МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»
(СПбГЭУ)

Факультет информатики и прикладной математики
Кафедра прикладной математики и экономико-математических методов

ОТЧЕТ по

производственной практике: научно-исследовательская работа

Наименование организации прохождения практики: ООО «ПЛАЗ»

Направление: 01.03.02 Прикладная математика и информатика

Направленность (профиль): Прикладная математика и информатика в экономике и управлении

Обучающийся: Широков Александр Анатольевич

Группа: ПМ-1701 Подпись 

Руководитель практики от СПбГЭУ:

Вилло Надежда Юрьевна, старший преподаватель

_____
(подпись руководителя)

Оценка по итогам защиты отчёта_____ _____
_____
_____

(подписи членов комиссии (при наличии))

Санкт-Петербург
2020 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3

1. ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ ПО ПРАКТИКЕ 4

1.1. Предобработка изображений 5

1.1.1. Изображение в памяти компьютер 5

1.1.2. Представление изображения в оттенках серого 5

1.1.3. Представление изображения RGB 6

1.2. Формирование тестовой и обучающей выборки 7

1.2.1. Rescaling изображений 7

1.2.2. Image Pyramids 8

1.3. Convolutional Neural Network 9

1.3.1. Strides 12

1.3.2. Padding 12

1.3.3. Pooling 13

1.4. Image to Image 14

1.4.1. Функция потерь 14

1.4.2. Архитектуры нейронной сети 14

1.5. Convolutional Autoencoders 16

1.6. Визуализация результатов 20

ЗАКЛЮЧЕНИЕ 21

СПИСОК ЛИТЕРАТУРЫ 22

ВВЕДЕНИЕ

В период с 9 июля 2020 года по 22 июля года я, Александр Широков, проходил производственную практику с выполнением научной-исследовательской работы в компании ООО «ПЛАЗ». В связи с эпидемиологической обстановкой в стране и мире данная практика проходила в удалённом формате.

Фирма ООО «ПЛАЗ» специализируется на разработке и производстве беспилотных авиационных (БАС) систем самолётного и коптерного типа, а также разработкой программного обеспечения для обработки фото и видеоинформации, получаемой БАС.

Практику в компании я проходил в качестве разработчика алгоритмов для эффективной обработки фото. Этому и было посвящено моё индивидуальное задание, о котором речь пойдёт ниже.

Основная цель учебной производственной практики – формирование практических навыков и умений, необходимых в будущем.

Основными задачами учебной производственной практики являлись:

1. Выполнение индивидуального задания с возможностью наглядной визуализации полученных результатов решения задачи

2. Получение глубокого понимания работы алгоритма/алгоритмов для решения индивидуального задания

3. Отработка навыков и знаний, полученных в университете и умение их применять в сжатые временные сроки

Актуальность практики состоит в формировании карьерного портфолио студента – после окончания университета всё больше появляется заинтересованность у работодателей в тех выпускниках, которые имеют не только теоретические знания по теме, но и применяли данные знания на практике.

1. ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ ПО ПРАКТИКЕ

Моим индивидуальным заданием являлась следующая задача: разработка алгоритмов для повышения тепловизионного изображения низкого разрешения (32x32).

Задача *Super Resolution* (улучшения изображения) является важной, но тем не менее уже решённой задачей в области задачи компьютерного зрения. Данная задача относится к категории Image-to-Image: в качестве входных данных должна подаваться определенная фотография небольшой размерности, а на выходе – фотография, не потерявшая свойства входящей фотографии, обработанная и в лучшем разрешении.

Данная объёмная задача была разбита мной на следующие подзадачи, с которыми я последовательно разбирался во время практики:

1. Способы обработки изображений и их представление в памяти компьютера

2. Формирование архива изображений – высокого разрешения и преобразование изображение высокого разрешения в низкое для увеличения размера обучающей выборки

3. Изучение принципов и архитектур свёрточных нейронных сетей для работы с изображениями

4. Формирование архитектуры AutoEncoder, позволяющая улучшать качество изображения

5. Построение архитектуры нейронной сети AutoEncoder, обучение модели на сформированной обучающей выборке и отображение полученных результатов

6. Изучение «гауссовских пирамид» - преобразований, повышающих размерность уже улучшенных изображений в несколько раз

Для реализации архитектур нейронных сетей использовался язык Python и библиотека для глубокого обучения Keras. Для сравнения результатов работы гауссовских пирамид была использована библиотека для компьютерного зрения OpenCV.

1.1. Предобработка изображений

Для того, чтобы работать с изображениями, рассмотрим некоторые фундаментальные теоретические основы работы с изображениями и их представлением в памяти компьютера.

1.1.1. Изображение в памяти компьютер

Любое изображение необходимо представлять в памяти компьютера. Известно, что каждое изображение состоит из *пикселей* – наименьших элементов, формирующих наше исходное изображение. Чем больше пикселей содержится на единицу площади в изображении, тем более оно детально. Изображение, представляющее собой сетку пикселей, называется *растровым.*

Размерность растровых изображений выражают в виде количества пикселей по горизонтали и вертикали: например 1600×1200. В дальнейшем за W (width) примем количество пикселей по горизонтали, длину изображения, а за H (height) – количество пикселей по вертикали, ширину изображения. Так как пиксел является мельчайшей частью изображения, то его *способ инициализации* задаёт изображение в памяти компьютера.

1.1.2. Представление изображения в оттенках серого

Для представления одного пикселя в памяти компьютера было придумано несколько способов.

Grayscale (оттенки серого) – цветовой режим изображения, имеющий следующую конструкцию инициализации: каждый пиксел отображает оттенок серого цвета. В свою очередь серый цвет является яркостью белого цвета.

Та как было определено, что изображение будет храниться в виде матрицы размерности $H \times W$, то необходимо определиться со способом кодирования элементов данной матрицы в цветовом режиме оттенков серого.

В компьютерном представлении была придумана и используется *серая шкала,* которая на каждый пиксел изображения использует 1 байт (8 бит) информации. Такая шкала передаёт $2^8 = 256$ оттенков серого цвета или яркости. Значение 0 – *чёрный цвет*, а значение 255 – *белый цвет*.

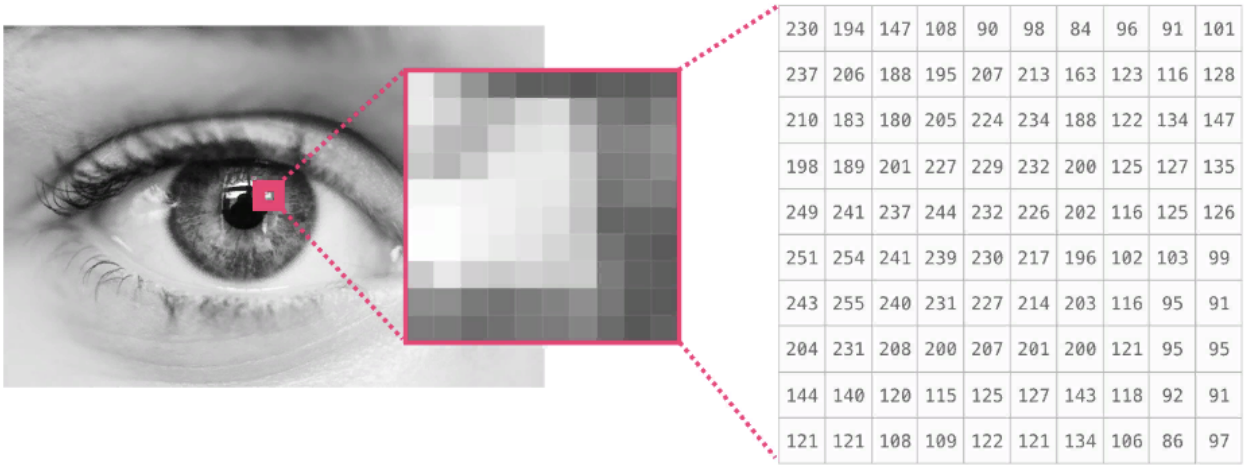

Рисунок 1 – представление изображения в цветовой схеме Grayscale

1.1.3. Представление изображения RGB

RGB – цветовое представление изображения, в котором каждый пиксел является комбинацией трёх цветов – красного, зелёного и синего. Так же, как и в случае цветовой гаммы Grayscale, каждый пиксел изображения представлен с помощью 3-х восьмибитных чисел, соответствующих Red (красный), Green (зелёный) и Blue (синий) цветам. Соответственно, вся совокупность пикселей даёт полное цветное изображение в цветовом представлении RGB.

Входная матрица изображения является при данном виде представления *трёхмерным тензором* и будет иметь размерность $H \times W \times 3$.

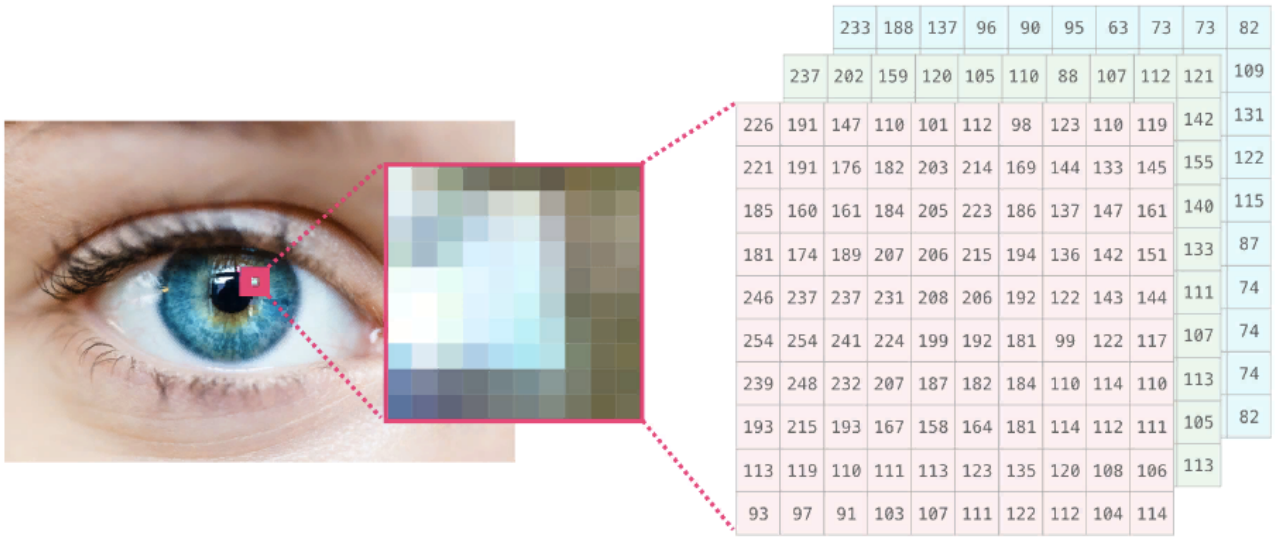

Рисунок 2 – представление изображения в виде RGB цветовой схемы

В условиях поставленной задачи поступающие на вход изображения с тепловизора будут представлены в виде цветовой схемы Grayscale. Было бы удобно иметь способ перевода одной цветовой схемы в другую.

Существует много способов перехода, но я буду использовать следующий способ, являющий алгоритмом нахождения взвешенного среднего RGB-значения яркости цвета:

$$Y = 0.299R + 0.587G + 0.114B$$

1.2. Формирование тестовой и обучающей выборки

1.2.1. Rescaling изображений

После того, как изображение было представлено в памяти компьютера, моей задачей являлось формирование обучающей выборки. Так как задача улучшения качества изображения в качестве выходных данных должна выводить улучшенное изображение, то было решено сформулировать задачу глубокого обучения следующим образом.

Сначала сформируем изображения, которые будут являться идеалом для модели – цветные изображения в отличном разрешении, без зашумления или других побочных эффектов. Ввиду наличия у компании высокотехнологичных фотокамер, данный этап не вызывает каких-то трудностей. Для обучения в упрощённом варианте я использовал цветные фотографии животных, машин из открытого репозитория фотографий ImageNet.

Второй шаг – преобразование данных идеальных изображений в оттенки серого с помощью формулы, введённой в предыдущем параграфе. Так мы сможем перейти к исходной постановке задачи.

Теперь применим следующую идею: раз задача состоит в улучшении изображения, то давайте в качестве входа модели подавать ухудшенные идеальные изображения, чтобы модель смогла обучиться восстанавливать из ухудшенного состояния идеальное. Ухудшение изображения можно получить с помощью операции *rescale*.

Преобразование и формирование архива фотографий будет выглядеть следующим образом

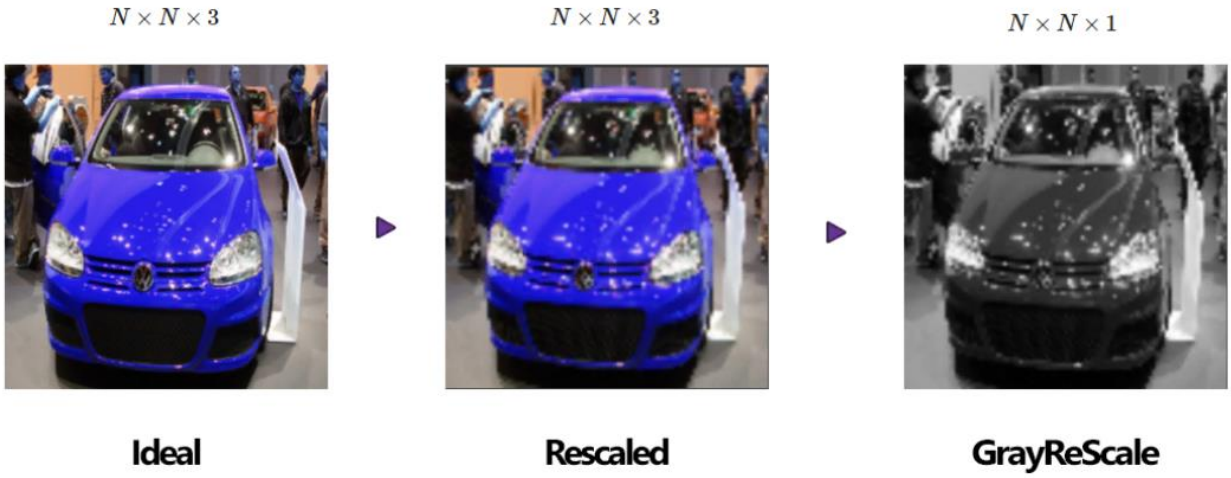

Рисунок 3 – формирование выборки для обучения модели

Заметим, что для обеспечения значимости результатов, выходное изображение (идеальное) изначально должно быть изначально достаточно большой размерности. Глупо обучать модель на идеальном изображении 32×32. Для определённости будем считать, что мы можем привести исходные фотографии к размерности $N = 256$.

Тогда возникает следующая проблема: исходное изображение тепловизора 32×32 и модель не сможет предсказать по данному изображению улучшенное просто-напросто из-за неправильной входящей размерности.

1.2.2. Image Pyramids

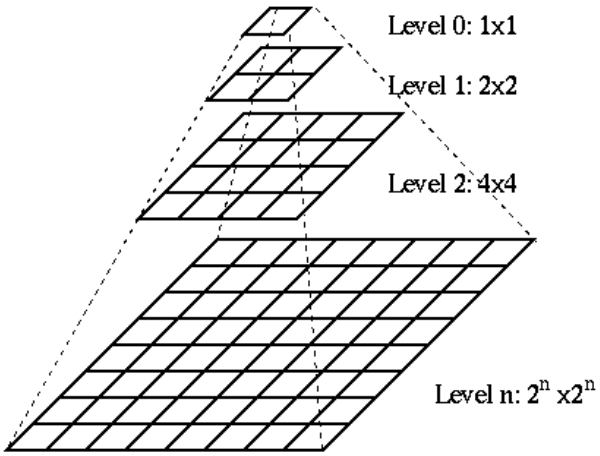

Рисунок 4 – Image Pyramids

Пирамиды изображений используются по той причине, что крупные детали изображения лучше видны на изображениях с небольшим разрешением, а мелкие детали изображение проявляются только на изображениях с высоким разрешением. Пирамида изображений преследует следующие цели – сокращение времени обработки изображений и определение более точных начальных приближений для обработки нижних уровней по результатам обработки верхних уровней изображения.

Пирамида изображений представляет собой последовательность N изображений, каждое из которых в два раз меньше/больше предыдущего и размерность является степенью двойки.

Процесс построения пирамид использует фильтрацию изображения ядром (kernel), с которым мы познакомимся более подробно в следующем параграфе.

Таким образом, мы можем передвигаться по ступеням пирамиды изображений и выбирать ту размерность, которая нам подойдёт. Сделав 3 обратных преобразования над изображением 32×32, получим изображение размерностью 256×256, которое уже может улучшать построенная нами в будущем модель.

1.3. Convolutional Neural Network

Для построения финальной архитектуры модели для решения задачи Super Resolution необходимо разобраться с одной из простейших моделей в классе моделей глубокого обучения – *свёрточные нейронные сети.*

Объяснение, почему свёрточные нейронные сети очень здорово работают с изображениями весьма простое. Рассмотрим, например, полносвязную нейронную сеть. На вход данной сети подаётся некоторый вектор признаков, который на каждом слое, но сначала на первом, при Forward Pass умножается на матрицу весов, совершающее преобразование линейного пространства в другое пространство признаков. При данном алгоритме распространения, мы предполагаем, что все новые признаки линейно зависят от предыдущих, а главное, что так как мы применяем линейное преобразование, то мы предполагаем, что все **признаки**

независимы в исходном векторе, иначе матрица будет вырожденной и весь процесс застопорится.

Ещё один минус полносвязных линейных архитектур при работе с изображениями – отсутствие инвариантности. Действительно, допустим на изображении нарисована кошечка. Подвинув кошечку вправо на несколько пикселей или повернув её на картинке кошечка все так же будет присутствовать, а вот вектор признаков, характеризующий данную кошку в пространстве изображения поменяется, что приведёт к проблемам задачи классификации (и не только).

При работе с изображениями, мы полагаем, что каждый пиксел точно *не независим* от остальных, потому что вся информация изображения зависит от пикселей изображения.

Предположим, что нас волнует **свойство локальности –** пиксели как-то упорядочены друг с другом, а также сделаем предположение о том, что нас волнуют только те пиксели, которые находятся *рядом.*

На данных двух свойствах и построены архитектуры и идеи свёрточных нейронных сетей. Для понимания работы свёрточных нейронных сетей введём несколько математических операций.

Операция свёртки – грубо говоря – матричное умножение на матрицу весов определенного фильтра и имеет обозначение $\otimes$. Для должного понимания рассмотрим эту операцию на примере.

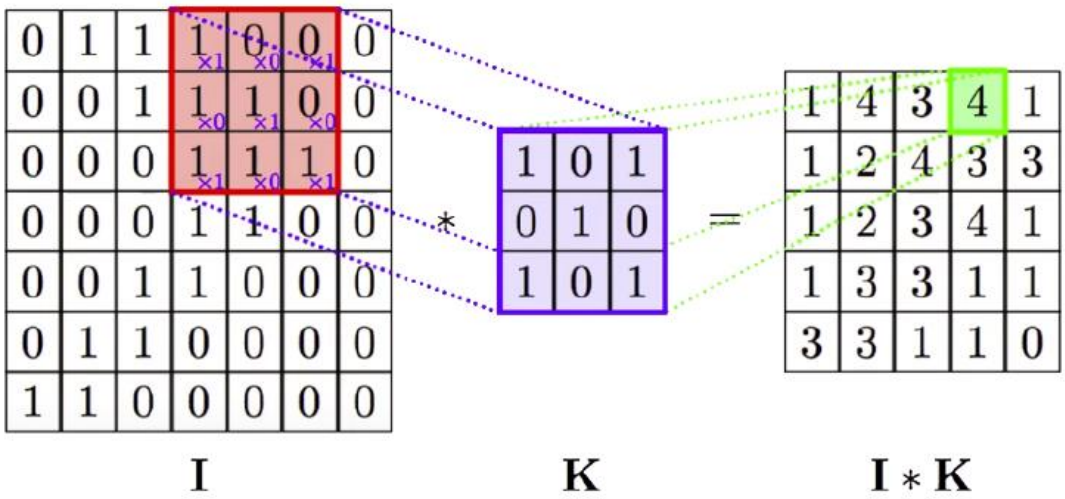

Рисунок 5 – операция свёртки

Операция свёртки для двумерной матрицы имеет следующий вид – мы накладываем определённый паттерн на картинку (фильтр) размерности $m \times m$ (в данном случае $m = 3$) и высчитываем количество попаданий в данный паттерн определенного маленького участка картинки.

Данное преобразование мы можем делать с кучей фильтров $W_1, \dots, W_n$ и мы сможем сделать N различных преобразований над нашим изображением и данные преобразования будут *локальными* – то есть они будут сделаны только с рядом стоящими пикселями

Теперь рассмотрим, что же происходит для изображений. Как мы уже определили ранее, каждое изображение задаётся тензором $H \times W \times N$, где $N = 3$ для RGB и $N = 1$ для Grayscale. Тогда нам необходимо свернуть трёхмерный тензор из карт, поэтому фильтр свёртки будет являться не матрицей, а тензором $3 \times 3 \times N$, где N – количество карт в изображении.

Операция абсолютно идентичная предыдущей – мы производим операцию применения фильтра, кладя его в нейрон следующего слоя. С помощью одного фильтра мы получаем одну карту, следовательно для того чтобы получить несколько карт на следующем слое, нужно применить K фильтров.

Тогда для того, чтобы определить слой свёрточной нейронной сети, нам необходимо применить тензор размерности $3 \times 3 \times N \times K$.

В этом и состоит основная идея свёрточных нейронных сетей – параметрами модели являются **фильтры**, а именно веса, из которых они состоят – тем самым свёрточная нейронная сеть учится искать важные признаки, уменьшая размерность изображения с каждым свёрточным слоем. Сеть формирует некоторые абстрактные признаки нашего изображения. Обучение сети происходит так же, как и в полносвязном перцептроне – с помощью алгоритма обратного распространения ошибки.

Заметим, что свёртки очень кстати были придуманы для уменьшения размерности изображения и выделения признака. Действительно, допустим, что мы хотим перевести изображение $\mathrm{A} = 150 \times 150 \times 3$ в $B = 148 \times 148 \times 3$. Тогда

матрица перехода должна иметь размерность $A \times B$ – мы имеем миллиард параметров на одно преобразование, а при свёрточных нейронных сетях их гораздо меньше.

Рассмотрим несколько преобразований, с помощью которых возможно снижать исходную размерность изображения.

1.3.1. Strides

Фильтре можно применять не ко всем данным, а с некоторым шагом. Данный параметр называется *stride*.

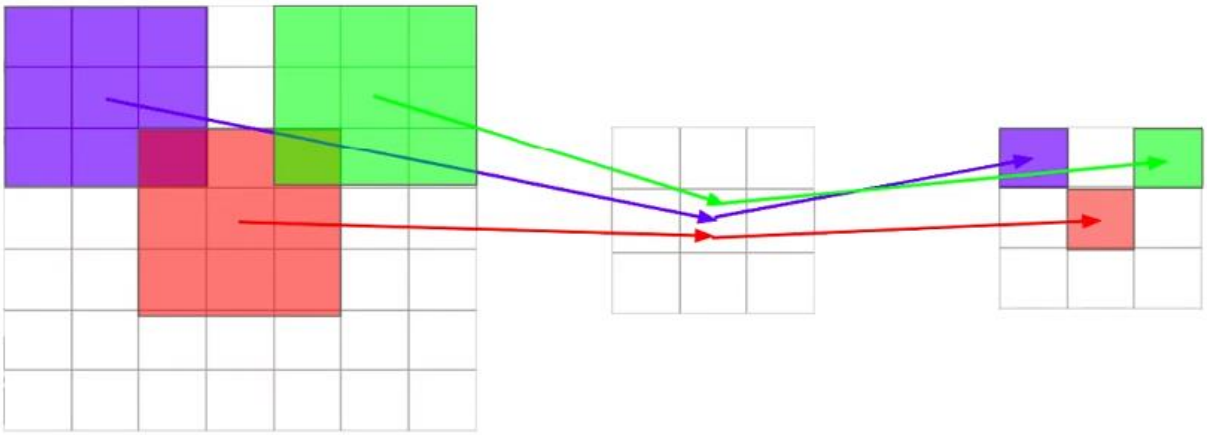

Рисунок 6 – Stride – уменьшение размерности, параметр свёрточного слоя

Стоит ясно осознавать, что при уменьшении размерности изображения *любыми методами*, мы **теряем информацию из изображения**, однако переобучение происходит в гораздо меньших масштабах. Это замечание мы используем для обоснования идеи при построении архитектуры сети Super Resolution.

1.3.2. Padding

При применении операции *Padding* можно сделать искусственное увеличение изображений, в результате применения свёртки к которому мы сможем не потерять размерность исходного изображения.

Padding бывает двух видов:

- Same – сохраняет размер изображения
- Valid – не использует padding, обрабатываем ту часть картинки, которая по-настоящему существует

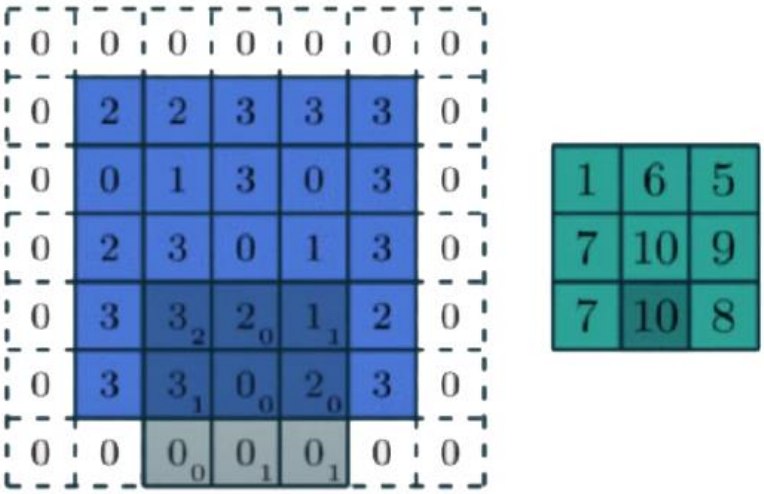

Рисунок 7 – Padding = 1, Stride = 2

1.3.3. Pooling

Данная операция является по-настоящему важной – из локального пространства в изображении мы вытягиваем наибольший элемент (MaxPooling), или мы хотим, чтобы наш паттерн работал для всех пикселей внутри локального пространства, тогда мы заменяем значение на среднее следующим образом: мы выбираем из каждого квадрата максимальное (среднее число) и уменьшаем размерность в k раз, где k – размерность фильтра.

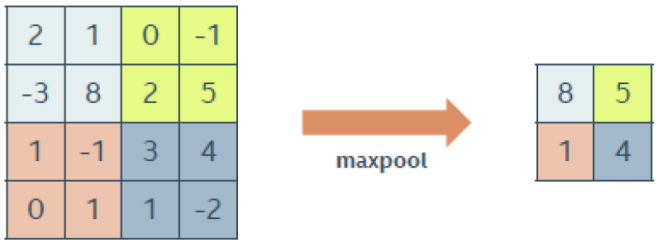

Рисунок 8 – Max Pooling

Данный слой не имеет параметров, что удобно. Числа, получающие при Pooling – мера присутствия какого-то элемента в данной части изображения.

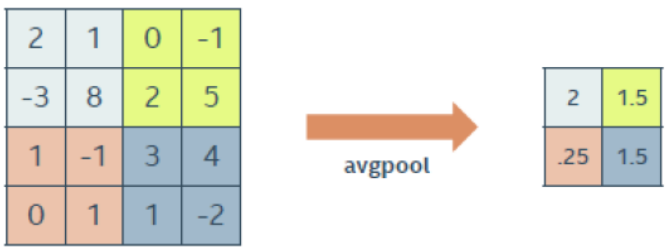

Рисунок 9 - Average Polling

1.4. Image to Image

Задача Image to Image формулируется следующим образом: на вход нам подаётся какое-то изображение и мы хотим предсказать другое изображение на выходе. Super Resolution – одна из таких задач.

Рассмотрим классический способ решения задачи, который приходит в голову и попытаемся перевести на язык задачи машинного обучения.

1.4.1. Функция потерь

У нас есть картинка на входе, будем обозначать x_i и картинка на выходе y_i, тогда возьмём нейронную сеть и будем по x_i предсказывать y_i. Будем оптимизировать некоторую функцию потерь – например разницу между пикселями картинки, которая является выходной и предсказанной нейросетью в ходе очередного Backward Pass- будем суммировать по всем каналам, всем размерностям и считать ошибку по метрике. Такими метриками является среднеквадратичное отклонение или абсолютная ошибка:

$$MSE\big(G(x_i, y_i)\big) = \frac{1}{3HW} \cdot \sum_{c=1}^{3} \sum_{h=1}^{H} \sum_{w=1}^{W} \big(y_i^{hwc} - G(x_i)^{hwc}\big)^2$$

$$MAE\big(G(x_i, y_i)\big) = \frac{1}{3HW} \cdot \sum_{c=1}^{3} \sum_{h=1}^{H} \sum_{w=1}^{W} |y_i^{hwc} - G(x_i)^{hwc}|$$

Та и та функция применяется, но модуль сложнее оптимизировать, однако он несёт большую смысловую составляющую и даёт MAE более точные результаты.

1.4.2. Архитектуры нейронной сети

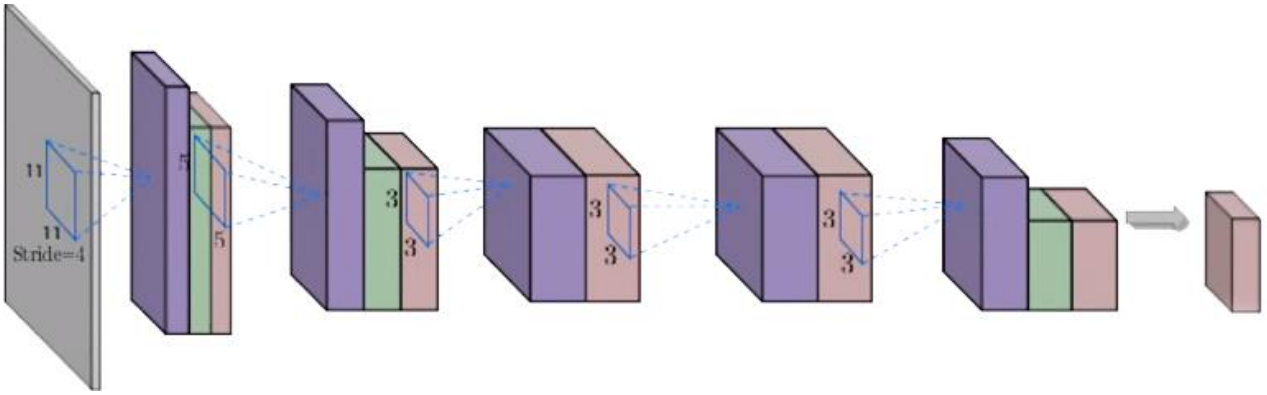

Рисунок 10 – архитектура нейронной сети для задач классификации

Для задач классификации нейронная сеть состояла из несколько свёрточных слоёв, нормализации батчей, нелинейности и слоёв, которые понижали размерности. Снижение размерности позволяла снижать вычислительные мощности, позволяя добавлять большее число каналов, а также увеличивала Receptive Field нейрона – на какую область изображения смотрит каждый последующий слой сети, позволяющая выучивать более глобальные признаки нейронной сети. В конце концов снижалась размерность, все вытягивалось в вектор и по результатам предсказывали принадлежность классам.

В задаче Image-to-Image необходимо предсказывать изображение – выходная размерность должна быть такой же, что и входная.

При этом, если мы будем использовать свёрточную архитектуру, как на рисунке 10, то все меньше информации будет в сжатом изображении внутри сети.

Поэтому приходит на ум следующая идея: нам необходимы слои, которые будут **увеличивать размерность изображения.**

Соответственно можно предложить такую архитектуру нейронной сети – каждый слой является свёрткой, сначала будет размерность понижаться, а потом будет увеличиваться до исходной. Осталось понять, какие слои будут увеличивать размерность.

Рассмотрим примеры таких слоёв.

Upsampling – допустим на вход подаётся карта размерности 2×2, тогда слой *upsampling* переводит в матрицу 4×4, продублировав ближайшую клетку ближайшим соседом.

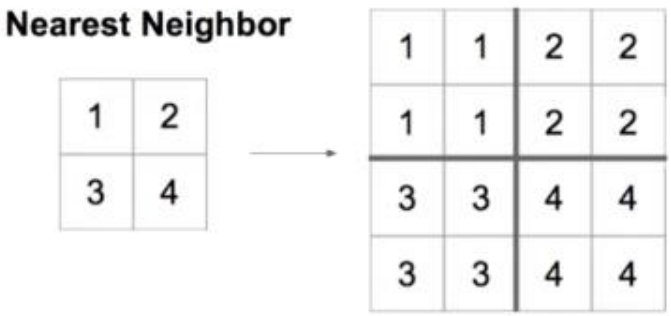

Рисунок 11 – Upsampling – *Nearest Neighbor*

Существует и другой способ – *Bed of Nails.*

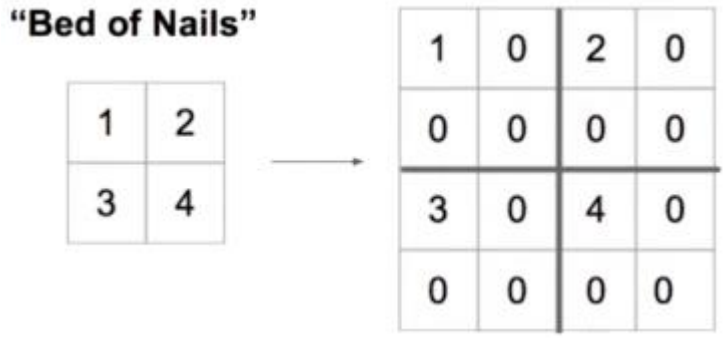

Рисунок 12 – Upsampling – *Bed of Nails*

Но более популярным подходом является транспонированная свёртка. На вход, допустим, подаётся карта размером 2×2 и фильтром размера 3×3. Транспонированная свёртка работает ровно наоборот: мы перемножаем значение ячейки на каждое число в фильтре, но результат мы не складываем, а записываем в соответствующую окрестность выхода. Таким образом можно увеличивать размеры карт в 2 раза и обучать веса.

Рисунок 13 – транспонированная свёртка

Итого, мы рассмотрели некоторые метода увеличения размерности. Перейдём к финальному построению архитектуры сети для задачи решения Super-Resolution.

1.5. Convolutional Autoencoders

Автоенкодеры являются тождественной функцией в глубоком обучении. Это нейронная сеть сочетает все идеи, которые были рассмотрены в предыдущих параграфах. Структурируем полученные идеи.

Модель автоенкодера состоит из двух главных частей:

- **Encoder –** предназначен для снижения размерности и преобразует исходное изображение в изображение меньшей размерности с помощью свёрточных слоёв
- **Decode –** пытается реконструировать изображение из пространства меньшей размерности в исходное изображение, является свёрточными UpSampling слоями.
- **Loss function –** как и рассматривалось ранее в качестве функции потерь используется попиксельная разница между идеальным изображением и реконструированным изображением.

Автоенкодеры в задаче Super-Resolution пытаются научиться восстанавливать из испорченного изображения улучшенное, обучаясь на функции потерь.

Рассмотрим теперь не идейную составляющую енкодера, а её наглядную архитектуру.

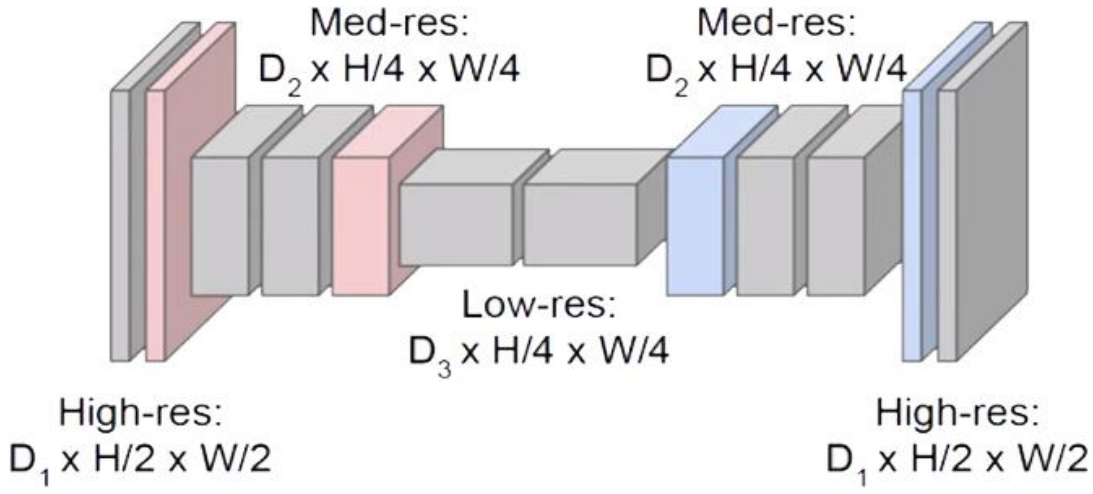

Рисунок 14 – архитектуре AutoEncoder

Сначала на вход приходит изображение, мы применяем к нему свёрточные слои, батч-нормализации, у нас выучивается представление изображения и мы его декодируем с помощью слоёв изображения декодера. Финальный свёрточный слой имеет один фильтр, так как оригинальное изображение подаётся в GrayScale. Её выходы мы интерпретируем как выходное изображение.

Но у данное архитектуры есть огромный недостаток. Если подумать, то данная архитектура должна суметь закодировать входное изображение в какое-

то внутреннее представление, но чем глубже сверточный слой, тем меньше размерность изображения и тем меньше информации содержится в нём – тем больше **глобальных признаков** содержится изображении. Задачей же декодера является восстановление *очень локальные детали* и это сложно сделать, когда представление хранит только глобальные признаки изображения.

Поэтому в эту архитектуру **Skip-Connection:**

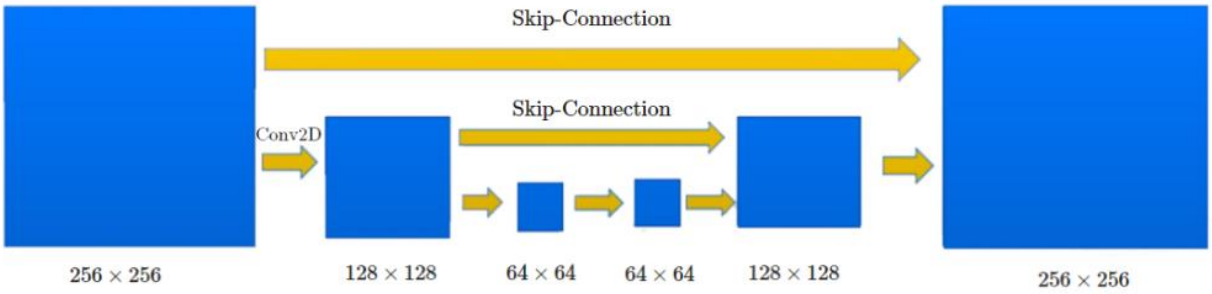

Рисунок 15 – Skip Connection

Skip Connection – это тождественная операция, которая конкатенирует или суммирует выходы слоёв свёртки и слоя после Upsampling. У данного преобразования «под капотом» весьма красивая идея: после того, как декодер реконструирует изображение путём UpSampling он сохранил некоторые глобальные признаки изображения и к данному изображению добавляется информация о более локальных признаках, полученных до применения свёртки.

Такая архитектура сети носит название Unet – в последнем блоке приходят не только информация о низкоуровневых признаках изображения, но и о глобальных. Данную структуру нейронной сети мы будем использовать для задачи Super-Resolution.

Для построения данной архитектуры использовался язык Python и библиотека для глубокого обучения Keras. В качестве оптимизатора использовался Adam: Adaptive Momentum: данный метод имеет более устойчивую сходимость из-за корректировки градиента и корректировки градиента и выглядит он следующим образом:

$$v_{t+1} = \gamma v_t + (1 - \gamma)\nabla f(x_t)$$

$$cache_{t+1} = \beta \cdot cache_t + (1-\beta)\left(\nabla f(x_t)\right)^2$$

$$x_{t+1} = x_t - \alpha \cdot \frac{v_{t+1}}{cache_{t+1} + \varepsilon}$$

Архитектура нейронной сети на Keras выглядит следующим образом:

```
Model: "Super_resolution"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
InputImage (InputLayer)      [(None, 256, 256, 1) 0
_________________________________________________________________________________________________
Convolutional_layer1_3x3 (Conv2 (None, 256, 256, 64) 640         InputImage[0][0]
_________________________________________________________________________________________________
Convolutional_layer2_3x3 (Conv2 (None, 256, 256, 64) 36928       Convolutional_layer1_3x3[0][0]
_________________________________________________________________________________________________
Maxpooling_2x2 (MaxPooling2D)   (None, 128, 128, 64) 0           Convolutional_layer2_3x3[0][0]
_________________________________________________________________________________________________
conv2d_51 (Conv2D)           (None, 128, 128, 128 73856       Maxpooling_2x2[0][0]
_________________________________________________________________________________________________
conv2d_52 (Conv2D)           (None, 128, 128, 128 147584      conv2d_51[0][0]
_________________________________________________________________________________________________
max_pooling2d_8 (MaxPooling2D)  (None, 64, 64, 128)  0           conv2d_52[0][0]
_________________________________________________________________________________________________
conv2d_53 (Conv2D)           (None, 64, 64, 256)  295168      max_pooling2d_8[0][0]
_________________________________________________________________________________________________
up_sampling2d_12 (UpSampling2D) (None, 128, 128, 256 0           conv2d_53[0][0]
_________________________________________________________________________________________________
conv2d_54 (Conv2D)           (None, 128, 128, 128 295040      up_sampling2d_12[0][0]
_________________________________________________________________________________________________
conv2d_55 (Conv2D)           (None, 128, 128, 128 147584      conv2d_54[0][0]
_________________________________________________________________________________________________
Skip_Connection_128 (Add)    (None, 128, 128, 128 0           conv2d_55[0][0]
                                                              conv2d_52[0][0]
_________________________________________________________________________________________________
up_sampling2d_13 (UpSampling2D) (None, 256, 256, 128 0           Skip_Connection_128[0][0]
_________________________________________________________________________________________________
conv2d_56 (Conv2D)           (None, 256, 256, 64) 73792       up_sampling2d_13[0][0]
_________________________________________________________________________________________________
conv2d_57 (Conv2D)           (None, 256, 256, 64) 36928       conv2d_56[0][0]
_________________________________________________________________________________________________
Skip_Connection_256 (Add)    (None, 256, 256, 64) 0           conv2d_57[0][0]
                                                              Convolutional_layer2_3x3[0][0]
_________________________________________________________________________________________________
To_output_image (Conv2D)     (None, 256, 256, 1)  577         Skip_Connection_256[0][0]
=================================================================================================
Total params: 1,108,097
Trainable params: 1,108,097
Non-trainable params: 0
_________________________________________________________________________________________________
```

Рисунок 16 – архитектура нейронной сети на Keras

Модель на протяжение часа обучалась на 256-размерных батчах, каждый раз дообучаясь на новых данных, не встречающихся до этого. Так как не было данных с тепловизора, то модель обучалась на данных ImageNet с котиками.

Перейдём к визуализации полученных результатов обучения модели на данном архиве фотографий.

1.6. Визуализация результатов

После обучения модели я попросил её предсказать результаты на изображении, которое не находилось в датасете и получил следующий результат

Рисунок 17 – поданное на вход изображение и улучшенное

У данного котика хорошо прорисовались зубки, однако фон восстановить не удалось, размытые изображения не удаётся обрабатывать – модель способна вылавливать некоторые признаки из исходных данных, но неспособна фантазировать – размытости так и остаются на изображениях, а значит модель необходимо улучшать.

Модель научилась улучшать кошек, но что произойдёт, если модели дать на вход, например, увеличенный с помощью гауссовских пирамид автомобиль?..

Рисунок 18 – модель обучена лишь на котиках

К сожалению, модель опять не восстановила запылённость, обучившись лишь на котиках она не смогла обучиться на машинах, а жаль.

ЗАКЛЮЧЕНИЕ

Несмотря на то, что реализованная архитектура нейронной сети на тестовом датасете даёт необходимые результаты, данная модель является устаревшей и в последнее время всё большее внимание отводится генеративным состязательным моделям (GAN) или более сложным архитектурам (HRNet) ввиду их способности обучения без запоминания идеального изображения. Однако, свёрточные нейронные сети, хоть и на ранних слоях занимают много памяти, работают гораздо быстрее GAN моделей, которым нужна мощность, отсутствующая на микропроцессорах дрона.

За время производственной практики я научился работать с полносвязными и свёрточными сетями, изучил необходимую теорию для понимания и улучшения их работы и реализовал архитектуру нейронной сети, хотя до начала практики даже не представлял, как это работает. Также данный алгоритм позволил мне впервые в жизни войти в Top-10 в соревновании по машинному обучению на Kaggle.

Недостаток времени и знаний для реализации более сложных архитектур, а также тренировочных данных с дронов немного разочаровывают. Но зато видны способы улучшения моделей: построение универсальной модели (данная модель была натренирована на отдельном классе, а хотелось бы, чтобы модель могла улучшать любое изображение); замена UpSampling слоёв на транспонированные свёртки для увеличения локальной информации на увеличивающих свёрточных слоях, разработка алгоритмов не только для квадратного изображения – данные проблемы говорят о том, что у данной задачи большое будущее для экспериментов и нахождения оптимального решения. Возможно, требуется обучить модель на более большом по объёме сете изображений.

В свою очередь за время проиводственной практики я дал некоторую базу для дальнейшего движения в данном направление и, надеюсь, что идеи, реализованные и предложенные во время работы помогут в дальнейшем.

Поставленные цели в начале практики были достигнуты и в точности реализованы, следуя намеченному плану.

СПИСОК ЛИТЕРАТУРЫ

1. Marc Peter Deisenroth, A. Aldo Faisal, Cheng Soon Ong. Mathematics for Machine Learning: Учебник –Cambridge University Press, 2020. – 411c.

2. Soroush Nasiriany, Garrett Thomas, William Wang: A Comprehensive Guide to Machine Learning: Учебник – Berkeley: University of California, 2019. – 185c.

3. Jain, V., Murray, J. F., Roth, F. Supervised learning of image restoration with convolutional networks: In Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, pages 1-8: 2007.