МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра дискретной математики и алгоритмики

МИЩЕНКО Никита Валерьевич

РАЗРАБОТКА АЛГОРИТМОВ ОБУЧЕНИЯ И ИСПОЛНЕНИЯ НЕЙРОННЫХ СЕТЕЙ С ИСПОЛЬЗОВАНИЕМ РАЗРЕЖЕННОГО ПРЕДСТАВЛЕНИЯ ДЛЯ ЗАДАЧ ОЦЕНКИ ПОЗЫ ЧЕЛОВЕКА

Магистерская диссертация
специальность 1-31 81 09 «Алгоритмы и системы обработки больших объемов информации»

Научный руководитель
Белоцерковский Алексей Маратович
кандидат технических наук

Научный консультант
Калиновский Александр Александрович
научный сотрудник

Допущена к защите
«____» _____2020 г.
Зав. Кафедрой дискретной математики и информатики
В. М. Котов
Доктор физико-математических наук, профессор

Минск, 2020

ОГЛАВЛЕНИЕ

РЕФЕРАТ 4

ВВЕДЕНИЕ 6

1. ЗАДАЧИ МАШИННОГО ОБУЧЕНИЯ НА МОБИЛЬНЫХ УСТРОЙСТВАХ 8

1.1 Машинное обучение и компьютерное зрение 8
1.2 Сверточные нейронные сети 9
1.2.1 Слой свертки 9
1.2.2 Полносвязный слой 11
1.2.3 Слой активации 12
1.2.4 Слой субдискретизации 13
1.2.5 Архитектуры сетей 13
1.3 Количество операций в сверточном слое 14
1.4 Различные оптимизации сверточной сети 15
1.4.1 Оптимизация архитектуры сети 15
1.4.2 Дистилляция сетей 17
1.4.3 Сжатие модели 18
1.4.4 Прунинг весов 18
1.5 Выводы 19

2. ИЗУЧЕНИЕ ВЛИЯНИЯ РАЗНЫХ СТРАТЕГИЙ ПРОРЕЖИВАНИЯ НА ТОЧНОСТЬ 20

2.1 Использованное окружение 20
2.2 Выбранная архитектура сети 21
2.3 Разработка алгоритма по прореживанию весов 23
2.4 Эксперименты с архитектурой 29
2.5 Эксперименты по-разному прореживанию сверточных слоев сети каждой стратегией 29
2.6 Выводы 32

3. ИЗУЧЕНИЕ ВЛИЯНИЕ ПРОРЕЖИВАНИЯ НА ВРЕМЯ ИСПОЛНЕНИЯ 33

3.1 Инструменты для запуска свертки на центральном процессоре 33
3.2 Инструменты для запуска свертки на графическом процессоре 34
3.2.1 Запуск на графическом процессоре персонального компьютера 34
3.2.2 Запуск на графическом процессоре мобильного устройства 34
3.3 Разработка свертки с прореживанием весов на Metal 35

3.4 Эксперимент на влияние времени исполнения от процента прореживания весов свертки на Metal 36
3.5 Выводы 41
ЗАКЛЮЧЕНИЕ 42
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 43

РЕФЕРАТ

Магистерская диссертация, 44 с., 25 рис., 5 таб., 2 ф., 18 источников.

Ключевые слова: МАШИННОЕ ОБУЧЕНИЕ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ, ОПТИМИЗАЦИЯ ВЕСОВ, РАЗРЕЖЕННОЕ ПРЕДСТАВЛЕНИЕ ВЕСОВ, ИСПОЛНЕНИЕ НА МОБИЛЬНЫХ УСТРОЙСТВАХ.

Объект исследования – проблема исполнения сложных сверточных нейронных сетей на мобильных устройствах. В частности, исследуются влияние использования разреженного представления весов в задачах машинного зрения.

Цель работы – изучить влияние разных способов прореживания весов сверточной нейронной сети на точность распознавания алгоритмов машинного зрения и на время работы сверточной сети на мобильном устройстве.

Методы проведения работы – изучение существующих методов оптимизаций обучения и исполнения сверточных нейронных сетей на мобильных устройствах. Разработка алгоритмов по минимизации количества весов сети разными стратегиями, а также разработка алгоритмов для быстрого исполнения слоев свертки. Проведение экспериментов на разработанном ПО, сравнительный анализ с обычным способом обучения и исполнения.

Результаты – новый способ оптимизации исполнения сверточных нейронных сетей на мобильных устройствам, позволяющий более чем в два раза улучшить производительность, почти не потеряв в точности решения задачи. Разработаны и реализованы на языке программирования Python алгоритмы по прореживанию весов сверточных слоев разными стратегиями. Разработаны и реализованы на языке программирования Metal и Swift алгоритмы по зануleнию одной стратегией весов сверточного слоя.

Область применения – различные задачи компьютерного зрения на мобильных устройствах, использующие нейронные сверточные сети.

ABSTRACT

Master thesis, 44 p., 25 fig., 5 tables, 2 formulas, 18 sources.

Keywords: MACHINE LEARNING, COMPUTER VISION, CONVOLUTIONAL NEURAL NETWORK, WEIGHTS OPTIMIZATION, SPARSE WEIGHTS REPRESENTATION, MOBILE DEVICES INFERENCE.

Object of research – the problem of executing complex convolutional neural networks on mobile devices. In particular, the influence of sparse weights representation in machine vision tasks.

Objective – to study the influence of convolutional neural network different zeroing weights strategies on the accuracy of recognition in machine vision task. And to study the influence on the inference time of the convolutional network on a mobile device.

Methods – study of existing methods for optimizing training and inference of convolutional neural networks on mobile devices. Development of algorithms with different zeroing strategies, as well as development of algorithms for fast convolution layers inference. Conducting experiments on the developed software, comparative analysis with the usual method of training and inference.

Results – a new way to optimize convolutional neural network inference on mobile devices, which allows to double the performance, almost without any drops in the algorithm accuracy. Zeroing algorithms have been developed and implemented in the Python programming language with three different zeroing strategies. Optimized inference with zeroing was developed and tested in the Metal and Swift programming language.

Application area – various computer vision tasks, especially ones using convolutional neural network with inference on mobile devices.

ВВЕДЕНИЕ

Развитие вычислительной техники и большое количество накопленных знаний позволило человечеству приблизиться к созданию искусственного интеллекта. Сейчас активно применяются методы машинного обучения для автоматизации различных видов операций в повседневной жизни. Однако, до недавнего времени, большинство методов машинного обучения были разработаны для использования на стационарных компьютерах.

В настоящее же время, высокую популярность приобрели мобильные устройства. Которые, в связи с увеличением мощностей вычислительных систем и повышением их энергоэффективности, стали обладать достаточной вычислительной мощностью для исполнения сложных методов машинного обучения. Однако, по сравнению с стационарными компьютерами, в условиях реального использования мобильных устройств, имеются определенные ограничения, влияющие на результат работы программного обеспечения. Такими ограничениями могут быть малая емкость аккумулятора и потенциальное снижение частоты процессора при перегреве, приводящее к понижению производительности. Описанные ограничения добавляют важные требования для применяемых алгоритмов – высокий уровень оптимизации и энергоэффективности. В случае использования нейронных сетей в программном обеспечении существует множество способов оптимизации сетей, позволяющих как ускорить работу нейросети, так и уменьшить размеры весов и занимаемого пространства в накопителе устройства.

В рамках этой работы, в первой главе будет проведен краткий обзор нынешней ситуации в мире машинного обучения и разных научных статей, целью которых является оптимизация работы нейронных сетей. Будет рассмотрен один из способов оптимизации перечисленных ранее параметров – использование разреженного представления весов нейронных сетей. Применение данного способа потенциально позволит уменьшить количество потребляемой памяти нейросетевыми алгоритмами и улучшить их энергоэффективность, не потеряв значительно в точности. Для оценки влияния выбранного метода на точность, во второй главе будет проведено несколько экспериментов по изучению изменения точности алгоритма при разных параметрах. В третей главе будут проведены эксперименты для вывода зависимости времени исполнения алгоритма от параметров выбранного метода, для оценки энергоэффективности.

Актуальность данной работы следует из высокой популярности мобильных устройств, для которых исследуемый метод и предназначается. Кроме того, ранее не были проведены эксперименты по оценке

энергоэффективности исследуемого метода на тех же мобильных устройствах, в чем и состоит новизна этой работы. По адресу github.com/GTnikito/sparse-net-study можно найти всё реализованное программное обеспечение, с результатами экспериментов.

1. ЗАДАЧИ МАШИННОГО ОБУЧЕНИЯ НА МОБИЛЬНЫХ УСТРОЙСТВАХ

1.1 Машинное обучение и компьютерное зрение

Машинное обучение – это область компьютерных алгоритмов, которые улучшают собственную точность работы по мере увеличения кол-во известных данных. Эта область информационных технологий рассматривается как подмножество искусственного интеллекта. Алгоритмы машинного обучения строят так называемую гипотезу – математическую модель на основе набора данных, который называется обучающей выборкой. Гипотезы строятся таким образом, чтобы делать прогнозы или принимать решения на ранее неизвестных данных. Точность гипотезы измеряется на основе набора данных, который называется тестовой выборкой. Уже сейчас алгоритмы машинного обучения используются повсеместно для решения самых разнообразных задач. Например, таких, как выделение спама среди писем электронной почты или как определения рукописных символов. И для многих других задач, которые трудно или невозможно решить классическими алгоритмами.

Точность алгоритмов машинного обучения тесно связано с качеством обучающей выборки: насколько много имеется данных, насколько хорошо данные описывают возможные комбинации. Большое накопление реальных данных за последнее время человечеством и стало одной из ключевых причин нынешней популярности методов машинного обучения. Кроме того, увеличение вычислительной мощности компьютеров, позволяет использовать более сложные модели в машинном обучении, применять более точные оптимизационные методы функции ошибки, которую часто называют «целевой» функцией.

Стоит отметить, что в наше очень много задач компьютерного зрения стали решаться методами машинного обучения. Основные цели задач машинного зрения – это получение высокоуровневых знаний из цифрового изображения или видео. Это попытка автоматизации целого класса задач, которые выполняются человеческой зрительной системой. [1]

Задача компьютерного зрения включает в себя получение изображения, обработка и анализ оного, а также извлечения высокоуровневых признаков для классификации и пост-обработки. Сами изображения могут быть одиночными кадрами, а также видеопоследовательностью с одной или нескольких камер и

многомерными данными с 3D-сканера или медицинского сканирующего устройства.

1.2 Сверточные нейронные сети

В настоящее время лучшие алгоритмы для решения задач компьютерного зрения основаны на сверточных нейронных сетях (Convolutional Neural Network или ConvNet). Этот тип нейронных сетей является одним из самых популярных архитектур искусственных нейронных сетей в наше время. Наиболее часто применяется для анализа визуальных образов. Большим преимуществом сверточной сети является то, что она не требует большой предварительной обработки входящих изображений, особенно по сравнению с другими алгоритмами классификации изображений. То есть сеть самостоятельно в процессе обучения выучивает фильтры для предобработки картинки, не требуя ручного создания фильтров, которые требовались в ранее используемых алгоритмов компьютерного зрения [2].

Работа сверточной нейронной сети обычно трактуется как переход от специфических особенностей изображения к более абстрактным деталям, вплоть до идентификации высокоуровневых понятий. В процессе обучения сеть адаптируется к входящим данным, генерируя необходимую иерархию признаков, выделяя важные детали и фильтруя несущественные.

Сверточная нейронная сеть состоит из большого количества слоев: входного слоя, выходного слоя и нескольких скрытых слоев между ними. В основном скрытые слои состоят из слоев свертки, слоев активации, пулинга и полносвязных слоев. Рассмотрим каждый из перечисленных слоев.

1.2.1 Слой свертки

Сверточный слой является основным блоком сверточной нейронной сети. Свертка – процесс добавления каждого элемента изображения к его локальным соседям, помноженное на соответствующий вес в ядре. Важно отметить, что эта операция не является традиционным матричным умножением, несмотря на то что она аналогично обозначается "*".

Рассмотрим операцию подробнее: пусть есть две матрицы, первая матрица – фрагмент изображения, а вторая матрица – ядро. Тогда свертка – это процесс суммирования результатов перемножений соответствующих элементов фрагмента изображения и ядра.

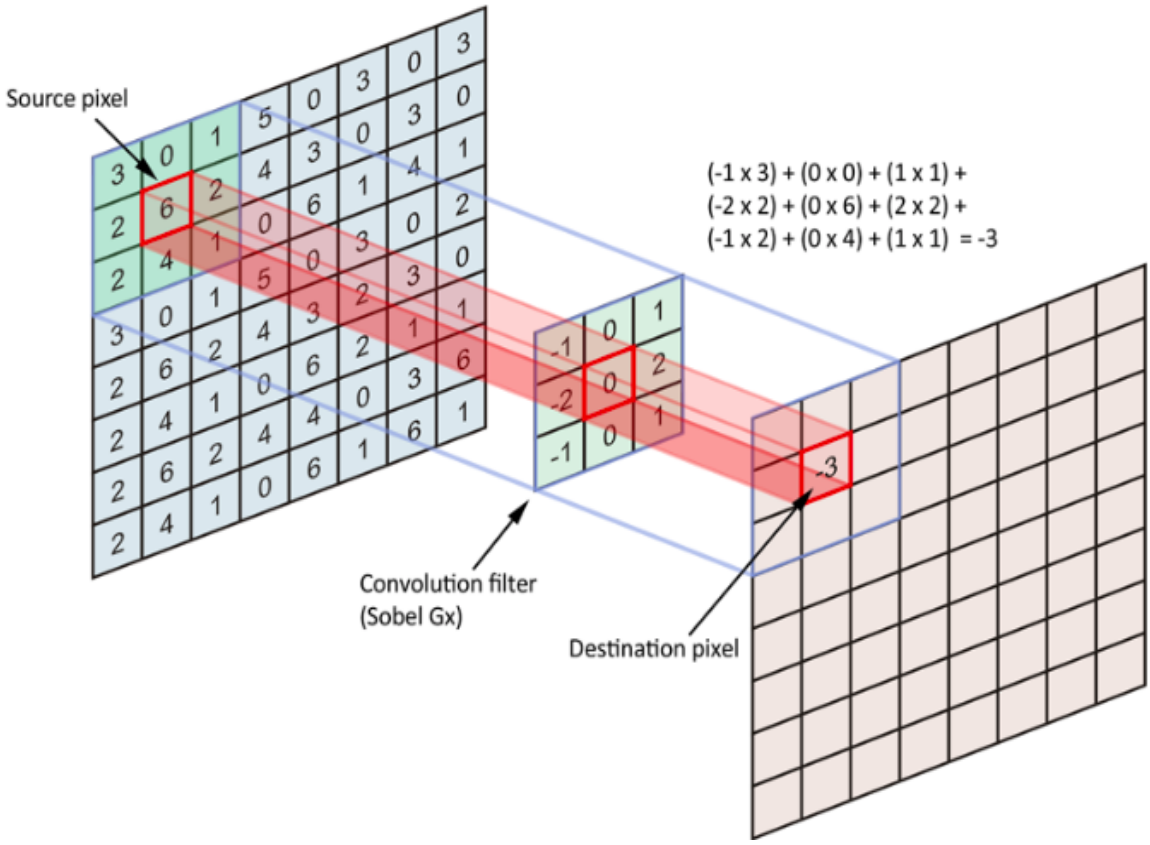

Рис. 1. Слой свертки. Синим цветом изображена входящая текстура. Зеленым цветом изображены весы свертки. Розовым цветом изображена выходящая текстура.

Элементы второй матрицы называются весами сверточного слоя. Для каждого канала исходного фрагмента изображения создается свой слой ядра. Веса сверточного слоя изначально неизвестны и настраиваются в процессе обучения согласно выбранному оптимизатору. Влияние частичного прореживания именно этих весов на точность всего алгоритма будет исследоваться в второй главе этой работы.

Важно отметить, что в большинстве реализаций сверточных слоев, кроме операции свертки, присутствует дополнительная операция суммирования, применяющаяся к результатам свертки. На Рис.2 отображен способ расчета одного конкретного значения. Сначала к нему применяется свертка с весами a_1, a_2, a_3 … a_n, а потом прибавляется b. Данная операция существуют для того, чтобы даже при нулевых входных данных свертки, получить не нулевой результат на выходе. В данной работе эксперименты проводились над весами w, веса смещений b обучались классическим способом.

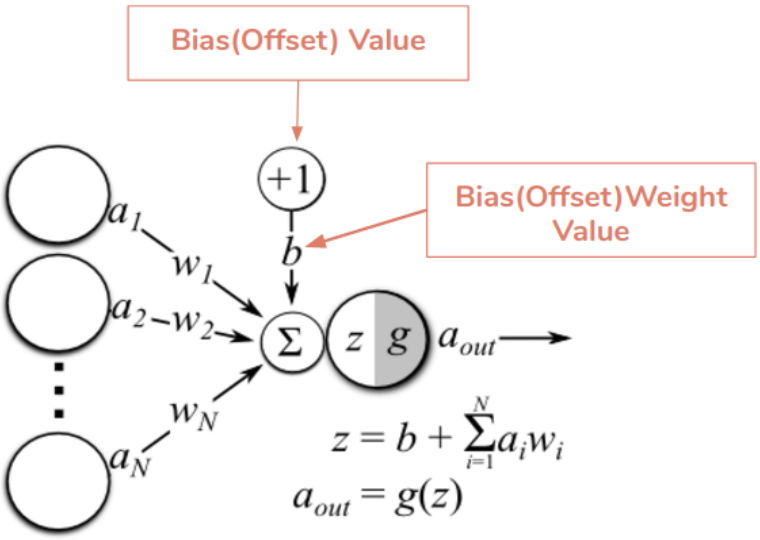

Рис. 2. Слой свертки со смещением [3]

1.2.2 Полносвязный слой

В сверточных нейронных сетях полносвязный слой необходим для финального представления данных. Так как после нескольких проходов слоев свертки, сетка пикселей с высоким разрешением преобразуется в большой набор каналов, хранящих небольшой объем данных, которые интерпретируются как наиболее абстрактные понятия, выделенные из исходного изображения. Эти данные передаются в полносвязный слой, который на выходе имеет количество нейронов, равное количество классифицируемых кластеров [4]. В полносвязном слое каждый выход является линейной комбинацией входов, умноженных на соответствующий вес.

Данная работа ориентируется на такие задачи как: семантическая сегментация, детектирование, распознавание и сегментация объектов в реальном времени. В этих задачах используется т.н. полноконволюционная архитектура сети [5], в которой не используются полносвязные слои. Но стоит отметить, что разреженное представление в полносвязных слоях будет иметь ещё большую эффективность, так как для этих слоёв характерна еще большая избыточность по числу параметров. Так же существуют другие архитектуры нейронных сетях, в некоторых все слои одного типа – полносвязные.

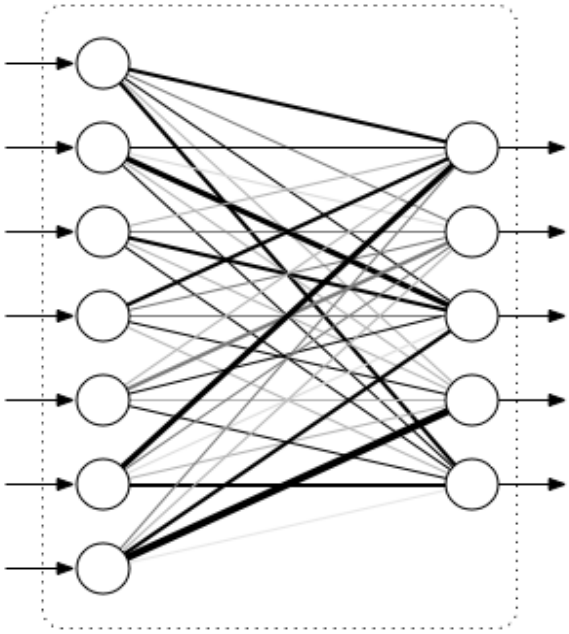

Рис. 3. Полносвязный слой

1.2.3 Слой активации

Зачастую данный слой используется для обработки результатов каждого сверточного слоя и добавляется для того, чтобы результат двух последовательных слоев свертки нельзя было представить одним слоем свертки. Функция активации может быть любой, ее тип зависит от настроек, применяется независимо к каждому значению входного фрагмента. Традиционно используются функции сигмоиды или гиперболического тангенса. В последнее время особую популярность приобрела функция активации ReLU (rectified linear unit), значительно ускоряющая процесс обучения и в то же время упрощающая вычисления. В экспериментах второй главы будет использоваться именно эта функция активации.

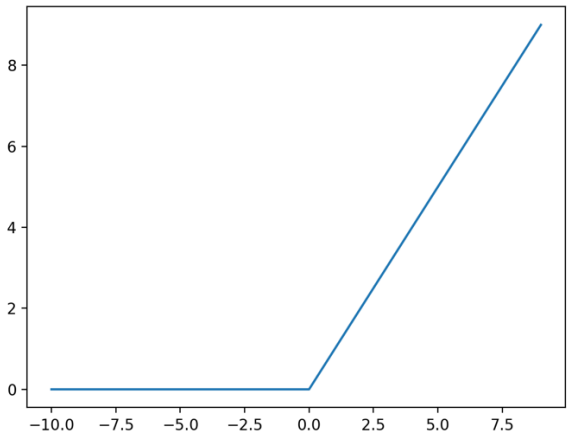

Рис. 4. График функции активации ReLU

1.2.4 Слой субдискретизации

Слой субдискретизации или пулинг - слой нелинейного уплотнения карты признаков. Например, квадрат 2x2 пикселя сжимается до одного пикселя. Чаще всего используется функция максимума. Цель этого слоя – уменьшение пространственного объема изображения. Кроме того, фильтрация некоторых деталей помогает модели избежать переобучения. Слой пулинга, в основном, вставляется после слоя свертки перед следующим слоем свертки.

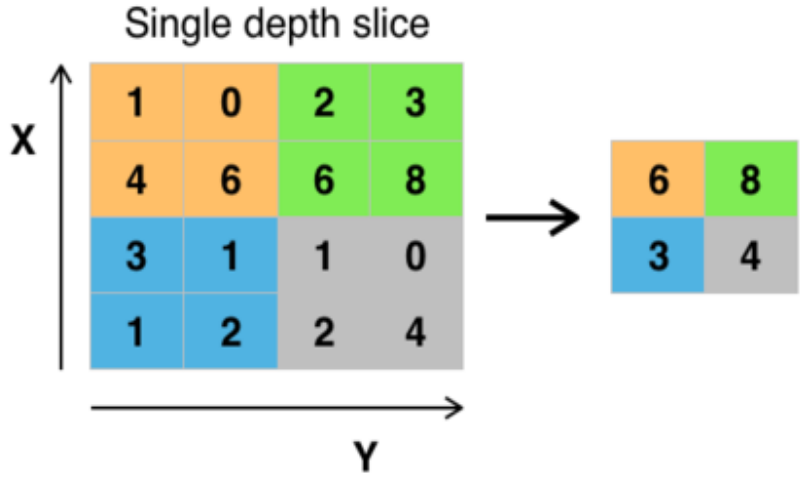

Рис. 5. Применение субдискретизации 2x2

1.2.5 Архитектуры сетей

Для того, чтобы построить работающую модель с хорошим результатом в поставленной задаче, требуется специальное расположение перечисленных слоев в нужном порядке. Первой архитектурой нейронной сверточной сети, которая добилась хороших результатов для своего времени, была LeNet. Она была предложена Яном Лекуном в 1998 году. Является простой комбинацией сверточных слоев и слоев активации, слоев субдискретизации и слоев полносвязных.

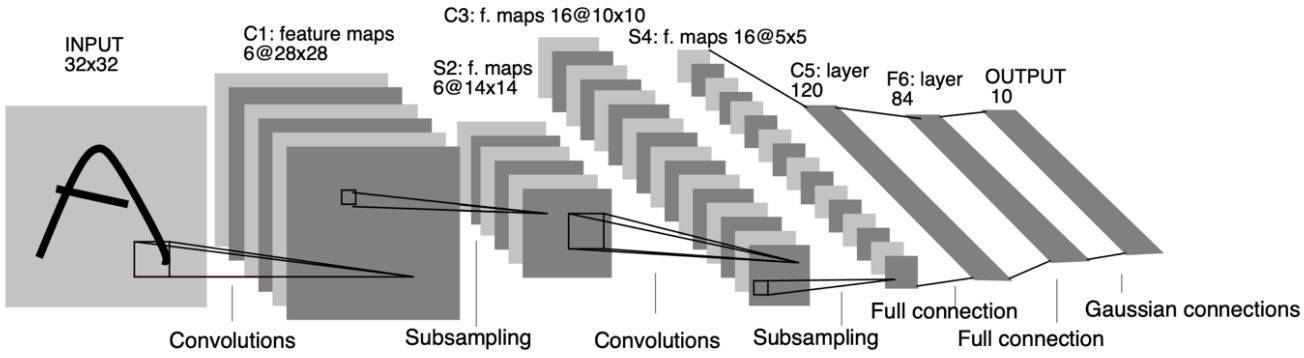

Рис. 6. Сеть LeNet [6]

Основные преимущества сверточной нейронной сети с такой конфигурацией:

- Является одним из лучших алгоритмов по распознаванию и классификации изображений;
- Намного проще обычной полносвязной нейронной сети;
- Возможность распараллелить вычисления, что открывает возможность запуска нейронной сети на видеопроцессорах.

Основной недостаток сверточной нейронной сети:

- Большое количество настраиваемых гиперпараметров, таких как: количество слоев, размер ядра свертки и его шаг сдвига, параметры пулинга, параметры полносвязных слоев, наличие которых усложняет настройку и обучение нейронной сети.

1.3 Количество операций в сверточном слое

Одна из двух целей данной работы является изучение влияния прореживания весов свертки на энергоэффективность сверточной нейронной сети. Добиться увеличения энергоэффективности можно путем уменьшения времени работы. Остальные параметры посчитаем зафиксированными в рамках этой работы.

Достичь уменьшения времени работы вполне легко, для этого необходимо уменьшить количество базовых операций, выполняемых сетью при работе. Для этого вспомним формулу вычисления результата обычного сверточного слоя из [7] с единичным сдвигом ядра фильтра и дополнением нулями по краям:

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m}$$

Формула 1. Вычисление результата свертки

В которой:

- K – ядро свертки размера [D_K, D_K, M, N];
- F – входные данные размера [D_F, D_F, M];
- G – выходные данные размера [D_F, D_F, N].

Очевидно, что для такой свертки с такой формулой, количество базовых операций такова:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

Формула 2. Количество операций в стандартной свертке

В которой:

- D_K, D_K – размер одного ядра свертки;
- M – количество каналов входящего изображения;
- N – количество каналов выходящего изображения;
- D_F, D_F – размер одного канала входящего и выходящего изображения.

Из приведенных формул выше заметно, что присутствует прямая зависимость времени исполнения от количества весов в ядре свертки. Влияние изменения числа базовых операций на время работы алгоритма будет исследовано в третей главе данной работы.

1.4 Различные оптимизации сверточной сети

В данном пункте будут рассмотрены различные существующие способы оптимизации сверточной нейронной сети для работы на мобильных устройствах.

1.4.1 Оптимизация архитектуры сети

Одним из самых популярных и эффективных способов оптимизации архитектуры нейронных сетей является замена каких-либо блоков сети на аналогичные более быстрые варианты. Например, в [7] авторы решили заменить каждый обычный сверточный слой на комбинацию поканальной свертки и сепарабельной свертки. На примере ниже слева изображен стандартный слой свертки 3x3, справа изображена использованная в работе комбинация. Данная оптимизация позволила уменьшить кол-во базовых операций в 8.5 раз, от 4866 миллионов до 569 миллионов. Кроме того, за счет этой же оптимизации было уменьшено и кол-во параметров в 7 раз, от 29.3 миллионов до 4.2 миллиона. При этом, потери в точности составили менее 1.5%.

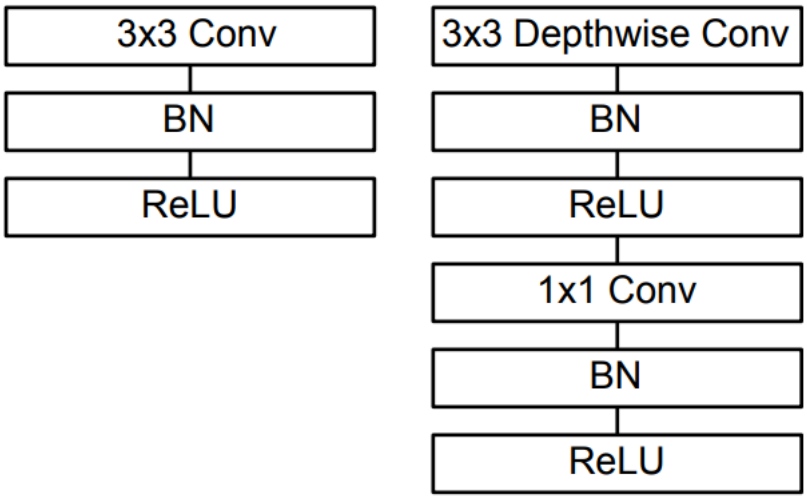

Рис. 7. Стандартный слой свертки и комбинация поканальной свертки с сепарабельной сверткой [7]

Кроме этого, в оригинальной работе так же было применено две других оптимизации:

- Уменьшение кол-во каналов входных и выходных данных каждой свертки в $\frac{1}{\alpha}$ раз, $\alpha \in (0, 1]$;
- Уменьшение размера входных данных в саму сеть в $\frac{1}{\rho}$ раз, $\rho \epsilon (0, 1]$. В рассмотренном эксперименте той работы, при значении множителя 1, размер входных данных был 224x224.

Влияние описанных оптимизаций представлено на следующих двух рисунках.

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Рис. 8. Влияние изменения параметра α на точность, кол-во базовых операция и кол-во параметров [7]

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Рис. 9. Влияние изменения параметра ρ на точность, кол-во базовых операция и кол-во параметров [7]

Согласно изученным результатам, возможна существенная оптимизация количество операций в нейронных сверточных сетей, без сильной потери в точности алгоритма.

1.4.2 Дистилляция сетей

Рассмотрим на примере бинарной классификации. Для данного способа оптимизации, необходимо иметь сразу две архитектуры сетей [8]:

- Первая сеть, с большим количеством весов, которая была заранее обучена на тренировочных данных и имеет хороший результат на валидационной выборке. Однако, слишком сложная для использования на мобильном устройстве, например;
- Вторая сеть, намного более простая по сравнению с первой. Может иметь подобную на первую сеть архитектуру, но это не обязательное требование. Архитектура подобрана так, чтобы сеть могла использоваться на мобильном устройстве с нужными результатами по производительности.

После завершения обучения большой сети, начинаем обучение маленькой сети. Однако, с небольшой поправкой – мелкую сеть немного модифицируем, а точнее срежем последний слой, чтобы получить вероятность каждого из возможных ответов. И в процессе обучения легкой сети, прогоняем соответствующий элемент из обучающей выборки и через тяжелую сеть. Корректируем функцию ошибки легкой сети так, чтобы она штрафовала не только за неправильный ответ, но также и за отклонение вероятностей по классам от соответствующих вероятностей из ответа тяжелой сети. Согласно [9], данный способ позволяет ускорить время работы нейронной сети более чем в пятнадцать раз, при не более чем пятипроцентной потере в точности.

1.4.3 Сжатие модели

Этот метод оптимизации уже реализован в большинстве популярных библиотек машинного обучения. И, по сравнению с перечисленными ранее, его цель оптимизировать размер весов модели, а не время ее работы. Это достигается путем квантизации весов.

Квантизация весов – уменьшение размерности весов, путем изменения типа их данных, с возможной потерей точности. Возможна как квантизация во время обучения, так и после. Например, при обучении использовался тип данных Float32 для хранения весов. Потом произведена квантизация и соответствующие веса сконвертированы в тип данных Uint8. В некоторых случаях, данная операция не сильно повляет на точность алгоритма, при этом уменьшит размер модели в 4 раза [10].

1.4.4 Прунинг весов

Один из недостатков нейронный сетей – большое количество весов, применяющиеся как в сверточных слоях, так и в полносвязных слоях. Второй недостаток нейронных сетей – их склонность переобучению на обучающей выборке, приводящий к потере точности [11] при запуске нейронное сети на неизвестных данных.

Прореживание некоторых весов, другими словами – прунинг – способно решить часть этих недостатков [12]. В этой работе было показано, что прореживание некоторых весов не всегда обязательно приводит к сильной потере в точности.

Одной из стратегией прореживания весов модели является L0 регуляризация весов. Концепция данного популярного подхода заключается в явном штрафовании за ненулевые значения весов без каких-либо дополнительных ограничений.

Сам метод L0 регуляризации весов в экспериментальной части этой работы рассмотрен не будет. Однако, согласно [11] применение L0 регуляризации весов позволит не только оптимизировать занимаемое моделью хранилище, а также ускорить работу нейронной сети и, соответственно, уменьшить количество выполненных базовых операций при запуске сверточной сети.

Network	CIFAR-10	CIFAR-100
original-ResNet-110 (He et al., 2016a)	6.43	25.16
pre-act-ResNet-110 (He et al., 2016b)	6.37	-
WRN-28-10 (Zagoruyko & Komodakis, 2016)	4.00	21.18
WRN-28-10-dropout (Zagoruyko & Komodakis, 2016)	3.89	18.85
WRN-28-10-$L_{0_{hc}}, \lambda = 0.001/N$	**3.83**	**18.75**
WRN-28-10-$L_{0_{hc}}, \lambda = 0.002/N$	3.93	19.04

Таблица 1. Сравнение процента ошибок разных нейронных сетей на тестах CIFAR10 и CIFAR100. В третьем ряду нейронные сети с регуляризацией весов L0 [13]

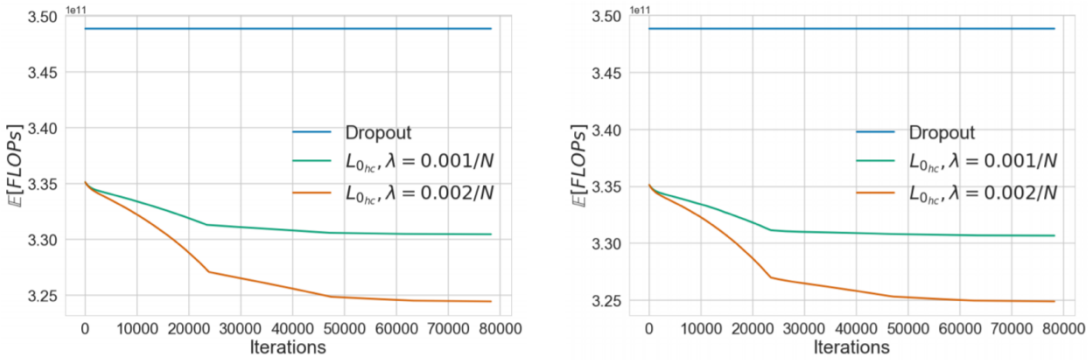

Рис. 10. Сравнение количества операций с плавающей точкой в секунду на тестах CIFAR10 и CIFAR100 [13]

1.5 Выводы

Существует множество разных способов оптимизации сверточных нейронных сетей. Целью оптимизации является как уменьшение размера весов модели, так и времени исполнения сверточной нейронной сети у конечного пользователя. В связи с высокой популярностью мобильных устройств, задачи оптимизации сейчас стали особо важными.

Один из способов оптимизации весов сверточной сети – прунинг весов – исследован в последующих главах, в которых испробовано прореживание весов разными стратегиями. Данный способ оптимизации выбран для исследования как из-за небольшого количества работ на данную тематику на момент написания этой работы, так из-за большого внимания к нему популярными библиотеками машинного обучения в данный момент [14].

2. ИЗУЧЕНИЕ ВЛИЯНИЯ РАЗНЫХ СТРАТЕГИЙ ПРОРЕЖИВАНИЯ НА ТОЧНОСТЬ

2.1 Использованное окружение

Согласно исследованиям [15] процесс обучения нейронных сетей при использовании Графических Процессоров (GPU) ускоряется в 10 раз по сравнению с обучением тех же моделей, используя Центральный Процессор (CPU). Такая огромная разница возможна благодаря применению архитектуры параллельных вычислений CUDA, от компании nvidia. Именно поэтому, все описанные в данной главе эксперименты с нейронными сетями будут проводиться с использованием GPU.

Для обучения нейронных сетей в рамках данной работы будет использоваться Keras. Это библиотека нейронных сетей с открытым исходным кодом, написанная на Python [16]. Keras ориентирован на удобство использования, модульность и расширяемость, был специально разработан для быстрого проведения экспериментов с нейронными сетями.

Однако, Keras является всего лишь интерфейсом между разработчиком и бэкендом, который и выполняет необходимые расчеты на устройстве. В качестве вычислительного бэкэнда будет использоваться Tensorflow, являющейся программной библиотекой с открытым исходным кодом для потоков данных и дифференцированного программирования для целого ряда задач. Эта библиотека используется как для исследований, так и в производстве программного обеспечения. Изначально TensorFlow разрабатывался для внутреннего использования Google командой Google Brain, однако позже был выпущен в свободный доступ под лицензией Apache.

Таким образом, в данной работе нейронная сеть будет написана на языке Python, используя интерфейс Keras с бэкендом Tensorflow. Процесс обучения нейронной сети будет проходить на Графическом Процессоре.

Кроме того, так же было изучено программное обеспечение Git для контроля версий разрабатываемого программного кода. Разработанный программный код был выложен в открытый доступ в репозиторий GitHub [17].

2.2 Выбранная архитектура сети

Для изучения воздействия различных манипуляций с весами модели в данной работе использована простая модель из примера запуска сверточной нейронной сети в Keras [18]. Выбранная модель позволяет достичь до 75% точности на валидационной выборке данных CIFAR10. Так как цель работы является улучшение производительности задач машинного зрения, то такая задача, ввиду своей простоты, отлично подходит для исследований.

Поэтому, для экспериментов в этой главе выберем задачу CIFAR10. Это один из наиболее широко используемых наборов данных для исследования машинного обучения. Набор данных содержит 60 000 цветных изображений 32x32 в таких десяти различных классах, как: самолеты, автомобили, птицы, кошки, олени, собаки, лягушки, лошади, корабли и грузовики. Поскольку изображения в CIFAR-10 имеют низкое разрешение (32x32), процесс обучения и проверки работоспособности алгоритмов занимает меньше времени, по сравнению с более большими наборами данных.

Так же, для улучшения точности распознавания, в процессе обучения используется аугментация выборки данных – исходные картинки будут подвергаться небольшим трансформациям, для искусственного увеличения размеров обучающей выборки.

На следующей странице представлена визуализация простой модели, на которой будут проводиться первые эксперименты. Сама модель очень проста – четыре сверточных слоя, два слоя пулинга, три слоя дропаута и два полносвязных слоя.

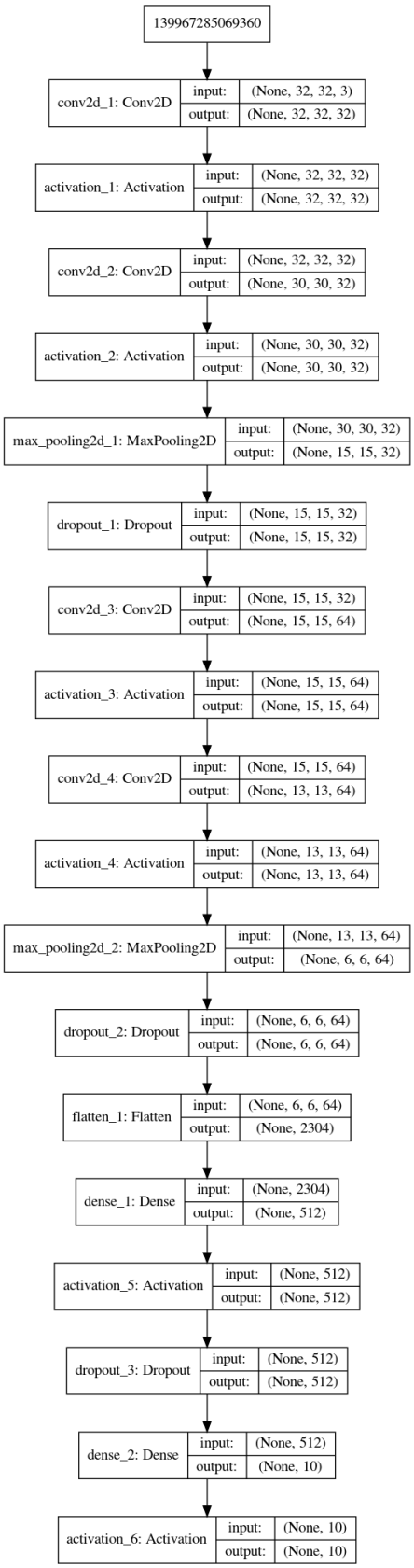

Рис. 11. Архитектура исследуемой модели сверточной нейронной сети

2.3 Разработка алгоритма по прореживанию весов

Рассмотрим стратегию прореживания №1. Согласно этой стратегии, собираются абсолютные значения каждого веса каждого оптимизируемого слоя в один контейнер. Там значения весов сортируются по возрастанию и выбирается порог, меньше которого будет $x\%$ весов, где x – и есть процент прореживания. Потом каждый вес в рассматриваемых слоях зануляется, если его абсолютное значение меньше порогового значения.

В разработке программного кода необходимо было изначально учесть наличие разных стратегий по прореживанию. Так же, кроме разных стратегий по прореживанию, необходимо было так же добавить опцию по выбору конкретных слоев, над весами которых проводить оптимизацию. Также, у выбранных слоев есть разные типы весов. Например, для сверточных слоев есть два типа весов: сами веса, участвующие в свертке; и веса, отвечающие за сдвиг.

Поэтому, согласно поставленным выше требованиям, было разработано программное обеспечение, которое создавало "маску" прореживания для выбранных слоев. То есть, нулевое значение в маске означало, что необходимо оставить значение веса неизменным. А ненулевое значение в маске означало, что соответствующий вес необходимо занулить.

Изначально, в программном обеспечении была реализована стратегия по прореживанию №1. После разработки, программное обеспечение необходимо было протестировать. Для этого был написан код, который обучал обсужденную ранее модель пять итераций обычным способом, получал график распределения величины весов. Потом применялось прореживание нужного процента весов, так же рисовался график распределения величины весов, графики сравнивались. Ниже представлен график распределения величины весов без зануления и с прореживанием 20% весов:

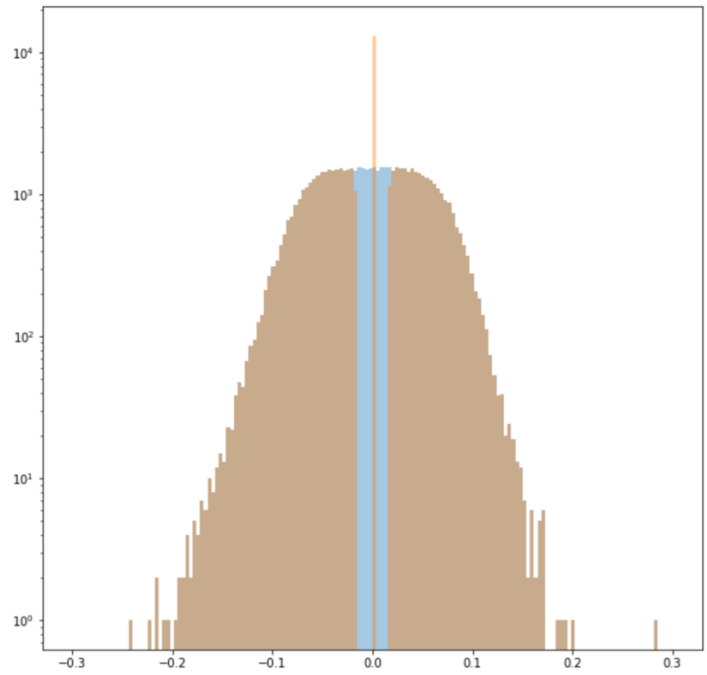

Рис. 12. График распределения величин весов, с 20% прореживанием по стратегии №1

Данная методика – визуализация количества весов выбранных слоев с близкими абсолютными значениями в кластере помогла отладить методы выбора весов для прореживания и самих методов прореживания. Ниже представлен график распределения величины весов без зануления и с прореживанием 50% весов:

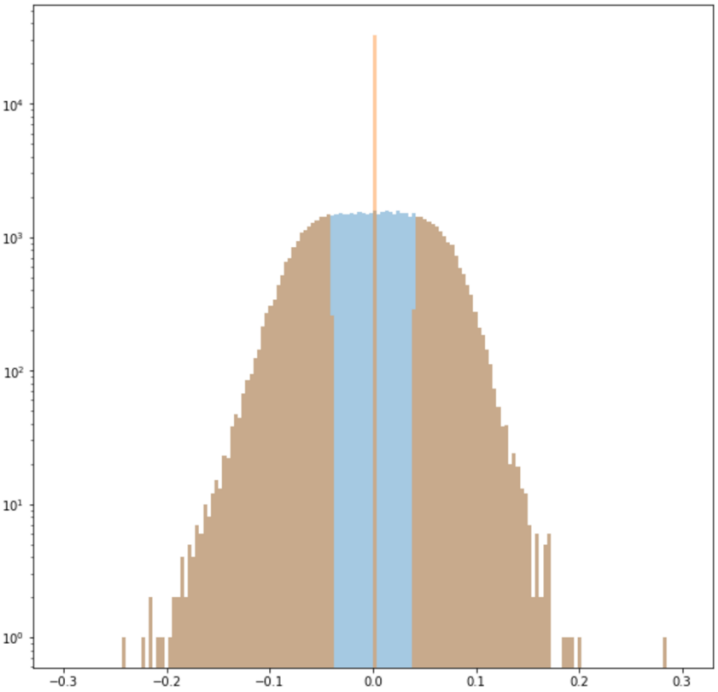

Рис. 13. График распределения величин весов, с 50% прореживанием по стратегии №1

Важно отметить, что представленная методика отладки помогла в процессе разработки найти ошибку и исправить ее. Из-за особенностей синтаксиса библиотеки NumPy, даже при нулевом проценте прореживания, а значит и при нулевой маске прореживания, часть весов все равно занулялась. Эта особенность была изучена и учтена. Ниже представлен график распределения величины весов без прореживания и с прореживанием 80% весов:

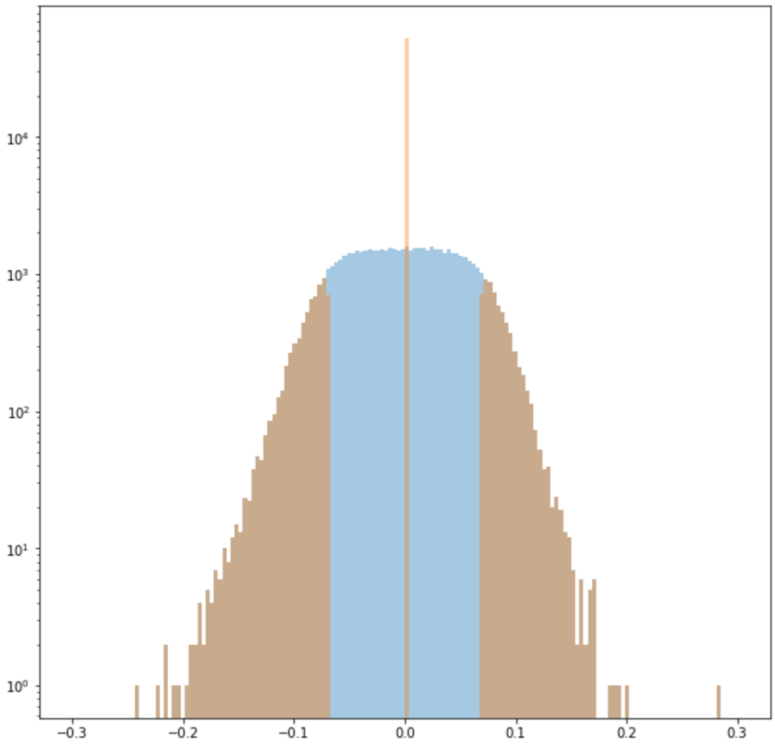

Рис. 14. График распределения величин весов, с 80% прореживанием по стратегии №1

Так же, были рассмотрены две других стратегии прореживания:

- Стратегия №2. Занулять целыми каналами. Причем, для прореживания выбираются слои с наименьшим средним абсолютным значением весов, в каждом фильтре независимо; Процент прореживания соотвествует количеству зануляемых каналов поделенное на общее количество каналов;
- Стратегия №3. Также занулять целыми каналами. Отличие от предыдущей стратегии в том, что при выборе слоев для прореживания, сравниваются все слои из оптимизируемых фильтров одновременно в одном контейнере.

Обе стратегии были реализованы в программном обеспечении. Детально рассмотрим реализованный алгоритм каждой из стратегий.

Алгоритм прореживания по стратегии №2:

Необходимо: процент прореживания z, массив оптимизируемых фильтров w[].

1. **Цикл** i = 1, …, размер(w[]):
2. filter – w[i] рассматриваемый фильтр
3. avg – контейнер для хранения поканальных средних
4. **Цикл** j = 1, …, размер(filter):
5. добавляем в avg модуль среднего значения по всем весам канала j
6. **Конец цикла**
7. avg сортируется
8. thr = avg(z * размер(avg)) извлекаем пороговое значение
9. **Цикл** j = 1, …, размер(filter):
10. считаем модуль среднего значения по всем весам канала j = ch_avg
11. **Если** ch_avg <= thr:
12. зануляем весы этого канала фильтра filter
13. **Иначе:**
14. оставляем весы этого канала фильтра filter без изменений
15. **Конец цикла**
16. **Конец цикла**

Возврат: новые веса оптимизируемых слоев. Некоторые значения были занулены, некоторые остались прежними.

Алгоритм прореживания по стратегии №3:

Необходимо: процент прореживания z, массив оптимизируемых фильтров w[].

1. avg – контейнер для хранения поканальных средних
2. **Цикл** i = 1, …, размер(w[]):
3. filter – w[i] рассматриваемый фильтр
4. **Цикл** j = 1, …, размер(filter):
5. добавляем в avg модуль среднего значения по всем весам канала j
6. **Конец цикла**
7. **Конец цикла**
8. avg сортируется
9. thr = avg(z * размер(avg)) извлекаем пороговое значение
10. **Цикл** i = 1, …, размер(w[]):
11. filter – w[i] рассматриваемый фильтр
12. **Цикл** j = 1, …, размер(filter):
13. считаем модуль среднего значения по всем весам канала j = ch_avg
14. **Если** ch_avg <= thr:
15. зануляем весы этого канала фильтра filter
16. **Иначе:**

17. оставляем весы этого канала фильтра filter без изменений
18. **Конец цикла**
19. **Конец цикла**

Возврат: новые веса оптимизируемых слоев. Некоторые значения были занулены, некоторые остались прежними.

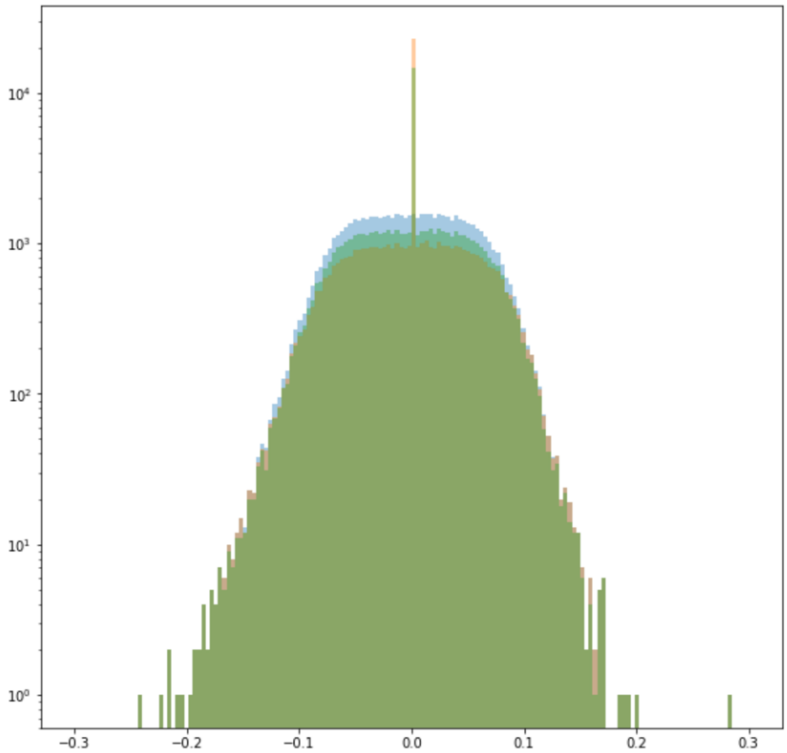

Рис. 15. График распределения величин весов, с 20% прореживания по стратегии №2 (зеленый цвет) и по стратегии №3 (желтый цвет)

На рис. 15 заметна разница в прореживании стратегией №1 и стратегиями №2 и №3. Если в стратегии №1 прореживались самые близкие к нулю весы, то в №2 и №3 прореживаются целые каналы, что срезает и веса с большими значениями, которые бы в стратегии №1 не занулились.

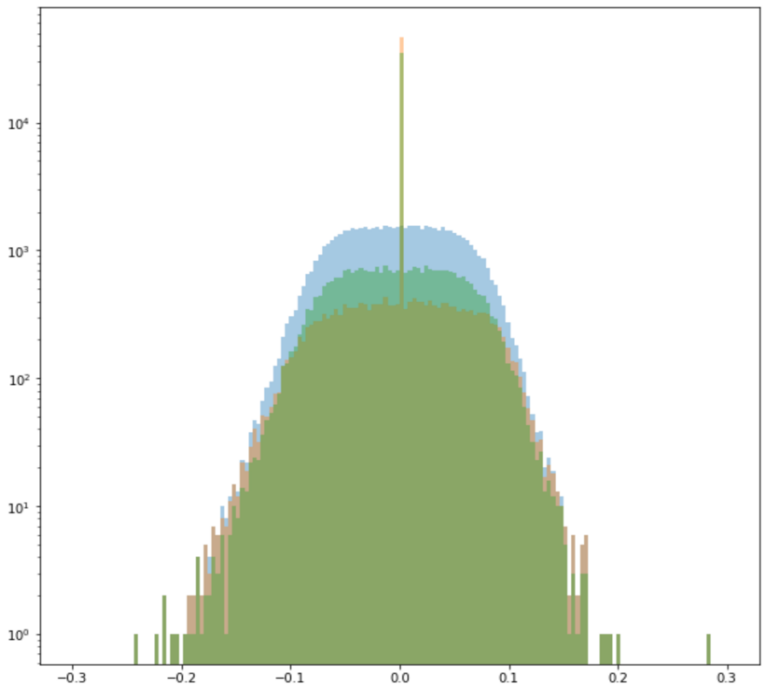

Рис. 16. График распределения величин весов, с 50% прореживания по стратегии №2 (зеленый цвет) и по стратегии №3 (желтый цвет)

Зеленые кластеры - стратегия №2. Оранжевые - стратегия №3. Как видно по графикам, сильно большой разницы нет, стратегия №3 при одинаковых параметрах прореживает больше весов.

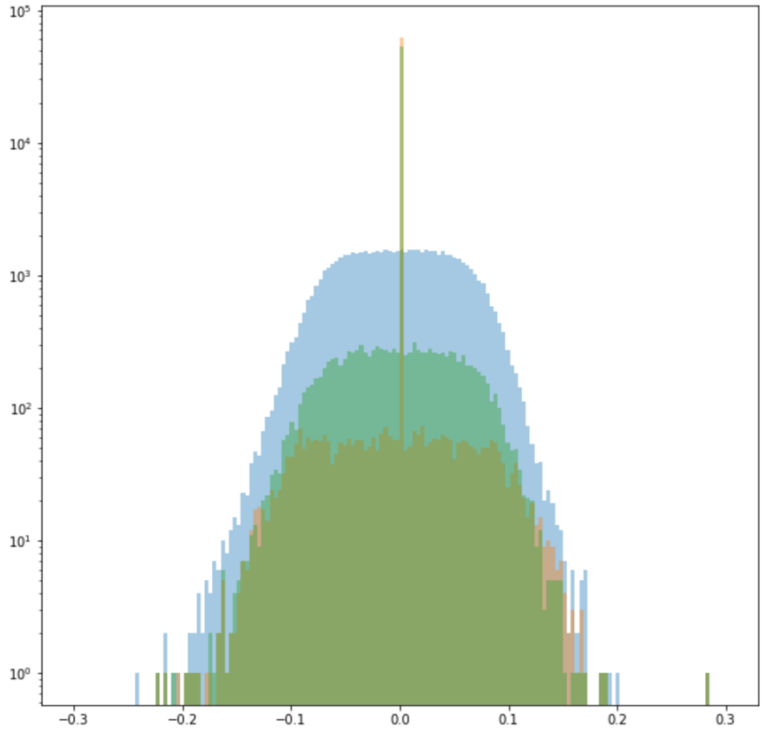

Рис. 17. График распределения величин весов, с 80% прореживания по стратегии №2 (зеленый цвет) и по стратегии №3 (желтый цвет)

Как видно по графику сверху, уже даже при 80% параметре, довольно большое кол-во весов занулено. Влияние же на точность распознавания и стоит ли вообще применять эти стратегии, нам предстоит исследовать в следующих пунктах.

2.4 Эксперименты с архитектурой

Также были проведены эксперименты с L1-регуляризацией для весов. Ведь регуляризация – добавление штраф за большие веса, что мотивирует сеть использовать веса с маленьким абсолютным значением при обучении. Также, были изучены результаты распознавания объектов с моделью из примера CIFAR10 [18]. Точность распознавания на валидационной выборке оказалась 76%, после 100 эпох. Следующим этапом была проверка результатов модели без дропаут слоев. Точность осталась примерно такой же – 77%.

Добавив L1-регуляризацию с коэффициентом 0.01, результаты сильно ухудшились, точность распознавания на валидационной выборке не превысила 50%. Итеративно уменьшая этот коэффициент, было найдено значение (0.00001), при котором точность достигла максимальное значение – 81%.

Однако, для дальнейших экспериментов и исследований было принято зафиксировать остальные гиперпараметры и выбрать максимально простую модель, без использования регуляризации и предобучения.

2.5 Эксперименты по-разному прореживанию сверточных слоев сети каждой стратегией

Было проведено много экспериментов, чтобы выяснить влияние процента зануляемых весов на точность распознавания. Количество эпох было зафиксировано на 40.

Процесс обучения был построен так:

Необходимо: процент прореживания z, модель сети, генератор данных.

1. **Цикл** i = 1, …, 2:
2. **Цикл** j = 1, …, 5:
3. Обычная эпоха обучения
4. **Конец цикла**
5. Расчет маски зануления по выбранной стратегии = mask_zeroing
6. **Цикл** j = 1, …, 5:
7. Обычная эпоха обучения
8. Зануление весов по маске mask_zeroing
9. **Конец цикла**

10. **Конец цикла**
11. **Цикл** i = 1, …, 5:
12. Обычная эпоха обучения
13. **Конец цикла**
14. **Цикл** i = 1, …, 3:
15. Расчет маски зануления по выбранной стратегии = mask_zeroing
16. **Цикл** j = 1, …, 5:
17. Обычная эпоха обучения
18. Зануление весов по маске mask_zeroing
19. **Конец цикла**
20. Сброс маски mask_zeroing
21. **Конец цикла**

Возврат: обученная модель.

В процессе обучения, записывалась точность на валидационной выборке в конце каждой эпохи, в файл csv. Для того, чтобы оценить влияние прореживания, описанный процесс был повторен для всех значений прореживания в пределах от 20% до 99%, с шагом в 5% от 20% до 90% и с шагом в 3% от 90% до 99%. В начале каждого запуска бралась пустая начальная модель.

Поместить результаты каждого запуска в работу нет смысла, достаточно сравнить между собой стратегии обычного обучения и №1, №2 и №3 на нескольких ключевых точках:

- При 20% прореживании, дабы показать, что с таким маленьким процентом прореживания стратегия не оказывает большого влияния на точность распознавания;

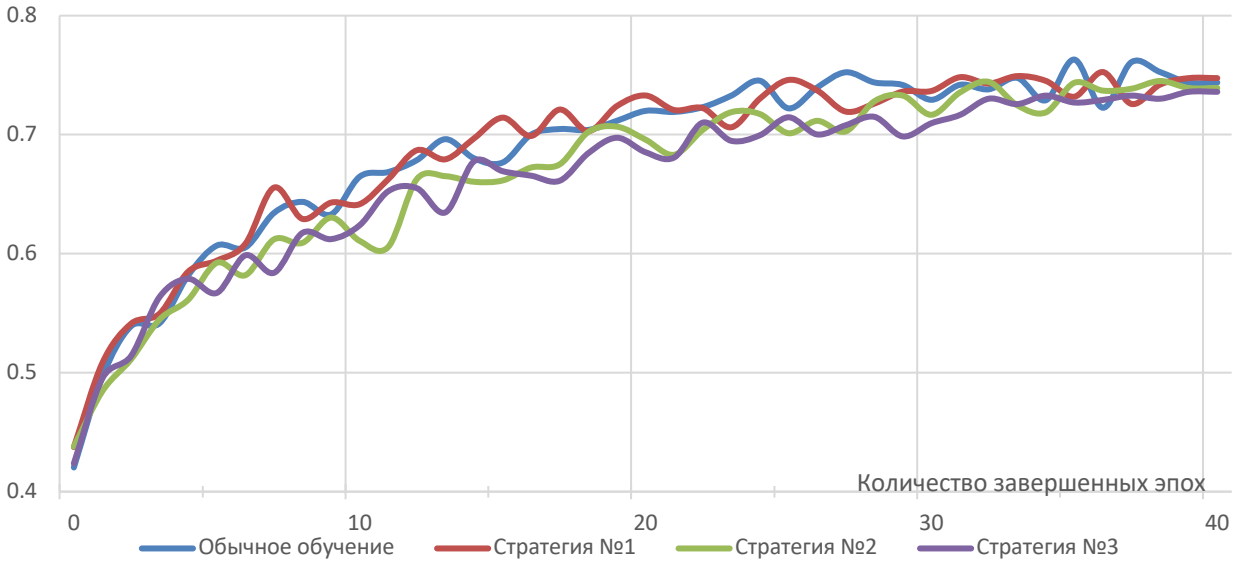

Рис. 18. График точности алгоритма при обучении. Отображен обычный процесс обучения, а также стратегии прореживания №1, №2, №3 с параметром прореживания 20%

- При 30% прореживании, когда стратегия №3 начала отставать в точности от остальных стратегий;

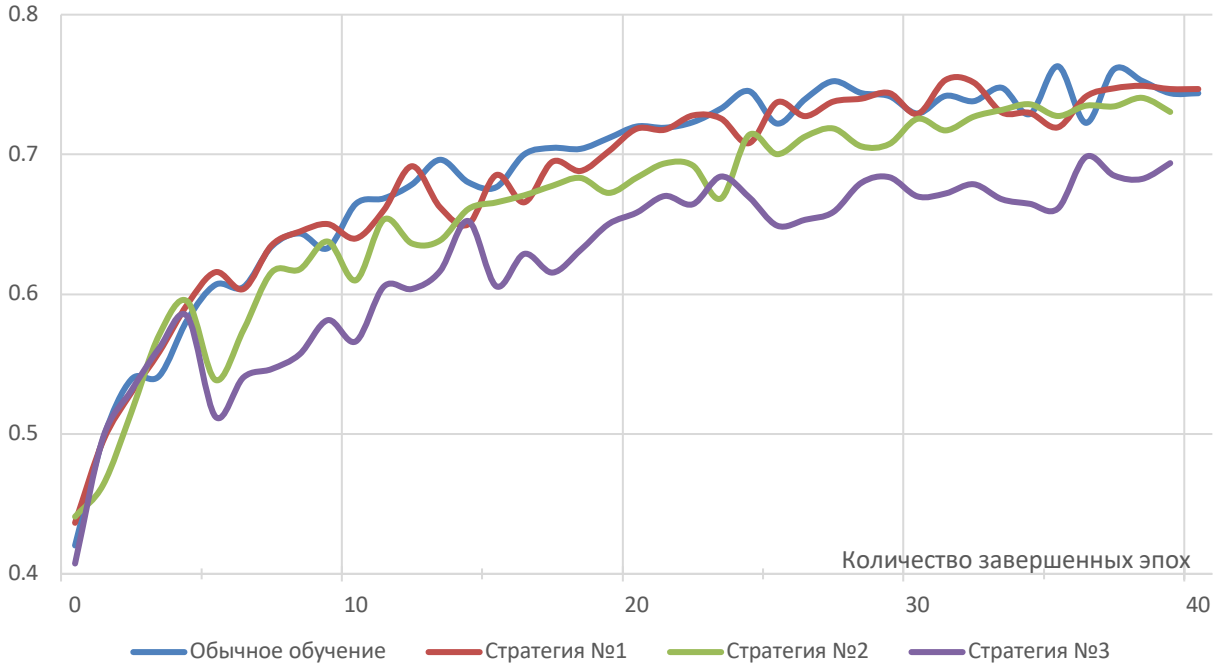

Рис. 19. График точности алгоритма при обучении. Отображен обычный процесс обучения, а также стратегии прореживания №1, №2, №3 с параметром прореживания 30%

- При 60% прореживании, для отслеживания динамики, насколько стратегия №2 хуже дает результаты, по сравнению со стратегией №1;

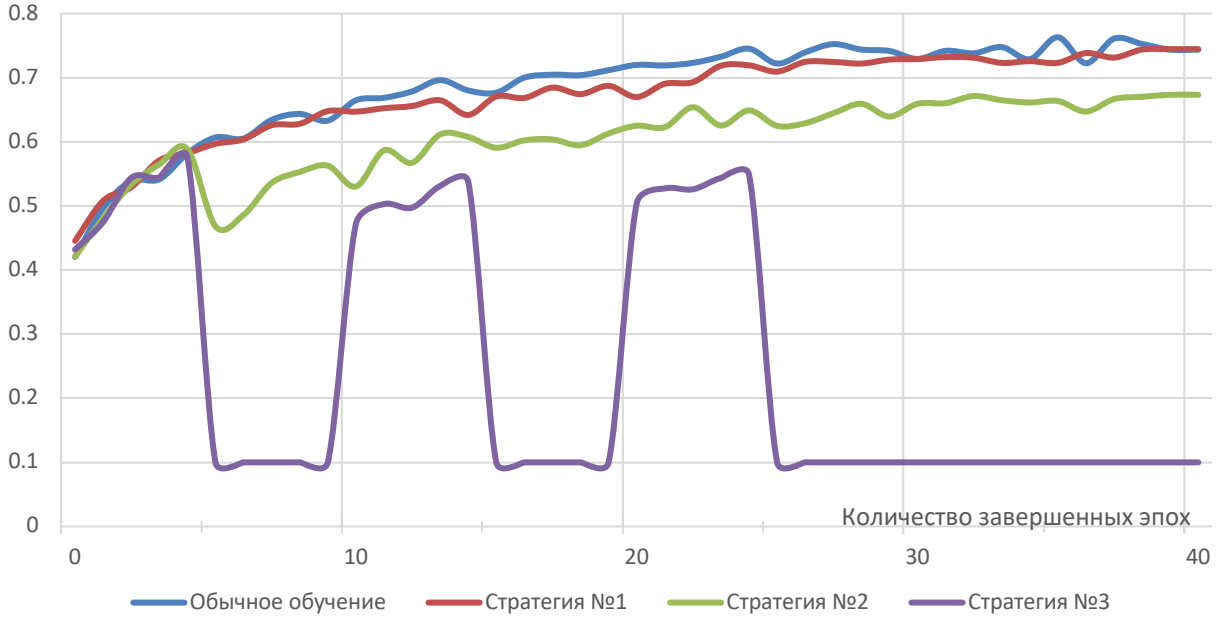

Рис. 20. График точности алгоритма при обучении. Отображен обычный процесс обучения, а также стратегии прореживания №1, №2, №3 с параметром прореживания 60%

- При 90% прореживании, при котором стратегия №1 дает еще вполне приемлемый результат, когда стратегия №2 уже сдала позиции.

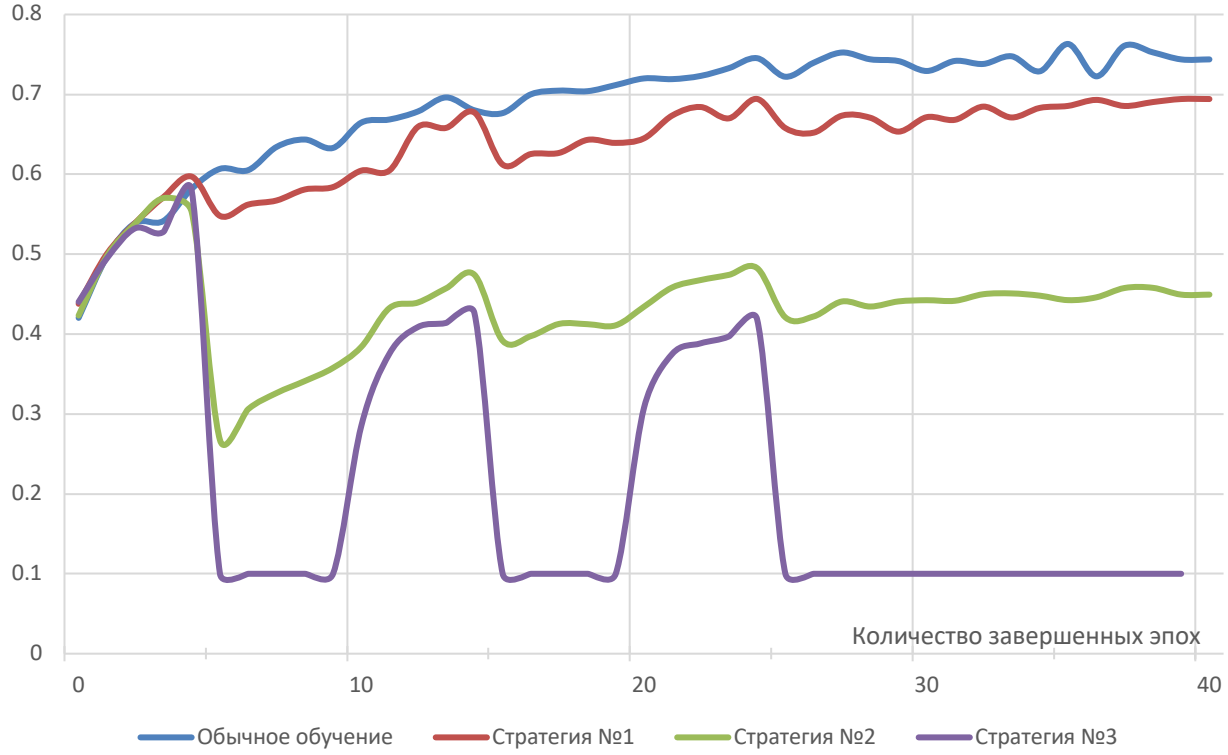

Рис. 21. График точности алгоритма при обучении. Отображен обычный процесс обучения, а также стратегии прореживания №1, №2, №3 с параметром прореживания 90%

2.6 Выводы

В данной главе изучены инструменты для проведения экспериментов со сверточными нейронными сетями. Были разработаны три разных стратегии прореживания. Каждая из стратегий была реализована на языке программирования Python, были проведены эксперименты по изучению влияния процента прореживания на точность работы алгоритма.

По результатам проведенных экспериментов понятно, что стратегия №3 самая неэффективная, так как даже при маленьком прореживании в 35%, точность катастрофически падает. Стратегия №2 позволяет занулить около 30% весов, при сохранении точности. Однако, фаворитом оказалась стратегия №1. В рамках исследованной задачи, она позволяет занулить более 60% весов, практически не потеряв в точности распознавания.

Кроме того, на момент написания работы, не было обнаружено статей с эффективной реализацией стратегии №1, когда в [7] реализован аналог стратегии №2. Поэтому, в следующей главе будет исследовано влияние стратегии №1 на время исполнения алгоритма на мобильном устройстве.

3. ИЗУЧЕНИЕ ВЛИЯНИЕ ПРОРЕЖИВАНИЯ НА ВРЕМЯ ИСПОЛНЕНИЯ

В прошлой главе было произведено исследование влияния разных стратегий прореживания на точность распознавания алгоритма. Так как уже существуют работы, которые уже применили аналогию прореживания или выбрасывания целых каналов сверточных фильтров, например [7] с изучением времени исполнения, в данной работе будет изучено влияние другой стратегии. А именно стратегии №1 из прошлой главы. Особенный интерес в том, насколько возможно улучшить производительность при больших процентах прореживания, при этом не пожертвовав сильно точностью работы алгоритма.

Для изучения влияния прореживания необходимо ознакомиться с возможными инструментами реализации стратегии №1. При том, надо выбрать инструмент среди наиболее популярных, для повышения ценности данной работы и возможной интеграции алгоритма в комплексное решение программного обеспечения. Таким образом, рассмотрим инструменты, которые могут быть использованы в повседневной жизни для написания простой многоканальной двумерной свертки и модифицированной двумерной свертки.

3.1 Инструменты для запуска свертки на центральном процессоре

Так как необходимо выбрать инструмент, позволяющий получить максимальную производительность при работе свертки, то рассмотрим язык программирования C. У этого подхода в данном контексте есть два плюса: простота реализации и наличие специальных инструкций SIMD типа, например SSE. SIMD – Single Instruction Multiple Data – принцип компьютерных вычислений, одиночный поток инструкций для работы с множественными потоками данных.

Однако, даже использование этих инструкций не позволяет добиться производительности, которая доступна используя инструменты для работы на графическом процессоре.

3.2 Инструменты для запуска свертки на графическом процессоре

У графического процессора есть особенность, которая очень сильно улучшает время работы свертки. Так как центральный процессор проектировался изначально под последовательные вычисления, графический процессор сразу разрабатывался под параллельные вычисления. Именно параллельная обработка входных данных свертки позволяет многократно улучшить производительность сверточного фильтра.

3.2.1 Запуск на графическом процессоре персонального компьютера

Компания Nvidia, один из лидеров на рынке персональных компьютеров, в 2007 году представила платформу для параллельных вычислений на графическом процессоре. Эта платформа получила название CUDA и она открывает разработчикам доступ к виртуальному набору команд графического процессора и параллельным вычислительным элементам для выполнения их на ядрах графического процессора, которых во много раз больше, по сравнению с центральным процессором.

Платформа CUDA поддерживает работу с такими популярными языками программирования, как C, C++ и Fortran. Что позволяет программисту, не обладающему узко специализированными знаниями Direct3D и OpenGL, закладывать параллельное использование ресурсов графического процессора в свою разработку. Однако, эта платформа спроектирована только для графического процессора персонального компьютера, а не мобильных устройств.

3.2.2 Запуск на графическом процессоре мобильного устройства

Именно разработка высокопроизводительных алгоритмов машинного обучения сейчас имеет особую актуальность, из-за стремительного роста популярности мобильных устройств и меньшим количеством доступной, по сравнению с персональным компьютером, вычислительной мощности.

Рассмотрим Metal Performance Shaders – высоко оптимизированная библиотека графических функций, созданная компанией Apple. Которая позволяет разработчикам приложения достичь высокой производительности благодаря использованию графического процессора. Именно эта библиотека будет использована в последующих экспериментах в этой главе, так как она обладает качественной документацией, большим количеством поддерживаемых мобильных устройств и огромным разнообразием примеров в сети интернет, что существенно упростит разработку.

3.3 Разработка свертки с прореживанием весов на Metal

Так как цель эксперимента в этой главе измерить улучшение в производительности при разных параметрах прореживания, необходимо так же измерить базовое время – то есть, сколько времени будет занимать аналогичная свертка, но без оптимизации в виде прореживания. То есть, необходимо разработать и обычную свертку, и свертку с прореживанием, для чистоты эксперимента.

Решено было использовать texture2d_array в качестве типа входящих и выходящих данных, так как этот тип позволяет хранить несколько двухмерных матриц, что отлично подходит для реализации многоканальной свертки. Несмотря на то, что обычно все текстуры на GPU являются четырехканальными – RGBA (красный, зеленый, синий, альфа канал), в данном эксперименте все данные будут храниться в красном канале для упрощения процесса разработки.

Функция обычной свертки на Metal принимает входящую картинку в texture2d_array с доступом на чтение, веса ядра свертки в texture2d_array с доступом на чтение и позицию вызываемой функции в сетке потоков. А на выходе получается результат свертки, также в texture2d_array с доступом на запись. В качестве входящей картинки и весов ядра свертки, подавался шум для того, чтобы исключить кеширование результатов прошлого эксперимента и не испортить эксперимент.

Алгоритм ядра шейдера обычной свертки:

Необходимо: массив двумерных текстур inTex, convKern, outTex; позиция потока в сетке gid.

1. res – переменная для хранения результата свертки, инициализируем 0
2. **Цикл** i = 1, …, высота(convKern):
3. **Цикл** j = 1, …, ширина(convKern):
4. считаем позицию пикселя в inTex. x = gid.x + i, y = gid.y + j
5. добавляем к res перемножение convKern[i, j] и inTex[y, x]
6. **Конец цикла**
7. **Конец цикла**
8. запись res outTex по позиции gid

Возврат: заполненная результатами вычислений outTex.

Функция свертки с прореживанием на Metal получает все данные точно также, как и обычная свертка. Но дополнительно добавляется texture1d с доступом на чтение. Эта текстура содержит на нулевой позиции количество активных весов, для исключения лишних проверок во время выполнения свертки. Последующие записи в этой текстуре обозначают индексы весов, которые будут использованы в свертке. Соответственно, весы, которые не будут использованы в свертке, будут считаться зануленными. Наличие этих данных

дает возможность итерироваться только по нужным весам, что позволяет уменьшить количество базовых операций для расчета одного результата свертки от размера ядра свертки в квадрате до количества незануленных весов, что теоретически должно дать уменьшение время работы свертки при увеличении процента зануляемых весов.

Алгоритм ядра шейдера свертки с прореживанием:

Необходимо: массив двумерных текстур inTex, convKern, outTex; одномерный массив индексов weightsInd, позиция потока в сетке gid.

1. res – переменная для хранения результата свертки, инициализируем 0
2. indNum – считываем количество активных весов, weightsInd[0]
3. **Цикл** i = 1, …, indNum + 1:
4. считаем индексы активного веса, wInd = weightsInd[i + 1]
5. считаем позицию пикселя в inTex. x =gid.x + wInd.x, y =gid.y + wInd.y
6. добавляем к res перемножение convKern[wInd.y, wInd.x] и inTex[y, x]
7. **Конец цикла**
8. запись res outTex по позиции gid

Возврат: заполненная результатами вычислений outTex.

3.4 Эксперимент на влияние времени исполнения от процента прореживания весов свертки на Metal

Как следует из прошлой части, где описывались обе свертки, у свертки с прореживанием присутствует небольшое усложнение, которое может сказаться, если использовать ее с нулевым процентом прореживания. Поэтому, производились замеры обычной свертки и свертки с прореживанием от 0% до 75%, с шагом в 25%.

Кроме того, в ходе эксперимента была обнаружена определенная граница по размерам ядра свертки, после преступления которой, время работы обоих сверток начинало сильно расти. Поэтому, были проведены эксперименты с разными размерами входящей картинки и разными размерами ядра. Наличие такой границы связываем с превышением размера кэш памяти, что приводит к лишним обращениям в оперативную память и сильно замедляет выполнение. Так же, чтобы нивелировать эти скачки по времени на графике и отобразить только интересующую нас информацию, на графике будет представлено соотношение времени работы обычной свертки, деленное на соответствующее время работы свертки с прореживанием, то есть, во сколько раз определенное прореживание ускоряет свертку.

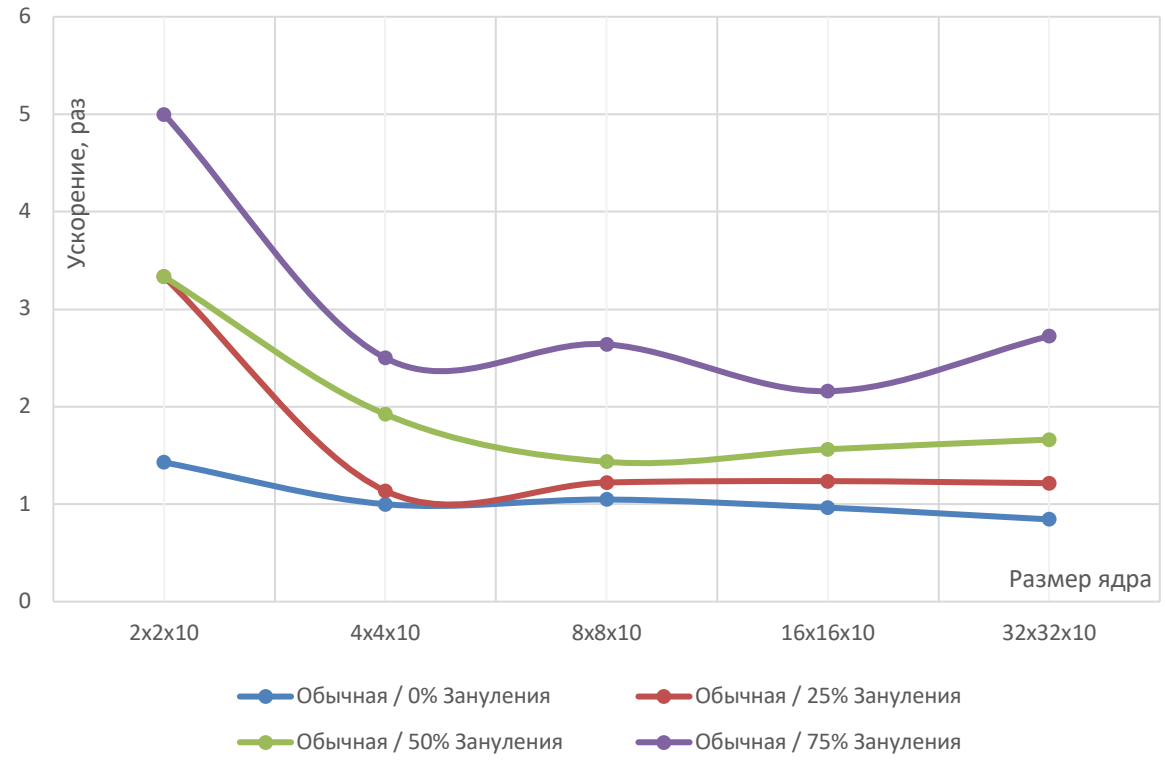

Рис. 22. График ускорения алгоритма от размера ядра свертки. Входная текстура 500x500x10

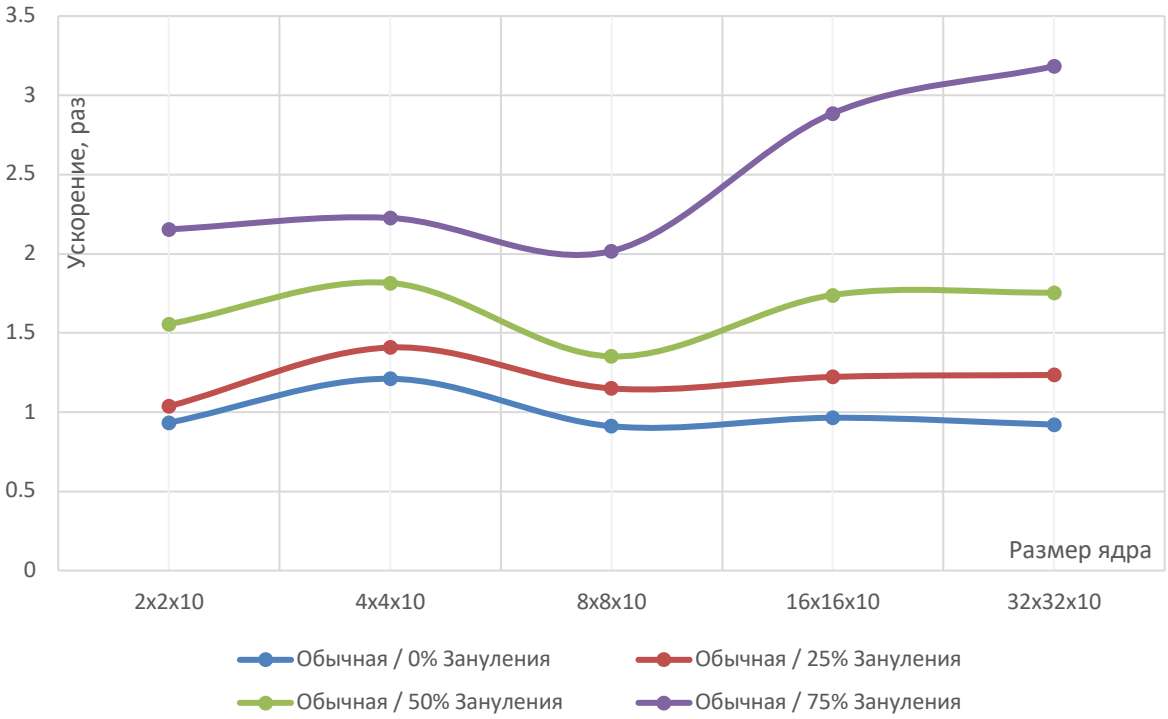

Рис. 23. График ускорения алгоритма от размера ядра свертки. Входная текстура 1000x1000x10

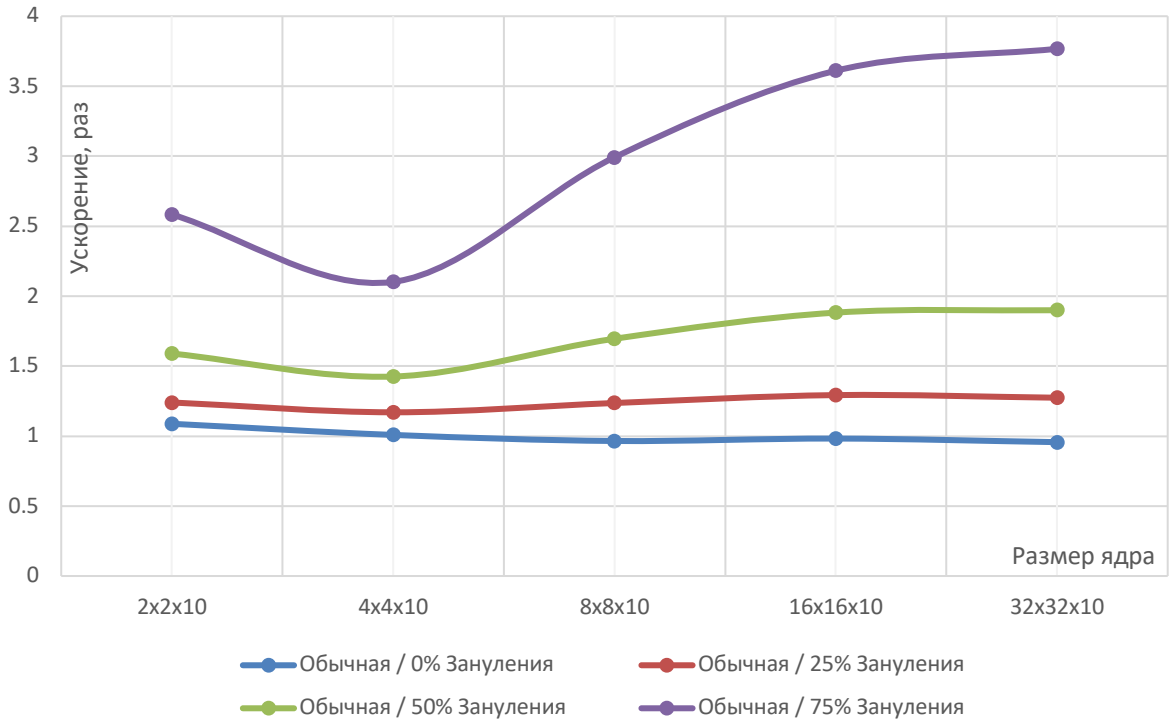

Рис. 24. График ускорения алгоритма от размера ядра свертки. Входная текстура 2000x2000x10

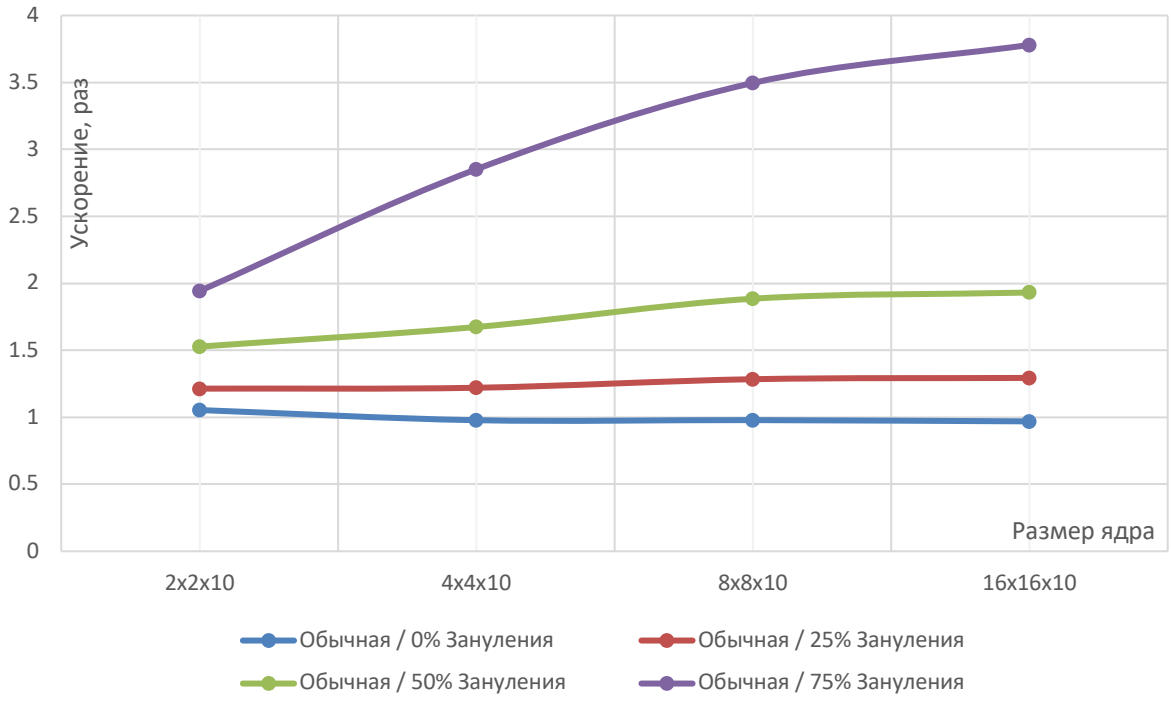

Рис. 25. График ускорения алгоритма от размера ядра свертки. Входная текстура 4000x4000x10

На последнем эксперименте, при входной текстуре 4000x4000x10 и размере ядра 32x32x10, обе свертки завершались неуспешно с ошибкой. Видимо, был преодолена временная граница ожидания ответа от графического процессора. Поэтому, данный результат отсутствует на графике.

Важно так же отметить, что перечисленные эксперименты проводились на iPhone Xs с версией iOs 13.4.1. Результаты на других версиях прошивок или на других мобильных устройствах будут отличаться.

Таким образом, из данного эксперимента следует, что используя данную свертку с 75% прореживанием весов, можно получить ускорение от двух раз и больше, не потеряв в точности работы алгоритма.

Размер ядра	Обычная свертка	Быстрая свертка			
		0% занулено	25% занулено	50% занулено	75% занулено
2x2x10	20	140	6	6	4
4x4x10	25	25	22	13	10
8x8x10	66	63	54	46	25
16x16x10	136	141	110	87	63
32x32x10	384	455	316	231	141

Таблица 2. Время в миллисекундах, требуемой для вычисления одной свертки, с выходной текстурой 500x500x10

Размер ядра	Обычная свертка	Быстрая свертка			
		0% занулено	25% занулено	50% занулено	75% занулено
2x2x10	28	30	27	18	13
4x4x10	69	57	49	38	31
8x8x10	123	135	107	91	61
16x16x10	407	422	333	234	141
32x32x10	1337	1451	1083	762	420

Таблица 3. Время в миллисекундах, требуемой для вычисления одной свертки, с выходной текстурой 1000x1000x10

Размер ядра	Обычная свертка	Быстрая свертка			
		0% занулено	25% занулено	50% занулено	75% занулено
2x2x10	62	57	50	39	24
4x4x10	124	123	106	87	59
8x8x10	356	369	288	210	119
16x16x10	1325	1349	1025	704	367
32x32x10	5078	5309	3986	2674	1348

Таблица 4. Время в миллисекундах, требуемой для вычисления одной свертки, с выходной текстурой 2000x2000x10

Размер ядра	Обычная свертка	Быстрая свертка			
		0% занулено	25% занулено	50% занулено	75% занулено
2x2x10	136	129	112	89	70
4x4x10	365	373	299	218	128
8x8x10	1318	1346	1026	699	377
16x16x10	5107	5266	3946	2644	1351
32x32x10	–	–	–	–	5264

Таблица 5. Время в миллисекундах, требуемой для вычисления одной свертки, с выходной текстурой 4000x4000x10

На представленных таблицах заметно, что при выбранной стратегии и 75% разреженности, время вычисления одной свертки совпадает с временем вычисления обычной свертки, но с размером ядра в четыре раза меньше. Само собой, имеется в виду, что обе свертки получают на вход текстуру одинаково размера. Таким образом, применение стратегии разреженного представления весов №1, при использовании 75% разреженности, относительно обычной стратегии позволяет сохранить производительность при увеличении ядра свертки в 4 раза. Увеличение размера ядра свертки позволит сверточной сети в процессе обучения обнаружить новые признаки, что потенциально должно улучшить точность. Поэтому, для оптимизации времени работы сверточной сети, рациональнее использовать стратегию разреженного представления весов №1, а не уменьшение размеров фильтров сверток.

3.5 Выводы

В данной главе изучены инструменты написания высокоэффективных алгоритмов для выполнения сверточных нейронных сетей, как на центральном процессоре, так и на графическом процессоре. Рассмотрены инструменты как для запуска на стационарном компьютере, так и на мобильном устройстве.

Стратегия разреженного представления весов сети №1 была реализована в виде оптимизированной свертки, используя библиотеку Metal на мобильном устройстве. Проведены эксперименты для измерения влияния значения параметра разреженности на улучшение производительности свертки.

По результатам проведенных экспериментов видно, что стратегия разреженного представления позволяет ускорить работу свертки более чем в 2 раза при использовании 75% разреженности весов. При этом, согласно экспериментам в второй главе, 75% разреженности весов дает не более чем 3% потери точности на тестовой выборке.

ЗАКЛЮЧЕНИЕ

Таким образом, были исследованы сверточные нейронные сети и их основные компоненты. Информация была изучена, результаты исследований приведены в отчете. Также была исследована работа на тему L0 регуляризации [11]. Было выявлено, что выбранный подход приведет не только к уменьшению занимаемого объема, но также уменьшит значение ошибки и увеличит скорость работы нейронной сети. Были разработаны три разных стратегии для прореживания весов нейронной сверточной сети, применимость которых также была изучена в работе.

Так же было изучено и выбрано окружение для проведения экспериментов для прореживания весов. Было разработано программное обеспечение для создания маски прореживания по трем стратегиям. Также было разработано программное обеспечение для прореживания весов модели согласно созданной маске прореживания. Перечисленное программное обеспечение было отлажено и протестировано наглядным способом в нынешней работе.

Был проведен эксперимент по изучению влияния процента зануляемых весов сверточных слоев на точность распознавания тремя разными стратегиями. По результатам этого эксперимента можно сказать, что первая стратегия позволяет сохранить хорошую точность даже при большом проценте прореживании весов.

Изучены способы программирования двух типов сверток на разные платформы: от центрального процессора стационарного компьютера, до графического процессора мобильного устройства. Была изучена библиотека Metal и с ее помощью реализованы обе свертки. Написанные свертки были протестированы и отлажены. Был поставлен эксперимент по изучению зависимости времени работы свертки от процента прореживания весов. Результаты были обработаны, проанализированы и внесены в работу. По результатам второго эксперимента получается, что используя первую стратегию прореживания и зануляя около 75% процентов весов многоканальной свертки, можно получить более чем двухкратное повышение производительности алгоритмов, использующих свертку.

В дальнейшем планируется расширить первый эксперимент: испробовать методы прореживания также других весов (биас весов и обоих весов сразу). Также запланированы дополнительные эксперименты для изучения влияния прореживания весов полносвязных и других слоев нейронных сетей на их точность распознавания.

Планируется также проверить применимость решение прореживания весов на задаче, более близкой к распознаванию позы – сегментационной задаче.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Computer vision, Wikipedia [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Computer_vision
2. Adit Deshpande, A Beginner's Guide To Understanding Convolutional Neural Networks [Электронный ресурс]. – Режим доступа: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
3. Kailash Ahirwar, Everything you need to know about Neural Networks [Электронный ресурс]. – Режим доступа: https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491
4. Артемов А., Сверточные нейронные сети [Электронный ресурс]. – Режим доступа: http://www.machinelearning.ru/wiki/images/1/1b/DL16_lecture_3.pdf
5. Jonathan Long, Evan Shelhamer, Trevor Darrell, Fully Convolutional Networks for Semantic Segmentation [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1411.4038
6. Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner, GradientBased Learning Applied to Document Recognition [Электронный ресурс]. – Режим доступа: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf
7. Andrew G. Howard, Menglong Zhu, Bo Chen, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications Recognition [Электронный ресурс]. – Режим доступа: https://arxiv.org/pdf/1704.04861.pdf
8. Павел Гладков, Простое руководство по дистилляции BERT [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/avito/blog/485290/
9. Raphael Tang, Yao Lu, Linqing Liu, Distilling Task-Specific Knowledge from BERT into Simple Neural Networks, [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1903.12136
10. Grigory Sapunov, Speeding up BERT, [Электронный ресурс]. – Режим доступа: https://blog.inten.to/speeding-up-bert-5528e18bb4ea

11. Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, Oriol Vinyals, Understanding deep learning requires rethinking generalization [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1611.03530
12. Dmitry Molchanov, Arsenii Ashukha, Dmitry Vetrov, Variational Dropout Sparsifies Deep Neural Networks [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1701.05369
13. Christos Louizos, Max Welling, Diederik P. Kingma, Learning Sparse Neural Networks through L0 Regularization [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1712.01312
14. Alanchiao, Sparsity Runtime Integration with TF/TFLite for Latency Improvements [Электронный ресурс]. – Режим доступа: https://github.com/tensorflow/model-optimization/issues/173
15. Параллельные вычисления CUDA [Электронный ресурс]. – Режим доступа: https://www.nvidia.ru/object/cuda-parallel-computing-ru.html
16. Keras: The Python Deep Learning library CPU [Электронный ресурс]. – Режим доступа: https://keras.io/
17. Nikita Mishchanka, sparse-net-study [Электронный ресурс]. – Режим доступа: https://github.com/GTnikito/sparse-net-study
18. Joosephook, cifar10_cnn [Электронный ресурс]. – Режим доступа: https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py