**Исследование методов быстрого поиска изображений в задаче распознавания лиц**

**Анотация**

Одной из востребованных областей поиска с Интернете является поиск по изображениям. Современные методы индексации данных показывают отличный результат поиска в многомиллиардных коллекциях и повсеместно применяются для данной задачи. В этой этой работе предлагается изучить существующие подходы индексации больших коллекций изображений и посмотреть на поведение этих алгоритмов в задаче распознавания лиц. Будем считать, что современные нейросетевые алгоритмы достаточно хорошо выделяют признаки лица на изображении и лишь немного затронем этот вопрос. Основной задачей ставим сравнение скорости поиска индексных методов на наборах данных лиц. Начнем рассмотрение с простейших алгоритмов поиска, таких как поиск по точному расстоянию и простая индексная структура. Затем на более продвинутых алгоритмах будем увеличивать скорость поиска и исследовать ухудшение точности. Эксперименты показывают, что в больших коллекция изображений лиц можно добиться приемлемой скорости поиска с хороших показателем точности. Сильное же ускорение приводит к большим потерям.

**Введение**

В последнее десятилетие визуальный поиск стал широко распространенной функцией многих поисковых систем. Большой объем сохраненных изображений в сети измеряется петабайтами, число которых с каждым днем только увеличивается. В связи с этим эффективный поиск ближайших соседей стал серьезной исследовательской проблемой [1-6]. Потребность быстрого поиска похожих изображений занимает большую нишу в современных приложениях компьютерного зрения, в том числе в задаче распознавания лиц [8]. Социальные сети, правоохранительные органы имеют огромные коллекции изображений лиц, среди которых надо уметь быстро извлекать нужную информацию. Для решения данной проблемы требуются эффективные и масштабируемые алгоритмы поиска с низкими временными затратами. Ожидается, что ответ на запросы к базам данных из миллиардов элементов будет занимать несколько миллисекунд.

Решение данной задачи можно рассматривать с двух сторон. Во-первых, даже самые современные алгоритмы поиска и обработки лиц на изображении не идеальны. Это открывает просторы для исследований. Во-вторых, подходящая структура данных для поиска может давать многократное увеличение скорости. Среди всех алгоритмов распознавания сверточные нейронные сети (CNN) показывают лучшие результаты поиска лиц на изображении и используются в большинстве исследованиях этой области [8,10]. В связи с этим основной задачей исследования будем считать проблему выбора поисковой структуры данных, а дескрипторы лиц для экспериментов будем строить по одной из общедоступных CNN.

Все существующие крупномасштабные поисковые системы избегают исчерпывающего поиска путем ограничения конечного набора кандидатов, который рассматривается для запроса. Данный подход называют приближенным поиском ближайших соседей (ANN). Современные алгоритмы ANN имеют три основных реализации: инвертированная индексация [1-6], хеширование [9], многомерная инвертированная индексация основанная на квантовании произведения (PQ) [4,5]. В этой работе основное внимание будет уделено инвертированному индексу и его оптимизации с помощью PQ.

Структуры индексации разбивают пространство поиска на большое количество непересекающихся областей, и в процессе поиска используются только малая часть коллекции, наиболее близкая к конкретному запросу. Отобранная часть данных образует короткий список кандидатов, и поисковая система исчерпывающе рассчитывает расстояния между запросом и всеми кандидатами. На этом этапе важно, чтобы список кандидатов был коротким, так как вычисление расстояния имеет линейную сложность по данной длине. Метод PQ для ANN используется в двух видах: для построения многомерного инвертированного индекса для приближенного поиска или для кодирования векторов в компактные коды для точного поиска. Идея этих подходов состоит в том, чтобы разложить пространство векторов на большое количество непересекающихся множеств и обучить запросы получать доступ к ближайшим из них.

Первая структура индексации, способная работать с миллиардным набором данных, была представлена в [1]. Она была основана на структуре инвертированного индекса, которая разбивает пространство признаков на диаграмму Вороного. Каждая область задается свои центроидом, который предварительно обучили алгоритмом K-средних. Было показано, что эта система достигает разумных скоростей поиска порядка нескольких десятков миллисекунд. Позже обобщение структуры инвертированного индекса было предложено в [3]. В этой работе был представлен инвертированный многомерный индекс или мультииндекс (IMI), который разбивает пространство признаков на несколько ортогональных подпространств и разбивает каждое подпространство на диаграммы Вороного независимо. Декартово произведение такого разбиения образует неявное разбиение всего пространства поиска. Обе эти структуры обладают своими недостатками, которые можно устранить с помощью различных оптимизаций PQ [6,7].

В данной работе будет описано несколько современных архитектур индексирования и путем экспериментов исследована применимость данного подходя для задачи распознавания лиц.

**Постановка задачи**

Главной задачей данной работы является исследование современных индексных структур быстрого поиска и поверка их эффективности в задаче распознавания лиц. Считается, что выделением лица на изображении и построением его признаков занимаются алгоритмы общедоступных сверточных нейронных сетей. Работа будет производится в рамках заранее подготовленных 128-мерных векторов, по одному для каждого изображения. Время построения каждого вектора учитывать не будем, так как во всех алгоритмах будет использоваться одна и та же нейронная сеть. Также не будем учитывать, но обратим внимание на время обучения индексных структур, так как для разных алгоритмов оно может отличаться на порядки.

Проверка заключается в измерении среднего времени поиска похожих лиц в большой коллекции изображений. Время, затраченное на поиск k (k = 1, 5, 10, 30, 50, 100) ближайших соседей будет основным критерием скорости алгоритма. Помимо скорости важно учитывать точность поиска. Во всех сопутствующих работах [1,3,5,6,7] точность измерялась как процент истинных ближайших соседей среди k найденных, где истинные ближайшие соседи определялись точным евклидовым расстоянием. Использовать данную метрику в этой работе следует аккуратно, так как она не отражает природы исследуемой области, а указывает только лишь на качество используемого алгоритма, который и без того много лет тестируется и улучшается. В связи со спецификой нашей задачи, наиболее правильным вариантом измерения точности будет получение процента лиц, отмеченных так же, как и запрос, среди k найденных ближайших соседей. Также в качестве альтернативного показателя качества будем измерять частоту вхождения каждого лица в k ближайших соседей. В случае совпадения лица запроса и лица с наибольшей частотой будем говорить об успешном распознавании. Эта метрика является самой популярной с сфере распознавания лиц.

Для применимости данного подхода к реальным задачам алгоритм должен удовлетворять нескольким требованиям:

1) скорость поиска в большом объеме изображений не должна превышать нескольких миллисекунд;

2) точность поиска должна быть в пределе допустимой для выбранного алгоритма, то есть не сильно отличаться от приводимой в статьях.

В качестве вывода следует оценить эти показатели на основе результатов экспериментов и сформулировать возможные пути улучшения.

**Обзор существующих решений**

В данной части работы будут кратко описаны несколько идей из существующих работ и попутно введены обозначения для дальнейшего исследования.

Начнем с того, что во всех упомянутых ниже алгоритмах решается одна и та же задача. А именно, вычисление евклидовых расстояний между векторами высокой размерности.

Рассмотрим D-мерное евклидово пространство R^D. Общая задача состоит в том, чтобы найти элемент NN(x) в конечном наборе Y ⊂ R^D, минимизируя расстояние до вектора запроса x ∈ R^D:

NN (x) = argmin(d (x, y)). (1)

y∈Y

Поиск ближайшего соседа по своей природе дорогой из-за влияния высокой размерности дескрипторов. Для сокращения времени поиска было предложено несколько методов многомерного индексирования, таких как популярное KD-дерево или другие методы ветвей и границ. Однако для больших размерностей оказывается [12], что такие подходы не намного эффективнее, чем исчерпывающий расчет расстояний, сложность которого составляет O(nD). Также одним из самых популярных алгоритмов ANN является евклидово локально-чувствительное хеширование (LSH) [9]. Однако наивный подход к реализации данного метода не учитывает требования к памяти структуры индексации [2]. В LSH использование памяти может оказаться даже выше, чем у исходных векторов. Этот факт, а также факт наличия случайности серьезно ограничивают применимость данного алгоритма.

В связи с этим более подробно обсудим алгоритмы индексирования основанные на квантовании. Так как полное сравнение вектора запроса со всеми векторами коллекции невозможно, список кандидатов нужно сокращать. Поэтому была придумана инвертированная индексная структура для быстрого доступа к наиболее значимым векторам. Общая идея в том, что инвертированый индекс содержит связные списки векторов, где каждый список является отображением некоторого вектора-центроида. На первом этапе запроса определяется центроид, затем по определенному этим вектором списку выполняется исчерпывающий поиск (Рисунок 1).

Простейший инвертированный индекс основан на векторном квантовании (VQ) [1,2,7]. Цель VQ уменьшить количество элементов представления пространства поиска. Формально VQ - это функция q, отображающая d-мерный вектор x ∈ R^D на вектор q (x) ∈ C = {ci; i ∈ I}, где I — конечное индексное множество: I = 0 , ... , k - 1. Вектора ci называются центроидами. Множество Vi векторов, отображаемых в данный индекс i, называется ячейкой Вороного и определяется как

Vi = {x ∈ R^D: q (x) = ci}. (2)

Ячейки VQ образуют разбиение пространства поиска. Все векторы, лежащие в одной и той же ячейке Vi, представляются одним и тем же центроидом ci. Качество VQ обычно измеряется по среднеквадратичной ошибке между вектором запроса x и его значением q(x):

MSE (q) = Ed (q (x), x)^2

где d (x, y) = || x - y || - евклидово расстояние между x и y.

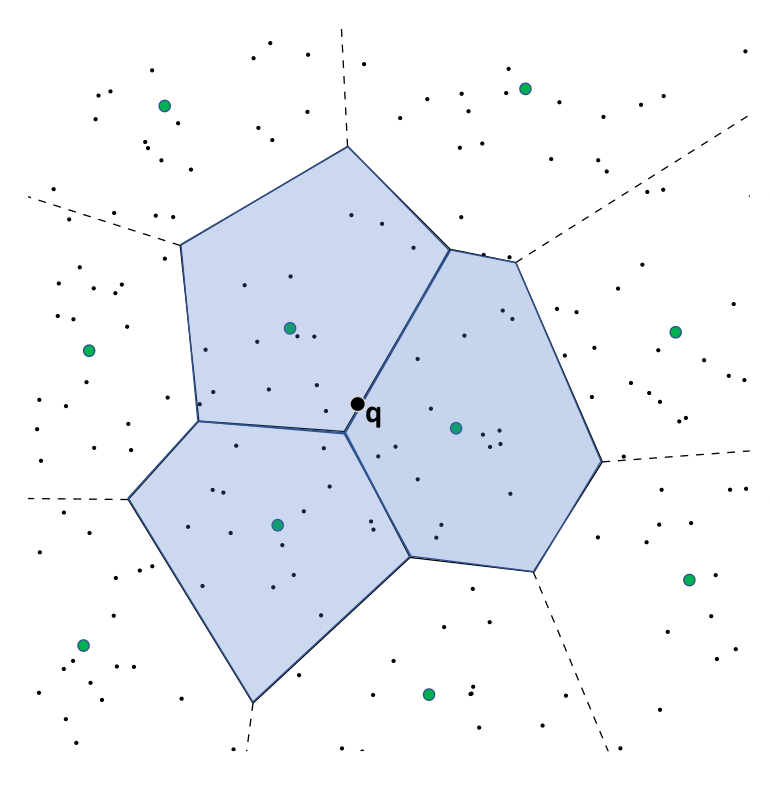
Стандартным методом обучения центроидов VQ является алгоритм кластеризации k-средних, который находит оптимальное разбиение путем итеративного сопоставления векторов набора центроидам и перестройки этих центроидов по сопоставленным векторам.

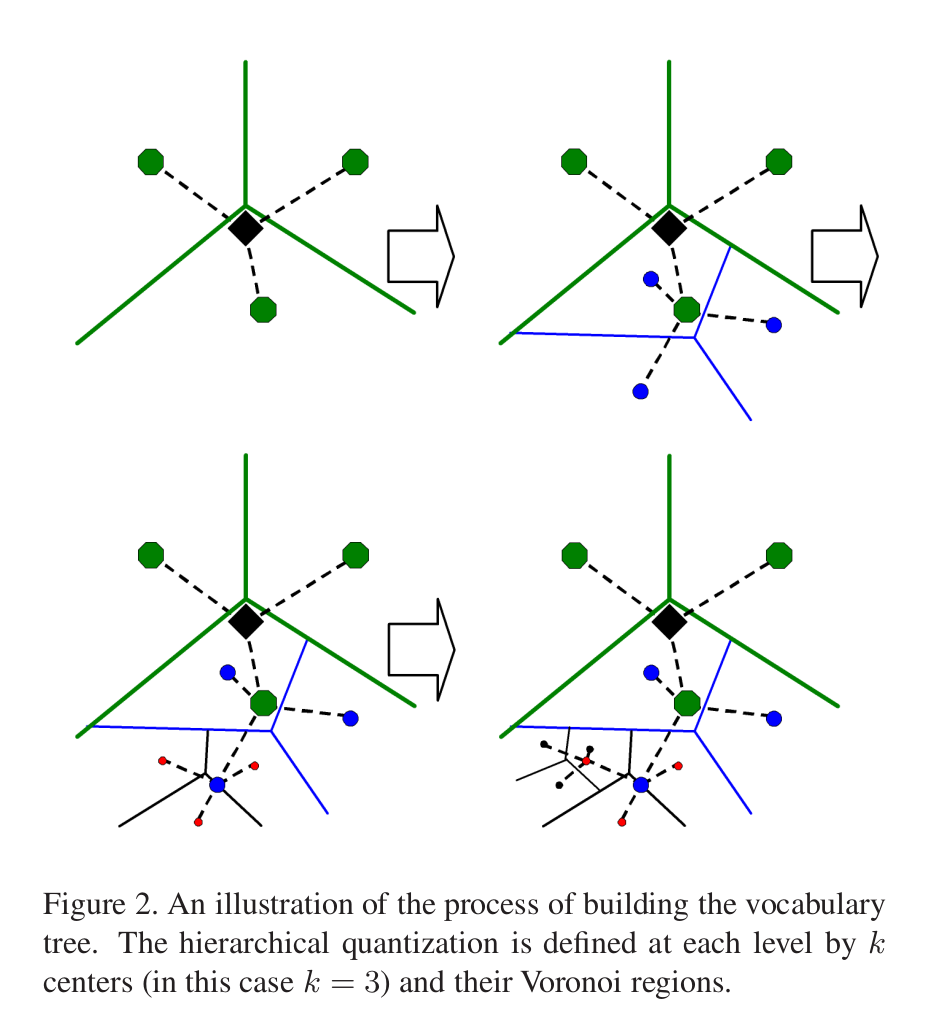
Поиск по данной структуре производится в два этапа:

1) поиск ближайшего к запросу центроида по короткому списку центроидов;

2) поиск по списку кандидатов соответствующих этому центроиду.

Чтобы обеспечить высокую скорость поиска с использование инвертированного индекса количество ячеек должно быть достаточно велико, что сильно снижает скорость обучения алгоритма на больших объемах данных.

Рисунок 1. Индексная структура. Зеленым цветом обозначены центроиды ячеек. Черная точка q — запрос поиска. Синим цветом выделены ближайшие ячейки. Вектора из них попадут в список кандидатов.

Рисунок 2. Иерархический индекс. Процесс построения иерархического инвертированного индекса с разбиением на k ячеек (k = 3). На рисунке представлено четырехуровневое разбиение.

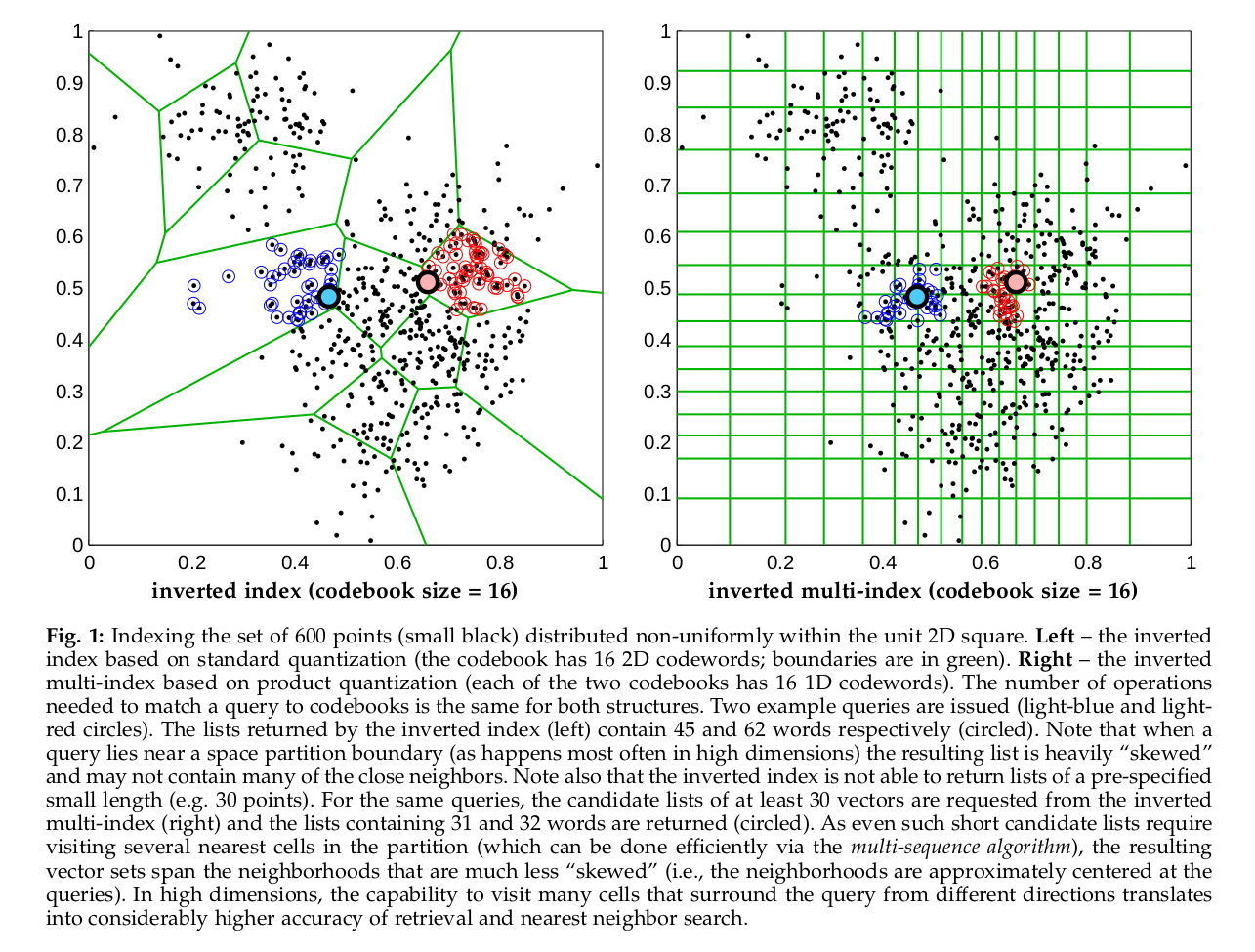
Повысить эффективность этапа обучения можно с помощью инвертированного иерархического индекса (HKM) [11], который помимо первичного разбиения на ячейки Вороного разбивает пространство каждой ячейки повторно (Рисунок 2). В огромных датасетах уровень вложенности разбиения выбирается достаточно большим, что гарантирует небольшой список кандидатов. Помимо скорости обучения, при правильном подборе параметров можно добиться ускорения поиска [11].

Еще одна идея использования индексов для быстрого поиска предлагает разбивать пространство векторов на несколько подпространств меньшей размерности и обучить каждое подпространство малой размерности отдельно. Это позволяет разбивать датасет на огромное количество ячеек и с помощью декартова произведения центроидов быстро получать доступ к ним (Рисунок 3). Данная структура называется инвертированным мульти-индексом (IMI) [3].

Более формально IMI основан на квантовании произведения (PQ) [2], которое представляет из себя схему сжатия с потерями. PQ кодирует каждый вектор x ∈ R^D как объединение M кодовых слов из M D/M-мерных списков C1,..., CM, каждый из которых содержит K кодовых слов. Другими словами, PQ разбивает вектор на M отдельных подвекторов и применяет векторное квантование (VQ) к каждому подвектору, используя при этом отдельный список центроидов. Следовательно процесс обучения IMI схож с процессом обучения обычного инвертированного индекса, за тем лишь исключением, что обучать приходится несколько списков центроидов. В результате каждый вектор x кодируется набором индексов [i1,. , , , iM] и аппроксимируется как x ≈ [C1 (i1),. , , , CM (iM)]. Быстрое вычисление евклидова расстояния становится возможным благодаря эффективному асиметричному вычислению расстояния [2] с использованием обученных центроидов:

||q - x||^2 ≈ ||q - [C1 (i1),…. , CM (iM)]||^2 = СУММА(m = 1, M) ||qm - Cm (im)||^2, (2)

где qm - m-й подвектор запроса q. С точки зрения геометрии PQ эффективно разделяет исходное векторное пространство на K^M ячеек, каждая из которых является декартовым произведением M ячеек меньшей размерности. Однако стоит учитывать, что из-за огромного количества кластеров в IMI некоторые области могут содержать содержать сравнительно малое количество кандидатов или не содержать совсем. Следовательно, IMI в процессе поиска тратит много времени на посещение пустых областей. Причина этого недостатка состоит в том, что IMI при обучении не учитывает зависимость подпространств при разделении, которые зачастую зависимы на практике. В частности, существуют значительные корреляции между различными подпространствами дескрипторов, построенных с помощью сверточных нейронных сетей, которые наиболее актуальны в наши дни. Для решения проблемы адаптации алгоритмов к коррелированным данным можно предварительно производить ортогональные преобразования над данными [4,6] или локальные оптимизации PQ [5].

Рисунок 3. Левый — инвертированный индекс. Правый — инвертированный мульти-индекс. На рисунках представлена разница в разбиении одного и того же набора данных из 600 векторов для разных структур. Для наглядности размерности векторов равны двум. Обе структуры разбивались на 16 ячеек (для каждой размерности). Для примера обозначим 2 запроса к базе данных, которые отмечены синим и красным цветом. На левом рисунке списки кандидатов содержат 45 и 62 кандидата соответственно. Обратите внимание, когда запрос лежит вблизи границы, список кандидатов получается «перекошенным». Для тех же запросов к мультииндексной структуре списки получаются короче — 31 и 32 вектора соответственно. Это связанно с тем, что каждая ячейка содержит меньшее количество векторов. Выбирая несколько близких ячеек, можно добиться более точного списка кандидатов.

**Исследование и построение решений задачи**

Начинать исследование поставленной задачи стоит с определения подходящих инструментов и ресурсов. Важно качественно подобрать эти элементы, чтобы их неточности не влияли на результаты экспериментов.

**Сверточная нейронная сеть**

На сегодняшний день существуют множество алгоритмов обработки лиц на изображении. Чтобы сократить список подходящих под нашу задачу алгоритмов, поставим конкретную задачу обработки изображения. Нужно подобрать алгоритм выделения лица и построения вектора признаков этого лица на изображении. Стоит учесть следующие факты:

1) лицо занимает существенную часть изображения;

2) лицо единственное на изображении.

Как уже было сказано, на сегодняшний день лучшими алгоритмами построения дескриптора лица являются сверточные нейронные сети (CNN) [8,10]. Ставилась задача с помощью экспериментов определить наиболее подходящую CNN под нашу постановку. Основным критерием отбора были показатели точности поиска по евклидову расстоянию с использованием соответствующего вектора признаков. Также учитывалось время обработки изображения. В эксперименте участвовали только общедоступные CNN. Оптимальным для нашей постановки оказался алгоритм из библиотеки Python facerecognition. Во всех дальнейших исследованиях использованы дескрипторы лиц, построенные этим алгоритмом.

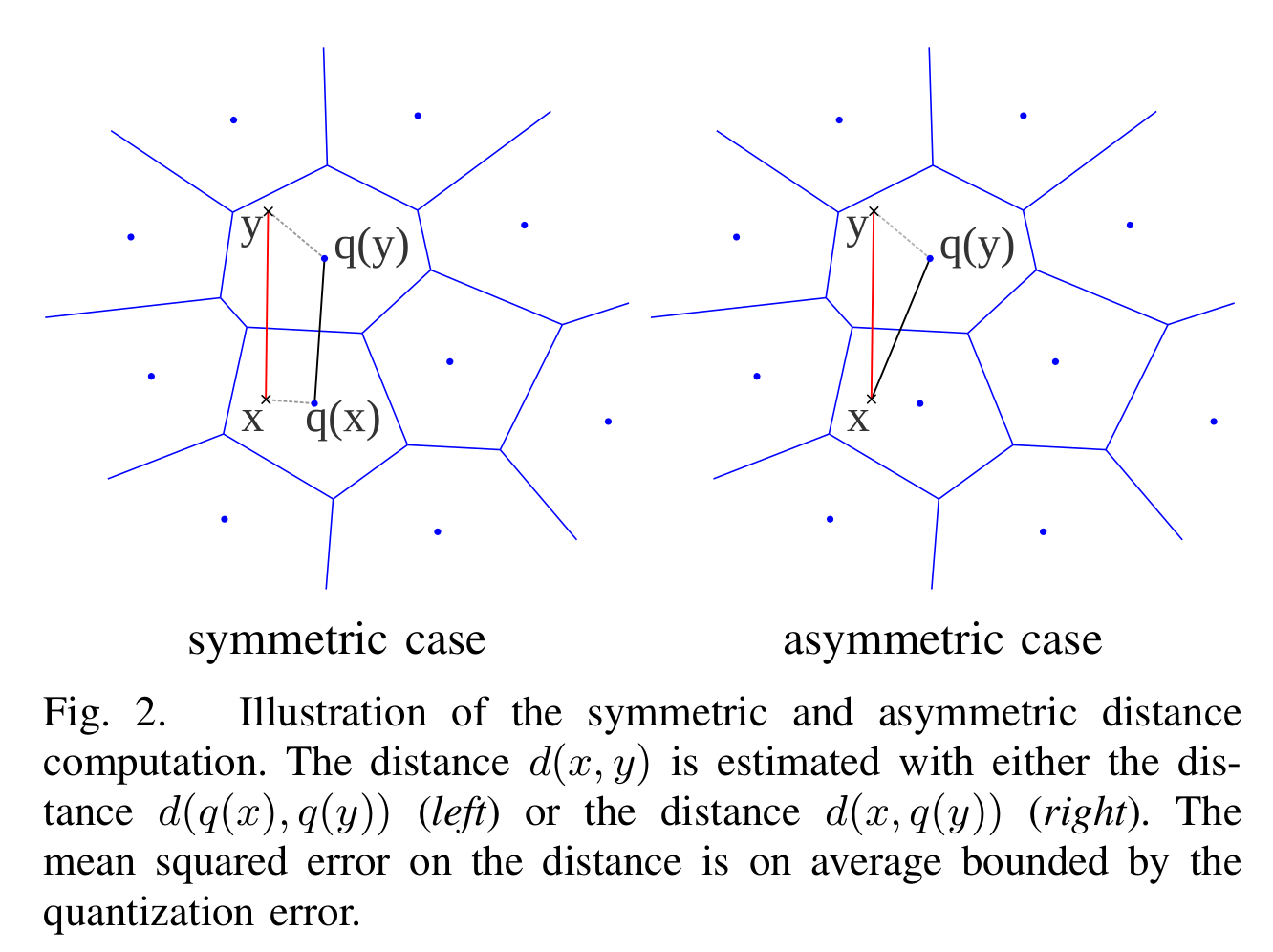
Здесь же стоит сказать про выбранную размерность вектора. Распространенной практикой является выбор размерности вектора признаков равной степени двойки натурального числа. Заметим, что чем больше размерность вектора, тем больше информации о лице он может содержать, однако вектора большой размерности увеличивают как временную, так и пространственную сложность алгоритмов. Учитывая опыт связных работ, размерность дескрипторов лиц была выбрана равной 128.

**Датасет**

Выбору данных для исследования стоит уделить особое внимание. Именно от этого этапа будет зависеть применимость данного подхода на практике. Насколько выбранный датасет будет совпадать с реальными дынными, настолько можно быть уверенным в качестве полученных результатов. Снова обозначим некоторые критерии, которые будем учитывать при выборе набора данных. Во-первых, набор данных должен содержать изображения лиц. Нас будут интересовать датасеты, собранные специально для задач распознавания. Во-вторых, данных должно быть много. Мы не будем рассматривать датасеты из нескольких тысяч изображений. В-третьих, лица на фотографиях должны быть представлены в разных условиях сложности распознавания (положение, освещенность, эмоция, возраст, этническая принадлежность). Одним из самых больших датасетов лиц, подходящих под данные критерии, является VGGFace2. Изображения в нем загружаются из поиска картинок Google, что отлично подходит под исследуемую задачу. Этот набор данных содержит около 3 миллионов изображений - фотографии более 9000 личностей, охватывающих широкий спектр разных национальностей, профессий и возрастов.

**Несколько ближайших ячеек для поиска.**

Еще раз обратимся к Рисунку 3. В разбиении обычной индексной структуры можно заметить «перекос» списка кандидатов. Заметим, что эта проблема появляется и в более сложных структур поиска. Это происходит потому, что запрос на поиск попадает близко к границе разбиения и вследствие этого ближайший центроид не точно приближает вектор запроса. Решением данной проблемы может быть расширение списка кандидатов векторами из нескольких ближайших ячеек. Чтобы контролировать размер списка кандидатов можно добавлять соседние ячейки в рассмотрение до тех пор, пока размер списка не превысит определенного значения. Это значение будем экспериментально для каждой структуры.

Рисунок 4. Левый — симметричное расстояние. Правый — асимметричное расстояние. Расстояние d(x,y) аппроксимируется расстоянием d(q(x), q(y)) и d(x, q(y)) для левого и правого рисунка соответственно.

**Вычисление расстояний с использованием закодированных векторов**

Перед непосредственным описанием экспериментов опишем еще один важный момент. Рассмотрим вектор запроса x и вектор индексированных данных y. Существует два метода вычисления приблизительного евклидова расстояния между этими векторами: симметричный и асимметричный (Рисунок 4).

Вычисление симметричного расстояния (SDC): оба вектора x и y представляются соответствующими центроидами q (x) и q (y). Расстояние d(x, y) аппроксимируется расстоянием d(x, y) = d (q(x), q(y)), которое в случае многомерной индексации эффективно вычисляется следующим образом:

d(x, y) = d(q(x), q(y)) = Корень( Сумма(по j)(d(qj(x), qj(y))^2)), (12)

где расстояние между центроидами предварительно подсчитано. Справочная таблица каждого центроида содержит все квадраты расстояний до остальных центроидов.

Вычисление асимметричного расстояния (ADC): вектор y представлен своим центроидом q(y), а запрос x не закодирован. Расстояние d(x, y) аппроксимируется расстоянием d(x, y) = d(x, q (y)), которое вычисляется с использованием разложения

d(x, y) = d(x, q(y)) =Корень( Сумма(по j)(d(uj(x), qj(y))^2)), (13)

где uj — функция выделения компонент j-ой размерности мультииндексного разбиения.

В обоих случаях функция корня обычно опускается. Единственное преимущество SDC перед ADC - это меньшее использования памяти вектором запроса, поскольку он определяется своим кодом. В большинстве случаев это неактуально, поэтому используют асимметричную версию, которая позволяет получить меньшие искажения расстояния для аналогичной сложности. В остальной части работы будем пользоваться ADC без явного упоминания.

**Эксперименты**

В данном разделе будут представлены описания экспериментов для каждой из исследуемых структур данных. Ограничимся здесь описанием основных принципов работы алгоритмов. Пошаговое описание работы каждой структуры будет представлено в следующей главе.

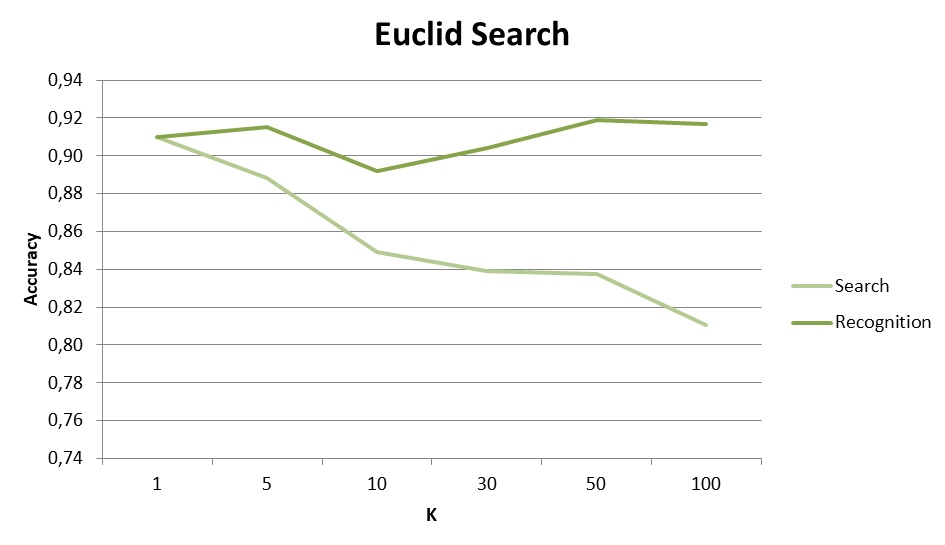
Начнем с того, что для исследования поставленной задачи требуется огромное количества ресурсов. В связи с некоторыми ограничениями, будем проводить эксперименты на меньшем количестве данных и обобщать результат на общий случай исследования.

Для проведения экспериментов было обработано свыше 150 тысяч изображений. Для каждого из которых было произведено выделение лица и построение 128-мерного вектора признаков. Данные были сохранены в один csv-файл, который содержит информацию о 500 личностях. Для чистоты экспериментов, все замеры проводились по 1000 раз и затем результат усреднялся.

Как упоминалось в постановке, нас будут интересовать соотношения точности и скорости поиска. На графиках ниже можно встретить две метрики точности «Search» и «Recognition». Ломанная «Search» обозначает процент правильно найденных лиц среди k ближайших соседей в одном запросе. Ломанная «Recognition» обозначает процент правильно распознанных людей во всех запросах, где успешным распознаванием считается совпадение лица запроса с лицом наибольшей частоты встречаемости среди k ближайших соседей.

**Точный поиск**

Для сравнения результатов исследования проведем замеры точности поиска по евклидову расстоянию (Рис. 5). Суть эксперимента в том, что данные не проходят предварительной обработки, а поиск происходит по всем имеющимся в базе данных векторам. Формируется общий ассоциативный список расстояний, который после сортировки показывает ближайшие к запросу вектора. Точность этого алгоритма будем считать эталонной, так как алгоритмы ANN не могут превысить точности исчерпывающего поиска.

Можно заметить и в чем мы убедимся позже, что вторая кривая находится в некоторой окрестности постоянного значения (0.90-0.92). Это означает, что с расширением объема поиска процент лиц соответствующих запросу держится на одном уровне. «Search» кривая явно убывает и связано это с тем, что при увеличении количества ближайших соседей вероятность попадания посторонних лиц в их число также увеличивается.

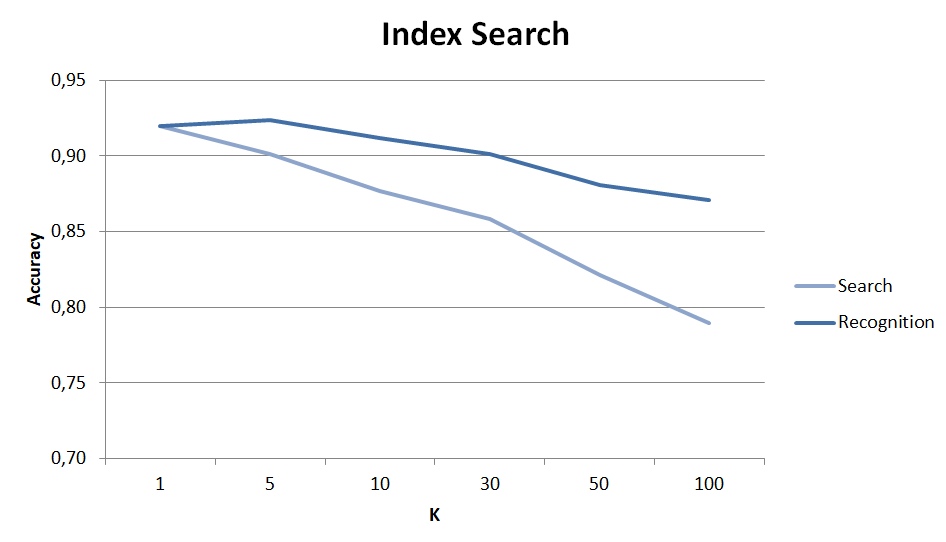
Перейдем к рассмотрению более продвинутых поисковых структур.

**Индексная структура**

Инвертированный индекс — самый простой способ индексирования данных, но тем не менее даже он дает большой прирост в скорости поиска без ощутимого снижения качества. Для его использования требуется предобработка данных. Заметим, что из всех индексных структур, этот алгоритм требует наибольшего времени предобработки, так как для построения индексной структуры используется классический алгоритм k-средних. Для исследуемых объемов данных он очень ресурсоемкий. Поиск по данной структуре разбивается на два этапа: 1) поиск ячейки; 2) поиск внутри ячейки. Для минимизации времени поиска требуется правильно подобрать параметры разбиения пространства. Путем поиска минимума функции F(k) — общее количество вычислений расстояния, добиваемся оптимальной сложности поика:

F=k + N/k,

где N — объем базы данных, k — количество ячеек разбиения.



Результаты исследования индексной структуры (Рис.6) дают похожую картину точности, за тем лишь исключением, что общий показатель точности распознавания несколько упал (0.88-0.91). Теперь при увеличении количества ближайших соседей «Search» кривая убывает быстрее. Это связанно с тем, что ближайшие к запросу центроиды не всегда точно определяют истинных ближайших соседей. Рассмотрение большего количества смежных ячеек улучшит результат, но негативно скажется на скорости поиска.

Иерархическая структура (описание+тесты)

Мультииндексная структура (описание+тесты)

Общий результат

Пути улучшения

**Перевернутый мультииндекс и мульти-D-АЦП.**

Inverted Multi-Index (IMI) [2] - это алгоритм индексации для многомерных пространств и очень больших наборов данных. IMI обобщает инвертированный индекс с помощью кодовых книг продукта для построения центроидов клеток (обычно рассматривается только два компонента в продукте). Таким образом, Inverted Multi-Index имеет две D 2-мерные кодовые книги произведений для разных половин вектора, каждая из которых содержит K субкодовых слов, что позволяет эффективно создавать ячейки K2, которые, как правило, на порядки превышают количество ячеек в IVFADC. система или другие системы, использующие инвертированные индексы. Большое количество ячеек обеспечивает очень плотное разделение пространства, а это означает, что для достижения высокого уровня отзыва (например, истинного ближайшего соседа) необходимо пройти небольшую часть набора данных. Для сжатия набора данных [2] следовал системе IVFADC и использовал Квантование продуктов с глобальными кодовыми книгами, общими для всех ячеек, чтобы кодировать смещения векторов от центроидов (эта система называется Multi-D-ADC).

**Исследование и построение решений задачи**

* CNN (почему CNN и какая использовалась)
* Датасет (Современный датасет VGGFace2)
* Почему 128 мерные вектора?
* ADC (Сравнение вычисления расстояния и почему именно это выбрано)
* Почему важно брать кандидатов не из одной ячейки
* Эксперименты:
  + Время поиска для разных k (один график Time от K)
  + Точность поиска для разных k (один график [Recall@](mailto:Recall@R)K от K)
  + Общая таблица результатов
* Пути улучшения (LOPQ)

**Описание практической части**

* Последовательность обучения и поиска.
* Структуры применяемые на разных этапах

Последовательность обучения и поиска

Индексирование вектора y происходит следующим образом:

1) квантование y в qc (y)

2) вычисление остаточного r (y) = y - qc (y)

3) квантование r (y) в qp (r (y)), которое для квантователя продукта - это присвоение uj (y) qj (uj (y)) для j = 1. , , м.

4) добавить новую запись в инвертированный список, соответствующий qc (y). Он содержит идентификатор вектора (или изображения) и двоичный код (индексы квантователя продукта).

Поиск ближайшего соседа (ей) запроса x состоит из

1) квантования x его w ближайшим соседям в кодовой книге qc; Для наглядности на следующих двух шагах мы просто обозначим через r (x) остатки, связанные с этими w-назначениями. Два шага применяются ко всем w назначениям.

2) вычислить квадрат расстояния d uj (r (x)), cj, i 2 для каждого субквантизатора j и каждого из его центроидов cj, i;

3) вычислить квадрат расстояния между r (x) и всеми индексированными векторами инвертированного списка. Используя расстояния между субвекторами и центроидами, вычисленные на предыдущем шаге, это заключается в суммировании m искомых значений, см.

4) выберите K ближайших соседей x на основе расчетных расстояний. Это эффективно реализуется за счет поддержания структуры Maxheap с фиксированной емкостью, которая хранит наименьшие значения K, которые мы когда-либо видели. После каждого расчета расстояния идентификатор точки добавляется в структуру, только если ее расстояние меньше наибольшего расстояния в Maxheap

**Заключение**

**ПОЛЕЗНЫЕ ВЫДЕРЖКИ**

**Статья (2)**

Квантователи решетки предлагают лучшие свойства квантования для равномерного распределения векторов, но это условие редко выполняется векторами реального мира. На практике эти квантователи выполняют значительно хуже, чем k-средних в задачах индексации [22]

В нашем методе используется аналогичное количество операций поиска в таблице, что обеспечивает несравнимую эффективность. Полное сравнение вектора запроса со всеми кодами запрещено для очень больших наборов данных. Поэтому мы вводим модифицированную инвертированную файловую структуру для быстрого доступа к наиболее значимым векторам. Грубый квантователь используется для реализации этой инвертированной файловой структуры, где векторы, соответствующие кластеру (индексу), сохраняются в связанном списке. Векторы в списке представлены короткими кодами, вычисленными нашим квантователем продуктов, который используется здесь для кодирования остаточного вектора относительно центра кластера

Б. Квантователи произведений. Рассмотрим 128-мерный вектор, например, дескриптор SIFT [23]. Квантизатор, создающий 64-битные коды, то есть «только» 0,5 бит на компонент, содержит k = 264 центроида. Следовательно, невозможно использовать алгоритм Ллойда или даже HKM, так как количество требуемых выборок и сложность изучения квантователя в несколько раз превышают k. Даже невозможно сохранить значения с плавающей запятой D × k, представляющие k центроидов. Квантование продукта является эффективным решением для решения этих проблем. Это обычная методика в исходном кодировании, которая позволяет выбирать количество компонентов, подлежащих квантованию совместно (например, группы из 24 компонентов могут быть квантованы с использованием мощной решетки пиявки). Входной вектор x разбит на m различных подвекторов uj, 1 ≤ j ≤ m размерности D ∗ = D / m, где D кратно m. Субвекторы квантуются отдельно с использованием m различных квантователей. Поэтому данный вектор x отображается следующим образом: x1, ..., xD ∗ | {z} u1 (x), ..., xD − D ∗ + 1, ..., xD | {z} um (x) → q1 u1 (x)), ..., qm (um (x), (8) где qj - квантователь с низкой сложностью, связанный с j-м субвектором. связать набор индексов Ij, кодовую книгу Cj и соответствующие значения воспроизведения cj, т. е. значение воспроизведения квантователя продукта идентифицируется элементом набора индексов продукта I = I1 × ... × Im. Поэтому кодовая книга определяется так как декартово произведение C = C1 ×.. × Cm, (9) и центроид этого множества является конкатенацией центроидов субквантизаторов m, то теперь мы предполагаем, что все субквантователи имеют одинаковое конечное число k ∗ значения воспроизводства. В этом случае общее число центроидов определяется как k = (k ∗) m. (10) Обратите внимание, что в экстремальном случае, когда m = D, компоненты вектора x все квантованы по отдельности. продукт квантования оказывается скалярным квантователем, где функция квантования, связанная с каждым компонентом, может быть разной.

, Преимущество квантователя продукта состоит в том, чтобы производить большой набор центроидов из нескольких небольших наборов центроидов: связанных с субквантователями. При изучении субквантователей с использованием алгоритма Ллойда используется ограниченное число векторов, но кодовая книга в некоторой степени все еще адаптирована к представлению данных для представления. Сложность изучения квантователя в m раз больше сложности выполнения кластеризации k-средних с k ∗ центроидами размерности D ∗

Поскольку подпространства ортогональны, квадрат искажения, связанный с квантователем произведения, равен MSE (q) = X j MSE (qj), (11) где MSE (qj) - искажение, связанное с квантователем qj. На рисунке 1 показано MSE как функция длины кода для различных (m, k ∗) наборов, где длина кода равна l = m log2 k ∗, если k ∗ - степень двойки. Кривые получены для набора 128-мерных дескрипторов SIFT, подробности см. В разделе V. Можно заметить, что для фиксированного числа битов лучше использовать небольшое количество субквантователей со многими центроидами, чем иметь много субкванизаторов с несколькими битами. В крайнем случае, когда m = 1, квантователь продукта становится обычной кодовой книгой k-средних.

Литература (По номерам статей из файла TextNotes):

[1] Jegou, H., Tavenard, R., Douze, M., Amsaleg, L.: Searching in one billion vectors: Re-rank with source coding. In: ICASSP. (2011)

[2]H. Jegou, M. Douze, and C. Schmid, “Product quantization for ´ nearest neighbor search,” TPAMI, vol. 33, no. 1, 2011.

[3]A. Babenko and V. Lempitsky, “The inverted multi-index,” 2014 IEEE

[4]Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization. Technical report (2013)

[5]Kalantidis, Y., Avrithis, Y.: Locally optimized product quantization for approximate nearest neighbor search. In: in Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR 2014), IEEE (2014)

[6]Babenko, A., Lempitsky, V.S.: Efficient indexing of billion-scale datasets of deep descriptors. In: CVPR. (2016)

[7]Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors Dmitry Baranchuk1,2 , Artem Babenko1,3 , Yury Malkov4

[8]Wang, D., Otto, C., Jain, A.K.: Face search at scale. TPAMI (2017)

[9] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, “Modeling LSH for performance tuning,” CIKM, 2008. 2, 4

[10]D. Yi, S. Liao, and S. Z. Li, “Learning face representation from scratch,” arXiv:1411.7923v1, 2014

Двухуровневая индексация

[11] D. Nister and H. Stew ´ enius, “Scalable recognition with a vocab- ´ ulary tree,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2161–2168, 2006

[12]A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces Weber, Roger and Schek, Hans-Jrg and Blott, Stephen