ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ (ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ)»

На правах рукописи

Бабенко Артем Валерьевич

ЭФФЕКТИВНЫЕ АЛГОРИТМЫ ПОИСКА ПО БОЛЬШИМ КОЛЛЕКЦИЯМ ИЗОБРАЖЕНИЙ

Специальность 05.13.11 —
«Математическое и программное обеспечение вычислительных машин,
комплексов и компьютерных сетей»

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
кандидат физико-математических наук,
доцент Сколковского института
науки и технологий
Лемпицкий Виктор Сергеевич

Москва — 2017

Оглавление

Стр.

Основные понятия и обозначения

Архитектура глубокой нейросети — структура связи между нейронами в нейросети, их число, тип функции активации.

Дескриптор (глобальный, локальный) — вектор фиксированной длины, характеризующий визуальные или семантические свойства изображения. Если дескриптор характеризует изображение в целом, то он называется глобальным, а если некоторый регион изображения — локальным.

Индексирующая структура — структура данных, позволяющая ускорить процедуру поиска по большим коллекциям, путем отбора небольшого числа элементов коллекции, наиболее близких к конкретному запросу. Индексацией называется процедура построения индексирующей структуры для поисковой коллекции.

Квантизация — приближение значений некоторой величины некоторым конечным множеством значений. В случае многомерной приближаемой величины квантизация называется векторной. Множество значений квантизованной величины обычно называется словарем и обозначается C (от англ. codebook).

Мультиквантизация — способ квантизации значений D-мерной величины $(x_1, \ldots, x_D)$, при котором значения M подмножеств соседних размерностей $\left(x_1, \ldots, x_{\frac{D}{M}}\right), \left(x_{\frac{D}{M}+1}, \ldots, x_{\frac{2D}{M}}\right), \ldots, \left(x_{\frac{D(M-1)}{M}+1}, \ldots, x_D\right)$ квантизуются независимо с различными словарями $C_1, \ldots, C_M$. Словарем мультиквантизации является множество кортежей из декартова произведения $C_1 \times \cdots \times C_M$.

Нейрон — функция, осуществляющая взвешенное суммирование всех входных аргументов и применяющая к полученному значению линейной комбинации нелинейное преобразование. Тип нелинейного преобразования называется **функцией активации** нейрона.

Нейронная сеть (нейросеть) прямого распространения — модель машинного обучения, предполагающая определенную зависимость между целевой переменной и наблюдаемыми переменными. В этой модели целевая переменная может быть вычислена по наблюдаемым переменным как результат применения композиции некоторого числа нелинейных функций-нейронов. Нейроны, аргументами которых являются непосредственно наблюдаемые переменные называются нейронами первого слоя, их выходы являются входами нейронов вто-

рого слоя, и т.д. Выходом нейронов последнего слоя является значение целевой переменной. Настраиваемыми параметрами в модели нейросети являются веса линейных комбинаций в каждом из нейронов.

Поле восприятия нейрона — совокупность пикселей изображения, от интенсивности которых зависит значение нейрона.

Полносвязный слой — слой, в котором входными аргументами каждого нейрона являются выходы всех нейронов с предыдущего слоя.

Полнота@T (полнота выдачи длины T) — метрика качества нахождения ближайшего соседа. Значение метрики равняется доле запросов, для которых истинный ближайший сосед оказался в топе размера T выдачи, сформированной в результате поиска.

Прямой унитарный код — способ представления натуральных чисел в диапазоне $[1, K]$ векторами размерностью K. Число $k \in [1, K]$ представляется вектором, в котором k-ая координата равна единице, а все остальные равны нулю.

Пулинг, макс-пулинг, суммирующий пулинг — операция агрегации значений нейронов некоторой пространственной области сверточного слоя определенным образом. При макс-пулинге выходом является максимальное среди агрегируемых значений, при суммирующем пулинге — их сумма.

Сверточный слой — слой, выходами нейронов которого является результат применения операции свертки с одной или несколькими ядрами к нейронам предыдущего слоя. Сверточный слой подразумевает, что совокупность его входов имеет пространственную структуру, что допускает применение операции свертки. В случае изображений наблюдаемые переменные (значения интенсивностей пикселей) имеют структуру массива определенной высоты и ширины, к которому можно применить операцию двухмерной свертки. Результатом такой операции также будет двумерный массив. Обычно в каждом слое применяется операция свертки с несколькими ядрами, поэтому выходом сверточного слоя является совокупность двухмерных массивов, называемых **картами** или **каналами**.

Словарь — конечное множество значений, которое может принимать некоторая величина после процедуры квантизации. Элементы словаря называются словами.

Средняя точность поиска — стандартная метрика качества поиска для сравнения различных методов. Как правило, в тестовых коллекциях имеется некоторое количество запросов, для которых известны правильные ответы, то есть изображения, содержащие тот же объект, что и изображение-запрос. Для конкретного запроса точность поиска для выдачи длины K равна доле правильных ответов в выдаче длины K, сформированной в результате поиска. Средней точностью поиска для каждого запроса является среднее всех значений точности по всем выдачам, длины которых равны позициям правильных ответов для него. Усредненное по всем запросам значение средней точности поиска является итоговой метрикой качества метода.

Введение

Данная диссертационная работа посвящена задаче поиска по большим коллекциям изображений. В этой задаче для поискового запроса-изображения необходимо с помощью поисковой системы найти все изображения из большой коллекции, содержащие ту же сцену или объект, что и запрос.

Благодаря повсеместному распространению цифровых фотоаппаратов и видеокамер, а также популярности различных социальных сетей, в Интернете ежедневно появляются миллионы новых фотографий. В связи с этим, современным поисковым системам приходится обрабатывать коллекции из миллиардов изображений, причем размер этих коллекций постоянно увеличивается. Для эффективной индексации и поиска по коллекциям такого объема необходимо разрабатывать специальные алгоритмы, позволяющие отвечать на поисковые запросы пользователей за несколько миллисекунд.

Эффективные методы поиска по изображениям необходимы не только в работе поисковых систем, но в более узких областях, таких как распознавание лиц, диагностика заболеваний на основе медицинских изображений, приложения для туризма и другие. Ключевым элементом каждого из этих приложений является поиск похожих, при котором для запроса необходимо найти семантически близкие изображения из большой коллекции.

Типичная схема работы системы визуального поиска представлена на рисунке 1. Поисковая система получает на вход изображение-запрос, затем представляет его в виде одного или нескольких векторов фиксированной длины. Эти векторы принято называть *дескрипторами* изображения. Все изображения из поисковой коллекции также представлены в ней в виде своих дескрипторов. Далее система находит изображения, дескрипторы которых оказались ближайшими к дескриптору запроса в смысле евклидовой метрики, и возвращает их пользователю в качестве результатов поиска. Описанный протокол работы визуального поиска сформировался в начале 90-ых годов с появлением первой системы QBIC (Query by Image Content) [1] и действует в настоящее время в современных сервисах Яндекс и Google.

Существует три основных направления исследований, актуальных для задач поиска по большим коллекциям изображений.

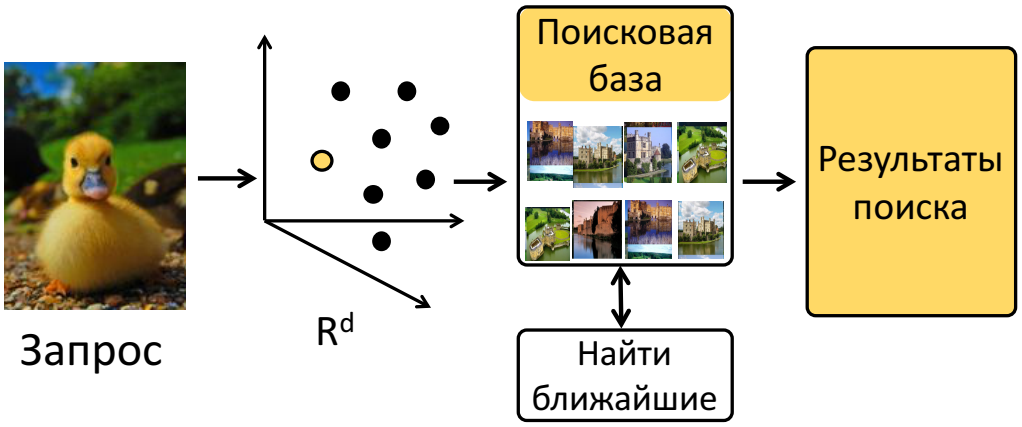


Рисунок 1 — Типичная схема поиска по изображениям в современных поисковых системах.

1. *Получение дескрипторов.* В первую очередь необходимо предложить алгоритм построения дескрипторов. Важно, чтобы дескрипторы изображений, которые содаржат семантически похожие объекты или сцены, были близки в смысле евклидовой метрики многомерного евклидового пространства. Самые первые системы визуального поиска [1–3] использовали простейшие дескрипторы: гистограммы цветов, описания текстур, информацию о границах и формах объектов на изображении. Такие тривиальные дескрипторы не могли обеспечить инвариантности к изменениям освещенности, поворотам, изменениям масштаба и угла зрения, необходимых для качественного поиска. В 1999 году в работе [4] были предложены СИФТ дескрипторы (SIFT, scale-invariant feature transform), основанные на гистограммах градиентов интенсивности и в некоторой степени обладавшие необходимыми типами инвариантности. Благодаря своей простоте и неплохому качеству работы СИФТы вплоть до недавнего времени были основным инструментом получения дескрипторов для визуального поиска. Для нескольких регионов изображения вычислялись локальные СИФТ векторы, и полученное множество являлось глобальным дескриптором изображения. В случае больших коллекций хранение всего множества локальных векторов для каждого изображения невозможно, так как их суммарный размер может превышать размер оперативной памяти поисковых серве-

ров. Для решения этой проблемы было разработано несколько методов агрегации множества локальных векторов в один глобальный вектор-дескриптор [5–7]. На данный момент основной интерес исследователей сосредоточен именно на задаче построения глобальных дескрипторов, так как их использование дает возможность эффективного поиска по большим коллекциям. В последние годы важным инструментов получения таких дескрипторов становятся глубокие сверточные нейросети. Векторы выходов нейронов, получаемых при применении нейросети к похожим изображениям, оказываются близкими, что позволяет использовать их в качестве глобальных дескрипторов.

2. *Сжатие дескрипторов.* Если коллекция содержит миллиарды изображений, то, даже в случае глобальных дескрипторов, их объем может намного превышать объем доступной оперативной памяти. Поэтому приходится сжимать имеющиеся дескрипторы и представлять каждый из них в виде компактного кода размером до нескольких байт. Такие компактные коды должны предоставлять возможность эффективно вычислять евклидово расстояние между сжатыми дескрипторами или несжатым дескриптором пользовательского запроса и сжатым дескриптором изображения из поисковой коллекции. Исторически первым подходом для решения этой задачи был подход семантического хэширования, предложенный в 2007 году в работе [8]. В этом подходе действительнозначный вектор-дескриптор отображался в бинарную строку, причем близкие в исходном пространстве векторы отображались в строки, расстояние Хэмминга между которыми мало. В дальнейшем идея такого бинарного кодирования получила развитие в работах [9–11]. Практическое преимущество этих методов заключается в том, что вычисления на бинарных строках, как правило, гораздо быстрее операций с плавающей точкой. В 2011 году был предложен альтернативный подход к задаче сжатия, основанный на идее *мультиквантизации* [12]. Этот подход эффективно аппроксимирует каждый вектор конкатенацией небольшого числа базовых векторов-*слов* из фиксированных множеств, а номера слов являются кодом вектора. Получив поисковый запрос, можно предподсчитать расстояния до базовых слов, что затем дает возможность быстрого вычисления расстояний до сжатых векторов. Преимущество

мультиквантизации состоит в отсутствии необходимости сжатия запроса, что приводит к меньшим потерям информации по сравнению с методами бинарного кодирования. На данный момент различные модификации подхода мультиквантизации активно исследуются [13–15].

3. *Поиск ближайших.* Наконец, для данного дескриптора запроса необходимо эффективно находить ближайших соседей среди миллиардов дескрипторов изображений поисковой коллекции. На таких объемах полный перебор невозможен, поэтому приходится использовать более быстрые приближенные алгоритмы. Приближенный поиск ближайших соседей — очень популярная задача в современном анализе данных. Большинство существующих методов избегают полного перебора, разбивая поисковое пространство на большое число непересекающихся регионов. После получения запроса алгоритм проверяет только те регионы, которые являются наболее перспективными для этого запроса. Примерами самых известных подходов для решения этой задачи являются разделяющие деревья поиска [16], иерархический метод к-средних [17], локально-чувствительное хэширование [18]. Все перечисленные методы требуют существенных затрат по памяти, поэтому неприменимы к нахождению ближайших соседей среди миллиардов векторов. Первый подход, применимый к задачам такого масштаба, был предложен в 2011 году в работе [19] и демонстрирует приемлемую точность поиска со временем работы порядка нескольких десятков миллисекунд на запрос. Для многих приложений в реальном времени такой скорости недостаточно, поэтому построение более эффективных методов является важной исследовательской задачей.

Целью данной работы является разработка алгоритмов для эффективной системы визуального поиска, позволяющей осуществлять поиск по коллекциям, содержащих миллиарды изображений.

Для достижения поставленной цели необходимо было решить следующие задачи:

1. Исследовать существующие методы построения дескрипторов изображений и разработать метод, формирующий более компактные и семантически значимые дескрипторы.

2. Исследовать существующие методы сжатия векторов высокой размерности, оценить границы их применимости, разработать новый метод сжатия с меньшими потерями без увеличения бюджета потребляемой памяти.
3. Разработать новый алгоритм поиска ближайших векторов, способный обрабатывать запросы к коллекциям из миллиарда векторов за несколько миллисекунд с бюджетом по памяти несколько байт на вектор.

Основные положения, выносимые на защиту:

1. Разработано два новых метода построения дескрипторов изображений на основе глубоких сверточных нейросетей. Во-первых, исследовано качество дескрипторов с полносвязных слоев предобученной нейросети и предложен способ повышения качества путем адаптации используемой нейросети к конкретной поисковой задаче. Также показано, что качество предложенных дескрипторов практически не снижается при сжатии их методом главных компонент. Во-вторых, предложен новый метод извлечения дескрипторов со сверточных слоев нейросети, преимущество которого заключается в том, что дескрипторы могут быть вычислены для изображения любого размера и формы. На большом количестве тестовых коллекций показано, что предложенные методы формируют более компактные и семантически значимые дескрипторы по сравнению с существующими подходами.
2. Предложена новая модель аппроксимации векторов с помощью суммы слов, в отличие от конкатенации в модели мультиквантизации. На основе этой модели разработано два новых алгоритма сжатия векторов высокой размерности. Первый алгоритм — Аддитивная квантизация — не накладывает ограничений на значения слов и в теории приводит к наивысшему качеству сжатия, но требует решения сложной оптимизационной задачи, что делает кодирование неэффективным при больших длинах кода. Второй алгоритм — Древесная квантизация — накладывает определенные ограничения на слова, что снижает количество настраиваемых параметров, но обладает высокой скоростью кодирования. Доказана теорема о том, что ошибка реконструкции, достигаемая предложенными методами, не превосходит ошибки, достигаемой существую-

щими подходами мультиквантизации. Экспериментально показано, что разработанные методы достигают лучшего качества сжатия по сравнению с передовыми алгоритмами для данных различной природы.

3. Разработаны ортогональный и неортогональный инвертированные мультииндексы — две структуры данных для эффективного поиска ближайших соседей, способные искать по миллиардам высокоразмерных векторов за несколько миллисекунд. Мультииндексы разбивают поисковое пространство на большое число регионов определенной структуры, позволяющей эффективно находить наиболее перспективные регионы для заданного запроса. Комбинация эффективности и большого числа регионов в разбиении позволяет обрабатывать запросы за несколько миллисекунд при высоком уровне точности поиска, что показано экспериментально. Также доказан теоретический результат об эффективности основных операций в предложенных структурах данных.

Научная новизна:

1. Предложены новые методы построения дескрипторов изображений для поиска на основе глубоких сверточных нейросетей, превосходящие существующие подходы к построению дескрипторов.
2. Предложены новые методы сжатия векторов высокой размерности, использующие квантизацию с неортогональными словарями и превосходящие существующие методы сжатия по точности.
3. Разработаны новые структуры данных, способные искать по коллекциям из миллиарда векторов за несколько миллисекунд с высокой точностью.

Научная и практическая значимость. Теоретическая значимость работы заключается в предложенных структурах данных для эффективного нахождения ближайших соседей. Эти структуры могут быть использованы в качестве составляющих более сложных систем, например, для непараметрической классификации или непараметрического восстановления плотности. Также для этих структур получены оценки алгоритмической сложности основных операций. Практическая значимость работы заключается в разработке целостной системы визуального поиска, включающей все основные компоненты: построение

дескрипторов, сжатие, эффективное нахождение соседей. Предложенные подходы используются в поиске по изображениям компании ООО "Яндекс".

Степень достоверности полученных результатов обеспечивается доказанными теоретическими результатами, а также детально описанными протоколами экспериментов, которые могут быть воспроизведены. Программная реализация предложенных алгоритмов и все данные для экспериментов доступны в сети Интернет.

Апробация работы. Основные результаты работы докладывались на:

1. 53-я научная конференция МФТИ, Долгопрудный, 2011, тема доклада: "Инвертированный мультииндекс"
2. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Providence RI, USA, 2012, тема доклада: "The inverted multi-index"
3. 55-я научная конференция МФТИ, Долгопрудный, 2013, тема доклада: "Аддитивная квантизация"
4. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, USA, 2014, тема доклада: "Additive quantization for extreme vector compression"
5. IEEE European Conference on Computer Vision (ECCV), Zurich, Switzerland, 2014, тема доклада: "Neural codes for image retrieval"
6. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, USA, 2015, тема доклада: "Tree quantization for large-scale similarity search and classification"
7. Научный семинар Школы анализа данных Яндекса, Москва, 2015, тема доклада: "Древесная квантизация для компактного хранения векторов".
8. IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, тема доклада: "Aggregating local deep features for image retreival"
9. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, USA, 2016, тема доклада: "Efficient Indexing of Billion-Scale Datasets of Deep Descriptors"
10. Научно-исследовательский семинар им. М. Р. Шура-Бура, ИПМ им. М.В.Келдыша, Москва, 2016, тема доклада: "Инвертированный мультииндекс"

Личный вклад диссертанта состоит в исследовании существующих методов, разработке предложенных алгоритмов и структур данных, доказательстве теоретических результатов, программной реализации предложенных подходов.

Диссертация состоит из введения, трех глав, заключения и списка литературы. Первая глава посвящена методам получения дескрипторов изображений для визуального поиска. В ней приводится обзор существующих подходов и предлагается два новых метода, основанных на использовании глубоких сверточных нейросетей. Во второй главе описываются методы сжатия векторов высокой размерности, приводится обзор существующих моделей сжатия, а также предлагается две новых модели сжатия, основанные на аппроксимации кодируемого вектора суммой слов. В третьей главе исследуются методы поиска ближайших соседей среди миллиардов векторов. В ней описаны существующие схемы и предложено две новые структуры данных с гораздо более высокой скоростью работы. В заключении сформулированы основные результаты работы.

Публикации. Основные результаты по теме диссертации изложены в 6 изданиях [20–25], 6 из которых в изданиях, рекомендованных ВАК [20–25].

Объем и структура работы. Диссертация состоит из введения, четырёх глав, заключения и двух приложений. Полный объём диссертации составляет 151 страницу с 37 рисунками и 14 таблицами. Список литературы содержит 73 наименования.

Глава 1. Построение дескрипторов изображений с помощью глубоких нейронных сетей

За последние несколько лет глубокие сверточные нейросети стали основным инструментом при решении задач классификации изображений. В частности, было показано [26], что векторы активаций нейронов с верхних слоев нейросетей являются дискриминативными дескрипторами изображения, причем семантически близким изображениям соответствуют близкие по евклидовой метрике дескрипторы. В этой главе автором исследуются различные варианты построения нейросетевых дескрипторов.

Глава состоит из трех частей. В первой части приведен краткий обзор существующих методов построения дескрипторов изображения, не использующих сверточные нейросети.

Во второй части исследуются методы извлечения нейросетевых дескрипторов с полносвязных слоев. В первую очередь автор оценивает качество дескрипторов, построенных с помощью нейросети, предобученной на 1000 классов из коллекции ImageNet [26]. Качество оценивается на стандартных тестовых коллециях: INRIA Holidays [27], Oxford Buildings, Oxford Building 105K [28], и University of Kentucky benchmark (UKB) [17]. Оказывается, что даже такие простые дескрипторы обеспечивают вполне разумное качество поиска, хотя и проигрывают передовым существующим методам. Далее автором исследуется возможность адаптации нейросети под конкретную задачу поиска. Коллекции INRIA Holidays и Oxford Buildings в основном содержат фотографии зданий, поэтому для адаптации под эти коллекции нейросеть дообучалась на большой коллекции изображений, содержащих здания. Экспериментально подтверждено, что подобная адаптация под конкретную поисковую задачу существенно повышает качество нейросетевых дескрипторов. Наконец, во второй части также исследуется возможность сжатия получаемых дескрипторов с помощью метода главных компонент. Показано, что даже при высокой степени сжатия (вплоть до 128 компонент) потери в качестве поиска пренебрежимо малы.

В третьей части исследуется извлечение нейросетевых дескрипторов со сверточных слоев нейросети. Этот подход является предпочтительным, так как может быть применен к изображению любого размера и любой формы. Бо-

лее того, выходы сверточных слоев можно интерпретировать, как локальные дескрипторы регионов изображения, или как аналогию плотным СИФТ дескрипторам [29]. Автор исследует применение существующих передовых методов агрегации к выходам сверточных слоев и экспериментально показывает, что простое покоординатное суммирование (*суммирующий пулинг*) является оптимальным способом агрегации. На основе суммирующего пулинга предлагается СПОК дескриптор, превосходящий все существующие глобальные дескрипторы по качеству поиска, а также простой в вычислении и не имеющий гиперпараметров.

1.1 Обзор существующих методов построения дескрипторов изображений

В существующих методах построения глобальных дескрипторов изображений первым шагом является извлечение некоторого числа характерных областей этого изображения (патчей). Затем каждый из полученных патчей описывается некоторым простым способом (например, с помощью вектора гистограммы градиентов интенсивности в этом патче по направлениям [29]). Таким образом, изображению I ставится в соответствие некоторое число *локальных* дескрипторов $\{x_1, \ldots, x_n\} \subset \mathbf{R}^d$, а задача представления изображения в виде *глобального* дескриптора фиксированной длины $\psi(I)$ сведена к задаче агрегации этих локальных дескрипторов.

Как правило, процесс построения дескриптора $\psi(I)$ включает в себя два этапа: *отображение* и *агрегацию*. На первом этапе каждый вектор x отображается в вектор большей размерности $\phi(x) \in \mathbf{R}^D$. Затем все полученные образы $\{\phi(x_1), \ldots, \phi(x_n)\} \subset \mathbf{R}^D$ агрегируются, обычно простым суммированием, хотя более сложные методы тоже возможны [7].

Большинство существующих методов отличаются лишь выбором отображения ϕ. Например, алгоритм VLAD [5] разбивает пространство локальных дескрипторов на K регионов с центроидами $\{c_1, \ldots, c_K\}$ и затем отображает вектор x в $\phi_{\text{VL}}(x) = [0,\ 0\,; \ldots, (x - c_k)\ \ldots, 0] \in \mathbf{R}^{K \times d}$, где k — индекс центроида, ближайшего к x. Протокол другого подхода, основанного на Фишеров-

ских векторах [30] использует вероятностное, а не детерминированное присваивание к ближайшим центроидам. Подход триангуляционного отображения [7] также использует центроиды регионов и отображает локальный вектор x в конкатенацию нормализованных смещений между ним и всеми центроидами $\phi_{\text{TE}}(x) = \left[\frac{x-c_1}{||x-c_1||}, \ldots, \frac{x-c_K}{||x-c_K||}\right]$. На практике отображения $\phi_{\text{TE}}(x)$ также центрируются и нормализуются.

Все существующие способы построения глобальных дескрипторов используют локальные дескрипторами, построенные по фиксированному алгоритму (например, СИФТы или гистограммы цветов). Качество глобальных дескрипторов могло бы быть существенно лучше, если бы они были получены с помощью глубоких сверточных нейросетей, которые формируют локальные дескрипторы в режиме обучения с учителем.

В настоящее время такие сети являются передовым алгоритмом классификации изображений, в связи с чем привлекают большое внимание исследователей в области компьютерного зрения. В недавних работах [26] было показано, что векторы активаций нейронов, образующиеся на верхних слоях нейросети, обученной классифицировать изображения, могут быть хорошими дескрипторами для многих других задач в компьютерном зрении. В этой главе автором исследуются различные способы получения нейросетевых дескрипторов такого типа для задачи визуального поиска.

1.2 Извлечение дескрипторов с полносвязных слоев нейросети

1.2.1 Использование предобученной нейросети

Сначала исследуем самый простой способ получить нейросетевой дескриптор изображения — пропустим изображение через глубокую сверточную сеть, обученную классифицировать на 1000 классов из коллекции Image-Net [26]. Будем использовать сеть с типичной архитектурой, включающей пять сверточных слоев, каждый из которых использует свертку, поэлементную нелинейную операцию вида $f(x) = \max(x,0)$, и макс-пулинг. Затем следуют три полносвязных

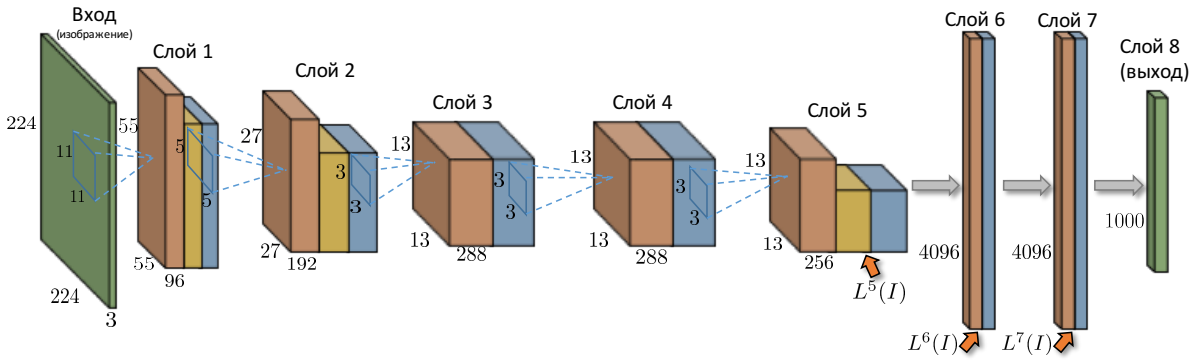


Рисунок 1.1 — Архитектура используемой глубокой сверточной нейросети. Зеленые слои соответствуют входному цветному изображению размера 224×224 и выходному вектору вероятностей каждого из 1000 классов. Красные слои соответствуют результатам свертки, желтые — результатам операции макс-пулинга, а синие — результатам нелинейного преобразования $f(x) = \max(x,0)$. Слои 6, 7 и 8 являются полносвязными по отношению к предыдущему слою. Выходы активаций, соответствующие нейросетевым дескрипторам, отмечены красными стрелками.

слоя (“слой 6”, “слой 7”, “слой 8”), каждый из которых принимает на вход выход предыдущего слоя, умножает его на матрицу и поэлементно применяет нелинейную операцию. Архитектура сети схематично изображена на рисунке 1.1.

Используемая сеть принимает на вход изображения размером 224×224, изображения других размеров сжимаются или увеличиваются. Для данного изображения I сеть формирует последовательность активаций нейронов на различных слоях. Вектора активаций со слоев 5, 6 и 7 обозначены как $L^5(I)$, $L^6(I)$, and $L^7(I)$ (до применения поэлементной нелинейной операции). Каждый из этих векторов является детерминированной функцией изображения и может служить его дескриптором.

Тестовые коллекции. Будем оценивать качество нейросетевых дескрипторов на коллекциях изображений, перечисленных ниже. Результаты работы передовых существующих дескрипторов (размерности до 32.000) приведены в таблице 1.1.

Oxford Buildings dataset [28] (Oxford).

Коллекция включает в себя 5062 фотографии, собранные с сайта Flickr, на которых изображены основные достопримечательности Оксфорда. Изобра-

жения, соответствующие 11 достопримечательностям (некоторые из которых имеют сложную структуру и включают в себя несколько зданий) размечены вручную. Для каждой достопримечательности также имеется 5 изображений-запросов, и качество дескрипторов оценивается путем измерения средней точности поиска [28] по этим запросам.

Oxford Buildings dataset+100K [28] (Oxford 105K). Та же коллекция с таким же протоколом оценки качества, но с дополнительными 100.000 изображениями, собранными из других источников.

INRIA Holidays dataset [27] (Holidays). Коллекция включает в себя 1491 фотографию, объединенных в 500 групп, которые соответствуют различным сценам или объектам. Одно изображение из каждой группы является запросом. Качество дескрипторов оценивается на основе средней точности поиска по всем запросам. Некоторые изображения находятся не в естественной ориентации (повернуты на 90 градусов). Так как стандартные глубокие нейросети обучаются на изображениях естественной ориентации, изображения были повернуты вручную. Все результаты ниже были получены на коллекции повернутых фотографий (средняя точность на оригинальной коллекции обычно ниже на 0.03). Это падение в качестве может быть отыграно путем поворота изображений на стороне запроса и на стороне коллекции.

University of Kentucky Benchmark dataset [17] (UKB). Коллекция включает в себя 10,200 фотографий 2550 различных объектов (4 фотографии на объект). Каждое изображение используется в качестве запроса. Качество оценивается как среднее число изображений того же объекта в топ-4 поисковой выдачи, и, таким образом, является числом от 0 до 4.

Результаты. Результаты для нейросетевых дескрипторов, полученных с помощью нейросети, обученной на классах из коллекции ILSVRC [31] приведены в средней части таблицы 1.1. Все результаты были получены с использованием евклидового расстояния на l_2-нормализованных нейросетевых дескрипторах. Приведены результаты для каждого из слоев 5,6,7. Оценивалось также качество слоя 8, однако качество его работы оказалось существенно ниже (например, средняя точность на 0.02 ниже по сравнению со слоем 5 на коллекции Holidays).

Рисунок 1.2 — Пример поисковой выдачи из коллекции Holidays, в котором наилучшие результаты достигаются с использованием слоя 5. Вероятное объяснение заключается в том, что по сравнению с другими слоями, активации на нем соответствуют детектированию низкоуровневых текстурных деталей, а не более абстрактных объектов. Самое левое изображение в каждой строке соответствует запросу, верные ответы обведены зеленым.

Среди всех слоев, самое лучшее качество получается с использованием дескрипторов с 6 слоя, однако для разных запросов, лучшее качество достигается с использованием различных дескрипторов (см. рисунки 1.2 и 1.3).

В целом, результаты, достигнутые с использованием нейросетевых дескрипторов, сопоставимы с результатами передовых существующих методов, но не превосходят их. Тем не менее, результаты довольно впечатляющие, так как нейросеть обучалась решать задачу классификации, а не задачу визуального поиска.

1.2.2 Дообучение нейросетевых дескрипторов

Естественной идеей повышения качества нейросетевых дескрипторов является дообучение сверточной нейросети на коллекции, содержащей изображения и метки классов, которые более релевантны коллекциям, использующимся во время поиска.

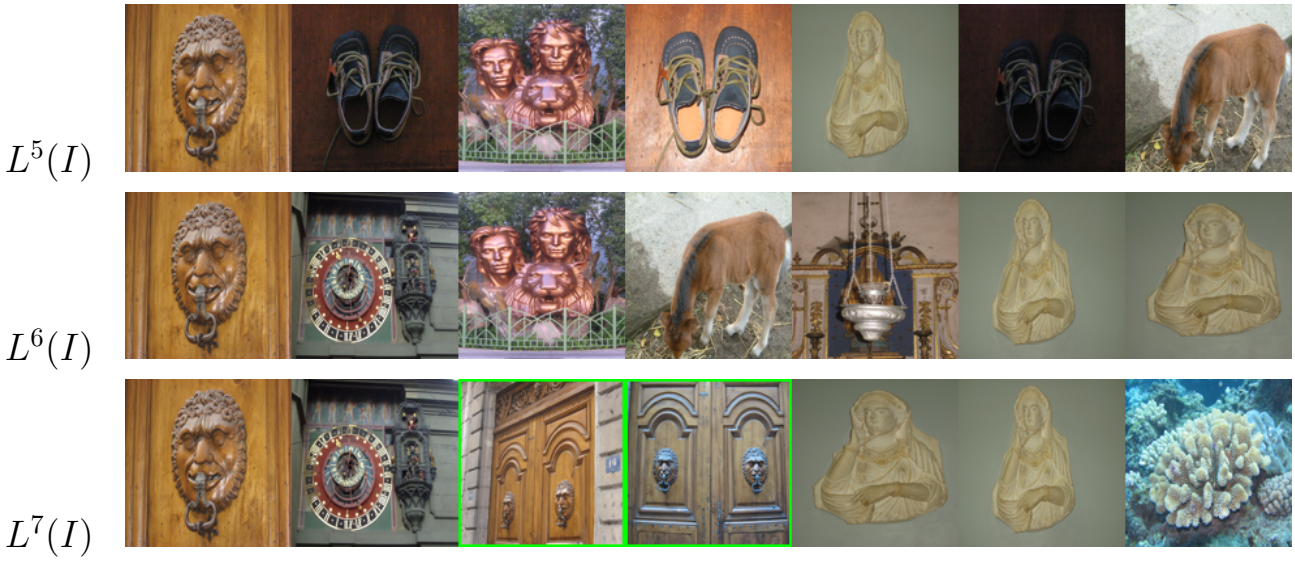


Рисунок 1.3 — Пример поисковой выдачи из коллекции Holidays, в котором наилучшие результаты достигаются с использованием слоя 7. Вероятное объяснение состоит в том, что этот слой детектирует объекты высокого уровня абстракции. Самое левое изображение в каждой строке соответствует запросу, верные ответы обведены зеленым.

Коллекция "Достопримечательности". В первую очередь была собрана коллекция, похожая на коллекции фотографий достопримечательностей (Holidays и Oxford Buildings). Подготовка такой коллекции является нетривиальной задачей, для которой использовался полуавтоматический подход. Сначала были отобраны 10000 страниц из Википедии, соответствующие достопримечательностям. Заголовок каждой страницы был использован в качестве запроса к системе поиска по изображениям Яндекс[1], а затем скачивались изображения из топ-1000 поисковой выдачи по каждому запросу (или меньше, если выдача имела меньший размер).

На втором этапе были вручную удалены "мусорные" изображения, не относящиеся к заданному запросу, а также запросы, по которым было найдено менее 100 релевантных изображений. Итогом этой фильтрации стала коллекция из 672 классов и 213,678 изображений. Во время сбора коллекции были удалены запросы относящиеся к Оксфорду, чтобы исключить эффект переобучения. Список запросов и изображений можно получить по ссылке[2].

Наш подход для сбора коллекции отличается от использованного в [32], где обходится сайт Flickr в абсолютно автоматическом режиме без человеческого

[1] http://images.yandex.ru

[2] http://sites.skoltech.ru/compvision/projects/neuralcodes/

Дескриптор	Размер	Oxford	Oxford 105K	Holidays	UKB
Fisher+color [33]	4096	—	—	**0.774**	3.19
VLAD+adapt+innorm [34]	32768	0.555	—	0.646	—
Sparse-coded features [35]	11024	—	—	0.767	**3.76**
Triangulation embedding [7]	8064	**0.676**	**0.611**	0.771	3.53
Дескрипторы с нейросети, обученной на ILSVRC					
Слой 5	9216	0.389	—	0.690*	3.09
Слой 6	4096	0.435	0.392	0.749*	3.43
Слой 7	4096	0.430	—	0.736*	3.39
Дескрипторы с нейросети, обученной на “Достопримечательностях”					
Слой 5	9216	0.387	—	0.674*	2.99
Слой 6	4096	0.545	0.512	**0.793***	3.29
Слой 7	4096	0.538	—	0.764*	3.19
Дескрипторы с нейросети, обученной на вращающихся объектах					
Слой 5	9216	0.348	—	0.682*	3.13
Слой 6	4096	0.393	0.351	0.754*	3.56
Слой 7	4096	0.362	—	0.730*	3.53

Таблица 1.1

Глобальные дескрипторы: сравнение с существующими методами. Качество нейросетевых дескрипторов сопоставимо с качеством существующих методов и сильно повышается при дообучении на коллекции с изображениями, релевантными тестовой поисковой коллекции (“Достопримечательности” для Oxford Buildings и Holidays; вращающиеся объекты для UKB). $\star$ означает, что результаты получены для повернутой версии Holidays, где все изображения имеют свою естественную ориентацию

участия. Распределения изображений в индексе поисковых систем и пользовательских фотографий могут сильно отличаться, поэтому было бы интересно оценить и этот подход к формированию коллекции для дообучения.

Затем собранная коллекция использовалась для дообучения сверточной нейросети такой же архитектуры, как изображенная на рисунке 1.1 (с тем исключением, что размер выходного слоя был изменен на 672). Все остальные детали обучения не изменялись.

Результаты для нейросетевых дескрипторов после дообучения. Результаты для нейросетевых дескрипторов, полученных с помощью сети, дообученной на коллекции “Достопримечательности”, приведены в таблице 1.1. Как и ожидалось, разница в качестве по сравнению с оригинальными нейро-

Рисунок 1.4 — Примеры изображений из классов “Замок Лидс” и “Киево-Печерская лавра” из коллекции “Достопримечательности”. Первый класс содержит в основном “чистые” иображения одного и того же здания, в то время как второй класс включает много изображений, сделанных внутри помещений, что приводит к тому, что изображения внутри класса могут не иметь ничего общего.

Рисунок 1.5 — Примеры запросов из коллекции Holidays, для которых поисковые выдачи с использованием и без использования дообучения сильно отличаются. В каждой паре строк самое левое изображение является запросом, верхняя строка соответствует выдаче без дообучения, а нижняя — выдаче с дообучением. Для большинства запросов дообучение улучшает качество выдачи. Последняя пара строк демонстрирует редкое исключение.

сетевыми дескрипторами зависит от того, насколько похожи изображения из коллекции для дообучения и конкретная тестовая коллекция для поиска. Таким образом, имеется значительное улучшение средней точности для коллекций Oxford и Oxford105K, которые целиком состоят из изображений достопримечательностей. Улучшение для коллекции Holidays меньше, но все равно существенно. Заметим, что качество дообученных дескрипторов $L^6(I)$ на коллекции Holidays выше, чем у всех ранее опубликованных систем. Репрезентативные примеры поисковой выдачи для дескрипторов до и после дообучения приведены в 1.5.

Дообучение на изображениях вращающихся объектов. После дообучения на коллекции "Достопримечательности" качество дескрипторов на коллекции UKB ухудшается. Это объясняется тем, что все изображения из UKB содержат лишь маленькие объекты под разными углами зрения и совсем не похожи на изображения достоприечательностей. Был проведен еще один эксперимент с дообучением, в котором использовалась коллекция Multi-view RGB-D [36], содержащая изображения 300 вращающихся объектов. Каждый из объектов рассматривался как отдельный класс, и для каждого класса было выбрано 200 изображений. На этой коллекции из 60000 изображений была дообучена нейросеть (такой же архитектуры, как и в остальных экспериментах). Это дообучение также привело к улучшению качества на релевантной тестовой коллекции, качество на UKB выросло 3.43 до 3.56. Средняя точность на нерелевантных коллекциях (Oxford, Oxford105K) после дообучения понизилась.

1.2.3 Сжатие нейросетевых дескрипторов

В экспериментах нейросетевые дескрипторы имеют высокую размерность (например, 4096 для $L^6(I)$). Хоть это и гораздо меньше по сравнению с существующими передовыми дескрипторами, это может быть ограничением для использования с коллекциями большого объема, поэтому необходимо исследовать вопрос их сжатия. В этой части исследуем качество нейросетевых дескрипторов при сжатии их методом главных компонент (МГК). Большинство экспериментов проводится с дескрипторами $L^6(I)$, так как их качество выше по сравнению

Размерность	16	32	64	128	256	512
Oxford						
Слой 6	0.328	0.390	0.421	0.433	0.435	0.435
Слой 6+“Достопримечательности”	0.418	0.515	0.548	0.557	0.557	0.557
Слой 6+вращающиеся объекты	0.289	0.349	0.377	0.391	0.392	0.393
Oxford 105K						
Слой 6	0.260	0.330	0.370	0.388	0.392	0.392
Слой 6+“Достопримечательности”	0.354	0.467	0.508	0.523	0.524	0.522
Слой 6+вращающиеся объекты	0.223	0.293	0.331	0.348	0.350	0.351
Holidays						
Слой 6	0.591	0.683	0.729	0.747	0.749	0.749
Слой 6+“Достопримечательности”	0.609	0.729	0.777	0.789	0.789	0.789
Слой 6+вращающиеся объекты	0.587	0.702	0.741	0.756	0.756	0.756
UKB						
Слой 6	2.630	3.130	3.381	3.416	3.423	3.426
Слой 6+“Достопримечательности”	2.410	2.980	3.256	3.297	3.298	3.300
Слой 6+вращающиеся объекты	2.829	3.302	3.526	3.552	3.556	3.557

Таблица 1.2

Качество нейросетевых дескрипторов (до и после дообучения) для различных степеней сжатия методом главных компонент. Качество дескрипторов почти не снижается вплоть до размерности 256, причем снижение качества при меньших размерностях также незначительно.

с другими на всех коллекциях. Обучение метода главных компонент осуществлялось на 100,000 случайных изображениях из коллекции “Достопримечательности”.

Качество нейросетевых дескрипторов $L^6(I)$ для различных степеней сжатия методом главных компонент представлено в таблице 1.2. В целом, сжатие методом главных компонент не приводит к существенным потерям в точности, нейросетевые дескрипторы могут быть сжаты до размерности 256 или даже 128 практически без снижения качества. Преимущество дообученных дескрипторов сохраняется на любых степеней сжатия. Таблица 1.3 приводит сравнение различных глобальных дескрипторов изображений, сжатых до размерности 128 (стандартная размерность для сравнения в литературе). Для коллекций Oxford и Holidays качество нейросетевых дескрипторов, дообученных на коллекции “Достопримечательности”, является максимальным среди глобальных дескрипторов размерности 128.

Дескриптор	Oxford	Oxford 105K	Holidays	UKB
Fisher+color [33]	—	—	0.723	3.08
VLAD+adapt+innorm [34]	0.448	0.374	0.625	—
Sparse-coded features [35]	—	—	0.727	**3.67**
Triangulation embedding [7]	0.433	0.353	0.617	3.40
Нейросетевые дескрипторы без дообучения				
Слой 6	0.433	0.386	0.747*	3.42
После дообучения на коллекции “Достопримечательности”				
Слой 6	**0.557**	**0.523**	**0.789***	3.29
После дообучения на изображениях вращающихся объектов				
Слой 6	0.391	0.348	0.756*	3.55

Таблица 1.3

Сравнение нейросетевых дескрипторов, сжатых методом главных компонент до размерности 128, с существущими дескрипторами той же размерности. Качество сжатых дообученных дескрипторов на коллекциях Holidays, Oxford и Oxford105K является максимальным среди всех существующих методов.

1.2.4 Выводы о нейросетевых дескрипторах с полносвязных слоев

Из экспериментов можно сделать вывод, что нейросетевые дескрипторы с полносвязных слоев являются хорошим инструментом для задачи поиска по изображениям, даже в случае, когда коллекции для поиска и обучения нейросети сильно разнятся. Качество нейросетевых дескрипторов можно существенно повысить путем дообучения нейросети на коллекции, содержащей изображения, семантически близкие к изображениям из тестовой поисковой коллекции. Еще одним важным результатом является тот факт, что нейросетевые дескрипторы могут быть существенно сжаты методом главных компонент практически без потери в качестве, что делает их привлекательными для использования в задачах с коллекциями большого объема.

1.3 Извлечение дескрипторов со сверточных слоев нейросети

Помимо извлечения нейросетевых дескрипторов с полносвязных слоев интересным вопросом является также извлечение таких дескрипторов со сверточных слоев нейросети. Преимуществом использования сверточных слоев является то, что их выходы могут быть вычислены для изображений любого размера и формы. Помимо этого, эти выходы можно интерпретировать как совокупность дескрипторов локальных участков изображения, соответствующих полям восприятия нейронов с конкретными пространственными координатами. Такие дескрипторы можно рассматривать как аналог плотных локальных СИФТ дескрипторов изображения [29; 37], поэтому ниже они будут называться просто глубокими локальными дескрипторами. В этой части будет описан алгоритм построения дескрипторов изображений на основе активаций нейронов в сверточных слоях нейросети.

1.3.1 Агрегация глубоких локальных дескрипторов

В этой части сперва проводится сравнение свойств распределений глубоких локальных дескрипторов и СИФТ дескрипторов, а также подчеркиваются различия между ними. Далее на основе этих различий будет предложен новый глобальный дескриптор изображений, который не включает себя этап отображения, необходимый для в методах, основанных на СИФТ дескрипторах. Также в этой части обсуждаются некоторые параметры построения нового глобального дескриптора.

В экспериментах глубокие локальные дескрипторы извлекаются путем применения предобученной нейросети к изображению, и полученная совокупность активаций нейронов на последнем сверточном слое рассматривается как трехмерный массив таких дескрипторов. Обозначим количество каналов в рассматриваемом слое за C, а пространственные размеры слоя за H и W. Тогда входное изображение I представляется множеством из $H \times W$ C-мерных векто-

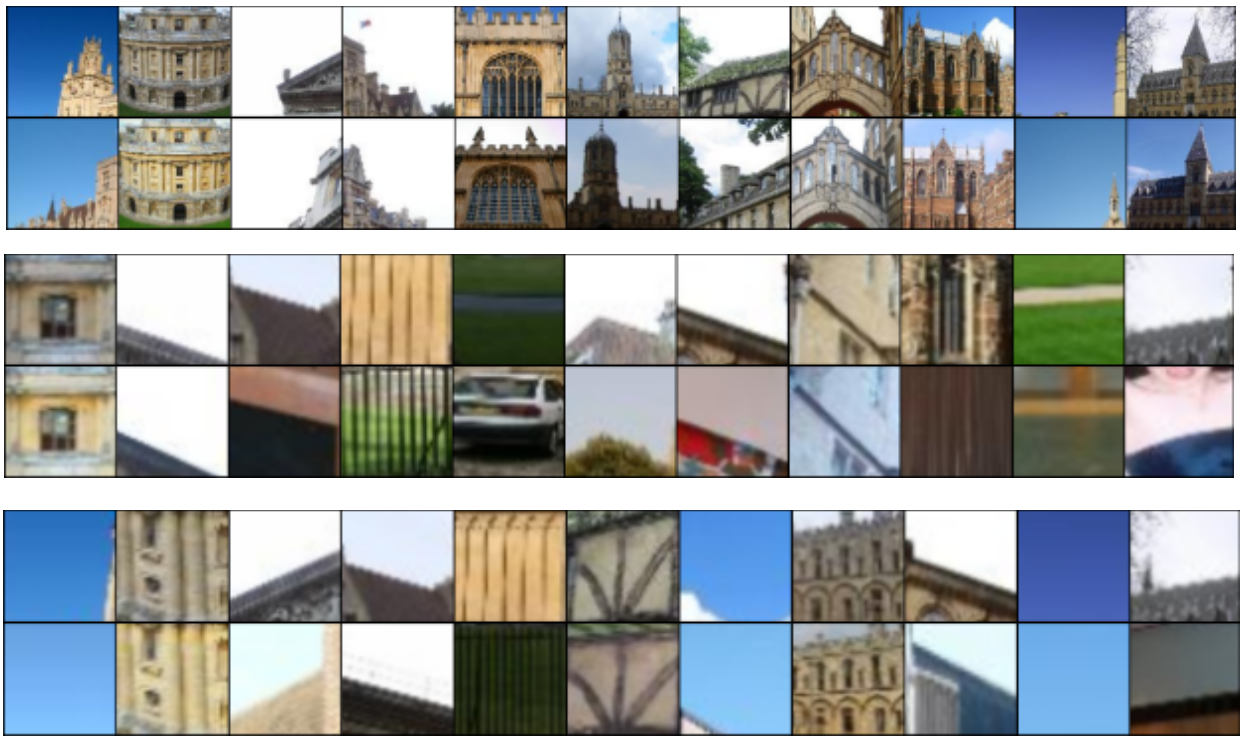


Рисунок 1.6 — Случайные примеры патчей, сопоставленных с помощью глубоких локальных дескрипторов (верхняя строка), с помощью СИФТ дескрипторов (средняя строка), с помощью СИФТ дескрипторов после отображения $\phi_{\mathrm{VL}}()$ (нижняя строка). Как правило, сопоставления патчей, полученные с помощью глубоких локальных дескрипторов, содержат гораздо меньше ошибочных сопоставлений.

ров, которые и являются рассматриваемыми глубокими локальными дескрипторами.

Свойства локальных глубоких дескрипторов

Как было показано в [7], СИФТ дескрипторы недостаточно хорошо отражают семантику описываемых патчей, то есть несвязанные семантически патчи могут описываться близкими в смысле евклидовой метрики СИФТ дескрипторами. Глубокие локальные дескрипторы должны отражать семантику лучше, так как они формируются в режиме обучения с учителем, обучаясь на больших размеченных коллекциях изображений. Для подтверждения этой гипотезы проведем два качественных эксперимента.

Эксперимент 1 сравнивает патчи, сопоставленные тремя типами дескрипторов (см. рисунок 1.6). Эти сопоставления формировались следующим образом:

- Для каждого изображения в коллекции Oxford Buildings извлекались глубокие локальные дескрипторы и СИФТ дескрипторы.
- Каждый из СИФТ дескрипторов отображался по закону $\phi_{\text{VL}}()$ с $K = 64$.
- Для каждого из трех типов локальных дескрипторов (глубокие, СИФТ, СИФТ с отображением) и для каждого изображения-запроса вычислялась косинусная мера близости между его дескрипторами и дескрипторами всех остальных изображений в коллекции.
- Рассматривались случайные пары локальных дескрипторов из десяти пар с максимальными мерами близости для каждого изображения. Соответствующие патчи изображений визуализировались.

На рисунке 1.6 изображено случайное подмножество пар патчей, выбранных процедурой, описанной выше. Верхняя строка соответствует сопоставлениям, полученным на основе глубоких локальных дескрипторов, средняя строка — полученным на основе СИФТ дескрипторов, нижняя строка — полученным на основе СИФТ дескрипторов с отображением. Как и ожидалось, сопоставления, полученные на основе глубоких локальных дескрипторов, содержат гораздо меньше ошибочно сопоставленных пар, так как часто соответствуют одному и тому же объекту и являются достаточно инвариантными к изменению освещенности и угла зрения. Сопоставления на СИФТ дескрипторах гораздо хуже и многие пары патчей содержат совершенно несвязанные семантически объекты. Использование отображения СИФТ дескрипторов повышает качество сопоставлений, но тем не менее их качество гораздо ниже по сравнению с сопоставлениями на глубоких локальных дескрипторах.

Эксперимент 2. Также исследуем различные статистики распределений глубоких локальных дескрипторов и СИФТ дескрипторов. Более всего интересно распределение глубоких локальных дескрипторов с большой нормой, так как они вносят наибольший вклад в глобальный дескриптор для любых разумных способов агрегации.

Для различных типов локальных дескрипторов необходимо исследовать надежность сопоставлений, полученных на основе их близости. Для этого срав-

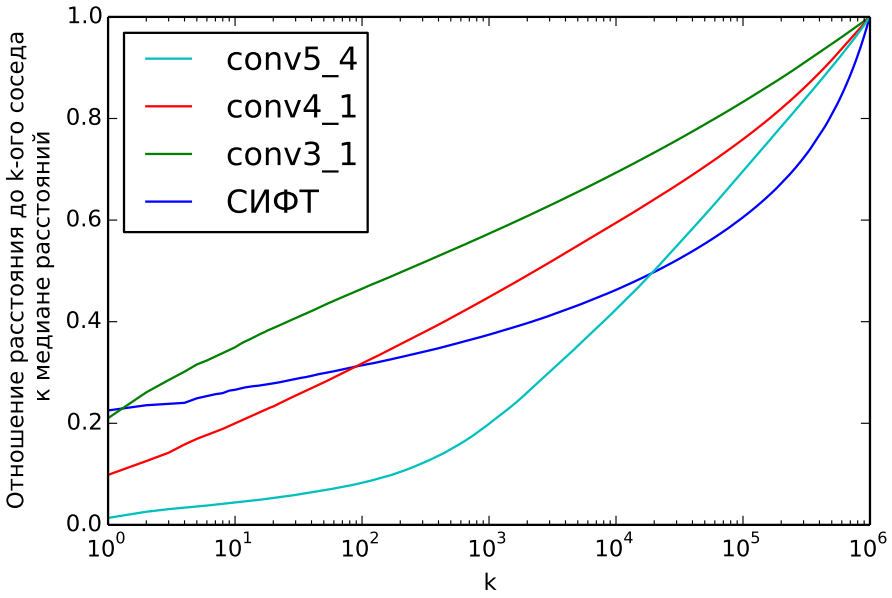


Рисунок 1.7 — Среднее отношение расстояние до k-ого соседа к медиане расстояния до всех дескрипторов для СИФТ дескрипторов и глубоких локальных дескрипторов с максимальной нормой (для трех различных сверточных слоев). Дескрипторы с последнего сверточного слоя имеют небольшое число очень близких соседей, несмотря на большую размерность. Этот факт демонстрирует существенные различия в пространственном распределении двух типов локальных дескрипторов в соответствующих пространствах большой размерности.

ниваются евклидовы расстояния от каждого локального дескриптора до его ближайших соседей с расстояниями до случайных дескрипторов из коллекции. Более детально протокол эксперимента описывается ниже. Для каждого изображения-запроса извлекаются десять глубоких локальных дескрипторов с максимальной нормой, и для каждого из них вычисляются расстояния до всех глубоких локальных дескрипторов всех изображений из коллекции. Затем строится график, демонстрирующий, как расстояние до k-ого соседа зависит от k. Для каждого дескриптора-запроса расстояния нормализуются путем деления на медиану всех расстояний между данным дескриптором и всеми остальными дескрипторами из коллекции.

Эта процедура была проведена для трех типов глубоких локальных дескрипторов, извлеченных со сверточных слоев различной глубины: “conv3_1”, “conv4_1” и “conv5_4” сети OxfordNet [38]. Также эта процедуры была проведена для СИФТ дескрипторов, хотя в этом случае использовались случайные признаки из каждого изображения, так как СИФТ дескрипторы имеют единичную норму. Для всех типов локальных дескрипторов в этом эксперименте использовалось подмножество из двух миллионов векторов.

Усредненные по всем запросам кривые представлены на рисунке 1.7. Кривые демонстрируют, что глубокие локальные дескрипторы с большой нормой со слоя “conv5_4” имеют небольшое количество “очень близких” соседей, которые гораздо ближе по сравнению с остальными точками. Кривая для СИФТ дескрипторов, наоборот, демонстрирует, что типичные расстояния до ближайших соседей сопоставимы по величине с расстояниями до случайных векторов в пространстве дескрипторов. Этот факт говорит о том, что близость СИФТ дескрипторов гораздо менее информативна и высокая мера сходства не является гарантией семантической близости соответствующих патчей. Можно также заметить, что сходство дескрипторов со слоев “conv3_1” и “conv4_1” менее информативно и надежно по сравнению с “conv5_4” (то есть более глубокие слои образуют дескрипторы с более надежной мерой сходства).

Второй эксперимент не использует разметку коллекции Oxford, то есть в нем не учитывается корректность сопоставлений при вычислении расстояний. Тем не менее, этот эксперимент подчеркивает важные различия в распределении глубоких локальных дескрипторов и СИФТ дескрипторов в соответствующих пространствах.

Результаты обоих экспериментов подтверждают гипотезу о том, что сходство глубоких локальных дескрипторов с последнего сверточного слоя является инфомативным и количество некорректных сопоставлений, полученных на основе глубоких локальных дескрипторов гораздо меньше по сравнению с СИФТ дескрипторы. Во-первых, сопоставление является более точным (что подтверждает первый эксперимент), во-вторых, потому что глубокие локальные дескрипторы с большой нормой имеют меньшее количество близких соседей (что подтверждает эксперимент 2). Эти результаты говорят о том, что можно опустить этап отображения при построении глобального дескриптора, когда локальные дескрипторы являются глубокими.

Алгоритм построения дескриптора СПОК

Теперь опишем СПОК дескриптор, который основан на агрегации глубоких локальных дескрипторов без какого-либо отображения. Каждый глубокий локальный дескриптор f, вычисленный с изображения I, ассоциирован с пространственными координатами (x,y), соответствующими его пространственному положению в выходе последнего сверточного слоя.

Суммирующий пулинг. Построение СПОК дескриптора начинается с суммирования всех глубоких локальных дескрипторов:

$$\psi_1(I) = \sum_{y=1}^{H} \sum_{x=1}^{W} f_{(x,y)} \tag{1.1}$$

Центрирование. Для многих поисковых коллекций объект интереса, как правило, находится в окрестности геометрического центра изображения. СПОК дескриптор можно модифицировать, чтобы учесть эту особенность, путем простого взвешенного суммирования. Это суммирование присваивает большие веса глубоким признакам центрального участка изображения, формула (1.1) видоизменяется до:

$$\psi_2(I) = \sum_{y=1}^{H} \sum_{x=1}^{W} \alpha_{(x,y)} f_{(x,y)} \tag{1.2}$$

Коэффициенты $\alpha_{(w,h)}$ зависят только от пространственных координат h и w. В частности, была использована схема взвешивания с весами:

$$\alpha_{(x,y)} = \exp\left\{ -\frac{\left(y - \frac{H}{2}\right)^2 + \left(x - \frac{W}{2}\right)^2}{2\sigma^2} \right\}, \tag{1.3}$$

где σ равняется одной третьей расстояния между центром изображения и ближайшей границей. Будучи очень простой в реализации, операция центрирования позволяет значительно улушить качество дескрипторов для некоторых коллекций, как будет показано в экспериментах.

Постобработка. Полученный вектор $\psi(I)$ l_2-нормализуется, затем применяются операции сжатия методом главных компонент и обесцвечивания:

$$\psi_3(I) = \operatorname{diag}\left(s_1, s_2, \ldots, s_N\right)^{-1} M_{\text{PCA}}\, \psi_2(I) \tag{1.4}$$

где M_{PCA} — прямоугольная матрица главных компонент размером $N \times C$, N — размерность, до которой происходит сжатие, а s_i — ассоциированные сингулярные числа.

Наконец, после операции обесцвечивания вектор l_2-нормализуется:

$$\psi_{SPOC}(I) = \frac{\psi_3(I)}{\|\psi_3(I)\|_2} \tag{1.5}$$

Необходимо заметить, что несжатый вектор $\psi_2(I)$ имеет размерность C, что равняется количеству каналов в выходе последнего сверточного слоя. Типичные значения для C составляют несколько сотен, то есть $\psi(I)$ обладает относительно небольшой размерностью. Поэтому для обучении матрицы главных компонент для сжатия СПОК дескриптора потребуется гораздо меньше обучающих данных по сравнению с методами Фишеровских векторов и триангуляционного отображения, так как соответствующие им дескрипторы имеют гораздо большую размерность, а следовательно, подвержены большему риску

Метод	Holidays	Oxford5K (полные)	Oxford105K (полные)	UKB
Фишеровский вектор, k=16	0.704	0.490	—	—
Фишеровский вектор, k=256	0.672	0.466	—	—
Трианг. отображение, k=1	0.775	0.539	—	—
Трианг. отображение, k=16	0.732	0.486	—	—
Макс-пулинг	0.711	0.524	0.522	3.57
Суммирующий пулинг	0.802	0.589	0.578	3.65
СПОК (с центрированием)	0.784	0.657	0.642	3.66

Таблица 1.4

Сравнение различных методов агрегации локальных глубоких дескрипторов. Глобальные дескрипторы, полученные каждым из методов, были сжаты методом главных компонент до размерности 256, затем применялись операции обесцвечивания и l_2-нормализации. Суммирующий пулинг (СПОК) превосходит по качеству все существующие методами на всех коллекциях. На коллекции Oxford использовались полные (необрезанные) изображения-запросы.

переобучения. Ниже будет показано, что это переобучение является серьезной проблемой на практике.

1.3.2 Эксперименты

Детали экспериментов. Глубокие локальные дескрипторы извлекались с помощью сети из работы [38] с использованием пакета `Caffe` [39]. Для сети такой архитектуры число каналов в выходе последнего сверточного слоя равно $C = 512$. Все изображения приводились к размеру 586×586 перед тем как быть поданными на вход нейросети, что приводило к тому, что пространственные размеры выхода после рассматриваемого слоя были равны $W \times H = 37 \times 37$. Итоговая размерность СПОК дескрипторов и дескрипторов, полученных другими методами, была равна $N = 256$.

Методы агрегации. Основной целью экспериментов являлось сравнение различных схем агрегации для глубоких локальных дескрипторов.

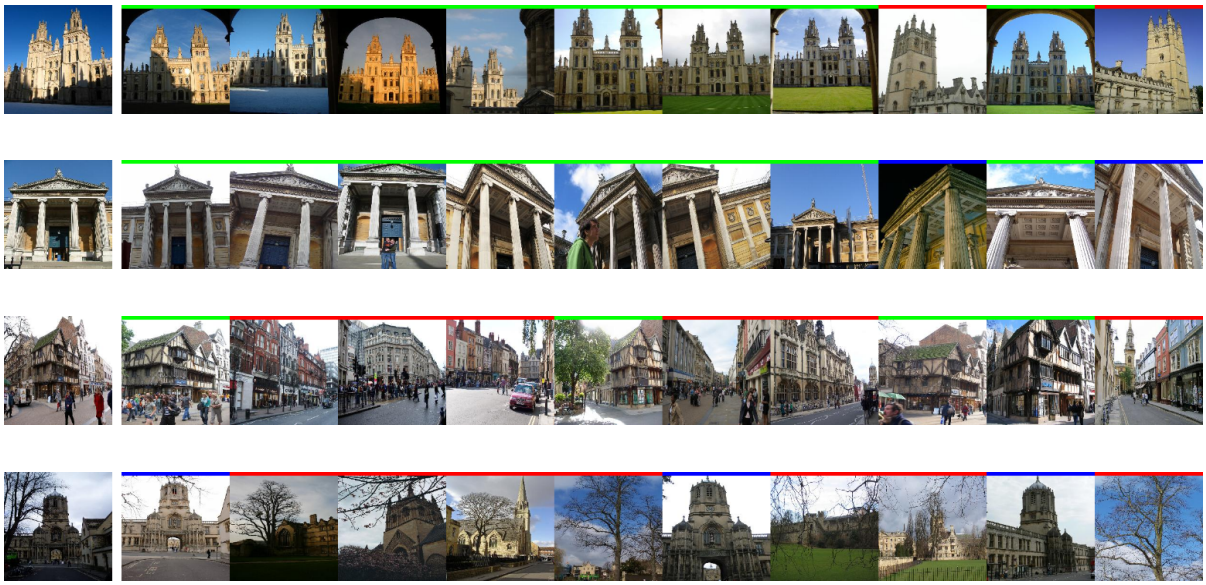


Рисунок 1.8 — Примеры поисковой выдачи с использованием СПОК дескрипторов на коллекции Oxford Buildings. В каждом примере продемонстрирован запрос и топ-10 найденных результатов. Красным цветом обозначены ошибочно найденные изображения, зеленым цветом обозначены изображения, найденные верно, синим цветом изображены “мусорные изображения”. Два верхних примера демонстрируют, что СПОК дескриптор устойчив к небольшим сдвигам, изменению угла зрения, изменению масштаба. Два нижних примера соответствуют запросам, в которых СПОК дескриптор работает плохо. В этих случаях, вместо того, чтобы искать здания, дескриптор возвращает изображения с нерелевантными объектами, такими как дерево или дорога.

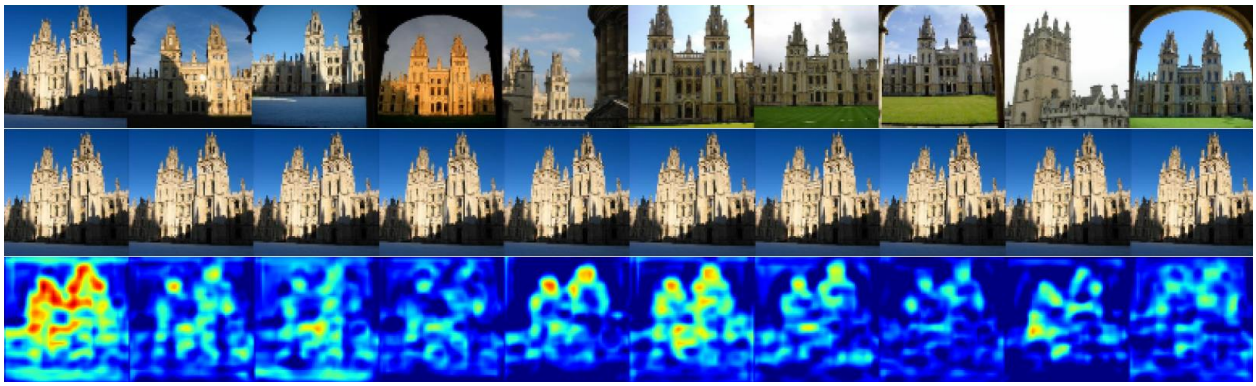


Рисунок 1.9 — Примеры карт сходства между локальными глубокими дескрипторами изображения-запроса и СПОК дескрипторов изображений из топ-10 поисковой выдачи. Глубокие локальные дескрипторы сжаты методом главных компонент с той же матрицей, какая использовалась при построении СПОК дескрипторов, и вычислялась косинусная мера близости между каждым локальным дескрипторами запроса и СПОК дескриптором изображения из поисковой выдачи. Карты сходства позволяют определить, какие участки изображения “ответственны” за то, что это конкретное изображение попало в топ поисковой выдачи. Например, для запроса выше пики двух башен “ответственны” за большинство найденных результатов.

Рассматривались простые варианты агрегации путем суммирующего пулинга и макс-пулинга. Также было проведено сравнение с двумя продвинутыми методами агрегации, а именно Фишеровский вектор [30] (использовалась реализация из библиотеки Yael [40]) и Триангуляционное отображение [7] (использовалась авторская реализация). Параметры всех методов были аккуратно настроены для наилучших результатов для работы с глубокими локальными дескрипторами.

Например, для Фишеровских векторов оказалось выигрышным сжимать глубокие локальные дескрипторы методом главных компонент до размерности 32. В свою очередь, триангуляционное отображение в случае глубоких локальных дескрипторов выигрывает от отсутствия поэлементного извлечения квадратного корня (что необходимо для СИФТ дескрипторов).

Глобальные дескрипторы, полученные всеми методами, сжимались с помощью метода главных компонент до размерности 256. Последним шагом для всех дескрипторов была l_2-нормализация. Во время поиска использовалась косинусная мера близости (что в случае нормализованных дескрипторов эквива-

Метод	Holidays	Oxford5K
Фишеровский вектор, k=16	0.704	0.490
Фишеровский вектор, МГК на тесте, k=16	0.747	0.540
Фишеровский вектор, k=256	0.672	0.466
Фишеровский вектор, МГК на тесте, k=256	0.761	0.581
Триангуляционное отображение, k=1	0.775	0.539
Триангуляционное отображение, МГК на тесте, k=1	0.789	0.551
Триангуляционное отображение, k=16	0.732	0.486
Триангуляционное отображение, МГК на тесте, k=16	0.785	0.576
Макс-пулинг	0.711	0.524
Макс-пулинг, МГК на тесте	0.728	0.531
СПОК без центрирования	0.802	0.589
СПОК без центрирования, МГК на тесте	0.818	0.593
СПОК с центрированием	0.784	0.657
СПОК с центрированием, МГК на тесте	0.797	0.651

Таблица 1.5

Сравнение эффекта переобучения, возникающего в обучении матрицы главных компонент (МГК) для СПОК дескриптора и других методов. Размерности всех дескрипторов уменьшались до 256 с помощью метода главных компонент. Эффект переобучения гораздо меньше в случае СПОК дескриптора и макс-пулинга по сравнению с передовыми существующими методами агрегации.

лентно евклидову расстоянию). Параметры метода главных компонент были обучены на отдельных коллекциях, не пересекающихся с тестовыми коллекциями для поиска.

Результаты. Сравнение различных методов агрегации приведено в таблицах 1.4 и 1.5. Необходимо отметить ряд важных наблюдений:

– Для глубоких локальных дескрипторов суммирующий пулинг является наилучшим способом агрегации. Он превосходит не только простой макс-пулинг, но и гораздо более сложно устроенные методы Фишеровского вектора и триангуляционного отображения (что неверно в случае СИФТ дескрипторов, с которыми суммирующий пулинг работает гораздо хуже).

– Подверженность переобучению для различных методов продемонстрирована в таблице 1.5. Можно заметить, что методы Фишеровского вектора и триангуляционного отображения сильно страдают от переобу-

Метод	D	Holidays	Oxford5K (полные)	Oxford5K (кропы)	Oxford105K (полные)	Oxford105K (кропы)	UKB
SIFT+Triang.+Dem. aggr. [7]	1024	0.720	–	0.560	–	0.502	3.51
SIFT+Triang.+Dem. aggr. [7]	128	0.617	–	0.433	–	0.353	3.40
Полносвязные [20]	256	0.749	0.435	–	0.386	–	3.42
Полносвязные+дообучение [20]	256	0.789	0.557	–	0.524	–	3.56
Сверточные+Макс-пулинг [41]	256	0.716	0.533	–	0.489	–	–
Полносвязные + VLAD [42]	512	0.783	–	–	–	–	–
СПОК без центрирования	256	0.802	0.589	0.531	0.578	0.501	3.65
СПОК с центрированием	256	0.784	0.657	0.592	0.642	0.561	3.66

Таблица 1.6

Сравнение с передовыми существующими методами построения компактных глобальных дескрипторов. Несмотря на простой алгоритм вычисления, качество работы СПОК дескрипторов существенно выше по сравнению с другими методами на всех коллекциях.

чения матрицы главных компонент. Если матрица главных компонент обучается на тестовой коллекции, качество этих методов существенно увеличивается. Из-за переобучения является выигрышным использовать более простые способы агрегации: 16 компонент в Фишеровском векторе (а не 256), 1 центроид в триангуляционном отображении (а не 16). Для простых методов суммирующего пулинга и макс-пулинга переобучение несущественно.

– В случае триангуляционного отображения вырожденная конфигурация с одним центроидом работает лучше всего. Даже без сжатия методом главных компонент качество практически не растет при увеличении количества центроидов, что подтверждает гипотезу об особенностях распределения глубоких локальных дескрипторов.
– Центрирование существенно повышает результаты на коллекциях Oxford и Oxford105K, несущественно повышает результаты на коллекции UKB, и ухудшает результаты на коллекции Holidays.
– Обесцвечивание существенно повышает качество работы суммирующего пулинга и несущественно снижает результаты макс-пулинга (например, средняя точность макс-пулинга с обесцвечиванием на коллекции Oxford равна 0.48, в то время как без обесцвечивания средняя точность равна 0.52). Дело в том, что обесцвечивание подавляет “очень популярные” локальные дескрипторы, часто встречающиеся во многих изображениях и внутри конкретного изображения.

– Сжатие методом главных компонент иногда повышает качество работы глубоких дескрипторов, как было отмечено в работе [20]. Качество несжатых обесцвеченных СПОК дескрипторов составляет 0.55 на коллекции Oxford (0.59 со сжатием) и 0.796 на коллекции Holidays (0.802 со сжатием).

Несколько качественных примеров хороших и плохих результатов поиска с использованием СПОК дескрипторов показано на рисунке 1.8. Также продемонстрировано несколько примеров карт сходства между локальными дескрипторами изображения-запроса и глобальными СПОК дескрипторами изображений из поисковой коллекции. Для получения этих карт к локальным признакам были применены те же операции сжатия с помощью метода главных компонент и обесцвечивания, что использовались для получения СПОК дескриптора. Затем вычислялись косинусные меры близости между СПОК дескриптором и всеми локальными дескрипторами, которые затем визуализировались путем тепловых карт. Тепловые карты позволяют локализовать участки изображения-запроса, похожие на конкретное изображения из поисковой выдачи.

Сравнение с передовыми существующими методами для компактных глобальных дескрипторов приведено в таблице 1.6. Существующие работы используют различные протоколы оценки качества для коллекции Oxford, например, работы [7;43] используют кропы изображений-запросов для поиска, а в работах [20;41;44;45] запросами являются необрезанные полные изображения. Качество СПОК дескриптора оценивалось по обоим протоколам. В случае кропов агрегировались только локальные дескрипторы, центры полей восприятия которых попадают в область кропа изображения-запроса. Так как в результате кроппинга часть информации о контексте теряется, обычно результаты для запросов на кропах хуже по сравнению с результатами для запросов на полных изображениях.

Оказывается, что разница в качестве для коллекций Oxford5K и Oxford105K довольно незначительна для всех протоколов и методов. Видимо, 100 тысяч дополнительных изображений существенно ухудшают работу методов, основанных на СИФТ дескрипторах, но не на глубоких локальных дескрипторов, так как распределения последних из коллекции Oxford и дополнительных изображений существенно отличаются.

СПОК дескрипторы обеспечивают существенное улучшение по сравнению с существующими передовыми методами, в том числе и основанными на нейросетевых дескрипторах [20;41;42]. Можно предложить несколько способов повысить качество СПОК дескрипторов. Например, можно совместно агрегировать локальные дескрипторы, извлеченные с различных масштабов изображения, или дообучить сеть на более релевантной коллекции (см. главу 1.2.2).

Выводы о нейросетевых дескрипторах со сверточных слоев

Таким образом, было исследовано несколько подходов к агрегированию глубоких локальных дескрипторов. Оказалось, что лучше всего работает простой суммирующий пулинг и на его основе был разработан СПОК дескриптор, превосходящий по средней точности поиска все существующие компактные глобальные дескрипторы изображений.

1.3.3 Заключение

В этой главе были исследованы различные варианты построения нейросетевых дескрипторов для задачи поиска семантически близких изображений. Основные выводы и экспериментальные наблюдения приведены ниже.

Самым главным выводом является то, что использование нейросетевых дескрипторов позволяет достичь существенно более высокой точности поиска по сравнению с предшествующими методами. Даже при использовании нейросети, предобученной для задачи классификации с классами, сильно отличающимися от изображений тестовых коллекций, качество нейросетевых дескрипторов является приемлемым. Экспериментально продемонстрировано, как можно существенно повысить качество дескрипторов путем адаптации нейросети под конкретную поисковую задачу или путем использования выхода сверточных слоев, позволяющих обрабатывать изображения любого размера. Практически важной особенностью нейросетевых дескрипторов является то, что их сжатие с

помощью метода главных компонент не приводит к снижению качества поиска. На нескольких тестовых коллекциях показано, что предложенные нейросетевые дескрипторы превосходят все предшествующие глобальные дескрипторы по качеству поиска.

Глава 2. Компактное кодирование дескрипторов

В связи с тем, что размеры коллекций, по которым необходимо искать современным поисковым системам, все время увеличиваются, все больший интерес вызывает задача компрессии высокоразмерных векторов. Большую коллекцию векторов (соответствующих дескрипторам изображений) необходимо компактно закодировать с большой степенью сжатия, так чтобы каждый вектор был представлен всего лишь небольшим количеством байт. Также необходимы эффективные операции вычисления скалярных произведений и евклидовых расстояний между несжатым запросом и закодированными векторами из поисковой коллекции. В этой главе исследуются различные варианты компрессии (сжатия) векторов, удовлетворяющие условиям выше.

Глава состоит из трех частей. В первой части описаны существующие методы сжатия векторов высокой размерности, охарактеризованы их преимущества и недостатки. Основным недостатком передовых существующих методов является неявное предположение об отсутствии статистической зависимости между распределениями подвекторов из различных подпространств, которое зачастую не выполняется для реальных данных.

Во второй главе автором предлагается новый метод компрессии: Аддитивная квантизация (АК). АК кодирует каждый вектор суммой небольшого числа векторов-слов и, в отличие от существующих методов, не предполагает статистической независимости распределений в различных подпространствах. Показано, что АК допускает эффективное вычисление скалярных произведений и евклидовых расстояний, и при этом существенно точнее кодирует векторы по сравнению с предшествующими методами. Единственным ограничением Аддитивной квантизации является высокая вычислительная сложность кодирования при больших длинах кода, что ограничивает применение АК на практике.

В третьей главе описан метод Древесной квантизации (ДК), который лишен недостатка Аддитивной квантизации и допускает эффективное кодирование при любых длинах кода. Экспериментально показано, что с помощью Древесной квантизации можно добиться существенно меньших потерь при сжатии, сохранив эффективность операций кодирования и вычисления расстояний и скалярных произведений.

2.1 Обзор методов сжатия дескрипторов изображений

В последние годы задача компрессии высокоразмерных векторов становится все более популярной среди исследователей в области компьютерного зрения. Более конкретно задачу можно сформулировать так: необходимо закодировать конечное множество векторов высокой размерности с большой степенью сжатия (бюджет памяти на каждый сжатый вектор не должен превышать определенного количества байт). В применении к задачам визуального поиска такие сжатые представления также должны позволять эффективно вычислять евклидовы расстояния и скалярные произведения между несжатыми векторами запросов и множеством сжатых векторов.

Существующие подходы для решения этой задачи можно разделить на две группы. Первая группа [9; 10] состоит из большого числа методов бинарного кодирования, которые представляют каждый вектор короткой последовательностью битов таким образом, что расстояние Хэмминга между двумя последовательностями аппроксимирует евклидово расстояние между исходными векторами. Преимущество этих методов в том, что они позволяют эффективную программную реализацию с использованием команд процессора.

Методы второй группы основаны на идее мультиквантизации (МК) [12]. Эти методы разбивают каждый вектор на несколько подвекторов, соответствующих ортогональным подпространствам, и затем применяют векторную квантизацию к каждому из подвекторов. Для квантизации каждого подвектора используется свой словарь небольшого размера. Для некоторые типов данных (например, СИФТ дескрипторов, представляющих собой гистограмму) наивное разбиение на подпространства приводит к высокому качеству, так как корреляции между размерностями из различных подпространств малы. Если же это не так, то необходимо использовать оптимизированную мультиквантизацию (ОМК) [46], которая применяет к исходным векторам ортогональное преобразование, которое обеспечивает низкие корреляции между размерностями различных подпространств преобразованных векторов. Это преобразование находится в результате решения оптимизационной задачи и является уникальным для конкретных данных.

Благодаря использованию таблиц поиска, сжатие мультиквантизацией позволяет эффективно вычислять расстояния и скалярные произведения между несжатыми запросами и большим количеством сжатых векторов с помощью процедуры асимметричного вычисления расстояния [12]. Помимо скорости работы мультиквантизация обеспечивает гораздо лучшее качество сжатия по сравнению с методами бинарного кодирования (для одного и того же бюджета по памяти) [12], так как не сжимает запрос, тем самым сохраняя всю информацию о нем. Таким образом, мультиквантизационные методы обеспечивают и малые потери информации при сжатии, и скорость вычисления расстояния с большим множеством сжатых векторов, что делает эти методы передовыми в задачах поиска. В то же время, мультиквантизационное разбиение на подпространства неявно предполагает, что распределения подвекторов из различных подпространств взаимно независимы. В реальных данных это предположение часто не выполняется, поэтому в некоторых случаях качество мультиквантизационных методов может быть невысоким.

2.2 Аддитивная квантизация

В этой части описывается новый метод кодирования — Аддитивная квантизация (АК), обощающий идею мультиквантизации, обеспечивающий меньшие потери кодирования и более высокое качество поиска.

2.2.1 Модель аддитивной квантизации

Опишем модель кодирования Аддитивной квантизации (АК) формально. Всюду ниже предполагается, что кодируемые векторы имеют размерность D.

Единственными параметрами Аддитивной квантизации являются M множеств $C^1, \dots, C^M \subset \mathbf{R}^D$, которые называются *словарями*. Каждый словарь содержит в точности K векторов (слов). Далее m-ый словарь обозначается как C^m, и k-ое слово в m-ом словаре как $c^m(k)$. Важно отметить, что, в отличие

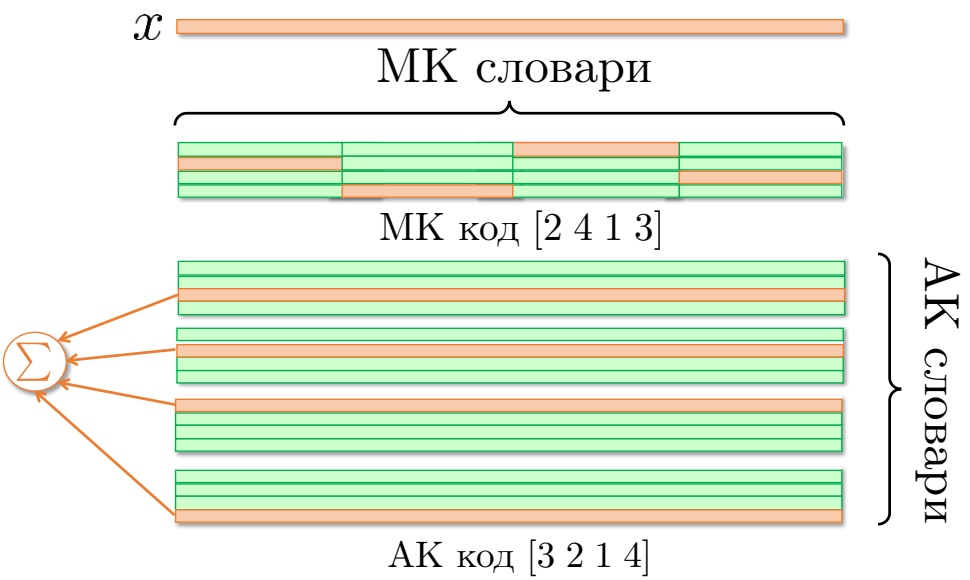


Рисунок 2.1 — Схема работы мультиквантизации (МК) и Аддитивной квантизации (АК) в случае M=4 словарей, каждый из которых имеет размер K=4. Оба метода кодирования кодируют входной вектор M числами от 1 до K. В случае МК такой код соответствует конкатенации M слов длины D/M. В случае АК, этот код соответствует сумме M слов длины D. При наличии обученных словарей АК достигает меньшей ошибки кодирования входного вектора за счет большего количества настраиваемых параметров.

от мультиквантизации, размерность векторов равна размерности кодируемых векторов.

Модель Аддитивной квантизации кодирует вектор $x \in \mathbf{R}^D$ как сумму M слов (по одному слову из каждого словаря). Более детально, вектор кодируется кортежем длины M, состоящим из номеров слов $[i_1,i_2,\ldots,i_M]$, где каждый номер находится в границах от 1 до K. Процесс кодирования осуществляется нахождением кода, минимизирующего расстояние между x и суммой соответсвующих слов.

$$x \approx \sum_{m=1}^{M} c^m(i_m), \quad i_m \in 1..K \tag{2.1}$$

Если $K = 256$ (как в большинстве работ), то вектора кодируются M байтами, и каждый байт кодирует номер одного слова. Таким образом, количество памяти, необходимое Аддитивной квантизации равно количеству памяти для мультиквантизации при тех же M и K, но Аддитивная квантизация может кодировать векторы с меньшей потерей информации, как будет продемонстрировано ниже. Словари в Аддитивной квантизации требуют в M раз больше памяти по сравнению со словарями мультиквантизации, однако их размер все равно гораздо меньше закодированных множеств векторов.

Эффективное вычисление расстояний и скалярных произведений

Сжатие на основе мультиквантизации позволяет эффективно вычислять расстояния между несжатым вектором (например, запросом при поиске ближайших соседей) и большим количеством $L \gg K$ сжатых мультиквантизацией векторов. Вычисление расстояний требует M операций обращения к таблицам поиска, $M-1$ операций сложения и небольшое количество дополнительных операций, независящее от L. Эта процедура асимметричного вычисления расстояния является основным залогом эффективности мультиквантизации. Оказывается, что такая же эффективная процедура возможна и в модели Аддитивной квантизации.

Предположим, что необходимо вычислить квадрат евклидова расстояния между запросом q и множеством из L векторов, закодированных Аддитивной квантизацией. Начнем с формулы:

$$\|q - x\|^2 = \|q\|^2 - 2\langle q,x\rangle + \|x\|^2 \tag{2.2}$$

Есть возможность преподсчитать и переиспользовать значение $\|q\|^2$ для каждого из L векторов в множестве, поэтому опишем только быстрое вычисление членов $\langle q,x\rangle$ и $\|x\|^2$.

Предположим, что x в (2.2) сжат с помощью Аддитивной квантизации, то есть $x = \sum_{m=1}^{M} c^m(i_m)$.

Вычисление скалярного произведения. Вычисление члена $\langle q,x\rangle$ можно реализовать напрямую через использование таблиц поиска. Действительно,

$$\langle q,x\rangle = \sum_{m=1}^{M} \langle q,c^m(i_m)\rangle = \sum_{m=1}^{M} T^m(i_m)\ , \tag{2.3}$$

где $T^m(i_m) = \langle q,c^m(i_m)\rangle$ можно преподсчитать и хранить для конкретного запроса q. Для конкретного запроса и L векторов, сжатых Аддитивной квантизацией, итоговая сложность включает в себя член $O(DMK)$ за вычисление таблиц поиска и член $O(ML)$ за вычисление скалярных произведений с каждым из L сжатых векторов. Обычно в прикладных задачах $L \gg DK$, поэтому сложность вычисления скалярных произведений в модели Аддитивной квантизации практически совпадает с таковой в модели мультиквантизации.

Вычисление $\|x\|^2$. Вычисление $\|x\|^2$ также можно осуществлять на основе таблиц поиска. Действительно,

$$\|x\|^2 = \|\sum_{m=1}^{M} c^m(i_m)\|^2 = \sum_{m=1}^{M}\sum_{m'=1}^{M} \langle c^m(i_m),c^{m'}(i_{m'})\rangle\ , \tag{2.4}$$

Каждый из членов правой стороны равенства можно преподсчитать и хранить в таблице поиска (важно, что они не зависят от запроса). Число операций, требуемых для вычисления $\|x\|^2$, составляет порядка $M^2/2$ обращений в таблицы поиска и $M^2/2$ сложений (в предположении, что используется симметричность скалярных произведений). Важно отметить, что сложность вычисления чле-

на $\|x\|^2$ не зависит от размерности данных D. Однако, эта сложность растет квадратично по M и может значительно замедлить вычисления расстояний. Этих дополнительных временных затрат можно избежать путем использования небольшого количества дополнительной памяти, используя тот факт, что $\|x\|^2$ не зависит от запроса q. Для этого можно закодировать член $\|x\|^2$ в один дополнительный байт путем одномерной квантизации этих скалярных значений. На этапе компрессии каждый код Аддитивной квантизации дополняется соответствующим байтом, кодирующим член $\|x\|^2$. В этом случае вычисление значения $\|x\|^2$ требует одной операции обращения к таблице поиска.

Наконец, можно заметить, что для многих приложений требуется вычислять лишь значение скалярного произведения, и в этом случае вычисление члена $\|x\|^2$ не осуществляется.

Кодирование векторов

Опишем алгоритм кодирования векторов в модели Аддитивной квантизации, то есть нахождение кода для конкретного вектора x с имеющимися словарями $C^1 \dots C^M$. Необходимо найти код, который минимизирует ошибку кодирования E:

$$E(i_1,i_2,\dots i_m) = \|x - \sum_{m=1}^{M} c^m(i_m)\|^2 \tag{2.5}$$

Используя (2.2), можно переписать выражение для ошибки следующим образом:

$$E(i_1,i_2,\dots i_m) = \sum_{m=1}^{M} \left[-2\langle x,c^m(i_m)\rangle + \|c^m(i_m)\|^2\right] + \sum_{1\le m<m'\le M} \left[2\langle c^m(i_m), c^{m'}(i_{m'})\rangle\right] + \|x\|^2 \tag{2.6}$$

Для заданного вектора x член $\|x\|^2$ постоянен, а члены $U_m(i_m) = -2\langle x,c^m(i_m)\rangle + \|c^m(i_m)\|^2$ могут быть преподсчитаны и сохранены в таблицах поиска. Члены $V_{m,m'}(i_m,i_{m'}) = 2\langle c^m(i_m), c^{m'}(i_{m'})\rangle$ можно вычислить независимо от

x. После этих предвычислений и отбрасывания константного члена функция ошибки (2.6) может быть переписана как

$$E(i_1,i_2,\dots i_m) = \sum_{m=1}^{M} U_m(i_m) + \sum_{1\le m<m'\le M} V_{m,m'}(i_m,i_{m'}) \ , \tag{2.7}$$

что представляет собой функцию энергии полносвязного марковского случайного поля с попарными потенциалами. Важно отметить, что фукнция ошибки не зависит от размерности данных D. Получившаяся задача оптимизации может быть решена любыми существующими алгоритмами из теории марковских случайных полей (например, Loopy Belief Propagation (LBP) [47], Iterative Conditional Models (ICM) [48]). Но из-за полносвязности попарных потенциалов алгоритмы LBP и ICM из эффективной реализации [49] работают неудовлетворительно.

Поэтому для эффективного построения кода предлагается другой приближенный алгоритм, основанный на идее лучевого поиска [50]. Предлагаемый алгоритм начинает свою работу, находя N элементов, ближайших к x, из множества $C{=}C_1{\cup}\ldots{\cup}C_M$. Эти элементы становятся начальными приближениями для кортежей, являющихся кандидатами на роль оптимального кода, обеспечивающего минимальную ошибку сжатия. Эти N неполных кортежей постепенно заполняются в течение следующих $M{-}1$ итераций алгоритма. На итерации m ($m{>}1$) лучевой поиск рассматривает каждый неполный кортеж, содержащий $m{-}1$ векторов из $m{-}1$ словаря. Алгоритм обрабатывает оставшиеся $M{+}1{-}m$ словаря и находит среди слов этих словарей N элементов, ближайших к вектору разности x и суммы слов, уже содержащихся в кортеже. Таким образом, после обработки всех из N кандидатов создается N^2 кортежей длины m. Из них алгоритм отбирает N кортежей, соответствующих минимальной ошибке сжатия, и отправляет их на следующую итерацию в качестве кандидатов. После M таких итераций, когда кортежы становятся полными, кортеж, соответсвующий минимальной ошибке, возвращается в качестве результата.

Во время лучевого поиска используются таблицы поиска, чтобы сложность всех операций не зависела от размерности кодируемых данных D. Для разумных значений N (например, $N{=}16$ или $N{=}32$) лучевой поиск решает оп-

тимизационную задачу (2.7) лучше, чем алгоритмы из теории марковских случайных полей (в рамках заданного временного бюджета). Поэтому для решения задачи (2.7) в приложениях автором рекомендуется использовать лучевой поиск.

Обучение словарей

В этой части рассматривается задача обучения словарей для модели Аддитивной квантизации, то есть задача нахождения множества из M словарей $\{C^1,C^2,\dots C^M\}$, которыми можно закодировать множество векторов $X = \{x_1,\dots,x_n\}$ с наименьшей ошибкой сжатия. Таким образом, необходимо решить следующую задачу минимизации:

$$\min_{\substack{C^1,\dots,C^M\subset\mathbf{R}^D\\ |C^m|=K\\ i_j^m\in 1..K}} \sum_{j=1}^{n} \|x_j - \sum_{m=1}^{M} c^m(i_j^m)\|^2 \tag{2.8}$$

Аналогично существующим подходам к обучению словарей, минимизация осуществляется блочно-координатным спуском, то есть целевая функция по очереди минимизируется по кодам i_j^m (дискретным переменным) и по словам $c^m(\cdot)$ (непрерывным переменным). Предложенный алгоритм оптимизации обобщает стандартный алгоритм k-средних (который эквивалентен случаю M=1).

Минимизация по дискретным переменным кодов i_j^m при фиксированных словарях эквивалентна задаче кодирования векторов x_j с этими словарями и обсуждалась в предыдущей части. Оптимизация по словам $c^m(\cdot)$ эквивалентна следующей задаче наименьших квадратов:

$$\min_{\{c^m(k)\}} \sum_{j=1}^{n} \|x_j - \sum_{m=1}^{M}\sum_{k=1}^{K} a_{km}^j c^m(k)\|^2 \tag{2.9}$$

$$a_{km}^j = \begin{cases} 1, & \text{если } i_j^m{=}k \\ 0, & \text{иначе} \end{cases}$$

Хотя задача наименьших квадратов (2.9) может казаться большой (так как могут быть велики значения n и D), ее можно декомпозировать на D задач, получая отдельную задачу для каждой размерности. Таким образом, одну задачу с KMD переменными можно свести к решению D задач с KM переменными, каждая из которых может быть сформулирована в виде переопределенной системы линейных уравнений:

$$\forall j = 1..n \quad \sum_{m=1}^{M} \sum_{k=1}^{K} a_{km}^{j} c^{m}(k)_{d} = x_{j,d} \ , \tag{2.10}$$

где $c^{m}(k)_{d}$ — d-ая координата $c^{m}(k)$, а $x_{j,d}$ — d-ая координата x_j. (2.10) определяет n уравнений с KM переменными, которые решаются в смысле наименьших квадратов. В качестве дополнительной оптимизации можно заметить, что D переопределенных систем (2.10) для различных размерностей отличаются только правыми частями равенств, а разреженная матрица с левой стороны может храниться и переиспользоваться для всех размерностей. Таким образом, основной вклад в сложность этапа обучения словарей вносит оптимизация по дискретным переменным.

В качестве инициализации дискретных переменных можно брать случайные числа от 1 до K. Еще один возможный вариант заключается в том, что можно инициализировать словари словами полученными путем мультиквантизации или оптимизированной мультиквантизации (слово из словаря мультиквантизации можно представить полноразмерным вектором, добавив нули вместо отсутствующих координат).

Аддитивная мультиквантизация

Сложность алгоритма лучевого поиска растет кубически по M. Для больших степеней сжатия (например, M=4) сложность кодирования может быть приемлемой даже для большого количества кодируемых векторов. Но для больших значений M необходимо разрабатывать более эффективные алгоритмы. Альтернативным вариантом снижения сложности кодирования в случае больших значений M является комбинация подходов Аддитивной и мультиквантизации

зации. В этой комбинации кодируемый вектор разбивается на M_1 ортогональных подвекторов (как в мультиквантизации), а затем каждый из подвекторов кодируется Аддитивной квантизацией в M_2 байта. Таким образом, можно имея малое значение M_2 кодировать каждый вектор $M_1 \times M_2$ байтами. Как правило, такая комбинация двух моделей квантизации обеспечивает более точное кодирование по сравнению с мультиквантизацией с $M = M_1 \times M_2$ байт. Такая модификация носит название Аддитивная мультиквантизация (АМК).

2.2.2 Эксперименты

В этой части приводятся результаты экспериментального сравнения модели Аддитивной квантизации (АК) с существующими методами сжатия — мультиквантизацией (МК) [12] и оптимизированной мультиквантизацией (ОМК) [14]. Эти методы, основанные на идеях квантизации, способны сжимать векторы высокой размерности с гораздо большей точностью по сравнению с методами бинарного кодирования согласно нескольким последним работам [14; 51].

Во всех экспериментах используются словари размера $K = 256$ и количество словарей M=4,8,16. Для кодов длины M=8 и более рассматривается также комбинация двух подходов под названием Аддитивная мультиквантизация, которая использует оптимизированную мультиквантизацию для ортогонального преобразования данных, а затем применяет Аддитивную квантизацию (на 4 байта) к подвекторам преобразованного вектора (например, к половинам в случае кодирования на 8 байт). Во всех экспериментах словари и матрицы преобразований обучались на множествах векторов, не пересекающихся с тестовым, чтобы исключить эффект переобучения. Параметр N в лучевом поиске равнялся 16 во время обучения и 64 во время кодирования.

Сначала сравниваются ошибки сжатия каждым из методов. На рисунке 2.2 изображена средняя ошибка сжатия как функция длины кода для датасета SIFT1M [12], содержащего один миллион 128-мерных СИФТ векторов. Для кодов всех длин (M=4,8,16) ошибка сжатия, получаемая Аддитивной (мульти)квантизацией значительно меньше, чем ошибки, полученные мультиквантизацией и оптимизированной мультиквантизацией. Также необходимо прове-

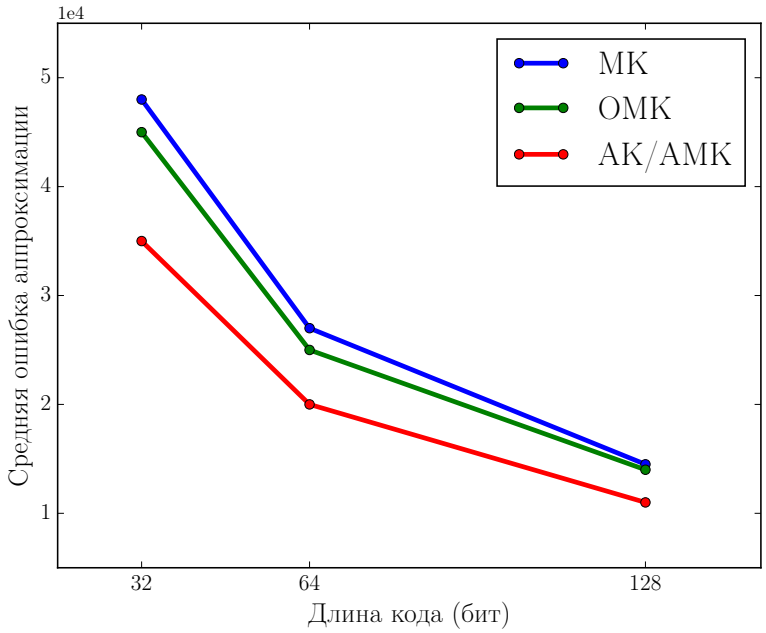


Рисунок 2.2 — Средние ошибки сжатия различными методами векторов из датасета SIFT-1M для кодов различной длины (4, 8, 16 байт). Аддитивная квантизация использовалась для 4 байт, Аддитивная мультиквантизация — для 8 и 16 байт. Для всех длин кодов ошибка кодирования Аддитивной (мульти)квантизации ниже по сравнению с существующими методами.

рить, что более качественное сжатие приводит к более высокому качеству в прикладных задачах. Ниже рассмотрим два приложения: (1) приближенный поиск ближайшего соседа, (2) классификация изображений с ограниченной памятью.

Также сравнивалось качество кодирования с помощью аддитивной квантизации и мультиквантизации для различных размеров словарей K. Рисунок 2.3 демонстрирует, что Аддитивная квантизация обеспечивает более качественное сжатие во всем диапазоне значений K.

Поиск ближайшего соседа

Приближенный поиск ближайшего соседа — естественная задача для сравнения различных способов кодирования. Множество векторов, по которым будет осуществляться поиск, сжимается различными методами, участвующими в

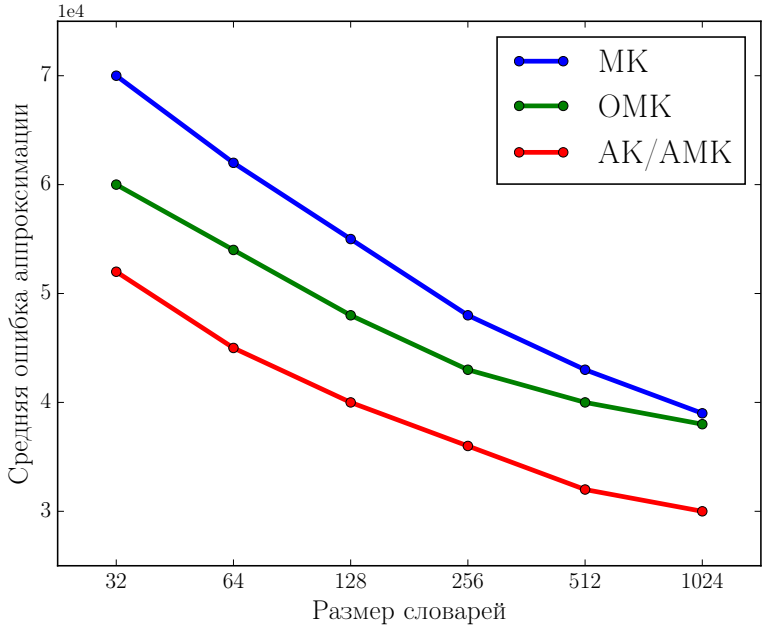


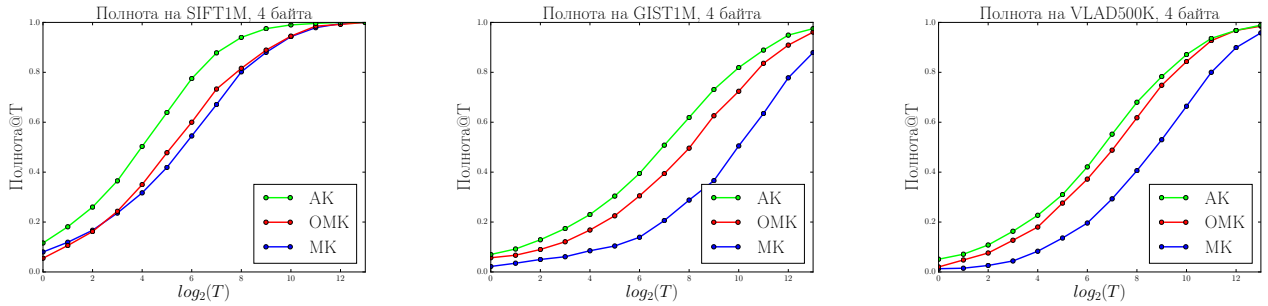
Рисунок 2.3 — Ошибки сжатия 10,000 СИФТ векторов с помощью Аддитивной квантизации и оптимизированной мультиквантизации с 4 словарями различных размеров. Для любого размера словарей Аддитивная квантизация обеспечивает меньшую ошибку сжатия.

сравнении. Также имеется некоторое количество несжатых векторов-запросов, для которых преподсчитаны истинные ближайшие соседи в смысле евклидовой метрики. Для каждого из векторов-запросов вычисляется расстояние до каждого из пожатых векторов поисковой базы, после чего вектора переранжируются по возрастанию расстояния до запроса. После переранжирования измеряется качество с помощью метрики полнота@T [12], которая равна доле запросов, для которых истинный ближайший сосед принадлежит множеству из T ближайших сжатых векторов. Качество методов продемонстрировано в виде кривых зависимости величины полнота@T от T.

Ниже описаны поисковые базы, на которых проводились эксперименты.

SIFT-1M: Поисковая база, предложенная в работе [12], содержит один миллион 128-мерных СИФТ дескрипторов [29], а также отдельное множество из 100.000 векторов для обучения. Поисковая база содержит 10.000 запросов, для которых известны истинные ближайшие соседи.

GIST-1M: Поисковая база, также предложенная в работе [12] содержит один миллион 960-мерных ГИСТ дескрипторов [52] в качестве основного мно-



SIFT-1M **GIST-1M** **VLAD-500K**

Рисунок 2.4 — Полнота@T для различных методов сжатия и поисковых баз SIFT-1M, GIST-1M, VLAD-500K для кодов длиной M=4 байт. Для всех поисковых баз значения метрики Полнота@T для Аддитивной квантизации гораздо выше, чем для различных версий мультиквантизации.

жества, и 500,000 векторов в множестве для обучения. Также имеется 1000 запросов с известными истинными ближайшими соседями.

VLAD-500K: Поисковая база содержит VLAD дескрипторы [5] естественных изображений. Дескрипторы сжаты методом главных компонент до размерности 128. Основное множество и множество для обучения содержать по 500.000 векторов. Имеется 1000 запросов, для которых преподсчитаны истинные ближайшие соседи в основном множестве.

СИФТ и ГИСТ дескрипторы представляют собой различные гистограммы градиентов, в то время как VLAD дескрипторы обладают более сложной структурой. С другой стороны СИФТ и VLAD дескрипторы имеют относительно небольшую размерность $D = 128$, а ГИСТ дескрипторы — гораздо большую размерность 960.

Рисунок 2.4 демонстрирует качество различных методов компрессии (AK, MK, OMK) для высокой степени сжатия, соответствующей кодам длиной 4 байта. Для всех поисковых баз максимальное качество достигается с помощью метода AK, причем максимальное преимущество достигается для поисковой базы из СИФТ дескрипторов.

Далее исследуем качество методов на множестве СИФТ дескрипторов для случая кодов длиной $M = 8$ байт. Помимо стандартной AK также в сравнении участвует две ее модификации: AMK (комбинация аддитивной и мультиквантизации) и AK-7, где $M = 7$ байт используется для кодирования исходного вектора и еще один восьмой байт для кодирования его длины (как обсуждалось в части 2.2.1). AK-7 требует тех же затрат по памяти, что и (O)MK **и**

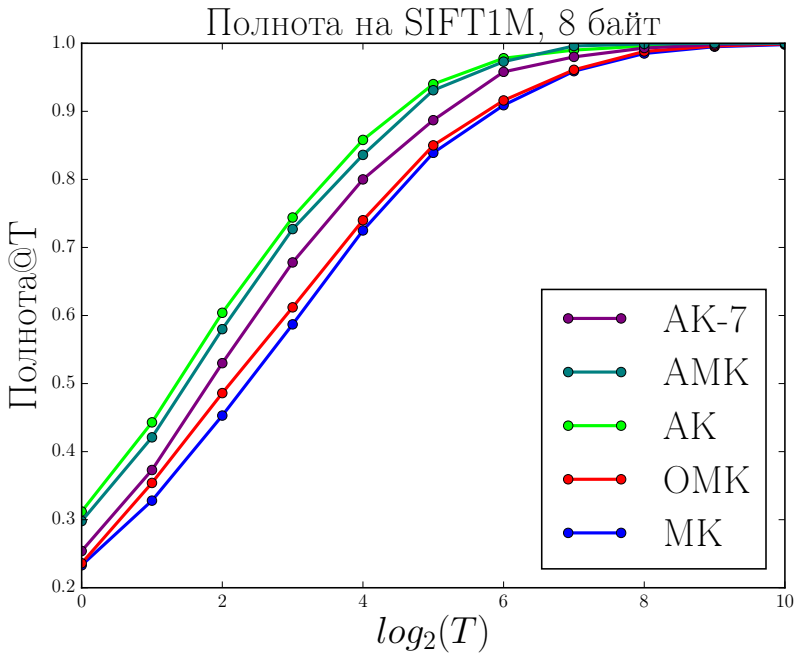


Рисунок 2.5 — Сравнение различных методов сжатия для кодов длиной $M = 8$ байт. Метод АК-8 достигает максимальной полноты, АМК-8 и АК-7 слегка уступают по полноте, но гораздо быстрее АК-8 в смысле скорости, см. таблицу 2.1. АК-7 также обходит по скорости все версии мультиквантизации, в то же время демонстрируя более высокие значения точности.

более низкую стоимость вычисления расстояний (см. таблицу 2.1). Как можно видеть на рисунке 2.5 все варианты Аддитивной квантизации достигают большей полноты, чем методы (оптимизированной)мультиквантизации.

Наконец, рисунок 2.6 демонстрирует кривые, соответствующие кодам различной длины для методов АК, АМК, МК, ОМК (для кодов длины $M = 4$ байта используется АК, для кодов большей длины — АМК). Качество всех методов растет при увеличении длины кода, а преимущество Аддитивной квантизации сохраняется для всего диапазона длин кодов. С увеличением длины кода M разница в качестве различных методов уменьшается.

Классификация

Еще одной прикладной задачей, в которой необходимо сжатие дескрипторов, является классификация изображений. Имеет место два сценария исполь-

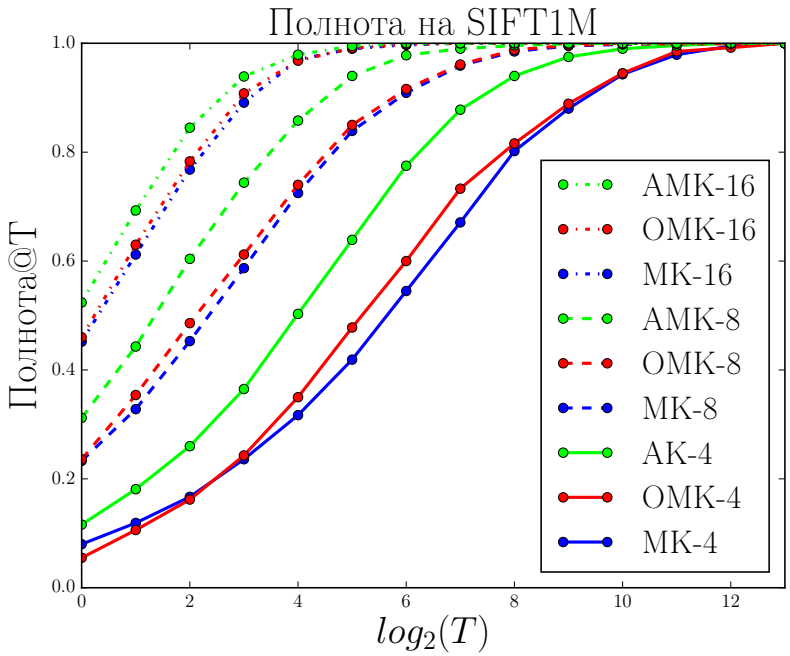


Рисунок 2.6 — Преимущество AK и AMK надо (O)MK для кодов различных длин. Самое большое преимущество методов Аддитивной квантизации имеет место в случае коротких кодов на четыре байта.

зования методов сжатия. В первом сценарии сжатие может быть применено к обучающему множеству дескрипторов, для того, чтобы ускорить обучение на очень больших коллекциях, когда необходимо, чтобы обучающее множество полностью помещалось в оперативную память [53; 54].

Во втором, и, вероятно, более важном сценарии сжатие дескрипторов применяется к тестовому множеству. В этом случае классификтор обучается на небольшом множестве изображений, а затем применяется к большой базе сжатых дескрипторов для того, чтобы извлечь изображения с максимальной уверенностью классификации [55–57]. Интересно, что в этом сценарии необходимо вычислять только скалярные произведения между запросом (классификатором) и сжатыми векторами. Так как вычисление члена $\|x\|^2$ в этом случае не нужно, эффективность поиска одинакова в случае использования AK или (O)MK даже без использования дополнительного байта памяти для хранения длины сжатого вектора.

Эксперименты, соответствующие обоим сценариям быди проведены на коллекции PASCAL VOC 2007 [58]. В качестве дескрипторов изображений использовались Фишеровские векторы [59] с 256 компонентами над СИФТ векторами, пожатыми до размерности 80. Пространственные пирамиды в этих экспе-

Метод	OMK	AK	AMK	AK-7
Замедление относительно MK, 4 байта	1.05	2.55	—	—
Замедление относительно MK, 8 байт	1.05	5.46	2.59	0.92

Таблица 2.1

Скорость нахождения ближайшего соседа относительно мультиквантизации для различных методов сжатия для кодов длин M=4 и M=8 байт. Скорость мультиквантизации предполагается равной единице. В случае M=8 байт метод AK-7 является самым быстрым и также превосходит MK и OMK по точности. AK и AMK достигают большей точности, но работают медленнее по сравнению с AK-7.

риментах не использовались. Целью эксперимента было пронаблюдать, каким образом снижается качество методов AK и OMK с увеличением степени сжатия. В обоих методах исходные Фишеровские векторы равномерно разбивались на R подвекторов и каждый подвектор сжимался методами OMK и AK до $M = 4$ байт (что дает степень сжатия, равную R). Различные степени сжатия могут быть получены путем изменения R. Для оценки используется метрика средней точности классификации.

Эксперимент 1: сжатие обучающего множества. В этом эксперименте для обучения словарей использовалось тестовое множество коллекции PASCAL. Классификаторы обучались на обучающей части коллекции с использованием как несжатых векторов, так и векторов, сжатых с помощью AMK или OMK (которые были реконструированы как в работе [53]). Как только классификаторы были выучены, их применяли к тестовому множеству и измерялась средняя точность классификации. Как можно видеть на рисунке 2.7, снижение качества гораздо меньше при использовании обучающих данных, сжатых методом AMK.

Эксперимент 2: сжатие тестового множества. В этом эксперименте исследовался второй сценарий и классификаторы обучались с использованием несжатых дескрипторов на обучающей части множества PASCAL. Далее оценивался эффект сжатия тестового множества (которое происходило с теми же параметрами, что и в Эксперименте 1). Как и в предыдущем эксперименте, рисунок 2.8 демонстрирует, что снижение качества вследствие сжатия гораздо меньше, если это сжатие осуществлялось методом AMK.

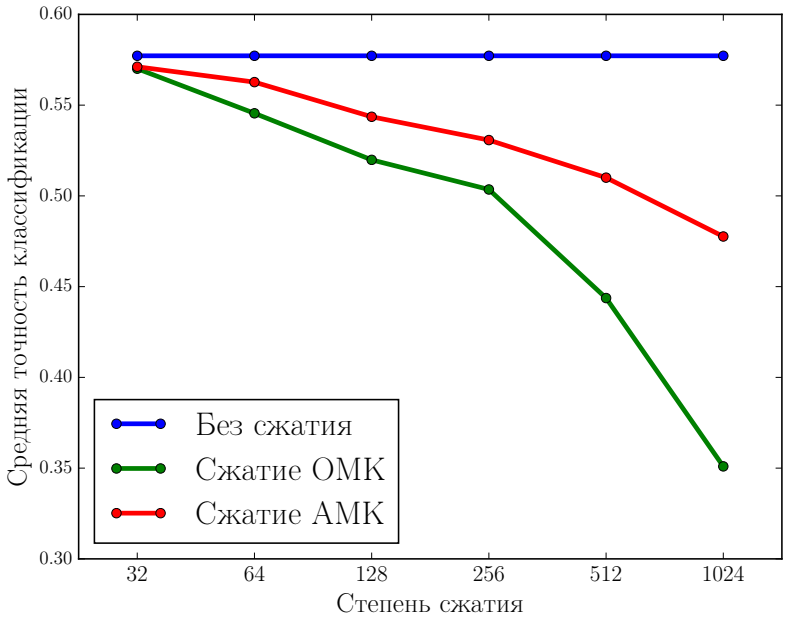


Рисунок 2.7 — Средняя точность классификации для **обучения на сжатых данных** и тестирования на несжатых данных. Словари для AMK И OMK были обучены на тестовом множестве и использовались для кодирования обучающего множества. Классификаторы были обучены на сжатом обучающем множестве и протестированы на тестовом множестве. Более точное кодирование с помощью AMK проводит к более высокой точности классификации.

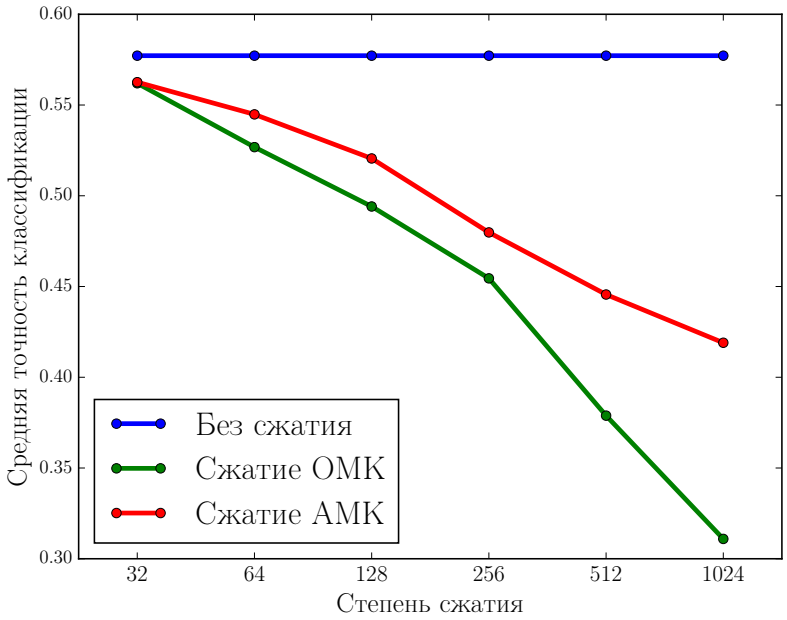


Рисунок 2.8 — Средняя точность классификации для обучения на несжатых данных и **тестирования на сжатых данных**. Классификаторы были обучены на обучающем множестве и были протестированы на тестовом множестве. Более точное кодирование с помощью AMK проводит к более высокой точности классификации.

2.2.3 Выводы о модели Аддитивной квантизации

Предложенный метод сжатия высокоразмерных векторов — Аддитивная квантизация — достигает более высокой точности кодирования по сравнению с передовыми существующими методами, что приводит к повышению качества решения прикладных задач: поиска ближашего соседа и классификации изображений.

По скорости Аддитивная квантизация не уступает существующим методам в случае вычисления скалярного произведения. Вычисление евклидового расстояния медленнее из-за необходимости вычислять член $\|x\|^2$ (см. таблицу 2.1), что можно обойти использованием дополнительного байта, кодирующего этот член (тогда скорость AK такая же, как у MK).

Основной практический недостаток предложенного метода — высокая вычислительная сложность этапов кодирования и обучения словарей, особенно в случае кодов большой длины.

2.3 Древесная квантизация

В предыдущей главе 2.2.1 описан метод сжатия векторов высокой размерности — Аддитивная квантизация (АК). Его основным недостатком является высокая вычислительная сложность кодирования векторов. АК предлагает использовать лучевой поиск для приближенного решения оптимизационной задачи, но даже в этом случае сложность остается высокой, особенно в случае кодов большой длины. На практике АК оказывается в несколько раз медленнее (оптимизированной)мультиквантизации ((О)МК), что является серьезным ограничением для использования в приложениях, особенно в тех случаях, когда необходимо быстро закодировать новый вектор.

В этой главе описывается еще один метод кодирования — Древесная квантизация (ДК), принадлежащий тому же семейству методов, что АК и МК. Так же как в АК и МК, ДК использует множество из M словарей и, по аналогии с АК, кодирует вектор суммой M слов, по одному из каждого словаря. Кодом в ДК, как и в других методах, является кортеж из M номеров слов. Отличие ДК от АК заключается в том, что ДК накладывает на словари некоторые ограничения, а именно структуру “кодирующего дерева”. “Кодирующее дерево” — граф, являющийся деревом, вершины которого соответствуют словарям, а каждая из D координат приписана к некоторому ребру этого графа. Каждый словарь кодирует только те координаты, которые приписаны к ребрам, инцидентным соответствующей вершине (см. рисунок 2.9). Все остальные размерности всех слов в этом словаре равны нулю.

Результаты эсперимента, приведенные ниже, демонстрируют, что благодяра тому, что алгоритм кодирования в Древесной квантизации оптимален (то есть находит глобально оптимальный код при заданных словарях), ошибка кодирования в ДК сопоставима с ошибкой кодирования, достигаемой методом АК (который не накладывает ограничений на словари, но не гарантирует оптимальности кодирования). Это приводит к тому, что в практических задачах ДК обеспечивает такое же качество, что и АК, при этому допуская гораздо более быстрое кодирование.

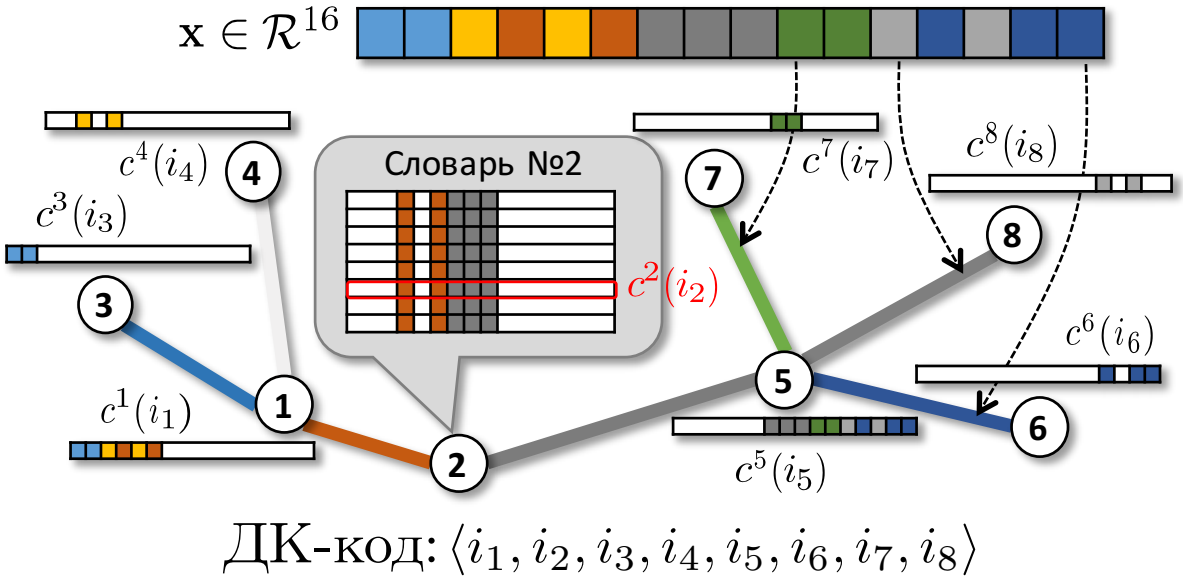


Рисунок 2.9 — **Древесная квантизация** для кодирования D-мерных векторов (здесь D=16). Каждая координата приписана к одному из ребер **кодирующего дерева** (приписывания отмечены цветами). Каждая из M=8 вершин кодирующего дерева содержит словарь (показан для вершины #2). Каждый словарь кодирует вершины с инцидентных ребер (также отмечены цветами). Кодируемый вектор $\mathbf{x}$ аппроксимируется суммой M слов $c^t(i_t)$ из словарей в вершинах (изображены прямоугольниками, в которых активные координаты отмечены цветами; положение слова $c^2(i_2)$ внутри второго словаря отмечено красным).

2.3.1 Модель Древесной квантизации

В этой части обсуждается модель кодирования в модели ДК, а также эффективное вычисление скалярных произведений и евклидовых расстояний между сжатыми и несжатыми векторами.

Кодирующее дерево

Предположим, что кодируются векторы из D-мерного евклидового пространства $\mathcal{R}^D$, и что векторы кодируются с помощью M словарей C^1,C^2,...,C^M. Каждый словарь содержит K слов, $c^m(i)$ — i-ое слово в m-ом словаре и $c^m(i)[d]$ — d-ая координата в этом слове. Также как и предыдущих моделях, вектор кодируется M числами от 1 до K, то есть M байтами, соответствующими номерам слов в каждом из словарей. Кортеж из номеров $[i_1,i_2,\ldots,i_M]$ означает, что вектор x аппроксимируется суммой соответствующих слов:

$$x \approx \sum_{m=1}^{M} c^m(i_m), \quad i_m \in 1..K \tag{2.11}$$

В отличие от АК, не накладывающей ограничений на слова, ДК использует "кодирующее дерево" (см. 2.9) в качестве таких ограничений. Кодирующее дерево $\mathcal{T}$ — граф, являющийся деревом с M вершинами, каждая из которых соответствует некоторому словарю. Далее обозначение $(m,n) \in \mathcal{T}$ для $m \in 1..M$ и $n \in 1..M$ подразумевает, что m-ая и n-ая вершины соединены в дерево ребром. Каждая из D координат в $\mathcal{R}^D$ приписана к единственному ребру в дереве $\mathcal{T}$. Обозначим за $\mathcal{D}_{m,n}$ множество координат, которые приписаны к ребру $(m,n) \in \mathcal{T}$. Так как каждая координата приписана только к одному ребру, эти множества имеют пустое пересечение. В этой и последующих частях обозначения (m,n) и (n,m) являются эквивалентными, так как кодирующее дерево — неориентированный граф.

Далее обозначим за $\mathcal{D}_m$ объединение всех множеств размерностей, приписанных к ребрам, инцидентным вершине m, то есть:

$$\mathcal{D}_m = \{\bigsqcup \mathcal{D}_{m,n} \,|\, n \in 1..M,\, (m,n) \in \mathcal{T}\}\,. \tag{2.12}$$

В модели ДК все слова из m-ого словаря могут иметь ненулевыми только координаты из $\mathcal{D}_m$, то есть:

$$\forall i, d \notin \mathcal{D}_m,\, c^m(i)[d] = 0\,. \tag{2.13}$$

В результате, каждая координата кодируется **двумя** словарями, соответствующими вершинам, инцидентным ребру, к которому она приписана. В этом заключается отличие модели ДК от существующих моделей: в мультиквантизации каждая координата кодируется строго одним словарем, а аддитивная квантизация кодирует каждую координату всеми M словарями. Важным следствием ограничений, накладываемых в модели ДК, является то, что два слова обязательно ортогональны, если они принадлежат словарям, сответствующие вершины которых не соединены ребром в кодирующем дереве.

$$\forall (m,n) \notin \mathcal{T}, \forall i,j: \quad \langle c^m(i), c^n(j)\rangle = 0\,. \tag{2.14}$$

Эффективное вычисление расстояний и скалярных произведений

Модель аппроксимации (2.11) допускает эффективное вычисление скалярных произведений и евклидовых расстояний между сжатыми и несжатыми векторами. Как и в предудыщих моделях эффективность достигается за счет использования таблиц поиска.

Рассмотрим несжатый вектор-запрос q и множество из L сжатых векторов $x_1, \ldots, x_L$, причем $L \gg K$. Каждый x_j преставлен своим кодом $[i_1^j, i_2^j, \ldots i_M^j]$, то есть:

$$x_j = \sum_{m \in 1..M} c^m(i_m^j)\,. \tag{2.15}$$

Тогда скалярное произведение q и x_j можно вычислить как:

$$\langle q,x_j\rangle = \sum_{m=1}^{M}\langle q,c^m(i_m^j)\rangle = \sum_{m=1}^{M} T^m(i_m^j)\ , \tag{2.16}$$

где члены $T^m(\cdot) = \langle q,c^m(\cdot)\rangle$ можно преподсчитать и хранить а таблицах поиска для данного запроса q. Сложность преподсчета этих таблиц не зависит от L и для достаточно больших множеств векторов этой сложностью можно пренебречь. Помимо этих затрат сложность вычисления скалярного произведения для одного вектора состоит из M обращений к таблицам поиска и $M-1$ сложений.

При вычислении евклидового расстояния между q и сжатым вектором необходимы дополнительные вычисления: Так как:

$$\|q-x\|^2 = \|q\|^2 - 2\langle q,x\rangle + \|x\|^2\,, \tag{2.17}$$

также необходимо вычислить $\|x_j\|^2$ для каждого сжатого вектора (член $\|q\|^2$ можно посчитать один раз для всех сжатых векторов). Вычисление нормы $\|x_j\|^2$ с использованием представления (2.11) можно сделать слудующим образом:

$$\begin{aligned}\|x_j\|^2 = \|\sum_{m=1}^{M} c^m(i_m^j)\|^2 = \sum_{m=1}^{M}\|c^m(i_m^j)\|^2 + \\ 2\sum_{(m,n)\in\mathcal{T}}\langle c^m(i_m^j), c^n(i_n^j)\rangle\end{aligned} \tag{2.18}$$

Здесь используется ортогональность (2.14) для удаления всех членов таких, что $(m,n)\notin\mathcal{T}$. Для более эффективных вычислений члены $\|c^m(\cdot)\|^2$ могут быть добавлены к значениям $T^m(\cdot)$ в таблицах поиска, описанных выше, а члены $\langle c^m(\cdot), c^n(\cdot)\rangle$ можно хранить в отдельной таблице поиска, не зависящей от запроса. Дополнительная сложность при вычислении евклидового расстояния относительно скалярного произведения таким образом составляет M операций обращения к таблицам поиска и столько же операций сложения. Важно отметить, что для АК дополнительная сложность квадратична по M, что делает процесс кодирования существенно более дорогим.

Кодирование и обучение модели

Модель Древесной квантизации полностью определяется кодирующим деревом $\mathcal{T}$ и словарями $C^m(k)$, согласованными с этим деревом. В этой части описывается (1) каким образом находится оптимальный код для кодируемого вектора при данных словарях и структуре кодирующего дерева, (2) каким образом словари и структура кодирующего дерева могут быть обучены по данным.

Кодирование В модели ДК, чтобы найти оптимальный код $[i_1,i_2,\dots i_m]$ для вектора x при данных словарях $C^1 \dots C^M$, необходимо минимизировать ошибку аппроксимации (2.11):

$$E(i_1,i_2,\dots i_M) = \|x - \sum_{m=1}^{M} c^m(i_m)\|^2 \longrightarrow \min_{i_m} \tag{2.19}$$

Используя (2.17), выражение можно переписать как:

$$E(i_1,i_2,\dots i_M) = \sum_{m=1}^{M} \left(-2\left\langle x,c^m(i_m)\right\rangle + \|c^m(i_m)\|^2\right) + \sum_{(m,n)\in\mathcal{T}} 2\left\langle c^m(i_m), c^n(i_n)\right\rangle, \tag{2.20}$$

где свойство ортогональности (2.14) используется, чтобы удалить члены $(m,n) \notin \mathcal{T}$, константный член $\|x\|^2$ также опущен.

Минимизация функции (2.20) тогда становится эквивалентной задаче вывода в марковском случайном поле, определяемом кодирующим деревом, где члены $U_m(i_m) = -2\langle x,c^m(i_m)\rangle + \|c^m(i_m)\|^2$ соответствуют унарным потенциалам. Члены $V_{m,n}(i_m,i_n) = 2\langle c^m(i_m), c^n(i_n)\rangle$ можно преподсчитать и они соответствуют бинарным потенциалам.

В этом случае задача может быть решена с использованием динамического программирования (алгоритм максимум-произведение) в графе со структурой дерева [47], которое в этом случае гарантированно находит глобальный минимум. Вычислительная сложность этого алгоритма составляет $O(MK^2)$, а предвычисление унарных потенциалов имеет сложность $O(KD)$. Как правило,

работа алгоритма максимум-произведение занимает основную часть времени, но тем не менее это кодирование гораздо быстрее по сравнению с кодированием на основе лучевого поиска в Аддитивной квантизации. Гарантированная оптимальность и эффективность кодирования — ключевое преимущество Древесной квантизации по сравнению с Аддитивной.

Обучение словарей. Опишем решение задачи обучения модели Древесной квантизации для конкретного множества векторов. Предполагается, что имеется обучающее множество $X = \{x_1,x_2,\ldots,x_L\}$ из L векторов, для которого минимизируется суммарная ошибка аппроксимации, причем минимизация осуществляется по кодам и по параметрам модели.

Введем некоторое количество бинарных переменных-индикаторов $A = \{a_{(m,n)}[d]\}$, таких что $a_{(m,n)}[d] = 1$ если и только если координата d приписана ребру (m,n). Важно заметить, что для каждой координаты d в точности одна переменная $a_{(m,n)}[d] = 1$. Напомним, что если обучающий вектор x_j имеет код $[i_1,i_2,\ldots,i_M]$ и размерность d приписана ребру (m,n), тогда $x_j[d]$ приближается суммой $x_j[d] \approx c^m(i_j^m)[d] + c^n(i_j^n)[d]$. Тогда суммарная ошибка аппроксимации всех обучающих векторов может быть записана как:

$$G(\{i_j^m\},\{C^m\},A;X) = \sum_{j=1}^{L}\sum_{d=1}^{D}\sum_{(m,n)\in\mathcal{F}} a_{(m,n)}\left(c^m(i_j^m)[d] + c^n(i_j^n)[d] - x_j[d]\right)^2 . \tag{2.21}$$

Здесь, $\mathcal{F}$ обозначает множество ребер в полном графе на M вершинах: $\mathcal{F} = \{(m,n)\,|\,1 \le m < n \le M\}$.

Для того, чтобы обучить оптимальную модель, необходимо минимизировать функционал G в выражении (2.3.1) по всем аргументам с тем ограничением, что все переменные-индикаторы должны быть консистентны с некоторым деревом ($\exists\mathcal{T} : a_{(m,n)}[d] = 1 \Rightarrow (m,n) \in \mathcal{T}$). По аналогии с существующими алгоритмами квантизации минимизация (2.3.1) осуществляется блочно-координатным спуском. Это означает, что происходит попеременная минимизация по переменным кодов $\{i_j^m\}$ при фиксированных параметрах модели $\{C^m\}$ и A (“Е-шаг”) и наоборот (“М-шаг”). Минимизация по кодам при заданных пара-

метрах модели эквивалентна задаче нахождения оптимального кода для каждого вектора и ее решение было описано в 2.3.1. Поэтому необходимо обсудить только М-шаг, то есть оптимизацию G по переменным $\{C^m\}$ и A при фиксированных кодах $\{i_j^m\}$.

Важно отметить, что когда переменные кодов $\{i_j^m\}$ зафиксированы, целевая функция декомпозируется на сумму ошибок реконструкции для независимых координат. Обозначим за $r_{(m,n)}[d]$ ошибку реконструкции для координаты d, которая суммируется по всем обучающим векторам если эта размерность приписана ребру (m,n). Тогда в предположении, что элементы словарей $c^m(k)[d]$,$c^n(k)[d]$ являются оптимальными для всех $k = 1..K$, имеем:

$$r_{(m,n)}[d] = \min_{\substack{c^m(\cdot)[d] \\ c^n(\cdot)[d]}} \sum_{j=1}^{L} \left(c^m(i_j^m)[d] + c^n(i_j^n)[d] - x_j[d] \right)^2 . \tag{2.22}$$

Следовательно, нахождение значения $r_{(m,n)}[d]$ требует решения задачи наименьших квадратов (2.22), в которой матрица имеет L строк и $2K$ столбцов, а также является очень разреженной (в каждой строке всего два ненулевых элемента, равных единице). Для заданного (m,n) матрица наименьших квадратов одинакова для всех координат d, так как только правая сторона равенства отличается у различных задач наименьших квадратов. Этим можно воспользоваться во время вычисления величин $r_{(m,n)}[d]$ для всех $d \in 1..D$.

Таким образом, минимизация по переменным словарей, может быть осуществлена вне обновления модели, и М-шаг сводится к только минимизации по переменным-индикаторам:

$$\min_{\{C^m\},A} G(\{i_j^m\},\{C^m\},A;X) =$$
$$\min_{A} \sum_{d=1}^{D} \sum_{(m,n)\in\mathcal{F}} a_{(m,n)}[d] \cdot r_{(m,n)}[d] . \tag{2.23}$$

Минимизация (2.3.1) осуществляется с теми ограничениями, что индикаторы должны быть согласованы с некоторым деревом. Введем еще одно семейство бинарных переменных индикаторов $e_{(m,n)}$, которые определяют, включено ли ребро (m,n) в кодирующее дерево. Минимизация с ограничениями функции

(2.3.1) тогда может быть сформулирована как задача бинарного целочисленного линейного программирования:

$$\underset{a_{(m,n)}[d],\, e_{(m,n)}}{\text{minimize}} \quad \sum_{d=1}^{D} \sum_{(m,n)\in\mathcal{F}} r_{(m,n)}[d] \cdot a_{(m,n)}[d] \tag{2.24}$$

$$\text{subject to} \quad a_{(m,n)}[d] \in \{0,1\},\ e_{(m,n)} \in \{0,1\},$$
$$(m,n) \in \mathcal{F},\ d = 1..D \tag{2.25}$$

$$\sum_{(m,n)\in\mathcal{F}} a_{(m,n)}[d] = 1, \quad d = 1..D \tag{2.26}$$

$$a_{(m,n)}[d] \leq e_{(m,n)},$$
$$(m,n) \in \mathcal{F},\ d = 1..D \tag{2.27}$$

$$\sum_{\substack{m\in V \\ n\in V, m<n}} e_{(m,n)} \leq |V| - 1,$$
$$V \subset \{1,\ldots,M\} \tag{2.28}$$

$$\sum_{(m,n)\in\mathcal{F}} e_{(m,n)} = M - 1\,. \tag{2.29}$$

Здесь (2.24) — линейная целевая функция, в которой $r_{(m,n)}[d]$ являются коэффициентами; (2.26) гарантирует, что каждая координата приписана единственному ребру; (2.27) — ограничение согласованности, гарантирующее, что размерности могут быть приписаны только к ребрам дерева; (2.28) — ограничения удаления циклов, определенные для всевозможных подмножеств вершин. Наконец, (2.29) гарантирует, что дерево имеет $M - 1$ ребро и таким образом является одиночным деревом, а не лесом.

Получившаяся задача целочисленного линейного программирования является нетривиальной. Однако, на практике оказывается, что для данных размерности несколько сотен и длин кодов 4–32 байт, существующие пакеты [60] способны решить ее за несколько минут. Следовательно, на M-шаге обучения словарей также может быть найден глобальный оптимум. Как только оптимальная структура дерева $\mathcal{T}$ и оптимальные значения индикаторов A получены из решения задачи линейного программирования, переменные $c^m(\cdot)[d]$ можно вос-

Algorithm 2.3.1: ОбучениеСловарейДревеснойКвантизации()

Вход: $X = \{x_1, \ldots, x_L\}$, M, K
$\{a_{(m,n)}[d]\}$, $\{C^m\}$, $\{i_j^m\}$ = Инициализация(X,M,K)
повторять до сходимости:
$\{C^m\}$ = РешитьНаименьшиеКвадраты() $\left(\{a_{(m,n)}[d]\}, \{i_j^m\}\right)$ // см. (2.22)
$\{a_{(m,n)}[d]\}$ = РешитьЦЛП() $\left(\{C^m\}, \{i_j^m\}\right)$ // см. (2.24) - (2.29)
$\{i_j^m\}$ = Максимум-Произведение $\left(\{C^m\}, \{a_{(m,n)}[d]\}\right)$ // см. (2.20)
результат $\{a_{(m,n)}[d]\}$, $\{C^m\}$

Рисунок 2.10 — Псевдокод процедуры обучения словарей.

становить с помощью решения задачи наименьших квадратов (2.22). Псевдокод полного протокола процедуры обучения словарей приведен на рисунке 2.10.

Глобальное вращение. Древесная квантизация допускает добавление в модель глобального ортогонального преобразования, по аналогии с оптимизированной мультиквантизацией [14; 46]. Можно использовать такой же самый подход с решением ортогональной Прокрустовой задачи для минимизации суммарной ошибки аппроксимации. Древесная квантизация с обученным ортогональным преобразованием называется оптимизированная Древесная квантизация (ОДК).

Инициализация. Процесс обучения модели ОДК, таким образом, итерируется между (1) М-шагом (переоценка дерева, приписывание координат к ребрам, обновление словарей), (2) Е-шагом (обновление переменных кодов для обучающих векторов), и (3) обновление матрицы ортогонального преобразования. Хотя каждый из этих шагов гарантированно достигает глобального оптимума и никогда не увеличивает ошибку аппроксимации, минимизация в целом может сходиться к локальному минимуму, зависящему от инициализации. Важно отметить, что можно доказать следующее утверждение о точности кодирования (О)ДК и О(МК) для кодов одинаковой длины:

Утверждение: *Пусть* $X = \{x_1, \ldots, x_L\}$ *— обучающее множество, пусть* $C^1, \ldots, C^M$, $|C^m| = K$ *— множество (О)МК словарей, обученных на этом множестве, и пусть* $\{i_j^m\}_{j=1\ldots L}^{m=1\ldots M}$ *— (О)МК коды обучающих векторов. Тогда возможно обучить (О)ДК словари для тех же* X, M, K, *такие что ошибка аппроксимации множества* X *с этим словарями не превышает ошиб-*

ку О(МК) со словарями $C^1,\ldots,C^M$*. Другими словами, можно гарантировать, что О(ДК) кодирует с не меньшей точностью, чем О(МК).*

Доказательство: Обучение модели (О)ДК состоит из трех итеративных шагов: оценивание структуры кодирующего дерева с фиксированными словарями и кодами, оценивание словарей при фиксированных кодах и структуре, нахождение оптимальных кодов обучающих векторов для данных словарей и структуры дерева. Решение каждого из шагов гарантированно оптимально, значит ошибка аппроксимации (2.3.1) гарантированно не возрастает от итерации к итерации. Покажем, что можно сформировать начальное приближение для (О)ДК словарей $(\{\tilde{i}_j^m\}_{j=1\ldots L}^{m=1\ldots M};\tilde{C}^1,\ldots,\tilde{C}^M;\{\tilde{e}_{(m,n)}\})$, которое приводит к такой же ошибке аппроксимации множества X, что и заданные (О)МК словари и коды. Тогда, если мы начнем обучающие итерации с этого начального приближения, итоговая ошибка аппроксимации гарантированно не будет превосходить изначальную.

Начнем строить начальное приближение для ДК, используя данные МК словари $C^1,\ldots,C^M$ и коды $\{i_j^m\}_{j=1\ldots L}^{m=1\ldots M}$. Начальные значения кодов обучающих точек положим равными соответствующим МК кодам.

$$\tilde{i}_j^m = i_j^m \quad j = 1..L, \quad m = 1..M$$

Определим начальную структуру дерева как цепочку, то есть:

$$\tilde{e}_{(m,n)} = 1 \iff n - m = 1.$$

В таком дереве размерности приписываются ребрам по следующему правилу:

$$\text{диапазон}\left(\frac{mD-D}{M},\frac{mD}{M}\right] \qquad \text{припишем к } \tilde{e}_{(m,m+1)},\ \ m = 1..M-1$$

$$\text{диапазон}\left(\frac{MD-D}{M},D\right] \qquad \text{припишем к } \tilde{e}_{(M-1,M)}$$

Наконец, построим элементы словарей $\tilde{C}^1,\ldots,\tilde{C}^M$. Заметим, что все m координат из диапазона $\left(\frac{mD-D}{M},\frac{mD}{M}\right]$ активны в словаре $\tilde{C}^m$. Это позволяет ини-

циализировать элементы $\tilde{C}^m$ значениями C^m по следующему правилу:

$$\tilde{c}^m(:)\left[\left(\frac{mD-D}{M}+1\right):\frac{mD}{M}\right]=c^m(:),\quad j=1..L$$
$$\tilde{c}^m(:)[d]=0,\quad \text{if}\quad d\notin\left[\left(\frac{mD-D}{M}+1\right),\frac{mD}{M}\right]$$

Теперь покажем, что начальное приближение, сформированное выше приводит к такой же ошибке аппроксимации множества X, как и с заданными МК словарями и кодами. Рассмотрим произвольную точку x_j и ее координату d. Без потери общности предположим, что $d\in\left[1,\frac{D}{M}\right]$, то есть d приписана к словарям $\tilde{C}^1$, $\tilde{C}^2$. Тогда вклад этих точки и координаты в ошибку таков:

$$r^j_{TQ}(d)=\left(\tilde{c}^1(i^1_j)[d]+\tilde{c}^2(i^2_j)[d]-x_j[d]\right)^2$$

По построению $\tilde{C}^2$ $\tilde{c}^2(i^2_j)[d]=0$, тогда:

$$r^j_{TQ}(d)=\left(\tilde{c}^1(i^1_j)[d]-x_j[d]\right)^2=$$
$$=\left(c^1(i^1_j)[d]-x_j[d]\right)^2=r^j_{PQ}(d)$$

Аналогичное рассуждение можно провести для остальный точек и координат. Так как ошибка аппроксимации является аддитивной функцией, можно сделать вывод, что ошибка аппроксимации ДК со словарями $\{\tilde{i}^m_j\}^{m=1...M}_{j=1...L}$ и $\tilde{C}^1,\ldots,\tilde{C}^M$ равна ошибке аппроксимации МК с кодами и словарями $\{i^m_j\}^{m=1...M}_{j=1...L}$ and $C^1,\ldots,C^M$.

Начальное приближение для модели ОДК формируется из ОМК словарей и кодов в точности тем же способом, что описан выше, с тем исключением, что матрица ортогонального преобразования инициализируется матрицей соответствующего преобразования ОМК.■

Хотя утверждение относится к кодированию обучающего множества, на практике метод не склонен к существенному переобучению, и (О)ДК существенно превосходит (О)МК в смысле точности кодирования тестового множества векторов.

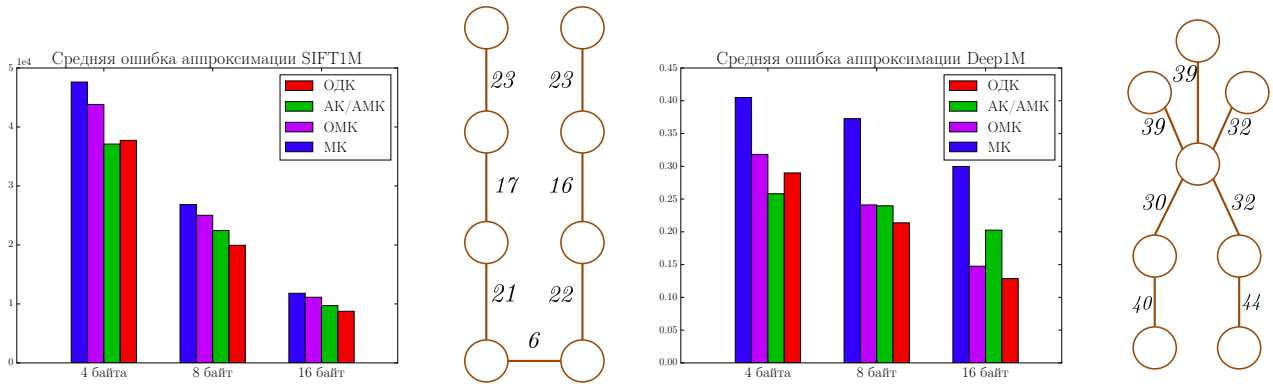


Рисунок 2.11 — Средняя ошибка аппроксимации коллекций SIFT1M (слева) и Deep1M с помощью различных методов сжатия и визуализация топологий кодирующих деревьев для ОДК в случае M=8. Для случая M=4 использовалась АК, для кодов большей длины использовалась АМК. ОДК достигает наименьшей ошибки аппроксимации для длинных кодов. На визуализациях деревьев каждое ребро помечено числом размерностей, приписанных к нему.

2.3.2 Эксперименты

В этой части оценивается качество модели ОДК для прикладных задач нахождения ближайшего соседа и классификации больших объемов данных. ОДК сравнивается с другими квантизационными подходами: МК, ОМК, АК. Так как АК становится слишком медленной для длинных кодов, используется АК для кодов длины 4 байта, а для кодов большей длины используется АМК, которая разбивает кодируемые векторы на несколько подвекторов и кодирует каждый подвектор 4 байтами с помощью АК.

Сначала методы сравниваются для **задачи поиска ближайшего соседа** на двух множествах векторов:

(I) SIFT1M: содержит один миллион 128-мерных СИФТ дескрипторов [29], а также отдельное множество из 100.000 векторов для обучения. Поисковая база содержит 10.000 запросов, для которых известны истинные ближайшие соседи.

(II) Deep1M: содержит нейросетевые дескрипторы естественных изображений, полученные с полносвязных слоев глубокой сверточной нейросети [26]. Дескрипторы L_2-нормализованы и сжаты методом главных компонент до раз-

мерности $D = 256$. Основное множество содержит один миллион векторов, а обучающее множество содержит 100,000 векторов. Также имеется 1000 запросов, для которых преподсчитаны истинные ближайшие соседи среди основного множества.

Все сравниваемые модели были обучены на обучающих множествах и были применены для сжатия основных множеств. Далее сравнивались (1) средние ошибки аппроксимации векторов основного множества и (2) качество нахождения ближайшего соседа по метрике полнота@T, которая равняется доле запросов, для которых истинный ближайший сосед оказался среди T ближйших сжатых векторов. Методы сравнивались для кодов длин M=4,8,16 байт.

Как можно видеть на рисунке 2.11, на обоих датасетах АК работает с наилучшим качеством с кодами длины 4 байта, но для более длинных кодов АМК теряет преимущество в качестве. В то же время ОДК превосходит другие методы в случае $M = 8,16$ и достигает сопоставимой точности с АК в случае $M = 4$. Интересно, что для разных датасетов топологии кодирующих деревьев в модели ОДК существенно различны, что видно на рисунке 2.11. Важно отметить, что ОДК преводсходит АМК несмотря на ограничения, наложенные на элементы словарей. Причина превосходства заключается в том, что кодирование в модели ОДК оптимально при данных словарях. Кодирование в модели АК (на основе лучевого поиска) является приближенным, что приводит к более высоким ошибкам аппроксимации для случаев $M > 4$, несмотря на большее количество обучаемых параметров.

Основное практическое преимущество модели ОДК перед моделью АК — это быстрое кодирование, особенно в случае больших значений M. Таблица 2.2 демонстрирует среднее время кодирования 128-мерного СИФТ дескриптора с помощью обоих моделей. Кодирование в модели ОДК требует в 17 раз меньше времени по сравнению с АМК и в 92 раза меньше времени по сравнению с АК.

Относительное качество всех методов с точки зрения ошибки аппроксимации является причиной более точного поиска ближайшего соседа (см. рисунок 2.12).

На этапе поиска скорость ОДК также превосходит скорость АК/АМК, как продемонстировано на рисунке 2.13. Причиной этого преимущества является то, что число членов в сумме (2.18) линейно зависит от M, в то время как в АК/АМК оно квадратично по M. В большинстве экспериментов поиск на

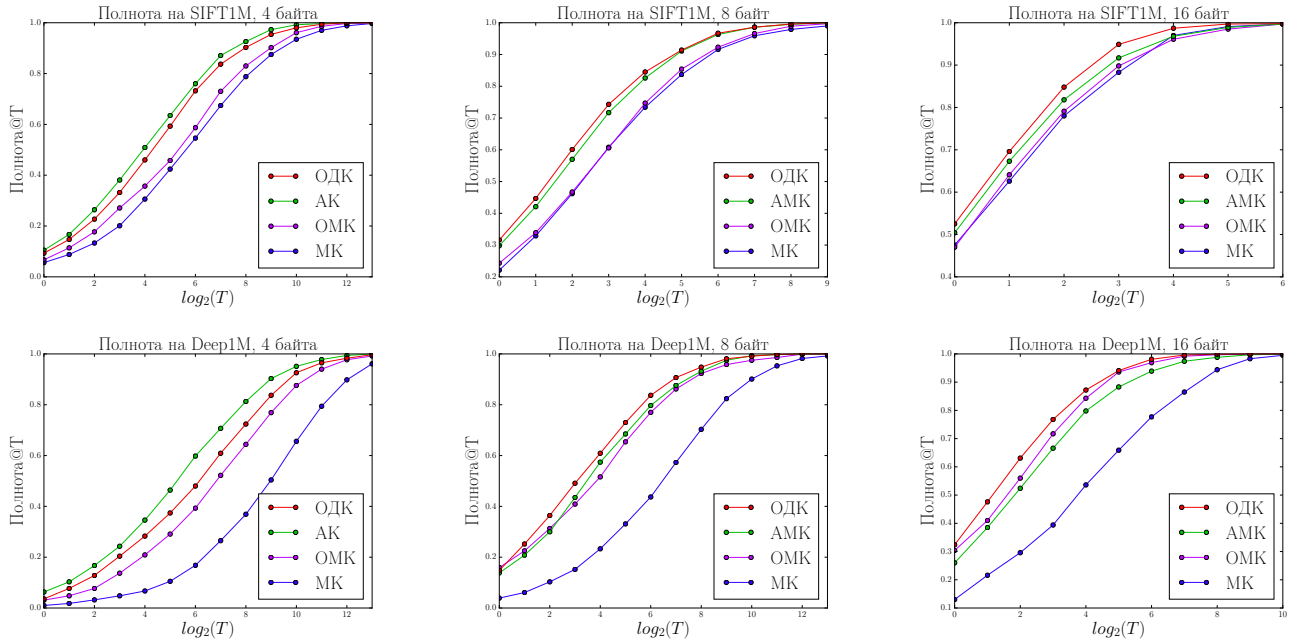


Рисунок 2.12 — Качество поиска ближайшего соседа с использованием различных моделей сжатия для датасетов SIFT1M и Deep1M и для кодов различной длины. Ось абсцисс демонстрирует количество блийших сжатых векторов T (в логарифмическом масштабе), а ось ординат — полноту@T, которая является эмпирической оценкой вероятности того, что найден истинный ближйший сосед. На обоих датасетах полнота@T, получаемая с использованием модели ОДК, существенно выше по сравнению с МК и ОМК. АК работает лучше ОДК для кодов длины 4, но для более длинных кодов качество ОДК становится выше.

	ОДК	АК	АМК
	$M = 4$		
Абсолютное время (мс)	2.9	52.6	52.6
Ускорение ОДК	—	**18x**	**18x**
	$M = 8$		
Абсолютное время (мс)	6.0	235.8	103.0
Ускорение ОДК	—	**39x**	**17x**
	$M = 16$		
Абсолютное время (мс)	12.3	1127.3	204.7
Ускорение ОДК	—	**92x**	**17x**

Таблица 2.2

Среднее время кодирования 128-мерного СИФТ дескриптора с помощью моделей ОДК, АК и АМК, а также ускорение, полученное благодаря использованию ОДК. Ускорение особенно значительно для больших значений M.

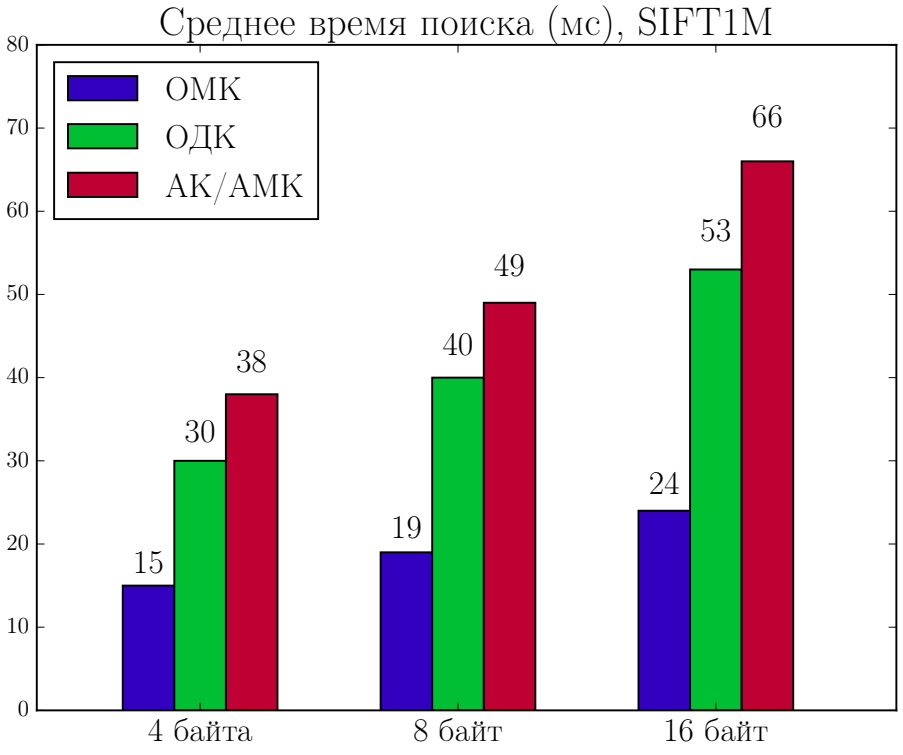


Рисунок 2.13 — Среднее время поиска ближайшего соседа полным перебором в датасете SIFT1M для моделей ОМК, ОДК и АК/АМК. АК использовалась для кодов длины 4 и АМК — для кодов длины 8 и 16. ОДК достигает более бытстрого поиска по сравнению с АК/АМК, так как сложность вычисления расстояния в ОДК линейна по M (см. главу 2.3.1).

	Ошибка аппроксимации			Средняя точность классификации		
	320x	640x	1280x	320x	640x	1280x
ОМК	0.711	0.846	0.957	0.464	0.389	0.306
АМК	0.548	0.637	0.737	0.490	0.459	0.431
ОДК	**0.521**	**0.607**	**0.637**	**0.497**	**0.467**	**0.441**

Таблица 2.3

Ошибка аппроксимации и средняя точность классификации изображений с Фишеровскими векторами в качестве дескрипторов. Обучение производилось на несжатых данных, тестирование проводилось на сжатых данных. Словари для ОМК и ОДК были обучены на обучающем множестве и использовались для кодирования тестового множества. Более точное кодирование с моделью ОДК приводит к более высокой точности классификации. Средняя точностью с использованием несжатых дескрипторов составляет 0.577.

основе ОДК оказывается приблизительно в два раза медленнее по сравнению с ОМК, в то же время достигается существенно более высокая точность.

Классификация. Эксперименты для задачи классификации проводились на коллекции PASCAL VOC 2007 [58]. Сравнивались ОМК и ОДК для сценария, в котором классификатор, обученный на несжатых дескрипторах, применяется к большому числу сжатых тестовых дескрипторов для того, чтобы найти изображения с максимальной уверенностью в классификации [55; 56]. В этом сценарии необходимо вычислить только скалярные произведения между запросов (классификатором) и сжатыми векторами.

В качестве дескрипторов были использованы Фишеровские векторы [61] с 256 компонентами, построенные на СИФТ дескрипторах, сжатых до размерности 80. Затем измерялись потери в средней точности классификации вследствие сжатия моделями ОМК и ОДК для различных степеней сжатия. Как и в секции выше, исходные Фишеровские векторы разбивались на R подвекторов и каждый подвектор сжимался до 8 байт моделями ОМК и ОДК. Различные степени сжатия соответствуют различным значениям R. В качестве метрики качества классификатора использовалась стандартная средняя точность. Словари для моделей ОМК и ОДК были обучены на тренировочном множестве.

Таблица 2.3 демонстрирует ошибку аппроксимации и точность классификации для трех степеней сжатия (320x, 640x, 1280x). Ошибка кодирования в модели ОДК гораздо ниже по сравнению с ОМК, что приводит к существенно

меньшим потерям в точности классификации. Преимущество особенно велико (разница в точности равна 0.135) для максимальной степени сжатия.

2.3.3 Выводы о модели Древесной квантизации

Как можно заметить из экспериментов, Древесная квантизация достигает более высокого качества кодирования по сравнению с существующими версиями мультиквантизации. С другой стороны, Древесная квантизация существенно обходит по скорости поиска и кодирования Аддитивную квантизацию. В целом, модель Древесной квантизации — практический компромисс между малой ошибкой сжатия и высокой скоростью работы. С теоретической точки зрения модель Древесной квантизации также более предпочтительна, так как гарантирует оптимальность кодирования для данных словарей, а также гарантированно не увеличивает ошибку моделей мультиквантизации.

2.3.4 Заключение

В этой главе были исследованы существующие методы компрессии векторов высокой размерности и было предложено два новых метода. Основным недостатком существующих подходов является неявное предположение о статистической независимости распределений подвекторов в различных подпространствах, которые не выполняется на практике. Поэтому основной целью автора была разработка методов, не имеющих этого ограничения. Были разработаны два новых метода: Аддитивная квантизация (АК) и Древесная квантизация (ДК). АК обеспечивает существенно меньшие потери при сжатии по сравнению с предшествующими методами, но обладает высокой вычислительной сложностью при больших длинах кода, поэтому применимость этого метода на практике ограничена небольшими кодами. ДК допускает эффективное кодирование при любых длинах кода и в то же время приводит к меньшим потерям при сжатии по сравнению с существующими подходами.

Глава 3. Эффективный поиск ближайших соседей

Современным поисковым системам необходимо искать по коллекциям из миллиардов изображений за несколько миллисекунд. В связи с этим подходы, используемые для нахождения ближайших соседей, должны быть масштабируемыми и эффективными. В этой главе исследуются различные методы эффективного поиска по коллекциям такого масштаба.

Глава состоит из трех частей. В первой части вводятся основные определения и приводится обзор существующих методов.

Во второй части описана новая структура данных для эффективного поиска: инвертированный мультииндекс (ИМИ). Приведены формальные алгоритмы построения инвертированного мультииндекса и поиска на его основе. Доказаны теоретические оценки сложности основных операций в мультииндексе. Экспериментально показано, что инвертированный мультииндекс способен с приемлемой точностью искать ближайших соседей в коллекции из миллиарда СИФТ векторов за несколько миллисекунд.

В третьей главе исследуется вопрос оптимальной структуры данных для эффективного поиска по миллиардам нейросетевых дескрипторов. Автор экспериментально показывает, что инвертированный мультииндекс не является оптимальным для такого типа данных, и предлагает новую структуру данных: неортогональный инвертированный мультииндекс (НО-ИМИ). Продемонстрировано, что для коллекции из миллиарда нейросетевых дескрипторов, предложенная структура НО-ИМИ обеспечивает наивысшую точность при заданном бюджете времени поиска.

3.1 Обзор существующих методов крупномасшабного поиска ближайших соседей

Современным поисковым системам [62] необходимо осуществлять поиск по коллекциям, содержащим миллиарды изображений, за несколько миллисекунд, чтобы успешно отвечать на пользовательские запросы. Это накладывает

серьезные требования к эффективности и масштабируемости алгоритмов нахождения ближайших соседей, используемых такими методами. На таких объемах полный перебор невозможен, поэтому приходится использовать приближенные методы, тем или иным образом ограничивающие часть коллекции, по которой будет осуществляться поиск соседей для конкретного запроса.

Передовые алгоритмы, способные работать с коллекциями из миллиардов векторов [19], избегают полного перебора путем использования *индексирующей структуры*. Для данного запроса индексирующая структура позволяет сформировать шорт-лист кандидатов, которые наиболее вероятно являются близкими к запросу. После получения шорт-листа векторы-кандидаты переранжируются на основе расстояний до запроса. Для таких методов время работы почти линейно зависит от размера шорт-листа. Поэтому критически важной является способность индексирующей структуры извлекать компактные шорт-листы, содержащие ближайших соседей с большой вероятностью. В этом случае переранжирование будет быстрым и эффективность метода в целом будет высокой.

Единственной существующей системой для поиска по миллиардным коллекциям является система IVFADC, предложенная в 2011 году [19] и основанная на идее инвертированного индекса. Эта система разбивает пространство поиска на непересекающиеся регионы, соответствующие регионам Вороного, полученным с помощью кластеризации методом K-средних. Центроиды этих клеток будем называть *словами*, а всю совокупность центроидов — *словарем*. Для каждого слова хранится список векторов коллекции, принадлежащих соответствующему региону Вороного. Во время поиска соседей для конкретного запроса определяется небольшое число ближайших слов и списки векторов, принадлежащих соответствующим регионам, конкатенируются, образуя шорт-лист кандидатов.

Однако, в случае миллиардных коллекций, скорость работы IVFADC (порядка 100 миллисекунд на запрос [19]) может быть недостаточной для многих приложений. Серьезное ограничение этой системы состоит в том, что количество регионов в разбиении пространства мало, поэтому количество векторов в каждом регионе довольно велико, что приводит к большим размерам шорт-листов. Увеличения количества регионов для более мелкого разбиения приведет к серьезному увеличению времени поиска ближайших слов, а также увеличению времени индексации. Конечно, во время поиска могут быть использованы

приближенные методы (например, kd-деревья [16]), но на практике это приводит к существенному снижению точности полученных шорт-листов, как будет показано ниже.

3.2 Инвертированный мультииндекс

В главе 3.1 упоминалось, что существующие структуры данных не способны эффективно решать задачу поиска ближайшего соседа для поисковых баз, содержащих миллиарды векторов. В этой части описывается новая структура данных — инвертированный мультииндекс, которая способна порождать очень мелкое разбиение пространства поиска на большое число регионов, тем самым позволяя находить ближайжих соседей среди миллиардов точек за несколько миллисекунд.

Фактически, инвертированный мультииндекс является обобщением стандартного инвертированного индекса, лежащего в основе IVFADC, в котором векторная квантизация заменена на мультиквантизацию [12]. Согласно идее мультиквантизации, инвертированный мультииндекс строится как многомерная таблица, элементы которой соответствуют всевозможным кортежам слов из словарей, построенных для различных подпространств. Эта многомерная таблица является обобщением одномерной таблицы, клетки которой соответствуют словам стандартного инвертированного индекса. По аналогии со стандартным инвертированным индексом, каждый элемент таблицы мультииндекса соответствует определенному региону пространства поиска и содержит список точек, попавших в этот регион. Важно отметить, что мультииндекс допускает простой и эффективный алгоритм получения последовательности регионов, оказавшихся ближайшими к данному вектору-запросу.

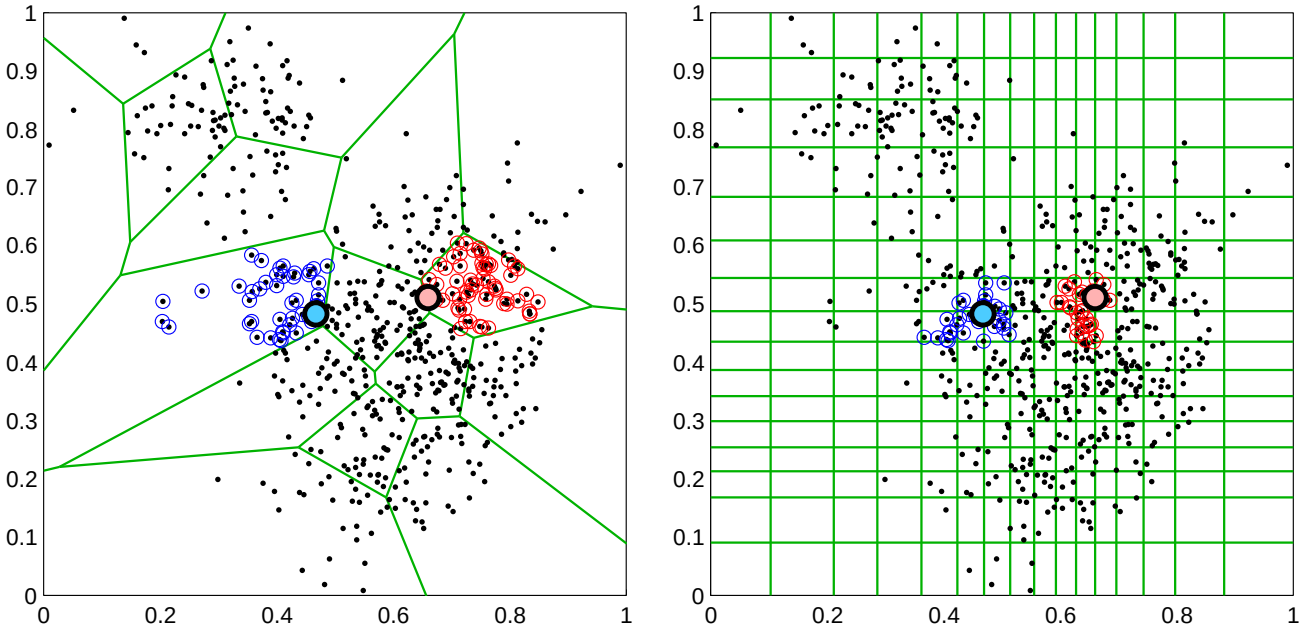


Рисунок 3.1 — Индексация множества из 600 точек (обозначены черным), распределенных неравномерно внутри двумерного квадрата со стороной единица. **Слева** – инвертированный индекс, основанный на стандартной квантизации (словарь имеет 16 двумерных слов; границы клеток обозначены зеленым). **Справа** – инвертированный мультииндекс, основанный на мультиквантизации (каждый из двух словарей имеет 16 одномерных слов). Число операций, необходимое для вычисления расстояний от запроса до всех слов, одинаковое для обеих индексирующих структур. Приведено два примера запросов, обозначенных синим и красным кругами. Списки кандидатов, образованные инвертированным индексом (слева) содержат 45 и 64 элементов соответственно (обведены кругом). Важно отметить, что когда запрос лежит около границы региона (что происходит очень часто в пространствах большой размерности), получившийся список кандидатов может не содержать многих близких точек. Также инвертированный индекс не имеет возможности возвращать список заранее установленной небольшой длины (например, 30 кандидатов). Для одних и тех же запросов, списки кандидатов как минимум длины 30 были сформированы инвертированных мультииндексом, и в этом случае списки содержали 31 и 32 элемента. Даже такие небольшие списки требуют обхода нескольких регионов из разбиения, и получившиеся списки кандидатов гораздо более надежны. В пространствах большой размерности способность обходить регионы, окружающие область запроса с разных направлений, приводит к значительному увеличению точности поиска ближайших соседей.

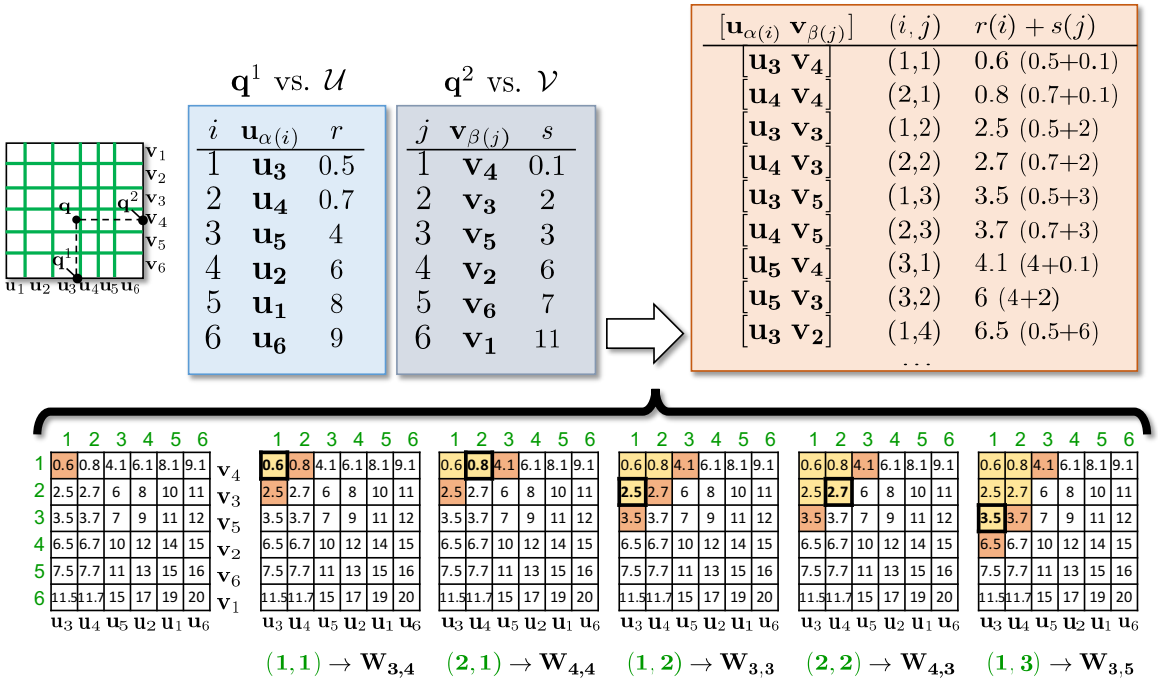


Рисунок 3.2 — **Сверху** – обработка запроса с использованием инвертированного мультииндекса. Сначала вычисляются расстояния от двух половин запроса q^1 и q^2 до словарей U и V соответственно, для того чтобы получить две последовательности слов, упорядоченных по возрастанию расстояний (обозначенных за r и s) до соответствующих половин запроса. Затем эти последовательности обрабатываются алгоритмом мультипоиска, результатом которого являются пары слов, упорядоченные по возрастанию расстояния от запроса. Списки точек, ассоциированные с этими парами, конкатенируются и образуют список кандидатов для этого запроса. **Снизу** – первые итерации алгоритма мультипоиска в этом примере. Красным обозначены пары, находящие в очереди с приоритетами, желтым — обработанные пары. Зеленые числа соответствуют индексам пары (i и j), а черным выделены слова $u_{\alpha(i)}$ и $v_{\beta(j)}$. Числа в клетках соответствуют расстояниям $r(i)+s(j) = d\left(q,[u_{\alpha(i)}\ v_{\beta(j)}]\right)$.

3.2.1 Описание модели инвертированного мультииндекса

Структура инвертированного мультииндекса. Опишем, как организован инвертированнный мультииндекс. Также сравним соответствующие детали инвертированного мультииндекса и стандартного инвертированного индекса.

Пусть $P = \{p_1,p_2,\ldots,p_N\}$, $p_i \in \mathbf{R}^D$ — множество из большого числа N D-мерных векторов. Тогда построение стандартного инвертированного индекса начинается с обучения словаря W из K D-мерных векторов $W = \{w_1,w_2,\ldots,w_K\}$ путем кластеризации алгоритмом к-средних. Затем исходное множество векторов разбивается на K списков $W_1,W_2,\ldots,W_K$, где каждый список W_i содержит все векторы, попавшие в соответствующий регион Вороного в $\mathbf{R}^D$, то есть $W_i = \{p \in P | i = \arg\min_j d(p,w_j)\}$. Здесь d — расстояние в $\mathbf{R}^D$. На практике каждый список W_i может быть представлен в памяти непрерывным массивом, в котором каждый элемент может содержать сжатую версию исходного вектора (что полезно при переранжировании) и некоторую метаинформацию об этом векторе (например, идентификатор изображения, которому принадлежит данный дескриптор).

Следуя идее мультиквантизации [12], инвертированный мультииндекс разбивает исходные M размерностей на несколько непересекающихся блоков. Число блоков влияет на скорость и точность поиска. В работах по кодированию самое лучшее качество достигалось с использованием 8 блоков [5;12;19]. Однако, в случае мультииндекса является оптимальным использовать всего два блока, по крайней мере для параметров задачи, рассматриваемой в экспериментах ниже и если точность и скорость поиска важнее, чем быстрое построение индекса. Более детально выбор числа блоков будет обсуждаться ниже. Для описания в этой части будет считать, что количество блоков равно двум, то есть исходные векторы разбиваются на две равные половины. Если необходимо, этот вариант будет называться инвертированным мультииндексом второго порядка. Обобщение предложенных алгоритмов на большее число блоков (мультииндексы более высоких порядков) тривиально.

Пусть $p_i = [p_i^1 \;\; p_i^2]$ — разбиение вектора $p_i \in \mathbf{R}^D$ на две половины, где $p_i^1 \in \mathbf{R}^{\frac{D}{2}}$,$\mathbf{p}_i^2 \in \mathbf{R}^{\frac{D}{2}}$. Как и в случае других алгоритмов, основанных на мультиквантизации, инвертированные мультииндексы работают лучше всего, когда

корреляции между $P^1 = \{p_i^1\}$ и $P^2 = \{p_i^2\}$ как можно меньше и дисперсии множеств P^1 и P^2 близки. Для СИФТ векторов разбиение на половины исходного вектора на практике является оптимальным, а в других случаях может понадобиться перегруппировка координат или применение случайного ортогонального преобразования для балансировки дисперсии между половинами [5; 12].

Словари мультиквантизации для инвертированного мультииндекса получаются путем независимых кластеризаций методом к-средних множеств P^1 и P^2, обозначим за $U = \{u_1, u_2, \ldots, u_K\}$ словарь для первых половин и за $V = \{v_1, v_2, \ldots, v_K\}$ — для вторых половин. Затем осуществляется мультиквантизация всех векторов из имеющегося множества, в результате чего образуются K^2 списков, соответствующих всевозможным парам слов $(u_i,v_j), i = 1 \ldots K, j = 1 \ldots K$. Обозначим каждый из этих списков как W_{ij}. Каждая точка $p = [p^1\ p^2]$ приписана блиайшему центроиду вида $[u_i\ v_j]$, так что:

$$W_{ij} = \{p = [p^1\ p^2] \in P \mid i = \arg\min_k d_1(p^1, \mathbf{u}_k) \wedge j = \arg\min_k d_2(p^2, v_k)\}\ . \tag{3.1}$$

Важно отметить, что каждый лист W_{ij} соответствует декартову произведению двух клеток Вороного в пространствах $\mathbf{R}^{\frac{D}{2}}$. В выражении (3.1), расстояния d_1 и d_2 в $\mathbf{R}^{\frac{D}{2}}$ индуцированы d так, что $\forall a,b:\ d(a,b) = d_1(a^1,b^1) + d_2(a^2,b^2)$. Простейшим и самым важным примером является случай, когда d, d_1, и d_2 — квадраты евклидовых норм в соответствующих пространствах, в этом случае полученный мультииндекс может быть использован для нахождения точек, близких к запросу в смысле евклидового расстояния.

Поиск с инвертированным мультииндексом. Для запроса $q = [q^1\ q^2] \in \mathbf{R}^{\frac{D}{2}}$ и требуемой длины списка кандидатов $T \ll N$ инвертированный мультииндекс может построить шорт-лист из T точек из P, которые с высокой вероятностью будут близки к q в смысле расстояния d. Это достигается нахождением достаточного числа пар слов $[u_i\ v_j]$, близких к q в $\mathbf{R}^D$ и конкатенацией их листов W_{ij}. Нахождение слов $[u_i\ v_j]$ осуществляется в два этапа (см. 3.2).

На первом этапе находятся расстояния от половин запроса q^1 и q^2 до слов соответствующих словарей. Затем для q^1 и q^2 находится L ближайших слов среди U and V соответственно (где $L < K$ зависит от T). Так как размеры U и V обычно невелики (тысячи векторов), этот этап осуществляется полным

перебором. Обозначим за $\alpha(k)$ индекс k-ого ближайшего соседа q^1 в U (то есть $u_{\alpha(1)}$ — ближайший сосед q^1 в U, $u_{\alpha(2)}$ — второй ближайший, и т.д.). Аналогично обозначим за $\beta(k)$ индекс k-ого ближайшего соседа q^2 в V. Также обозначим за $r(k)$ и $s(k)$ расстояния от q^1 и q^2 до $u_{\alpha(k)}$ и $\mathbf{v}_{\beta(K)}$ соответственно, то есть $r(k) = d_1\left(q^1,u_{\alpha(k)}\right)$ и $s(k) = d_2\left(q^2,v_{\beta(k)}\right)$.

На втором этапе, имея две монотонно возрастающие последовательности $r(1),r(2),\ldots,r(L)$ и $s(1),s(2),\ldots,s(L)$, необходимо обойти множество пар $\{(r(i),s(j)) \mid i = 1\ldots L, j = 1\ldots L\}$ в порядке увеличения суммы $r(i)+s(j)$ (что равно $d(q,[u_{\alpha(i)}\ v_{\beta(j)}])$). Тогда центроиды $[u_{\alpha(i)}\ v_{\beta(j)}]$ будут посещены в порядке увеличения расстония от запроса q. Обход начинается с пары (1,1), соответствующей региону с центроидом $[u_{\alpha(1)}\ v_{\beta(1)}]$, в которую попал запрос. В процессе обхода, списки $W_{\alpha(i)\,\beta(j)}$ конкатенируются до тех пор, пока длина итогового списка не превосходит требуемую величину T, после достижения которой обход останавливается.

Для осуществления такого обхода предлагается *алгоритм мультипоиска* (см. рисунки 3.2, 3.3). Этот алгоритм основан на очереди с приоритетами, хранящей пары индексов (i,j), где приоритет каждой пары определяется как $-\left(r(i)+s(j)\right) = -d\left(q,[u_{\alpha(i)}\ v_{\beta(j)}]\right)$. Очередь инициализируется парой (1,1). На каждом следующем шаге t из очереди извлекается пара (i_t,j_t) с максимальным приоритетом (минимальным расстоянием до q), после чего эта пара считается обработанной (ассоциированный список точек $W_{\alpha(i)\,\beta(j)}$ добавляется к результирующему шорт-листу). Затем пары (i_t+1,j_t) и (i_t,j_t+1) пытаются попасть в очередь. Пара (i_t+1,j_t) добавляется в очередь, если вторая предшествующая ей пара (i_t+1,j_t-1) также уже является обработанной (или если j_t=1). Аналогично, пара (i_t,j_t+1) добавляется в очередь, если вторая предшествующая ей пара (i_t-1,j_t+1) уже является обработанной (или если i_t=1). Основная идея алгоритма заключается в том, что каждая пара добавляется в очередь только один раз, когда обе предшествующие ей пары обработаны.

Результатом алгоритма мультипоиска является последовательность пар (i,j), чьи списки $W_{i,j}$ конкатенируются в итоговой шорт-лист кандидатов. Можно доказать корректность данного алгоритма:

Утверждение 1 (корректность): Результатом алгоритма мультипоиска является последовательность пар, упорядоченная по увеличению величины

$r(i)+s(i)$, причем любая пара из $\{1\dots L\}\otimes\{1\dots L\}$ встречается в этой последовательности.

Доказательство: Докажем Утверждение 1 в два этапа. Во-первых, докажем, что любая пара из $\{1\dots L\}\otimes\{1\dots L\}$ встречается в результирующей последовательности (*полнота* – при условии, что требуется достаточно много кандидатов), и, во-вторых, докажем, что все пары упорядочены по возрастанию расстояния (*монотонность*).

Полнота: Докажем утверждение о полноте индукцией по величине суммы $(i+j)$ для фиксированного значения L. База индукции, то есть случай $i+j=2$, тривиальна, так как пара (1,1) всегда обрабатывается на первом шаге. Предположим, что все пары $(i,j), i+j\le k$ обработаны. Рассмотрим пару (i,j), такую что $i+j=k+1$. Обе пары, предшествующие ей, $(i-1,j)$ и $(i,j-1)$ будут помещены в очередь и извлечены из нее по предположению индукции. Тогда после обработки обеих предшествующих пар, пара (i,j) также будет помещена в очередь. Таким образом, шаг индукции доказан и утверждение о полноте корректно.

Монотонность: Давайте покажем, что для любых двух пар (i_1,j_1), (i_2,j_2), таких что (i_2,j_2) была обработана сразу же после (i_1,j_1), выполняется свойство монотонности, то есть $r(i_1)+s(j_1)\le r(i_2)+s(j_2)$.

Предположим обратно, то есть $r(i_1)+s(j_1)>r(i_2)+s(j_2)$. Это означает, что (i_2,j_2) была помещена в очередь после того, как (i_1,j_1) была обработана (иначе из очереди была бы извлечена пара (i_2,j_2)). Однако, после обработки (i_1,j_1) алгоритм может поместить в очередь только пары (i_1+1,j_1) или (i_1,j_1+1). Так как $r(i+1)\ge r(i)$ и $s(i+1)\ge s(i)$, изначальное предположение $r(i_1)+s(j_1)>r(i_2)+s(j_2)$ не может быть выполнено ни в одном из этих двух случаев. ■

Касательно эффективности алгоритма, можно доказать, что размер очереди с приоритетами растет достаточно медленно:

Утверждение 2: на t-ом шаге алгоритма, когда обработано t пар, размер очереди с приоритетами не превышает величину $0.5+\sqrt{2t+0.25}$.

Доказательство: Оценим минимальное число пар, которое должен обработать алгоритм мультипоиска, чтобы очередь с приоритетами достигла размера q. Рассмотрим число обработанных пар в каждой строке (обозначим его за n_i). Несложно увидеть, что 1) n_i монотонно не возрастает; 2) в очереди находится не более одной пары из каждой строки; 3) если $n_i=n_{i+1}$, то в очереди

Algorithm 3.2.1: Алгоритм мультипоиска()

```
INPUT :   r(:), s(:)   % Две последовательности
OUTPUT :  out(:)   % Последовательность пар индексов
% Инициализация:
out ← ∅
traversed (1:length(r), 1:length(s)) ← false
pqueue ← new PriorityQueue
pqueue.push ( (1,1) ,r(1)+s(1))
% Traversal:
repeat
 ((i,j),d) ← pqueue.pop()
 traversed(i,j) ← true
 out ← out ∪ {(i,j)}
 if i < length(r) and  (j=1 or traversed(i+1,j−1))
   then pqueue.push ( (i+1,j) ,r(i+1)+s(j))
 if j < length(s) and  (i=1 or traversed(i−1,j+1))
   then pqueue.push ( (i,j+1) ,r(i)+s(j+1))
until (enough traversed)
```

Рисунок 3.3 — Псевдокод алгоритма мультипоиска. Итерации алгоритма заканчиваются после того, как получен шорт-лист кандидатов требуемой длины. Обобщение алгоритма на большее число входных последовательностей (что соответствует случаю мультииндексов высших порядков) тривиально.

нет пар из строки $i+1$. Все три утверждения следуют из того, что каждая пара добавляется в очередь строго после того, как обработаны предшествующие ей пары, то есть все пары с обеими индексами меньшими либо равными данному.

Таким образом, чтобы в очереди оказалось q пар, необходимо, чтобы в таблице было как минимум $q-1$ непустых строк с суммарным числом обработанных пар равным $1+2+\cdots+(q-1)=\frac{q(q-1)}{2}$. То есть $\frac{q(q-1)}{2} \le t$, где t — число шагов (=число обработанных пар. Решение данного неравенства приводит к доказываемой оценке.■

Сравнение инвертированного индекса и инвертированного мультииндекса.

Обсудим относительную эффективность обеих индексирующих структур при одинаковых размерах словарей K. В этом случае разбиения пространства с помощью индекса и мультииндекса существенно разнятся (см. 3.1). В частности, стандартный индекс поддерживает K списков, соответствующих разбие-

нию пространства поиска на K регионов, в то время как мультииндекс поддерживает K^2 списков, соответствующих гораздо более мелкому разбиению пространства. Длины списков в инвертированном индексе более менее сбалансированы (как следствие алгоритма к-средних), а распределение длин списков в мультииндексе очень неравномерно. Например, имеется большое число пустых списков, соответствующих u_i и v_j, которые никогда совместно не встречаются (регионы в правом нижнем углу на рисунке 3.1). Тем не менее, несмотря на неравномерное распределение длин списков, инвертированные мультииндексы выигрывают за счет более мелкого разбиения.

Более того, несмотря на увеличение мелкости разбиения в мультииндексе, вычисление расстояния от запроса до элементов словарей требует одинакового числа операций. Так, в инвертированном индексе необходимо вычислить K расстояний между D-мерными векторами, то в случае мультииндекса нужно вычислить $2K$ расстояний между $D/2$-мерными векторами. Обработка запроса в мультииндексе требует дополнительных вычислительных затрат на работу алгоритма мультипоиска. Но на практике, эти затраты оказываются малыми по сравнению со сложностью переранжирования кандидатов.

Использование мультииндекса также приводит к дополнительным затратам по памяти, так как необходимо поддерживать K^2 списков вместо K. Однако, общая длина списков остается прежней (так как суммарная длина равна числу векторов в поисковой базе, то есть N). Таким образом, если все списки хранятся одним непрерывным массивом, поддержание одного списка требует всего одной целочисленной переменной (хранящей позицию начала списка в массиве). Например, для значений $N = 10^9$ и $K = 2^{14}$, дополнительные затраты составляют около байта на вектор (4 байта$^*K/N$). Такие затраты обычно малы по сравнению с размером хранящейся метинформации или сжатого вектора.

Если говорить о мультииндексах более высоких порядков, которые разбивают векторы на большее число блоков, дополнительные затраты по памяти в этом случае могут быть очень большими, так как число регионов в разбиении очень быстро увеличивается с ростом K, также как и количество пустых регионов в разбиении пространства. Из-за этого ограничения использование мультииндексов более высоких порядков возможно только для малых значений K, при которых точность поиска недостаточна. В целом, мультииндекс

второго порядка оказывается оптимальным компромиссом между инвертированным индексом (малые затраты по памяти, дорогое вычисление расстояний до элементов словарей) и мультииндексов высших порядков (большие затраты по памяти, быстрое вычисление расстояний).

3.2.2 Приближенный поиск ближайшего соседа на основе инвертированного мультииндекса

Самым важным практическим применением инвертированного мультииндекса является приближенный поиск ближайшего соседа в поисковых база большого объема. Действительно, мультииндекс является индексирующей структурой, которая для заданного запроса может возвращать список кандидатов, которые с высокой вероятностью близки к нему. Затем эти кандидаты переранжируются на основе некоторой информации, хранящейся с каждым вектором. По аналогии с работой [12], рассмотрим два варианта кодирования информации о каждом векторе.

Во-первых, можно хранить код каждого вектора, полученный с помощью метода мультиквантизации (этот вариант называется Мульти-АВР). В случае с Мульти-АВР мультииндекс используется для того, чтобы формировать список кандидатов, каждый из которых является вектором, сжатым с помощью мультиквантизации. Во время переранжирования используется стандартная процедура асимметричного вычисления расстояния между несжатым запросом и кодом сжатого вектора.

Во-вторых, можно кодировать мультиквантизацией вектор разности между исходным вектором и ближайшим центроидом. Этот вариант называется Мульти-О-АВР и является обобщением системы IVFADC, предложенной в работе [12]. Таким образом, система Мульти-О-АВР второго порядка включает два словаря для индексации U, V, которые полностью определяют структуру мультииндекса. Также Мульти-О-АВР включает набор словарей для переранжирования $R_1, \ldots, R_M$, которые используются для сжатия векторов разностей между точками поисковой базы и ближайших центроидов. Предполагается, что

словари $R_1, \ldots, R_{\frac{M}{2}}$ используются для кодирования первой половины векторов разностей, а словари $R_{\frac{M}{2}+1}, \ldots, R_M$ — для кодирования второй половины.

Как правило, при одном и том же числе байт для переранжирования Мульти-О-АВР достигает большей точности поиска, чем Мульти-АВР, так как векторы разностей меньше по норме, чем исходные векторы, поэтому их проще закодировать с меньшими потерями.

В системе IFVADC, переранжирование кандидатов, сжатых мультиквантизацией, можно реализовать очень эффективно. В каждой обойденном регионе вычисляется вектор разности и преподсчитываются таблицы поиска для процедуры асимметричного вычисления расстояния [12], что позволяет быстро оценить расстояния между вектором разности для запроса и векторами разностей точек из поисковой базы, попавших в этот регион. В случае Мульти-О-АВР такой подход не является эффективным, так как в каждом регионе содержится всего несколько точек и затраты на преподсчет таблиц поиска не оправданы. Опишем, как можно вычислять расстояния в системе Мульти-О-АВР более эффективно.

Рассмотрим случай системы Мульти-О-АВР и евклидова расстояния между запросом $q \in \mathbf{R}^D$ и точкой поисковой базы , принадлежащей региону W_{ij} с центроидом $[u_i, v_j]$. Пусть разность вектора p и центроида закодирована мультиквантизацией кодом $[r_1, \ldots, r_M]$. С таким кодом p фактически аппроксимируется суммой:

$$p \approx \begin{pmatrix} u_i \\ v_j \end{pmatrix} + \begin{pmatrix} r_1 \\ \vdots \\ r_M \end{pmatrix} \tag{3.2}$$

Тогда расстояние от запроса до точки может быть аппроксимировано с использованием (3.2):

$$\|q-p\|^2 \approx \left\| q - \begin{pmatrix} u_i \\ v_j \end{pmatrix} - \begin{pmatrix} r_1 \\ \vdots \\ r_M \end{pmatrix} \right\|^2 = \tag{3.3}$$

$$||q||^2 - 2\left\langle q, \begin{pmatrix} u_i \\ v_j \end{pmatrix} \right\rangle - 2\left\langle q, \begin{pmatrix} r_1 \\ \vdots \\ r_M \end{pmatrix} \right\rangle + \left\| \begin{pmatrix} u_i \\ v_j \end{pmatrix} + \begin{pmatrix} r_1 \\ \vdots \\ r_M \end{pmatrix} \right\|^2$$

Как обычно, можно преподсчитать скалярные произведения подвекторов запроса со словами из словарей для индексации U, V и словарей для переранжирования $R_1, \ldots, R_M$. Преподсчитанные произведения хранятся в таблицам поиска и переиспользуются в каждом регионе при вычислении членов $\left\langle q, \begin{pmatrix} u_i \\ v_j \end{pmatrix} \right\rangle$ и $\left\langle q, \begin{pmatrix} r_1 \\ \vdots \\ r_M \end{pmatrix} \right\rangle$. Так как скалярные произведения преподсчитаны, вычисление этих членов для каждого кандидата требует $O(M)$ операций.

Также заметим, что члены $\left\| \begin{pmatrix} u_i \\ v_j \end{pmatrix} + \begin{pmatrix} r_1 \\ \vdots \\ r_M \end{pmatrix} \right\|^2$ не зависят от запроса. Их можно преподсчитать заранее и также хранить в таблицах поиска. Благодаря свойствам мультиквантизации этот член также можно упростить:

$$\left\| \begin{pmatrix} u_i \\ v_j \end{pmatrix} + \begin{pmatrix} r_1 \\ \vdots \\ r_M \end{pmatrix} \right\|^2 = \left\| u_i + \begin{pmatrix} r_1 \\ \vdots \\ r_{\frac{M}{2}} \end{pmatrix} \right\|^2 + \left\| v_j + \begin{pmatrix} r_{\frac{M}{2}+1} \\ \vdots \\ r_M \end{pmatrix} \right\|^2 = \tag{3.4}$$

$$\|u_i\|^2 + \|v_j\|^2 + \sum_{k=1}^{M} \|r_k\|^2 + 2\sum_{k=1}^{\frac{M}{2}} \langle u_i, r_k \rangle + 2 \sum_{k=\frac{M}{2}+1}^{M} \langle v_j, r_k \rangle$$

Таким образом, достаточно преподсчитать и хранить квадраты норм слов и скалярные произведения слов для индексации и для переранжирования. Имея эти значения, вычисление члена, не зависящего от запроса, также требует $O(M)$

операций. В результате все члены в выражении (3.3) могут быть вычислены за $O(M)$ операций.

3.2.3 Эксперименты

Основная цель экспериментов — подтвердить эффективность инвертированного мультииндекса и схемы Мульти-О-АВР для приближенного поиска ближайшего соседа. Большинство экспериментов сравнивают мультииндекс со стандартным инвертированным индексом и схемой IVFADC — единственной схемой, способной работать с миллиардами векторов. Также сравниваются различные варианты мультииндексов, включая мультииндексы второго и четвертого порядков, комбинацию мультининдекса с методом главных компонент, модификацию схемы Мульти-О-АВР, использующую локальные словари для переранжирования.

Эксперименты можно разделить на три группы согласно задачам, которые решаются в каждой из групп:

1. *Индексация.* Здесь сравниваются различные индексирующие структуры на множествах из миллиарда векторов. Структуры сравниваются на основе их способности эффективно формировать списки кандидатов для данного запроса с высокой полнотой.
2. *Эффективный приближенный поиск ближайшего соседа.* Здесь сравниваются различные системы поиска ближайшего соседа, включающиеся в себя индексирующую структуру и этап переранжирования (IVFADC, Мульти-АВР, Мульти-О-АВР).
3. *Детектирование изображений-полудубликатов.* В этой группе экспериментов различные методы сравниваются на основе их способности находить полудубликаты с использованием ГИСТ дескрипторов изображений.

В экспериментах использовались следующие датасеты:

BIGANN: Этот датасет, использующийся в большинстве экспериментов, был предложен в работе [19] и содержит миллиард 128-мерных СИФТ дескрипторов [29], извлеченных из естественных изображений. Датасет содержит так-

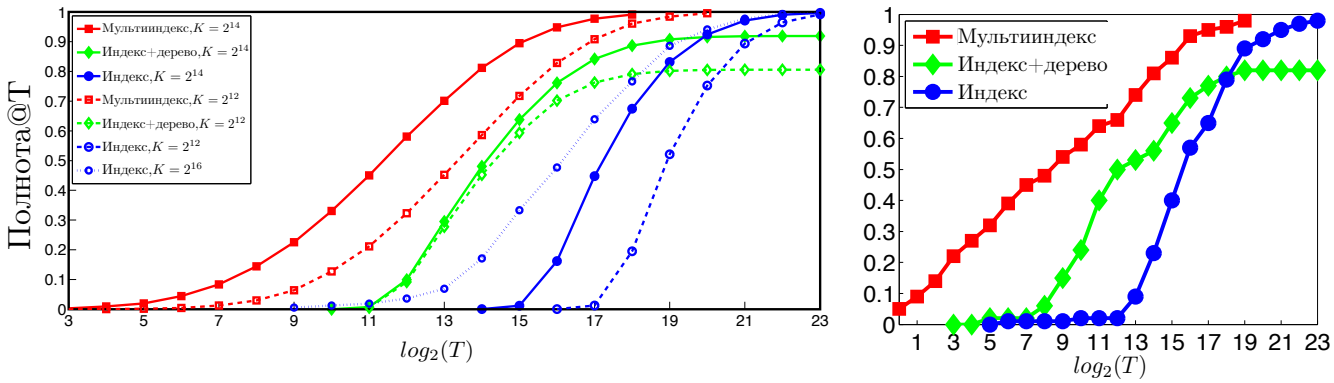
же 10000 запросов, для которых преподсчитаны истинные ближайшие соседи смысле евклидового расстояния.

Tiny Images: Этот датасет содержит 80 миллионов 384-мерных ГИСТ дескрипторов [52], соответствующих изображениям из датасета *Tiny Images* [63]. В этом датасете было вручную отобрано множество из 100 запросов и преподсчитаны истинные ближайшие соседи.

Copydays: Этот датасет был предложен в [64] для оценивания устойчивости дескрипторов к различным искусственным трансформациям. Датасет содержит 157 оригинальных изображений и их трансформации (результаты JPEG сжатия, кроппинга и других). Для каждого из оригинальных изображений четыре самые близкие изображения (10% и 20% кропы, а также 75-ое и 50-ое JPEG качество) считались его дубликатами. ГИСТ дескрипторы этих изображений были вычислени и добавлены к 80 миллионам ГИСТ векторов из датасета Tiny Images, таким образом образовав датасет для задачи поиска дубликатов в больших множествах векторов.

Качество индексации

В этих экспериментах исследуется качество листов кандидатов, полученных с помощью мультииндекса. Измеряются длины листов, времена их получения и *полнота*, то есть доля запросов, для которых истинный ближайший сосед оказался в списке кандидатов. На практике часто является качество нахождения нескольких ближайших соседей, однако эта метрика сильно коррелирует с качеством нахождения ближайшего соседа, поэтому не измеряется. Все времена получены без использования многопоточности на машине Intel Xeon 2.40 GHz CPU с использованием BLAS инструкций.



1 миллиард СИФТ векторов 80 миллионов ГИСТ векторов

Рисунок 3.4 — Полнота как функция длины списка кандидатов. Для одинаковых размеров словарей K сравниваются три системы со схожими вычислительными сложностями поиска и построения: инвертированный индекс с K словами, инвертированный индекс с большим словарем (2^{18} слов) и приближенным поиском ближайших слов kd-деревом с K сравнениями, инвертированный мультииндекс второго порядка со словарями размера K. Также представлены результаты для инвертированного индекса с $K = 2^{16}$ словарями, требующим гораздо больше времени для вычисления расстояний до элементов словаря. Во всех экспериментах мультииндекс достигает лучшей точности с более короткими списками кандидатов.

Качество списков кандидатов

Чтобы оценить качество списка кандидатов сравниваются значения полноты, достигаемые инвертированным мультииндексом второго порядка и инвертированным индексом с одинаковыми размерами словарей K. В экспериментах $K = 2^{14}$ для датасетов BIGANN и Tiny Images, а также дополнительно $K = 2^{12}$ для датасета BIGANN. Для множества различных длин списков T (степени двойки) и для каждого запроса обе структуры данных формируют списки кандидатов необходимой длины. Рисунок 3.4 демонстрирует полноту полученных списков в зависимости от длины T (используется обозначение полнота@T). Для фиксированного K преимущество мультииндексов над индексами велико для любых длин списков кандидатов. Для датасета BIGANN также продемонстрировано качество инвертированного индекса с большим словарем $K = 2^{16}$. Даже со словарем гораздо большего размера инвертированный индекс все равно проигрывает мультииндексу на любых значениях T.

Также в сравнении участвует еще один метод. Этот метод использует инвертированный индекс со словарем размера 2^{18}, но использует приближенный поиск ближайшего центроида во время запроса на основе kd-деревьев [16]. Для честного сравнения поиск в kd-дереве был ограничен K (2^{14} или 2^{12}) операциями вычисления расстояний. Как можно видеть на рисунке 3.4, предложенный метод неплохо работает в области низкой полноты, но работает гораздо хуже в областях высокой полноты. В целом, полнота@T обоих методов-конкурентов равномерно хуже, чем полнота@T инвертированного мультииндекса. И kd-деревья, и мультииндексы требуют некоторое количество дополнительных операций по сравнению со стандартными индексами и в следующем эксперименте показывается, что эти затраты несущественны.

Скорость поиска

В этом эксперименте измерялись времена работы инвертированного мультииндекса второго порядка ($K = 2^{12}, K = 2^{14}$) на датасете BIGANN для раз-

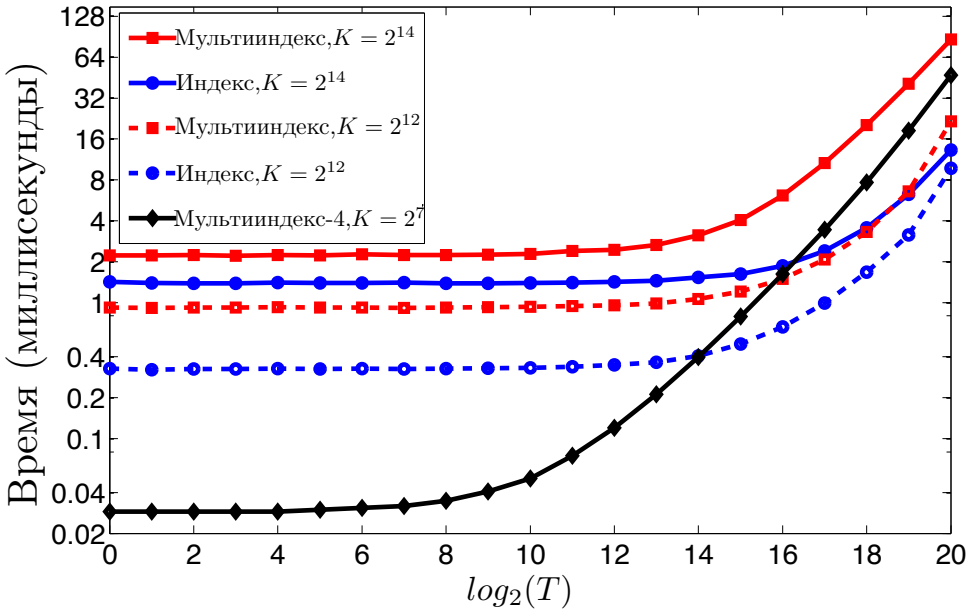


Рисунок 3.5 — Время (в миллисекундах), необходимое для получения списка кандидатов заданной длины с помощью мультииндекса и индекса на датасете BIGANN.

личных длин списков кандидатов. Результаты приведены на рисунке 3.5. Время поиска на основе мультииндекса значительно не меняется до тех пор, пока требуемое количество кандидатов не достигает нескольких тысяч, что подтверждает тот факт, что вычислительные затраты алгоритма мультипоиска незначительны по сравнению со стоимостью вычисления расстояний до слов. Также на рисунке 3.5 приведены времена работы для инвертированного индекса с $K = 2^{12}$,2^{14}. Их двукратное преимущество над мультииндексом второго порядка объясняется высокой эффективностью BLAS операций. Эта эффективность позволяет вычислять расстояния быстрее в случае стандартного индекса, несмотря на одинаковое число скалярных операций.

В совокупности рисунки 3.4 и 3.5 демонстрируют практическое преимущество мультииндекса второго порядка надо стадартным инвертированным индексом. Например, мультииндекс с $K = 2^{12}$ быстрее достигает более высокой точности, чем индекс с $K = 2^{14}$, даже с использованием BLAS инструкций. На рисунке 3.5 также изображено время работы мультииндекса четвертого порядка для малого K. Для малого числа кандидатов запрос гораздо быстрее, однако для большого числа кандидатов дополнительные вычисления в алгоритме мультипоиска становятся значительными, что и является основным практическим недостатком инвертированных индексов высших порядков.

Мультииндекс с методом главных компонент

Как обсуждалось выше, основное время работ инвертированного мультииндекса уходит на вычисление расстояний между вектором-запросом и словами. Конечно, можно уменьшить размер словаря, но требуемое время все равно останется линейно по размерности пространства M. Поэтому, было бы разумно скомбинировать мультииндекс с методом снижения размерности, например, с методом главных компонент (МГК).

Ниже описаны эксперименты, проведенные на датасете BIGANN, с мультииндексом второго порядка, $K = 2^{14}$ и четырехкратным снижением размерности, которое заменяет исходные 128-мерные СИФТ дескрипторы на 32-мерные векторы.

Можно предложить два способа скомбинировать МГК с инвертированным мультииндексом, эффективность которых сильно отличается.

Наивный подход. Это самая очевидная стратегия. Сначала исходные 128-мерные векторы сжимаются МГК до размерности 32. Чтобы дисперсии обеих половин полученного вектора были одинаковы, полученные векторы домножаются на случайную ортогональную матрицу. Затем на полученных векторах строится мультииндекс.

Продвинутый подход. Более эффективная стратегия применяет МГК с учетом того, что данные будут разбиты на половины. Поэтому применяется два независимых МГК сжатия к 64-мерным половинам исходных векторов, причем каждое множество половин сжимается до размерности 16. Инвертированный мультииндекс строится на конкатенациях пар полученных 16-мерных векторов.

Качество индексации для обоих подходов представлено на рисунке 3.6 в виде кривых полнота@T. Снижение полноты значительно ниже для продвинутого подхода комбинирования мультииндекса и МГК так как процесс формирования сжатых векторов учитывает независимость между половинами. Фактически, падение полноты в продвинутом подходе довольно незначительно, и им можно пренебречь во многих приложениях, что означает, что сжатие МГК не оказывает существенного влияния на качество списков кандидатов в случае инвертированного мультииндекса второго порядка.

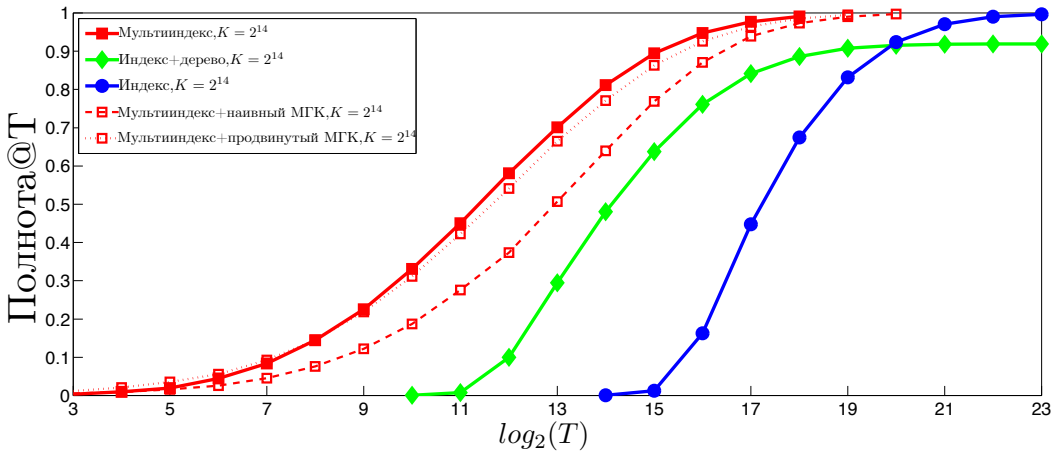


Рисунок 3.6 — Полнота как функция длины списка кандидатов по аналогии с рисунком 3.4 с добавленными кривыми для различных комбинаций мультииндекса и МГК. Даже со сжатием МГК качество списков кандидатов, полученных мультииндексом, выше , чем у других систем. Также важно отметить, что качество кандидатов продвинутого подхода существенно превосходит таковое у наивного подхода.

Приближенный поиск ближайшего соседа

Целью экспериментов в этой части является оценить качество поиска ближайших соседей системами Мульти-ABP и Мульти-O-ABP (основанных на инвертированном мультииндексе второго порядка).

Система Мульти-ABP

В первом эксперименте измеряется качество системы Мульти-ABP с $m = 8$ дополнительными байтами на вектор для переранжирования (каждый вектор делится на 8 подвекторов и используется мультиквантизация со словарями размера 256). Рисунок 3.7 демонстрирует значения метрики полнота@T^* для $T^* = 1{,}10{,}100{,}1000{,}10000$ (различные кривые) на датасете BIGANN как функции размера списка кандидатов, полученного мультииндексом. Мультииндекс сравнивается с системой, которая переранжирует весь датасет ($T = 10^9$), что

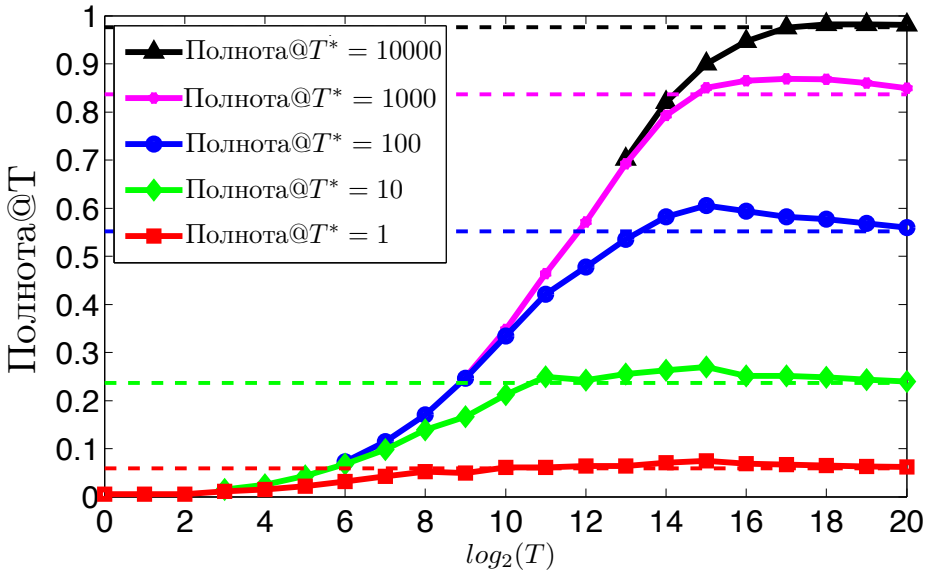


Рисунок 3.7 — Полнота@T^* ($T^* = 1$ до 10000) для системы Мульти-ABP (использующей 8 байт для переранжирования) на датасете BIGANN. Кривые соответствуют системе Мульти-ABP, которая переранжирует список кандидатов определенной длины T (ось x) полученный с помощью мультииндекса второго порядка ($K = 2^{14}$). Пунктирные прямые соответствуют системе, которая переранжирует весь датасет. После переранжирования небольшого количества точек датасета Мульти-ABP достигает той же точности, что и система с полным перебором.

занимается несколько секунд на запрос. Из рисунка 3.7 можно заметить, что в зависимости от T^* системе Мульти-ABP достаточно переранжировать всего несколько десятков тысяч точек (малую долю всего множества) чтобы достичь той же полноты, что и система с полным перебором. Видимо, потери информации за счет сжатия превосходят потери вследствие приближенного поиска мультииндексом. Интересно, что кривые для Мульти-ABP иногда достигают полноты, большей, чем система с полным перебором. Так как мультиквантизация приводит к потере информации, некоторые шумовые векторы оказываются более близкими, чем истинные ближайшие соседи после переранжирования. В некоторых случаях по мере роста T истинные соседи сначала попадают в шорт-лист длины T^*, но потом "вымываются"оттуда, так как все больше и больше шумовых векторов попадают в множество кандидатов.

Сравнение систем Мульти-O-ABP и IVFADC В этой серии экспериментов оценивается качество (полнота@T^* и время работы) системы Мульти-O-ABP для $T^* = 1,10,100$, $T = 10000,30000,100000$, и числа байт для переранжирования $m = 8,16$. Основные показатели представлены в таблице 3.1. Для датасета *Tiny Images* также визуализировано несколько результатов поиска системой Мульти-O-ABP на рисунке 3.8. Для датасета BIGANN также приведены полнота и время работы систем IVFADC и IVFADC+R, предложенных в работе [19].

Из таблицы 3.1 можно сделать вывод, что для одинакового бюджета памяти использование инвертированного мультииндекса дает возможность системе Мульти-O-ABP осуществлять поиск существенно быстрее существующей системы IVFADC. Причина в том, что Мульти-O-ABP приходится переранжировать гораздо меньшее количество кандидатов (десятки тысяч вместо сотен тысяч) для получения сопоставимой или лучшей полноты, чем IVFADC. Дополнительные затраты по памяти в системе Мульти-O-ABP относительно IVFADC составляет около 8% (∼13Гб и ∼12Гб) для $m = 8$ и около 5% (∼21Гб и ∼20Гб) для $m = 16$

Число слов в IVFADC. Основным соперником в сравнении является система IVFADC с $K = 2^{16}$ словами в словаре. Как правило, словари большего размера приводят к лучшему качеству шорт-листов (из-за более мелкого разбиения пространства) и более точному переранжированию (так как коди-

Система	Регионы	T	П@1	П@10	П@100	Время (мс)	Память (Гб)
BIGANN, 1 миллиард СИФТ векторов, 8 байт на вектор							
IVFADC [19]	2^{13}	8 млн	$0.112_{(0.088)}$	$0.343_{(0.372)}$	$0.728_{(0.733)}$	$155_{(74)}$	12
IVFADC [19]	2^{16}	600000	0.124	0.414	0.772	25	12
Мульти-О-АВР	$2^{14} \times 2^{14}$	10000	0.153	0.473	0.707	2	13
Мульти-О-АВР	$2^{14} \times 2^{14}$	30000	0.161	0.506	0.813	4	13
Мульти-О-АВР	$2^{14} \times 2^{14}$	100000	0.162	0.515	0.854	11	13
BIGANN, 1 миллиард СИФТ векторов, 16 байт на вектор							
IVFADC+R [19]	2^{13}	8 млн	(0.262)	(0.701)	(0.962)	(116*)	20
IVFADC [19]	2^{16}	600000	0.311	0.750	0.923	28	20
Мульти-О-АВР	$2^{14} \times 2^{14}$	10000	0.303	0.672	0.742	2	21
Мульти-О-АВР	$2^{14} \times 2^{14}$	30000	0.325	0.762	0.883	5	21
Мульти-О-АВР	$2^{14} \times 2^{14}$	100000	0.332	0.799	0.959	16	21
Tiny Images, 80 миллионов ГИСТ векторов, 8 байт на вектор							
Мульти-О-АВР	$2^{14} \times 2^{14}$	10000	0.317	0.455	0.604	3	<1
Мульти-О-АВР	$2^{14} \times 2^{14}$	30000	0.317	0.485	0.673	4	<1
Мульти-О-АВР	$2^{14} \times 2^{14}$	100000	0.317	0.485	0.673	11	<1
Tiny Images, 80 миллионов ГИСТ векторов, 16 байт на вектор							
Мульти-О-АВР	$2^{14} \times 2^{14}$	10000	0.317	0.544	0.653	3	<1
Мульти-О-АВР	$2^{14} \times 2^{14}$	30000	0.326	0.574	0.733	5	<1
Мульти-О-АВР	$2^{14} \times 2^{14}$	100000	0.327	0.584	0.852	17	<1

Таблица 3.1

Качество работы (полнота для топ-1, топ-10 и топ-100 после переранжирования и время поиска в миллисекундах) для системы Мульти-О-АВР (основанной на мультииндексе второго порядка с K=2^{14}) для различных датасетов и разном бюджете памяти на вектор. Также приведено качество для систем IVFADC и IVFADC+R (наша реализация и числа из работы [19] в скобках).

руемые векторы разностей имеют меньшую норму и их проще закодировать). Но использование словарей большого размера в IVFADC приводит к большому времени вычисления расстояний от запроса до слов. Например, в случае $K = 2^{16}$ это вычисление занимает $3 - 4$ миллисекунды, а для больших словарей ($K > 2^{16}$) оно бы требовало времени больше, чем полное время поиска с переранжированием в системе Мульти-О-АВР (см. таблицу 3.1). Конечно, можно использовать приближенные методы нахождения ближайших слов, но рисунок 3.4 демонстрирует, что для областей высокой полноты такое решение неоптимально. Например, для значений полноты выше 0.9 точное вычисление ближайших слов при $K = 2^{16}$ приводит к лучшему качеству шорт-листов, чем приближенное вычисление при $K = 2^{18}$.

Второй недостаток больших размеров словарей K — большое время построения индекса. Это время практически линейно по K (строгой линейности нет из-за эффективности BLAS инструкций). Таким образом, построение мультииндекса для датасета большого размера требует гораздо меньше времени, чем построение стандартного инвертированного индекса, способного достичь сопоставимых значений полносты.

Сравнение мультииндексов различных порядков

В этой части сравнивается качество работы мультииндексов второго и четвертого порядков с одинаковым числом эффективных слов. Результаты сравнения представлены в таблице 3.2. Как можно видеть, система Мульти-4-О-АВР (основанная на мультииндексе четвертого порядка) формирует список кандидатов гораздо быстрее, чем система Мульти-О-АВР (основанная на мультииндексе второго порядка), но полнота системы четвертого порядка значительно ниже. Причина низкой полноты в том, что эффективные центроиды регионов в разбиениях Мульти-О-АВР и Мульти-4-О-АВР образованы в предположении независимых половин и четвертей исходных векторов соответственно. Так как предположение о независимости четвертей более сильное, эффективные центроиды регионов в системе Мульти-4-О-АВР описывают датасет хуже, что приводит к падению полноты. Более того, с точки зрения времени поиска у Мульти-

Система	П@1	П@10	П@100	Время(мс)
BIGANN, 1 миллиард СИФТ векторов, 8 байт на вектор				
Multi-D-ADC	0.153	0.473	0.707	2
Multi-4-D-ADC	0.093	0.276	0.407	1
BIGANN, 1 миллиард СИФТ векторов, 16 байт на вектор				
Multi-D-ADC	0.303	0.672	0.742	2
Multi-4-D-ADC	0.191	0.391	0.427	1
Tiny Images, 80 миллионов ГИСТ векторов, 8 байт на вектор				
Multi-D-ADC	0.317	0.455	0.604	3
Multi-4-D-ADC	0.247	0.426	0.495	1
Tiny Images, 80 миллионов ГИСТ векторов, 16 байт на вектор				
Multi-D-ADC	0.317	0.544	0.653	3
Multi-4-D-ADC	0.307	0.475	0.523	1

Таблица 3.2

Качество работы систем Мульти-О-АВР и Мульти-4-О-АВР с одинаковым числом эффективных слов (K=2^{14} в системе второго порядка и K=2^7 в системе четвертого порядка) для различных датасетов. Во всех экспериментах система Мульти-4-О-АВР формирует шорт-листы заданной длины быстрее благодаря быстрому вычислению расстояний от запроса до слов. Но качество этих шорт-листов существенно ниже, чем у системы второго порядка. Разница в качестве шорт-листов приводит к тому, что полнота после переранжирования у системы четвертого порядка значительно хуже.

4-О-АВР нет существенного преимущества, что говорит о том, что на практике стоит использовать систему Мульти-О-АВР. Ниже системы также сравниваются для задачи поиска дубликатов.

Мульти-О-АВР и МГК

В этой серии экспериментов система Мульти-О-АВР комбинируется с методов главных компонент (МГК) с использованием наивного и продвинутого подходов (см. 3.2.3). Подходы сравниваются на датасете BIGANN с четырехкратным снижением размерности до 32 компонент. В части 3.2.3 оценивалось влияние сжатия МГК лишь на качество индексации. В рассматриваемых экс-

Система	П@1	П@10	П@100	Время(мс)
BIGANN, 1 миллиард СИФТ векторов, 8 байт на вектор				
Без МГК	0.153	0.473	0.707	2
Наивный подход	0.116	0.348	0.525	1
Продвинутый подход	0.116	0.387	0.645	1

Таблица 3.3

Качество поиска для различных подходов комбинирования мультииндекс второго порядка и четырехкратное снижение размерности методом главных компонент. Для обоих подходов полученное ускорение одинаково, но полнота выше с использованием продвинутого подхода, так как этот подход учитывает независимость подвекторов при снижении размерности.

периментах это сжатие также влияет на качество переранжирования, так как дополнительные байты в системе Мульти-О-АВР кодируют неоптимальные (из-за потери информации в МГК) векторы разностей. Качество обоих подходов представлено в таблице 3.3. Как ожидалось, ускорение одинаково для обоих подходов. Рисунок 3.6 говорит о том, что качество индексации практически не страдает от использования МГК с продвинутым подходом, что означает, что ближайшие слова в мультииндексе можно находить на основе нескольких главных компонент запроса. К сожалению, таблица 3.3 демонстрирует значительное падение полноты после переранжирования даже с продвинутым подходом. Видимо, менее значительные главные компоненты, которыми МГК при сжатии пренебрегает, необходимы для точного переранжирования.

Улучшенные модификации системы Мульти-О-АВР

После первой публикации о системе Мульти-О-АВР [67] было предложено две модификации для ее улучшения. Во-первых, в работе [65] было предложено повысить качество путем замены мультиквантизации на оптимизированную мультиквантизацию для обучения словарей для индексации и переранжирования. Благодаря этой замене, полученная система ОМульти-О-ОАВР достигает значительного повышения полноты поиска.

Рисунок 3.8 — Примеры поисковой выдачи на датасете Tiny Images. В каждой из трех пар строк, самое левое изображение соответствует запросу, верхняя строка в паре соответствует истинным ближайшим соседям по евклидовой метрике, найденным полным перебором. Нижняя строка в паре соответствует соседям, найденным системой Мульти-O-ABP ($K = 2^{14}$, $m = 16$ байт для переранжирования). Визуально результаты, полученные системой Мульти-O-ABP сопоставимы по качеству с результатами, полученными полным перебором.

Система	Регионы	l	П@1	П@10	П@100	Время (мс)	Память (Гб)
BIGANN, 1 миллиард СИФТ векторов, 8 байт на вектор							
IVFOADC	2^{16}	600000	0.138	0.451	0.810	25	12
ОМульти-О-ОАВР [65]	$2^{14} \times 2^{14}$	10000	0.180	0.518	0.747	2	13
ОМульти-О-ОАВР [65]	$2^{14} \times 2^{14}$	30000	0.184	0.548	0.848	4	13
ОМульти-О-ОАВР [65]	$2^{14} \times 2^{14}$	100000	0.186	0.559	0.892	11	13
ОМульти-О-ОАВР-Лок [66]	$2^{14} \times 2^{14}$	10000	0.268	0.644	0.776	6	15
ОМульти-О-ОАВР-Лок [66]	$2^{14} \times 2^{14}$	30000	0.280	0.704	0.894	16	15
ОМульти-О-ОАВР-Лок [66]	$2^{14} \times 2^{14}$	100000	0.286	0.729	0.952	50	15
BIGANN, 1 миллиард СИФТ векторов, 16 байт на вектор							
IVFOADC	2^{16}	600000	0.315	0.764	0.955	28	20
ОМульти-О-ОАВР [65]	$2^{14} \times 2^{14}$	10000	0.339	0.704	0.769	2	21
ОМульти-О-ОАВР [65]	$2^{14} \times 2^{14}$	30000	0.360	0.792	0.901	5	21
ОМульти-О-ОАВР [65]	$2^{14} \times 2^{14}$	100000	0.367	0.834	0.969	16	21
ОМульти-О-ОАВР-Лок [66]	$2^{14} \times 2^{14}$	10000	0.421	0.755	0.782	7	23
ОМульти-О-ОАВР-Лок [66]	$2^{14} \times 2^{14}$	30000	0.454	0.862	0.908	19	23
ОМульти-О-ОАВР-Лок [66]	$2^{14} \times 2^{14}$	100000	0.467	0.914	0.976	66	23

Таблица 3.4

Качество поиска для модификаций системы Мульти-О-АВР. Модификация ОМульти-О-ОАВР, предложенная в работе [65] использует оптимизированную мультиквантизацию для обучения словарей для индексации и переранжирования. Система ОМульти-О-ОАВР-Лок, предложенная в работе [66] достигает более высокой полноты путем использования локальных словарей для переранжирования в каждом регионе индекса. IVFOADC — модификация системы IVFADC, которая использует оптимизированную мультиквантизацию для сжатия точек.

Во-вторых, в работе [66] было предложено использовать отдельные *локальные* словари для переранжирования в каждом регионе мультииндекса, которые используются только для кодирования точек, попавших в этот регион. Полученная система называется ОМульти-О-ОАВР-Лок.

Таблица 3.4 демонстрирует качество работы обеих модификаций ОМульти-О-ОАВР и ОМульти-О-ОАВР-Лок для датасета BIGANN. Система ОМульти-О-ОАВР-Лок достигает значительно более высокой полноты и сейчас является передовым методом для поиска на этом датасете. Несмотря на повышение полноты благодаря локальным словарям, они делают невозможной оптимизацию из части 3.2.2, что отражено в увеличении времени поиска. Также необходима дополнительная память для хранения словарей (около 2 Гб). Система IVFADC также допускает использование ОМК для сжатия точек и эта модификация носит название IVFOADC, качество ее работы также приведено в таблице 3.4 для сравнения.

Важно отметить, что разница в качестве между IVFADC и Мульти-О-АВР остается неизменной после замены МК на ОМК. Использование локальных словарей в системе ОМульти-О-ОАВР-Лок еще более увеличивает эту разницу. В принципе, есть возможность использовать локальные словари в IVFOADC, однако количество памяти, необходимое для их хранения было бы очень большим (например, 8 Гб для $K = 2^{16}$).

Детектирование полудубликатов

В этой части применим инвертированный мультииндекс к задаче детектирования полудубликатов [68], [69], [70]. Все эксперименты проводятся с использованием ГИСТ дескрипторов из коллекции Copydays [64], которые добавлены к ГИСТ дескрипторам датасета *Tiny Images* [63], таким образом получая датасет, в котором известны множества полудубликатов. На этом датасете строятся мультииндексы второго ($K = 2^{14}$) и четвертого ($K = 2^7$) порядков, а дескрипторы исходных изображений из Copydays являются запросами к этим мультииндексам. Основной метрикой качества детектирования полудубликатов является полнота@T, то есть доля инстинных полудубликатов в списке кандидатов

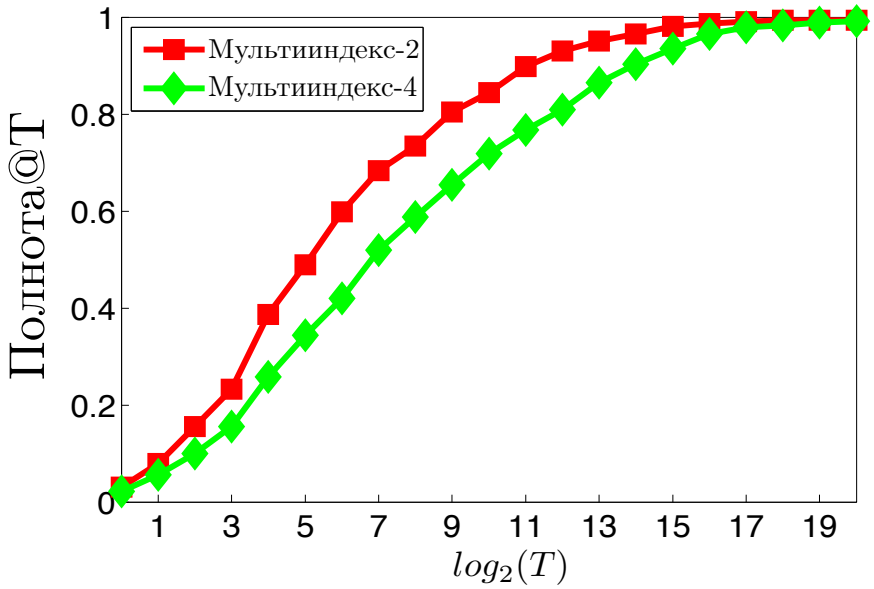


Рисунок 3.9 — Полнота@T детектирования полудубликатов с использованием мультииндексов второго и четвертого порядков как функция T. Для всех разумных значений длин списков мультииндекс второго порядка достигает более высоких значений полноты.

длины T, сформированном мультииндексом, усредненная по всем запросам. Результаты, представленные на рисунках 3.9 и 3.10, созвучны результатам, полученным для задачи поиска ближайшего соседа (3.2).

Как показано на рисунке 3.10, мультииндекс четвертого порядка формирует список кандидатов с заданной точностью быстрее. Однако, как можно видеть из рисунка 3.9, для любого уровня полноты списки кандидатов, сформированные мультииндексом второго порядка, гораздо короче. Как правило, в большинстве систем списки кандидатов переранжируются, поэтому скорее всего мультииндекс второго порядка использовать предпочтительнее, благодаря более коротким спискам, что сэкономит время переранжирования.

3.2.4 Выводы о модели инвертированного мультииндекса

В этой главе описана новая структура данных — инвертированный мультииндекс, который является обобщением стандартного инвертированного индекса для поиска по большим коллекциям высокоразмерных векторов. Экспериментально показано, что мультииндекс второго порядка значительно превос-

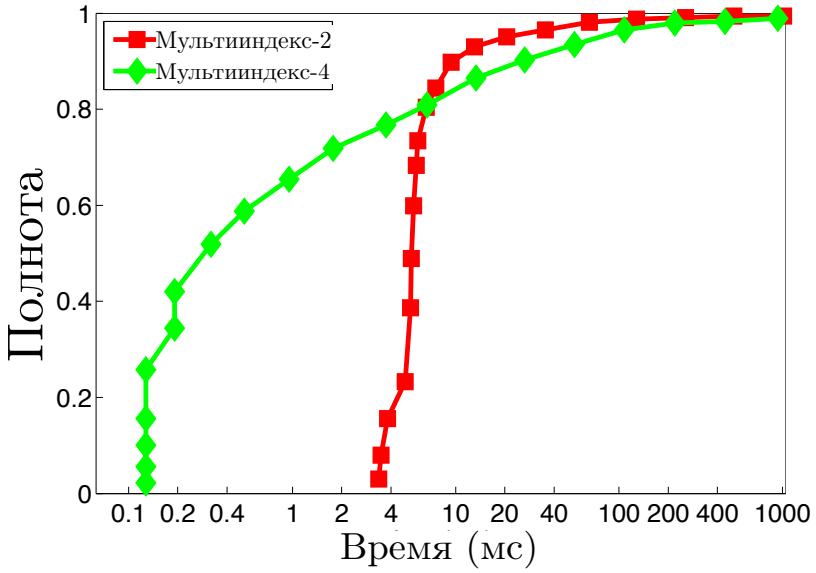
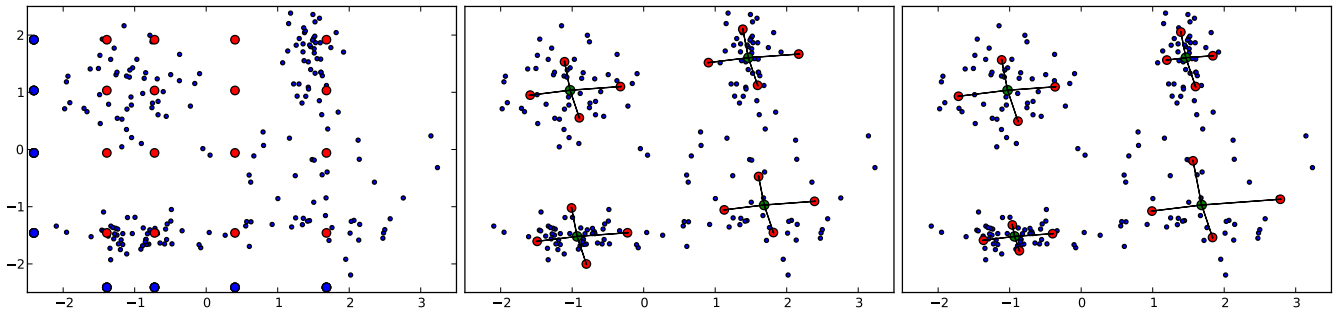


Рисунок 3.10 — Полнота@T детектирования полудубликатов с использованием мультииндексов второго и четвертого порядков как функция времени. Благодаря быстрому выичслению расстояний о запроса до слов, мультииндекс четвертого порядка быстрее обходит регионы, ближайшие к запросу и достигает средних значений полноты быстрее. В области высоких значений полноты предпочтительнее использовать мультииндекс второго порядка: он достигает высоких значений полноты быстрее.



ИМИ НО-ИМИ ОНО-ИМИ

Рисунок 3.11 — Центроиды регионов (красные точки), сформированные различными индексирующими структурами для множества двумерных векторов (синие точки). Для всех индексирующих структур размер словаря равнялся четырем, общее количество центроидов равнялось 16. Левый рисунок соответствует инвертированному мультииндексу и большие синими точками на осях обозначены слова каждого из двух словарей. Средний и правый рисунки соответствуют структурам НО-ИМИ и ОНО-ИМИ соответственно. На обоих рисунках зелеными точками обозначены центроиды первого порядка $S_1, \ldots, S_4$. Центроиды, сформированные ОНО-ИМИ лучше всего описывают действительное распределение данных.

ходит стандартный индекс с точки зрения полноты поиска для одинакового бюджета по времени.

3.3 Неортогональный инвертированный мультииндекс

В последние годы большинство задач компьютерного зрения решаются с помощью дескрипторов образованных глубокими сверточными нейросетями. Несколько работ [23; 41; 42; 45] продемонстрировали, что такие дескрипторы существенно повышают качество поиска на стандартных тестовых коллекциях.

В этой главе изучается построение оптимальной индексирующей структуры для нейросетевых дескрипторов. Во-первых, показано, что для дескрипторов такого типа инвертированный мультииндекс (ИМИ) формирует большое количество пустых регионов. Причина заключается в том, что ИМИ использует независимые словари для различных подпространств пространства поиска. Но так как корреляции между подпространствами в случае нейросетевых дескрип-

торов высоки, независимое построение словарей приводит к недостаточному адекватному описанию распределения.

Во-вторых, предлагается две новые индексирующие структуры: неортогональный инвертированный мультииндекс (НО-ИМИ) и обобщенный неортогональный инвертированный мультииндекс (ОНО-ИМИ). НО-ИМИ формирует центроиды регионов как суммы двух векторов из двух неортогональных словарей, таким образом избегая разбиения на подпространства. В результате НО-ИМИ формирует регионы с центроидами более высокого качества, точнее отражающими действительное распределение точек. ОНО-ИМИ еще более улучшает качество центроидов, формируя их как линейные комбинации слов. Дополнительные линейные коэффициенты также обучаются по данным, тем самым лучше адаптируясь под конкретное распределение.

3.3.1 Описание модели неортогонального инвертированного мультииндекса

В этой части описывается структура неортогонального инвертированного мультииндекса и интуиция, стоящая за ней. Также описаны процедура обучения словарей, формирование списков кандидатов и их переранжирование.

Структура неортогонального инвертированного мультииндекса

Пусть $P = \{p_1, \ldots, p_N\}$ — датасет из N D-мерных векторов. Также как и остальные индексирующие структуры неортогональный мультииндекс разбивает пространство на большое число регионов и распределяет точки датасета по этим регионам. Регионы в неортогональном мультииндексе формируются на основе двух словарей $S = \{S_1, \ldots, S_K\}$ и $T = \{T_1, \ldots, T_K\}$, каждый из которых содержит K слов. Слова в каждом из словарей D-мерные, $S, T \subset \mathbf{R}^D$.

НО-ИМИ разбивает пространство на K^2 регионов с центроидами c_i^j, определяемыми формулой:

$$c_i^j = S_i + T_j, \qquad i,j = 1,\ldots,K \tag{3.5}$$

Таким образом, регионы в НО-ИМИ можно формально определить следующим выражением:

$$C_i^j = \{x \in \mathbf{R}^D | i,j = \arg\min_{k,l} \|x - (S_k + T_l)\|^2\} \tag{3.6}$$

Предлагаемая конструкция похожа на процесс формирования регионов в инвертированной мультииндексе. Однако, в случае НО-ИМИ отсутствует декомпозиция на ортогональные подпространства, игнорирующая значительные корреляции между ними в случае нейросетевых дескрипторов. Вместо этого для формирования центроидов используется сумма слов, которая, как будет показано ниже, позволяет эффективно находить ближайшие регионы. При таком подходе слова НО-ИМИ $S_1,\ldots,S_K$ можно интерпретировать как центры кластеров в исходном пространстве, в связи с этим они называются *центроидами первого порядка*. Аналогично, слова $T_1,\ldots,T_K$ могут быть проинтерпретированы как центроиды кластеров в пространстве разностей между векторами из P и ближайшими к ним центроидами первого порядка. Ниже эти слова будут называться *центроидами второго порядка*. В результате индексирующая структура имеет K^2 центроидов вида $(S_i + T_j)$, $i,j = 1,\ldots,K$ и такое же число регионов, к каждому из которых приписан *индекс первого порядка* i и *индекс второго порядка* j.

Конструкция (3.5) также похожа на иерархический метод к-средних [17]. Этот метод также кластеризует исходное пространство, формируя K центроидов первого порядка. Затем метод кластеризует разности в каждом кластере, формирую K центроидов второго порядка в каждом кластере. Для больших K использование иерархического метода к-средних невозможно, так как это бы требовало хранения K^2 D-мерных векторов в оперативной памяти. Затраты по памяти могут быть уменьшены путем использования одних и тех же центроидов второго порядка во всех кластерах. Такое переиспользование этих центроидов во всех кластерах делает иерархический метод к-средних эквивалентным НО-ИМИ.

Обобщенный неортогональный инвертированный мультииндекс

НО-ИМИ предполагает, что распределения векторов разностей во всех кластерах первого порядка одинаковы, так как он использует одно и то же множество центроидов второго порядка. Однако, это предположение может быть чересчур сильным, так как, например, форма и размеры регионов Вороного могут различаться, особенно когда в пространстве данные есть плотные и разреженные области. Чтобы ослабить это предположение, в неортогональный мультииндекс добавляется α-*матрица*, учитывающая специфическую информацию о каждом кластере. В частности, элемент $\alpha[i,j]$ является множителем для j-ого центроида второго порядка T_j в i-ом кластере первого порядка. С использованием таких множителей выражение для соответствующего центроида становится таким:

$$c_i^j = S_i + \alpha[i,j]T_j, \qquad i,j = 1,\ldots,K \tag{3.7}$$

Добавление α-матрицы позволяет адекватно отражать распределение данных даже с одним набором центроидов второго порядка. Ниже версия НО-ИМИ с α-матрицей называется *обобщенным неортогональным мультииндексом* (ОНО-ИМИ). Примеры центроидов, сформированных разными индексирующими структурами для двумерной коллекции небольшого размера приведены на рисунке 3.11.

Индексация

Параметры ОНО-ИМИ, а именно S,T,α могут быть обучены по данным, как описано ниже. Пока же предположим, что S,T,α даны и опишем процесс индексации, то есть распределение точек $p_1,\ldots,p_N$ по регионам индекса. Формально, для каждой точки p необходимо найти индексы ближайшего центроида:

$$i,j = \arg\min_{k,l} \|p - (S_k + \alpha[k,l]T_l)\|^2 \tag{3.8}$$

Минимизация такой функции полным перебором требовала бы вычисления K^2 различных вариантов, что непрактично. Вместо этого используется простая эвристика, позволяющая отбросить большинство неоптимальных вариантов и существенно сократить перебор. Перепишем выражение выше:

$$\begin{aligned}\|p - (S_k + \alpha[k,l]T_l)\|^2 = \|p - S_k\|^2 + \alpha[k,l]^2\|T_l\|^2 \\ -2\alpha[k,l]\langle p, T_l\rangle + 2\alpha[k,l]\langle S_k, T_l\rangle\end{aligned} \tag{3.9}$$

Заметим, что первый член $\|p - S_k\|^2$ — это расстояние от p до центроида первого порядка S_k. Эвристика заключается в том, чтобы проверить только r значений $k_1, \ldots, k_r$, соответствующих индексам центроидов $S_{k_1}, \ldots, S_{k_r}$, оказавшимися ближайшими к p. С таким отфильтровыванием необходимо проверить всего rK возможных вариантов. Еще одна оптимизация заключается в том, что преподсчитать члены $\|p - S_k\|^2$ и $\langle p, T_l\rangle$, что можно сделать за $O(KD)$ операций. Если эти члены преподсчитаны, то каждый вариант можно проверить за $O(1)$ операций. После проверки всех rK вариантов p приписывается к региону, соответствующему оптимальной паре индексов.

Таким образом, сложность индексации одной точки равна сумме сложностей преподсчета и проверки вариантов, которые равны $O(DK)$ и $O(rK)$ соответственно. Следовательно, полная сложность индексации равна $O(DK + rK)$, в то время как в стандартном мультииндексе эта сложность равна $O(DK)$. Как правило, на практике r в несколько раз меньше D и времена индексации для ИМИ и ОНО-ИМИ довольно близки.

Обучение словарей

Теперь перейдем к решению задачи получения словарей S,T и α-матрицы, хорошо описывающих распределения заданного набора данных. Обозначим обучающее множество векторов за $P = \{p_1, \ldots, p_N\}$. Обучение осуществляется ми-

нимизацией суммарной ошибки аппроксимации всех обучающих векторов:

$$\sum_{i=1}^{N} \left\| p_i - (S_{k^i} + \alpha[k^i,l^i]T_{l^i}) \right\|^2 \to \min_{\substack{S_k \in \mathbf{R}^D, |S|=K, \\ T_k \in \mathbf{R}^D, |T|=K, \\ \alpha \in \mathbf{R}^{D \times D} \\ k^i,l^i \in \{1,\dots,K\}}} \tag{3.10}$$

В данном случае минимизация осуществляется по четырем группам переменных, где переменные S,T и α непрерывны, а переменные индексов $\{k^i, l^i\}$ дискретны. Будем минимизировать функцию блочно-координатным спуском, по очереди минимизируя по одной группе переменных при фиксированных переменных остальных групп. Ниже опишем, как осуществляется минимизация по переменным каждой из групп.

Минимизация по переменным индексов. На этом шаге словари S,T и α-матрица зафиксированы, а минимизация осуществляется по переменным $\{k^i, l^i\}$. Таким образом, этот шаг эквивалентен задаче индексации, эффективное решение которой описано в части 3.3.1.

Минимизация по α. Теперь зафиксируем индексы $\{k^i, l^i\}$, переменные словарей S,T и миниммизируем функцию по элементам α-матрицы. Покажем, что можно независимо осуществлять минимизацию по каждому элементу $\alpha[k,l]$. Для этого декомпозируем целевую функцию следующим образом:

$$\begin{aligned} &\sum_{i=1}^{N} \left\| p_i - (S_{k^i} + \alpha[k^i,l^i]T_{l^i}) \right\|^2 = \\ &= \sum_{k=1}^{K} \sum_{l=1}^{K} \sum_{i:k_i=k,l_i=l} \left\| p_i - S_k - \alpha[k,l]T_l \right\|^2 \end{aligned} \tag{3.11}$$

Заметим, что каждый член $\sum\limits_{i:k_i=k,l_i=l} \left\| p_i - S_k - \alpha[k,l]T_l \right\|^2$ зависит то единственного элемента матрицы $\alpha[k,l]$. Следовательно, минимизация исходной функции эквивалентна независимым минимизациям каждого члена:

$$\sum_{i:k_i=k,l_i=l} \left\| p_i - S_k - \alpha[k,l]T_l \right\|^2 \to \min_{\alpha[k,l]} \tag{3.12}$$

Эта подзадачу можно решить аналитический и ее решение в замкнутом виде таково:

$$\alpha[k,l] = \frac{\sum\limits_{i:k_i=k,l_i=l} \langle p_i - S_k, T_l \rangle}{\sum\limits_{i:k_i=k,l_i=l} \|T_l\|^2} \tag{3.13}$$

Минимизация по словарю второго порядка T. Аналогично, минимизируем по словам второго порядка T_l, снова декомпозируя целевую функцию:

$$\sum_{i=1}^{N} \left\| p_i - (S_{k^i} + \alpha[k^i,l^i] T_{l^i}) \right\|^2 =$$
$$\sum_{l=1}^{K} \sum_{k=1}^{K} \sum_{i:k_i=k,l_i=l} \|p_i - S_k - \alpha[k,l] T_l\|^2 \tag{3.14}$$

Затем каждый член $\sum\limits_{k=1}^{K} \sum\limits_{i:k_i=k,l_i=l} \|p_i - S_k - \alpha[k,l] T_l\|^2$ независимо минимизируется по T_l. Эту минимизацию тоже можно провести аналитически и решение имеет вид:

$$T_l = \frac{\sum\limits_{k=1}^{K} \alpha[k,l] \sum\limits_{i:k_i=k,l_i=l} (p_i - S_k)}{\sum\limits_{k=1}^{K} \sum\limits_{i:k_i=k,l_i=l} \alpha[k,l]^2} \tag{3.15}$$

Минимизация по словарю первого порядка S. Эта задачу также можно декомпозировать и минимизация осуществляется аналогично предыдущему шагу. В этом случае решение имеет вид:

$$S_k = \frac{\sum\limits_{l=1}^{K} \sum\limits_{i:k_i=k,l_i=l} (p_i - \alpha[k,l] T_l)}{\sum\limits_{l=1}^{K} \sum\limits_{i:k_i=k,l_i=l} 1} \tag{3.16}$$

Инициализация. Начальное приближение в обучении ОНО-ИМИ формируется следующим образом. α-матрица инициализируется матрицей из единиц, а словарь S инициализируется центроидами, полученными кластеризацией

исходного множества векторов методом к-средних. Словарь второго порядка T инициализируется центроидами, полученными кластеризацией векторов разностей точек из P и ближайших слов первого порядка.

На практике обучение требует всего нескольких итераций до достижения сходимости. Некоторым парам индексов k,l может соответствовать пустое множество точек, то есть числитель и знаменатель в выражении (3.13) равны нулю. В этом случае $\alpha[k,l]$ устанавливается равным единице.

Формирование списков кандидатов

Опишем процедуру, получающую запрос q и формирующую упорядоченную последовательность регионов индекса, центроиды которых являются ближайшими к этому запросу. Последовательность может иметь любую предустановленную длину и векторы, приписанные к полученным регионам, формируют список кандидатов, который ОНО-ИМИ возвращает для данного запроса. Процедура состоит из трех шагов:

1. Во-первых, вычисляются расстояния $\|q - S_k\|^2$ между запросом и центроидами первого порядка. Обозначим за $k_1,\ldots,k_r$ индексы r центроидов, ближайших к q.

$$distances[k] = \|q - S_k\|^2, \quad k = 1,\ldots,K$$
$$k_1,\ldots,k_r = distances.argsort()[1:r]$$

   Затем отбросим все регионы, индекс первого порядка которых не принадлежит множеству $\{k_1,\ldots,k_r\}$. Дальнейший поиск будет осуществляться среди оставшихся rK регионов.

$$regions = \{(k_i; l)\}$$
$$k_i \in \{k_1,\ldots,k_r\}, \quad l = 1,\ldots,K$$

   Вычислительная сложность этого шага составляет $O(KD + K\log K)$, два слагамых которой соответствуют сложностям вычисления расстояний и сортировки.

2. Во-вторых, процедура создает массив длины rK и заполняет его расстояниями до центроидов перспективных регионов, отобранных на первом шаге:

   for each $region = (k_i; l)$ **in** $regions$**:**

   $$regionDistances[(k_i; l)] = \|q - S_{k_i}\|^2 + \alpha[k_i,l]^2\|T_l\|^2 \\ -2\alpha[k_i,l]\langle q, T_l\rangle + 2\alpha[k_i,l]\langle S_{k_i}; T_l\rangle$$

   Заметим, что члены $\langle q, T_l\rangle$ можно преподсчитать и переиспользовать для всех k_i и l. Члены $\langle S_{k_i}, T_l\rangle$ и нормы $\|T_l\|^2$ преподсчитываются перед получением запросов и хранятся в таблицах поиска. Таким образом, сложность этого шага составляет $O(rK)$.
3. Наконец, применяется частичная сортировка массива расстояний с линейной сложностью (std::nth_element из стандартной библиотеки STL). Для любого входного массива такая сортировка гарантирует, что L минимальных элементов будут находиться на первых L позициях в выходном массиве, где L — желаемая длина последовательности регионов. Затем сортируются L регионов, соответствующих минимальным расстояниям и отсортированная последовательность возвращается в качестве результата.

   $$PartialSort(regionDistances[], L)$$
   $$Output = Sort(regionDistances[1 : L])$$

   Сложность этого шага составляет $O(rK + L \log L)$.

Суммарная сложность процедуры складывается из сложностей трех шагов и равняется $O(KD + K \log K + rK + L \log L)$. В этой сумме члены $O(KD)$ и $O(K \log K)$ также неизбежны для других индексирующих структур, так как все они также вычисяют расстония и сортируют их. Член $O(rK + L \log L)$ отражает дополнительные затраты структуры ОНО-ИМИ, но на практике он не оказывает значительного влияния.

После формирования последовательности регионов как описано выше, метод начинает по очереди обходить регионы и добавлять точки из них в список кандидатов..

Переранжирование

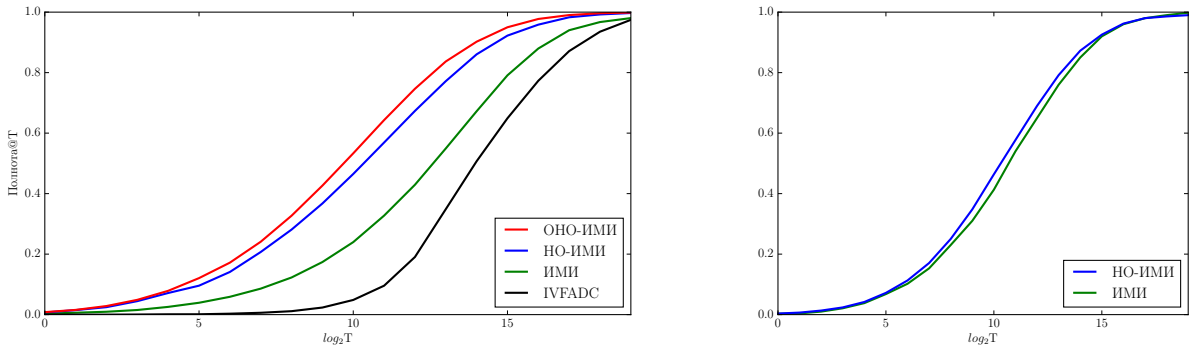
После получения списка кандидатов ОНО-ИМИ переранжирует их по аналогии с существующими системами [66; 67]. Во время индексации для каждой точки вычислялся вектор разности до ближайшего центроида и эти разности сжимались с помощью оптимизированной мультиквантизации (ОМК). В каждом кластере первого порядка используется уникальный локальный набор словарей для переранжирования.

Во время запроса, каждый кандидат реконструируется на основе ОМК-кода и вычисляется расстояние между ним и запросом. Наконец, кандидаты сортируются по возрастанию полученных расстояний.

3.3.2 Эксперименты

В этой части приведем результаты экспериментов, сравнивающим неортогональные мультииндексы со стандартным инвертированным мультииндексом. Эксперименты проводились на двух датасетах:

1. *SIFT1B* [19] содержит миллиард СИФТ векторов с преподсчитанными соседями для 10000 запросов. Также содержит обучающее множество из 100 миллионов дескрипторов.
2. *DEEP1B*. Это новый датасет, подготовленный автором работы. Дескрипторы для этого датасета формировались методом, описанный в работе [45]. Для миллиарда случайных изображений в Интернете были вычислены выходы последнего полносвязного слоя в сети GoogLeNet [71], обученной на датасете ImageNet [31]. Выходы были сжаты методом главных компонент для размерности 96 и l_2-нормализованы. Аналогич-



DEEP1B SIFT1B

Рисунок 3.12 — Полнота поиска как функция длины списка кандидатов. На датасете DEEP1B (слева) сравниваются четыре системы: IVFADC с 2^{17} словами, ИМИ с $K = 2^{14}$, НО-ИМИ и ОНО-ИМИ с $K = 2^{14}$. Всех уровней полноты ОНО-ИМИ достигает с гораздо более короткими списками кандидатов. Для датасета SIFT1B(справа) преимущество НО-ИМИ незначительно.

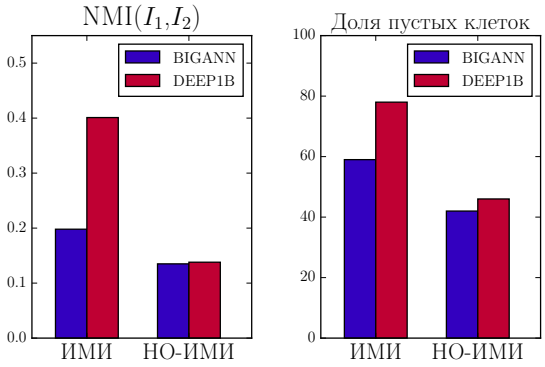


Рисунок 3.13 — Нормализованная взаимная информация между индексами ближайших слов в двух словарях (слева) и доля пустых регионов (справа) для систем ИМИ и НО-ИМИ. Высокое значение взаимной информации в случае датасета DEEP1B и системы ИМИ говорит о том, что неявное предположение о независимости подпространств в ИМИ не выполняется. Большой процент пустых клеток также говорит о том, что центроиды регионов в ИМИ недостаточно хорошо приближают действительное распределение точек в DEEP1B.

Датасет	ИМИ	НО-ИМИ	ОНО-ИМИ
SIFT1B	35923	35207	34981
DEEP1B	0.321	0.272	0.255

Таблица 3.5

Средние расстояния от векторов датасета до ближайших центроидов. Меньшие расстояния обычно соответствуют более точным спискам кандидатов.

ным образом был подготовлен датасет для обучения, содержащий 350 миллионов векторов, и 10000 запросов с преподсчитанными соседями.

Для обоих датасетов сравнивались три индексирующие структуры:

1. инвертированный мультииндекс (ИМИ);
2. неортогональный инвертированный мультииндекс (НО-ИМИ);
3. обобщенный неортогональный инвертированный мультииндекс (ОНО-ИМИ).

Для всех структур использовалось одинаковое значение $K = 2^{14}$, таким образом, все структуры имеют одинаковое число регионов. Обучение параметров всех структур осуществлялось на обучающих множествах.

Качество центроидов. Сначала проверим интуицию о том, что различные половины глубоких дескрипторов значительно коррелируют по сравнению с СИФТ дескрипторами. Как сказано выше, центроиды в модели ИМИ формируются из слов двух словарей C_1 и C_2, соответствующим ортогональным подпространствам. Введем две дискретные случайные переменные I_1, I_2, соответствующие индексам ближайших слов из C_1 и C_2 соответственно для заданного вектора. Измерим нормализованную взаимную информацию между I_1 и I_2, используя все точки из датасета. Высокие значения взаимной информации говорят о значительных корреляциях между половинами.

Рисунок 3.13 (слева) демонстрирует значения нормализованной взаимной информации (НВИ) для двух датасетов. Для датасета DEEP1B НВИ в два раза выше, чем для датасета SIFT1B. Этот факт подтверждает интуицию о том, что разбиение пространства глубоких дескрипторов на ортогональные подпространства приводит к снижению качества индексации. Рисунок 3.13 также демонстрирует значения НВИ для структуры НО-ИМИ. в этом случае измеряется взаимная информация между индексами ближайших слов в словарях S и T. Для обоих датасетов НВИ значительно ниже в случае НО-ИМИ. Так-

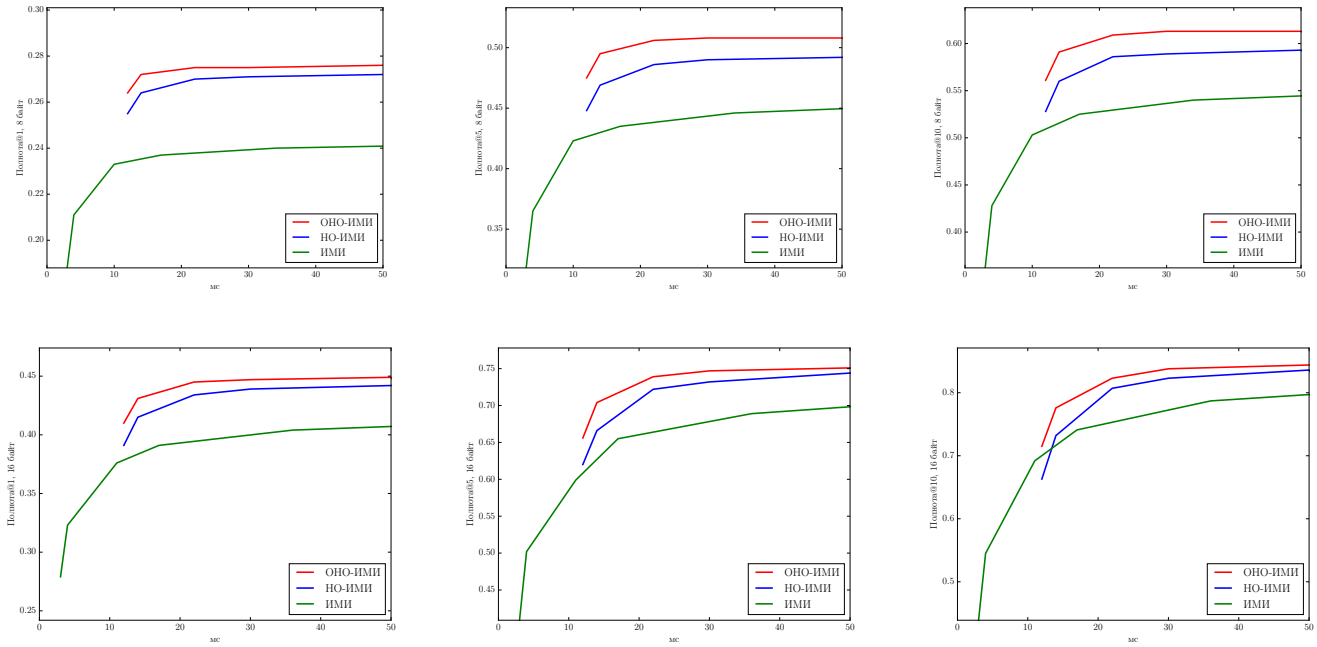


Рисунок 3.14 — Сравнение сруктур ИМИ, НО-ИМИ, ОНО-ИМИ с точки зрения полноты после переранжирования и времени поиска на датасета DEEP1B. Во всех системах для переранжирования использовалось сжатие OMK. Для любого бюджета времени более 11 миллисекунд ОНО-ИМИ достигает значительно более высокой точности по сравнению со стандартным мультииндексом.

же вычислен процент пустых регионов для обеих индексирующих структур, результаты продемонстрированы на рисунке 3.13(справа).

Чтобы дополнительно продемонстрировать преимущество структуры ОНО-ИМИ для глубоких дескрипторов, измерим средние расстояния от векторов индексируемого датасета до центроида регионов, которым они принадлежат. Обычно меньшие расстояния говорят о том, что центроиды приближают инстинное распределение данных лучше. Более того, меньшие расстония означают, что вектора разностей точек датасета и центроидов имеют меньшие нормы, а значит могут быть точнее закодированы, что приводит к более точному переранжированию. Средние расстояния для двух датасетов и трех систем приведены в таблице 3.5.

Для глубоких дескрипторов НО-ИМИ снижает среднее расстояние на 15% относительно стандартного инвертированного мультииндекса с тем же числом регионов. Это означает, что центроиды в НО-ИМИ лучше приближают истинное распределение данных и, как будет показано ниже, это приводит к лучшей

полноте поиска. Структура ОНО-ИМИ приводит к еще большему снижению на 21%. Для датасета SIFT1B использование НО-ИМИ снижает среднее расстояние всего на три процента, что говорит о том, что в этом случае использование предлагаемой структуры не оправдано.

Качество списков кандидатов.

Сравним качество поиска с использованием различных структур. В большинстве экспериментов используется метрика качества Полнота@T, равная доле запросов, для которых истинный ближайший сосед был найден в списке кандидатов длины T.

На рисунке 3.12 приведены значения Полноты@T для различных значений T. Для датасета DEEP1B (слева) использование НО-ИМИ и ОНО-ИМИ приводит к значительному улучшению качества списка кандидатов по сравнению со стандартным ИМИ. Например, уровня полноты 0.5 ОНО-ИМИ достигает с использованием списка в восемь раз короче. Для датасета SIFT1B (слева) преимущество НО-ИМИ незначительно, качество списков практически не изменяется. Так как (О)НО-ИМИ требует дополнительных вычислительных затрат по сравнению с ИМИ, использование ътих структур с датасетом СИФТ дескрипторов не оправдано.

В этом эксперименте также проводится сравнение с системой IVFADC с гораздо большим словарем размера $K = 2^{17}$, но полнота этой системы существенно ниже по сравнению с другими.

Переранжирование и время поиска. Наконец, сравним использование различных структур с последующим переранжированием. Основной метод соперник в этом эксперименте — система ОМульти-О-ОАВР-Лок, приводящая к самым лучшим результатам на датасете SIFT1B.

Эксперименты проводились с $M = 8$ и 16 словарями для переранжирования, что соответствует 8 и 16 байтам на кодируемый вектор соответственно. Параметр r в системе (О)НО-ИМИ равнялся 32 во всех экспериментах. Для обоих значений M на рисунке 3.14 изображены значения Полноты@1, Полноты@5, Полноты@10 для различных длин список кандидатов, как функции времени поиска. Приведем несколько основных выводов:

– Времена поиска для систем НО-ИМИ и ОНО-ИМИ начинается с 11 миллисекунд, что отражает дополнительные вычислительные затраты этих схем. За это время ОМульти-О-ОАВР-Лок успевает переранжировать

около 25 тысяч точек. Рисунок 3.12 демонстрирует, что такие короткие списки в ИМИ содержат ближайшего соседа всего для 70% запросов, следовательно такие короткие списки ненадежны.

– Для любого бюджета времени большего 11 миллисекунд (О)НО-ИМИ достигает более высоких значений полноты по сравнению с ОМульти-О-ОАВР-Лок. Преимущество доходит до 6 абсолютных проценто полноты, что соответствует 13% относительного улучшения.

Потребление памяти. Сравним потребление памяти у различных систем. НО-ИМИ необходим один дополнительный гигабайт для хранения членов $\langle S_i, T_j \rangle$, $i,j = 1, \ldots, K$. ОНО-ИМИ также необходим еще один гигабайт для хранения K^2 элементов α-матрицы.

3.3.3 Выводы о модели неортогонального инвертированного мультииндекса

В этой части предложено две индексирующие структуры для глубоких дескрипторов: неортогональный инвертированный мультииндекс и обобщенный неортогональный инвертированный мультииндекс. Обе структуры способны формировать списки кандидатов лучшего качества относительно стандартного мультииндекса для случая глубоких дескрипторов.

3.3.4 Заключение

В этой части был исследован вопрос построения оптимальной структуры данных для эффективного поиска ближайших соседей по коллекциям из миллиарда векторов. Автором было разработано три структуры данных: ИМИ, НО-ИМИ, ОНО-ИМИ, каждая из которых способна осуществлять поиск по коллекциям из миллиардов векторов за несколько миллисекунд. Экспериментально продемонстрировано качество работы всех предложенных структур данных на

коллекциях из миллиарда СИФТ векторов и миллиарда нейросетевых дескрипторов.

Заключение

Основные результаты данной диссертационной работы заключаются в следующем.

1. В главе 1 предложено два метода построения дескрипторов изображения на основе глубоких сверточных сетей для задачи визуального поиска: нейросетевые дескрипторы с полносвязных слоев и СПОК дескрипторы. Описан способ адаптации дескрипторов к конкретной поисковой коллекции путем дообучения нейросети, детально представлен протокол сбора коллекции для дообучения. Также исследован вопрос оптимального сжатия нейросетевых дескрипторов, продемонстрировано, что дескрипторы с полносвязных слоев сжимаются методом главных компонент практически без потерь в качестве, в то время как СПОК дескрипторы необходимо сжимать с использованием операции обесцвечивания. На большом количестве тестовых коллекций экспериментально показано, что предложенные нейросетевые дескрипторы более компактны и значительно превосходят предшествующие подходы по точности поиска.
2. В главе 2 предложены два метода сжатия векторов высокой размерности: Аддитивная квантизация и Древесная квантизация, основанные на новой модели аппроксимации векторов с помощью суммы слов. Для обеих моделей описаны процедуры обучения параметров, кодирования и эффективного вычисления расстояний и скалярных произведений. Преимущество модели Аддитивной квантизации состоит в отсутствии каких-либо ограничений на параметры слов, что приводит к наивысшему качеству кодирования в случае коротких кодов. Для длинных кодов кодирование Аддитивной квантизацией вычислительно неэффективно. Модель Древесной квантизации ограничивает возможные значения слов, но допускает эффективное обучение для кодов любой длины. Доказана теорема о том, что ошибка аппроксимации предложенных методов не превосходит ошибки аппроксимации существующих подходов мультиквантизации. Экспериментально показано, что предложен-

ные методы значительно снижают потери информации при сжатии данных различной природы из прикладных задач.

3. В главе 3 разработаны две структуры данных для эффективного поиска ближайших соседей в поисковых коллекциях из миллиардов векторов: инвертированный мультииндекс и неортогональный инвертированный мультииндекс. Демонстрируется, каким образом мультииндекс разбивает пространство поиска на очень большое число регионов и в то же время позволяет эффективно находить центроиды регионов, ближайшие к запросу. Описано, каким образом структура мультииндекса может быть использована для поиска ближайших соседей и детектирования изображений-дубликатов. Приводятся теоретические оценки алгоритмической сложности операций в мультииндексе. Экспериментально показано, что инвертированный мультииндекс является единственной структурой данных, позволяющей искать ближайших соседей среди миллиардов векторов за несколько миллисекунд.

Наиболее перспективными направлениями дальнейшего развития методов поиска по большим коллекциям изображений автор считает следующие:

1. **Построение дескрипторов изображений с помощью нейросетей, архитектура которых учитывает специфику задачи поиска.** В данной диссертации автором рассмотрены методы построения дескрипторов нейросетями, решающими задачу классификации, которая, вообще говоря, не является эквивалентной задаче поиска. Кроме того, архитектуры нейросетей в рассмотренных методах никаким образом не учитывают того факта, что близость между извлеченными дескрипторами будет вычисляться посредством евклидовой метрики. В связи с этим перспективным направлением исследований является изучение архитектур, допускающих совместное обучение дескрипторов и обучение метрики между ними, таких как сиамские [72] или триплетные [73] архитектуры.
2. **Совместное обучение нейросети для извлечения дескрипторов и модели их компрессии.** Во всех рассмотренных методах обучение нейросети для извлечения дескрипторов и обучение словарей в модели компрессии (например, методом мультиквантизации) осуществляется последовательно и независимо. Одним из возможных способов повы-

сить качество компрессии является обучение нейросети, учитывающей, что построенные на нее основе дескрипторы будут сжиматься на основе модели мультиквантизации, при котором параметры нейросети и параметры модели сжатия обучаются совместно. Кроме того, данный подход позволит оптимизировать именно качество поиска для несжатого запроса по сжатых дескрипторам из коллекции непосредственно при обучении.

3. **Построение дескрипторов, оптимизированных под имеющиеся индексирующие структуры.** Основной причиной неоптимальности структуры инвертированного мультииндекса для коллекций нейросетевых дескрипторов являются большие корреляции между различными размерностями этих дескрипторов. Эти корреляции приводят к тому, что большинство регионов в разбиении, образованном мультииндексом являются пустыми, что существенно понижает эффективность поиска. В связи с этим важным направлением исследований является построение дескрипторов с отсутствием корреляций между различными размерностями, чего можно добиться, например, минимизацией взаимной информации между ними во время обучения нейросети.

Список литературы

1. The QBIC Project: Querying Images by Content, Using Color, Texture, and Shape / Wayne Niblack, Ron Barber, William Equitz et al. // Storage and Retrieval for Image and Video Databases, San Jose, CA, USA, January 31 - February 5, 1993. — 1993. — Pp. 173–187.

2. Virage Image Search Engine: An Open Framework for Image Management / Jeffrey R. Bach, Charles Fuller, Amarnath Gupta et al. // Storage and Retrieval for Still Image and Video Databases IV, San Diego/La Jolla, CA, USA, January 28 - February 2, 1996. — 1996. — Pp. 76–87.

3. *Pentland Alex, Picard Rosalind W., Sclaroff Stan.* Photobook: Content-based manipulation of image databases // *International Journal of Computer Vision.* — 1996. — Vol. 18, no. 3. — Pp. 233–254.

4. *Lowe David G.* Object Recognition from Local Scale-Invariant Features // ICCV. — 1999. — Pp. 1150–1157.

5. Aggregating local descriptors into a compact image representation / Herve Jegou, Matthijs Douze, Cordelia Schmid, Patrick Pérez // CVPR. — 2010.

6. Large-Scale Image Retrieval with Compressed Fisher Vectors / Florent Perronnin, Yan Liu, Jorge Sanchez, Herve Poirier // CVPR. — 2010.

7. *Jégou Hervé, Zisserman Andrew.* Triangulation Embedding and Democratic Aggregation for Image Search // IEEE Conference on Computer Vision and Pattern Recognition, CVPR. — 2014. — Pp. 3310–3317.

8. *Salakhutdinov Ruslan, Hinton Geoffrey E.* Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure // Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS 2007, San Juan, Puerto Rico, March 21-24, 2007. — 2007. — Pp. 412–419.

9. *Weiss Yair, Torralba Antonio, Fergus Robert.* Spectral Hashing // NIPS. — 2008. — Pp. 1753–1760.

10. *Gong Yunchao, Lazebnik Svetlana.* Iterative Quantization: A Procrustean Approach to Learning Binary Codes // CVPR. — 2011.

11. Spherical Hashing: Binary Code Embedding with Hyperspheres / Jae-Pil Heo, Youngwoon Lee, Junfeng He et al. // *IEEE Trans. Pattern Anal. Mach. Intell.* — 2015. — Vol. 37, no. 11. — Pp. 2304–2316.

12. *Jégou Hervé, Douze Matthijs, Schmid Cordelia.* Product Quantization for Nearest Neighbor Search // *TPAMI.* — 2011. — Vol. 33, no. 1.

13. *Chen Yongjian, Guan Tao, Wang Cheng.* Approximate Nearest Neighbor Search by Residual Vector Quantization // Sensors. — 2010.

14. *Norouzi Mohammad, Fleet David J.* Cartesian k-means // CVPR. — 2013.

15. Optimized Product Quantization for Approximate Nearest Neighbor Search / Tiezheng Ge, Kaiming He, Qifa Ke, Jian Sun // CVPR. — 2013.

16. *Bentley Jon Louis.* Multidimensional Binary Search Trees Used for Associative Searching // *Commun. ACM.* — 1975. — Vol. 18, no. 9.

17. *Nistér David, Stewénius Henrik.* Scalable Recognition with a Vocabulary Tree // CVPR. — 2006.

18. *Indyk Piotr, Motwani Rajeev.* Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality // STOC. — 1998.

19. Searching in one billion vectors: Re-rank with source coding / Hervé Jégou, Romain Tavenard, Matthijs Douze, Laurent Amsaleg // ICASSP. — 2011.

20. Neural Codes for Image Retrieval / Artem Babenko, Anton Slesarev, Alexander Chigorin, Victor Lempitsky // Lecture Notes in Computer Science. Proceedings of the 13th European Conference on Computer Vision (ECCV 2014), Springer. — Zurich, Switzerland: 2014. — Pp. 584–599.

21. *Babenko Artem, Lempitsky Victor.* Tree quantization for large-scale similarity search and classification // Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. — Boston, USA: 2015. — Pp. 4240–4248.

22. *Babenko Artem, Lempitsky Victor.* Additive Quantization for Extreme Vector Compression // Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. — Columbus, USA: 2014. — Pp. 931–938.

23. *Babenko Artem, Lempitsky Victor.* Aggregating Deep Convolutional Features for Image Retrieval // Proceedings of the IEEE International Conference on Computer Vision. — Santiago, Chile: 2015. — Pp. 1269–1277.

24. *Babenko Artem, Lempitsky Victor.* The Inverted Multi-Index // *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).* — 2015. — Vol. 37, no. 6. — Pp. 1247–1260.

25. *Babenko Artem, Lempitsky Victor.* Efficient Indexing of Billion-Scale Datasets of Deep Descriptors // Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. — Las Vegas, USA: 2016. — Pp. 2055–2063.

26. *Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey E.* ImageNet Classification with Deep Convolutional Neural Networks // Advances in Neural Information Processing Systems (NIPS). — 2012. — Pp. 1106–1114.

27. *Jégou Hervé, Douze Matthijs, Schmid Cordelia.* Hamming embedding and weak geometric consistency for large scale image search // European Conference on Computer Vision. — 2008.

28. Object retrieval with large vocabularies and fast spatial matching / James Philbin, Ondrej Chum, Michael Isard et al. // CVPR. — 2007.

29. *Lowe David G.* Distinctive Image Features from Scale-Invariant Keypoints // *IJCV.* — 2004. — Vol. 60, no. 2.

30. Large-scale image retrieval with compressed Fisher vectors / Florent Perronnin, Yan Liu, Jorge Sánchez, Herve Poirier // The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR. — 2010. — Pp. 3384–3391.

31. *A Berg and J Deng and and L Fei-Fei.* Large scale visual recognition challenge (ILSVRC). — `http://www.image-net.org/challenges/LSVRC/2010/`. — 2010.

32. *Li Yunpeng, Crandall David, Huttenlocher Dan.* Landmark Classification in Large-scale Image Collections // International Conference on Computer Vision. — 2009.

33. Leveraging category-level labels for instance-level image retrieval / Albert Gordo, José A. Rodríguez-Serrano, Florent Perronnin, Ernest Valveny // Computer Vision and Pattern Recognition. — 2012.

34. *Arandjelović R., Zisserman A.* All about VLAD // Computer Vision and Pattern Recognition. — 2013.

35. *Ge Tiezheng, Ke Qifa, Sun Jian.* Sparse-Coded Features for Image Retrieval // British Machine Vision Conference. — 2013.

36. A Large-Scale Hierarchical Multi-View RGB-D Object Dataset / Kevin Lai, Liefeng Bo, Xiaofeng Ren et al. // ICRA. — 2011.

37. *Vedaldi A., Fulkerson B.* VLFeat: An Open and Portable Library of Computer Vision Algorithms. — `http://www.vlfeat.org/`.

38. *Simonyan Karen, Zisserman Andrew.* Very Deep Convolutional Networks for Large-Scale Image Recognition // *CoRR.* — 2014. — Vol. abs/1409.1556.

39. Caffe: Convolutional Architecture for Fast Feature Embedding / Yangqing Jia, Evan Shelhamer, Jeff Donahue et al. // Proceedings of the ACM International Conference on Multimedia, MM. — 2014. — Pp. 675–678.

40. *Douze Matthijs, Jégou Hervé.* The Yael Library // Proceedings of the ACM International Conference on Multimedia, MM. — 2014. — Pp. 687–690.

41. Visual Instance Retrieval with Deep Convolutional Networks / Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, Stefan Carlsson // *CoRR.* — 2014. — Vol. abs/1412.6574.

42. Multi-scale Orderless Pooling of Deep Convolutional Activation Features / Yunchao Gong, Liwei Wang, Ruiqi Guo, Svetlana Lazebnik // 13th European Conference on Computer Vision (ECCV). — 2014. — Pp. 392–407.

43. *Tolias Giorgos, Avrithis Yannis S., Jégou Hervé.* To Aggregate or Not to aggregate: Selective Match Kernels for Image Search // IEEE International Conference on Computer Vision, ICCV. — 2013. — Pp. 1401–1408.

44. From Generic to Specific Deep Representations for Visual Recognition / Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan et al. // *CoRR.* — 2014. — Vol. abs/1406.5774.

45. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition / Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, Stefan Carlsson // IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops. — 2014. — Pp. 512–519.

46. Tech. Rep.: / Tiezheng Ge, Kaiming He, Qifa Ke, Jian Sun: MSR-TR-2013-59, 2013.

47. *Pearl Judea.* Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. — Morgan Kaufmann, 1988.

48. *Besag J.* On the Statistical Analysis of Dirty Pictures. — J. Roy. Stat. Soc. B, 1986.

49. A Comparative Study of Modern Inference Techniques for Discrete Energy Minimization Problems / Jörg H. Kappes, Björn Andres, Fred A. Hamprecht et al. // CVPR. — 2013. — Pp. 1328–1335.

50. *Shapiro Stuart C.* Encyclopedia of Artificial Intelligence. — 1987. — Pp. 56–58.

51. *Arandjelović R., Zisserman A.* Tech. Rep.: : Department of Engineering Science, University of Oxford, 2013.

52. *Oliva Aude, Torralba Antonio.* Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope // *IJCV.* — 2001. — Vol. 42, no. 3.

53. *Sánchez Jorge, Perronnin Florent.* High-dimensional signature compression for large-scale image classification // CVPR. — 2011. — Pp. 1665–1672.

54. *Vedaldi Andrea, Zisserman Andrew.* Sparse kernel approximations for efficient classification and detection // CVPR. — 2012. — Pp. 2320–2327.

55. *Bergamo Alessandro, Torresani Lorenzo, Fitzgibbon Andrew W.* PiCoDes: Learning a Compact Code for Novel-Category Recognition // NIPS. — 2011. — Pp. 2088–2096.

56. *Bergamo Alessandro, Torresani Lorenzo.* Meta-class features for large-scale object categorization on a budget // CVPR. — 2012. — Pp. 3085–3092.

57. *Arandjelović R., Zisserman A.* Multiple queries for large scale specific object retrieval // British Machine Vision Conference. — 2012.

58. *Everingham M., Van Gool L., Williams C. K. I. et al.* The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. — http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

59. *Perronnin Florent, Sánchez Jorge, Mensink Thomas.* Improving the Fisher Kernel for Large-Scale Image Classification // ECCV. — 2010.

60. Gurobi Optimizer. — `http://www.gurobi.com/`. — 2013.

61. Image Classification with the Fisher Vector: Theory and Practice / Jorge Sánchez, Florent Perronnin, Thomas Mensink, Jakob J. Verbeek // *International Journal of Computer Vision.* — 2013. — Vol. 105, no. 3. — Pp. 222–245.

62. *Sivic Josef, Zisserman Andrew.* Video Google: A Text Retrieval Approach to Object Matching in Videos // ICCV. — 2003.

63. *Torralba Antonio, Fergus Robert, Freeman William T.* 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition // *TPAMI.* — 2008. — Vol. 30, no. 11.

64. Evaluation of gist descriptors for web-scale image search / M. Douze, H. Jegou, H. Sandhawalia et al. // CIVR. — 2009.

65. Tech. Rep.: / Tiezheng Ge, Kaiming He, Qifa Ke, Jian Sun: 2013.

66. *Kalantidis Yannis, Avrithis Yannis.* Locally Optimized Product Quantization for Approximate Nearest Neighbor Search // in Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR 2014). — IEEE, 2014.

67. *Babenko Artem, Lempitsky Victor.* The inverted multi-index // Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. — Providence, USA: 2012. — Pp. 3069–3076.

68. *David C. Lee Qifa Ke Michael Isard.* Partition Min-Hash for Partial Duplicate Image Discovery // ECCV. — 2010.

69. Scalable Near Identical Image and Shot Detection / Ondrej Chum, James Philbin, Michael Isard, Andrew Zisserman // CIVR. — 2007.

70. *Ke Yan, Sukthankar Rahul, Huston Larry.* Efficient Near-duplicate Detection and Sub-image Retrieval // ACM Multimedia. — 2004.

71. Going deeper with convolutions / Christian Szegedy, Wei Liu, Yangqing Jia et al. // CVPR. — 2015.

72. *Chopra Sumit, Hadsell Raia, LeCun Yann.* Learning a Similarity Metric Discriminatively, with Application to Face Verification // 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA. — 2005. — Pp. 539–546.

73. *Hoffer Elad, Ailon Nir.* Deep Metric Learning Using Triplet Network // Similarity-Based Pattern Recognition - Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015, Proceedings. — 2015. — Pp. 84–92.

Список рисунков

Список таблиц