МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БАШКИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
КАФЕДРА ПРОГРАММИРОВАНИЯ И ЭКОНОМИЧЕСКОЙ ИНФОРМАТИКИ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПО ПРОГРАММЕ БАКАЛАВРИАТА

БОРИСОВ НИКИТА ЕВГЕНЬЕВИЧ

ВОССТАНОВЛЕНИЕ ИЗОБРАЖЕНИЯ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

Выполнил:
Студент 4 курса очной формы обучения
Направление подготовки
прикладная математика и информатика
Направленность (профиль)
системное программирование и
компьютерные технологии

Допущено к защите в ГЭК и проверено на
объем заимствования:

Заведующий кафедрой
Юлмухаметов Ринад Салаватович
д.ф.-м.н., профессор

/ Р.С. Юлмухаметов

« » 20 г.

Руководитель
к ф.-м. н., доцент

/ К.В. Трунов

УФА – 2022

Содержание

Введение

В наше время области науки связанные с распознаванием и обработкой изображений стремительно развиваются. Вместе с развитием технологий происходит заметный рост количества данных, хранимых в фото и видео форматах. С такими типами данных, а также со средствами, которые позволяют проводить над ними различные операции (хранение, изменение, обработка), приходится сталкиваться и работать большому количеству людей практически во всех сферах деятельности – охранные организации, СМИ, медицина и т.д.

В ходе создания фотографий и видеозаписи картинка может получиться искажённой. На это может влиять множество факторов, таких как, плохое освещение, помехи при передаче сигнала из-за неисправности оборудования, неподходящие погодные условия, низкие характеристики аппаратуры, которая использовалась в ходе съёмки. Поэтому проблема восстановления изображения является актуальной, и привлекает к себе много внимания. Наиболее актуальное направление в восстановлении изображений – это подавление шумов и затемнений.

Целью данной выпускной квалификационной работы является создание нейронной сети, способной решить задачу восстановления изображения.

За последнее десятилетие было создано множество алгоритмов для цифровой обработки изображения. Создание методов шумоподавления строится на основе вероятностных моделей изображения или шума, и поиска различных статистических критериев оптимальности. Улучшение изображения возможно путём устранения шумов, повышения резкости изображения или его осветление что позволит упростить поиск этих критериев. Именно благодаря этому удалось создать такое разнообразие способов фильтрации изображения.

В данной выпускной квалификационной работе было принято решение использовать менее исследованный метод восстановления изображения, такой как применение нейронной сети.

Предметом исследования является знакомство с пакетами необходимыми для написания и обучения нейронной сети.

Объектом исследования является процесс написания нейронной сети для восстановления изображения.

Структурно выпускная квалификационная работа будет состоять из введения и трёх глав.

В первой главе будет рассматриваться постановка задачи.

Вторая глава – теоретическая. В ней изучается история, принципы работы, разновидности нейронных сетей.

Третья глава – практическая. В ней рассматривается выбор нейронной сети для поставленной задачи, а также написание и тестирование программы на языке Python.

1 Постановка задачи

В современных реалиях человечество активно пытается внедрить искусственный интеллект в повседневную жизнь. Это показывает, что разработка нейронных сетей привлекает к себе с каждым годом всё больше внимания. Они используются в различных сферах деятельности, в том числе их можно задействовать и для реставрации изображения.

Одной из лидирующих качеств нейросети можно выделить её обучаемость. В процессе обучения она тренируется определять закономерности среди множества данных, которые поступают к ней на вход, а также объединяет и структурирует полученную информацию. Именно поэтому нейросеть способна вернуть правильный ответ даже для тех данных, которые не участвовали в её обучении.

Целью этой выпускной квалификационной работы является восстановление изображения с помощью нейронных сетей.

Для выполнения поставленной цели нужно решить следующий перечень задач:

Изучить историю и выработать представление о функционировании нейросетей;

Определиться с типом нейросети, наиболее пригодной для решения поставленной задачи;

Исследовать и подобрать библиотеки для создания нейросети;

Построить и протестировать нейронную сеть, оценить результаты её работы.

2 Теоретическая часть

2.1 История развития

Веками человечество было заинтересовано вопросом, каким образом работает человеческий мозг. Вместе с технологическим прогрессом появилась возможность создания нейросетей.

Рассмотрим области наук, благодаря которым стало возможным появление искусственного интеллекта.

В истории сложилось так что первые вопросы касаемые мышления человека зародились ещё у древних философов. Намного позднее выдвинулась теория, о том, что полезные вычисления можно совершать не только с помощью человеческого мозга, но и с помощью машин.

В начале XX века появилась теория логического позитивизма, говорившая что с помощью логических теорий возможно описать любую информацию и опыт, что соответствовало бы сенсорным данным.

Позднее философы Р. Карнап и К. Хемпель попытались разобраться, каким именно происходит процесс обучения на основе опыта. Эти мысли и стали первым шагом для появления идеи, что мышление – это вычислительный процесс.

Для описания теорий, сформулированных философами, было необходимо развитие другой научной сферы, такой как математика. Именно благодаря таким разделам математики как, алгебра логики, теория вероятности и исчисление, стало возможно появление и развитие искусственных нейронных сетей.

Основателем математической логики является Джордж Буль, создавший общий способ записи логических высказываний, который в дальнейшем стал основой для булевой алгебры.

А. Тьюрингу продемонстрировать какие из функций возможно вычислить.

Огромным шагом для развития НС стало возникновение теории вероятности. Впервые понятие вероятности было описано Д. Кардано. Он

использовал её для записи определений событий с несколькими исходами, появляющиеся в азартных играх. Некоторое время спустя было подставлено правило для обновления вероятностей в случае возникновения новых данных. Теорема Байеса стала фундаментом для множества актуальных подходов к рассуждениям учитывая неясности в системах ИИ.

Природа мышления изучалась долгое время, но особых результатов на практике достигнуто не было. Объясняется это тем, что исследования проводившиеся в области физики, математики и других наук, не приносили результатов при анализе человеческого мозга.

Главный скачок в развитии нейронных сетей произошёл в двадцатом столетии вместе с развитием таких наук как психология, нейроанатомия и нейрофизиология. В это же время благодаря объединению достижений в разных научных сферах, появилось такое направление как искусственный интеллект.

В 60-х годах XX века нейробиологи установили, что человеческий мозг состоит из миллиардов нейронов, соединённых между собой. Это послужило причиной для разработки искусственных нейросетей, ведь теперь появилась возможность строить математические модели работы нейронов и их связей.

Первые искусственные НС были выполнены как электронные схемы. Позднее, с появлением новых технологий, их начали программировать на компьютерах. [10]

В развитии нейросетей выделяются три этапа:

Первый этап – это появление интереса к нейросетям.

В 1943 году был описан термин нейросети и показана её модель в виде электросхем.

В 1949 году Д. Хебб первым представил алгоритм обучения нейронной сети.

В 1949 году учёный Ф. Розенблаттон изобрёл однослойный перцептрон.

В это время учёные думали, что для создания аналога мозга человека необходимо только построить нейронную сеть большого размера.

Второй этап – это спад интереса к нейронным сетям.

Как выяснилось, однослойные нейросети в теории не способны решить большое количество простых задач, к примеру с помощью них нельзя организовать функцию «исключающее ИЛИ».

В 1969 году М. Минский опубликовал работу, в которой он доказывает, что «увеличение размера персептрона не приводит к улучшению способности решения сложных задач».

Именно этот труд стал причиной потери интереса к нейросетям на долгое время.

Третий этап – это возвращение интереса.

Учёные Гроссберг, Андерсен, Конохен смогли создать внушительную теоретическую базу, благодаря которой стало возможным создание глубоких многослойных нейросетей. Но стоял вопрос с методами их обучаемости.

В 1974 году появился метод «обратного распространения ошибки». Благодаря ему удалось превзойти ограничение, которое Минский описывал в своих работах.

На практике этот метод показал хорошие результаты, однако позднее выяснилось, что он не универсален. Проблемой было долгое время обучения, а в некоторых случаях существовала вероятность что нейросеть вообще не обучится. Причиной стало паралич сети и попадание в локальных минимум.

В 1975 году изобрели когнитрон – сеть на основе самоорганизации. Архитектурой он похож на строение зрительный коры, и использовался он для распознавания образов. Это многослойная искусственная нейросеть которая обучалась без учителя. [12]

В 1980 году Кунихикой Фукусимой был разработан неокогнитрон. Это улучшенная версия когнитрона, способная распознавать образы в независимости от вращений, искажений, преобразований и изменений масштаба. [11]

В 1982 году благодаря Д. Хопфилду появились полносвязные нейронные сети, которые содержали в себе один слой. От количества нейронов зависело

количество входов и выходов. Нейросеть была способна решать задачи, связанные с оптимизацией, но у нее были и недостатки. Однако Хопфилду удалось заложить фундамент для развития рекуррентных сетей.

В попытках обойти ограничения времени для сетей обратного распространения Р. Хехт-Нильсон в 1987 году создал сети встречного распространения ошибки. Они могли обучатся в несколько раз быстрее чем сети обратного распространения. [13]

Ещё одной проблемой искусственных нейросетей с которой пришлось столкнуться учёным был вопрос стабильности-пластичности, ведь когда сеть определяет новый признак она может исказить уже натренированные веса, а из-за стабилизации, новые признаки игнорируются, что влияет на обучаемость сети.

Решением этой проблемы стала адаптивная резонансная теория, которая предлагала делить всю информацию на классы. У каждого класса будет свой образ. Нейросеть проверяет принадлежность новых данных к какому-то из существующих классов. Если таких данные не уникальны, они используются для улучшения классов, в противном случае создаётся новый класс.

Позднее были разработаны стохастические методы. Благодаря этому удалось решить проблему попадания в локальный минимум. Принцип их работы заключался в том, что коэффициенты весов немного изменялись случайным образом и запоминались только те, что улучшали полученный результат. [5]

В наше время существует много разновидностей нейросетей, и методов для их обучения. Каждая и них имеет свои характерные признаки и проскоблена для решения конкретного круга задач. Так же существует множество библиотек, которые предоставляют удобный инструментарий для работы с нейронными сетями. К примеру, самыми популярными фреймворками для работы с нейросетями являются: TensorFlow, PyTorch, Keras, XGBoost, Darknet. [9]

2.2 Принцип работы нейронной сети

Нейросети работают точно так же, как биологические нейроны в мозге человека. Часто отвечая на вопрос «Почему же нейросети работают?» можно услышать, что каждый из нейронов находящийся в сети способен обрабатывать поступающие сигналы, а так как внутри нейросети миллиарды нейронов, то поступающая информация на выходе становится правильным ответом. Однако все нейроны функционируют по одному и тому же принципу. Но если это так, то почему результат работы нейросетей из раза в раз отличается? Это объясняется тем, что кроме нейронов, существуют ещё и синапсы. Синапс – это точка соединения связанных друг с другом нейронов. Он имеет свойство усиливать или ослаблять сигнал, который через него проходит (см. рис. 2.1).

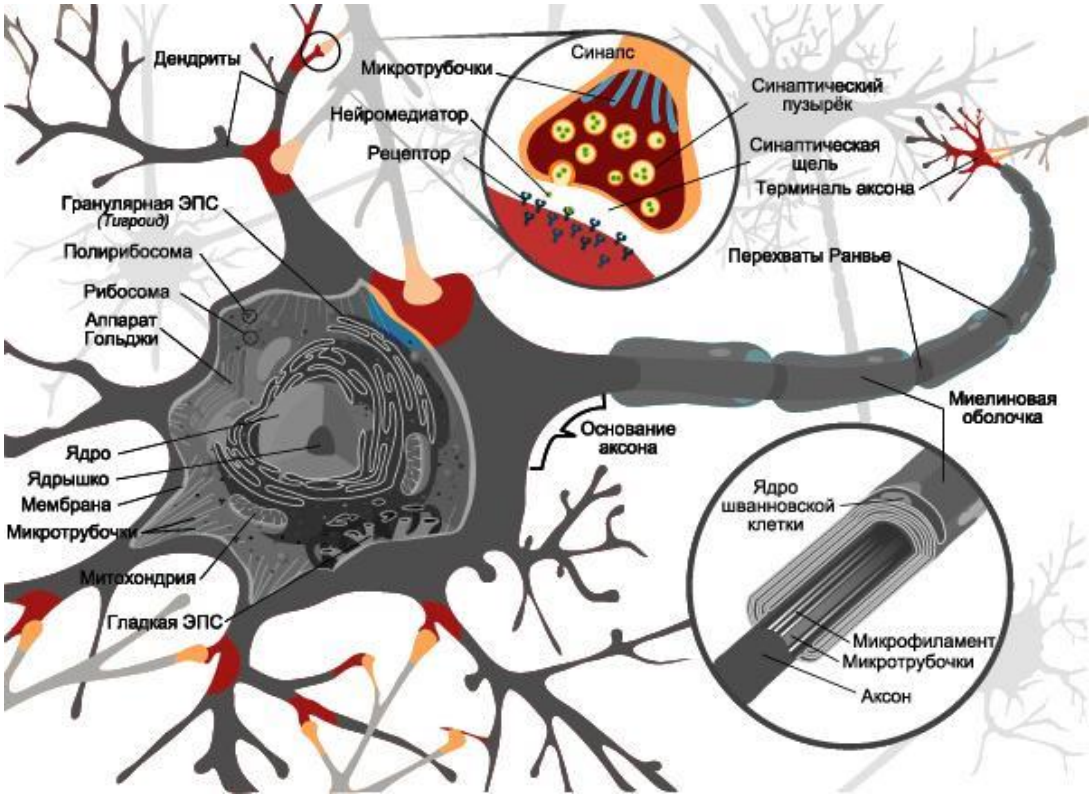


Рисунок 2.1 Биологический нейрон

В процессе жизни синапсы изменяются, соответственно меняется и сигнал, который они через себя пропускают. Благодаря слаженной работе синапса и нейрона поступающие сигналы на выходе становятся верными решениями.

Теперь рассмотрим внутреннюю структуру искусственного нейрона, и то, как он обрабатывает входящий сигнал и то, что он возвращает на выходе (см. рис. 2.2).

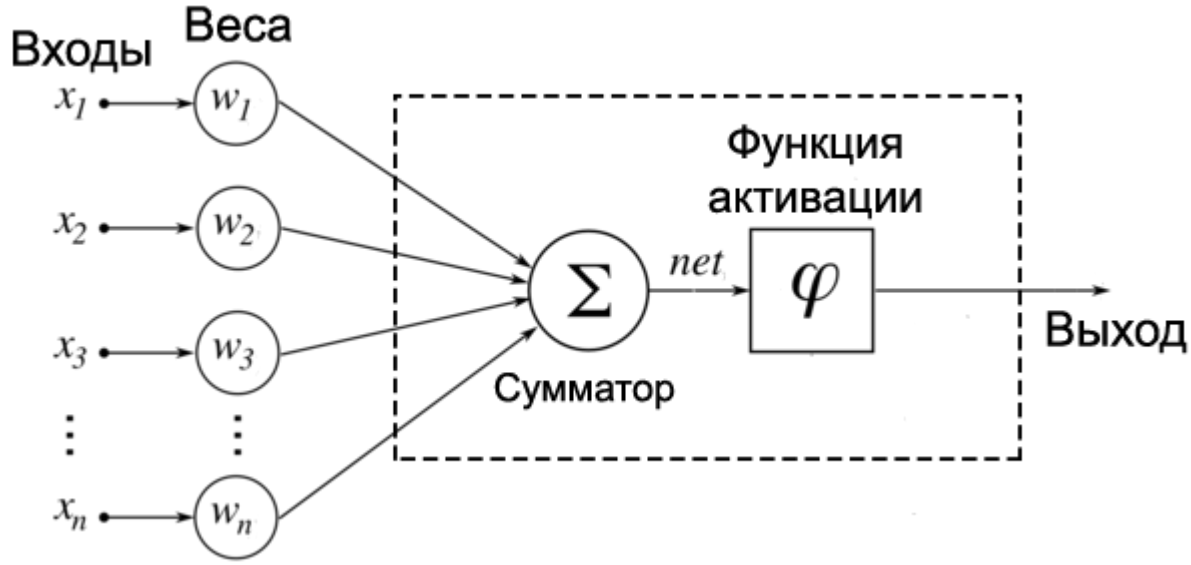


Рисунок 2.2 Искусственный нейрон

Для каждого нейрона есть свои входы, через которые поступает сигнал. Такими входами являются $x_1, x_2, x_3, \dots, x_n$. Соответственно $w_1, w_2, w_3, \dots, w_n$ – это веса, на которые умножаются сигналы нейронов. В последствии полученное произведение $x_n w_n$ передаётся в сумматор, где происходит сложение всех сигналов. Результат работы сумматора называют – взвешенной суммой и обозначается как net. [1]

$$net = \sum_{i=1}^{n} x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_n w_n$$

Роль сумматора заключается в том, что он объединяет все поступающие сигналы в одно число, которое описывает поступившую на нейрон информацию в общем.

Дальше сигнал net проходит через функцию активации, обозначенную на рисунке как φ, после которой и получается сигнал на выходе нейрона (out).

2.3 Функции активации

Подавать на выход нейрона просто взвешенную сумму не имеет никакого смысла, ведь из этой суммы должен быть сформирован корректный сигнал out. Именно для этого нам нужна функция активации.

Функция активации ($\varphi(net)$) – функция, которая принимает как аргумент взвешенную сумму нейрона.

$$out = \varphi(net)$$

2.3.1 Функция единичного скачка

Самой элементарной функцией активации является – функция единичного скачка. Результатом ее работы может быть 0 или 1. Значение нейрона будет равно 0, если net меньше установленного порогового значения b, в противном случае out примет значение 1. Запись это функции выглядит следующим образом (см. рис. 2.3):

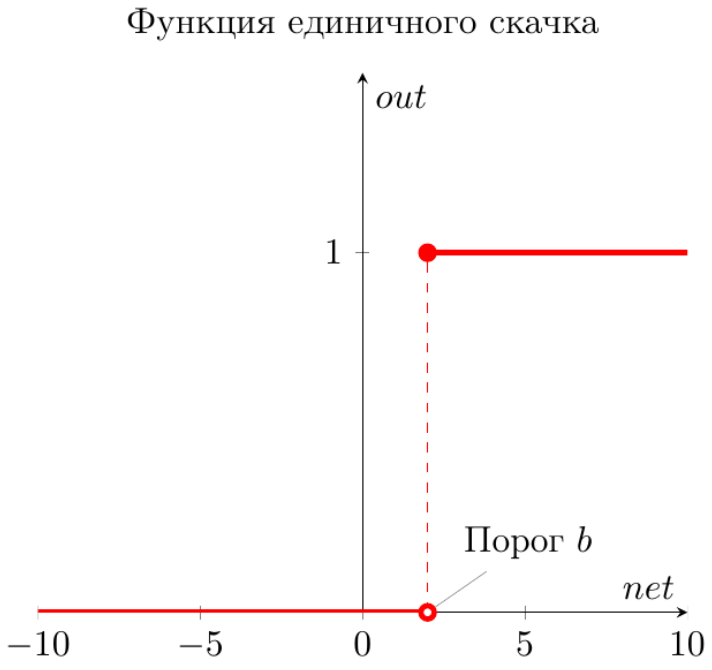


Рисунок 2.3 Функция единичного скачка

$$out(net) = \begin{cases} 0, net < b \\ 1, net \geq b \end{cases}$$

2.3.2 Сигмоидальная функция

Существует большое множество сигмоидальных функций, которые могут использовать для функции активации в нейросети. Чаще всего на практике встречается логистическая функция (см. рис. 2.4).

$$out(net) = \frac{1}{1 + \exp(a \cdot net)}$$

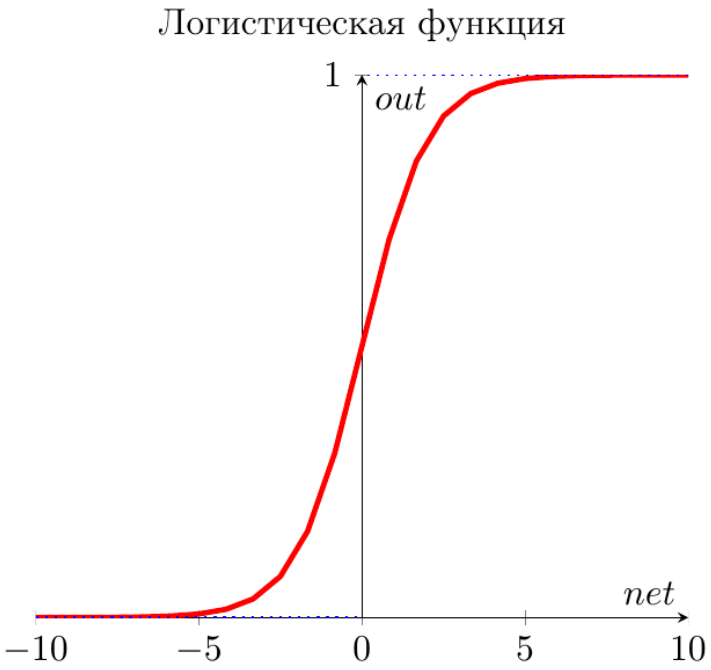


Рисунок 2.4 Логистическая функция

Эта функция может принимать значение в пределах от 0 до 1. Так как она не линейна, то с помощью нее можно конструировать многослойную нейросеть. Плюсом логистической функции является ее не бинарность в отличии от функции единичного скачка, что позволяет получить конкретные значения на выходе. В промежутке $net\,(-2; 2)$ величина out может стремительно меняться. Соответственно малейшее изменение net в это промежутке влечёт за собой изменение out. Такое свойство функции указывает на то, что значение out стремится к одной из сторон промежутка.

Логистическая функция часто используется в виде активационной функции, однако стоит учитывать при выборе и ее недостатки. [8]

Хоть в центре диапазона и характерно быстрое изменение значения out, по краям это значение почти не будет реагировать на изменение net. Это

значит, что по краям градиент будет стремиться к 0. Это означает что обучение нейронной сети будет очень медленным иди вовсе остановится.

2.3.3 Функция ReLU

За последние годы особую популярность получила функция активации, которая называется ReLU (rectified linear unit). Функция описывает пороговый переход в нуле и записывается следующим образом:

$$out(net) = \max(0, net)$$

Основываясь на определении, можно понять, что ReLU возвращает net без изменений если $net > 0$ иначе возвращает 0. На графике функция будет выглядеть так (см. рис. 2.5):

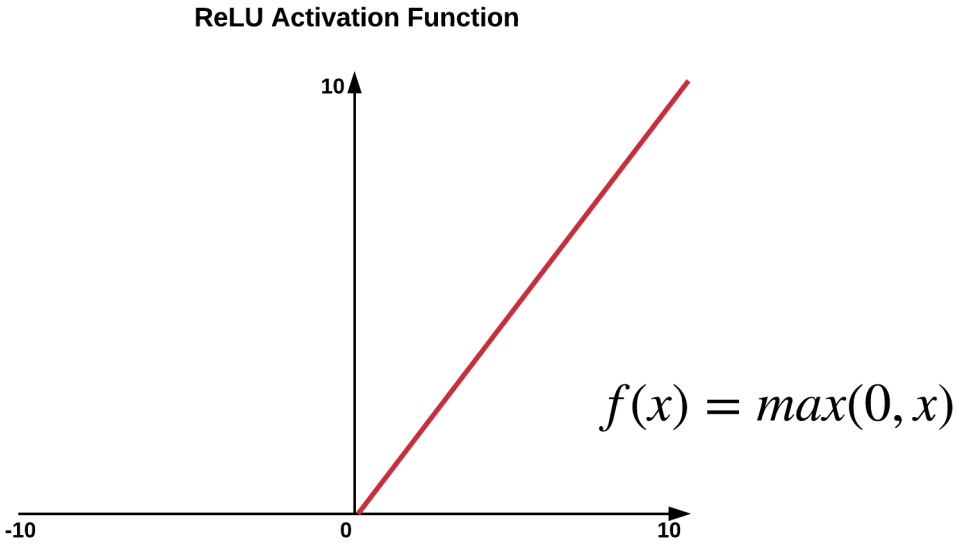


Рисунок 2.5 Функция активации ReLU

Функция ReLU не линейна. К тому же по сравнению с сигмоидами вычисление этой функции не требует ресурсоёмких операций. Использование этой функции заметно увеличивает сходимость градиентного спуска. Это характеризуется линейным характером функции и такому свойству как отсутствие насыщения. [2]

2.4 Типы нейронных сетей

2.4.1 Однослойный перцептрон

В одиночку нейрон может совершать простейшие вычисления, но основные функции нейросети работают не на единичных нейронах, а на их группах, связанных между собой. Самая простая нейросеть состоит из одного слоя нейронов, на вход каждого из которых подаётся вектор данных $x = (x_1, x_2, \dots, x_i)$ (рис. 6). Называют такую сеть однослойный персептрон. Внутри неё, нейроны по одиночке совершают вычисления, в последствии объединяя их в выходное значение. Размерность такого выходного значения будет прямо пропорциональна количеству нейронов в сети, а количество синапсов нейрона должно соответствовать размеру входящего сигнала (см. рис. 2.6).

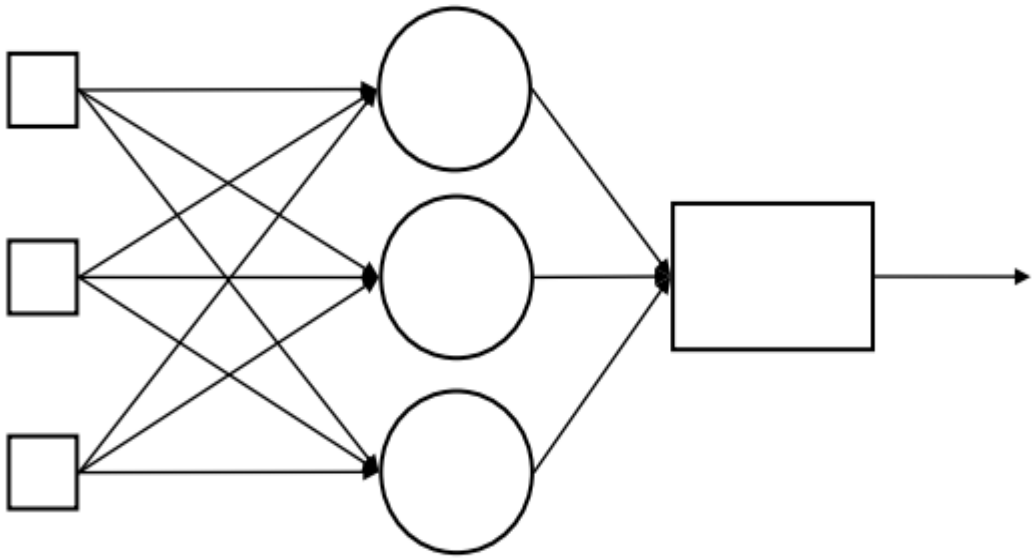


Рисунок 2.6 Схема однослойного перцептрона

На первый взгляд однослойный перцептрон выглядит весь простой системой, однако он способен большое множество задач, где поступающие на вход данные возможно разделить по размерам, например классификация образов.

2.4.2 Многослойные нейросети

Ещё один вид нейронных сетей носит название многослойные нейронные сети. Кроме основного слоя нейронов она включает в себя один или несколько скрытых слоёв. Такая сеть имеет много возможностей.

Скрытые слои берут на себя роль промежуточного звена между входящим и выходящим сигналом нейросети. Чем многослойней сеть, тем больше сложных признаков она способна выявлять (см. рис. 2.7).

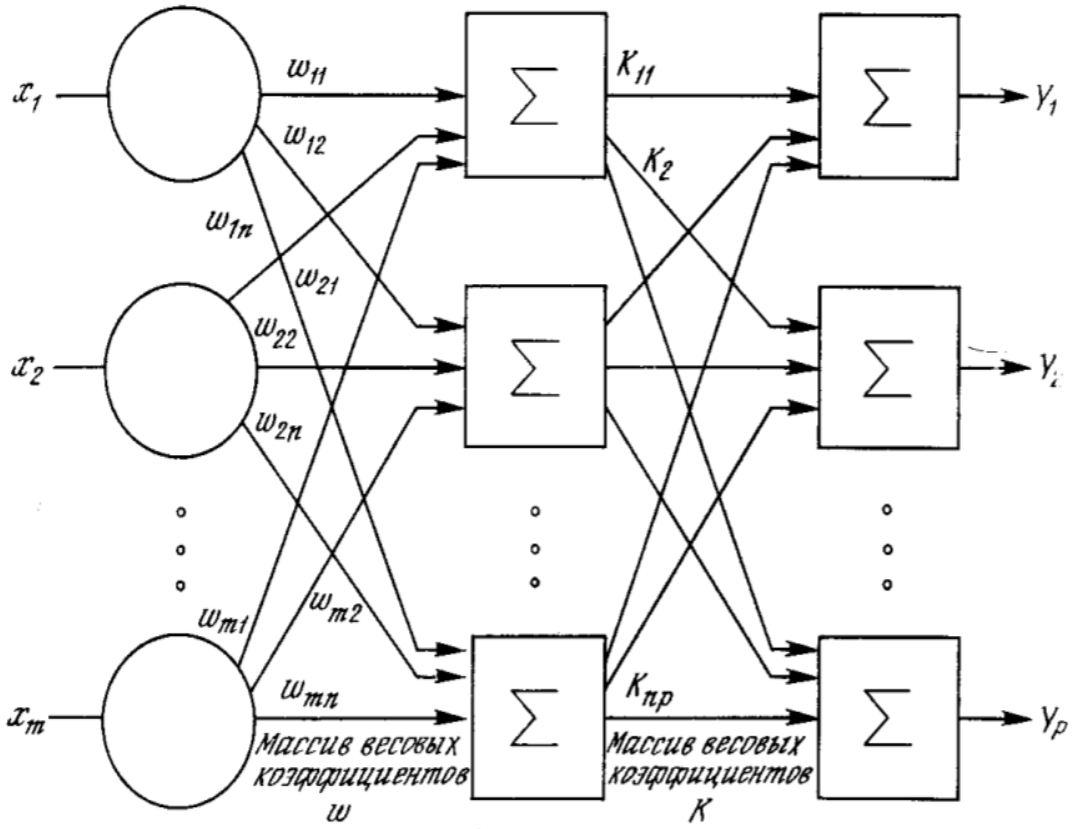


Рисунок 2.7 Многослойная нейросеть

Из-за того, что нейроны многослойной сеть взаимодействуют на более высоком уровне и между ними дополнительно существую синаптические связи, сеть способна распознавать глобальные признаки поступающих данных. Чем больше нейронов в одном слое, тем сильнее проявляется способность нейронов выделять качественные зависимости.

2.4.3 Полносвязные нейросети

На рисунке 8 изображена полносвязная нейросеть. Её характерным признаком является то, что каждый нейрон, находящийся в одном слое связан со всем нейронами соседних слоёв (см. рис. 2.8).

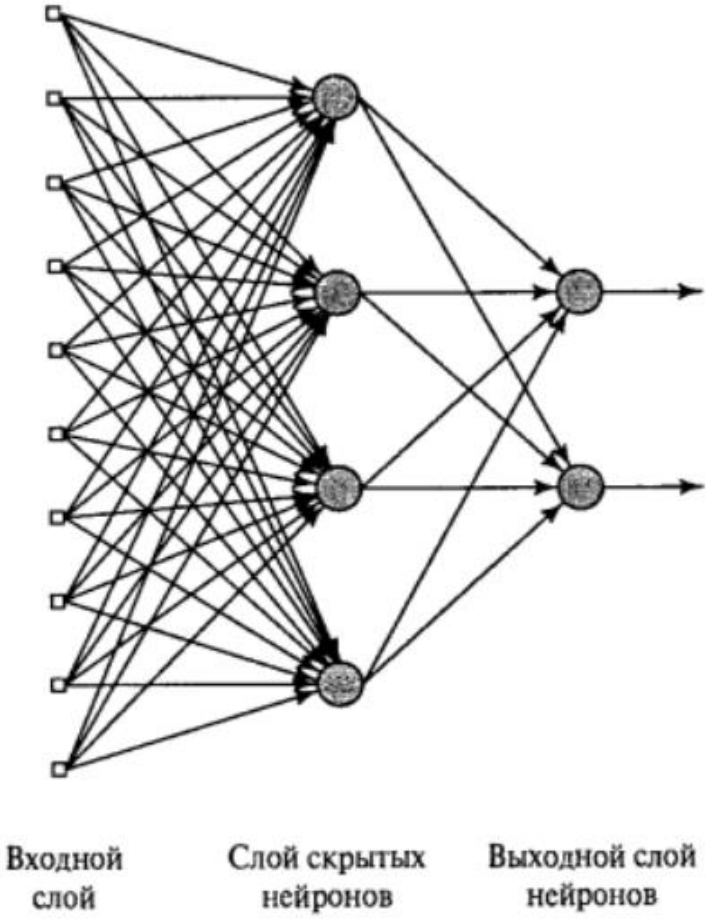


Рисунок 2.8 Схема полносвязной нейронной сети

Неполносвязной, считается нейросеть у которой отсутствует одна или множество синаптических связей.

При использовании многослойный сетей происходит рост мощностей, которые необходимы для вычисления. Этого можно избежать если функция активации между слоями будет нелинейной. Сигнал выхода считается как произведение вектора входящего сигнала на первую матрицу синапсических весов и следующее умножение, в случае если функция не линейна, получившегося вектора на следующую весовую матрицу.

$$(x \cdot w_1) \cdot w_2$$

Так как умножение матриц ассоциативно из этого следует что:

$$\mathrm{x} \cdot (w_1 \cdot w_2)$$

Это значит, что 2 идущих подряд слоя можно заменить одним, матрица весов которого будет равна произведению матриц весов 2 этих слоёв.

Получается, что для любой линейной многослойной нейронной сети можно найти аналог в виде равносильной однослойной сети. Однако из-за того, что однослойные нейронные сети, сильно ограничены в вычислениях, появляется нужда в использовании не линейный функций активации.

2.4.4 Рекуррентные нейросети

Теперь рассмотри класс сетей с обратными связями – рекуррентные нейронные сети. В таких сетях сигналы нейронов выходят не только к нейрону следующего входа, но и обратно на вход того нейрона, из которого выходил сигнал. До это го были рассмотрены сети без обратных связей, соответственно они не имели кратковременной памяти (см. рис. 2.9).

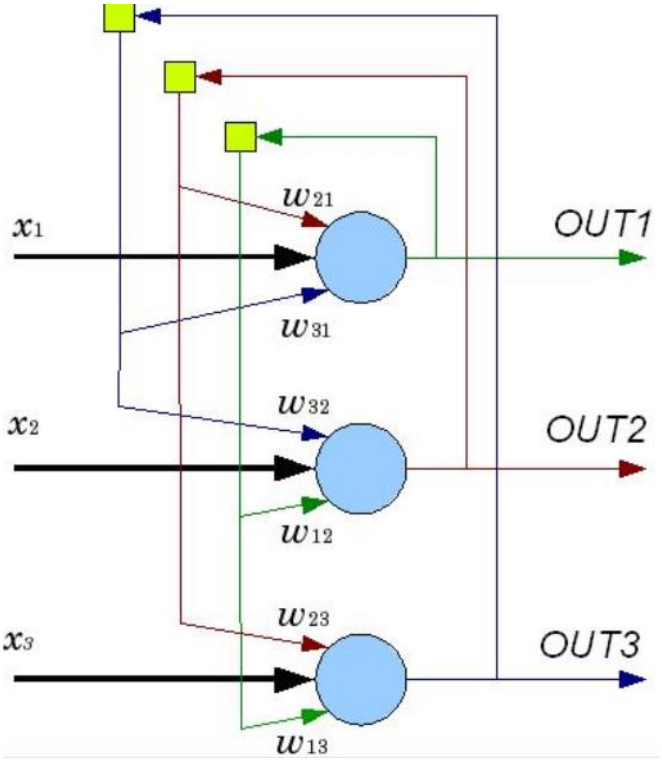


Рисунок 2.9 Рекуррентная нейронная сети

Форма сетей с обратными связями образует связь между текущими слоями нейронов и следующими после них. Это означает что рекуррентные сети способны проявлять свойства короткой памяти человека. Сеть, изображенная на рисунке 9, содержит в себе один слой нейронов. От наличия обратных связей сильно зависит предрасположенность сети к обучению, а также их производительность.

3 Практическая часть

Для написания программного кода был выбран пакет Python 3.10.1. К нему возможно подключить готовые библиотеки, для различных нужд во время реализации поставленной задачи, начиная от работы со строками и массивами до обучения нейронных сетей и компьютерного зрения. В этом плане Python предоставляет гибкий инструментарий, который можно настроить под свои нужды.

Для работы с нейронными сетями было выбрано 2 библиотеки для машинного обучения: Keras и PyTorch.

Keras работает поверх библиотеки TensorFlow, имеет интуитивно понятный синтаксис и включает в себя множество популярных строительных блоков для создания нейронных сетей. Библиотека имеет подробную документацию и активно поддерживается разработчиками. [3]

PyTorch позволяет ускорить необходимые для работы вычисления за счёт GPU компьютера. Эта технология называется CUDA. Так же эта библиотека имеет в своей коллекции большой выбор заранее обученных моделей нейронных сетей, которые возможно переучить при необходимости. Однако PyTorch имеет очень плохую документацию, что замедляет изучение этой библиотеки. [4]

В ходе работы была построена нейронная сеть, способная восстанавливать изображение. В основе архитектуры были использованы сверхточные слои, с дополнительными идейными особенностями позаимствованными из архитектуры VGG19 и ResNet.

Перед тем как начать проектировать собственную нейронную сеть, было принято решение посмотреть, как с задачей восстановления изображения будут справляться уже готовые архитектуры нейросетей. Для этого выбрана архитектура VGG19, доступная в пакете PyTorch. Перед тем как преступить к работе с ней изучим подробнее её архитектуру (см. рис. 3.10).

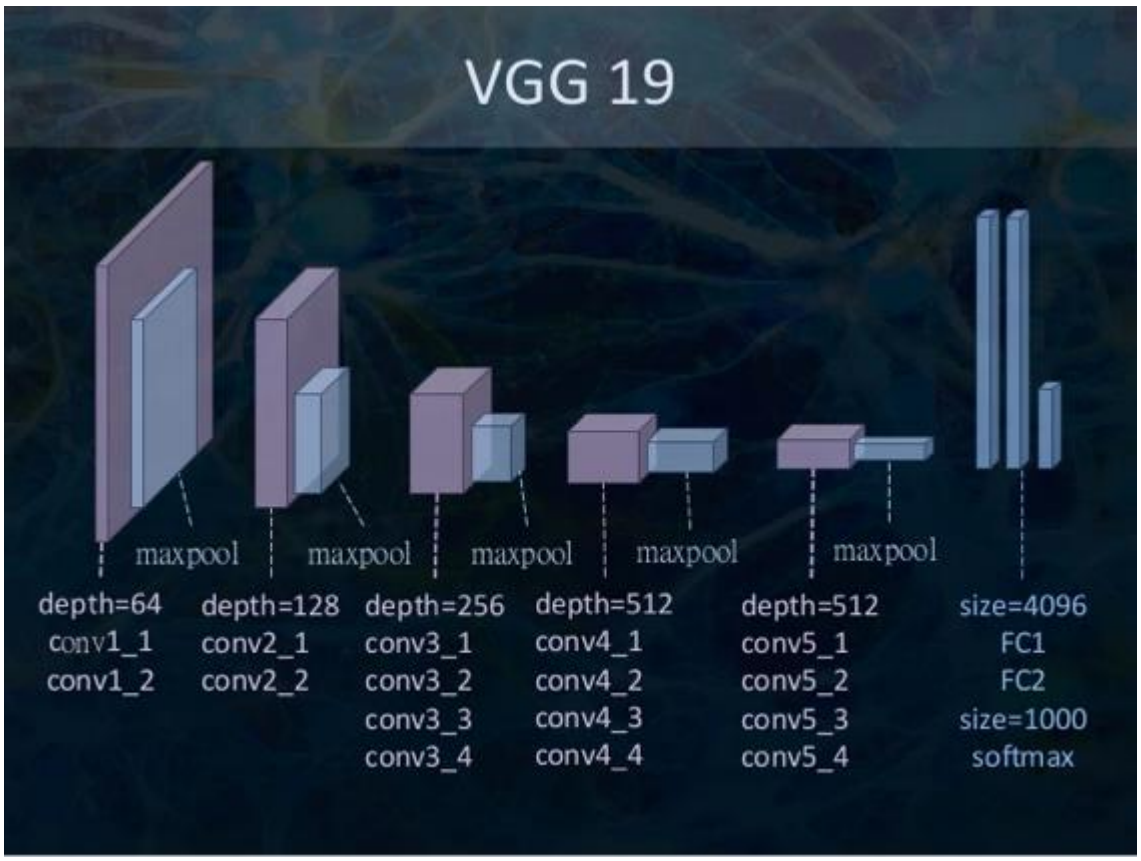


Рисунок 3.10. Архитектура VGG19

На вход нейронной сети подаётся изображение размером 224x224 в трёхканальном виде RGB. После чего изображение поочерёдно проходит через свёрточные слои conv1_1 и conv1_2 которые содержат в себе 64 фильтра с ядром, размером 3 на 3. Следующий слой maxpool уменьшает карту признаков в 2 раза, после чего снова идут 2 свёрточных слоя, но уже имеющие 128 фильтров. После снова слой maxpool, а за ним уже 4 свёрточных слоя с 256 признаков. Такая последовательность повторяется ещё 2 раза, но уже со слоями, имеющими 512 фильтров. Полученный результирующий тензор подаётся в полносвязную нейросеть имеющую 2 скрытых слоя с 4096 нейронами и выходным слоем в 1000 нейронов. 1000 нейронов описывает множество классов, для которых она обучалась.

Идея VGG19 заключается именно в свёрточных слоях, идущих друг за другом. На месте слоёв conv1_1 и conv1_2 размером 3x3 мог использоваться один равносильный слой 5x5. Однако количество настраиваемых параметров у ядра 5x5 равно

$$5 \cdot 5 + 1 = 26$$

А у пары ядер 3x3

$$2 \cdot (3 \cdot 3 + 1) = 20$$

Чем меньше количество настраиваемых параметров, тем быстрее будет обучаться нейросеть. В этом и заключается идея VGG 19.

Для задачи восстановления изображения мной была сформирована выборка, состоящая из 31000 цветных изображений в формате jpg, и дополнительно 1000 изображений для тестирования полученных результатов.

Задача восстановления изображения для нейронной сети подразумевает под собой, что, подавая во внутрь изображение, на выходе мы должны получить его восстановленным от всех искажений.

Работа в системе, содержащей в себе процессор AMD Ryzen 5 3600 6-Core Processor с тактовой частой 3.59 GHz и видеокарту MSI GeForce GTX 1050 TI 4GB, что помогло ускорить обучение нейронной сети.

При обучении все изображения форматировались под размер 500x500 пикселей, для того чтобы особо не терять качество исходных изображений, при этом сильно не нагружать систему.

Для тренировки сети необходимо 2 пакета изображений.

- оригинальные изображения;
- испорченные изображения.

Для этого сначала нужно было создать функцию, которая бы искажала исходное изображение понижением яркости и накладыванием различных шумов.

```
def noisy(noise_typ,image):
    if noise_typ == "gauss":
        row,col,ch= image.shape
        mean = 0
        var = 0.0001
        sigma = var**0.05
        gauss = np.random.normal(mean,sigma,(row,col,ch))
        gauss = gauss.reshape(row,col,ch)
        noisy =  gauss + image
        return noisy
    elif noise_typ == "s&p":
        row,col,ch = image.shape
        s_vs_p = 0.5
        amount = 1.0
        out = np.copy(image)
        # Salt mode
        num_salt = np.ceil(image.size * s_vs_p)
        coords = [np.random.randint(0, i, int(num_salt))
              for i in image.shape]
        out[coords] = 1

        # Pepper mode
        num_pepper = np.ceil(image.size * (1. - s_vs_p))
        coords = [np.random.randint(0, i , int(num_pepper))
              for i in image.shape]
        out[coords] = 1
        return out
    elif noise_typ == "poisson":
        vals = len(np.unique(image))
        vals = 2 ** np.ceil(np.log2(vals))
        noisy = np.random.poisson(image * vals) / float(vals)
        return noisy
    elif noise_typ =="speckle":
        row,col,ch = image.shape
        gauss = np.random.randn(row,col,ch)
        gauss = gauss.reshape(row,col,ch)
        noisy = image + image * gauss
        return noisy
```

Рисунок 3.11. Функция наложения шума

Это функция позволяет накладывать на изображение различные виды шумов, такие как гауссовский шум, дробовой шум, спекл-шум и «salt and peper» (см. рис. 3.11).

Искажение изображения происходило следующим образом:

1) исходное изображение затемнялось;
2) на затемнённое изображение накладывались шумы.

Получая из нужное изображение, его размер меняется на 500x500, после чего изображения искажаются. Полученные данные записываются в 2 массива X (искажённые картинки) и y (оригинальные картинки) (см. рис. 3.12).

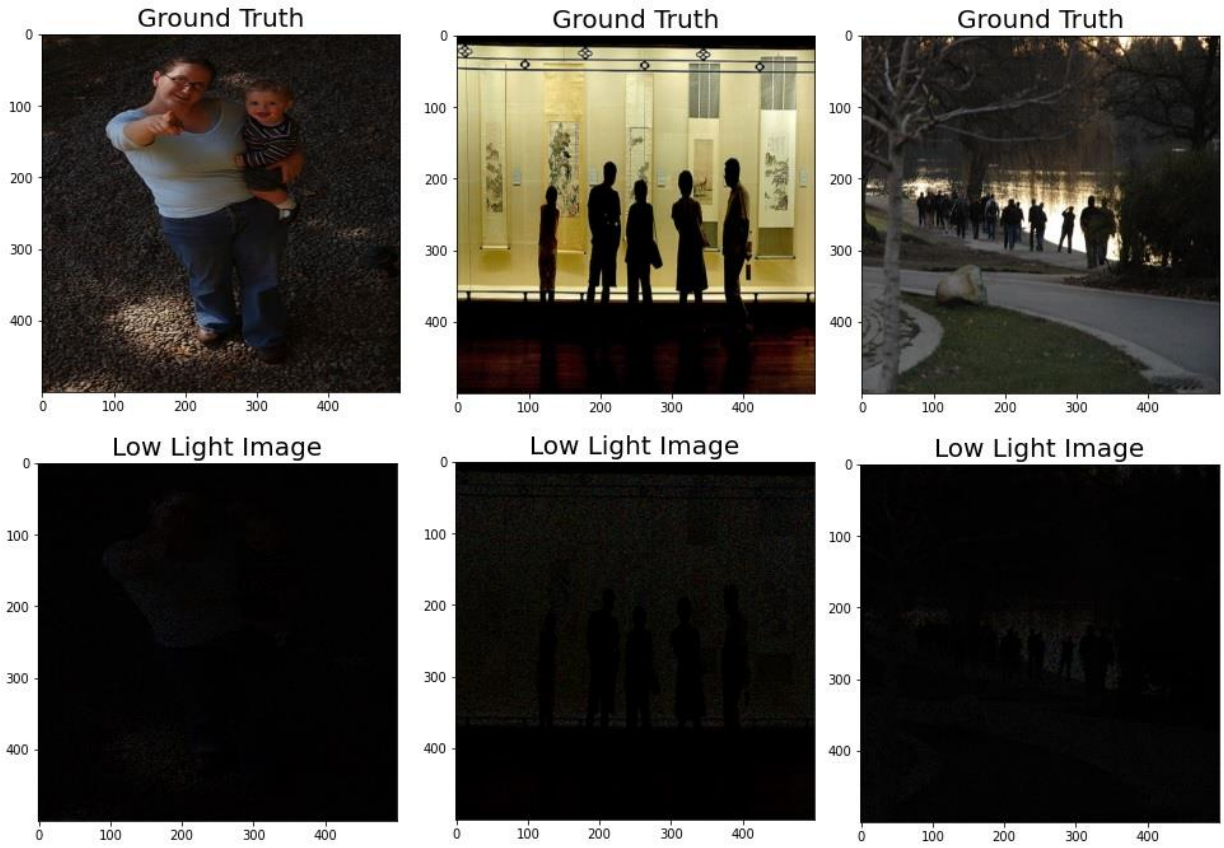


Рисунок 3.12. Примеры Оригинального и искажённого изображения

Так как VGG19 изначально была обучена для распознавания образов её необходимо было переучить для поставленной задачи. Нейросеть была переобучена заранее с использованием сервиса Google Colab, который бесплатно предоставляет аппаратное обеспечение для облачного обучения нейронных сетей.

```
vgg = models.vgg19(pretrained=True).features

for param in vgg.parameters():
    param.requires_grad_(False)
```

Рисунок 3.13. Импорт VGG19

Подгружаем заранее обученную нейросеть VGG19 (см. рис. 3.13). И присваиваем параметру requires_grad значение false, для того чтобы использовать исходные настроенные веса для работы.

При работе с VGG19 мне хотелось добиться максимального результата нейросети. Для этого входящее изображение будет пропущено через нейросеть 200 раз.

```
show_every = 50     # Мы показываем результаты итераций после каждых 50 изображений.
optimizer = optim.Adam([target], lr=0.003)  # Мы просто корректируем параметры целевого изображения.
steps = 200 # Количество итераций, которые мы хотим сделать для достижения лучшей точности. Чем больше шагов, тем выше точность.
```

Рисунок 3.14. Настройки для работы VGG19

Результат будет выводится на экран каждые 50 итераций (см. рис. 3.14). Оптимизатором для работы нейронной сети будет функция Adam основанная на алгоритмах с моментом и квадрата градиентов. Этот оптимизатор является наиболее используемым и для большинства задач даёт хорошую сходимость.

```
for ii in range(1, steps+1):
  target_features = get_features(target, vgg) # Сначала мы получаем все признаки для целевого изображения, когда оно помещается в предварительно обученную модель vgg19.
  total_loss = torch.mean((target_features['conv4_2'] - Noise_features['conv4_2'])**2) # Затем мы вычитаем эти значения характеристик из значений характеристик contect_image для того же слоя, чтобы получить потери.

  if  ii % show_every == 0:   # Показывать прогресс вывода изображения после каждых 50 шагов.
    print('Total loss: ', total_loss.item())
    print('Iteration: ', ii)
    plt.imshow(im_convert(target))
    plt.axis("off")
    plt.show()
```

Рисунок 3.15. Вывод работы нейросети VGG19

В матрице target_features записываются все признаки изображения, после прохождения через нейросеть (см. рис. 3.15). После чего высчитывается total loss путём вычитания характеристик обработанного изображения из характеристик оригинального изображения. На 200 итерации потери были незначительны. Это означает что выходное изображение было схоже с оригиналом (см. рис. 3.16).

```
Total loss:  0.00246405391953885556
Iteration:  200
```

Рисунок 3.16. Потери при работе с VGG19

Выходное изображение стало немного не чётким, но затемнение и шумы были полностью убраны без потерь в деталях (см. рис. 3.17).

Рисунок 3.17. Результат работы VGG19

Теперь, когда имеется представление о том, как нейронные сети способны справляться с задачей восстановления изображения можно приступать к созданию собственной нейросети. Изучая теорию, было установлено что для задач восстановления изображения подходят свёрточные нейронные сети. Поэтому в основе структуры нейросети будут использоваться свёрточные слои. Дополнительно к этому, будет использована идейные заимствования архитектур VGG19 и ResNet. У VGG19 это – использование подряд идущих слоёв с ядром размером 3x3 для оптимизации количества настраиваемых параметров. У ResNet это использование обходных путей для

упрощения обучения глубокой нейронной сети, так как проектируемая нейросеть будет состоять из 16 слоёв.

Рассмотрим подробнее архитектуру ResNet. Архитектура появилась в ходе реализации идеи глубокого остаточного обучения. Нейросеть разбита на блоки, которые включают в себя стандартные слои, однако у каждого блока есть особенность – обходной путь. Этот путь напрямую соединяет входной и выходной сигналы. В итоге, сигнал, который выходил из этого блока имел вид:

$$y_k = F(x_k) + x_k$$

Преимущество обходного пути в том, что если, к примеру, нейросети нужно будет аппроксимировать функцию:

$$y_k = S(x)$$

То благодаря этому принципу получится

$$F(x) = S(x) - x$$

Это означает, что аппроксимировать придётся не всю функцию $S(x)$, а только остаток от неё. Поэтому эти блоки носят название – остаточные блоки. Поэтому будет достаточно научить нейросеть воспроизводить разницу в значениях для получения необходимой функции (см. рис. 3.18).

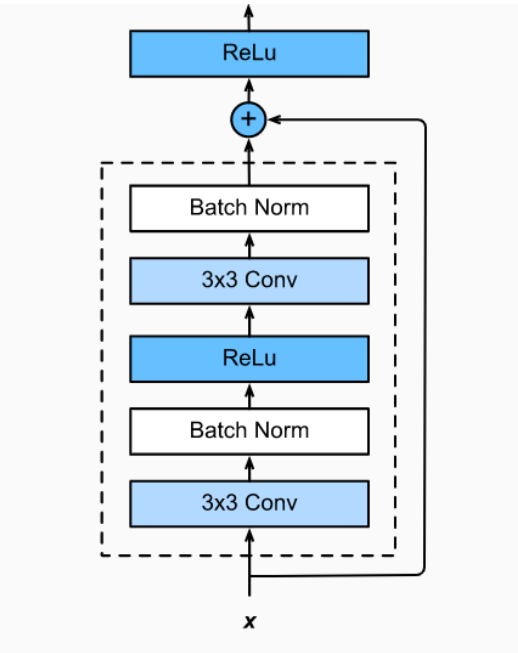


Рисунок 3.18. Остаточный блок архитектуры ResNet

Такую функцию обучать намного проще, чем начальную. Это объясняется ещё тем, что градиент остаточного блока имеет формулу:

$$\frac{\partial y_k}{\partial x_k} = 1 + \frac{\partial F(x_k)}{\partial x_k}$$

Из-за единицы, возникающей благодаря «обходному пути», градиент не затухает от нижних слоёв к верхним. Это означает что нейросеть будет обучаться равномерно везде, какой бы глубокой она не была.

Рассмотрим, как выглядит созданная мной нейронная сеть (см.рис. 3.19):

```
K.clear_session()
def InstantiateModel(in_):

    model_1 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(in_)
    model_1 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_1)
    model_1 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_1)

    model_2 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(in_)
    model_2 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_2)

    model_2_0 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_2)

    model_add = add([model_1,model_2,model_2_0])

    model_3 = Conv2D(64,(3,3), activation='relu',padding='same',strides=1)(model_add)
    model_3 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_3)
    model_3 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_3)

    model_3_1 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_add)
    model_3_1 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_3_1)

    model_3_2 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_add)

    model_add_2 = add([model_3_1,model_3_2,model_3])

    model_4 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add_2)
    model_4_1 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add)

    model_add_3 = add([model_4_1,model_add_2,model_4])

    model_5 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add_3)
    model_5 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_add_3)

    model_5 = Conv2D(3,(3,3), activation='relu',padding='same',strides=1)(model_5)

    return model_5
    X_ = np.array(X_)
    y_ = np.array(y_)

    return X_,y_
X_,y_ = PreProcessData(InputPath)
```

Рисунок 3.19. Программа создания нейросети №1

На вход нейронной сети подаётся изображение размером 500x500 в формате BRG (blue, red, green). Вся нейросеть состоит из свёрточных слоёв Conv2D и суммарных сигналов функции add.

Если в обычной полносвязной нейросети, каждый нейрон связан со всеми нейронами соседних слоёв, то в свёрточных сетях осуществляется процесс свёртки, для которой используются матрицы весов ограниченного размера, которая двигается по всему слою. Эту матрицу называют ядром свёртки. Она несёт в себе графическую информацию о каком-либо признаке, который обнаружила нейросеть. Следующий слой, появившийся в процессе свёртки этой матрицы весов, указывает на существование этого признака в слое и его координаты. Набор таких признаков в совокупности образует карту признаков. Прохождение разными наборами весов образует собственные карты признаков, делая нейросеть многоканальной (множество разных карт признаков на одном слое сети). [6]

В самом начала используется число фильтров равное 16. Задаётся ядро размером (3,3), эти числа задают ширину и высоту свёртки.

Activation = ‘relu’, означает что функцией активации для этого слоя будет функция Relu. [14]

Padding = ‘same’ означает что размер слоя останется без изменений.

Слои с функцией add совершают сложение тензоров из указанных выходов, создавая обходные пути, как в архитектуре ResNet.

На выходе нейронной сети будет получено изображение такого же типа данных что и на входе.

```
Input_Sample = Input(shape=(500, 500,3))
Output_ = InstantiateModel(Input_Sample)
Model_Enhancer = Model(inputs=Input_Sample, outputs=Output_)

Model_Enhancer.compile(optimizer="adam", loss='mean_squared_error')
Model_Enhancer.summary()
```

Рисунок 3.20. Сбор нейронной сети.

После того как конструирование нейронной сети завершено, можно её собрать (см. рис. 3.20). Оптимизатором, как и при работе с VGG19 будет adam. Он выдаёт лучший показатель сходимости для задач восстановления изображения. Критерием качества будет mean_squared_error (т.е. средний квадрат ошибок). Его формула выглядит так:

$$E = \frac{1}{N}\sum_{i=1}^{N}(\mathrm{d}_i - y_i)^2$$

Этот критерий будет сильно увеличивать показатель качество, что должно помочь нейросети лучше настраивать веса. [7]

Выведем summary для сконструированной модели (см. табл. 1):

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 128, 128, 3)	0
conv2d (Conv2D)	(None, 128, 128, 16)	448
conv2d_3 (Conv2D)	(None, 128, 128, 32)	896
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4640
conv2d_4 (Conv2D)	(None, 128, 128, 64)	8256
conv2d_2 (Conv2D)	(None, 128, 128, 64)	8256
conv2d_5 (Conv2D)	(None, 128, 128, 64)	16448
add (Add)	(None, 128, 128, 64)	0
conv2d_6 (Conv2D)	(None, 128, 128, 64)	16928
conv2d_9 (Conv2D)	(None, 128, 128, 32)	18464
conv2d_7 (Conv2D)	(None, 128, 128, 32)	18464
conv2d_10 (Conv2D)	(None, 128, 128, 16)	2064
conv2d_11 (Conv2D)	(None, 128, 128, 16)	4112
conv2d_8 (Conv2D)	(None, 128, 128, 16)	2064
add_1 (Add)	(None, 128, 128, 16)	0
conv2d_13 (Conv2D)	(None, 128, 128, 16)	9232
conv2d_12 (Conv2D)	(None, 128, 128, 16)	2320
add_2 (Add)	(None, 128, 128, 16)	0
conv2d_15 (Conv2D)	(None, 128, 128, 16)	1040
conv2d_16 (Conv2D)	(None, 128, 128, 16)	435
Total params: 134,067		
Trainable params: 134,067		
Non-trainable params: 0		

Таблица 1. Summary конструирования модели нейронной сети

С помощью библиотеки Keras можно графически изобразить построенную нейросеть (см. рис. 3.21).

```
from keras.utils.vis_utils import plot_model
plot_model(Model_Enhancer,to_file='model.png',show_shapes=True, show_layer_names=True)
from IPython.display import Image
Image(retina=True, filename='model_.png')
```

Рисунок 3.21. Код для графического отображения нейронной сети

Функция plot_model преобразует построенную модель в точечный формат и сохраняет её в файл. Параметр show_shapes включает вывод информацию о форме, а show_layer_name отображает имя каждого слоя. Теперь у нас есть модель нейросети в виде графа (см. рис. 3.22).

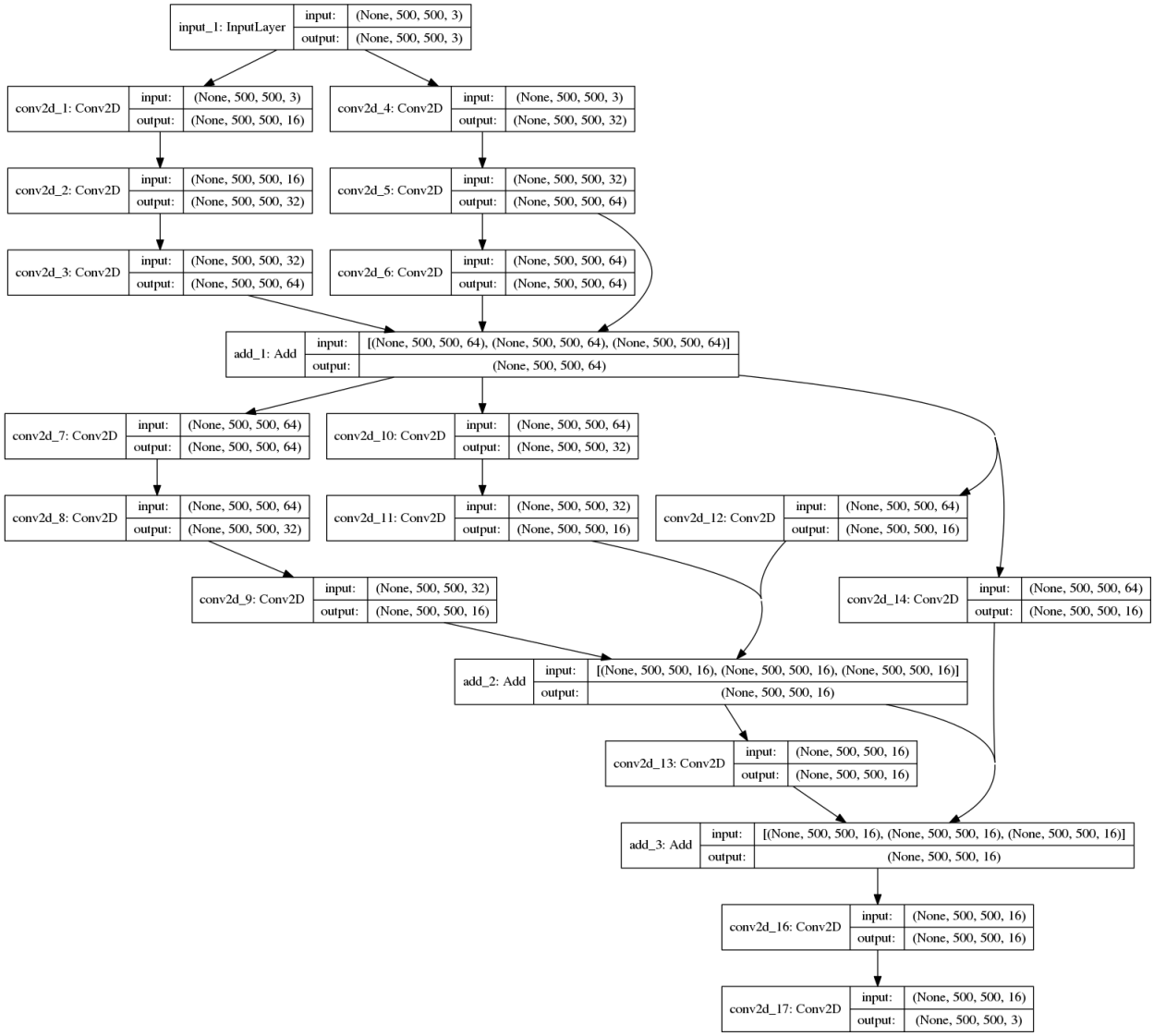


Рисунок 3.22. Схема нейронной сети

Теперь можно приступать к обучению нейронной сети. Сеть будет обучаться на 31000 изображений. Всего в процессе обучение пройдёт 31 эпоха, то есть 1000 картинок на одну эпоху. У нас уже имеется 2 массива с картинками, которые были подготовлены заранее. Но перед тем, как нейросеть получит эти данные необходимо произвести их нормализацию. Для этого каждое изображения мы приводим к тензору вида (1,500,500,3). Первое значение — это число партии. Второе и третье – размер изображение. Четвертое это количество каналов. На входе у нас подаётся трёхканальное изображение формата BRG, соответственно четвёртое значение равно 3.

За обучение отвечает функция fit. У неё тоже есть множество настраиваемых параметров. Сначала подаётся массив с искажёнными изображениями, потом с оригинальными. Параметр epoch отвечает за количество эпох. Verbose отвечает за то как будет отображаться процесс обучения (в данном случае шкала прогресса). Steps_per_epoch – это количество шагов в одной эпохе. Shuffle отвечает за перемешивание тренировочных данных (см. рис. 3.23).

```
def GenerateInputs(X,y):
    for i in range(len(X)):
        X_input = X[i].reshape(1,500,500,3)
        y_input = y[i].reshape(1,500,500,3)
        yield (X_input,y_input)
Model_Enhancer.fit(GenerateInputs(X_,y_),epochs=31,verbose=1,steps_per_epoch=1000,shuffle=True)
```

Рисунок 3.23. Код для обучения нейросети

Из-за нехватки ресурсов компьютера пришлось разделить обучение на 10 этапов каждый из которых содержал в себе 3000 изображений. Покажем вывод последнего этапа обучения для моей нейронной сети.

Epoch 1/3: 3000/3000 [==========] - 2554s 3s/step - loss: 243.4323

Epoch 2/3: 3000/3000 [==========] - 2532s 3s/step - loss: 230.6354

Epoch 3/3: 3000/3000 [==========] - 2594s 3s/step - loss: 235.7621

Одна эпоха изучалась около 48 минут. Всего на обучение нейронной сети понадобилось примерно 24 часа.

Рисунок 3.24. Результаты работы нейросети №1

На рисунках видно, что в местах, где изначально нет яркого белового цвета нейросеть отрабатывает почти идеально (см. рис. 3.24). Однако в местах, где он присутствует происходят сбои в работе нейросети. Это может быть связано с тем, что на этапе обучений нейросети, я столкнулся с её переобучение. Такой результат не был удовлетворителен. Поэтому была предпринята попытка решить проблему с переобучением и улучшить результат работы нейросети.

Больших изменений в архитектуру нейросети внесено не было. Но перед каждым слоем, который участвовал в закрытии остаточного блока добавлена процедура dropout.

Алгоритм dropout помогает избавиться от переобучения. Цель этого метода снизить специализацию каждого отдельного нейрона и сделать из них специалистов более широкого профиля. Именно в этом корень проблемы переобучения. Алгоритм работает следующим образом: на каждой итерации обучения нейронной сети, часть нейронов следует отбросить с вероятностью p. В результате общее количество нейронов остаётся прежним, а их специализация расширяется. Так работает алгоритм dropout. [15]

```
K.clear_session()
def InstantiateModel(in_):

    model_1 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(in_)
    model_1 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_1)
    model_1 = Dropout(0.4)(model_1)
    model_1 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_1)
    model_2 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(in_)
    model_2 = Dropout(0.4)(model_2)
    model_2 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_2)
    model_2_0 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_2)
    model_2_0 = Dropout(0.4)(model_2_0)
    model_add = add([model_1,model_2,model_2_0])
    model_3 = Conv2D(64,(3,3), activation='relu',padding='same',strides=1)(model_add)
    model_3 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_3)
    model_3 = Dropout(0.4)(model_3)
    model_3 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_3)
    model_3 = Dropout(0.4)(model_3)
    model_3_1 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_add)
    model_3_1 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_3_1)
    model_3_1 = Dropout(0.4)(model_3_1)
    model_3_2 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_add)
    model_3_2 = Dropout(0.4)(model_3_2)
    model_add_2 = add([model_3_1,model_3_2,model_3])
    model_4 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add_2)
    model_4 = Dropout(0.4)(model_4)
    model_4_1 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add)
    model_add_3 = add([model_4_1,model_add_2,model_4])
    model_5 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add_3)
    model_5 = Dropout(0.4)(model_5)
    model_5 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_add_3)
    model_5 = Dropout(0.4)(model_5)
    model_5 = Conv2D(3,(3,3), activation='relu',padding='same',strides=1)(model_5)
    return model_5
```

Рисунок 3.25. Программа для создания нейросети №2

Изменённая модель будет выглядеть так (см. рис. 3.25). Ещё одно изменение было совершено на стадии сборки нейронной сети. Критерием ошибки для нейросети будет средний модуль ошибки (mean absolute error). Резкое увеличение критерия качества для данной задачи не подходит так как при задаче восстановления изображения большие ошибки не всегда визуально влияют на итоговый результат. Поэтому было принято решение отказаться от критерия mean squared error.

Средний модуль ошибок имеет такую формулу:

$$E = \frac{1}{N}\sum_{i=1}^{N}|\mathrm{d}_i - y_i|$$

Так как в отличии от среднего квадрата ошибок, ошибка здесь, не возводится в квадрат, большие ошибки не так сильно будут влиять на критерий качества.

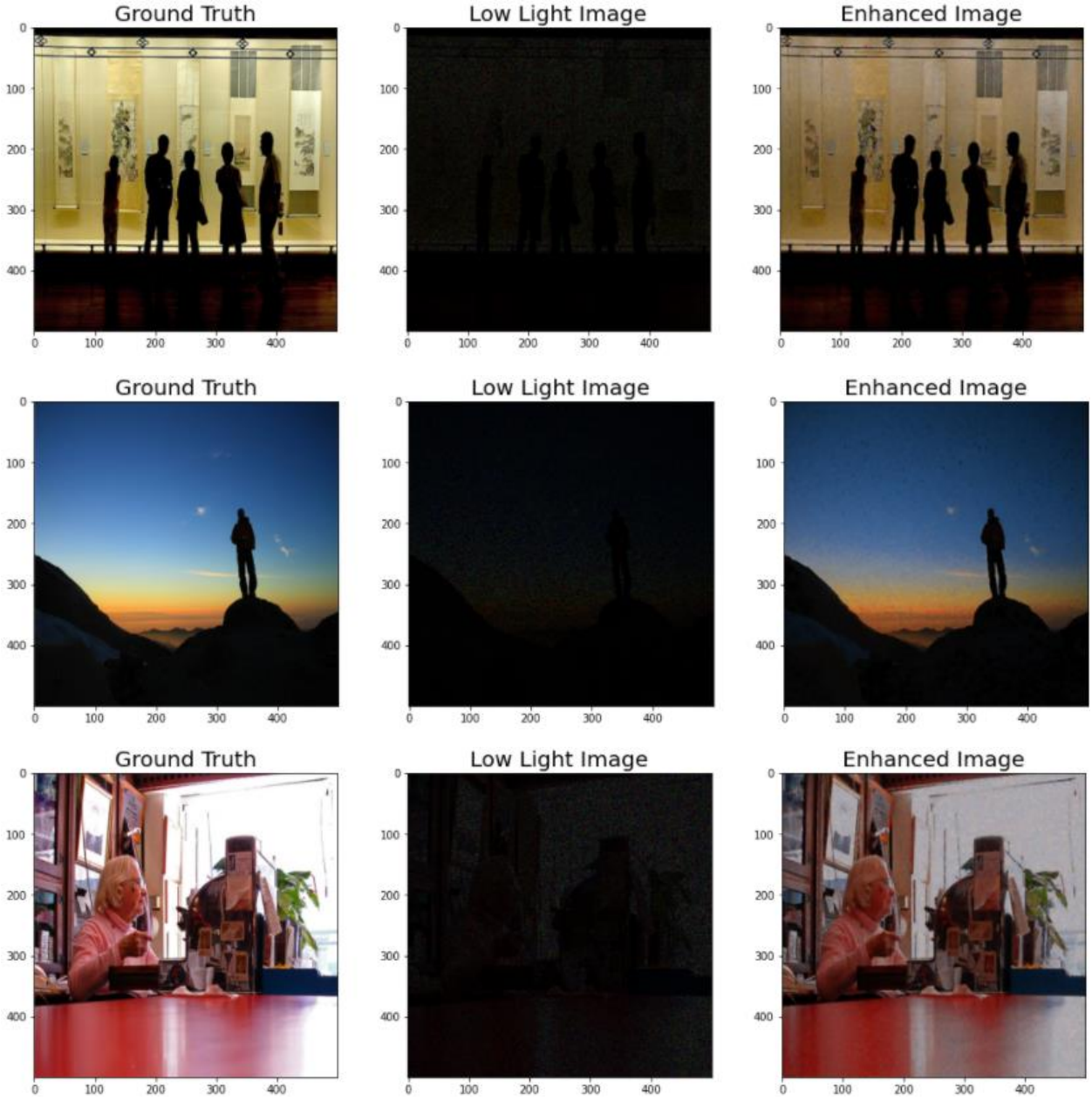
Теперь обучим нейросеть заново. Для примера показан последний этап обучения нейросети.

Epoch 1/3 1000/1000 [==========] - 2554s 3s/step - loss:13.9054

Epoch 2/3 1000/1000 [==========] - 2532s 3s/step - loss: 12.5530

Epoch 3/3 1000/1000 [==========] - 2594s 3s/step - loss: 12.6597

Общее время обучения так же, как и в первом случае заняло примерно 24 часа.



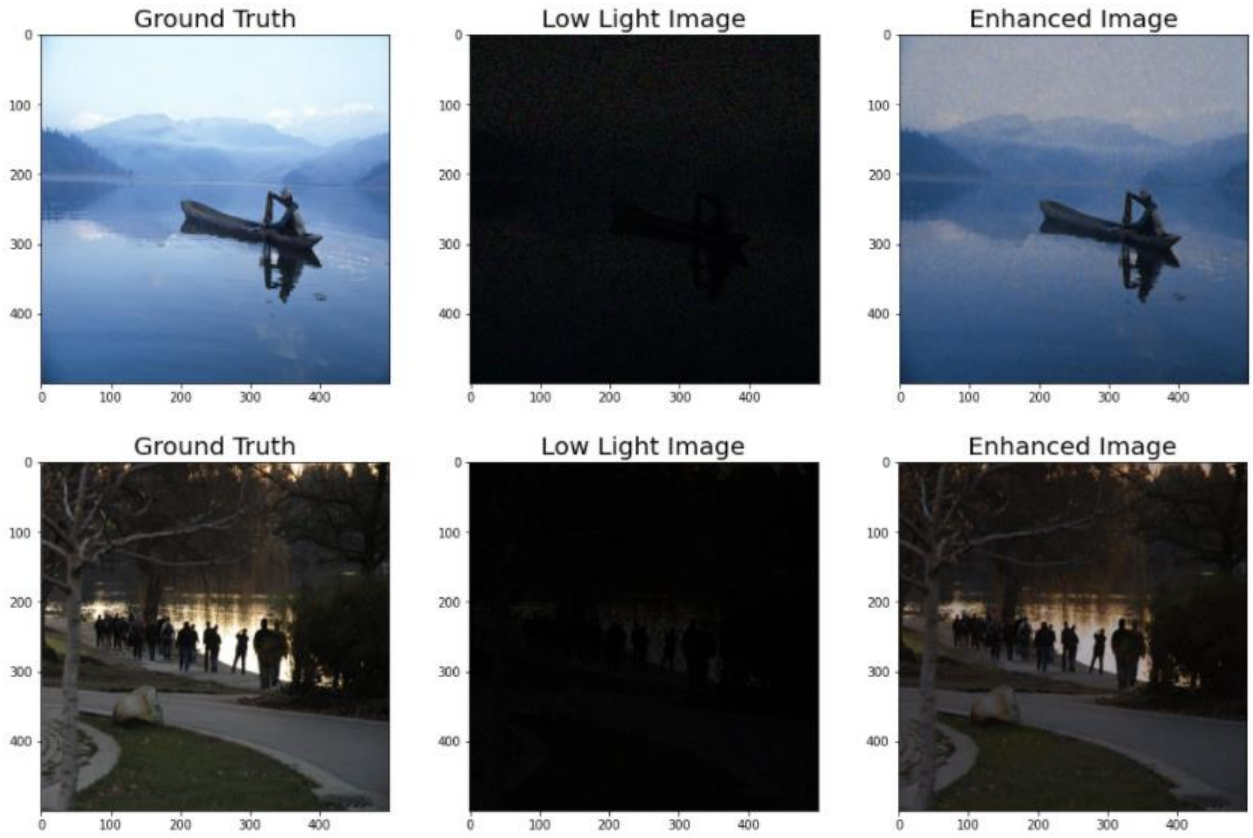


Рисунок 3.26 Результаты работы нейронной сети №2

Заметно что в этот раз нейросеть обучилась гораздо лучше. Выходное изображение отличается от оригинала лишь тем, что оно немного тусклее (см. рис. 3.26). Но никаких сбоев и повреждений не наблюдается

Таким образом можно сказать, что использование нейросетей очень хорошо справляется с задачей восстановления изображения.

Плюсом метода является то с какой скоростью обученная нейросеть способна восстановить искажённое изображение, почти не используя при этом ресурсы компьютера.

Из минусов можно выделить то, что обучение нейронной сети очень время затратный процесс.

Заключение

Целью данной выпускной квалификационной работы было восстановление изображения с помощью нейронных сетей.

Для достижения поставленной задачи были выполнены задачи:

- изучена история и выработано понимание о работе нейронных сетей;
- выбран наилучший тип нейронной сети для задачи восстановления изображении;
- изучены методы работы с библиотеками глубокого обучения Keras и PyTotch;
- приобретённые знания были применены на практике. Была переобучена существующая архитектура VGG19 для задачи восстановления изображения. Была создана нейросеть, которая после получения неудовлетворительных результатов была модифицирована. Получены хорошие результаты.
- Выявлены плюсы и минусы использования нейросетей для восстановления изображения.

Таким образом было установлено, что нейросети очень хорошо справляются с задачей восстановления изображения. Полученный результат возможно улучшить если увеличить обучающую выборку нейросети. Но поставленная задача была выполнена.

Список литературы

1) Глава 3. Основы ИНС [Электронный ресурс]. – Режим доступа: https://neural.radkopeter.ru/chapter/основы-инс/. – Дата доступа: 25.05.2022.
2) ФУНКЦИИ АКТИВАЦИИ НЕЙРОСЕТИ: СИГМОИДА, ЛИНЕЙНАЯ, СТУПЕНЧАТАЯ, RELU, TAHN [Электронный ресурс]. – Режим доступа: https://elar.rsvpu.ru/bitstream/123456789/28261/1/978-5-8295-0623-0_2019_031.pdf. – Дата доступа: 26.05.2022.
3) Библиотека Keras – Русскоязычная документация Keras [Электронный ресурс]. – Режим доступа: https://ru-keras.com/home/. – Дата доступа: 20.05.2022.
4) PYTORCH DOCUMENTATION [Электронный ресурс]. – Режим доступа: https://pytorch.org/docs/stable/index.html. – Дата доступа: 19.05.2022.
5) Нейронные сети [Электронный ресурс]. – Режим доступа: https://proproprogs.ru/neural_network. – Дата доступа: 06.06.2022.
6) Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/348000/. – Дата доступа: 01.06.2022.
7) Методы оптимизации нейронных сетей [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/318970/. – Дата доступа: 01.06.2022.
8) Метод обратного распространения ошибки: математика, примеры, код [Электронный ресурс]. – Режим доступа: https://neurohive.io/ru/osnovy-data-science/obratnoe-rasprostranenie/. – Дата доступа: 01.06.2022.
9) Минский, Марвин Ли [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Минский,_Марвин_Ли. – Дата доступа: 22.05.2022.
10) Гудфуллоу, Я. Глубокое обучение / Я. Гудфуллоу, И. Бенджио, А. Курвиль. – Москва : ДМК Пресс, 2018. – 653 с.

11) История возникновения нейронных сетей [Электронный ресурс]. – Режим доступа: https://pikabu.ru/story/istoriya_vozniknoveniya_neyronnyikh_setey_7152766. – Дата доступа: 24.05.2022.
12) Перцептрон [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Перцептрон. – Дата доступа: 29.05.2022.
13) Горбачевская, С.С. Краснов // История развития нейронных сетей // Вестник Волжского университета им. В. Н. Татищева. – 2015.
14) Реализуем и сравниваем оптимизаторы моделей в глубоком обучении [Электронный ресурс]. – Режим доступа: https://itnan.ru/post.php?c=1&p=525214. – Дата доступа: 25.05.2022.
15) Dropout — метод решения проблемы переобучения в нейронных сетях [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/wunderfund/blog/330814/?hl=ru_RU&fl=ru,en. – Дата доступа: 06.06.2022.

Приложение А

Создание и работа с нейронной сетью

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import cv2 as cv
import matplotlib.pyplot as plt
from keras.preprocessing import image
from keras.applications.inception_resnet_v2 import InceptionResNetV2
from keras.applications.inception_resnet_v2 import
preprocess_input,decode_predictions
from keras import backend as K
from keras.layers import add,
Conv2D,MaxPooling2D,UpSampling2D,Dropout, Dense,
Input,BatchNormalization, RepeatVector, Reshape
from keras.layers.merge import concatenate
from keras.models import Model
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
tf.random.set_seed(2)
np.random.seed(1)
from tensorflow import keras
model = keras.models.load_model("./Model_Enhancer")
Dropmodel = keras.models.load_model('./Model_Enhancer_2.0')
opmod = keras.models.load_model('./Model_Optimal')

def noisy(noise_typ,image):
if noise_typ == "gauss":
row,col,ch= image.shape
```

```
mean = 0
var = 0.0001
sigma = var**0.05
gauss = np.random.normal(mean,sigma,(row,col,ch))
gauss = gauss.reshape(row,col,ch)
noisy =  gauss + image
return noisy
elif noise_typ == "s&p":
row,col,ch = image.shape
s_vs_p = 0.5
out = np.copy(image)
# Salt mode
num_salt = np.ceil(image.size * s_vs_p)
coords = [np.random.randint(0, i, int(num_salt))
for i in image.shape]
out[coords] = 1

# Pepper mode
num_pepper = np.ceil(image.size * (1. - s_vs_p))
coords = [np.random.randint(0, i , int(num_pepper))
for i in image.shape]
out[coords] = 1
return out
elif noise_typ == "poisson":
vals = len(np.unique(image))
vals = 2 ** np.ceil(np.log2(vals))
noisy = np.random.poisson(image * vals) / float(vals)
return noisy
elif noise_typ =="speckle":
row,col,ch = image.shape
```

```
gauss = np.random.randn(row,col,ch)
gauss = gauss.reshape(row,col,ch)
noisy = image + image * gauss
return noisy
def PreProcessData(ImagePath):
X_=[]
y_=[]
count=0
for imageDir in os.listdir(ImagePath):
if count<3000:
try:
count=count+1
img = cv.imread(ImagePath + imageDir)
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
img_y = cv.resize(img,(500,500))
hsv = cv.cvtColor(img_y, cv.COLOR_BGR2HSV)
hsv[...,2] = hsv[...,2]*0.2
img_1 = cv.cvtColor(hsv, cv.COLOR_HSV2BGR)
Noisey_img = noisy("s&p",img_1)
X_.append(Noisey_img)
y_.append(img_y)
except:
pass
X_ = np.array(X_)
y_ = np.array(y_)

return X_,y_

K.clear_session()
def InstantiateModel(in_):
```

```
        model_1 = Conv2D(16,(3,3),
activation='relu',padding='same',strides=1)(in_)
        model_1 = Conv2D(32,(3,3),
activation='relu',padding='same',strides=1)(model_1)
        model_1 = Conv2D(64,(2,2),
activation='relu',padding='same',strides=1)(model_1)
        model_2 = Conv2D(32,(3,3),
activation='relu',padding='same',strides=1)(in_)
        model_2 = Conv2D(64,(2,2),
activation='relu',padding='same',strides=1)(model_2)
        model_2_0 = Conv2D(64,(2,2),
activation='relu',padding='same',strides=1)(model_2)
        model_add = add([model_1,model_2,model_2_0])
        model_3 = Conv2D(64,(3,3),
activation='relu',padding='same',strides=1)(model_add)
        model_3 = Conv2D(32,(3,3),
activation='relu',padding='same',strides=1)(model_3)
        model_3 = Conv2D(16,(2,2),
activation='relu',padding='same',strides=1)(model_3)
        model_3_1 = Conv2D(32,(3,3),
activation='relu',padding='same',strides=1)(model_add)
        model_3_1 = Conv2D(16,(2,2),
activation='relu',padding='same',strides=1)(model_3_1)
        model_3_2 = Conv2D(16,(2,2),
activation='relu',padding='same',strides=1)(model_add)
        model_add_2 = add([model_3_1,model_3_2,model_3])
        model_4 = Conv2D(16,(3,3),
activation='relu',padding='same',strides=1)(model_add_2)
```

```
        model_4_1 = Conv2D(16,(3,3),
activation='relu',padding='same',strides=1)(model_add)
        model_add_3 = add([model_4_1,model_add_2,model_4])
        model_5 = Conv2D(16,(3,3),
activation='relu',padding='same',strides=1)(model_add_3)
        model_5 = Conv2D(16,(2,2),
activation='relu',padding='same',strides=1)(model_add_3)
        model_5 = Conv2D(3,(3,3),
activation='relu',padding='same',strides=1)(model_5)

        return model_5
        K.clear_session()
        def InstantiateModel(in_):

            model_1 = Conv2D(16,(3,3),
activation='relu',padding='same',strides=1)(in_)
            model_1 = Conv2D(32,(3,3),
activation='relu',padding='same',strides=1)(model_1)
            model_1 = Dropout(0.4)(model_1)
            model_1 = Conv2D(64,(2,2),
activation='relu',padding='same',strides=1)(model_1)
            model_2 = Conv2D(32,(3,3),
activation='relu',padding='same',strides=1)(in_)
            model_2 = Dropout(0.4)(model_2)
            model_2 = Conv2D(64,(2,2),
activation='relu',padding='same',strides=1)(model_2)
            model_2_0 = Conv2D(64,(2,2),
activation='relu',padding='same',strides=1)(model_2)
            model_2_0 = Dropout(0.4)(model_2_0)
            model_add = add([model_1,model_2,model_2_0])
```

```
        model_3 = Conv2D(64,(3,3),
activation='relu',padding='same',strides=1)(model_add)
        model_3 = Conv2D(32,(3,3),
activation='relu',padding='same',strides=1)(model_3)
        model_3 = Dropout(0.4)(model_3)
        model_3 = Conv2D(16,(2,2),
activation='relu',padding='same',strides=1)(model_3)
        model_3 = Dropout(0.4)(model_3)
        model_3_1 = Conv2D(32,(3,3),
activation='relu',padding='same',strides=1)(model_add)
        model_3_1 = Conv2D(16,(2,2),
activation='relu',padding='same',strides=1)(model_3_1)
        model_3_1 = Dropout(0.4)(model_3_1)
        model_3_2 = Conv2D(16,(2,2),
activation='relu',padding='same',strides=1)(model_add)
        model_3_2 = Dropout(0.4)(model_3_2)
        model_add_2 = add([model_3_1,model_3_2,model_3])
        model_4 = Conv2D(16,(3,3),
activation='relu',padding='same',strides=1)(model_add_2)
        model_4 = Dropout(0.4)(model_4)
        model_4_1 = Conv2D(16,(3,3),
activation='relu',padding='same',strides=1)(model_add)
        model_add_3 = add([model_4_1,model_add_2,model_4])
        model_5 = Conv2D(16,(3,3),
activation='relu',padding='same',strides=1)(model_add_3)
        model_5 = Dropout(0.4)(model_5)
        model_5 = Conv2D(16,(2,2),
activation='relu',padding='same',strides=1)(model_add_3)
        model_5 = Dropout(0.4)(model_5)
```

```
            model_5 = Conv2D(3,(3,3),
activation='relu',padding='same',strides=1)(model_5)
    return model_5
        Input_Sample = Input(shape=(500, 500,3))
        Output_ = InstantiateModel(Input_Sample)
Model_Enhancer = Model(inputs=Input_Sample, outputs=Output_)
        Model_Enhancer.compile(optimizer="adam", loss='mean_absolute_error')
Model_Enhancer.summary()

        from keras.utils.vis_utils import plot_model
        plot_model(Model_Enhancer,to_file='model.png',show_shapes=True,
show_layer_names=True)
        from IPython.display import Image
Image(retina=True, filename='model_.png')

        def GenerateInputs(X,y):
            for i in range(len(X)):
                X_input = X[i].reshape(1,500,500,3)
                y_input = y[i].reshape(1,500,500,3)
                yield (X_input,y_input)
opmod.fit(GenerateInputs(X_,y_),epochs=3,verbose=1,steps_per_epoch=1000,shu
ffle=True)

        Image_test=TestPath+"91875542.jpg"
        plt.figure(figsize=(30,30))
        plt.subplot(5,5,1)
        img_1 = cv.imread(Image_test)
        img_1 = cv.cvtColor(img_1, cv.COLOR_BGR2RGB)
        img_1 = cv.resize(img_1, (500, 500))
        plt.title("Ground Truth",fontsize=20)
```

```
        plt.imshow(img_1)

        plt.subplot(5,5,1+1)
        img_ = ExtractTestInput(Image_test)
        Prediction = opmod.predict(img_)
        img_ = img_.reshape(500,500,3)
        plt.title("Low Light Image",fontsize=20)
        plt.imshow(img_)

        plt.subplot(5,5,1+2)
        img_[:,:,:] = Prediction[:,:,:]
        plt.title("Enhanced Image",fontsize=20)
plt.imshow(img_)
```