САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ - ПРОЦЕССОВ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ И МНОГОПРОЦЕССОРНЫХ СИСТЕМ

Кулинкин Антон Борисович

Выпускная квалификационная работа бакалавра

Нейросетевое детектирование объектов в условиях ограниченного времени

Направление 010400

Прикладная математика и информатика

Заведующий кафедрой,
доктор физ.-мат. наук,
профессор Андрианов С. Н.

Научный руководитель,
проф. Андрианов С. Н.

Рецензент:
Генеральный директор ООО "Kuznech" Погребняк М. Л.

Санкт-Петербург
2016

Содержание

Введение

Теория обработки цифровых изображений — один из основных разделов в информатике. В рамках него разработаны алгоритмы сжатия графических данных, их анализа, машинного зрения. Человек давно пытается научить компьютер понимать, что находится на изображении. То что для нас кажется задачей вполне тривиальной, разработанным алгоритмам все еще не под силу. Хотя уже имеются примеры создания методов, составляющих вербальное описания сцен [1]. Однако для начала нужно решить более простые проблемы, а именно, необходимо научиться находить ключевые объекты реального мира на изображении и понимать, что это за объекты. Первая задача также называется задачей детектирования. Определение конкретного класса предмета на изображении называется задачей классификации.

К решению задач детектирования и классификации применяются различные подходы: статистические, специально разработанные теории ключевых точек, генетические алгоритмы. Другим методом является машинное обучение, зачастую использующее в работе дополнительно один из предыдущих. Тем не менее есть примечательный независимый инструмент — нейронные сети.

В контексте машинного обучения нейронные сети являются наиболее перспективным методом для детектирования и классификации объектов на изображении. Нейронные сети построены по принципу моделирования взаимодействия биологических нейронов в коре головного мозга живых организмов. Некоторые модели кроме общих представлений о работе мозга используют и более специфические знания о, например, устройстве зрительной коры, и реализуются они в свёрточных нейронных сетях [2].

Для оценки качества алгоритмов детектирования и классификации собраны базы данных, содержащие изображения и вспомогательную информацию об объектах на них. Самая большая — ImageNet включает в себя более 14 миллионов изображений и 20 тысяч классов объектов [3]. Каждый год на ней проводится соревнование ImageNet Large

Scale Visual Recognition Challenge (ILSVRC), в котором выявляются лучшие алгоритмы в решении задачи детектирования и классификации. И если еще 6 лет назад большая часть алгоритмов была построена на идее ключевых точек и их дескрипторов [4], то сейчас участники соревнования активно развивают нейросетевые модели, показывающие существенный прирост по всем метрикам качества [5].

Однако несмотря на возросшую за последние несколько лет популярность нейронных сетей и значительный прогресс в оптимизации их быстродействия и качества работы, их применение в реальных задачах все еще ограничено. Это связано как с тем, что работа алгоритмов требует существенных вычислительных ресурсов, так и с тем, что обучение сети — их главное отличие от классических алгоритмов, не программируется и не задается фиксированным образом, это вносит элемент случайности в решение поставленных задач. Обучаясь на тренировочных данных, сеть, исключительно за счет своего внутреннего устройства, выделяет общие и частные признаки объектов, которые на стадии тестирования используются для поиска и классификации объектов на тестовом изображении. Сложность заключается именно в поиске оптимального внутреннего устройства, процесс этот не алгоритмизируем и требует существенного опыта и, возможно, даже удачи. Часть представленной работы будет посвящена поиску архитектурных изменений для двух сетей и описанию достигнутых результатов.

Другой модной тенденцией последнего времени стали онлайн-магазины. Конкуренция заставляет их предлагать новый функционал, повышающий удобство выбора и покупки товаров. Так, один магазин одежды и аксессуаров обратился в компанию, где проводилось представленное исследование, с просьбой разработать систему, позволяющую пользователю искать похожие предметы по фотографиям, сделанным камерой смартфона или загруженным из интернета. Руководство магазина справедливо полагало, что подобная возможность сможет привлечь больше покупателей. Описывали же будущую систему следующим образом: это должно быть мобильное приложение, в которое пользователь может загружать фотографии (или делать прямо

в нем) интересующих его предметов одежды, получая в ответ список наиболее похожих товаров из ассортимента магазина с ценами и прочей вспомогательной информацией.

При детальном рассмотрении задачу можно разделить на подзадачи:

- разработка мобильного приложения;
- разработка системы поиска и классификации объектов на фотографии;
- разработка поиска похожих объектов;
- создание сервиса для взаимодействия с мобильным приложением.

Настоящая работа будет посвящена второму и четвертому пунктам, поэтому подробнее опишем связанные с ними требования, поступившие от заказчика. Были выставлены ограничения не только на качество системы, но и на время работы. Основной целевой аудиторией станут пользователи мобильных устройств, которых легко потерять большим временем ожидания результатов. На качество детектирования и классификации объектов условия даны в двух метриках: полноты (recall) и точности (precision). Первая характеризует количество правильно найденных объектов относительно общего количества интересующих нас объектов на изображении. Точность же показывает, сколько объектов правильно детектировалось среди всего множества найденных предметов.

Эталонными стали следующие значениями:

- Recall $> 80\%$;
- Precision $> 80\%$;
- Время работы < 1 сек.

Другое требование, выдвинутое заказчиком и не относящееся напрямую к взаимодействию с пользователем, — расширяемость построенной системы. Под расширяемостью подразумевается возможность быстрой адаптации к изменяющемуся ассортименту магазину. Дополнительно

решено реализовать возможность простой смены алгоритмов на случай, если появится необходимость задействовать в приложении более современные и оптимизированные их версии.

Постановка задачи

Строго формализуя предъявленные требования, получаем, что:

- необходимо построить систему детектирования и классификации предметов одежды;
- типы объектов, с которыми работает система, определяет заказчик;
- показатель полноты поиска и классификации должен превышать 80%;
- показатель точности поиска и классификации должен превышать 80%;
- пользователь должен получать результат по своему запросу не более чем за 1 секунду при условии стабильного доступа в интернет;
- необходима расширяемость системы для работы с новыми типами объектов, которые могут появиться в ассортименте заказчика;
- требуется предусмотреть переход на более совершенные алгоритмы поиска и классификации объектов;
- требуется разработать API для взаимодействия мобильного приложения с основной системой.

Глава 1. Обзор литературы

Задачей детектирования и распознавания объектов на изображения занимаются с конца прошлого века, поэтому первым делом были изучены уже существующие подходы к её решению.

1.1. Поиск объектов на изображении

Детектирование объектов — нахождение предметов реального мира: лиц, велосипедов, зданий и т. п. на изображениях и видео. Несмотря на то, что для человека поиск и распознавание объекта происходит крайне легко, для компьютерных же систем эта задача до сих пор решается не всегда и не полностью. Множество разных подходов было разработано в течение нескольких десятилетий.

Методы на основе внешнего вида объектов, такие как сопоставление границ [6, 7], сопоставление градиентов [8], статистические методы на основе гистограмм [9] в целом имеют много недостатков. Используя для «обучения» примеры объектов интересующего класса, методы являются не устойчивыми к изменению освещения или цвету искомых образцов. Также объекты одного класса могут иметь множество форм, а с разных ракурсов выглядеть по-разному. Все это не позволило бы успешно применять алгоритмы, основанные на внешнем виде объекта, в задаче поиска предметов одежды. С одной стороны, в этой категории наблюдается невероятное разнообразие по цвету и форме, с другой, предметы абсолютно не похожие внешне, могут принадлежать к одному типу.

Второй класс алгоритмов основан на поиске особых точек на изображении. Здесь из обучающих данных получается набор особых точек с помощью одного из методов описания областей на изображении, например, SIFT [10] или SURF [11]. Далее по изображению, где требуется найти объект, строится аналогичный набор особых точек и их дескрипторов. Оба набора сравниваются на близость [12], и в случае удовлетворительного совпадения объект считается найденным. В от-

личие от упомянутых ранее, эти методы обладают большей устойчивостью к изменению масштаба объектов, их внешнему виду. Происходит это за счет того, что зачастую особыми точками считаются углы, прямые линии или характерные текстуры. Проблемы начинаются при поиске объекта необычной формы и определении степени похожести множества особых точек. Кроме того, согласно статье [11], получение небольшого числа ключевых точек требует порядка 610 миллисекунд, что достаточно близко к максимальному времени работы предполагаемой системы. Из этих соображений от данного класса методов также решено было отказаться.

Еще несколько лет назад, уложиться во временные рамки, установленные заказчиком, было бы очень сложно или практически невозможно, имея в виду еще и показатели качества. Но с развитием популярности нейронных сетей и идеи математических вычислений на GPU, это стало возможным. В наиболее современных статьях по теме поиска и распознавания предметов одежды используются глубокие сверточные нейронные сети на основе окон кандидатов (Regions with convolutional neural networks features — R-CNN) [13–15]. И хотя в указанных статьях авторы не достигли требуемой точности, виден потенциал их подхода. В условиях, когда нейронные сети непрерывно совершенствуются, повышая качество работы и быстродействие, было решено углубиться в поиск подходящей к нашим условиям модели. Ею стала Fast R-CNN [16]. Подробное описание будет представлено в соответствующем разделе.

1.2. Предложение окон-кандидатов

Модели нейронных сетей на основе окон-кандидатов детектируют объекты не на целом изображении, а только в некоторых его областях. Выбор этих областей оставлен на усмотрение пользователя сети. В общем следует выбирать не случайные области, а те, которые потенциально могут содержать интересующие нас объекты. Для этого предложено несколько методов с абсолютно разными концепциями, лежащими в их основе: Selective Search [17], Geodesic Object Proposals [18], Edge

Boxes [19], MultiBox [20].

В статьях [13, 16] использовался Selective Search, но согласно авторам, поиск достаточного количества окон занимает приблизительно 2,6 секунды, что в два с половиной раза превышает установленные временные рамки.

Следующий метод, Geodesic Object Proposals, эффективно строит сегментацию входного изображения менее, чем за секунду. Однако качество быстрой реализации алгоритма не удовлетворяет поставленной цели: 62% против требуемых минимум 80%. Поиски были продолжены.

Edge Boxes — метод основан на анализе границ объектов на изображении. Имеет впечатляющие показатели качества и времени работы. Авторы заявляют, что установив критерием принятия окна половину пересечения площади искомого объекта (т.е. если окно содержит в себе по крайней мере половину объекта, мы считаем, что он найден), можно добиться полноты поиска в 96%, а скорости работы в 0,25 секунды. Edge Boxes целиком удовлетворял требованиям к системе, при этом оставляя запас для работы детектора и вспомогательных процедур.

Тем не менее, оставался еще один не менее перспективный метод — MultiBox. В статье не указано качество в метрике полноты, поэтому было проведено собственное тестирование. Результаты показали, что выдавая удовлетворительную точность, MultiBox работает почти в 6 раз быстрее EdgeBox. Другим доводом в пользу MultiBox стало то, что это так же нейросетевой метод, как и Fast R-CNN, а значит их можно оптимизировать параллельно. Что и было сделано в дальнейшем.

1.3. Взаимодействие с мобильными устройствами и web API

В области построения API существует сложившийся и общепринятый стиль под названием REST (Representational State Transfer) [21]. Под этим подразумевается клиент-серверная система, удовлетворяющая некоторым ограничениям. Существенными в контексте рассмат-

риваемой задачи будут следующие:

- на хранение состояния — каждый запрос должен обрабатываться независимо от предыдущих и содержать в себе всю необходимую для обработки информацию;
- ресурсы приложения и их представление для передачи пользователю должны быть независимы;
- представление ресурсов должно быть унифицированным, т. е. использовать один из популярных стандартов представления данных, например, JSON;
- каждое сообщение должно быть достаточно информативно для того, чтобы описать каким образом его обрабатывать.

Доступ к API происходит через HTTP запросы. В роли HTTP сервера выступает nginx [22], а обмен данными с приложением идет по протоколу Fast CGI [23].

Глава 2. План реализации системы

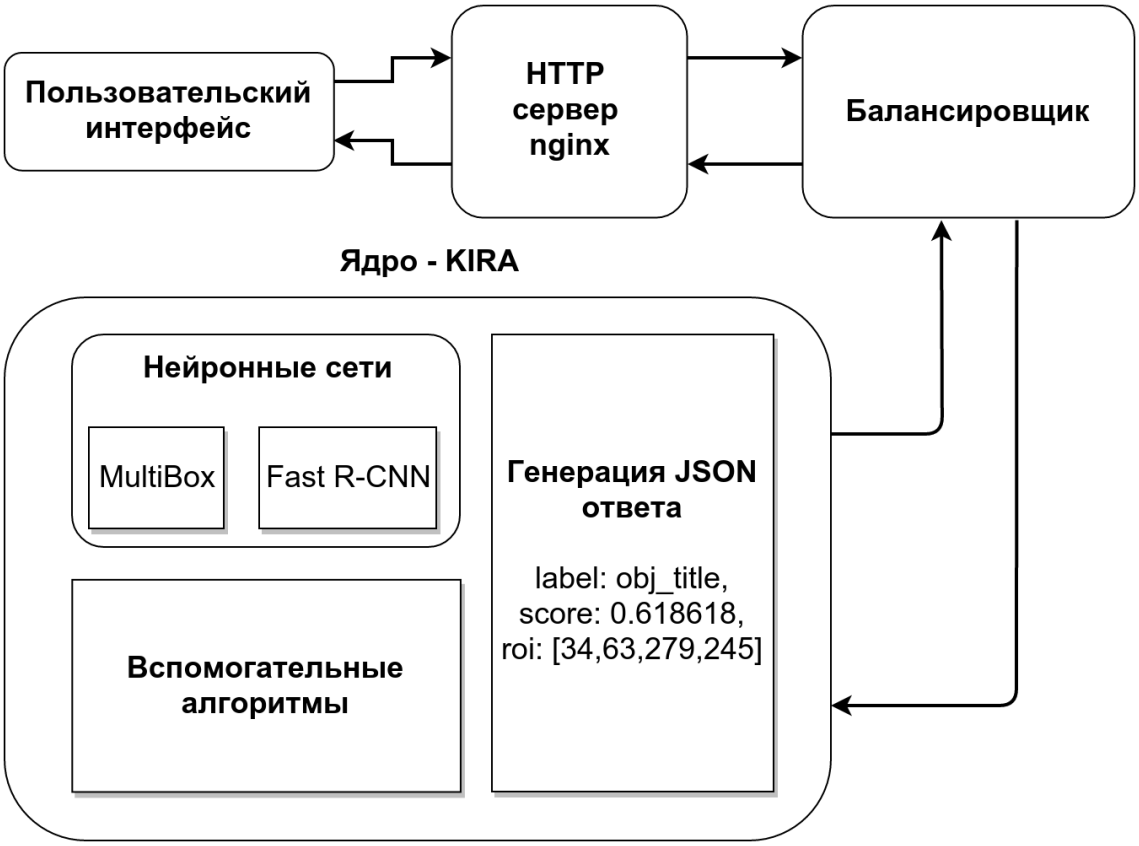

Рис. 1. Схематичное изображение системы

Изучив имеющиеся решения и стандартные походы к поставленной задаче, был создан предварительный набросок системы. Его схематичное представление изображено на рисунке 1.

Здесь мы видим разделение системы на 4 крупных логических части: клиентская сторона — мобильное или web приложение; nginx — HTTP сервер, принимающий запросы от пользователей; интерфейс API и балансировщик — он будет следить за количеством запросов, обработкой служебных и пользовательских вызовов, а также, в будущем, перераспределять запросы с учетом нагрузки последующих узлов; ядро KIRA (Kuznech Image Recognition Architecture) — под его управлением будет происходить основная обработка пользовательских данных.

Так как для поиска и распознавания объектов решено использовать нейронные сети, требовалось определиться со способом встраивания их в систему. На момент проведения исследований автор имел опыт рабо-

ты с двумя фреймворками, готовыми для использования в коммерческих приложениях: cuda-convnet [24] и Caffe [25].

Ядра обоих фреймворков написаны на C++ с вынесением большинства вычислений на GPU, но первый имеет только Python интерфейс. Это обстоятельство делает затруднительным встраивание cuda-convnet в код, написанный на C++, т.к. код для взаимодействия с Python углубляется на несколько абстракций вниз. Оценив количество усилий, необходимых для создания C++ интерфейса, было решено отказаться от cuda-convnet в пользу Caffe.

Решение оказалось правильным. Фреймворк Caffe, разработанный в университете Беркли, за последние несколько лет добился значительной популярности в научной среде. Многие авторы при публикации своих статей предоставляют доступ к исходному коду методов, встроенных в этот фреймворк. Также у Caffe имеется простой интерфейс на нескольких языках: Python, Matlab, C++. Для прототипирования, обучения и тестирования сетей удобно использовать интерфейс к скриптовым языкам, а финальную разработку приложения производить на относительно низкоуровневом и быстром C++.

2.1. Система детектирования предметов одежды

В этом разделе будут рассмотрены нейронные сети, использующиеся для детектирования целевых объектов, и модификации, благодаря которым удалось повысить качество и скорость работы сетей, описанных в оригинальных статьях.

В качестве первичного инструментария был выбран самый быстрый метод детектирования объектов на мент проведения исследований — Fast R-CNN, который состоит из алгоритма поиска областей (окон) на изображении, с наибольшей вероятностью содержащих объекты, и нейронной сети, используемой для классификации каждой из таких областей.

Обычно [16,26,27] в качестве алгоритма поиска окон-кандидатов используется метод Selective Search, но время его работы слишком велико (2,6 сек.) и не удовлетворяет предъявленным требованиям, поэтому использовалась модифицированная версия метода MultiBox, который даёт лучшую точность детектирования (0,417 mAP в задаче ILSVRC-2014 Detection против 0,413 mAP у Selective Search) при меньшем времени работы.

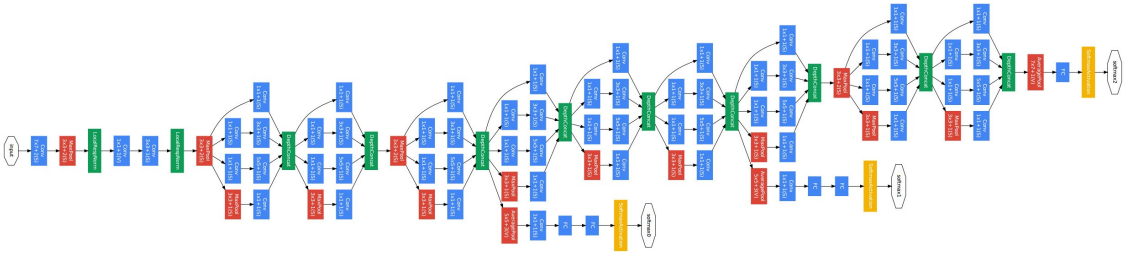

Рис. 2. Архитектура сети GoogLeNet

В результате модификации сетей MultiBox и Fast R-CNN удалось:

- достичь необходимой точности в условиях нехватки обучающих данных;
- построить систему, в 10 раз превосходящую по скорости метод Fast R-CNN + Selective Search, описанный в [16];
- инициализировать модели сетью GoogLeNet (рис. 2) [28], что позволило не только увеличить скорость обучения, но и повысить итоговое качество детектирования;
- повысить устойчивость к изменениям входных данных и общее качество работы системы.

Построенная система детектирования состоит из трех блоков: поиск объектов, классификация предложенных окон-кандидатов, постобработка результатов детектирования. Далее каждая часть будет описана подробно.

2.1.1. Поиск объектов

В основе предложенного метода генерации окон-кандидатов лежит MultiBox. Это нейронная сеть, на вход получающая изображение и на

выходе выдающая для N окон-кандидатов их координаты и значения уверенности сети в том, что данное окно содержит объект. В оригинальной статье в качестве функции ошибки для обучения предсказанию координат использовалось евклидово расстояние (L_2 $Loss$) [20]:

$$F_{match}(x, l) = \frac{1}{2} \sum_{i,j} x_{ij} \left\| l_i - g_j \right\|_2^2 . \tag{1}$$

В ходе исследования было обнаружено, что при замене функции ошибки на $Smooth$ L_1 $Loss$ [16]:

$$L_{loc}(t^u, v) = \sum_{i \in x,y,w,h} smooth_{L_1}(t_i^u - v_i), \tag{2}$$

$$smooth_{L_1}(x) = \begin{cases} 0,5x^2, \ \|x\| < 1, \\ \|x\| - 0,5, \ \|x\| \geqslant 1, \end{cases} \tag{3}$$

разметка исходного изображения становится более равномерной — в первом случае большая часть окон лежала в области центра, что оставляло вне поля зрения объекты, находящиеся по краям изображения. Архитектура выходных слоёв сети представлена на рисунке 3. Кроме того, для ускорения тренировки модель инициализировалась весами сети GoogLeNet, обученной задаче классификации изображений ImageNet [28], и дообучалась на поставленной задаче. В процессе использования обученной сети для отсеивания окон, явно не содержащих объектов, установлен настраиваемый порог по уверенности. Для увеличения количества релевантных окон-кандидатов и улучшения точности детектирования маленьких объектов MultiBox может применяться к пирамиде изображений, построенной на основе исходного. В работе используется пирамида из 14 изображений (см. рисунок 4).

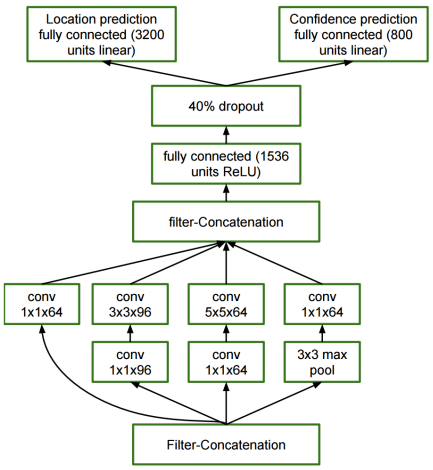

Рис. 3. Верхние слои сети MultiBox

Рис. 4. Пирамида входных изображений Multibox

2.1.2. Классификация окон-кандидатов

В результате работы MultiBox генерируется определённое количество окон-кандидатов. Теперь требуется определить для каждого из окон содержит ли оно объект, и если да, то какого он класса. Для этого используются свёрточные нейронные сети [2]. В ранних версиях модели R-CNN [13] каждое из полученных окон-кандидатов просто приводилось к одному и тому же размеру и пропускалось через нейронную сеть, которая выдавала вектор из $N + 1$ значений, соответствующий вероятностям принадлежности данного окна к каждому из N классов или же «фону» (так обозначается отсутствие объекта). В случае, когда требуется оценить большое количество окон (а Selective Search, например, выдаёт более 1000 окон), это приводило к значительным временным затратам. Отличие модели Fast R-CNN в том, что в ней через нейронную сеть пропускается не каждое окно в отдельности, а всё изобра-

Рис. 5. Пример искажений при обучении

жение целиком. Проходя через последовательность свёрточных слоёв до верхних уровней сети, изображение превращается в набор карт признаков (feature maps). Окна-кандидаты в этом случае вырезаются не из исходного изображения, а из этих карт признаков, и пропускаются поотдельности только через самые верхние слои сети. Таким образом достигается значительное ускорение работы сети, так как вместо тысячи проходов изображения через сеть требуется совершить лишь один. Кроме того, благодаря использованию MultiBox для генерации окон–кандидатов, можно, настроив пороговое значение уверенности, заранее сократить их количество с тысяч до сотен или десятков, что также значительно ускоряет работу системы. Также в отличие от версии Fast R-CNN, описанной в статье [16], за основу взята более быстрая архитектура сети — GoogLeNet, которая, как и в случае с MultiBox, была обучена на задаче классификации ImageNet и дообучена уже на поставленной задаче детектирования. Кроме того, в связи со сравнительно малым размером обучающей выборки (по 100–150 изображений на каждый класс), активно использовались разнообразные искажения для увеличения количества обучающих данных: зеркалирование, растяжения изображения по ширине и высоте, вырезание случайных под-областей и т. п. Пример для одного предмета показан на рисунке 5.

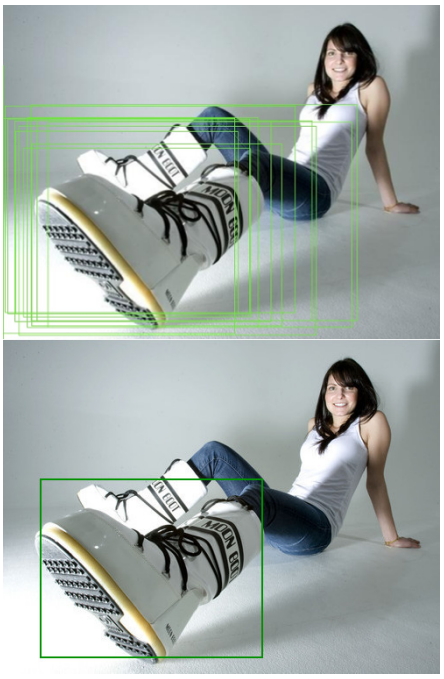

Рис. 6. Пример работы NMS

2.1.3. Пост-обработка результатов

На стадии пост-обработки результатов для избавления от лишних детектирований одного и того же объекта применялся метод Non Maximum Suppression (NMS) [13] (см. рис. 6): среди значительно пересекающихся между собой областей, классифицированных как имеющие объекты одного и то-

го же класса, выбирались области, имеющие наивысшую вероятность содержания объекта этого класса.

2.2. Программное обеспечение детектора

Завершив подготовку выбранных нейронных сетей и методов детектирования объектов, необходимо было реализовать систему, под управлением которой будет происходить обработка пользовательских данных. Основными требованиями, предъявляемыми к ней стали: гибкость, модульность, отказоустойчивость. Так как приложение разрабатывалось для реального коммерческого применения, отказоустойчивость его важна ни чуть не меньше, чем расширяемость — внутреннее качество системы, скрытое от глаз пользователя.

На рисунке 1 обозначены отдельные части разрабатываемой системы. В этом разделе будет рассмотрено создание интерфейса API и балансировщика, ядра, а также приведены краткие примеры конфигурации HTTP сервера nginx, использующегося для взаимодействия клиентского приложения с сервисом.

Языком разработки выбран C++ за его быстродействие в сравнении с ныне популярными в enterprise-разработке Java и C#. Помимо этого, фреймворк Caffe, который решено было использовать в качестве площадки для запуска нейронных сетей, предоставляет простой и интуитивно понятный интерфейс на C++, что стало еще одним доводом в пользу выбора этого языка, как рабочего инструмента.

В процессе разработки использовались некоторые сторонние библиотеки. Среди них:

- OpenCV [29] — для пред-обработки изображений перед отправкой их в сеть
- Boost [30] — предоставляет множество расширений STL (стандартной библиотеки C++), использовался для реализации компиляторонезависимой многопоточности, работы с файловой системой, упрощения кода за счет использования его алгоритмов, логирования информации

- fast-cgi [31] — для обмена данными с между приложением и HTTP сервером

2.2.1. Ядро (KIRA)

KIRA — Kuznech Image Recognition Architecture — система обработки изображений, разработанная в компании Kuznech в ходе решения задачи, которой посвящена настоящая работа. Основная особенность — предоставление пользователю широких возможностей для конфигурации процесса обработки заданием интересующих его методов, их параметров и последовательности выполнения. В файле настроек пользователем описывается структура системы обработки изображений, а при запуске приложения, выстраивается её модель в соответствии с конфигурацией. Взаимодействие с пользователем происходит через простой интерфейс, состоящий из двух функций: инициализации ядра и запуска обработки на предоставленных данных. Результатом работы системы служит строка с результатом детектирования: список объектов, их названия, местоположение на изображение и уверенность сети в принадлежности объекта к выбранному классу.

Архитектура

Рассмотрим UML диаграмму интерфейсов ядра:

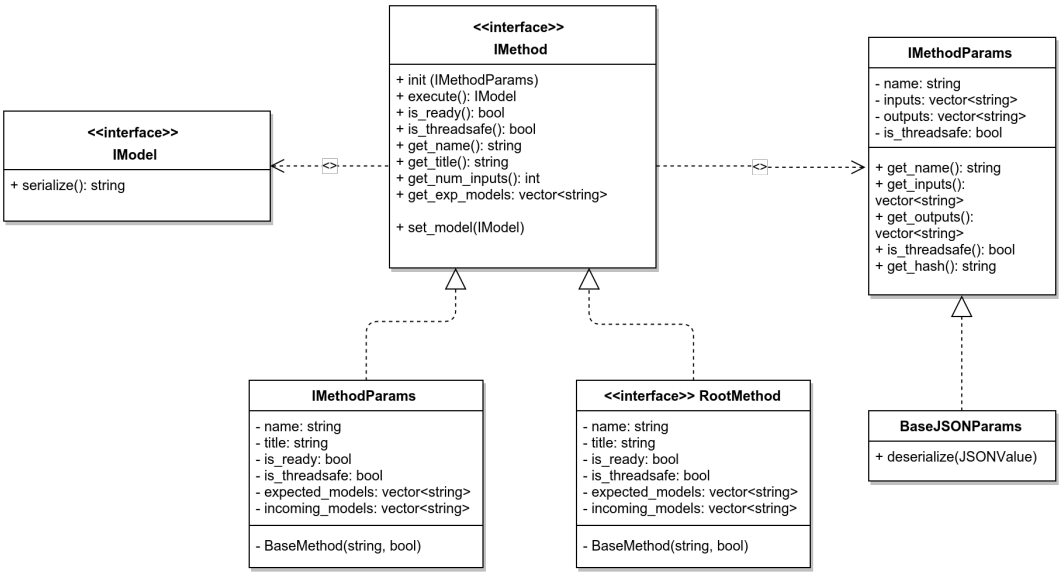

Рис. 7. Диаграмма интерфейсов KIRA

Каждый метод, доступный пользователю для использования в конфигурации, реализует интерфейс абстрактного базового класса IMethod, наследуясь от одного или обоих его прямых предков: BaseMethod и RootMethod. Разделение это сделано намеренно, так как первый метод в цепочке принимает от пользователя данные в произвольном формате. Для приведения их к формату, принятому в KIRA, требуется конвертация — интерфейс для неё предоставляет абстрактный класс RootMethod.

Процесс создания методов инкапсулирован посредством создания отдельного абстрактного класса для параметров метода — IMethodParams. Сделано это в первую очередь для поддержания модульности системы — чем сильнее изолированы части, тем яснее модель их взаимодействия друг с другом, тем меньше кода дублируется и т. д. Если рассматривать данное решение не только в контексте хорошего стиля программирования, но и с учетом ограничений рассматриваемой задачи, это позволяет создавать новые методы и их параметры, ни в коей мере не затрагивая уже реализованные.

Далее, специалисту, поддерживающему приложение, предоставлена свобода в реализации получения параметров от пользователя. Автором был выбран формат JSON за его распространенность и удобство чтения человеком. Абстрактный класс BaseJSONParams предоставляет интерфейс для парсинга пользовательского конфигурационного файла, который будут реализовывать классы параметров для каждого конкретного метода.

Для создания экземпляров классов IMethod и IMethodParams используются абстрактные фабрики [32] с регистрацией созданных методов. При реализации метода или его параметра достаточно добавить вызов одного макроса в тело класса и еще одного - в его реализацию. Классический шаблон проектирования «абстрактная фабрика» требует ручного добавления функции-члена в код фабрики для создания экземпляра каждой реализации базового класса. Для удобства поддержки и уменьшения вероятности совершить ошибку при добавлении новых методов, было задействовано описанное выше архитектурное решение. За

основу взят код [33], доработанный под конкретные особенности разрабатываемой системы.

Следующий важный момент — передача данных внутри системы между экземплярами класса IMethod. Точный тип и набор данных, необходимый для функционирования всех возможных обработчиков был не известен изначально: каждый новый добавленный метод может потребовать дополнительной информации. Поэтому передача данных реализована через интерфейс базового класса IModel, он позволяет абстрагироваться от природы данных для конкретного метода, сосредоточившись на их общих моментах в представлении. Также базовый класс предоставляет интерфейс для сериализации модели. Сериализацией в программировании называется процесс перевода какой-либо структуры данных в последовательность битов, в нашем случае — в JSON.

В процессе работы над KIRA определены две минимально необходимые модели. Первая описывает выход нейронных сетей и постпроцессоров вроде Non Maximum Suppression метода. Вторая описывает результат детектирования системой объектов на изображении, а также позволяет сериализовать себя в JSON следующего вида:

```
''detectedObjects'':
 [
     {
         ''label'':''high_heel_shoes'',
         ''score'':0.618618,
         ''roi'':[34,63,279,245]
     }, ... {}
 ],
```

где:

- detectedObjects — список объектов, найденных на изображении;
- label — класс объекта, наиболее вероятно содержащегося в области roi;
- score — вероятность, с которой объект принадлежит к обозначенному классу;

- roi — координаты области, содержащей объект; первые две компоненты — абсцисса и ордината верхнего левого угла описывающего объект прямоугольника, третья и четвёртая — его ширина и высота соответственно.

Использование

Ниже приведен список части методов, реализованных на момент написания работы в системе KIRA, и их краткое описание:

- B64ToJpeg — RootMethod, конвертирует входящее изображение из кодировки Base64 в формат JPEG. Если на вход принимает абсолютный путь до изображения на жестком диске — передает дальше;
- MultiBox — RootMethod, взаимодействует с фреймворком Caffe, ищет окна, в которых могут быть объекты на изображении;
- FastFeatureExtractor — взаимодействует с фреймворком Caffe, детектирует объекты, содержащиеся в окнах-кандидатах, полученных от MultiBox или другого метода предложений;
- NmsMethod — удаляет пересекающиеся детекты объектов;
- LabelMapper — последний метод, присваивает найденным объектам имена в соответствии с конфигурацией.

Пара слов о работе с фреймворком Caffe

Прежде чем продолжить описание KIRA, требуется углубиться в тонкости работы с Caffe. В фреймворке для конфигурации сетей используются по крайней мере два файла:

- .prototxt файл — содержит читаемое описание модели нейронной сети в удобном для её конфигурации формате;
- .caffemodel файл — содержит бинарное представление весов нейронов обученной модели, а также некоторые вспомогательные данные.

Помимо двух основных файлов Caffe может использовать вспомогательные. Из применяемых на данный момент:

- prior_boxes — содержит список prior boxes для сети MultiBox, их концепция описана в статье [20];
- classes — для сети детектора. Содержит список классов, на которых обучена сеть, и их ID. Должен быть представлен в формате:

<id класса><один пробельный символ><метка класса>.

Использование. Продолжение

Каждый метод имеет от 3 до 10 параметров, ниже приведены таблицы, их описывающие.

Общие параметры методов

Их всего три. Отвечают за создание и связывание методов в цепочке обработки между собой.

Таблица 1. Общие параметры методов

Имя параметра	Описание	Значения
name	Имя метода	В соответствии со списком методов выше
input	Список методов, от которых текущий получает данные	Список имен в соответствии с конфигурацией
output	Список методов, которым метод возвращает данные	Список имен в соответствии с конфигурацией

Параметры Caffe методов

Их тоже можно разделить на группы: MultiBox методы, FeatureExtractor методы, их Fast версии: Таблицы 2, 3, 4.

Таблица 2. Общие параметры Caffe методов

Имя параметра	Описание	Значения
mode	Режим работы Caffe: на GPU или CPU	GPU, CPU
net_path	Путь до .prototxt файла	
model_path	Путь до .caffemodel файла	
feature_layer	Название слоя, с которого будут получаться признаки	Обычно - имя последнего слоя в .prototxt файла
device_id	ID GPU карты	Обычно 0
pr_size	Длина в пикселях, к которой будет приводиться наименьшая сторона изображения. Наибольшая изменится с сохранением пропорций	224 для MultiBox и медленной R-CNN, 600 и больше для Fast R-CNN

Таблица 3. Параметры MultiBox

Имя параметра	Описание	Значения
threshold	Порог уверенности, при котором найденное окно помечается как содержащее объект	Обычно 0.0001 и меньше
with_conf	Получать ли значения уверенности от сети	Обычно true
prior_path	Путь до priors.txt	
part_coords	Относительные координаты частей изображения, на которых будет запущен MultiBox	Список списков из трех элементов: относительное значение x верхнего левого угла части, аналогично y, длина стороны части относительно размеров изображения

Таблица 4. Параметры FastFeatureExtractor

Имя параметра	Описание	Значения
max_size	Максимальная длина наибольшей стороны при растяжении наименьшей к pr_size. Если наибольшая превысит max_size, изображение будет сжато к max_size по наибольшей стороне, наименьшая сжимается пропорционально	$\leqslant 1500$
bbox_layer	Слой bounding box regression коэффициентов [16]	Берется из .prototxt: обычно bbox_pred

Параметры Label Mapper

У всех один параметр: labels_csv - путь до файла с классами.

Параметры Non Maximum Suppression

У метода Non Maximum Suppression один параметр, влияющий на строгость отбрасывания перекрывающихся найденных объектов — это величина перекрытия в метрике Intersection Over Union. Вычисляется значение по формуле:

$$IoU = \frac{A \cap B}{A \cup B}, \tag{4}$$

где A и B - площади найденных объектов.

Таблица 5. Параметры FastFeatureExtractor

Имя параметра	Описание	Значения
overlap	Значение перекрытия в метрике Intersection Over Union	0.001–0.3
noise_cls	ID класса background или аналогичного	

Пример конфигурации KIRA можно посмотреть в Приложении 1. Написав конфигурацию системы, пользователь инициализирует её ядро, которое дальше готово принимать запросы на обработку данных. В контексте текущей задачи данные приходят от балансировщика, ответственного за обработку запросов.

2.2.2. Балансировщик

Приложение балансировщика так же написано на C++. Его основные задачи на настоящий момент — обработка HTTP запросов к API, получаемых сервером; определение их приоритета; запуск процесса обработки данных и возврат результата серверу. В качестве HTTP сервера используется nginx, он прост в базовой настройке и имеет хорошую документацию [34]. Взаимодействие между ним и балансировщиком происходит по протоколу Fast CGI - наследнику протокола CGI.

CGI — стандарт интерфейса, используемого для связи внешней программы с веб-сервером. Для обработки каждого нового запроса, поступившего к серверу, запускается отдельная копия внешней программы. Fast CGI, в отличие от своего предшественника, инициализирует исполняемые скрипты единственный раз при запуске приложения, что дает

существенный прирост в производительности у программ, которым требуется значительное время для подготовки к обработке запроса.

Балансировщик запрограммирован обрабатывать GET и POST запросы, приходящие на заданные URL. Если URL и параметры указаны правильно, в соответствии с заданной таблицей определяются: 1 из 5 уровней приоритета запроса и сложность его выполнения - низкая или высокая. Если запрос имеет низкую сложность выполнения, что означает отсутствие вычислительных затрат на его обработку, он выполняется сразу по поступлении. Запросы, имеющие высокую сложность выполнения, то есть те, для обработки которых потребуется заметное количество процессорного времени, выстраиваются в очередь в соответствии с приоритетом. Первыми будут выполнены задачи с наивысшим приоритетом, после чего очередь перейдет к следующим уровням.

Если выполнение запроса завершилось успешно, пользователю возвращается ответ в формате JSON (при этом необходимо указывать соответствующий HTTP заголовок content-type: application\json) следующего вида:

```
{
 ''status'':''ok'',
 ''detectedObjects'':
     [
         ...
     ]
 }
```

для успешного запроса, и

```
{
    ''code'':<error code>,
    ''message'':<error description>,
    ''status'':''failed''
}
```

для не успешного.

Ниже приведены описания полей:

- status — дает пользователю знание о состоянии запроса: ok в случае успеха, failed — в случае неудачи;
- detectedObjects — список с найденными объектами, описывался выше;
- message — поле, присутствующее только в ответе на не удачный запрос. Содержит подробное описание ошибки;
- code — код ошибки в соответствии с таблицей HTTP кодов состояния [35];

2.2.3. Конфигурация nginx

Вопросам правильной конфигурации HTTP сервера nginx посвящено множество статей. С другой стороны, для обеспечения бесперебойной работы серверного приложения с небольшим входящим потоком данных можно не вдаваться во все тонкости его настройки, ограничившись базовой конфигурацией.

Для правильной работы сервиса необходимо добавить одну запись в файл /etc/nginx/sites-available/<файл конфигурации детектора>.

Пример добавленного раздела:

```
server {
    server_name detector_server; # 1)
    listen 5000; # 2)
    client_max_body_size 100m; # 3)
    location / {
        fastcgi_pass 127.0.0.1:6000; # 4)
        include fastcgi_params;
    }
},
```

где:

1) имя сервиса;
2) входящий порт, по нему сервис будет доступен с других компьютеров;

3) максимальный размер входящих данных в мегабайтах;

4) внутренний unix socket для обмена данными с детектором.

После обновления конфигурационного файла необходимо перезапустить сервер командой: sudo service nginx restart. На этом базовую настройку можно считать оконченной. Запущенное приложение станет доступно по своему API.

2.2.4. Доработка фреймворка Caffe

Метод Fast R-CNN реализован авторами и встроен в Caffe, однако для его полноценного использования потребовалось добавить в фреймворк поддержку нового входного слоя для пользовательских данных.

Как описывалось ранее, сеть Fast R-CNN сначала вычисляет низкоуровневые признаки для изображения целиком, после чего высокоуровневые признаки вычисляются только для проекций окон-кандидатов на полученные карты признаков. Координаты окон, полученные от внешнего источника, передаются сразу в Roi Pooling Layer, через интерфейс слоя. В работе Росса Гирщика [16] тестирование происходило в Python, поэтому для C++ авторы отдельной реализации делать не стали.

Для использования метода внутри KIRA в Caffe был добавлен новый слой, принимающий данные от MultBox или другого генератора окон и записывающий данные в память так, как того требуют стандарты фреймворка. Его реализация доступна на GitHub [36].

Глава 3. Обучение сетей и тестирование системы

3.1. Обучение MultiBox и Fast R-CNN

Тренировка сетей происходила на отдельном сервере, в составе которого находится GPU Nvidia Tesla K40 с 12Гб памяти — самая современная графическая карта на момент выполнения работы. Несмотря на внушительные вычислительные мощности, дообучение сети MultiBox на подготовленных данных занимает около недели, а сети Fast R-CNN примерно в 3–4 раза меньше.

В основе дообученных сетей лежала уже готовая модель сети GoogLeNet, обучавшаяся на 1,2 миллиона изображений из базы ImageNet. Среди них имелись все заявленные на странице [3] категории. Далее была подготовлена и размечена база предметов одежды и акссессуаров, состоящая из 25000 изображений. Каждая сеть дообучалась на новых данных, предыдущие классы отмечены как background для повышения надежности работы сетей. В процессе обучения каждое изображение из базы проходило через ряд трансформаций, описанных выше, тем самым искусственно увеличивая объем данных для обучения.

3.2. Тестирование детектора

Для тестирования детектора была подготовлена вторая база данных, состоящая из 6500 изображений и такого же количества файлов разметки, где описывались все интересующие заказчика объекты. Все данные можно разделить на 4 больших группы: сумки, одежда женская и мужская, и женская обувь. Оценивалось в отдельности как качество нахождения объекта, так и качество определение его класса. В ходе тестов показатели получились несколько ниже, чем они есть в реальности, связано это с отсутствием четких границ между предметами разных классов: зачастую предмет можно отнести сразу к нескольким

категориям, и для пользователя все они будут представляться верными, но с точки зрения разметки и специалиста, создававшего базу, у объекта может быть только один истинно верный класс.

В приложениях 2, 3, 4 содержатся графики 10, 11, 12. На графике 10 выбраны 10 классов с наилучшими показателями Recall и Precision. Из него видно, что лучше всего детектор справляется с поиском и классификацией сумок и одежды. По графику 11 видно, что хуже всего дела обстоят с детектированием и распознаванием мужской обуви. На графике 12 представлены средние данные по качеству детектора на каждой из четырёх групп объектов и всей тестовой базе целиком.

Сравнение скорости разработанного метода со скоростью модели Fast R-CNN + Selective Search приведено в таблице 6. Сравнение метода MultiBox с другими методами генерации окон-кандидатов приведено в таблице 7.

Таблица 6. Сравнение Multibox + Fast R-CNN с Selective Search + Fast R-CNN

Модель	**Время работы, с**
Selective Search + Fast R-CNN	3
MultiBox + Fast R-CNN	0,3

Таблица 7. Сравенение методов генерации окон-кандидатов

Метод	**Время работы, с**
Selective Search [17]	2,6
GOP [18]	0,6
Edge Boxes [19]	0,25
MultiBox	0,04
MultiBox (пирамида 14 изобр.)	0,14

3.3. Тестирование быстродействия и устойчивости системы

Для проведения стресс-тестов на устойчивость системы была использована утилита Apache JMeter [37]. Она позволяет отсылать на HTTP сервер запросы с выбранным содержанием, вести учет времени получения ответов от сервера и его состояния. Конфигурация была

выбрана следующая: каждые полсекунды на сервер поступала команда на обработку случайно выбранного изображения из базы, тест длился в течении получаса. Ниже представлен график зависимости времени ответа сервера от времени отправки запроса. Видно, что оно редко превышает 350мс.

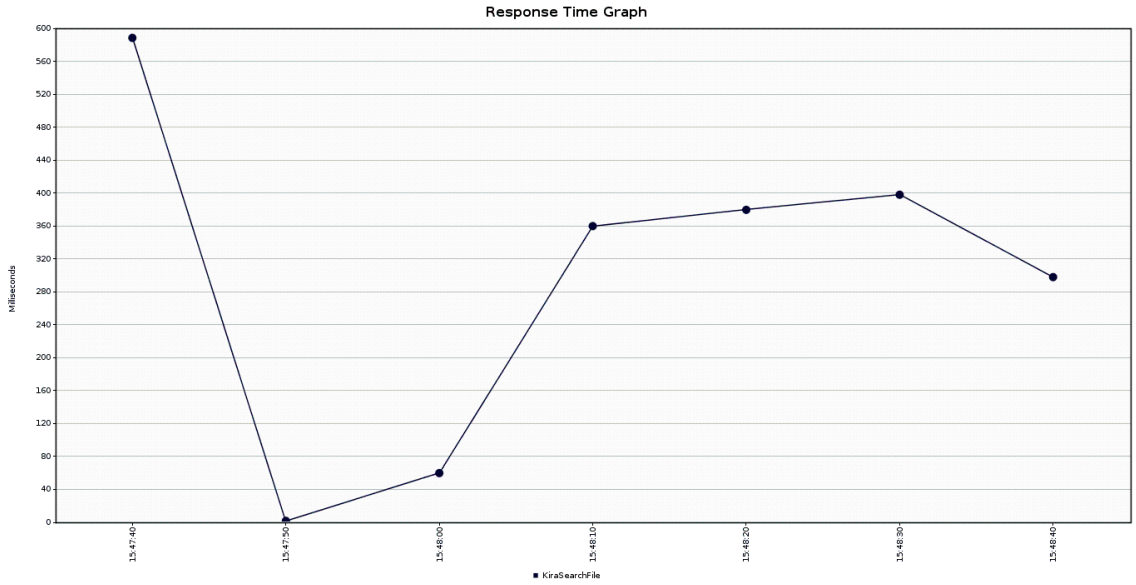

Рис. 8. График времени ответа сервиса

Кроме того, для тестирования устойчивости самого ядра и проверки его на утечки памяти был запущен бесконечный цикл обработки загруженных из интернета случайных изображений. После трёх дней непрерывной работы детектора ошибок выявлено не было, поэтому решено тестирование прекратить. Обычно приложения тестируются на утечки памяти с помощью специально разработанного программного обеспечения, так называемых профилировщиков памяти. В ОС семейства Linux наибольшую популярность имеет утилита Valgrind [38]. После трёхдневного стресс-теста приложение было запущено в его среде, где так же не было выявлено проблем с выделением и освобождением памяти.

В конечном счете, после всех проверок к детектору был подключен web интерфейс, как один из конечных узлов взаимодействия с пользователем. Протестировать его можно по адресу `mobilerecognition.kuznech.com`.

Рис. 9. Примеры результов запросов в web интерфейс

Выводы

Благодаря использованию модели Fast R-CNN, объединённой с модифицированным методом MultiBox для генерации окон-кандидатов, удалось получить систему, достаточно точно детектирующую объекты 156 классов на фотографиях, и при этом работающую достаточно быстро для использования в онлайн-режиме. Итоговая скорость работы системы в несколько раз превысила имеющиеся на тот момент аналоги, а использование обученных классификационных сетей для инициализации моделей, модифицированная функция ошибки в MultiBox и дополнительные методы искажения изображений позволили успешно обучаться на небольшом количестве данных. Тем не менее, на данный момент система всё ещё несовершенна. Для предсказания областей, содержащих объекты, и их классификации используются две различные сети, что требует повторных вычислений низкоуровневых признаков. Объединив две сети в одну, можно было бы использовать на стадии предсказания окон и классификации объектов одни и те же признаки. Таким образом, вычислительные затраты можно было бы существенно сократить. Эта идея была воплощена авторами Fast R-CNN в начале 2016 года. Они представили новую нейронную сеть — Faster R-CNN, которая совмещает в себе сеть предложения окон-кандидатов и сеть-детектор объектов. Чуть позже авторы MultiBox выпустили еще более быструю и качественную модель. Заявляется, что она способна обрабатывать 58 изображений в секунду, что означает обработку одного изображения всего за 16 миллисекунд.

Программное обеспечение детектора (KIRA) получилось достаточно устойчивым для использования его в реальных коммерческих приложениях. Оно справляется с нагрузкой около 200 запросов в минуту в пиковые часы, при этом, время ожидания у пользователя не увеличивается или увеличивается незначительно. Основной архитектурный концепт выбран удачно, модульность и расширяемость несомненно присущи приложению. Однако в процессе использования и адаптации KIRA под другие задачи, связанные с обработкой аудио и видео дан-

ных, выявились узкие моменты проектирования, которые в заставляют программиста писать слишком конкретный код и производить много необходимых утилитарных действий. В целом, ядру требуется рефакторинг, модель обмена данными между экземплярами IMethod должна быть полностью переработана, так же, как и модель установки данных пользователем. Второй существенный недостаток системы: все алгоритмы сейчас находятся непосредственно в классах методов, которые их используют. Здесь прослеживается нарушение принципов инкапсуляции, что так же ведет к излишнему дублированию кода, когда невозможно наследование, или требуется изменить лишь отдельный аспект в работе алгоритма.

Заключение

В рамках проведенной работы были выполнены все поставленные цели:

- удалось достигнуть требуемого качества детектирования предметов одежды;
- система справляется с нагрузкой и остается стабильной в пиковые часы;
- реализованная гибкость позволяет добавлять новые методы обработки изображений и новые классы в систему распознавания с минимальными затратами времени;
- скорость обработки запросов находится на уровне в три раза меньшем верхней границы.

Разработано уникальное программное обеспечение, успешно применяемое в коммерческих приложениях уже на протяжении года. За это время получена награда в конкурсе стартапов Burda International Startup Competition [39], где система оценивалась международным жюри.

Успех разработки еще раз показывает перспективность исследований в сфере нейронных сетей и оправданность их применения в реальных задачах компьютерного зрения.

Список литературы

[1] O Vinyals., A Toshev, Bengio.S, D. Erhan. Show and Tell: A Neural Image Caption Generator. — 2015. — URL: http://arxiv.org/pdf/1411.4555v2.pdf (online; accessed: 11.05.2016).

[2] LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-Based Learning Applied to Document Recognition // Proceedings of the IEEE. — 1998. — November. — Vol. 86, no. 11. — P. 2278–2324.

[3] ImageNet Stats. — 2016. — URL: http://image-net.org/about-stats (online; accessed: 11.05.2016).

[4] ImageNet 2010 results. — 2010. — URL: http://www.image-net.org/challenges/LSVRC/2010/results (online; accessed: 11.05.2016).

[5] ImageNet 2015 results. — 2015. — URL: http://image-net.org/challenges/LSVRC/2015/results (online; accessed: 11.05.2016).

[6] S. Ludger. Shape Matching. — 2011. — URL: http://ganymed.imib.rwth-aachen.de/deserno/seminare/SMBV12_02.pdf (online; accessed: 11.05.2016).

[7] J. Canny. A Computational Approach to Edge Detection. — 1986. — URL: https://perso.limsi.fr/vezien/PAPIERS_ACS/canny1986.pdf (online; accessed: 11.05.2016).

[8] D. Scharstein. Matching Images by Comparing their Gradient Fields. — URL: http://www.cs.middlebury.edu/~schar/papers/gradient.pdf (online; accessed: 11.05.2016).

[9] S. Ekvall, S. Kragic´. Receptive Field Cooccurrence Histograms for Object Detection. — 2005. — URL: http://www.nada.kth.se/~ekvall/ekvall_iros2005.pdf (online; accessed: 11.05.2016).

[10] SIFT // Lowe D. — 2003. — URL: http://www.cs.ubc.ca/~lowe/keypoints/ (online; accessed: 11.05.2016).

[11] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool. SURF. — 2006. — URL: http://www.vision.ee.ethz.ch/en/publications/papers/articles/eth_biwi_00517.pdf (online; accessed: 11.05.2016).

[12] Locality-sensetive hashing. — 2015. — URL: https://en.wikipedia.org/wiki/Locality-sensitive_hashing (online; accessed: 11.05.2016).

[13] R. Girshick, J Donahue, T. Darrell, J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. — 2014. — URL: http://arxiv.org/pdf/1311.2524v5.pdf (online; accessed: 11.05.2016).

[14] B. Lao, K. Jagadeesh. Convolutional Neural Networks for Fashion Classification and Object Detection. — 2015. — URL: http://cs231n.stanford.edu/reports/BLAO_KJAG_CS231N_FinalPaperFashionClassification.pdf (online; accessed: 11.05.2016).

[15] K. Hara, V. Jagadeesh, R. Piramuthu. Fashion Apparel Detection: The Role of Deep Convolutional Neural Network and Pose-dependent Priors. — 2016. — URL: http://arxiv.org/pdf/1411.5319v2.pdf (online; accessed: 11.05.2016).

[16] R. Girshick. Fast R-CNN. — 2015. — URL: http://arxiv.org/pdf/1504.08083v2.pdf (online; accessed: 11.05.2016).

[17] J. Uijlings, van de Sande K., T. Gevers, A. Smeulders. Selective Search for Object Recognition. — 2012. — URL: http://koen.me/research/pub/uijlings-ijcv2013-draft.pdf (online; accessed: 11.05.2016).

[18] P. Krahenbuhl, V. Koltun. Geodesic Object Proposals. — 2014. — URL: https://38fc75ec1deae427f6b0d7aacf75f8d36406fe02.googledrive.com/host/0B6qziMs8hVGieFg0UzE0WmZaOW8/papers/gop.pdf (online; accessed: 11.05.2016).

[19] Edge Boxes // arxiv.org. — 2015. — URL: http://research.microsoft.com/pubs/220569/ZitnickDollarECCV14edgeBoxes.pdf (online; accessed: 11.05.2016).

[20] C. Szegedy, S. Reed, D. Erhan, D. Anguelov. Scalable, High-Quality Object Detection. — 2015. — URL: http://arxiv.org/pdf/1412.1441v2.pdf (online; accessed: 11.05.2016).

[21] REST. — URL: https://ru.wikipedia.org/wiki/REST (online; accessed: 11.05.2016).

[22] Nginx. — URL: http://nginx.org/ru/ (online; accessed: 11.05.2016).

[23] Fast CGI. — URL: https://github.com/FastCGI-Archives/FastCGI.com (online; accessed: 11.05.2016).

[24] A. Krizhevsky. Cuda Convnet. — 2012. — URL: https://code.google.com/p/cuda-convnet/ (online; accessed: 11.05.2016).

[25] Caffe, Deep Learning Framework. — URL: http://caffe.berkeleyvision.org/ (online; accessed: 11.05.2016).

[26] R. Girshick, J. Donahue, T. Darrell, J. Malik. Region-based convolutional networks for accurate object detection and segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2016. Vol. 38, No 1. P. 142–158.

[27] H. Kaiming, Z. Xiangyu, R. Shaoqing, S. Jian. Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. IEEE International Conference on Computer Vision (ICCV). 2015. P. 1026–1034.

[28] C. Szegedy, W. Liu, Y. Jia, et al. Going deeper with convolutions. — URL: http://arxiv.org/pdf/1409.4842v1.pdf (online; accessed: 11.05.2016).

[29] OpenCV. — URL: http://opencv.org/ (online; accessed: 11.05.2016).

[30] Boost C++ Libraries. — URL: http://www.boost.org/ (online; accessed: 11.05.2016).

[31] FastCGI, CGI C++ Library. — URL: http://cgi.sourceforge.net/ (online; accessed: 11.05.2016).

[32] Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — 5th edition. — Massachusetts : Addison-Wesley, 1995.

[33] D. Rager. Factory Design Pattern in C++. — URL: http://blog.fourthwoods.com/2011/06/04/factory-design-pattern-in-c/ (online; accessed: 11.05.2016).

[34] Nginx documentation. — URL: http://nginx.org/ru/docs/ (online; accessed: 11.05.2016).

[35] List of HTTP status codes. — URL: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes (online; accessed: 11.05.2016).

[36] GitHub/Zoellick/caffe-fast-rcnn. — 2016. — URL: https://github.com/Zoellick/caffe-fast-rcnn/tree/fast-rcnn (online; accessed: 11.05.2016).

[37] Apache JMeter. — URL: http://jmeter.apache.org/ (online; accessed: 11.05.2016).

[38] Valgrind. — URL: http://valgrind.org/ (online; accessed: 11.05.2016).

[39] Burda International Startup Competition. — 2015. — URL: http://goo.gl/xtTSGK (online; accessed: 11.05.2016).

Приложение 1

Пример конфигурационного файла KIRA.

```
{
  ''methods'':
  [
    {
      ''name'':''B64ToJpeg'',
      ''input'':[''user''],
      ''output'':[''Multibox'']
    },
    {
      ''name'':''Multibox'',
      ''input'': [''B64ToJpeg''],
      ''output'': [''FastFeatureExtractor''],
      ''mode'': ''GPU'',
      ''model_path'': ''MultiBox156_medium__iter_
                          100000.caffemodel'',
      ''net_path'': ''deploy_MultiBox156.prototxt'',
      ''device_id'': 0,
      ''feature_layer'': ''concat_predictions'',
      ''prior_path'' : ''MultiBoxPriors156.txt'',
      ''threshold'': 0.0001,
      ''part_coords'': [[0,0,1],[0,0,0.33],[0.33,0,0.33],
                        [0.66,0,0.33],[0,0.33,0.33],
                        [0.33,0.33,0.33],[0.66,0.33,0.33],
                        [0,0.66,0.33],[0.33,0.66,0.33],
                        [0.66,0.66,0.33],[0,0,0.66],
                        [0.33,0,0.66],[0,0.33,0.66],
                        [0.33,0.33,0.66]],
      ''pr_size'':224,
      ''with_conf'':true
    },
```

```
    {
      ''name'':''FastFeatureExtractor'',
      ''input'': [''Multibox''],
      ''output'': [''BBoxNmsMethod''],
      ''mode'': ''GPU'',
      ''model_path'': ''googlenet_fast_rcnn_156_iter_
                        65000.caffemodel'',
      ''net_path'': ``test_googlenet.prototxt'',
      ''device_id'': 0,
      ''feature_layer'':''cls_prob'',
      ''bbox_layer'':''bbox_pred'',
      ''pr_size'':700,
      ''max_size'':1500
    },
    {
      ''name'': ''BBoxNmsMethod'',
      ''input'' : [''FastFeatureExtractor''],
      ''output'' : [''BBoxMapper''],
      ''overlap'' : 0.1,
      ''noise_cls'' : 0
    },
    {
      ''name'':''BBoxMapper'',
      ''input'': [''BBoxNmsMethod''],
      ''output'': [''user''],
      ''labels_csv'':''Fashion156_classes_det.txt''
    }
  ]
}
```

Приложение 2

Диаграмма распределения показателей точности и полноты. Выбраны 10 классов с наибольшими значениями.

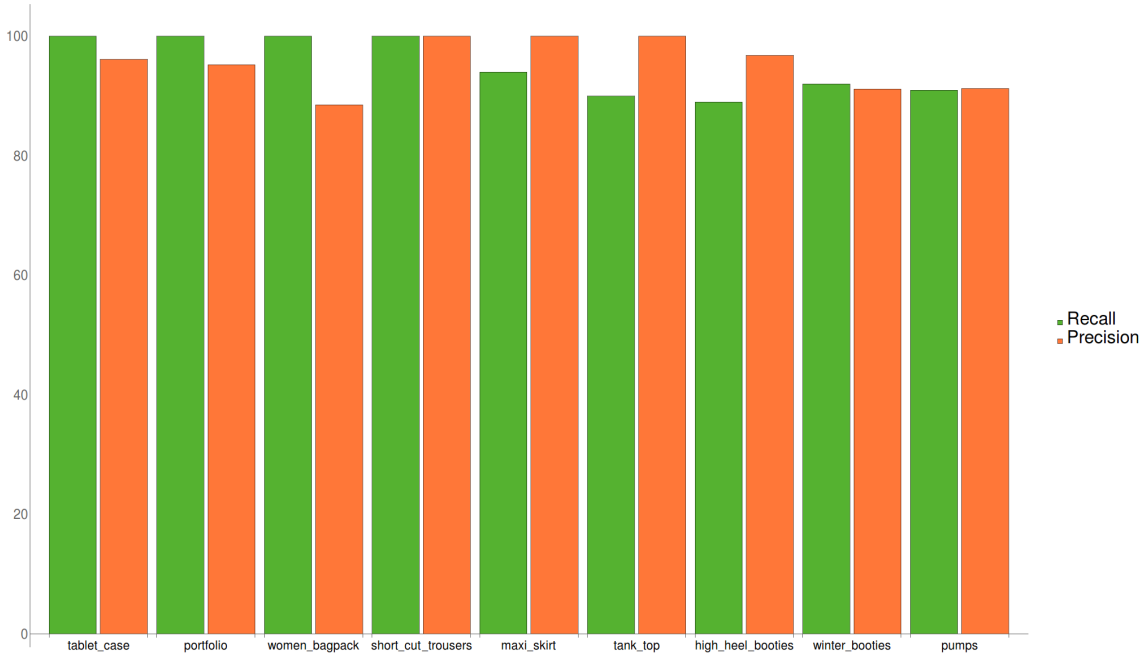

Рис. 10. Recall и precision для 10 «лучших» классов

Приложение 3

Диаграмма распределения показателей точности и полноты. Выбраны 10 классов с наименьшими значениями.

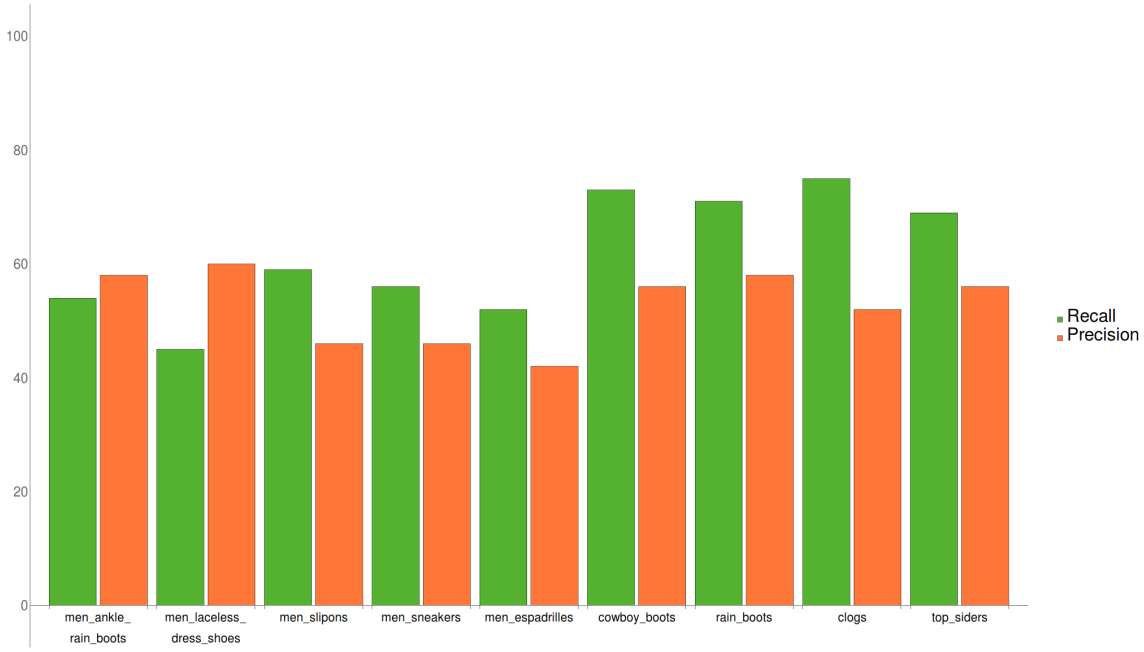

Рис. 11. Recall и precision для 10 «худших» классов

Приложение 4

Диаграмма распределения показателей точности и полноты для 4 общих категорий в отдельности и по полной базе.

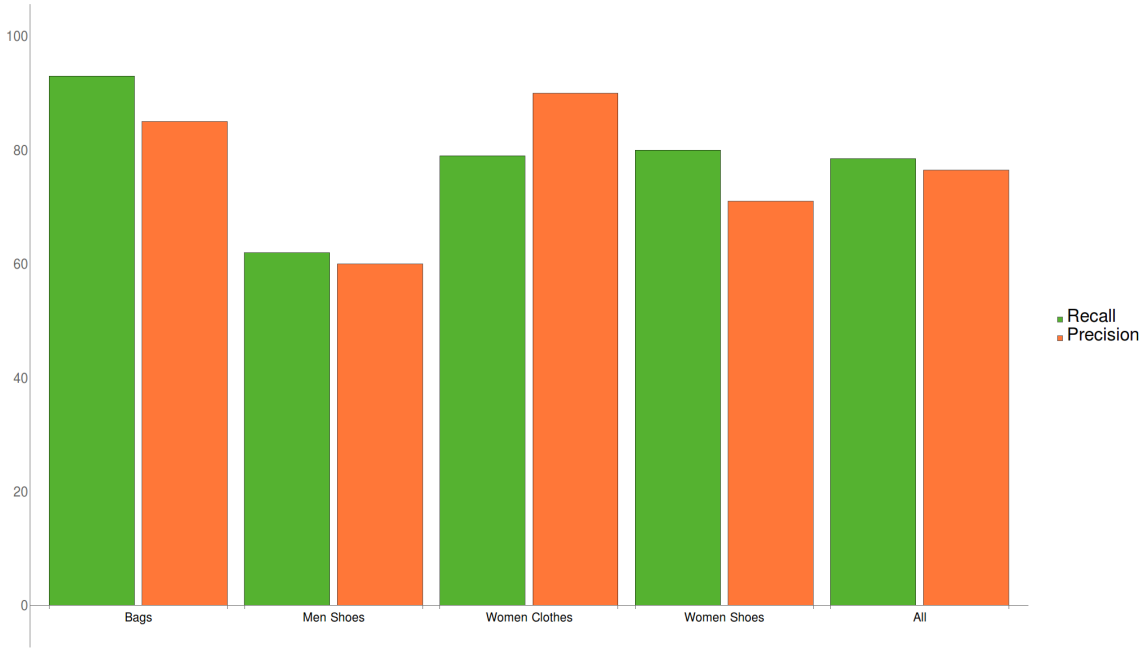

Рис. 12. Recall и precision для общих категорий и всей тестовой базы