Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки: 01.03.02 Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент, доцент кафедры УМФ,
к.ф.-м..н.
_____/ О.Н. Ципленкова
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
профессор
_____/А.А.Замышляева
«____»_____20___ г.

Разработка системы управления проектом «Виртуальный музей писателей Южного Урала»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.02.2020.073.ПЗ ВКР

Руководитель работы,
ст. преподаватель
_____/В.А. Сурин
«____»_____2020 г.
Автор работы
Студент группы ЕТ-412
_____/ А.А. Фатеев
«____»_____2020 г.
Нормоконтролер,
ст. преподаватель
_____/Н.С. Мидоночева
«____»_____2020 г.

Челябинск
2020

АННОТАЦИЯ

Фатеев А.А. Преобразование RAW изображений в восьмибитный формат с применением нейронных сетей. – Челябинск: ЮУрГУ, ЕТ-412, 65 с., 18 ил., 1 табл, библиогр. список – 31 наим.

Целью данной работы является разработка алгоритма на основе искусственных нейронных сетей, выполняющего задачу демозаики RAWизображений, их преобразования в восьмибитный формат RGBизображения. Решается задача проектирования архитектуры свёрточной нейросети, способной эффективно выполнять демозаику.

Программная реализация данного алгоритма выполнена на языке программирования C#, используются библиотеки для реализации нейронных сетейCUDNNи CUBLAS, разработка производилась в среде MicrosoftVisualStudio.

Результаты работы программы – изображения с глубиной цвета 8 бит – являются конечным форматом, пригодны для просмотра и могут использоваться для публикации в интернете и т. д.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

RAW-формат цифровыхфотографий часто используютсякак профессиональными фотографами, так и любителями. Данный формат отличается тем, что он сохраняет большее количество информации, чем стандартные цифровые изображения с глубиной цвета 8 бит (в RAWхранится информации до 16 бит на каждый пиксель), поэтому при дальнейшей обработке в фоторедакторах можно получить результат гораздо лучше, чем при работе с данными, урезанными до восьми бит.

Однако файлы в форматеRAW не являются полноценными изображениями, которые можно считать и просмотреть напрямую с экранов электронных устройств. Это специальный формат файла, хранящий в себе необработанные значения сигналов, полученных сенсором фотокамеры.

Для получения изображения из RAW файла выполняетсядемозаика, которая преобразует специфичную структуру файла в вид восьмибитных RGBизображений (значения красного, зелёного и синего каналов в пределах от 0 до 255), пригодный для показа на экранах.

Существующие алгоритмы демозаики отличаются своей сложностью и продолжительностью работы. В своей работе я поставил задачу попробовать оптимизировать процесс демозаики путём применения нового алгоритма на основе нейронных сетей, поскольку нейронные сети хорошо справляются с задачамивыявления неявных закономерностей, решающимися в ходе демозаики.

Следовательно, результат работы нейронной сети, решающей задачу демозаики может превзойти по скорости стандартные алгоритмы и занять среднюю нишу между быстрыми алгоритмами с низким качеством и долгими алгоритмами с высоким качеством.

1 ДЕМОЗАИКА RAW ИЗОБРАЖЕНИЙ

1.1 Методы демозаики RAW изображений

1.1.1 Матрица цифровых фотокамер

Фотоматрица – это специализированная аналоговая или цифро-аналоговая интегральная микросхема, состоящая из светочувствительных элементов, называемых фотодиодами. Она является основным элементом цифровых фотоаппаратов, современных видео- и телевизионных камер, фотокамер, встроенных в мобильный телефон, камер систем видеонаблюдения и многих других устройств.

В качестве примера, на рисунке 1.1 представлена фотография фотоматрицы внутри цифрового фотоаппарата.

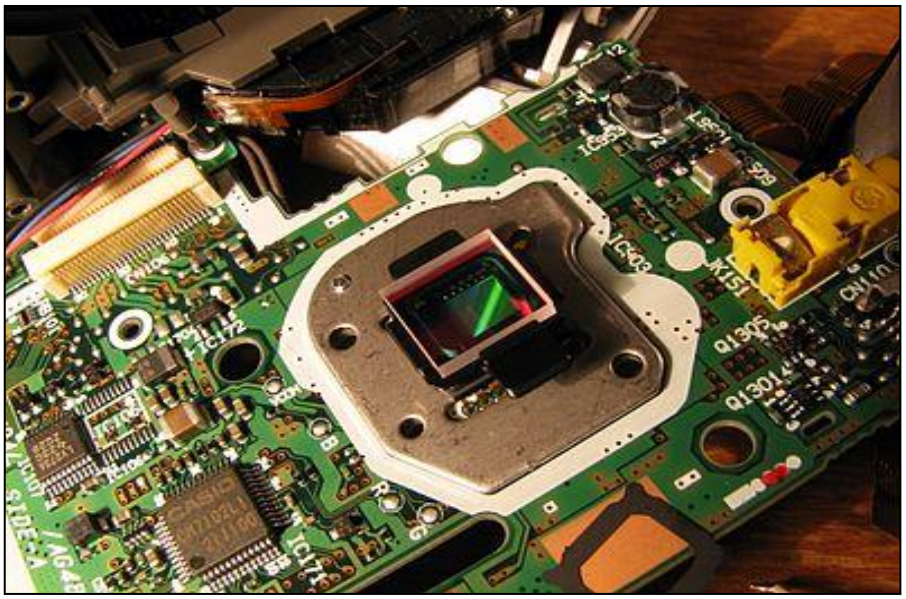

Рисунок 1.1 – Внешний вид настоящей фотоматрицы

В цифровых камерах свет проходит через систему линз и зеркал, после чего попадает на сенсор с миллионами светочувствительных элементов. На светочувствительные элементы накладывается светофильтр для получения отдельных цветов, таким образом информация об интенсивности света оцифровывается. Примерная схема этого процессапредставлена на рисунке 1.2.

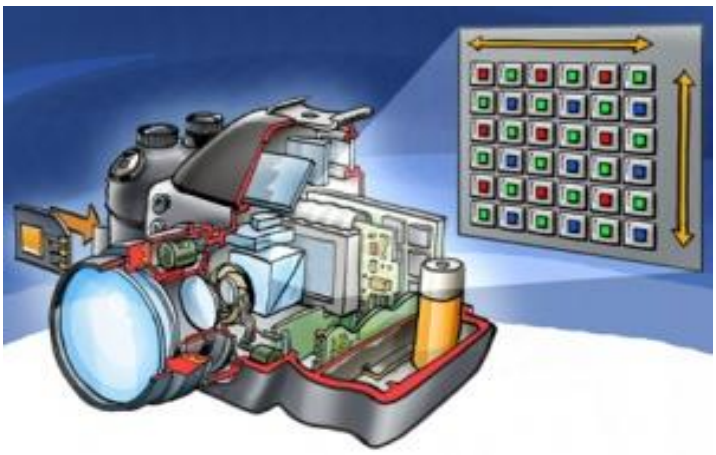

Рисунок 1.2 –Схема оцифровки
изображения внутри фотокамеры

Светочувствительные сенсоры фотоматрицы считывают количество фотонов, информация сохраняется в виде числав пределах от нуля (при условии, что не попал свет на сенсор) домаксимальной глубины цвета (наиболее яркий регистрируемый свет на сенсоре). Поскольку информации об одной лишь интенсивности (яркости) света в каждом пикселе недостаточно для построения полноценного трёхцветного изображения, применяются специальные методики.

Для получения цвета определённой длины волны – зелёный, синий, красный, – на сенсор цифровых камер накладывается специальный светофильтр. В большинстве случаев применяется фильтр Байера – сетка из повторяющихся шаблонов размером 2×2 с цветами: синий, зелёный, зелёный, красный, изображённая на рисунке 1.3.

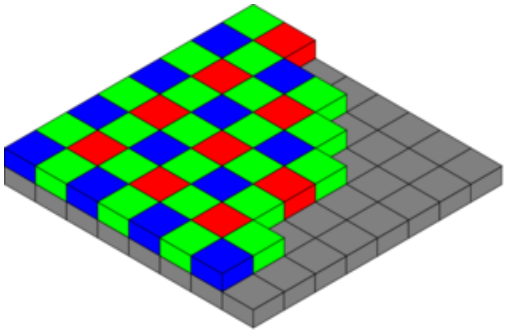

Рисунок 1.3 – Фильтр Байера

Такой фильтр позволяет сенсорам камеры получать информацию только о цвете, прошедшем через соответствующий светофильтр. Однако, информации об интенсивности света в одном цветовом канале снова недостаточно для полноцветного изображения – остальные два цветовых канала необходимо рассчитать из интенсивности света соседних сенсоров.

На рисунке 1.4 представлена схема работы светофильтров при прохождении света через них.

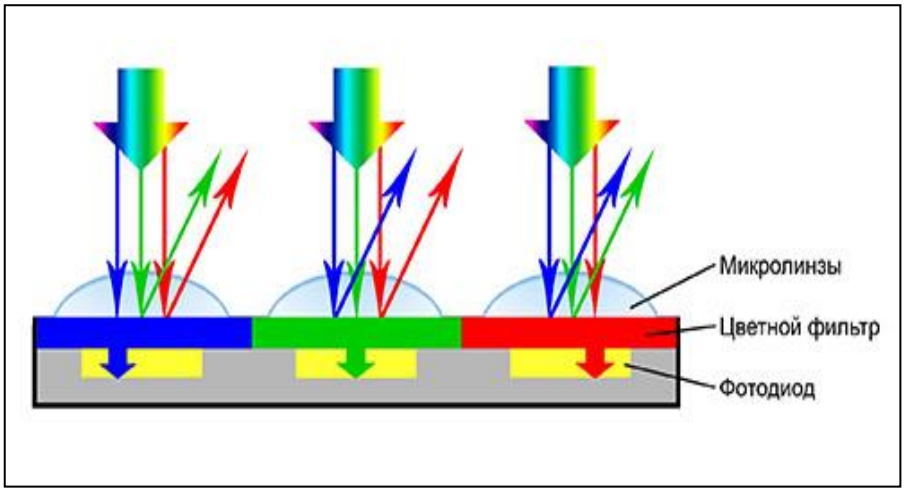

Рисунок 1.4 – Работа светофильтров

На рисунке видно, как на светофильтр попадает свет, содержащий множество волн различной длины (разного цвета), после чего отсеиваются лишние цвета и на фотодиоды попадают только волны определённой длины (цвета, соответствующего цвету светофильтра).

1.1.2 Формат RAW

Формат файла RAW (англ. сырой, необработанный) – это данные с минимальной обработкой, полученные с фотоматрицы цифровой камеры.

Данные представляют собой информацию об интенсивности света, полученной в момент открытия диафрагмы камеры. Чаще всего глубина цвета у изображений в данном формате составляет 12 или 14 бит, следовательно, цветовое пространство более обширно, нежели у изображений с глубиной цвета 8 бит.

К достоинствам данного формата можно отнести:

– широчайшие возможности в обработке изображения;

– возможность коррекции экспозиции и баланса белого после съемки;

– коррекция хроматических аберраций после съемки;

– цветокоррекция после съемки;

– эффективное усиление резкости и подавление шумов;

– полные данные об изображении.

Данные достоинства позволяют фотографам «вытянуть» изображение, снятое в неудачных условиях – к примеру, в условиях недостаточной или избыточной освещённости.

Однако у данного формата есть и недостатки:

– большой размер RAW-файла;

– RAW-файлы нельзя использовать сразу (требуется демозаика);

– требуются отдельные программы для обработки или конвертации изображения;

– из-за большого объема файлов ограничено количество кадров серийной съемки на фотокамере;

– для записи на карту памяти требуется больше времени;

– процесс обработки зачастую занимает много ресурсов ПК и времени.

Получается, что формат RAWпредставляет большую ценность для фотографов – его применение позволяет улучшить неудачные снимки, но в то же время этот формат требует ресурсоёмкой и длительной обработки – демозаики, чтобы его можно было просмотреть. Далее рассмотрим подробнее процесс демозаики и как его можно ускорить.

1.1.3 Демозаика RAWизображений

Имеются специфичные для RAW-формата проблемы:

– на данном этапе ещё отсутствует концепция цвета – каждый элемент сенсора лишь регистрирует количество фотонов, проходящих через фильтр;

– элементов, чувствительных к зелёному цвету в 2 раза больше, чем к красному и синему цвету;

– в половине (зелёные) или трёх четвертях (красные, синие) каждого цветового канала недостаёт данных.

Отображение изображения с камеры с Байеровым сенсором, таким образом, нетривиально – мозаику дискретных точек нужно конвертировать в цветовое изображение. Этот процесс называется демозаикой.

Наиболее простое решение задачи демозаики – взять информацию о трёх цветовых каналов в каждом квадрате 2×2 и объединить в один пиксель. Данный метод не требует сложных вычислений и быстр в работе, но его главным недостатком является уменьшение разрешения изображения в 2 раза по ширине и высоте – теряется 75% полезной информации, а 25% информации не используется вообще (поскольку берётся информация только у одного из двух пикселей зелёного цвета).

С целью обойти столь значительные потери полезных данных, были разработаны специальные алгоритмы демозаики. Наиболее популярные алгоритмы уже реализованы и включены в современные приложения, специализирующиеся на обработке RAWизображений.

Приложение RawTherapee является одним из популярных программных продуктов,оно предлагает на выбор множество стандартных алгоритмов демозаики со своими особенностями. Выбор алгоритма довольно сильно влияет на результатпреобразования изображения, особенно сильно это заметно при увеличении изображения. Самым заметным эффектом при выборе алгоритма является качество сохранения мелких деталей и видимость артефактов обработки (falsemaze).

Для камер с Байеровым фильтром чаще всего применяется метод под названием «AMaZE»при низком ISO, на втором месте идут алгоритмы«LMMSE»и«IGV», которыелучше себя показывают при высоком ISO.

1.1.4 Методы демозаики

Рассмотрим подробнее, какие методы демозаики доступны в приложении RawTherapee.

Имеются две категории методов: простые и Байеровские. Простые методы отличаются тем, что выполняют быструю обработку с низким качеством. Байеровские методы выполняют обработку дольше, но качество конечных изображений значительно лучше.

Рассмотрим простые методы:

– Fast – быстрый, но простой алгоритм с низким качеством, нерекомендуется к использованию;

– Mono – применяется только для монохромных камер или камер со снятым светофильтром, обработанные изображения не имеют цвета;

– None – демозаика не производится. Применяется для диагностики.

Байеровы методы:

– AMaZE (AliasingMinimizationandZipperElimination) – стандартный метод демозаики, показывающий хорошие результаты в большинстве случаев;

– LMMSE и IGV – эти алгоритмы рекомендуются при работе с сильно зашумлённымизображением с высоким ISO, в сочетании с инструментом уменьшения шума. Они предотвращают формирование артефактов и избавляют от замутнения изображения после удаления шума.

Ввиду того, что простые методы возвращают изображения с низким качеством, они используются редко. Байеровские методы обработки являются приоритетным выбором для фотографов, поскольку обработанные изображения имеют разрешение, равное исходному и сохраняют высокое качество.

Таким образом, Байеровский метод AMaZEявляется наиболее часто используемым методом демозаики. Основным недостатком этого метода, однако, является скорость работы – получение высокого

качестваизображения требует более длительной обработки. Если фотографу понадобится посмотреть в обработке большое количество RAWфотографий (порядков сотен снимков, как на фотосессии или событии с большим количеством людей), то ожидание может занять продолжительное время – вплоть до десятков минут, в зависимости от мощности используемого компьютера.

Своей задачей я поставил разработать алгоритм демозаики, который превзойдёт по скорости AMaZE, но позволит получить изображения с качеством напорядок выше, чем быстрые алгоритмы обработки. Ввиду особенностей задачи, свой алгоритм демозаики я решил разработать с применением нейросетей.

1.2 Применение нейронных сетей

1.2.1 Искусственные нейронные сети

Искусственные нейронные сети (ИНС) – это математическая модель, построенная по принципам организации и функционирования биологических нейронных сетей. Они возникли при изучении процессов, происходящих в мозге и при попытке смоделировать данные процессы.

ИНС состоит из искусственных нейронов (англ. artificial neuron), каждый из которых представляет собой упрощенную модель биологического нейрона. Искусственный нейрон принимает входные сигналы, обрабатывает их единым образом и передает результат на многие другие искусственные нейроны, как биологический нейрон.

Биологические нейроны связаны между собою аксонами, места их соединений называются синапсами. В синапсах происходит усиление или ослабление электрохимического сигнала. Связи между искусственными нейронами называются синаптическими, или просто синапсами. У синапса имеется один параметр – весовой коэффициент, в зависимости от его

значения происходит то или иное изменение информации, когда она передается от одного нейрона к другому.

Именно так входная информация обрабатывается и превращается в результат, а обучение нейронной сети основано на экспериментальном (эмпирическом) подборе такого весового коэффициента для каждого синапса, который приводит к получению требуемого результата.

Пример простейшей искусственной нейронной сети представлен на рисунке 1.5.

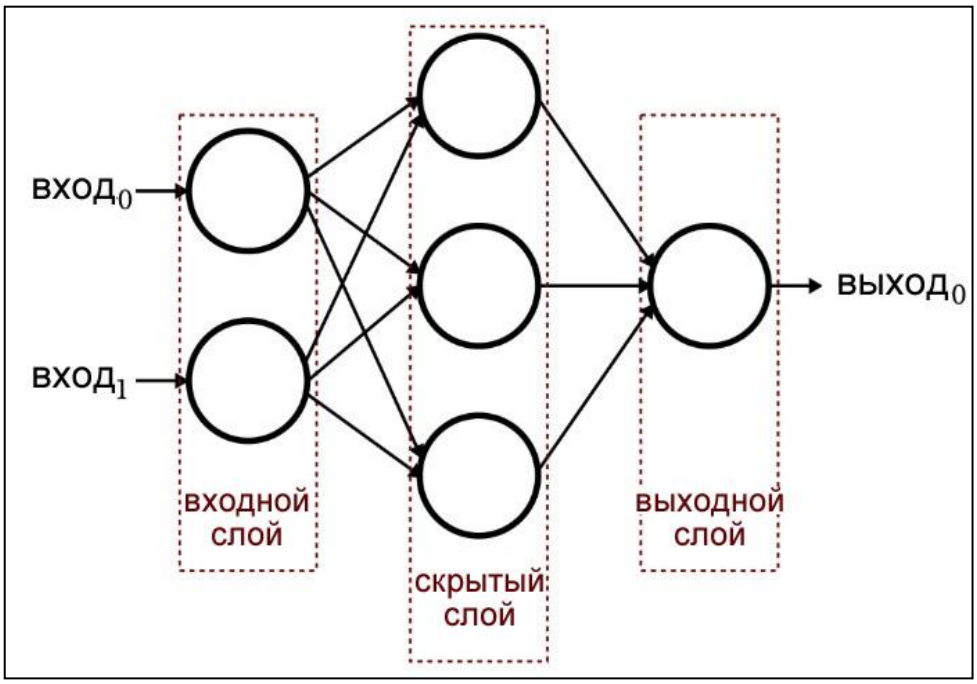

Рисунок 1.5 – Схема простейшей нейросети

Здесь показаны основные элементы нейронных сетей:

– входной слой, на который поступают сигналы;

– скрытые слои (их может не быть) – они добавляют нелинейность в алгоритм, которая приводит к более точным результатам работы нейросети;

– выходной слой, на который приходит результат работы алгоритма – в зависимости от решаемой задачи выходов может быть множество.

На каждом слое нейросети представлены математические нейроны – перцептроны (на рисунке выше они представлены окружностями). Эти нейроны хранят вектор коэффициентов-множителей (весов).

Математически, выход каждого нейрона формируется скалярным умножением вектора внутренних весов нейрона на вектор входных значений нейрона (получаем сумму поэлементно перемноженных значений векторов, отсюда название сумматор на схеме ниже), после чего результат подаётся на функцию активации.Схематично, получение выхода математического нейронаискусственной нейросети представлено на рисунке 1.6.

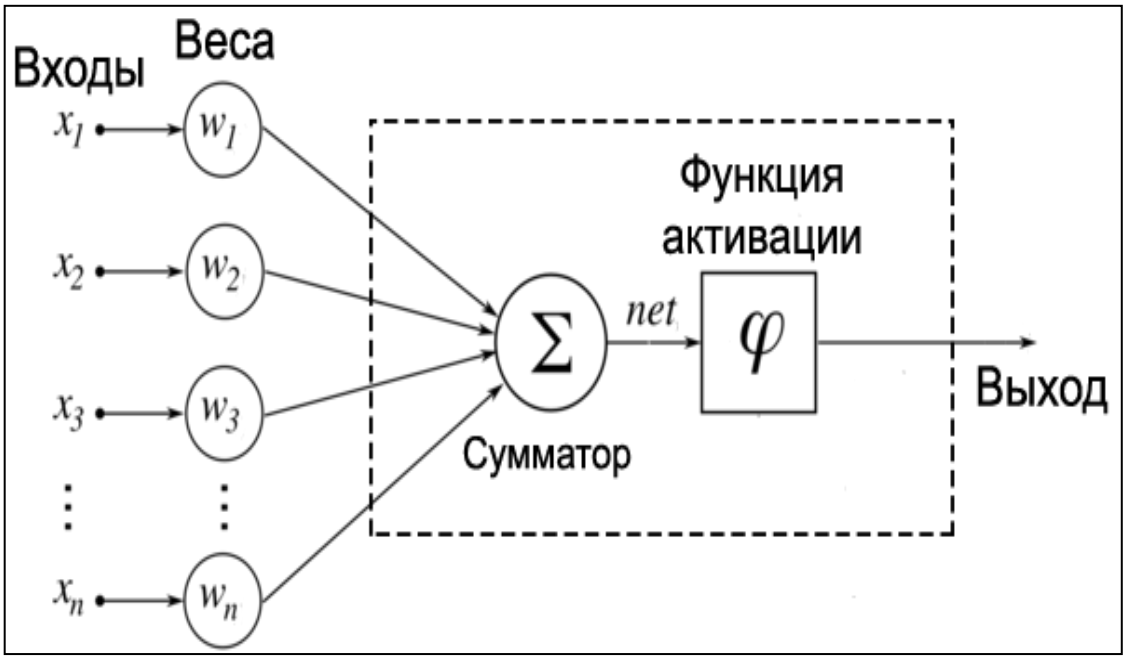

Рисунок 1.6 – Схема искусственного нейрона

При обучении нейронных сетей с учителем чаще всего применяется метод обратного распространения ошибки (англ. backpropagation). Суть этого метода заключается в вычислении ошибки (разница между ожидаемым и полученным выходом) и распространении этой ошибки по нейронам от выхода в сторону входа – т.е. в обратном направлении работы алгоритма. Распространение ошибки является модификацией метода градиентного спуска и как следствие, требует, чтобы функция активации нейрона была дифференцируема.

Метод обратного распространения ошибки является наиболее распространённым, поскольку большая часть ошибки распространяется на

нейроны, которые создают наибольшую ошибку – так обучение сети происходит быстрее.

1.2.2 Классы нейронных сетей

Рассмотрим подробнее классы нейросетей и выделим из них те особенности, которые наиболее подходят для решения моей задачи.

По типу входных данных нейросети делят на следующие классы: аналоговые (на входе действительные числа), двоичные (на входе двоичные числа) и образные (на входе знаки, иероглифы, символы) нейронные сети. Для решения моей задачи подходит сеть аналоговая, поскольку я работаю с целыми положительными числами, а не двоичными числами или текстовыми символами.

По характеру обучения есть нейросети с обучением с учителем (выходное пространство решений нейронной сети известно), обучением без учителя (выходное пространство решений формируется только на основе входных воздействий; такие сети называют самоорганизующимися); обучением с подкреплением (используется система назначения штрафов и поощрений, получаемых в результате взаимодействия ИНС со средой). В моём случае наиболее подходящей является сеть, обучающаяся с учителем, поскольку для каждого набора входных данных имеется эталонное выходное изображение, получаемое обработкой другими алгоритмами.

По характеру настройки синапсов есть сети с фиксированными связями (весовые коэффициенты нейронной сети выбираются сразу, исходя из условий задачи), сети с динамическими связями (у этих сетей в процессе обучения происходит настройка синаптических связей) – последние применяются чаще всего для вычисления неявных закономерностей. Поскольку в моей задаче также является неявным вычисляемое значение цвета по двум цветовым каналам каждого пикселя, то целесообразно применить сеть с динамическими связями.

По характерам связей есть следующие сети: сети прямого распространения (все связи направлены строго от входных нейронов к выходным), рекуррентные сети (сигнал с выходных нейронов или нейронов скрытого слоя частично передается обратно на входы нейронов входного слоя), рекуррентная сеть Хопфилда (фильтрует входные данные, возвращаясь к устойчивому состоянию и, таким образом, позволяет решать задачи сжатия данных и построения ассоциативной памяти), двунаправленные сети (между слоями существуют связи как в направлении от входного слоя к выходному, так и в обратном). Выбор характера связей не является тривиальным, подбор наиболее подходящей сети будет производиться эмпирически, на основе получаемых результатов.

1.2.3 Применение искусственных нейронных сетей

С практической точки зрения, искусственные нейронные сети представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Такие процессоры обычно довольно просты (особенно в сравнении с процессорами, используемыми в персональных компьютерах). Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые процессоры вместе способны выполнять довольно сложные задачи.

Рассматривая нейросети в разных областях науки, получаем:

– с точки зрения машинного обучения, нейронная сеть представляет собой частный случай методов распознавания образов, дискриминантного анализа, методов кластеризации и др.;

– с точки зрения математики, обучение нейронных сетей – это многопараметрическая задача нелинейной оптимизации;

– с точки зрения кибернетики, нейронная сеть используется в задачах адаптивного управления и как алгоритмы для робототехники;

– с точки зрения развития вычислительной техники и программирования, нейронная сеть – способ решения проблемы эффективного параллелизма;

– с точки зрения искусственного интеллекта, искусственная нейросеть является основой философского течения коннекционизма и основным направлением в структурном подходе по изучению возможности построения (моделирования) естественного интеллекта с помощью компьютерных алгоритмов.

Практика показала, что можно выделить определённые классы задач, успешно решаемые нейросетевыми алгоритмами. Рассмотрим их далее.

1. Распознавание образов –в качестве образов могут выступать различные объекты: символы текста, изображения, образцы звуков и т. д. В настоящее время это наиболее широкая область применения нейронных сетей. В частности, эта их способность используется в поисковиках при поиске по изображению или в камере смартфона, когда она определяет положение вашего лица и выделяет его, и во многих других приложениях.

2. Классификация – распределение данных по параметрам. Например, на вход ИНС подается набор данных о людях и нужно решить, кому можно давать кредит, а кому нет. Эту работу может выполнить нейронная сеть, анализируя такую информацию, как возраст, платежеспособность, кредитная история и т. д.

3. Принятие решений и управление – эта задача близка к задаче классификации. Классификации подлежат ситуации, характеристики которых поступают на вход нейронной сети. На выходе сети в результате должен появиться признак решения, которое она приняла. При этом в качестве входных сигналов используются различные критерии состояния управляемой системы.

4. Кластеризация – разбиение множества входных сигналов на классы, при этом ни количество, ни признаки классов заранее не известны. После обучения такая сеть способна определять, к какому классу относится входной сигнал. Сеть также может сигнализировать о том, что входной сигнал не относится ни к одному из выделенных классов – это является признаком появления новых данных, отсутствующих в обучающей выборке. Таким образом, подобная сеть может выявлять новые, неизвестные ранее классы сигналов.

5. Прогнозирование – способности нейронной сети к прогнозированию напрямую следуют из её способности к обобщению и выделению скрытых зависимостей между входными и выходными данными. После обучения сеть способна предсказать будущее значение некой последовательности на основе нескольких предыдущих значений и (или) каких-то существующих в настоящий момент факторов.

6. Аппроксимация – Нейронная сеть способна аппроксимировать любую непрерывную функцию с некоторой наперед заданной точностью.

В моей работе наиболее подходящим классом нейросети является нейросеть, решающая задачи аппроксимации, поскольку задача преобразования RAWизображений в RGBсводится к примерному вычислениюнедостающих значений двух цветовых каналов для каждого пикселя на основе имеющейся информации в самом пикселе и его соседних элементах.

1.2.4 Преимущества и недостатки применения нейросетей

Основными преимуществами нейронных сетей перед традиционными вычислительными методами являются следующие факторы.

1. Решение задач в условиях неопределенности. Благодаря способности к обучению нейронная сеть позволяет решать задачи с неизвестными закономерностями и зависимостями между входными и выходными данными, что позволяет работать с неполными данными.

2. Устойчивость к шумам во входных данных. Нейронная сеть может самостоятельно выявлять неинформативные для анализа параметры и производить их отсев, в связи с чем отпадает необходимость в предварительном анализе входных данных. В контексте изображений это означает избавление от шума, появляющегося в результате неточного получения входных данных с фотоматрицы.

3. Высокое быстродействие. При аппаратной реализации нейросети на графическом процессоре входные данные обрабатываются многими нейронами одновременно, благодаря чему нейронные сети решают задачи быстрее, чем большинство других алгоритмов.

4. Отказоустойчивость нейронных сетей. На неблагоприятное изменение условий нейросеть реагирует лишь незначительным снижением производительности. Эта особенность объясняется распределенным характером хранения информации в нейронной сети, поэтому только серьезные повреждения структуры могут существенно повлиять на работоспособность нейросети.

Данные достоинства являются ключевым фактором в пользу моего выбора нейросетей как метода решения задачи демозаики. Рассмотрим далее подробнее недостатки нейросетей, с которыми придётся работать.

1. Ответ, выдаваемый ИНС, всегда приблизительный. Нейронные сети не способны давать точные и однозначные ответы. Но задачи, в которых надо применять ИНС и одновременно получать точные ответы, встречаются довольно редко. Поскольку моя задача не требует абсолютно точного получения корректных значений (так как нет способа получить точные значения), то данный недостаток не является проблемой. Однако, результаты работы сети должны быть максимально близки к результатам работы стандартных алгоритмов, иначе не достигается сама цель – получения высокого качества при высокой скорости работы.

2. Неспособность принятия решений в несколько этапов. Нейронная сеть не может решать задачи, которые требуют последовательного

выполнения нескольких шагов; она способна решать задачу только «в один заход»,поэтому нейросеть не может, например, доказать математическую теорему. Задача демозаики не требует нескольких этапов, поэтому данный недостаток нейросетей никак не сказывается на моём решении.

3. Неспособность решать вычислительные задачи. На ИНС нельзя, например,подать математическое уравнение и получить его решения для различных параметров. Но это и не является предназначением нейронных сетей и не является проблемой для задачи демозаики.

4. Трудоемкость и длительность обучения. Для того чтобы нейронная сеть могла наиболее корректно решать поставленную задачу, требуется провести ее обучение на миллионах наборов входных данных. Но уже разработаны различные технологии ускоренного обучения, современные видеокарты позволяют обучать нейросети в сотни раз быстрее. Данный недостаток может сказаться на способности моего приложения решать задачу демозаики с получением высокого качества конечного изображения, однако можно искусственным образом увеличивать выборку во много раз, если разбивать большие фотографии на маленькие участки и использовать их в качестве выборки. Так, проблема выборки будет сведена к минимальному влиянию на конечный результат.

Стандартные алгоритмы демозаики RAWизображений, таким образом, можно заменить на новые, основанные на нейросетях. Данный подход может привести к значительному увеличению скорости преобразования с низкими потерями качества конечного изображения.

Это предоставит альтернативу для профессиональных фотографов – помимо быстрой некачественной обработки и долгой качественной, появится промежуточное звено, применяющее современные нейросетевые алгоритмы для быстрого решения задачи с получением качественного изображения в результате.

1.3 Свёрточные нейронные сети

В результате рассмотрения различных архитектур нейросетей я пришёл к выводу, что наиболее подходящей архитектурой сети является свёрточная нейронная сеть. Далее рассмотрим подробнее, что из себя представляет данная архитектура, её достоинства и недостатки и почему она лучше всего подходит для решения моей задачи.

Свёрточная нейронная сеть (СНС, на англ. –Convolutional neural network, CNN) –это специальная архитектура искусственных нейронных сетей, нацеленная на эффективное распознавание образов и в целом работу с изображениями. Особенности архитектуры свёрточных нейросетей позволяют эффективно использовать ресурсы компьютера для решения задач обработки больших изображений ввиду малого количества полносвязных слоёв.

Архитектура свёрточной нейросетей была предложена Яном Лекуном в 1988 году и опиралась на принцип работы зрительной коры, в которой были открыты простые и сложные клетки, выполняющие разные задачи. Простые клетки анализируют свет, попадающий на сетчатку глаза под разными углами, а у сложных реакция связана с активацией наборов простых клеток.

Основной причиной успеха СНС является концепция общих весов. Несмотря на свой большой размер, свёрточные 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. Свёрточные нейросети могут быстро работать на последовательной машине и быстро обучаться благодаря чистому распараллеливанию процесса свертки по каждой карте, а также обратной свертки при распространении ошибки по сети.

Основнаяидеясвёрточныхнейронныхсетейзаключаетсявчередованиисвёрточныхслоёв (англ. convolutionlayers) исубдискретизирующихслоёв (англ. subsamplinglayersилиангл. poolinglayers, слоёвподвыборки). Структура сети – однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) – любая, по выбору исследователя.

1.3.1 Свёрточные нейросети в работе с изображениями

Для решения задачи демозаики требуется работа алгоритма с изображениями. Входной файл RAWимеет достаточно большие размеры, как и выходное изображение в восьмибитном формате. Свёрточные нейросети отличаются от других архитектур нейросетей тем, что лучше всего справляются с задачами, связанными с изображениями. Далее рассмотрим подробнее их достоинства и недостатки, которые повлияют на использование данной архитектуры для решения задачи демозакики.

Преимущества свёрточных нейросетей:

– свёрточные нейросети показали себя на практике как один из лучших алгоритмов по распознаванию и классификации изображений;

– по сравнению с полносвязной нейронной сетью (типа перцептрона) свёрточные нейросети имеют гораздо меньшее количество настраиваемых весов, так как одно ядро весов используется целиком для всего изображения, вместо того, чтобы делать для каждого пикселя входного изображения свои персональные весовые коэффициенты. Это подталкивает нейросеть при обучении к обобщению демонстрируемой информации, а не попиксельному запоминанию каждой показанной картинки, как это делает перцептрон;

– удобное распараллеливание вычислений, а следовательно, возможность реализации алгоритмов работы и обучения сети на графических процессорах;

– устойчивость к различным трансформациям – таким как повороты и сдвиги – распознаваемого изображения ввиду использования слоёв свёртки;

– обучение свёрточной нейросети производится при помощи классического метода обратного распространения ошибки.

С другой стороны, к недостаткам свёрточных нейросетей можно отнести следующие факторы:

– большое количество различных варьируемых параметров сети;

– неизвестно заранее, для какой задачи и вычислительной мощности какие нужны настройки.

1.3.2 Параметры свёрточной нейросети

При рассмотрении различных задач, решаемых свёрточной нейронной сетью, стало известно, что чаще всего в реализации сети варьируются следующие параметры:

– количество чередующихся слоёв свёртки и субдискретизации;

– размерность ядра свёртки для каждого из слоёв;

– количество ядер для каждого из слоёв;

– шаг сдвига ядра при обработке слоя;

– необходимость слоёв субдискретизации;

– степень уменьшения ими размерности;

– конкретная функция уменьшения размерности субдискретизирующих слоёв (выбор максимума, среднего и т. д.) – чаще всего в этом параметре выбирается функция максимума;

– передаточная функция нейронов;

– наличие и параметры выходной полносвязной нейросети на выходе свёрточной нейронной сети.

Все эти параметры существенно влияют на результат, но выбираются исследователями и разработчиками программной реализации эмпирически. Опытный путь показал, что существует несколько выверенных и прекрасно

работающих конфигураций сетей для конкретных решённых задач, но не хватает рекомендаций, по которым нужно строить сеть для новой задачи.

1.3.3 Архитектура свёрточных нейросетей

Примерархитектурысвёрточной искусственной нейронной сети приведён на рисунке 1.7.

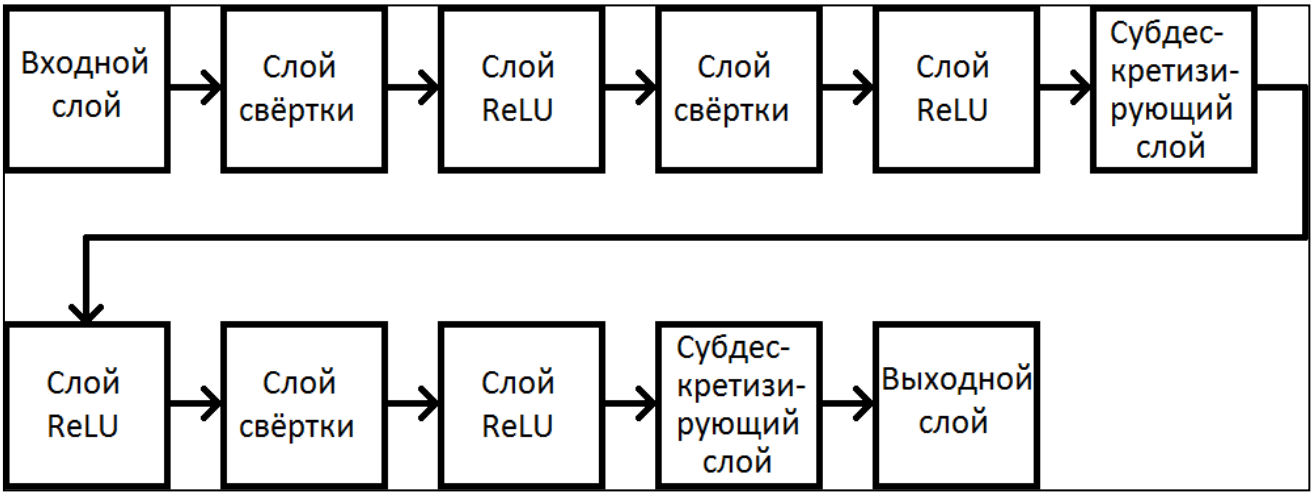

Рисунок 1.7 – Пример схемы свёрточной нейросети

Работа свёрточной нейронной сети заключается в переходе от конкретных особенностей изображения к абстрактным деталям, а после – к ещё более абстрактным деталям, вплоть до выделения понятий высокого уровня. При этом сеть самонастраивается и вырабатывает сама необходимую иерархию абстрактных признаков, фильтруя маловажные детали и выделяя существенное.

В обычном перцептроне, который представляет собой полносвязную нейронную сеть, каждый нейрон связан со всеми нейронами предыдущего слоя, причём каждая связь имеет свой персональный весовой коэффициент. В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале – непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для

различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют ядром свёртки.

1.3.4 Слои свёртки

Таким образом, отличительная особенность свёрточных нейросетей, выделяющая их на фоне других архитектур нейросетей – использование так называемых слоёв свёртки. Они эффективны для работы с изображениями, поскольку избегают запоминания конкретных изображений, как это было бы в случае с полносвязными слоями. Так же, слои свёртки значительно экономят ресурсы компьютера ввиду размеров ядра.

Слой свёртки отличается от полносвязного слоя тем, что он использует ядро небольшого размера (например, 3×3 или 5×5) и поочерёдно применяется к группе элементов входного слоя. Схематично операциясвёртки представлена на рисунке 1.8.

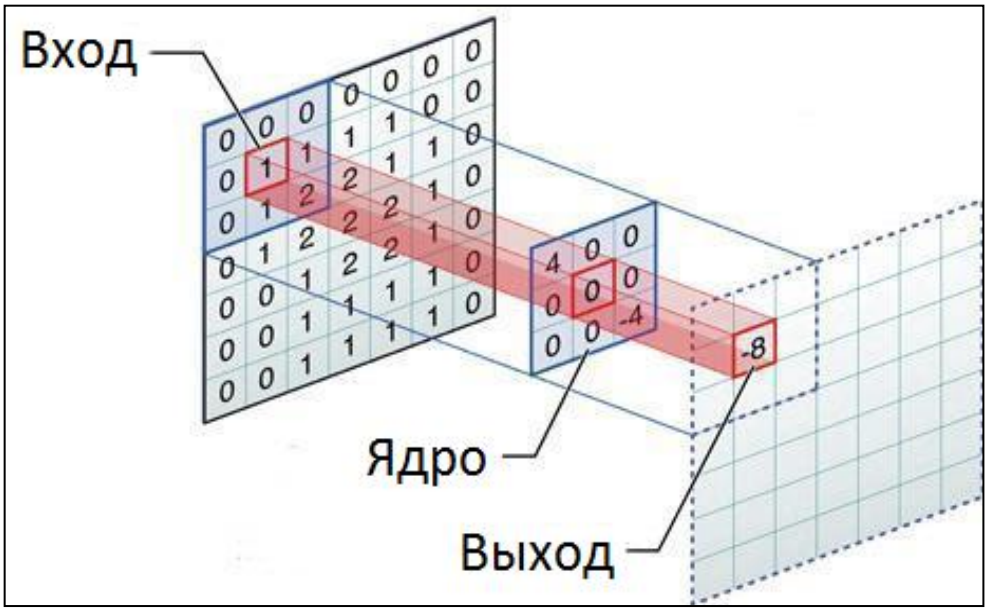

Рисунок 1.8 – Операция свёртки

Слой свёртки (англ. convolutionallayer) – это основной блок свёрточной нейронной сети. Слой свёртки включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты поэлементного произведения для каждого

фрагмента). Весовые коэффициенты ядра свёртки (небольшой матрицы) неизвестны и устанавливаются в процессе обучения.

Одной из важных особенностейсвёрточного слоя является относительно небольшое количество параметров, устанавливаемое при обучении. Например, если исходное изображение имеет размерность 100×100 пикселей по трём цветовым каналам (то есть 30000 входных нейронов), а свёрточный слой использует фильтры с ядром 3×3 пикселя с выходом на 6 каналов, то в процессе обучения определяется только 9 весов ядра, однако по всем сочетаниям каналов, то есть 9×3×6 =162, в таком случае данный слой требует нахождения только 162 параметров, что существенно меньше количества искомых параметров полносвязной нейронной сети.

Слои свёртки дают большой выигрыш в количестве весов, хранимых нейронами. Этот выигрыш сказывается в программной реализации меньшим расходом оперативной памяти компьютера (а так же, в зависимости от реализации– меньшим расходом видеопамяти), и как следствие – менее жёсткими системными требованиями. При работе с большими изображениями использование полносвязных слоёв приводит к большому расходу памяти – как следствие, требования к аппаратной части компьютера будут значительно выше.

Благодаря своим особенностям свёрточные нейронные сети чаще всего применяются для работы с изображениями.Именно поэтому для решения своей задачи я выбрал именно свёрточные сети – данный тип нейронных сетей лучше всего подходитдля решения задачи демозаики и удаления шума.

1.4 Существующие методы обработки RAW нейросетями

Обзор статьи за авторством MichaelKunz.

На сайтепредставлена англоязычная статья спримерами сравнения работы стандартных алгоритмов демозаики (не использующими нейросети) с авторским алгоритмом, применяющим свёрточные нейронные сети. При

детальном рассмотрении можно увидеть, что результаты нейросети получаются более сглаженные, что в целом положительно сказывается на качестве картинки.

Автор статьи применяет свёрточную нейронную сеть для преобразования RAW изображения в обычное цифровое изображение.

Одновременно с этим, автор поставил перед собой задачи исправления баланса белого, исправления бликов, а также преобразования цветового пространства.

Алгоритм хорошо показывает себя в деталях – к примеру, на фотографии воробьёв после применения алгоритма перья получились гладкие, незашумлённые, а на ночном фото города при детальном рассмотрении хорошо различимы детали зданий – окна, различные выступы; шума так же не наблюдается.

На рисунке1.9 представлены фотографии с большим количеством деталей, с которыми работал алгоритм. Более детально их можно посмотреть в самой статье.

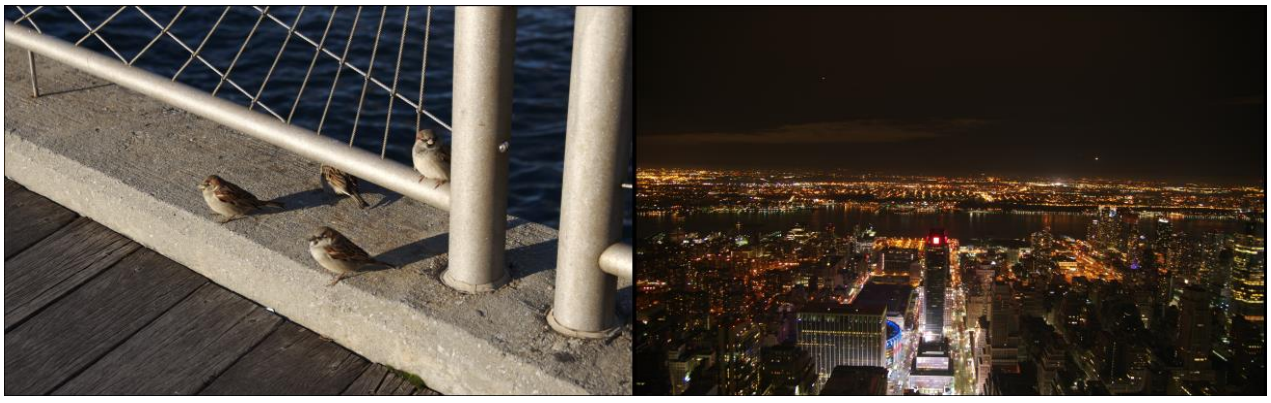

Рисунок 1.9 – Фото, обработанные нейросетями

Имеет место возможность оптимизации алгоритма – в некоторых местах сглаживание приводит к проблемам с градиентными переходами.

В своих выводах, автор статьи указал, что его собственный алгоритм демозаики, основанный на свёрточных нейросетях действительно хорошо показал себя как замена стандартным алгоритмов. Таким образом, поставленная задача действительно может быть эффективно решена при

помощи данных нейросетей. Автором статьи не была опубликована информация о скорости работы алгоритма, поэтому неприменимы сравнительные метрики при сравнении моего и его решения.

1.5 Вывод

Для преобразования RAW в восьмибитные изображения требуется определить два значения цветовых каналов для каждого пикселя. Сделать это можно, считав значения из соседних пикселей и совершив над ними преобразования. Данный процесс называется демозаикой и сравним с задачей вычисления значений некой неявной функции – задача примерного выявления закономерности входных и выходных данных может быть эффективно решена после обучения искусственной нейросети.

Существует множество стандартных алгоритмов обработки RAW. Программы, реализующие стандартные алгоритмы охватывают спектр различных задач фотографов – дополнительная обработка изображений включает в себя исправление освещённости участков изображений и других дефектов. Существуют алгоритмы с высокой скоростью работы и низким качеством (простые алгоритмы) и алгоритмы с высоким качеством и низкой скоростью (Байеровские алгоритмы).

Алгоритм, основанный на нейросетях, может показать значительные улучшения в скорости обработки изображений с незначительным ухудшением качества конечного изображения, т.е. занять нишу между быстрой и качественной обработкой.

Лучше всего для демозаики подойдут свёрточные нейронные сети, поскольку данная архитектура нейросетей лучше всего показывает себя в задачах, связанных с обработкой изображений или классификацией объектов на них.

Цель моей работы– написать приложение для преобразования RAWв восьмибитный RGB, а также разработать для него алгоритм демозаики,

основанный на нейросетях. Применение нейросетей может привести к ускорению процесса демозаики, что существенно сказывается на продолжительностипреобразования большого количества изображений. Такие изображения можно использовать для срочной публикации в интернете или для предварительного просмотра без дополнительной обработки.

2 АРХИТЕКТУРА НЕЙРОСЕТИ ДЛЯ ДЕМОЗАИКИ

2.1 Постановка задачи

Пусть $X = \{X_i\}_1^n$ – множество входных матриц (набор исходных изображений), $Y = \{Y_i\}_1^n$ – множество выходных матриц (набор преобразованных изображений), f – целевая функция, отображающая множество X на множество Y:

$$f: X \to Y.$$

Пусть $X^* \subset X$ – набор изображений обучающей выборки, тогда значения функции известны на наборе пар $(X_i^*; Y_i^*)$:

$$Y^* = f(X^*).$$

Необходимо построить алгоритм A, позволяющийопределить функциональную зависимость между изначальными изображениями и обработанными изображениями:

$$A: X^* \to Y^*.$$

2.2 Исходные данные

В качестве исходных входных данных используются фотографии в формате RAW 12-бит. Ввиду особенностей данного формата, необработанные изображения представлены массивами чиселразмером mнаn (mстрок, nстолбцов). Математически эти данные можно представить как матрицы:

$$A_{m,n} = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{pmatrix},$$

где$a_{ij} \in [0{,}4095], i = \overline{1,m}, j = \overline{1,n}$.

Каждый элемент матрицыa_{ij} содержит число от 0 до 4095. Это число соответствует значению, полученному на сенсоре фотокамеры в пикселе с координатами $(i; j)$ соответственно.

На матрицу каждого цифрового фотоаппарата наложен цветовой фильтр Байера, представленный на рисунке 2.1.

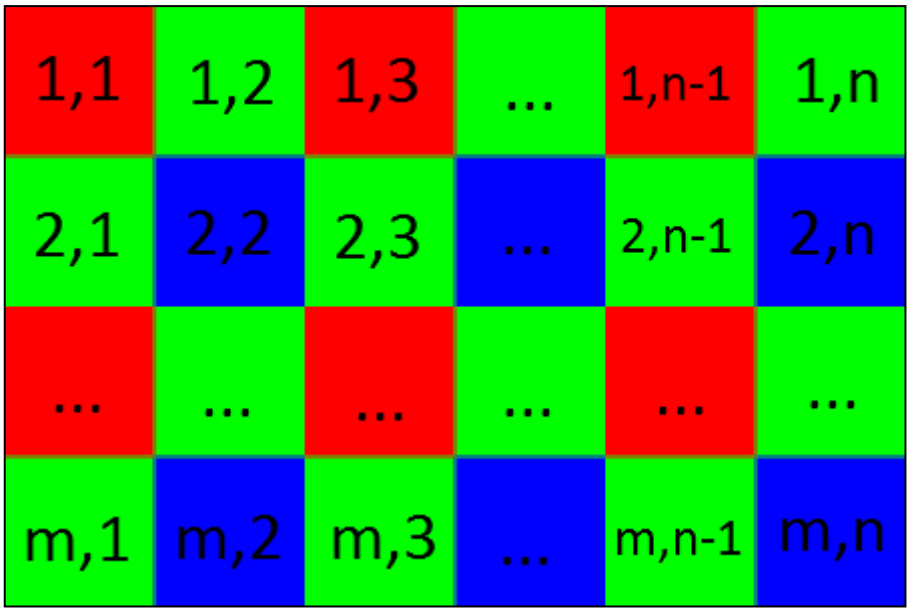

Рисунок 2.1 – Матрица Байера

Это означает, что входные массивы данных имеют неполную информацию о цвете пикселя – только значение одного цветового канала на каждый пиксель. Цвет фильтра в каждом конкретном пикселе строго определён координатами в матрице Байера.

Поскольку матрица Байера состоит целиком из множественного повторения матрицы:

$$B_0 = \begin{pmatrix} R & G \\ G & B \end{pmatrix},$$

где B_0 – фрагмент матрицы Байера, R, G, B – красный, зелёный, синий светофильтры соответственно, то определить цвет пикселя по его координатам можно, используя следующий алгоритм:

– если строка имеет нечётный номер, то при нечётном столбце элемента цвет фильтра – красный, при чётном – зелёный;

– если строка имеет чётный номер, то при нечётном столбце цвет фильтра – зелёный, при чётном – синий.

Эталонный выход нейросети (для обучения с учителем) формируется путём демозаики исходных изображений стандартными алгоритмами обработкиRAW.

2.3 Архитектура нейронной сети

Для решения задачи применяется свёрточная нейронная сеть.

Схема применяемой сети представлена на рисунке 2.2.

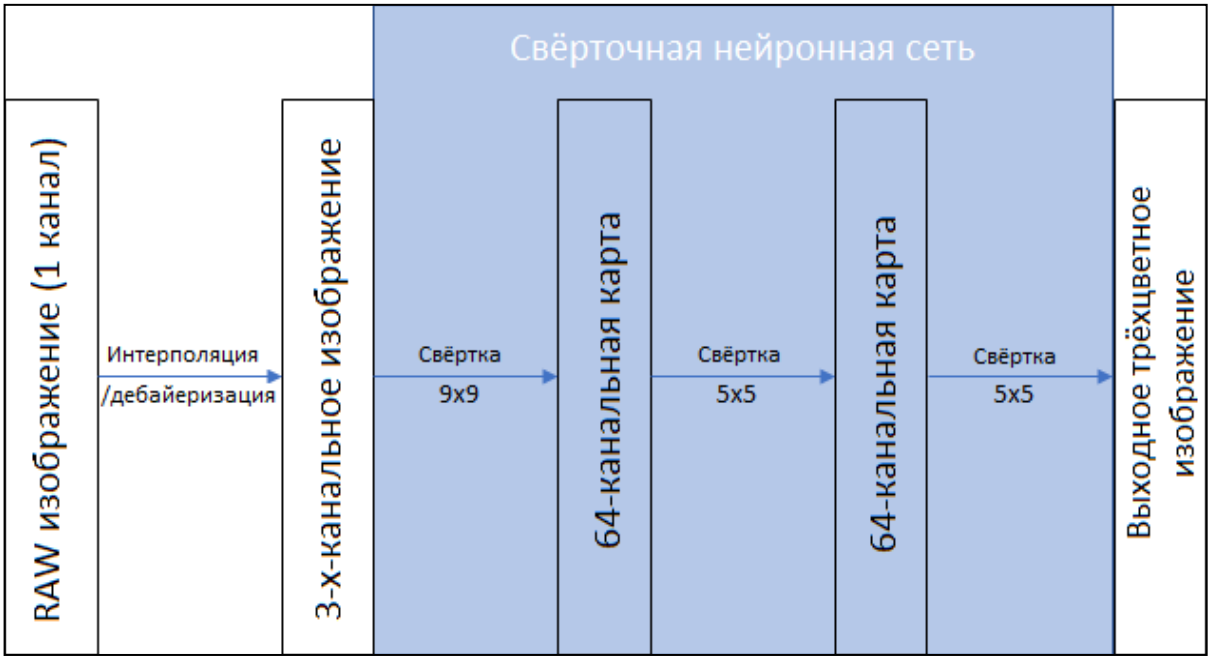

Рисунок 2.2 – Схема нейронной сети

Архитектура сети спроектирована не очень глубокой, главным образом потому, что для всего этапа демозаики и преобразования изображения в восьмибитный RGBосновной задачей является высокая скорость вывода.

Функцией активации в сверточных слоях нейронной сетивыбрана параметрическая функция активацииReLU.

2.4 Функция активации

Функция активация в алгоритме обратного распространения ошибки должна обладать несколькими важными характеристиками: непрерывностью, дифференцируемостью и, кроме этого, она должна являться монотонно

неубывающей. Также, для ради эффективности вычислений, необходимо, чтобы нахождение производной функции активации не было трудоемким. С учетом необходимых характеристик, в качестве функции активации была выбрана функция PReLU. Данная функция вычисляется следующим образом:

$$f(a,x) = \begin{cases} ax, x < 0 \\ x, x \geq 0 \end{cases}, \quad (2.1)$$

На рисунке 2.3 представлено наглядно отличие между функциями активации ReLU и PReLU.

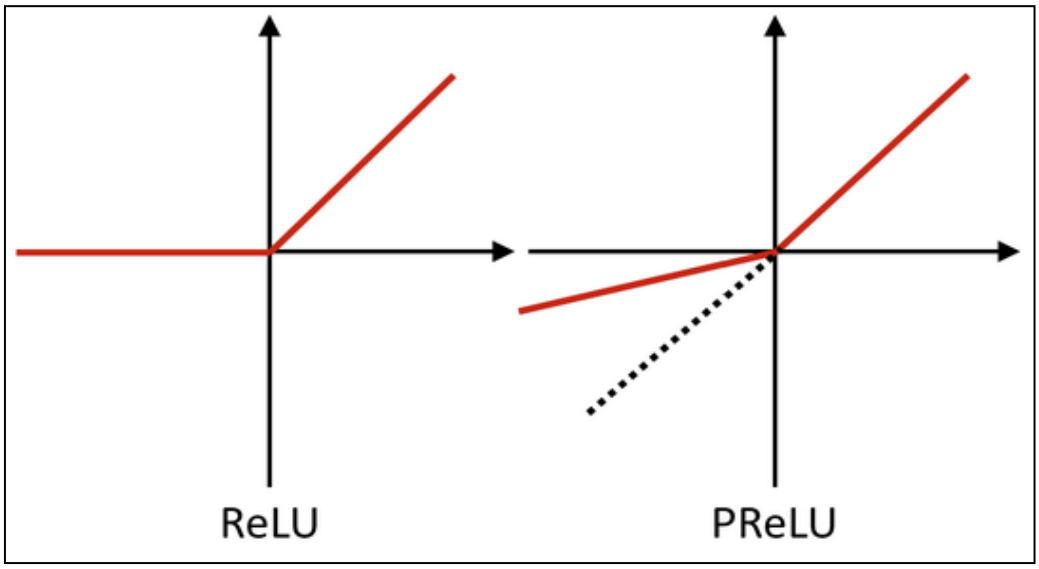

Рисунок 2.3 – Отличия функций активации ReLUи PReLU

PReLU или Parametric ReLU является логическим развитием обыкновенной активационной функции ReLU. Основное отличие между ними состоит в том, что у PReLU есть обучаемые параметры.

Функция PReLU хороша тем, что с её добавлением, количество параметров сети увеличивается незначительно: общее количество новых параметров каждого слоя всего лишь равно количеству каналов этого слоя.

Если значение параметра a мало и фиксированно, например 0.01, PReLU становится Leaky ReLU (LReLU). Основной идеей создания LReLU было устранение нулевого градиента, однако, на практике LReLU даёт ничтожно малый прирост точности, по сравнению с обычной функцией ReLU, поэтому она не была выбрана в качестве функции активации.

2.5 Подготовка данных

Исходные изображения обрабатываются стандартным алгоритмом демозаики под названием AMaZE. Входными данными этого алгоритма являются изображения в формате RAW, выходными – готовые к просмотру фотографии с глубиной цвета 8 бит.

Исходные и обработанные изображения применяются для обучения нейронной сети.

Чистые данные RAW не сбалансированы по белому и приведены в специфичном для устройства основном линейном цветовом пространстве RGB, в основном определяемом цветными фильтрами на датчике. Для демозаики по краям применяются настройки баланса белого, сохраненные в файле RAW так же, как работает настройка «Как снимок» в Adobe Camera RAW, после дебайеризации цвета масштабируются до исходного разрешения, чтобы гарантировать, что уровни шума для красного и синего каналов не усиливаются.

Таким образом, свёрточная нейронная сеть обучается только на не сбалансированных по белому цвету изображениях. Тем не менее, сеть также хорошо подходит для хорошо сбалансированных тестовых изображений в нелинейном цветовом пространстве sRGB, но иногда в результатах виден небольшой цветовой оттенок.

Для увеличения тренировочной выборки нейронной сети изображения разбиваются на фрагментыменьшего размера.Размер одного фрагмента изображения для обучения составляет 31x31 пикселей.

Ядро MS-SSIM представляет собой приблизительную метрику полной ошибки: ошибка вычисляется только для центрального пикселя, а не как среднее значение всего фрагмента для ускорения вычислений.

Чтобы избежать смещения к цвету Байеровской матрицывнутри центрального пикселя, объединяются фрагменты 2x2 в групповойфрагмент, который используется для первого этапа демозаики. При этом все четыре

пикселя паттерна Байера (RGGB) являются центральными пикселями такого сгруппированного фрагмента.

В качестве входных данных эти сгруппированные фрагментысохраняются в виде тайла размером 66x66 пикселей, так как 2 пикселя теряются на каждой границе при демозаике. Пример такого фрагмента представлен на рисунке 2.4.

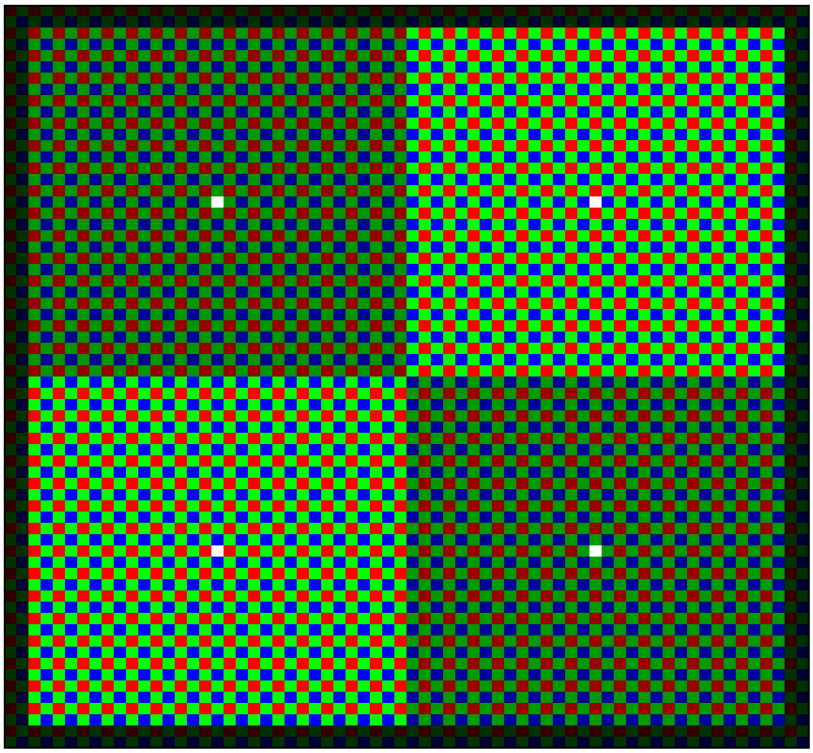

Рисунок 2.4 – Фрагмент входных данных

Обычно зеленый канал – это первый канал, который насыщается при чрезмерной экспозиции датчика, что приводит к розовому цвету в насыщенных областях изображения. В этом случае зеленый канал затем восстанавливается из информации в красном и синем канале.

2.6 Обучение нейросети

При обучении нейросети используются исходные изображения в формате RAW, в качестве эталонного выхода подаются изображения, полученные в результате обработки изображений стандартным алгоритмомAMaZE.

Для обучения используется метод обратного распространения ошибки с использованием градиентного спуска. В основе этого метода лежит использование выходной ошибки нейронной сети для вычисления величин коррекции весов нейронов в её скрытых слоях.

Алгоритм является итеративным и использует принцип обучения «по эпохам», при котором изменение весов производится после подачи на вход нейронной сети нескольких экземпляров обучающего множества, а ошибка усредняется по всем примерам.

$$E = \frac{1}{n}\sum_{i=1}^{n} loss_i, \tag{2.2}$$

где n – количество объектов в одной итерации обучения;

$loss_i$ – значение функции потерь для -го объекта выборки;

E – усредненная ошибка на одной итерации.

На каждой итерации происходит несколько прямых проходов нейронной сети и один обратный.

На прямом проходе входной вектор распространяется от входов сети к ее выходам и формирует некоторый выходной вектор, соответствующий текущему состоянию весов. Затем вычисляется ошибка нейронной сети, как разность между фактическим и целевым значениями.

На обратном проходе эта ошибка распространяется от выхода сети к ее входам, и производится коррекция весов нейронов в соответствии с формулой:

$$\Delta w_{ij} = -\mu \frac{\partial E}{\partial w_{ij}}, \tag{2.3}$$

где w_{ij} – вес -ойсвязиj-тогонейрона;

μ – параметрскоростиобучения.

Данный способ обучения, в сравнении собучением по определённому шагу, является более устойчивым к выбросам и аномальным значениям целевой переменной за счет усреднения ошибки по многим примерам.

Поскольку алгоритм распространяет ошибку так, что наибольшее изменение получают веса, создающие наибольшую ошибку, то достигается наиболее высокая скорость обучения для используемой архитектуры нейронной сети.

Общий алгоритм обратного распространения ошибки можно описать следующим образом:

1) инициализировать веса, количество итераций I;

2) повторять для каждой итерации:

а) прямой ход для каждого слоя:

$$y_i = f_i\left(\sum_{j \in Output} w_{ji}x_j + w_{0i}\right), \tag{2.4}$$

где w_{ji} – весовой коэффициент связи -го и i-го нейронов;

x_j – сигнал с -го нейрона предыдущего слоя;

w_{0i} – сигнал нейрона -го смещения текущего слоя;

$Output$ – множество нейронов предыдущего слоя;

f_i – функция активации -го нейрона текущего слоя;

б) вычисление ошибки выходного слоя:

$$\delta_{Mi} = E, \tag{2.5}$$

где $loss$ – усредненная функция потерь;

δ_{Mi} – ошибка сигнала с -го нейрона выходного слоя M;

в) обратный ход:

$$\delta_{ki} = \sum_{j \in Output_{k+1}} w_{ji}\delta_{k+1j}, \tag{2.6}$$

где δ_{k+1j} – ошибка сигнала -го нейрона слоя $k + 1$;

δ_{kj} – ошибка сигнала -го нейрона слоя k;

w_{ji} – весовой коэффициент связи -го и i-го нейронов;

$Output_{k+1}$– множество нейронов слоя $k + 1$;

г) градиентный шаг для каждого слоя:

$$w_{ji} = w_{ji} + \mu\delta_i f_i'(y_i), \quad (2.7)$$

где w_{ji} – весовой коэффициент связи -го и i-го нейронов;

μ – скорость обучения;

δ_i – ошибка -го нейрона предыдущего слоя;

y_i – сигнал -го нейрона предыдущего слоя.

2.7 Вывод

В своём методе решения я применяю свёрточную нейронную сеть.

Подготовлена обучающая и тестовая выборка для обучения нейросети. В качестве входных данных подаются массивы, считанные из фотографий в формате RAW, обучающий вывод представляет собой массивы, считанные из изображений восьмибитного формата RGB, полученных путём обработки изображений стандартными алгоритмами.

Для быстрого обучения нейросети используется алгоритм обратного распространения ошибки, описанный выше.Выбранная функция активации нейросети – PReLU.

3 ПРОГРАММА ДЛЯ ПРЕОБРАЗОВАНИЯ RAW НЕЙРОСЕТЯМИ

3.1 Блок-схема программы

Общий алгоритм работы разработанного приложения представлен на рисунке 3.1.

Рисунок 3.1 – Блок-схема работы приложения

3.2 Сравнение качества работы алгоритмов

Важным фактором моего алгоритма является то, что качество выходного изображения должно получиться достаточно высоким.

Наглядно увидеть результаты работы программы можно, посмотрев на рисунок 3.2. Здесь представлено сравнение изображения, полученного при обработке стандартным алгоритмом «Fast» и изображения, полученного при обработке нейронной сетью.

Рисунок 3.2 – Сравнение Fast и нейросети

Особенно хорошо видны дефектыобработки Fastна градиентах – в фотографиях их часто можно увидеть на небе. На рисунке можно увидеть это отличие между Fast-демозаикойи нейросетевой: переход цвета получается плавным, шум незаметен при сильном приближении изображения.

Рассмотрим далее тот же фрагмент изображения, обработанного алгоритмом AMaZEи нейросетью. Сравнение представлено на рисунке 3.4.

Ещё одно сравнение работы алгоритмов для фото, снятого в помещении, представлено на рисунке 3.4. Здесь представлены результаты обработки алгоритмом Fast, нейросетью и AMaZE. Результат работы нейросети по качеству в данном случае находится между двумя другими алгоритмами – лучше Fast, но хуже AMaZE.

Рисунок 3.3 – Сравнение AMaZE и нейросети

Рисунок 3.4 – Сравнение трёх алгоритмовна фото в помещении

Ещё один пример обработки изображения разными алгоритмами представлен на рисунке 3.5. Данное фото было снято на улице в условиях дневной освещённости. Результаты работы всех алгоритмов близки, при близком рассмотрении сохраняется тенденция – Fast, нейросеть, AMaZE идут в порядке улучшения качества обработанного изображения.

При сравнении алгоритма с нейросетями с её учителем – алгоритмом AMaZE –видны небольшие отличия в работе, более долгий алгоритм AMaZEвсё ещё является выигрышным по качеству изображения.

Тем не менее, можно считать, что алгоритм на основе нейронной сети не сильно уступает в качестве конечного изображения алгоритму AMaZE – следовательно, разработанный алгоритм является конкурентоспособным.На

всех изображениях видно, что алгоритм на основе нейросети превосходит по качеству алгоритм Fast.

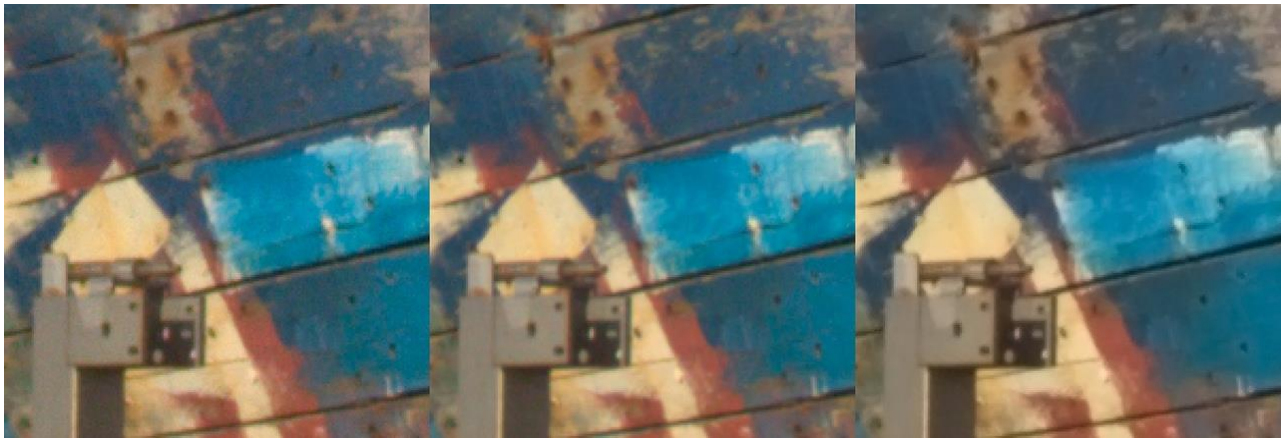

Рисунок 3.5 – Сравнение трёх алгоритмовна фото при дневном освещении

Далее сравним скорость преобразования изображений стандартных алгоритмов с разработанным алгоритмом на основе нейронной сети и составим сравнительные метрики.

3.3 Скорость преобразования (сравнительные метрики)

Следующим этапом сравнения данных алгоритмов возьмём скорость преобразования RAW изображения в выходное восьмибитное изображение.

Было проведено тестирование разработанного приложения на 10 тестовых изображениях. Измерена скорость преобразования алгоритмом с применением свёрточной нейронной сети и скорость преобразования стандартным алгоритмом Fast, а также Байеровским алгоритмом AMaZE.

По результатам данныхтестовбыли составлены сравнительные метрики, представленные в таблице 3.1. В первом столбце указан номер изображения, в трёх остальных – время в секундах, потраченное на преобразование соответствующего изображения.

Полученные данные используем для вычисления среднего времени работы алгоритмов для тестовых изображений:

$$T_{avg} = \frac{1}{n}\sum_{i=1}^{n} t_i. \quad (3.1)$$

Таблица 3.1 – Метрики обработки изображений

№	Fast	Нейросеть	AMaZE
1	0.27	0.86	1.42
2	0.32	0.78	1.40
3	0.30	0.92	1.36
4	0.21	0.79	1.47
5	0.17	0.95	1.56
6	0.19	0.93	1.52
7	0.19	0.87	1.54
8	0.20	1.01	1.48
9	0.25	0.94	1.55
10	0.30	0.87	1.53

Получаем следующие усреднённые значения:

– время работы алгоритма Fast – 0.24с;

– время работы нейросети – 0.892с;

– время работы алгоритма AMaZE – 1.483с.

Таким образом, прирост в скорости алгоритма на основе нейросетей относительно алгоритма AMaZEсоставляет примерно 66%, но алгоритм всё ещё проигрывает в скорости алгоритму Fast.

3.4 Вывод

Программа, реализующая алгоритм демозаики на основе нейронной сети действительно занимает промежуточное место между алгоритмами Fastи AMaZE.

В отношении качества разработанный алгоритм показывает себя лучше, чем Fast, но хуже, чем AMaZE, в то время как по скорости он превосходит AMaZE, но уступает Fast.

Разработанное приложение может использоваться для преобразования RAWизображений в восьмибитные. Преимуществом разработанного алгоритма является скорость – он работает примерно на 66% быстрее, чем реализация алгоритма AMaZEв приложении RawTherapee.

4 ЗАКЛЮЧЕНИЕ

В данной работе было проведено исследование RAW формата изображений, был изучен принцип демозаикии разработано приложение, реализующее демозаику при помощи нейронных сетей. В настоящее время не существует идеальных алгоритмов преобразования изображений в формат RGB, поэтомудо сих пор возможно добиться лучших результатов обработки с точки зрения улучшения качества выходного изображения или скорости работы алгоритма.

Применение свёрточных нейронных сетей позволило разработанному алгоритмудостичь приемлемых результатов по скорости работы.

Оптимизация алгоритма преобразования имеет практическую пользу – фотографам, применяющим приложение с таким алгоритмом, потребуется меньше времени и вмешательства для преобразования большого количества фотографий в пригодный формат для публикации в интернет и просмотра на экранах электронных устройств – таких как телефоны и компьютеры.

Разработанное приложение может быть улучшено в дальнейшем применением другой архитектуры нейронной сети или увеличением размера обучающей выборки.

5 БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Алгоритмы демозаики RawThrapee[Электронныйресурс] – URL: https://instantframe.ru/ programmnoe-obespechenie/demozaiki-rawthrapee.html (дата обращения: 20.03.2020)

2 Вьюнин, В.В. Математические основы теории машинного обучения и прогнозирования / В.В. Вьюнин. – М.: МЦНМО, 2013. – 390 с.

3 Использование проявщика RAWTherapee в прикладных задачах фотолюбителя [Электронныйресурс] – URL: https://habr.com/ru/post/216465 (дата обращения: 13.04.2020)

4 Магистерская диссертация «Реставрация поврежденных и зашумленных изображений» Богославец А.И., 2018г[Электронный ресурс].– СПГУ – URL:https://dspace.spbu.ru/bitstream/11701/12146/1/diplom.pdf (дата обращения: 25.04.2020)

5 Нейронная сеть – URL: https://ru.wikipedia.org/wiki/Искусственная_нейронная_сеть (дата обращения: 25.02.2020)

6 Свёрточная нейронная сеть[Электронныйресурс] – URL: https://ru.wikipedia.org/ wiki/Свёрточная_нейронная_сеть (дата обращения: 12.03.2020)

7 Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / П. Флах; пер. с англ. А.А. Слинкин. – М.: ДМК, 2016. – 400 с.

8 Kriesel, D.A. Brief Introduction to Neural Networks / D. Kriesel. – 244 p.

9 Alleysson, D., Linear Minimum Mean Square Error Demosaicking / D. Alleysson, B. Chaix de Lavarène, S. Süsstrunk, J. Hérault// CRC Press, Sep. 2008, – Pp. 213–237.

10 Alleysson, D., Linear demosaicing inspired by the human visual system / D. Alleysson, S. Süsstrunk, J. Hérault// IEEE Transactions on Image Processing 14 (4), Apr. 2005.– Pp. 439–449.

11 Astola, J., Vector median filters / J.Astola,P. Haavisto,Y. Neuvo, Apr. 1990 // Proceedings of the IEEE 78 (4). – Pp 678–689.

12 Deep Learning (Adaptive Computation and Machine Learning series) / Ian Goodfellow // The MIT Press, 2016 – 800 p.

13 Demosaicing – RawPedia – URL: https://rawpedia.rawtherapee.com/Demosaicing (датаобращения: 20.03.2020)

14 Eskicioglu, A.Image quality measures and their performance / Eskicioglu, A.M., Fisher, P. S., Dec. 1995. // IEEE Transactions on Communications 43 (12). – Pp. 2959–2965.

15 Faugeras, O.D., Digital color image processing within the framework of a human visual model / O.D. Faugeras // IEEE Transactions on Acoustics, Speech, and Signal Processing 27 (4), Aug. 1979. – Pp 380–393.

16 Freeman, W.T. Median filter for reconstructing missing color samples / W.T. Freeman // U.S. patent 4,724,395, to Polaroid Co., Patent and Trademark Office, Washington D.C.

17 Henrik, B. Machine Learning / B. Henrik, R. Joseph, M. Fetherolf. – New-York: Manning Publication Co, 2016. – 264 p.

18 Hirakawa, K., Sep. 2008. Color filter array image analysis for joint denoising and demosaicking. In: Lukac, R. (Ed.), Single-Sensor Imaging: Methods and Applications for Digital Cameras. CRC Press. – Pp. 239–261.

19 Kunz, M. Joint demosaicing and denoising of RAW images with a CNN / M. Kunz [Электронныйресурс]– URL: https://kunzmi.github.io/NNDemosaicAndDenoise (датаобращения: 20.04.2020)

20 Laroche, C.A. Apparatus and method for adaptively interpolating a full color image utilizing chrominance gradients. / C.A. Laroche, M.A. Prescott // U.S. patent 5,373,322, to Eastman Kodak Co., Patent and Trademark Office, Washington D.C., Jun. 1993

21 Leitão, J.A. Content-adaptive video up-scaling for high definition displays. / J.A.Leitão, M.Zhao, G.de Haan //In: Proceedings of the SPIE

Conference on Image and Video Communications and Processing (IVCP’03).Santa Clara, California, U.S.A.Jan. 2003. –Pp. 612–622.

22 Lukac, R. Normalized color-ratio modeling for CFA interpolation / R. Lukac, K.N. Plataniotis // IEEE Transactions on Consumer Electronics 50 (2). – Pp. 737–745.

23 Lukac, R.A.Normalized model for color-ratio based demosaicking schemes / R. Lukac, K.N.Plataniotis //In : Proceedings of the 11th International Conference on Image Processing (ICIP’04). Singapore. – Pp. 1657–1660.

24 Lukac, R. Color filter arrays : Design and performance analysis / R. Lukac, K.N.Plataniotis // IEEE Transactions on Consumer Electronics 51 (4). – Pp. 1260–1267.

25 Lukac, R. Universal demosaicking for imaging pipelines with an RGB color filter array. Pattern Recognition 38 / R. Lukac, K.N.Plataniotis. – 2208–2212 p.

26 Marsland, S. Machine Learning: An Algorithmic Perspective / S. Marsland. – Florida: CRC, 2015. – 390 p.

27 Simon, S. Haykin Neural Networks and Learning Machines 3rd ed / S. Simon // Pearson, 2008. – 936 p.

28 Rojas, R.Neural Networks: A Systematic Introduction 1st ed / R. Rojas // Springer Berlin Heidelberg, 1996 – 522 p.

29 Raw (формат изображения)[Электронныйресурс] – URL: https://ru.wikipedia.org/ wiki/Raw_(формат_изображения) (дата обращения: 15.02.2020)

30 RGB (аддитивная цветовая модель)[Электронныйресурс] – URL: https://ru.wikipedia.org/ wiki/RGB (дата обращения: 15.02.2020).

ПРИЛОЖЕНИЕ 1

Код программы обучения нейросети

```
usingSystem;
usingSystem.Collections.Generic;
usingSystem.IO;
usingSystem.Globalization;
usingSystem.Runtime.InteropServices;
usingSystem.Drawing;
using System.Drawing.Imaging;
using ManagedCuda;
using ManagedCuda.BasicTypes;
using ManagedCuda.VectorTypes;
using ManagedCuda.CudaBlas;
using ManagedCuda.CudaDNN;
using ManagedCuda.NPP;
using KernelClasses;
using NeuralNetworkTraining;

namespace TrainNetwork
{
class Program
   {
static double learning_rate = 0.005;
static int deviceID = 0;
static string ISO = "100";
static bool crosscheck = false;
static bool saveImages = false;
static int warmStart = 0;

static float3[][] baOriginal;
static float[][] baRAW;

static List<string> fileRawList = new List<string>();
```

```
static List<string> fileTrouthList = new List<string>();

static DeBayerGreenKernel deBayerGreenKernel;
static DeBayerRedBlueKernel deBayerRedBlueKernel;
static PrepareDataKernel prepareDataKernel;
static RestoreImageKernel restoreImageKernel;
static CudaContext ctx;

private static float[] ReadRAWFloat(string filename)
      {
         FileStream fs = new FileStream(filename, FileMode.Open, FileAccess.Read);
         BinaryReader bw = new BinaryReader(fs);
int dimX = bw.ReadInt32();
int dimY = bw.ReadInt32();
int c = bw.ReadInt32();
if (c != 1)
throw new FileLoadException("This is not a one channel image!");

float[] data = new float[dimX * dimY];

unsafe
         {
fixed (float* ptr = data)
            {
byte[] buffer = bw.ReadBytes(dimX * dimY * sizeof(float));
Marshal.Copy(buffer, 0, (IntPtr)ptr, dimX * dimY * sizeof(float));
            }
         }
fs.Close();
bw.Dispose();
fs.Dispose();
return data;
      }

private static float3[] ReadRAWFloat3(string filename)
```

```
        {
            FileStream fs = new FileStream(filename, FileMode.Open, FileAccess.Read);
            BinaryReader bw = new BinaryReader(fs);
int dimX = bw.ReadInt32();
int dimY = bw.ReadInt32();
int c = bw.ReadInt32();
if (c != 3)
throw new FileLoadException("This is not a three channel image!");

float3[] data = new float3[dimX * dimY];

unsafe
            {
fixed (float3* ptr = data)
                {
byte[] buffer = bw.ReadBytes(dimX * dimY * (int)float3.SizeOf);
Marshal.Copy(buffer, 0, (IntPtr)ptr, dimX * dimY * (int)float3.SizeOf);
                }
            }
fs.Close();
bw.Dispose();
fs.Dispose();
return data;
        }

static float GetLearningRate(int iteration)
        {
double lr_gamma = 0.00001;
double lr_power = 0.75;
return (float)(learning_rate * Math.Pow((1.0 + lr_gamma * iteration), (-lr_power)));
        }

static void Main(string[] args)
        {
            //Read CL arguments
```

```
for (int i = 0; i < args.Length; i++)
        {
if (args[i] == "-d")
            {
deviceID = int.Parse(args[++i]);
            }
if (args[i] == "-lr")
            {
               learning_rate = double.Parse(args[++i],
System.Globalization.NumberStyles.AllowDecimalPoint, CultureInfo.InvariantCulture);
            }
if (args[i] == "-iso")
            {
               ISO = args[++i];
            }
if (args[i] == "-t")
            {
crosscheck = true;
            }
if (args[i] == "-w")
            {
warmStart = int.Parse(args[++i]);
Console.WriteLine("Start with epoch " + warmStart);
            }
if (args[i] == "-s")
            {
saveImages = true;
            }
        }

Console.WriteLine("Using device ID: " + deviceID);
Console.WriteLine("Learning rate: " + learning_rate);

        //Init Cuda stuff
ctx = new PrimaryContext(deviceID);
```

```
ctx.SetCurrent();
Console.WriteLine("Context created");
        CUmodule modPatch = ctx.LoadModulePTX("PatchProcessing.ptx");
Console.WriteLine("modPatch loaded");
        CUmodule modBorder = ctx.LoadModulePTX("BorderTreatment.ptx");
Console.WriteLine("modBorder loaded");
        CUmodule modError = ctx.LoadModulePTX("ErrorComputation.ptx");
Console.WriteLine("modError loaded");
        CUmodule modPRelu = ctx.LoadModulePTX("PRelu.ptx");
Console.WriteLine("modPRelu loaded");
        CUmodule modDeBayer = ctx.LoadModulePTX("DeBayer.ptx");
Console.WriteLine("all modules loaded");
deBayerGreenKernel = new DeBayerGreenKernel(modDeBayer, ctx);
deBayerRedBlueKernel = new DeBayerRedBlueKernel(modDeBayer, ctx);
        //Both deBayer kernels are load from the same module: setting the constant variable for
bayer pattern one is enough...
        deBayerGreenKernel.BayerPattern = new BayerColor[] { BayerColor.Red,
BayerColor.Green, BayerColor.Green, BayerColor.Blue };

prepareDataKernel = new PrepareDataKernel(modPatch, ctx);
restoreImageKernel = new RestoreImageKernel(modPatch, ctx);
Console.WriteLine("kernels loaded");


int countOwn = 468083;
int count5k = 33408;


string fileBase = @"/ssd/data/TrainingsDataNN/";



        List<float3> WhiteBalanceFactors = new List<float3>();
        FileStream fs1 = new FileStream(fileBase + "FromOwnDataset/WhiteBalancesOwn.txt",
FileMode.Open, FileAccess.Read);
```

```
        FileStream fs2 = new FileStream(fileBase + "From5kDataset/WhiteBalances5k.txt",
FileMode.Open, FileAccess.Read);
        StreamReader sr1 = new StreamReader(fs1);
        StreamReader sr2 = new StreamReader(fs2);

for (int i = 0; i < countOwn; i++)
        {
fileRawList.Add(fileBase + "FromOwnDataset/ISO" + ISO + "/img_" + i.ToString("0000000")
+ ".bin");
fileTrouthList.Add(fileBase + "FromOwnDataset/GroundTruth/img_" + i.ToString("0000000")
+ ".bin");

string line = sr1.ReadLine();
string[] values = line.Split('\t');
float3 wb = new float3(float.Parse(values[1],
System.Globalization.NumberStyles.AllowDecimalPoint, CultureInfo.InvariantCulture),
float.Parse(values[2], System.Globalization.NumberStyles.AllowDecimalPoint,
CultureInfo.InvariantCulture),
float.Parse(values[3], System.Globalization.NumberStyles.AllowDecimalPoint,
CultureInfo.InvariantCulture));

WhiteBalanceFactors.Add(wb);
        }
for (int i = 0; i < count5k; i++)
        {
fileRawList.Add(fileBase + "From5kDataset/ISO" + ISO + "/img_" + i.ToString("0000000") +
".bin");
fileTrouthList.Add(fileBase + "From5kDataset/GroundTruth/img_" + i.ToString("0000000") +
".bin");

string line = sr2.ReadLine();
string[] values = line.Split('\t');
float3 wb = new float3(float.Parse(values[1],
System.Globalization.NumberStyles.AllowDecimalPoint, CultureInfo.InvariantCulture),
```

```
float.Parse(values[2], System.Globalization.NumberStyles.AllowDecimalPoint,
CultureInfo.InvariantCulture),
float.Parse(values[3], System.Globalization.NumberStyles.AllowDecimalPoint,
CultureInfo.InvariantCulture));

WhiteBalanceFactors.Add(wb);
        }
sr2.Close();
sr1.Close();

baOriginal = new float3[countOwn + count5k][];
baRAW = new float[countOwn + count5k][];

        Random rand = new Random(0);

        //random order for the image patches
for (int i = 0; i < countOwn + count5k - 1; i++)
        {
int r = i + (rand.Next() % (countOwn + count5k - i));
string temp = fileRawList[i];
fileRawList[i] = fileRawList[r];
fileRawList[r] = temp;

temp = fileTrouthList[i];
fileTrouthList[i] = fileTrouthList[r];
fileTrouthList[r] = temp;

float3 tempf = WhiteBalanceFactors[i];
            WhiteBalanceFactors[i] = WhiteBalanceFactors[r];
            WhiteBalanceFactors[r] = tempf;
        }


Console.WriteLine("Initialization done!");
```

```
int trainingSize = (int)((countOwn + count5k) * 0.9f); //4 patches per file
int testSize = fileRawList.Count - trainingSize;

        CudaBlas blas = new CudaBlas(PointerMode.Host);
        CudaDNNContext cudnn = new CudaDNNContext();

int patchSize = 31;
int patchSize4 = 66; //Size of an 2x2 patch read from file
int batch = 64;
float normalization = 0.5f;

        //define neural network:
        StartLayer start = new StartLayer(patchSize, patchSize, 3, batch);
        FinalLayer final = new FinalLayer(patchSize, patchSize, 3, batch, FinalLayer.Norm.Mix,
ctx, modError);
        ConvolutionalLayer conv1 = new ConvolutionalLayer(patchSize, patchSize, 3,
patchSize, patchSize, 64, batch, 9, 9, ConvolutionalLayer.Activation.PRelu, blas, cudnn, ctx,
modBorder, modPRelu);
        ConvolutionalLayer conv2 = new ConvolutionalLayer(patchSize, patchSize, 64,
patchSize, patchSize, 64, batch, 5, 5, ConvolutionalLayer.Activation.PRelu, blas, cudnn, ctx,
modBorder, modPRelu);
        ConvolutionalLayer conv3 = new ConvolutionalLayer(patchSize, patchSize, 64,
patchSize, patchSize, 3, batch, 5, 5, ConvolutionalLayer.Activation.None, blas, cudnn, ctx,
modBorder, modPRelu);
start.ConnectFollowingLayer(conv1);
conv1.ConnectFollowingLayer(conv2);
conv2.ConnectFollowingLayer(conv3);
conv3.ConnectFollowingLayer(final);

        CudaDeviceVariable<float3> imgA = new CudaDeviceVariable<float3>(patchSize4 *
patchSize4);
        CudaDeviceVariable<float3> imgB = new CudaDeviceVariable<float3>(patchSize4 *
patchSize4);
        CudaDeviceVariable<float> rawd = new CudaDeviceVariable<float>(patchSize4 *
patchSize4);
```

```
        CudaDeviceVariable<float> inputImgs = new CudaDeviceVariable<float>(patchSize *
patchSize * 3 * batch);
        CudaDeviceVariable<float> groundTrouth = new CudaDeviceVariable<float>(patchSize
* patchSize * 3 * batch);
        NPPImage_8uC3 imgU3a = new NPPImage_8uC3(patchSize, patchSize);
        NPPImage_8uC3 imgU3b = new NPPImage_8uC3(patchSize, patchSize);
        NPPImage_8uC3 imgU3c = new NPPImage_8uC3(patchSize, patchSize);

        Bitmap a = new Bitmap(patchSize, patchSize, PixelFormat.Format24bppRgb);
        Bitmap b = new Bitmap(patchSize, patchSize, PixelFormat.Format24bppRgb);
        Bitmap c = new Bitmap(patchSize, patchSize, PixelFormat.Format24bppRgb);

        Random randImageOutput = new Random(0);
        Random randForInit = new Random(0);
start.InitRandomWeight(randForInit);
conv1.SetActivation(0.1f);
conv2.SetActivation(0.1f);

int startEpoch = warmStart;

        FileStream fs;
        //restore network in case of warm start:
if (warmStart > 0)
        {
fs = new FileStream("epoch_" + learning_rate.ToString(CultureInfo.InvariantCulture) + "_" +
ISO + "_" + (warmStart - 1) + ".cnn", FileMode.Open, FileAccess.Read);
start.RestoreValues(fs);
fs.Close();
fs.Dispose();
        }

        //validate results on validation data set
if (crosscheck)
        {
```

```
            FileStream csvResult = new FileStream("results_" +
learning_rate.ToString(CultureInfo.InvariantCulture) + "_" + ISO + ".csv", FileMode.Append,
FileAccess.Write);
            StreamWriter sw = new StreamWriter(csvResult);

sw.WriteLine("L1;L2;Mix;Filename");
for (int i = 0; i < 2000; i += 1)
            {
string filename = "epoch_" + learning_rate.ToString(CultureInfo.InvariantCulture) + "_" + ISO
+ "_" + i + ".cnn";
try
                {
                   FileStream cnn = new FileStream(filename, FileMode.Open, FileAccess.Read);
start.RestoreValues(cnn);
cnn.Close();
cnn.Dispose();

                }
catch (Exception)
                {
Console.WriteLine("Skipping: " + i);
continue;
                }

doubleerrorL1 = 0;
doubleerrorL2 = 0;
doubleerrorMix = 0;
for (int iter = 0; iter < testSize / batch * 4; iter++)
                {
                   //Prepare batch for training:
for (int ba = 0; ba < batch / 4; ba++)
                   {
int idx = iter * (batch / 4) + ba + trainingSize;
```

```
float3[] original;
float[] raw;
if (baRAW[idx - trainingSize] == null)
                    {
original = ReadRAWFloat3(fileTrouthList[idx]);
raw = ReadRAWFloat(fileRawList[idx]);
baOriginal[idx - trainingSize] = original;
baRAW[idx - trainingSize] = raw;
                    }
else
                    {
original = baOriginal[idx - trainingSize];
raw = baRAW[idx - trainingSize];
                    }

rawd.CopyToDevice(raw);
imgA.CopyToDevice(original);

deBayerGreenKernel.RunSafe(rawd, imgB, patchSize4, new float3(0, 0, 0),
WhiteBalanceFactors[idx]);
deBayerRedBlueKernel.RunSafe(rawd, imgB, patchSize4, new float3(0, 0, 0),
WhiteBalanceFactors[idx]);
prepareDataKernel.RunSafe(imgA, imgB, groundTrouth, inputImgs, ba, normalization,
WhiteBalanceFactors[idx]);
                }

start.SetData(inputImgs);
final.SetGroundTrouth(groundTrouth);

floaterr = start.InferenceTraining(inputImgs);

errorMix += err;
errorL1 += final.GetError(FinalLayer.Norm.L1);
errorL2 += final.GetError(FinalLayer.Norm.L2);
```

```
                }
Console.WriteLine("Results for: " + filename);
Console.WriteLine("Mean Error L1: " + errorL1 / testSize * batch / 4);
Console.WriteLine("Mean Error L2: " + errorL2 / testSize * batch / 4);
Console.WriteLine("Mean Error Mix: " + errorMix / testSize * batch / 4);
sw.Write((errorL1 / testSize * batch / 4).ToString().Replace(".", ","));
sw.Write(";");
sw.Write((errorL2 / testSize * batch / 4).ToString().Replace(".", ","));
sw.Write(";");
sw.Write((errorMix / testSize * batch / 4).ToString().Replace(".", ","));
sw.Write(";");
sw.WriteLine(filename);
sw.Flush();
            }
sw.Close();
csvResult.Close();
csvResult.Dispose();
        }
        //or train existing network:
else
        {
doubleerror = 0;
doubleerrorEpoch = 0;
for (int epoch = startEpoch; epoch < 2000; epoch++)
            {
errorEpoch = 0;
error = 0;

for (int iter = 0; iter < trainingSize / batch * 4; iter++)
                {
                   //Prepare batch for training:
for (int ba = 0; ba < batch / 4; ba++)
                    {
int idx = iter * (batch / 4) + ba;
```

```
float3[] original;
float[] raw;
if (baRAW[idx] == null)
                        {
original = ReadRAWFloat3(fileTrouthList[idx]);
raw = ReadRAWFloat(fileRawList[idx]);
baOriginal[idx] = original;
baRAW[idx] = raw;
                        }
else
                        {
original = baOriginal[idx];
raw = baRAW[idx];
                        }

rawd.CopyToDevice(raw);
imgA.CopyToDevice(original);

deBayerGreenKernel.RunSafe(rawd, imgB, patchSize4, new float3(0, 0, 0),
WhiteBalanceFactors[idx]);
deBayerRedBlueKernel.RunSafe(rawd, imgB, patchSize4, new float3(0, 0, 0),
WhiteBalanceFactors[idx]);
prepareDataKernel.RunSafe(imgA, imgB, groundTrouth, inputImgs, ba, normalization,
WhiteBalanceFactors[idx]);
                    }

start.SetData(inputImgs);
final.SetGroundTrouth(groundTrouth);

floaterr = start.InferenceTraining(inputImgs);

final.BackPropagation(groundTrouth);

start.UpdateWeights(GetLearningRate(epoch * (trainingSize) / batch * 4 + iter));//*0+951342
```

```
error += err;
errorEpoch += err;
if ((epoch * trainingSize / batch * 4 + iter) % 1000 == 0 && iter != 0)
                {
                    FileStream status = new FileStream("status_" +
learning_rate.ToString(CultureInfo.InvariantCulture) + "_" + ISO + ".csv", FileMode.Append,
FileAccess.Write);
                    StreamWriter sw = new StreamWriter(status);

sw.WriteLine((error / 1000.0).ToString().Replace(".", ",") + ";" + GetLearningRate(epoch *
trainingSize / batch * 4 + iter).ToString().Replace(".", ","));

sw.Close();
status.Close();
status.Dispose();
error = 0;
                }

                //if ((epoch * trainingSize / batch * 4 + iter) % 10000 == 0)
                //{
                //    fs = new FileStream("iter_" +
learning_rate.ToString(CultureInfo.InvariantCulture) + "_" + ISO + "_" + (epoch * trainingSize /
batch * 4 + iter) + ".cnn", FileMode.Create, FileAccess.Write);
                //    start.SaveValues(fs);
                //    fs.Close();
                //    fs.Dispose();
                //    Console.WriteLine("Network saved for iteration " + (epoch * trainingSize /
batch * 4 + iter) + "!");
                //}

Console.WriteLine("Epoch: " + epoch + " Iteration: " + (epoch * trainingSize / batch * 4 + iter) +
", Error: " + err);

if (saveImages && iter == 0)//(epoch * trainingSize / batch * 4 + iter) % 10000 == 0 &&
```

```
                {
for (int i = 0; i < 1; i++)
                    {
int imgidx = randImageOutput.Next(batch);
float3 wb = WhiteBalanceFactors[iter * (batch / 4) + imgidx / 4];
restoreImageKernel.RunSafe(groundTrouth, imgU3a, imgidx, wb.x, wb.y, wb.z, normalization);
restoreImageKernel.RunSafe(inputImgs, imgU3b, imgidx, wb.x, wb.y, wb.z, normalization);
                    CudaDeviceVariable<float> res = final.GetResult();
restoreImageKernel.RunSafe(res, imgU3c, imgidx, wb.x, wb.y, wb.z, normalization);

imgU3a.CopyToHost(a);
imgU3b.CopyToHost(b);
                    imgU3c.CopyToHost(c);

a.Save("GroundTrouth_" + learning_rate.ToString(CultureInfo.InvariantCulture) + "_" + ISO +
"_" + epoch + "_" + imgidx + ".png");// * trainingSize / batch * 4 + iter
b.Save("Input_" + learning_rate.ToString(CultureInfo.InvariantCulture) + "_" + ISO + "_" +
epoch + "_" + imgidx + ".png");
c.Save("Result_" + learning_rate.ToString(CultureInfo.InvariantCulture) + "_" + ISO + "_" +
epoch + "_" + imgidx + ".png");
                    }
                }
            }
errorEpoch /= trainingSize / batch * 4;
fs = new FileStream("errorEpoch_" + learning_rate.ToString(CultureInfo.InvariantCulture) + "_"
+ ISO + ".csv", FileMode.Append, FileAccess.Write);
            StreamWriter sw2 = new StreamWriter(fs);
sw2.WriteLine(errorEpoch.ToString().Replace(".", ","));
sw2.Close();
fs.Close();
fs.Dispose();

fs = new FileStream("epoch_" + learning_rate.ToString(CultureInfo.InvariantCulture) + "_" +
ISO + "_" + epoch + ".cnn", FileMode.Create, FileAccess.Write);
start.SaveValues(fs);
```

```
fs.Close();
fs.Dispose();
                }
            }
        }
    }
}
```