- --checkpoint_every – частота сохранения результатов обучения;
- --test – путь к изображению для тестирования работы генеративной сети;

4.4 Процесс обучения

Для обучения генеративной сети, необходимо на вход сверточной нейронной сети подать изображения стиля, контента и сгенерированное первой сетью изображение. На основании входных данных необходимо рассчитать значение функции потерь и передать его генеративной сети для обновления весов сети.

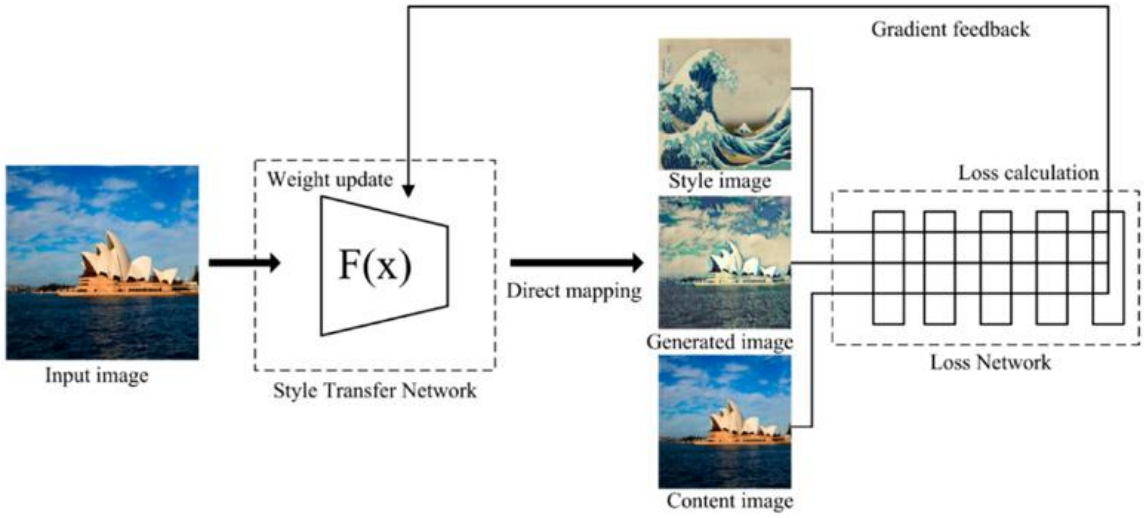


Рисунок 23 – Схема обучения генеративной нейронной сети.

Для начала процесса обучения необходимо открыть командную строку и прописать там следующую команду:

python run_train.py --style <style directory> --output <mode directory> --trainDB <trainDB directory> --vgg_model <vgg directory>

Нейронная сеть обучилась со следующими параметрами:

- Эпох – 2;
- Размер батча – 8;
- Весовой коэффициент потерь контента – 0,75;
- Весовой коэффициент потерь стиля – 0,25;
- Слой для расчета потерь контента – relu_4_2;

Обучение генеративной сети осуществлялось с использованием GPU. Процесс обучения занимает порядка 5 часов. При использовании CPU для обучения, процесс занимает 60 часов.

4.5 Тестирование полученных результатов

Для тестирования работы нейронной сети было разработано несколько модулей.

Модуль run_test.py необходим для передачи параметров тестирования: путь к директории с обученной моделью для определенного стиля, путь к изображению для обработки.

Модуль style_transfer_trainer.py необходим для инициализации обученной модели для стилистической обработки изображения и последующего ее использования.

Используем обученные нейронные сети для стилистической обработки изображений.

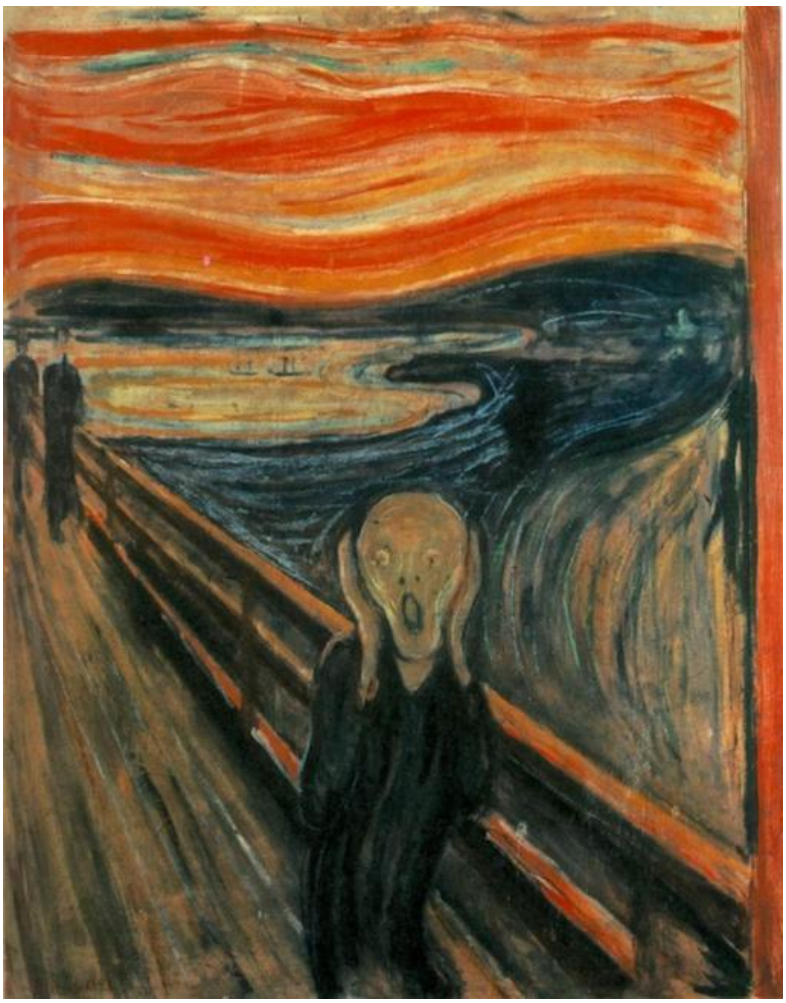
а)

б)

в)

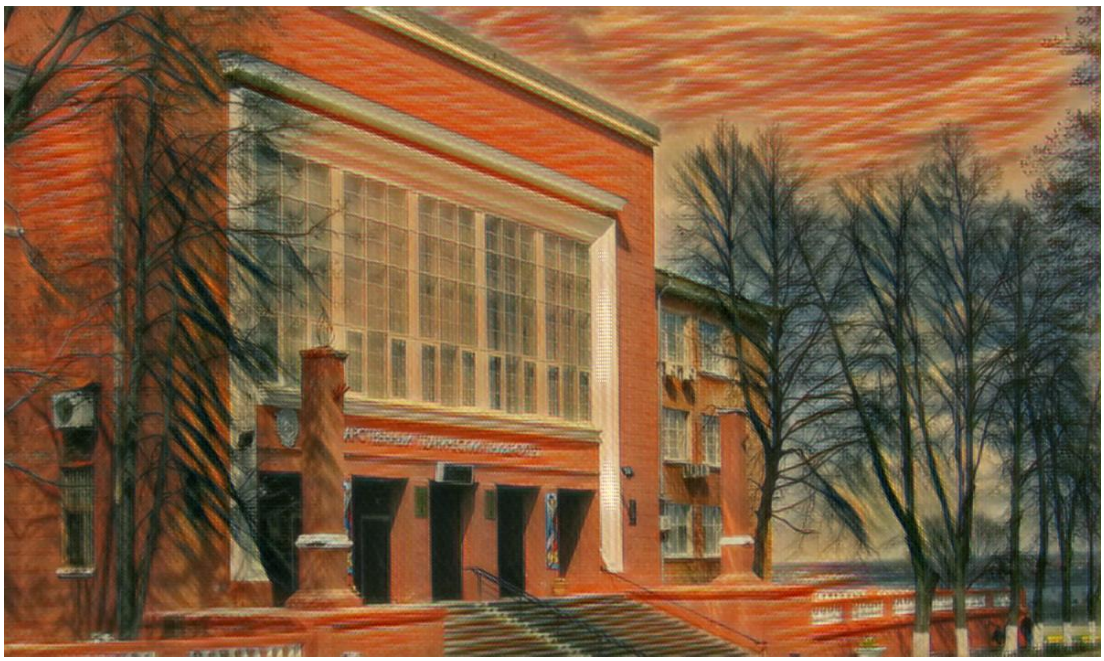
Рисунок 24 – Пример стилистической обработки изображений.

а) Изображение стиля картина Франсиса Пикабиа “Udnie”; б) Фотография Нижегородского Кремля; в) Стилизованное изображение.



а)

б)



в)

Рисунок 25 Пример стилистической обработки изображения. а) Изображение стиля картина Эдварда Мунка “Крик”; б) Фотография первого корпуса НГТУ им. Р.Е. Алексеева; в) Стилизованное изображение.

4.6 Тестирование скорости алгоритма

Протестируем скорость работы алгоритм, в зависимости от разрешения изображения. В качестве тестовых данных будет использоваться изображение, представленное на Рисунке 26, в разных разрешениях.

Рисунок 26 – Изображение для тестирования скорости работы алгоритма.

Тестирование проводился для CPU и GPU. Результаты тестирования представлены в Таблице 3.

Таблица 3 Сравнение скорости работы алгоритма на CPU и GPU

Разрешение изображения	Время обработки (CPU), мсек	Время обработки (GPU), мсек
720x480	14,415	31,511
1024x768	30,511	45,934
1366x768	38, 665	58,639
1920x1080	82,124	90,165
2880x1800	178,750	179,618
3840x2160	268,917	267,311

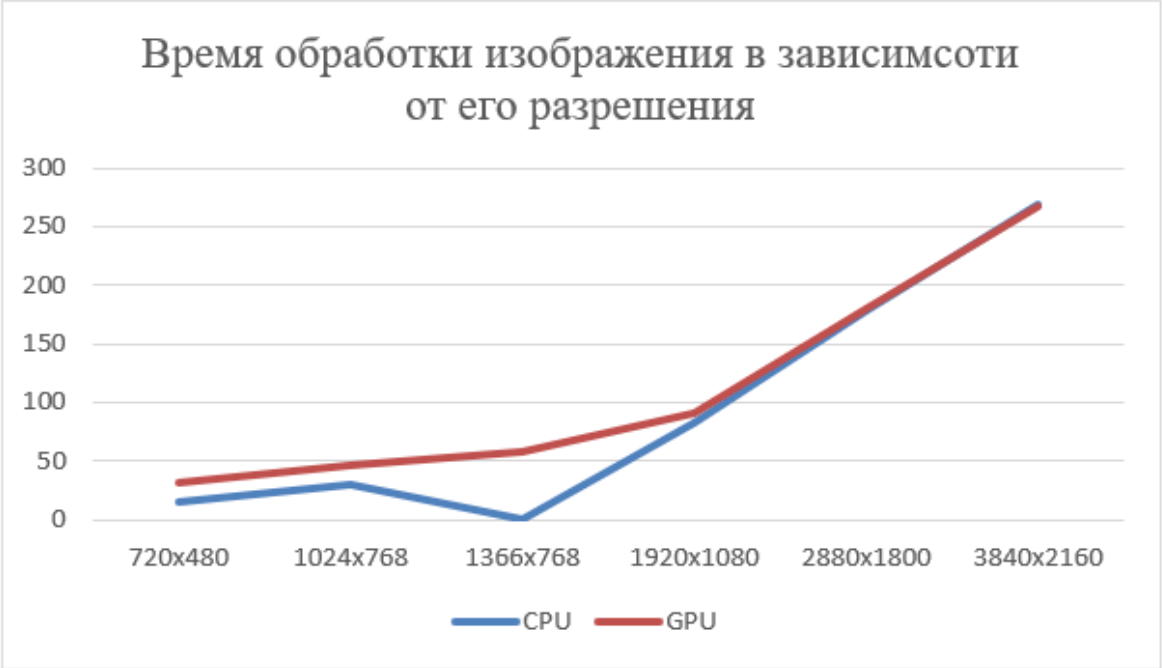


Рисунок 27 – График скорости работы алгоритма в зависимости от разрешения изображения.

На основе приведенных результатов можно сделать вывод, что для изображений с небольшим разрешением алгоритм переноса стиля на CPU работает быстрее в 1,5 – 2 раза, чем на GPU. При увеличении разрешения изображения скорость работы алгоритма одинакова, независимо от используемого типа процессора.

Нейронная сеть в режиме использовании занимает порядка 4Gb оперативной памяти или же графической памяти. Из-за этого при использовании GPU нет возможности

параллельно использовать несколько нейронных сетей. Решением данной проблемы может быть использование видеокарты с большим объемом памяти или же стоит использовать CPU для обработки изображений, так как объем оперативной памяти больше, чем графической, в 4 раза.

Заключение

В данной выпускной квалификационной работе была разработана архитектура нейронной сети для стилистической обработки изображений и ее реализация на языке Python.

В ходе работы были рассмотрены существующие алгоритмы стилистической обработки изображений. На основании собранной информации и функциональных требований для построения системы стилизации изображений был выбран алгоритм оптимизации модели [7]. На основании данного алгоритма была обучена и протестирована генеративная сеть, которая способна создавать действительно красивые изображения.

Обученная генеративная нейронная сеть позволяет обрабатывать изображения в максимальном разрешении 3840x2160. Процесс обработки одного изображения занимает не более 10 секунд. Также процесс добавления нового стиля занимает порядка 6 часов.

На основании реализованной системы был разработан программный продукт, основанный на технологии чат-ботов в мессенджере Telegram.

Список литературы

1. **Haeberli, Paul.** Paint by numbers: abstract image representations. [В Интернете] [Цитировано: 13 06, 2020 г.] https://dl.acm.org/doi/10.1145/97879.97902.

2. **Holger Winnemöller, Sven C. Olsen, Bruce Gooch.** Real-time video abstraction. [В Интернете] [Цитировано: 13 05, 2020 г.] https://dl.acm.org/doi/10.1145/1179352.1142018.

3. **Michael P. Salisbury, Sean E Anderson, Ronen Barzel, David H. Salesin.** Interactive pen-and-ink illustration. [В Интернете] [Цитировано: 13 05, 2020 г.] https://dl.acm.org/doi/10.1145/192161.192185.

4. **Litwinowicz, Peter.** Processing images and video for an impressionist effect. [В Интернете] [Цитировано: 05 05, 2020 г.] https://dl.acm.org/doi/10.1145/258734.258893.

5. **Aaron Hertzmann, Charles E. Jacobs , Nuria Oliver, Brian Curless, David H. Salesin.** Image analogies. [В Интернете] [Цитировано: 08 05, 2020 г.] https://dl.acm.org/doi/10.1145/383259.383295.

6. **Leon A. Gatys, Alexander S. Ecker, Matthias Bethge.** A Neural Algorithm of Artistic Style. [В Интернете] [Цитировано: 24 05, 2020 г.] https://arxiv.org/pdf/1508.06576.pdf.

7. **Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky.** Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis. [В Интернете] [Цитировано: 28 05, 2020 г.] https://arxiv.org/pdf/1701.02096.pdf.

8. **Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.** Deep Residual Learning for Image Recognition. [В Интернете] [Цитировано: 03 06, 2020 г.] https://arxiv.org/pdf/1512.03385.pdf.

9. **Nakano, Reiichiro.** Adversarially Robust Neural Style Transfer. [В Интернете] [Цитировано: 17 06, 2020 г.] https://distill.pub/2019/advex-bugs-discussion/response-4/.

Приложение А

utils.py

```
import numpy as np
import PIL.Image
import os
import scipy

"""Helper-functions to load MSCOCO DB"""
# borrowed from https://github.com/lengstrom/fast-style-
transfer/blob/master/src/utils.py
def get_img(src, img_size=False):
   img = scipy.misc.imread(src, mode='RGB')
   if not (len(img.shape) == 3 and img.shape[2] == 3):
      img = np.dstack((img,img,img))
   if img_size != False:
      img = scipy.misc.imresize(img, img_size)
   return img

def get_files(img_dir):
    files = list_files(img_dir)
    return list(map(lambda x: os.path.join(img_dir,x), files))

def list_files(in_path):
    files = []
    for (dirpath, dirnames, filenames) in os.walk(in_path):
        files.extend(filenames)
        break
    return files

"""Helper-functions for image manipulation"""
# borrowed from https://github.com/Hvass-Labs/TensorFlow-
Tutorials/blob/master/15_Style_Transfer.ipynb

# This function loads an image and returns it as a numpy array of floating-points.
```

```
# The image can be automatically resized so the largest of the height or width
equals max_size.
# or resized to the given shape
def load_image(filename, shape=None, max_size=None):
    image = PIL.Image.open(filename)

    if max_size is not None:
        # Calculate the appropriate rescale-factor for
        # ensuring a max height and width, while keeping
        # the proportion between them.
        factor = float(max_size) / np.max(image.size)

        # Scale the image's height and width.
        size = np.array(image.size) * factor

        # The size is now floating-point because it was scaled.
        # But PIL requires the size to be integers.
        size = size.astype(int)

        # Resize the image.
        image = image.resize(size, PIL.Image.LANCZOS) # PIL.Image.LANCZOS is
one of resampling filter

    if shape is not None:
        image = image.resize(shape, PIL.Image.LANCZOS) # PIL.Image.LANCZOS
is one of resampling filter

    # Convert to numpy floating-point array.
    return np.float32(image)

# Save an image as a jpeg-file.
# The image is given as a numpy array with pixel-values between 0 and 255.
def save_image(image, filename):
    # Ensure the pixel-values are between 0 and 255.
    image = np.clip(image, 0.0, 255.0)

    # Convert to bytes.
    image = image.astype(np.uint8)
```

```
    # Write the image-file in jpeg-format.
    with open(filename, 'wb') as file:
        PIL.Image.fromarray(image).save(file, 'jpeg')
```

transform.py

```
import tensorflow as tf

class Transform:
    def __init__(self, mode='train'):
        if mode == 'train':
            self.reuse = None
        else:
            self.reuse = True

    def net(self, image):
        image_p = self._reflection_padding(image)
        conv1 = self._conv_layer(image_p, 32, 9, 1, name='conv1')
        conv2 = self._conv_layer(conv1, 64, 3, 2, name='conv2')
        conv3 = self._conv_layer(conv2, 128, 3, 2, name='conv3')
        resid1 = self._residual_block(conv3, 3, name='resid1')
        resid2 = self._residual_block(resid1, 3, name='resid2')
        resid3 = self._residual_block(resid2, 3, name='resid3')
        resid4 = self._residual_block(resid3, 3, name='resid4')
        resid5 = self._residual_block(resid4, 3, name='resid5')
        conv_t1 = self._conv_tranpose_layer(resid5, 64, 3, 2, name='convt1')
        conv_t2 = self._conv_tranpose_layer(conv_t1, 32, 3, 2, name='convt2')
        conv_t3 = self._conv_layer(conv_t2, 3, 9, 1, relu=False, name='convt3')
        preds = (tf.nn.tanh(conv_t3) + 1) * (255. / 2)
        return preds

    def _reflection_padding(self, net):
        return tf.pad(net,[[0, 0],[40, 40],[40, 40], [0, 0]], "REFLECT")

    def _conv_layer(self, net, num_filters, filter_size, strides, padding='SAME',
relu=True, name=None):
        weights_init = self._conv_init_vars(net, num_filters, filter_size, name=name)
```

```
        strides_shape = [1, strides, strides, 1]
        net = tf.nn.conv2d(net, weights_init, strides_shape, padding=padding)
        net = self._instance_norm(net, name=name)
        if relu:
            net = tf.nn.relu(net)

        return net

    def _conv_tranpose_layer(self, net, num_filters, filter_size, strides,
name=None):
        weights_init = self._conv_init_vars(net, num_filters, filter_size,
transpose=True, name=name)

        batch_size, rows, cols, in_channels = [i.value for i in net.get_shape()]
        new_rows, new_cols = int(rows * strides), int(cols * strides)

        new_shape = [batch_size, new_rows, new_cols, num_filters]
        tf_shape = tf.stack(new_shape)
        strides_shape = [1,strides,strides,1]

        net = tf.nn.conv2d_transpose(net, weights_init, tf_shape, strides_shape,
padding='SAME')
        net = self._instance_norm(net, name=name)
        return tf.nn.relu(net)

    def _residual_block(self, net, filter_size=3, name=None):
        batch, rows, cols, channels = [i.value for i in net.get_shape()]
        tmp = self._conv_layer(net, 128, filter_size, 1, padding='VALID', relu=True,
name=name+'_1')
        return self._conv_layer(tmp, 128, filter_size, 1, padding='VALID', relu=False,
name=name+'_2') + tf.slice(net, [0,2,2,0], [batch,rows-4,cols-4,channels])

    def _instance_norm(self, net, name=None):
        batch, rows, cols, channels = [i.value for i in net.get_shape()]
        var_shape = [channels]
        mu, sigma_sq = tf.nn.moments(net, [1,2], keep_dims=True)
        with tf.variable_scope(name, reuse=self.reuse):
```

```
            shift = tf.get_variable('shift', initializer=tf.zeros(var_shape),
dtype=tf.float32)
            scale = tf.get_variable('scale', initializer=tf.ones(var_shape),
dtype=tf.float32)
        epsilon = 1e-3
        normalized = (net-mu)/(sigma_sq + epsilon)**(.5)
        return scale * normalized + shift

    def _conv_init_vars(self, net, out_channels, filter_size, transpose=False,
name=None):
        _, rows, cols, in_channels = [i.value for i in net.get_shape()]
        if not transpose:
            weights_shape = [filter_size, filter_size, in_channels, out_channels]
        else:
            weights_shape = [filter_size, filter_size, out_channels, in_channels]
        with tf.variable_scope(name, reuse=self.reuse):
            weights_init = tf.get_variable('weight', shape=weights_shape,
initializer=tf.contrib.layers.variance_scaling_initializer(), dtype=tf.float32)
        return weights_init
```

vgg16.py

```
# Copyright (c) 2015-2016 Anish Athalye. Released under GPLv3.
# Most code in this file was borrowed from https://github.com/anishathalye/neural-
style/blob/master/vgg.py

import tensorflow as tf
import numpy as np
import scipy.io

# download URL : http://www.vlfeat.org/matconvnet/models/imagenet-vgg-
verydeep-19.mat
MODEL_FILE_NAME = 'imagenet-vgg-verydeep-19.mat'

def _conv_layer(input, weights, bias):
    conv = tf.nn.conv2d(input, tf.constant(weights), strides=(1, 1, 1, 1),
            padding='SAME')
    return tf.nn.bias_add(conv, bias)
```

```
def _pool_layer(input):
    return tf.nn.max_pool(input, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1),
            padding='SAME')

def preprocess(image, mean_pixel):
    return image - mean_pixel

def undo_preprocess(image, mean_pixel):
    return image + mean_pixel

class VGG19:
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

        'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',

        'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
        'relu3_3', 'conv3_4', 'relu3_4', 'pool3',

        'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
        'relu4_3', 'conv4_4', 'relu4_4', 'pool4',

        'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
        'relu5_3', 'conv5_4', 'relu5_4'
    )

    def __init__(self, data_path):
        data = scipy.io.loadmat(data_path)

        self.mean_pixel = np.array([123.68, 116.779, 103.939])

        self.weights = data['layers'][0]

    def preprocess(self, image):
        return image-self.mean_pixel

    def undo_preprocess(self,image):
```

```
        return image+self.mean_pixel

    def feed_forward(self, input_image, scope=None):
        net = {}
        current = input_image

        with tf.variable_scope(scope):
            for i, name in enumerate(self.layers):
                kind = name[:4]
                if kind == 'conv':
                    kernels = self.weights[i][0][0][2][0][0]
                    bias = self.weights[i][0][0][2][0][1]

                    # matconvnet: weights are [width, height, in_channels, out_channels]
                    # tensorflow: weights are [height, width, in_channels, out_channels]
                    kernels = np.transpose(kernels, (1, 0, 2, 3))
                    bias = bias.reshape(-1)

                    current = _conv_layer(current, kernels, bias)
                elif kind == 'relu':
                    current = tf.nn.relu(current)
                elif kind == 'pool':
                    current = _pool_layer(current)
                net[name] = current

        assert len(net) == len(self.layers)
        return net
```

run_train.py

```
import tensorflow as tf
import numpy as np
import utils
import vgg19
import style_transfer_trainer
import os

import argparse
```

```
"""parsing and configuration"""
def parse_args():
    desc = "Tensorflow implementation of 'Image Style Transfer Using Convolutional Neural Networks"
    parser = argparse.ArgumentParser(description=desc)

    parser.add_argument('--vgg_model', type=str, default='pre_trained_model', help='The directory where the pre-trained model was saved', required=True)
    parser.add_argument('--trainDB_path', type=str, default='train2014',
                        help='The directory where MSCOCO DB was saved', required=True)
    parser.add_argument('--style', type=str, default='style/wave.jpg', help='File path of style image (notation in the paper : a)', required=True)
    parser.add_argument('--output', type=str, default='models', help='File path for trained-model. Train-log is also saved here.', required=True)

    parser.add_argument('--content_weight', type=float, default=7.5e0, help='Weight of content-loss')
    parser.add_argument('--style_weight', type=float, default=5e2, help='Weight of style-loss')
    parser.add_argument('--tv_weight', type=float, default=2e2, help='Weight of total-variance-loss')

    parser.add_argument('--content_layers', nargs='+', type=str, default=['relu4_2'], help='VGG19 layers used for content loss')
    parser.add_argument('--style_layers', nargs='+', type=str, default=['relu1_1', 'relu2_1', 'relu3_1', 'relu4_1', 'relu5_1'],
                        help='VGG19 layers used for style loss')

    parser.add_argument('--content_layer_weights', nargs='+', type=float, default=[1.0], help='Content loss for each content is multiplied by corresponding weight')
    parser.add_argument('--style_layer_weights', nargs='+', type=float, default=[.2,.2,.2,.2,.2],
                        help='Style loss for each content is multiplied by corresponding weight')
```

```
    parser.add_argument('--learn_rate', type=float, default=1e-3, help='Learning rate
for Adam optimizer')
    parser.add_argument('--num_epochs', type=int, default=2, help='The number of
epochs to run')
    parser.add_argument('--batch_size', type=int, default=4, help='Batch size')

    parser.add_argument('--checkpoint_every', type=int, default=1000, help='save a
trained model every after this number of iterations')

    parser.add_argument('--test', type=str, default=None,
                        help='File path of content image (notation in the paper : x)')

    parser.add_argument('--max_size', type=int, default=None, help='The maximum
width or height of input images')

    return check_args(parser.parse_args())

"""checking arguments"""
def check_args(args):

    # --vgg_model
    model_file_path = args.vgg_model + '/' + vgg19.MODEL_FILE_NAME
    try:
        assert os.path.exists(model_file_path)
    except:
        print('There is no %s' % model_file_path)
        return None
    try:
        size_in_KB = os.path.getsize(model_file_path)
        assert abs(size_in_KB - 534904783) < 10
    except:
        print('check file size of \'imagenet-vgg-verydeep-19.mat\')
        print('there are some files with the same name')
        print('pre_trained_model used here can be downloaded from bellow')
        print('http://www.vlfeat.org/matconvnet/models/imagenet-vgg-verydeep-
19.mat')
        return None
```

```
# --trainDB_path
try:
    assert os.path.exists(args.trainDB_path)
except:
    print('There is no %s' % args.trainDB_path)
    return None

# --style
try:
    assert os.path.exists(args.style)
except:
    print('There is no %s' % args.style)
    return None

# --output
dirname = os.path.dirname(args.output)
try:
    if len(dirname) > 0:
        os.stat(dirname)
except:
    os.mkdir(dirname)

# --content_weight
try:
    assert args.content_weight > 0
except:
    print('content weight must be positive')

# --style_weight
try:
    assert args.style_weight > 0
except:
    print('style weight must be positive')

# --tv_weight
try:
    assert args.tv_weight > 0
except:
```

```
    print('total variance weight must be positive')

# --content_layer_weights
try:
    assert len(args.content_layers) == len(args.content_layer_weights)
except:
    print ('content layer info and weight info must be matched')
    return None

# --style_layer_weights
try:
    assert len(args.style_layers) == len(args.style_layer_weights)
except:
    print('style layer info and weight info must be matched')
    return None

# --learn_rate
try:
    assert args.learn_rate > 0
except:
    print('learning rate must be positive')

# --num_epochs
try:
    assert args.num_epochs >= 1
except:
    print('number of epochs must be larger than or equal to one')

# --batch_size
try:
    assert args.batch_size >= 1
except:
    print('batch size must be larger than or equal to one')

# --checkpoint_every
try:
    assert args.checkpoint_every >= 1
except:
```

```
        print('checkpoint period must be larger than or equal to one')

    # --test
    try:
        if args.test is not None:
            assert os.path.exists(args.test)
    except:
        print('There is no %s' % args.test)
        return None

    # --max_size
    try:
        if args.max_size is not None:
            assert args.max_size > 0
    except:
        print('The maximum width or height of input image must be positive')
        return None

    return args

"""add one dim for batch"""
# VGG19 requires input dimension to be (batch, height, width, channel)
def add_one_dim(image):
    shape = (1,) + image.shape
    return np.reshape(image, shape)

"""main"""
def main():

    # parse arguments
    args = parse_args()
    if args is None:
        exit()

    # initiate VGG19 model
    model_file_path = args.vgg_model + '/' + vgg19.MODEL_FILE_NAME
    vgg_net = vgg19.VGG19(model_file_path)
```

```
# get file list for training
content_images = utils.get_files(args.trainDB_path)

# load style image
style_image = utils.load_image(args.style)

# create a map for content layers info
CONTENT_LAYERS = {}
for layer, weight in zip(args.content_layers,args.content_layer_weights):
   CONTENT_LAYERS[layer] = weight

# create a map for style layers info
STYLE_LAYERS = {}
for layer, weight in zip(args.style_layers, args.style_layer_weights):
   STYLE_LAYERS[layer] = weight

# open session
sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True))

# build the graph for train
trainer = style_transfer_trainer.StyleTransferTrainer(session=sess,
                                    content_layer_ids=CONTENT_LAYERS,
                                    style_layer_ids=STYLE_LAYERS,
                                    content_images=content_images,
                                    style_image=add_one_dim(style_image),
                                    net=vgg_net,
                                    num_epochs=args.num_epochs,
                                    batch_size=args.batch_size,
                                    content_weight=args.content_weight,
                                    style_weight=args.style_weight,
                                    tv_weight=args.tv_weight,
                                    learn_rate=args.learn_rate,
                                    save_path=args.output,
                                    check_period=args.checkpoint_every,
                                    test_image=args.test,
                                    max_size=args.max_size,
                                    )
# launch the graph in a session
```

```
    trainer.train()

    # close session
    sess.close()

if __name__ == '__main__':
    main()
```

style_transfer_trainer.py

```
import tensorflow as tf
import numpy as np
import collections
import transform
import utils
import style_transfer_tester

class StyleTransferTrainer:
    def __init__(self, content_layer_ids, style_layer_ids, content_images,
style_image, session, net, num_epochs,
                 batch_size, content_weight, style_weight, tv_weight, learn_rate,
save_path, check_period, test_image,
                 max_size):

        self.net = net
        self.sess = session

        # sort layers info
        self.CONTENT_LAYERS =
collections.OrderedDict(sorted(content_layer_ids.items()))
        self.STYLE_LAYERS =
collections.OrderedDict(sorted(style_layer_ids.items()))

        # input images
        self.x_list = content_images
        mod = len(content_images) % batch_size
        self.x_list = self.x_list[:-mod]
        self.y_s0 = style_image
```

```
        self.content_size = len(self.x_list)

        # parameters for optimization
        self.num_epochs = num_epochs
        self.content_weight = content_weight
        self.style_weight = style_weight
        self.tv_weight = tv_weight
        self.learn_rate = learn_rate
        self.batch_size = batch_size
        self.check_period = check_period

        # path for model to be saved
        self.save_path = save_path

        # image transform network
        self.transform = transform.Transform()
        self.tester = transform.Transform('test')

        # build graph for style transfer
        self._build_graph()

        # test during training
        if test_image is not None:
            self.TEST = True

            # load content image
            self.test_image = utils.load_image(test_image, max_size=max_size)

            # build graph
            self.x_test = tf.placeholder(tf.float32, shape=self.test_image.shape,
name='test_input')
            self.xi_test = tf.expand_dims(self.x_test, 0)  # add one dim for batch

            # result image from transform-net
            self.y_hat_test = self.tester.net(
                self.xi_test / 255.0)  # please build graph for train first. tester.net reuses
variables.
```

```
        else:
            self.TEST = False

    def _build_graph(self):

        """ prepare data """

        self.batch_shape = (self.batch_size,256,256,3)

        # graph input
        self.y_c = tf.placeholder(tf.float32, shape=self.batch_shape, name='content')
        self.y_s = tf.placeholder(tf.float32, shape=self.y_s0.shape, name='style')

        # preprocess for VGG
        self.y_c_pre = self.net.preprocess(self.y_c)
        self.y_s_pre = self.net.preprocess(self.y_s)

        # get content-layer-feature for content loss
        content_layers = self.net.feed_forward(self.y_c_pre, scope='content')
        self.Ps = {}
        for id in self.CONTENT_LAYERS:
            self.Ps[id] = content_layers[id]

        # get style-layer-feature for style loss
        style_layers = self.net.feed_forward(self.y_s_pre, scope='style')
        self.As = {}
        for id in self.STYLE_LAYERS:
            self.As[id] = self._gram_matrix(style_layers[id])

        # result of image transform net
        self.x = self.y_c/255.0
        self.y_hat = self.transform.net(self.x)

        # get layer-values for x
        self.y_hat_pre = self.net.preprocess(self.y_hat)
        self.Fs = self.net.feed_forward(self.y_hat_pre, scope='mixed')

        """ compute loss """
```

```
        # style & content losses
        L_content = 0
        L_style = 0
        for id in self.Fs:
            if id in self.CONTENT_LAYERS:
                ## content loss ##

                F = self.Fs[id]           # content feature of x
                P = self.Ps[id]           # content feature of p

                b, h, w, d = F.get_shape()  # first return value is batch size (must be one)
                b = b.value               # batch size
                N = h.value*w.value       # product of width and height
                M = d.value               # number of filters

                w = self.CONTENT_LAYERS[id] # weight for this layer

                L_content += w * 2 * tf.nn.l2_loss(F-P) / (b*N*M)

            elif id in self.STYLE_LAYERS:
                ## style loss ##

                F = self.Fs[id]

                b, h, w, d = F.get_shape()        # first return value is batch size (must be
one)
                b = b.value                     # batch size
                N = h.value * w.value           # product of width and height
                M = d.value                     # number of filters

                w = self.STYLE_LAYERS[id]         # weight for this layer

                G = self._gram_matrix(F, (b,N,M))   # style feature of x
                A = self.As[id]                   # style feature of a

                L_style += w * 2 * tf.nn.l2_loss(G - A) / (b * (M ** 2))
```

```
        # total variation loss
        L_tv = self._get_total_variation_loss(self.y_hat)

        """ compute total loss """

        # Loss of total variation regularization
        alpha = self.content_weight
        beta = self.style_weight
        gamma = self.tv_weight

        self.L_content = alpha*L_content
        self.L_style = beta*L_style
        self.L_tv = gamma*L_tv
        self.L_total = self.L_content + self.L_style + self.L_tv

        # add summary for each loss
        tf.summary.scalar('L_content', self.L_content)
        tf.summary.scalar('L_style', self.L_style)
        tf.summary.scalar('L_tv', self.L_tv)
        tf.summary.scalar('L_total', self.L_total)

    # borrowed from https://github.com/lengstrom/fast-style-
transfer/blob/master/src/optimize.py
    def _get_total_variation_loss(self, img):
        b, h, w, d = img.get_shape()
        b = b.value
        h = h.value
        w = w.value
        d = d.value
        tv_y_size = (h-1) * w * d
        tv_x_size = h * (w-1) * d
        y_tv = tf.nn.l2_loss(img[:, 1:, :, :] - img[:, :self.batch_shape[1] - 1, :, :])
        x_tv = tf.nn.l2_loss(img[:, :, 1:, :] - img[:, :, :self.batch_shape[2] - 1, :])
        loss = 2. * (x_tv / tv_x_size + y_tv / tv_y_size) / b

        loss = tf.cast(loss, tf.float32)
        return loss
```

```
    def train(self):
        """ define optimizer Adam """
        global_step = tf.contrib.framework.get_or_create_global_step()

        trainable_variables = tf.trainable_variables()
        grads = tf.gradients(self.L_total, trainable_variables)

        optimizer = tf.train.AdamOptimizer(self.learn_rate)
        train_op = optimizer.apply_gradients(zip(grads, trainable_variables),
global_step=global_step,
                                             name='train_step')

        """ tensor board """
        # merge all summaries into a single op
        merged_summary_op = tf.summary.merge_all()

        # op to write logs to Tensorboard
        summary_writer = tf.summary.FileWriter(self.save_path,
graph=tf.get_default_graph())

        """ session run """
        self.sess.run(tf.global_variables_initializer())

        # saver to save model
        saver = tf.train.Saver()

        # restore check-point if it exits
        checkpoint_exists = True
        try:
            ckpt_state = tf.train.get_checkpoint_state(self.save_path)
        except tf.errors.OutOfRangeError as e:
            print('Cannot restore checkpoint: %s' % e)
            checkpoint_exists = False
        if not (ckpt_state and ckpt_state.model_checkpoint_path):
            print('No model to restore at %s' % self.save_path)
            checkpoint_exists = False

        if checkpoint_exists:
```

```
            tf.logging.info('Loading checkpoint %s',
ckpt_state.model_checkpoint_path)
            saver.restore(self.sess, ckpt_state.model_checkpoint_path)

        """ loop for train """
        num_examples = len(self.x_list)

        # get iteration info
        if checkpoint_exists:
            iterations = self.sess.run(global_step)
            epoch = (iterations * self.batch_size) // num_examples
            iterations = iterations - epoch*(num_examples // self.batch_size)
        else:
            epoch = 0
            iterations = 0
        my_file = open("some.txt", "w")
        my_file.write("Step L_total\n")
        while epoch < self.num_epochs:
            while iterations * self.batch_size < num_examples:

                curr = iterations * self.batch_size
                step = curr + self.batch_size
                x_batch = np.zeros(self.batch_shape, dtype=np.float32)
                for j, img_p in enumerate(self.x_list[curr:step]):
                    x_batch[j] = utils.get_img(img_p, (256, 256, 3)).astype(np.float32)

                iterations += 1

                assert x_batch.shape[0] == self.batch_size

                _, summary, L_total, L_content, L_style, L_tv, step = self.sess.run(
                    [train_op, merged_summary_op, self.L_total, self.L_content,
self.L_style, self.L_tv, global_step],
                    feed_dict={self.y_c: x_batch, self.y_s: self.y_s0})
                L_total = L_total / 1e+06
                res_string = "{} {}\n".format(step, L_total)
                my_file.write(res_string)
```

```
            # write logs at every iteration
            summary_writer.add_summary(summary, iterations)

            if step % self.check_period == 0:
                res = saver.save(self.sess, self.save_path + '/final.ckpt', step)

                if self.TEST:
                    output_image = self.sess.run([self.y_hat_test],
feed_dict={self.x_test: self.test_image})
                    output_image = np.squeeze(output_image[0])  # remove one dim
for batch
                    output_image = np.clip(output_image, 0., 255.)

                    utils.save_image(output_image, self.save_path + '/result_' + "%05d"
% step + '.jpg')
            epoch += 1
            iterations = 0
        res = saver.save(self.sess,self.save_path+'/final.ckpt')

    def _gram_matrix(self, tensor, shape=None):

        if shape is not None:
            B = shape[0]  # batch size
            HW = shape[1] # height x width
            C = shape[2]  # channels
            CHW = C*HW
        else:
            B, H, W, C = map(lambda i: i.value, tensor.get_shape())
            HW = H*W
            CHW = W*H*C

        # reshape the tensor so it is a (B, 2-dim) matrix
        # so that 'B'th gram matrix can be computed
        feats = tf.reshape(tensor, (B, HW, C))

        # leave dimension of batch as it is
        feats_T = tf.transpose(feats, perm=[0, 2, 1])
```

```
        # paper suggests to normalize gram matrix by its number of elements
        gram = tf.matmul(feats_T, feats) / CHW

        return gram
```

run_test.py

```
import tensorflow as tf
import os
import utils
import style_transfer_tester
import argparse
import time

"""parsing and configuration"""
def parse_args():
    desc = "Tensorflow implementation of 'Perceptual Losses for Real-Time Style Transfer and Super-Resolution'"
    parser = argparse.ArgumentParser(description=desc)

    parser.add_argument('--style_model', type=str, default='models/wave.ckpt', help='location for model file (*.ckpt)',
                        required=True)

    parser.add_argument('--content', type=str, default='content/female_knight.jpg',
                        help='File path of content image (notation in the paper : x)', required=True)

    parser.add_argument('--output', type=str, default='result.jpg',
                        help='File path of output image (notation in the paper : y_c)', required=True)

    parser.add_argument('--max_size', type=int, default=None, help='The maximum width or height of input images')

    return check_args(parser.parse_args())

"""checking arguments"""
```

```
def check_args(args):
    # --style_model
    try:
        #Tensorflow r0.12 requires 3 files related to *.ckpt
        assert os.path.exists(args.style_model + '.index') and
os.path.exists(args.style_model + '.meta') and os.path.exists(
            args.style_model + '.data-00000-of-00001')
    except:
        print('There is no %s'%args.style_model)
        print('Tensorflow r0.12 requires 3 files related to *.ckpt')
        print('If you want to restore any models generated from old tensorflow
versions, this assert might be ignored')
        return None

    # --content
    try:
        assert os.path.exists(args.content)
    except:
        print('There is no %s' % args.content)
        return None

    # --max_size
    try:
        if args.max_size is not None:
            assert args.max_size > 0
    except:
        print('The maximum width or height of input image must be positive')
        return None

    # --output
    dirname = os.path.dirname(args.output)
    try:
        if len(dirname) > 0:
            os.stat(dirname)
    except:
        os.mkdir(dirname)

    return args
```

```
"""main"""
def main():

    # parse arguments
    args = parse_args()
    if args is None:
        exit()

    # load content image
    content_image = utils.load_image(args.content, max_size=args.max_size)

    # open session
    soft_config = tf.ConfigProto(allow_soft_placement=True)
    soft_config.gpu_options.allow_growth = True # to deal with large image
    sess = tf.Session(config=soft_config)

    # build the graph
    transformer = style_transfer_tester.StyleTransferTester(session=sess,
                                                             model_path=args.style_model,
                                                             content_image=content_image,
                                                             )
    # execute the graph
    start_time = time.time()
    output_image = transformer.test()
    end_time = time.time()

    # save result
    utils.save_image(output_image, args.output)

    # report execution time
    shape = content_image.shape #(batch, width, height, channel)
    print('Execution time for a %d x %d image : %f msec' % (shape[0], shape[1],
1000.*float(end_time - start_time)/60))

if __name__ == '__main__':
    main()
```

style_transfer_tester.py

```
import tensorflow as tf
import transform

class StyleTransferTester:

    def __init__(self, session, content_image, model_path):
        # session
        self.sess = session

        # input images
        self.x0 = content_image

        # input model
        self.model_path = model_path

        # image transform network
        self.transform = transform.Transform()

        # build graph for style transfer
        self._build_graph()

    def _build_graph(self):

        # graph input
        self.x = tf.placeholder(tf.float32, shape=self.x0.shape, name='input')
        self.xi = tf.expand_dims(self.x, 0) # add one dim for batch

        # result image from transform-net
        self.y_hat = self.transform.net(self.xi/255.0)
        self.y_hat = tf.squeeze(self.y_hat) # remove one dim for batch
        self.y_hat = tf.clip_by_value(self.y_hat, 0., 255.)

    def test(self):

        # initialize parameters
        self.sess.run(tf.global_variables_initializer())
```

```
# load pre-trained model
saver = tf.train.Saver()
saver.restore(self.sess, self.model_path)

# get transformed image
output = self.sess.run(self.y_hat, feed_dict={self.x: self.x0})

return output
```