Федеральное государственное автономное образовательное учреждение
высшего образования
“Московский физико-технический институт (государственный университет)”

Факультет радиотехники и кибернетики
Кафедра теоретической и прикладной информатики

Направление подготовки/специальность: 03.04.01 Прикладные математика и физика

Направленность (профиль) подготовки: Математические и информационные технологии

Форма обучения: очная

Допущен к ГИА

Руководитель учебного подразделения

_____/_____

(подпись) расшифровка подписи

“___” _____ 20___г.

Поиск областей выделения текста на изображении методами машинного обучения.

(магистерская диссертация)

Обучающийся: Рудой Артем Игоревич

_____

(подпись обучающегося)

Научный руководитель: Пашков Руслан Анатольевич, к.ф.-м.н.

_____

(подпись научного руководителя)

Москва 2018

Аннотация

Благодаря активному развитию мощностей компьютеров и мобильных устройств на сегодняшний день они способны решать сложные вычислительные задачи. К классу таких задач можно отнести и задачи машинного обучения. Крупные технологические компании все больше и больше ресурсов инвестируют в эту сферу[1]. Из-за всех этих факторов за последние годы эта сфера активно развивается. Алгоритмы машинного обучения активно применяются во многих областях промышленности и повседневной жизни: в задачах предсказательной аналитики, распознавание речи, человеческого генома, биоинформатика, оптического распознавание символов, распознавание лиц, самоуправляемых автомобилей, компьютерного зрения и тд.

Кроме того, активно развиваются технологии, позволяющие запускать алгоритмы машинного обучения на мобильном устройстве[2]. В данной работе детально рассмотрено одно из применений алгоритмов машинного обучения для решения задачи компьютерного зрения на мобильном устройстве.

Оглавление

1. **Введение**……4
   1.1 Предпосылки……4
   1.2 Проблемная область……....5
   1.3 Описание системы……...6
   1.4 Назначение и применение……...8
2. **Постановка задачи**……...9
   2.1 Общие требования……...9
   2.2 Требования к модулю генерации обучающей выборки……....9
   2.3 Требования к модулю распознавания областей выделения……...10
3. **Разработка модуля по генерации обучающей выборки**……..11
   3.1 Предпосылки……..……11
   3.2 Алгоритм формирования результирующего изображения……..……..13
      3.2.1 Поиск на получившемся изображении полигонов
         и заливка их белым цветом……15
      3.2.2 Математическая морфология…..……...17
         3.2.2.1 Описание математической морфологии……....17
         3.2.2.2 Перенос…….....18
         3.2.2.3 Наращивание……....18
         3.2.2.4 Эрозия……...19
      3.2.3 Финальный алгоритм генерации обучающей выборки……...21
   3.3 Утилита для генерации обучающей выборки……22

4. **Разработка модуля по поиску областей выделения**……..……...29
4.1 Предпосылки……...…….…….....29
4.2 Сверточные нейронные сети……...……..30
4.2.1 Слой свертки……...……...…....31
4.2.2 Слой субдискретизации…….33
4.2.3 Полносвязный слой……34
4.3 Подбор архитектуры сверточной нейронной сети…….…….35
4.3.1 Метрики……...35
4.3.2 Подбор базовой архитектуры сети……....37
4.3.3 Влияние размера батча……...43
4.3.4 Влияние батч-нормализации…….44
4.4 Пример изображения. Анализ производительности……..48
5. **Заключение**…….50
5.1 Выводы……...……...……..50
5.2 Планы на дальнейшую работу……..51

Глава 1

Введение

1.1 Предпосылки

С каждым годом число покупаемых мобильных устройств растет. Также растет и их техническое оснащение и вычислительные мощности. Развиваются и приложения под мобильные устройства, которые помогают решать повседневные задачи. Под самые популярные мобильные операционные системы Android и iOS уже доступны самые необходимые офисные программы такие как Microsoft Office Word, Excel, Skype, продукты от компании Google , которая тоже предлагает набор приложения для работы с документами: Sheets, Docs, Presentation. Причем приложения для мобильных устройств несильно уступают аналогам для персональных компьютеров.

Однако не всегда такие офисные программы полностью удовлетворяют запросам пользователя: иногда требуется сделать какие-либо действия непосредственно с рабочего компьютера или требуется непосредственный доступ к персональному компьютеру для извлечения каких-либо данных или выполнения определенных действий. Для решения это класса задач уже существуют мобильные приложения от известных компаний: TeamViewer, Parallels Access, Parallells RAS, которые прибегают для этого к трансляции видео экрана хостовой

операциооной системы на экран устройства клиента. Но для бесшовного взаимодействия пользователя с такого рода программами необходима их дополнительная доработка. Особо это актуально для приложений, запускаемых на мобильных устройствах, поскольку необходимо отобразить очень большое количество информации на маленьком дисплее. К тому же необходимо правильно учитывать еще то, что на данный момент большинство мобильных устройств имеют сенсорное управление и дополнительные кнопки навигации, которые должны быть адаптированы так, чтобы пользователю было интуитивно понятно использование приложения. Эта работа описывает решение одной из таких проблем: это поиск областей выделения текста на изображении и адаптации этого решения для запуска на мобильном устройстве.

В силу роста вычислительных мощностей персональных компьютеров и мобильных устройств на данный момент они становятся способны производить тяжелые вычисления очень быстро. К тому же активно развиваются технологии компьютерного зрения, которые помогают решать разные технологические задачи. К тому же крупная технологическая компания Google представила несколько лет назад кроссплатформенную библиотеку Tensorflow для реализации и запуска алгоритмов машинного обучения на устройствах на базе операционных систем Android и iOS[2] . Поэтому будет испробован метод решения проблемы поиска областей выделения методами машинного обучения.

1.2 Проблемная область

Поскольку задача решается методами машинного обучения, то необходимо первым делом подготовить и разметить обучающую выборку. Но поскольку изображений должно быть большое количество и они должны браться из разных

приложений, то процесс генерации и разметки должен быть автоматизирован. Кроме того, данные для обучения должны браться из наиболее популярных программ, используемых на персональном компьютере для учета всех особенностей.

К тому же важно понимать, что от приложения к приложения области выделения могут отличаться. Наиболее частым отличием является цвет, также может отличаться поведение букв внутри области выделения: в одних случаях они остаются такими же, как и до выделения, в других их цвет инвертируется, также может вокруг области выделения по-другому подсвечиваться интерфейс приложения, например в продукте от Microsoft : Excel - помимо выделения текста выделяются дополнительно сами ячейки и еще название строк и столбцов в которых произошло выделение, а также задача осложнена тем, что нужно отличать настоящие области выделения от просто полигонов с таким же цветом заливки. Все эти особенности должен учитывать алгоритм по поиску областей выделения текста.

Еще одним важным моментом является то, что алгоритм по поиску областей выделения будет запускаться на мобильном устройстве, поэтому должен быть не сильно требователен к энергопотреблению и вычислительным мощностям и способный обрабатывать потоковое видео не сильно уменьшая производительность

1.3 Описание системы

Предполагается, что есть приложение для обеспечения удалённой работы пользователя с сервером, на котором запущен сервис терминальных подключений

и на клиент поступает видеопоток данных от хостовой операционной системы: пользователь может активно пользоваться через клиентское приложение своим персональным компьютером.

В результате действий пользователя на клиентском приложении может произойти выделение какого-либо участка текста или изображения на хостовом компьютере. Поскольку, предполагается, что никакой дополнительной информации кроме видеопотока не поступает, то клиентское приложение не будет уведомлено о факте выделения, что необходимо для предоставления пользователю нативных контролов для управления области выделения.

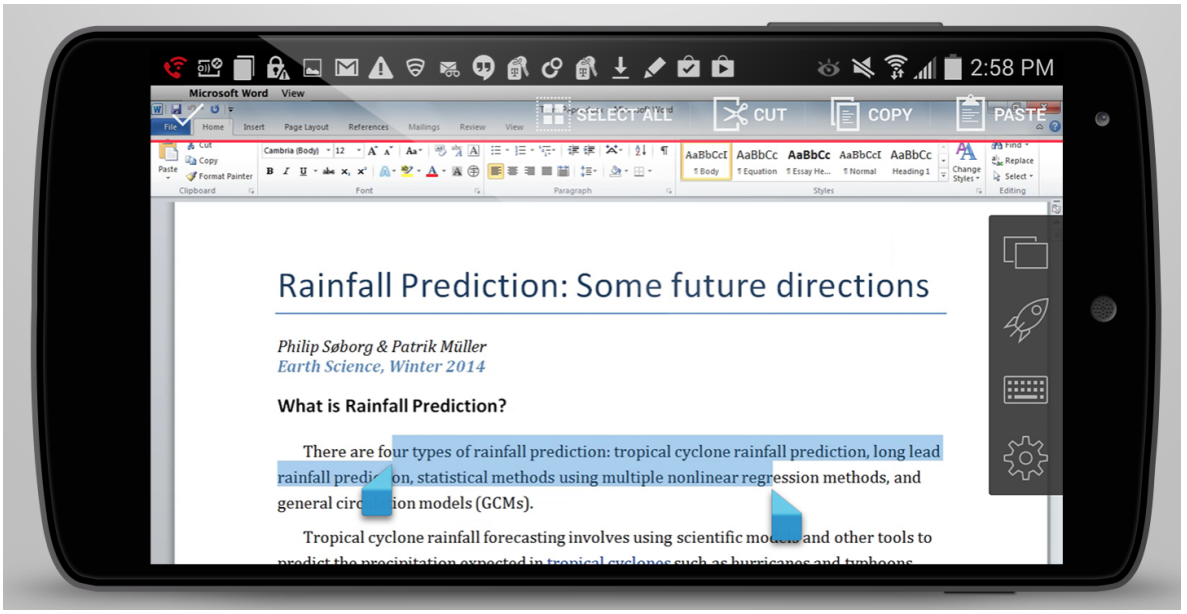


Рис 1.1. Пример выделения

Собственно требуется найти положение таких контролов после произошедшего выделения и передать их координаты во внутренний API клиентского приложения.

1.4 Назначение и применение

Основной задачей разрабатываемого модуля будет являться решение проблемы нахождения областей выделения текста по изображению. На данный момент эту задачу решает эвристический алгоритм и он будет заменен в случае если разрабатываемый модуль покажет лучшие результаты в ходе тестирования.

Глава 2

Постановка задачи

2.1 Общие требования

Перед началом выполнения работы выдвигались следующие требования:

1. Разработать алгоритма по генерации и разметке обучающей выборке
2. Автоматизировать процесс генерации и разметки обучающей выборки
3. Найти подходящие архитектуры нейронных сетей
4. Произвести оптимизацию гиперпараметров нейронных сетей
5. Запустить полученную нейросеть на устройствах Android и iOS
6. Измерение производительности и точности полученной системы на платформах iOS и Android с целью получения необходимого результата

2.2 Требования к модулю генерации обучающей выборки

Необходимо разработать модуль по генерации обучающей выборки, который будет облагаться следующими требованиями:

1. Разработать алгоритм по генерации одного изображения для обучающей выборки
2. Написать кроссплатформенное приложение, использующее этот алгоритм для автоматизации процесса генерации обучающей выборки
3. Сгенерировать и разметить обучающую выборку для популярных приложений (Microsoft Office, Microsoft Outlook, Microsoft Excel, Google Chrome)

2.3 Требования к модулю распознавания областей выделений текста

1. Высокая скорость работы алгоритма (ориентир - существующая реализация)
2. Низкое энергопотребление
3. Точность поиска выделения должна быть не хуже, чем в существующем алгоритме

Глава 3

Разработка модуля по генерации обучающей выборки

3.1 Предпосылки

Из-за того, что способом решения поставленной выбраны алгоритмы машинного обучения, то, соответственно, перед выбором архитектуры требуется сгенерировать и разметить большое количество данных. Поскольку модуль по обнаружению областей выделения текста по изображению должен работать в рамках продукта компании Parallels, то была взята статистика использования продукта Parallels Access и на ее основании были выбраны самые используемые пользователями программы: Microsoft Office, Microsoft Outlook, Microsoft Excel, Google Chrome. Собственно для них и было принято решение генерировать и размечать обучающую выборку.

Поскольку количество изображений в обучающей выборке должно измеряться тысячами, то процесс генерации должен хорошо автоматизирован.

Изначально было решено, попробовать такой вариант, в котором нейронная сеть использовалась в качестве фильтра для входного изображения и ее основной задачей являлось дать на выходе изображение, в котором подсвечена только область выделения текста. А далее по этому изображению уже намного проще определить границы выделения текста.

В силу такого решения обучающая выборка представляет собой множество пар изображений, где первое изображение является обычным изображением с выделением, а второе: изображение, где белым изображена только область выделения:

Исходное изображение:

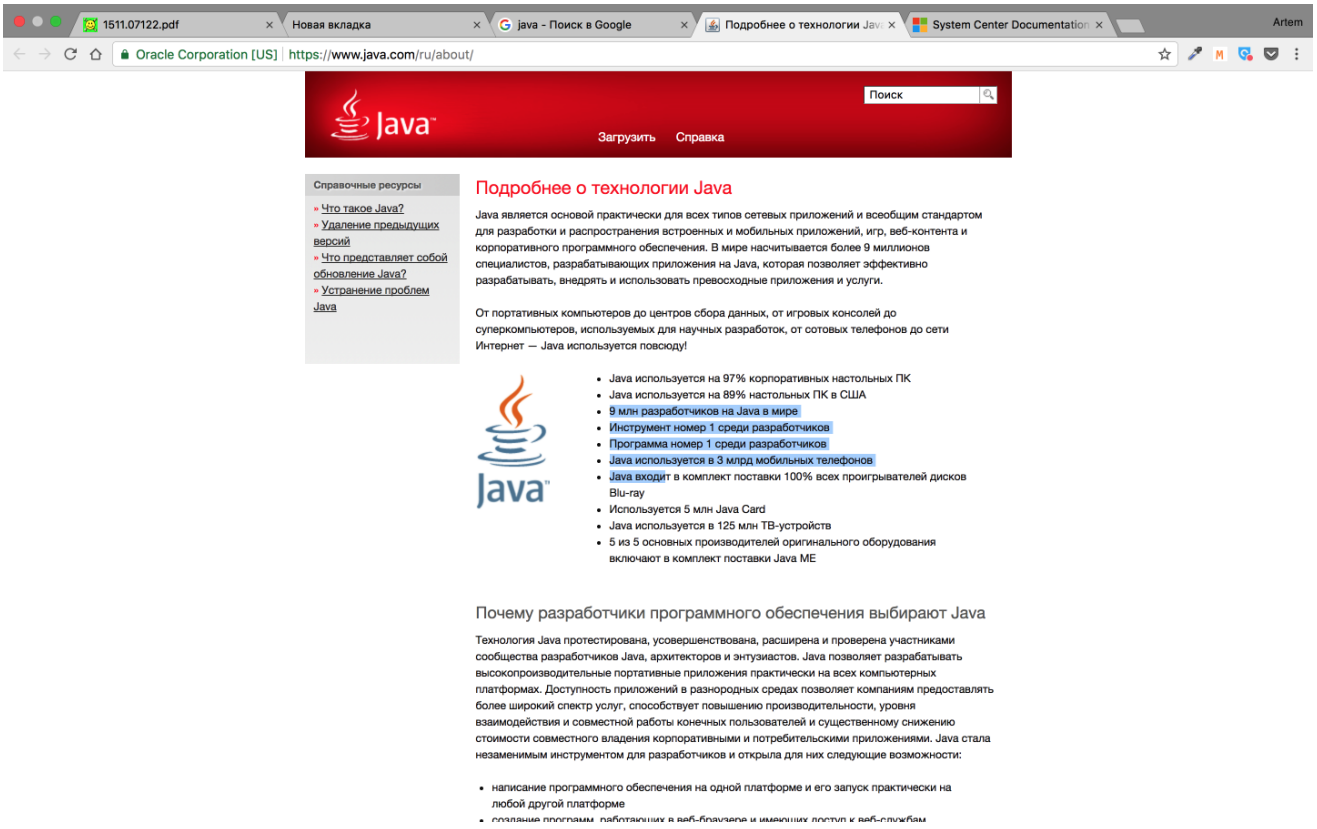


Рис. 3.1 Пример изображения с выделением

Изображение, в котором только области выделения отмечены белым:

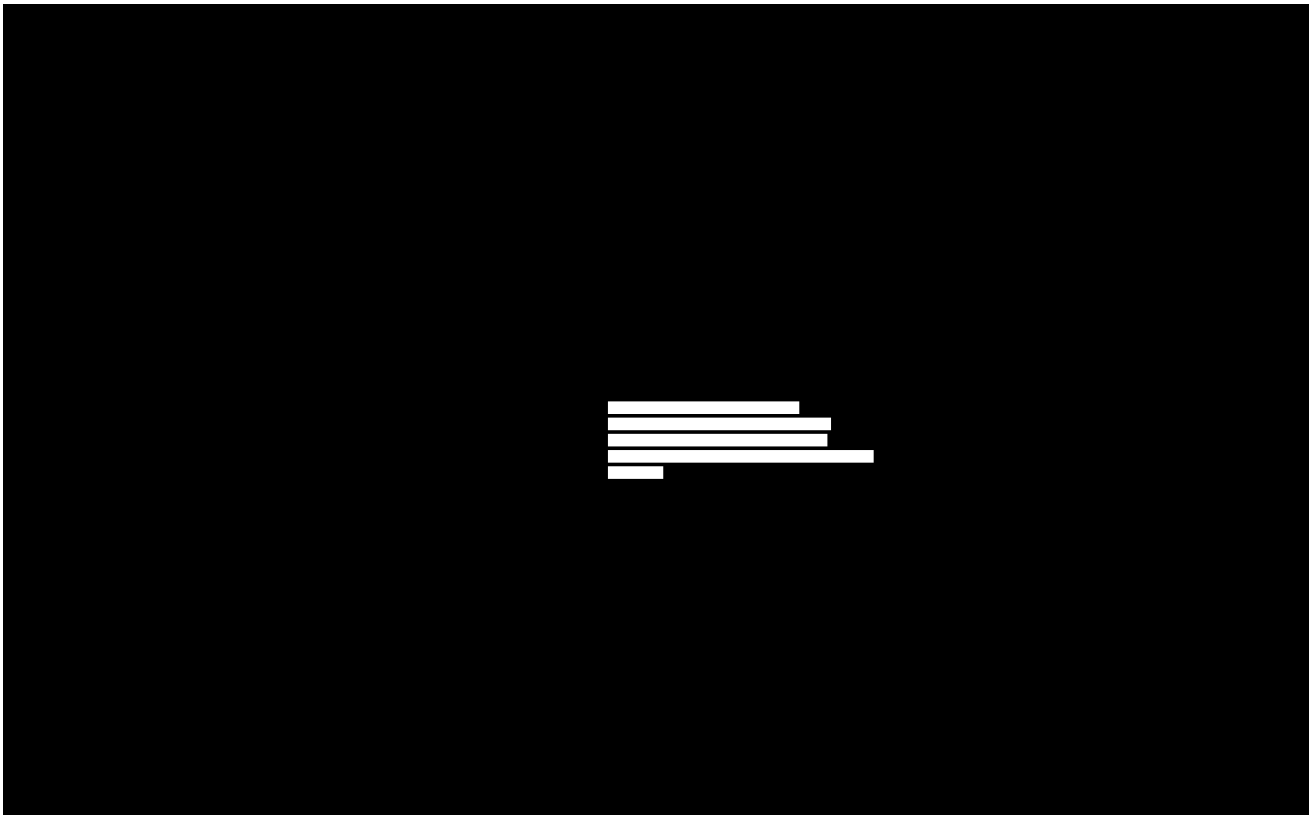


Рис. 3.2 Пример изображения с отмеченной областью выделения

3.2 Алгоритм формирования результирующего изображения

Чтобы получить такую пару изображений было решено попиксельно сравнивать два изображение - первое до выделения, а второе уже после выделения и на базе этого сравнения генерировать третье изображение, в котором черным обозначался пиксель, если на первых двух цвета пикселей совпали и белым, если цвета пикселей не воспали.

Такой алгоритм был быстро реализован, однако он оказался полностью рабочим лишь на изображениях из программы IntelliJ IDEA. Проблема была в том, что когда сравнивались попиксельно изображения не было учтено то, что для полной правильной работы такого алгоритма буквы внутри области выделения тоже должны отличаться по цвету, иначе они их будут закрашиваться черным:

Разница между:

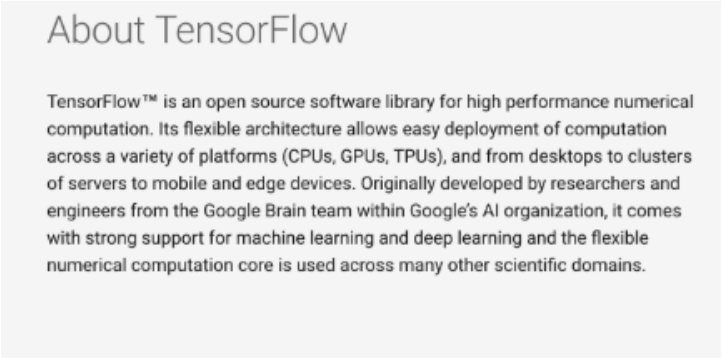


Рис. 3.3 Текст до выделения

и

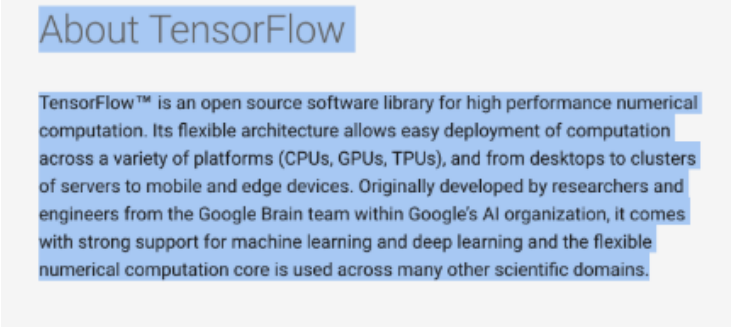


Рис. 3.4 Текст после выделения

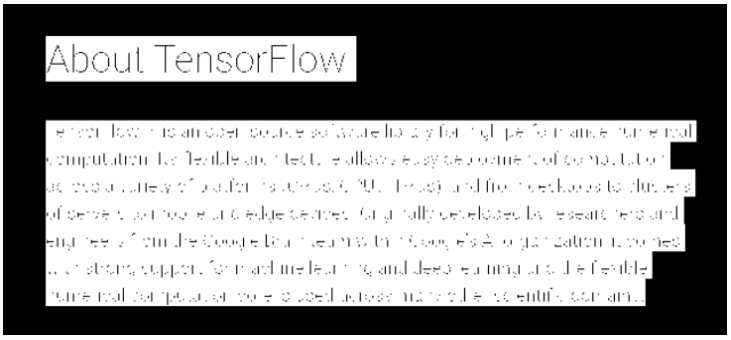


Рис. 3.5 Результат попиксельного сравнения

Как видно, полученное результирующее изображение имеет дефекты внутри белой области, а это несло бы дополнительный сторонний сложно прогнозируемый эффект для уже обученной на таких данных нейронной сети, поэтому было решено избавляться от таких дефектов.

Были испробовано два разных способа избавления от таких дефектов:

1. Поиск на получившемся изображении полигонов и заливка их белым цветом
2. Математическая морфология

3.2.1 Поиск на получившемся изображении полигонов и заливка их белым цветом

Изначально был опробован следующий подход: поскольку после попиксельного сравнения проблемы возникали только со внутренними символами, а границы казались ровными, то возникла идея обходить на получившемся изображении все полигоны, запоминать их координаты и на заранее полностью черном изображении заново рисовать белые полигоны и заливать их белым цветом.

Алгоритм работал так, что начиная с левого верхнего угла слева направо и соответственно сверху вниз двигался курсор, который искал область выделения. Как только он доходил до белого пикселя происходил обход полигона по часовой стрелке, далее, когда курсор возвращался в исходную точку, координаты полигона добавлялись в список и курсор начиная с правой верхней точки опять двигался слева направо, чтобы найти другие области выделения, которые могут лежать на одной линии. После того, как обходилось все изображение, генерировалась новая картинка на основании найденных полигонов или записывалась ошибка в файл, если не удавалось обойти полигон.

Примерное описание алгоритма:

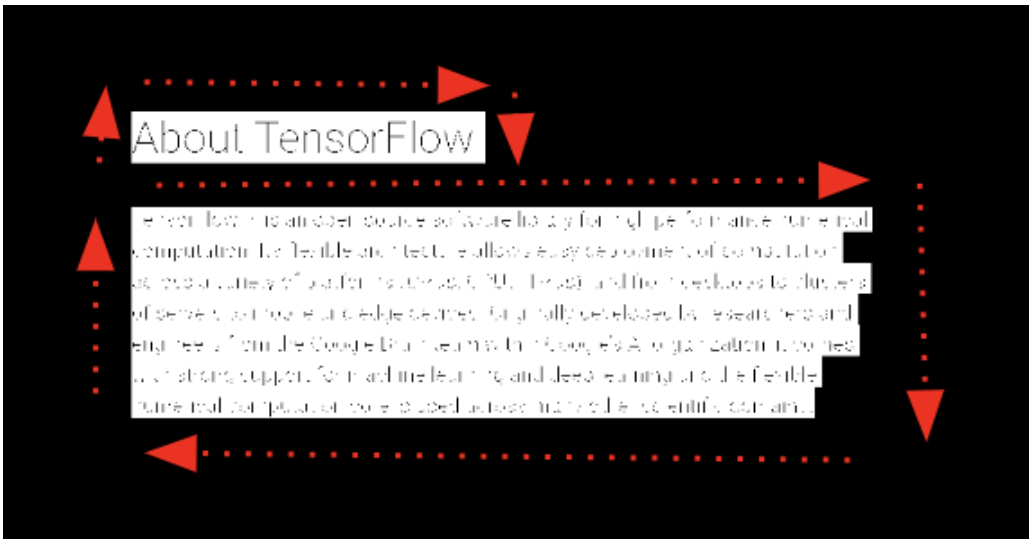


Рис. 3.5 Обход по часовой стрелке

Однако, как показала практика такой алгоритм в ряде случаев не работал: иногда получалось так, что буква граница буквы совпадала с границей области выделения:

Рис. 3.6 Пример проблемного выделения

и из-за этого при по пиксельном сравнение полигон не имел четкой границы, соответственно при обходе этой области выделения возникала ошибка.

3.2.2 Математическая морфология

Тогда было принято решения попробывать более сложные методы обработки изображения - техники математической морфологии.

3.2.2.1 Описание математической морфологии

Математическая морфология [3] - это хорошо зарекомендовавшая себя методология для анализа геометрических структур. Ее теоретические корни основаны на теории множеств, топологии, стохастической геометрии, нелинейных уравнениях с частными производными и т. д. Математическая морфология применяется к обработке цифровых изображений и других форм пространственных структур таких как графов, поверхностных сеток, облаков данных и т.д.

В бинарной морфологии под областью изображения подразумевается некоторое подмножество точек самого изображения, а оно само представляется в виде набора белых и черных пикселей. Результатом преобразования двоичного изображения B совместно со структурным элементом S является также двоичное изображение.

Структурный элемент является некоторым бинарным изображением любой формы и структуры. Во всех структурных элементах выделяется начальная точка, которая может находиться в произвольном месте структурного элемента S.

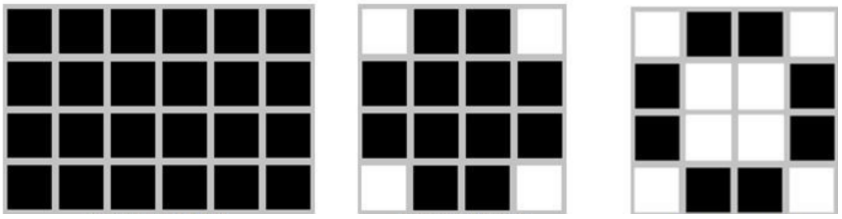


Рис. 3.7 Наиболее распространенные структурные элементы: BOX[H,W] -прямоугольник заданного размера, DISK[R] — диск заданного размера, RING[R] — кольцо заданного размера

3.2.2.2 Перенос

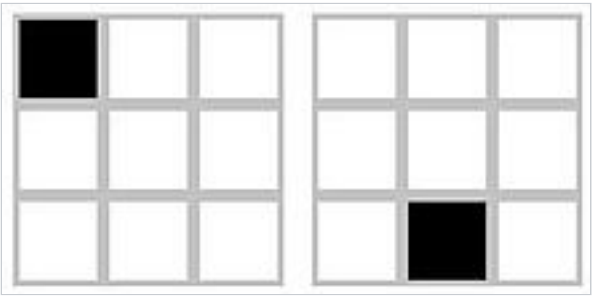


Рис. 3.8 Пример переноса при t=(2,1).

Операция переноса X_t множества пикселов X на вектор t задаётся в виде $X_t=\{x+t|x\in X\}$. Следовательно, перенос множества единичных пикселов на бинарном изображении сдвигает все пикселы множества на заданное расстояние. Вектор переноса t может задаваться в виде упорядоченной пары $(\Delta r,\Delta c)$, где Δr — компонент вектора переноса в направлении строк, а Δc — компонент вектора переноса в направлении столбцов изображения.

3.2.2.4 Наращивание

Наращивание бинарного изображения A структурным элементом B обозначается $A \oplus B$ и задаётся выражением:

$$A \oplus B = \bigcup_{b \in B} A_b \,.$$

В данном выражении оператор объединения можно считать оператором, применяемым в окрестности пикселов. Структурный элемент B применяется ко всем пикселам бинарного изображения. Каждый раз, когда начало координат структурного элемента совмещается с единичным бинарным пикселом, ко всему структурному элементу применяется перенос и последующее логическое сложение (логическое ИЛИ) с соответствующими пикселами бинарного изображения. Результаты логического сложения записываются в выходное бинарное изображение, которое изначально инициализируется нулевыми значениями.

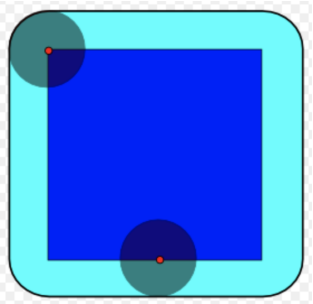


Рис. 3.9 Пример наращивания темно синего квадрата дисковым структурным элементом

3.2.2.5 Эрозия

Эрозия бинарного изображения A структурным элементом B обозначается $A \ominus B$ и задаётся выражением:

$$A \ominus B = \{z \in A | B_z \subseteq A\}.$$

При выполнении операции эрозии структурный элемент тоже проходит по всем пикселам изображения. Если в некоторой позиции каждый единичный пиксел структурного элемента совпадет с единичным пикселом бинарного изображения, то выполняется логическое сложение центрального пиксела структурного элемента с соответствующим пикселом выходного изображения. В результате применения операции эрозии все объекты, меньшие чем структурный элемент, стираются, объекты, соединённые тонкими линиями становятся разъединёнными и размеры всех объектов уменьшаются.

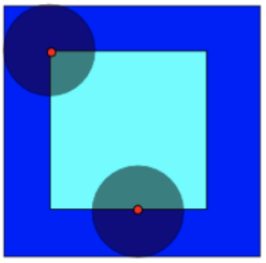


Рис. 3.10 Пример эрозии.

Были изучены все эти методы и в итоге было решено воспользоваться операцией наращивания границ.

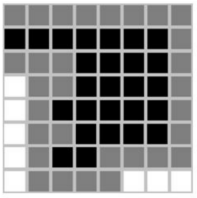


Рис. 3.10 Серый цветом обозначена расширенная граница черной области.

И чтобы убрать черные дефекты внутри полигонов пару раз применяется расширение белой области, а потом столько же раз применяется расширение черной области, чтобы убрать дополнительный сторонний эффект.

Итого данный алгоритм выглядел таким образом:

1. Генерация нового изображения на основе попиксельного сравнения
2. N шагов расширения белой области
3. N шагов расширения черной области

Однако такой метод дает ощутимый недостаток, что число шагов для расширения должно определяться размером выделенных букв: чем больше шрифт и размер - тем больше шагов, иначе все равно останутся черные дефекты. Но, к сожалению, найти способ, который позволял узнать сколько шагов необходимо применить для расширения не удалось. К тому же, при выделении текста возможны случаи, когда под область выделения попадают фрагменты текста разного размера и шрифта, например, заголовок и содержимое под ним, к тому же, если применять несколько раз расширение белой области в случае если несколько областей выделения находятся очень близко, то они могут слиться в одну большую область и обратно не разобьются при расширении черной области.

3.2.3. Финальный алгоритм генерации обучающей выборки

В итоге получив, опыт в разработке этих двух алгоритмов было решено их объединить. Чтобы исключить проблемные места, сначала применяется алгоритм математической морфологии по расширению белой области, в результате граница

расширяется и размывает пограничные буквы, потом происходит расширение черной области,а потом уже алгоритм по поиску полигонов и на основе него генерируется новое изображение

Финальная схема:

1. Попиксельный дифф
2. Расширение
3. Поиск полигонов
4. Генерация нового изображения
5. Сужение

Однако данный подход не учитывает более сложные случаи, но для простоты реализации , там где не получилось найти области выделения просто откидываем.

3.3 Утилита для генерации обучающей выборки

После того, как был разработан алгоритм, необходимо было реализовать программу, в основе которой лежит этот алгоритм для автоматизации генерации и разметки обучающей выборки.

Поскольку, предполагалось, что программы, для которых будут генерироваться скриншоты будут запускаться на разных операционных системах, причем для простоты пользования утилитой, было также решено написать графическую оболочку то было решено реализовывать утилиту сразу для всех популярных платформ. Соответственно, выбор для языка реализации пал на

следующие языки и соответствующие кроссплатформенные библиотеки:

1. C++: QT[7]
2. Python: PyGUI[8], Pyforms [9]
3. Java, Kotlin: Swing

При помощи Qt появляется возможность запускать написанное на этой библиотеке программное обеспечение почти на всех современных операционных системах: для этого необходимо лишь перекомпилировать исходный код под нужную ОС. Библиотека содержит большинство основных классов, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML.

PyGUI и Pyforms и соответственно написаны на языке программирования Python и также позволяют решать задачи кроссплатформенного написания программного обеспечения для популярных операционных систем. Однако, вместо перекомпиляции под разные платформы требуется установка Python-интерпретатора на целевой операционной системе.

Swing является библиотекой для удобного создания графического интерфейса для программ на языке Java, которая была разработана Sun Microsystems. Он содержит ряд графических компонентов, таких как кнопки, поля ввода, таблицы и т. д. Swing относится к библиотеке классов JFC, которая представляет собой набор библиотек для разработки графических оболочек. К этим библиотекам относятся Java 2D, Accessibility-API, Drag & Drop-API и AWT.

Проанализировав все решения, было решено остановиться на библиотеке

Swing и языке программировании Java и Kotlin. Все решения в общем случае подходили, но остановиться на Swing было решено из-за удобной среды разработки от компании JetBrains: IntelliJ IDEA в котором представлен графический редактор для удобного добавления основных UI-компонентов таких как: кнопки, поля ввода, текстовые строки, картинки и т.д.

В качестве основного языка был выбран язык Kotlin, поскольку, как более молодой язык, в котором разработчики постарались включить только лучшие качества современных языков программирования, он является предельно лаконичным, компактным и понятным. Огромное количество кода, которое приходилось раньше писать на Java теперь можно просто проигнорировать.

При разработке утилиты использовался архитектурный паттерн MVP (Model-View-Presenter или Модель-Представление-Презентер).

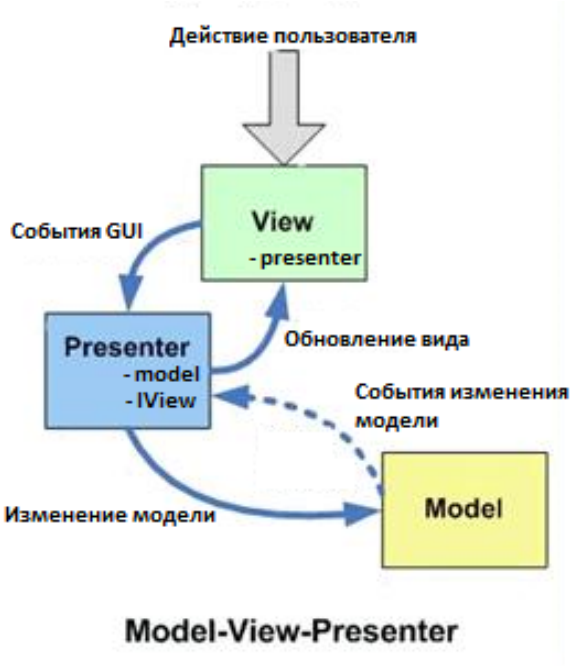


Рис 3.11 MVP-структурная схема

Основная идея подхода заключается в том, что представление оборачивается интерфейсом с ранее определенным протоколом взаимодействия, и этот интерфейс передается в презентер. Представление отвечает за взаимодействие с пользователем: при каком-либо нажатии передает управление презентеру, который реагирует на событие, выполняет бизнес-логику, в которой может менять состояние модели и обновляет состояния представления если необходимо.

Основными характеристиками, который этот паттерн обладает является

- Представление не имеет доступа к Модели;
- Презентер привязан к одному Представлению;
- Представление полностью пассивно;

Такое разделение слоёв позволяет лучше тестировать приложение, поскольку заранее известно, на каком уровне нужно искать код.

В роле Представления выступал класс, в котором происходит автоматическое связывание виджетов, которое генерируется на основе XML-файла полученный средой разработки при ручном добавлении кнопок, текстовых элементов и других виджетов на панель. В Презентере находилась основная бизнес-логика утилиты: открытие файлов, реализация алгоритма по поиску областей выделения и т.д.

Разработка утилиты происходила поэтапно на первом этапе были поддержаны такие функции как:

1. Автоматическая генерация двух изображений подряд:

- первое, простое изображение
- второе, после выделения в случайной области экрана

2. Выбор времени задержки между выделениями
3. Выбор режима остановки генерации

- по времени
- по количеству изображений

Далее, в ходе генерации обучающей выборки появилось потребность в реализации дополнительного функционала, который был реализован на втором этапе:

1. Выбор области, в котором будет происходить выделение
2. Выбор изображения, с которым будет происходить сравнение

Полученная утилита изображена ниже:

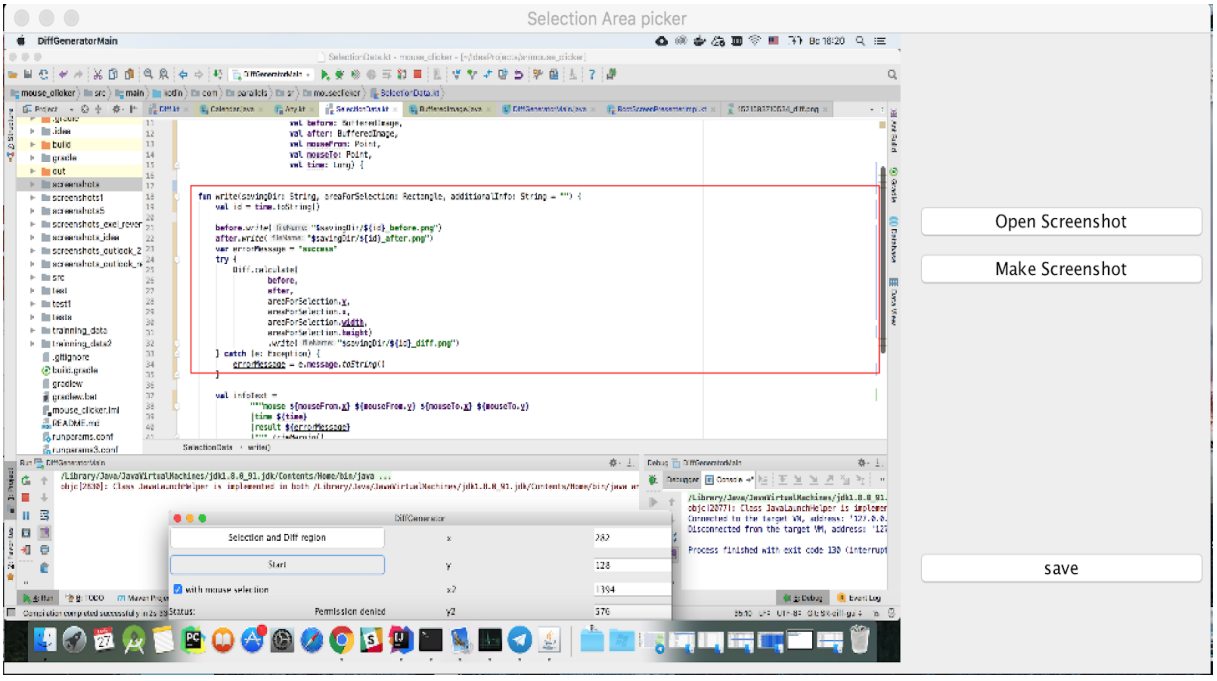


Рис 3.12 Пример выделения активной области

Красным отмечена область, в которой будут происходить выделения. Это очень важная доработка, так как она помогла исключить случайные нажатия мышкой активных элементов в ходе генерации обучающей выборки. Эти случайные нажатия возникали по двум причинам, во-первых, потому что перед генерацией следующего изображения необходимо было сбросить область выделения предыдущего и для этого просто вызывался клик мышью, который и приводил к нажатиям на случайные элементы. Во-вторых, при первоначальном выделение происходило нажатие левой кнопкой мыши, чтобы инициировать начало выделения, однако, иногда в ходе таких действий тоже случайно происходили нажатия на активные элементы, которые приводили к непредвиденным последствиям, вплоть до закрытия программы, для которой генерировались изображения.

Изображение стартового экрана утилиты:

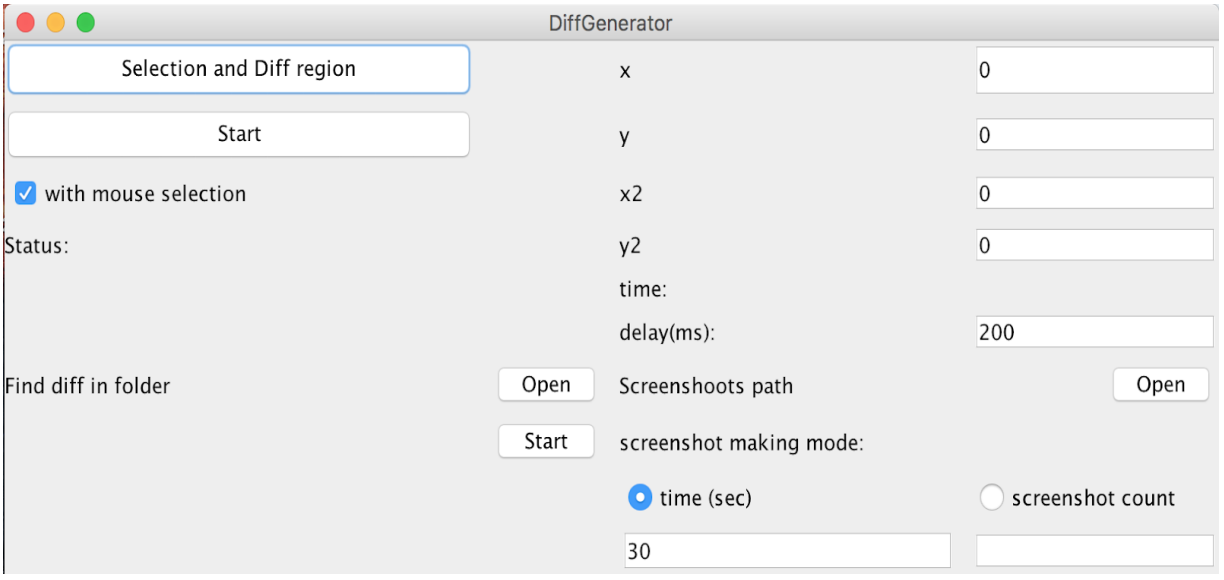


Рис 3.12 Главный экран утилиты

Но все равно практика показала, что генерация обучающей выборки нетривиальная задача даже с таким вспомогательным инструментом. Поскольку

на первых шагах алгоритма по поиску областей выделения происходит попиксельное сравнение, то очень важно, чтобы в двух изображениях не было отличий, кроме собственно самого выделенного текста. Однако практика показала, что этого сложно добиться, например в MS Office при выделении текста автоматически появлялось контекстное меню, в IntelliJ IDEA появлялась лампочка с подсказками и дополнительно выделялись слова, на которые попадал фокус и т.д. Чтобы добиться приемлемого результата приходилось к каждой программе подходить индивидуально и выключать активные элементы в настройках или грамотно подбирать область выделения при помощи нового функционала, в которой нет дополнительных областей выделения.

После того, как утилита для генерации обучающей выборки была готова, было сгенерировано порядка 8000 изображений для самых популярных программ, таких как:

- Microsoft Office
- Microsoft Outlook
- Microsoft Excel
- Google Chrome

Глава 4

Разработка модуля по поиску областей выделения

4.1 Предпосылки

После того, как была сгенерирована обучающая выборка можно было приступать к разработке модуля по поиску областей выделения текста на изображении. Иными словами, решить задачу сегментации.

Сегментация - это процесс разбиения цифрового изображения на несколько сегментов (множества пикселей, также называемые суперпикселями). Цель сегментации состоит в том, чтобы упростить и / или изменить представление изображения во что-то более содержательное и более легкое для анализа. В данном случае изображение будет разбиваться на области где есть выделение и где его нет. Сегментация изображений обычно используется для поиска объектов и границ (линий, кривых и т. д.)Точнее, сегментация изображения представляет

собой процесс присвоения метки каждому пикселю изображения, так что пиксели с одним и тем же ярлыком имеют определенные характеристики.

А в качестве алгоритма, который будет сегментировать изображения по принципу наличия областей выделения были взяты нейронные сети, а в частности сверточные нейронные сети.

4.2. Сверточные нейронные сети

Сверточные нейронные сети [5] очень похожи на обычные нейронные сети: они также построены на основе нейронов, которые обладают изменяющимся весом и смещениями. Каждый нейрон получает некоторые входные данные, выполняет скалярное произведение информации и в отдельных ситуациях сопровождает это нелинейностью. Как и в случае с обычными нейронными сетями, вся сверточная нейронная сеть выражает одну дифференцируемую функцию взноса (эффективный взнос): с одной стороны это необработанные пиксели изображения, с другой — вывод класса или группы вероятных классов, характеризующих картинку. Здесь также присутствует функция потери на последнем (полностью подключенном) слое, а все «фишки» и хитрости, разработанные и изученные в течение знакомства с нейронными сетями, остаются справедливыми при работе с CNN.

Очень важным преимуществом сверточных нейронных сетей является то, что они хорошо масштабируются для изображений больших размеров. Предположим, у нас обычная нейронная сеть, которой на вход передаются изображения размером [32×32×3] , где ширина и высота изображения равны 32 и оно трехканальное. Тогда один полностью подключенный нейрон в первом скрытом слое обычной нейронной сети обладает весом 3 072 (32*32*3). В случае же если входное изображение имеет размер [1920×1080×3] (обычный Full-HD

экран), то соответственно вес будет 6 220 800, а обычно вставляют несколько таких нейронов, а с учетом того, что это все будет запускаться на мобильных устройствах, то размер и производительность нейронной сети тут крайне важны.

Основными слоями из которых состоят сверточная нейронная сеть являются:

1. свертка
2. субдискретизация
3. полносвязный слой

4.2.1 Слой свертки

Слой свертки умножает значения фильтра на исходные значения пикселей изображения, после чего все эти умножения суммируются. Каждая уникальная позиция введенного изображения производит число. Пример: Если используется 12 фильтров, объем будет равен 32*32*12 [32×32×12].

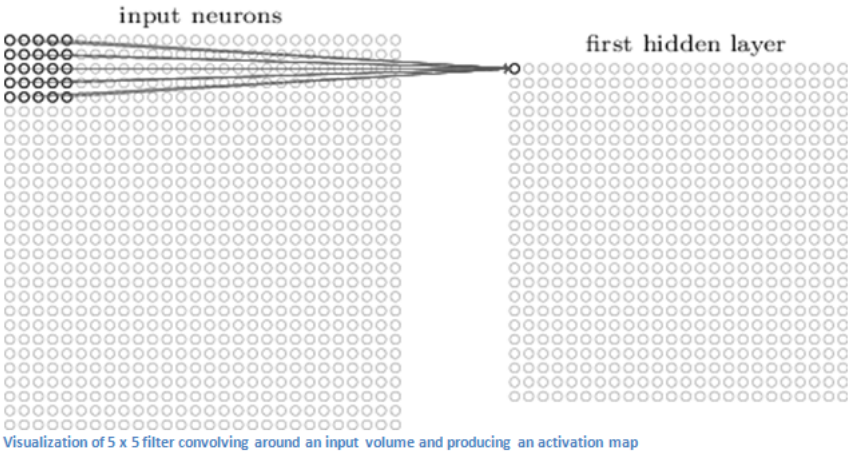


Рис 4.1 Свертка

Параметры слоя свертки состоят из набора обучающихся фильтров. Каждый фильтр имеет малые пространственные габариты: ширину и высоту, но проходит по всей глубине объема ввода. Например, стандартный фильтр первого слоя сверточной нейронной сети может быть размера [5x5x3]. Во время прохода вперед, мы он проскальзывает каждый фильтр по ширине и высоте входных данных и вычисляем скалярное произведение между записями фильтра и входом в любое положение. По мере прохождения фильтра по ширине и высоте изображения, мы составляется 2-мерную активационная карта, которая предоставляет отклики этого фильтра на каждой пространственной позиции. Сеть выучивает фильтры, активирующиеся при обнаружении некоторой визуальной особенности. Это может быть грань некоторой направленности, пятнистость конкретного цвета на первом слое или кольцеобразные узоры на более высоких уровнях сети. А когда работа будет происходить с целым набором фильтров в каждом слое свертки, то каждый из них будет формировать отдельную 2-мерную активационную карту. Так, будут складываться эти активационные карты вдоль измерения «глубина» и формироваться выходной объем.

В таком построении можно нейроном назвать запись в выходном объеме, которая смотрит на небольшой участок входного объема и пространственно делит параметры со всеми нейронами слева и справа.

Для сверточного слоя существует три гиперпараметра, которые контролируют размер выходной информации:

1. **Глубина**. Она эквивалентна числу фильтров, которые мы бы хотели использовать для данного сверточного слоя; каждый из фильтров обучается нахождению различных данных на входе. Например, первый сверточный слой в качестве входной информации берет

необработанное изображение, после различные нейроны, располагающиеся вдоль глубины, могут активироваться в случае наличия, например, сгустков определенного цвета.

2. **Шаг**, с которым фильтр выполняет проход. Когда шаг равен 1, то за один раз фильтры перемещаются на 1 пиксель. Когда шаг равен 2, фильтры «перепрыгивают» через 2 точки за раз при каждом обращении к ним. Это позволит вырабатывать меньшие выходные объемы пространственных измерений.
3. Иногда бывает удобно границу входного изображения **окружать нулями**. Размер этого дополнения нулями и есть гиперпараметром. Ключевой особенностью дополнения нулями является то, что оно позволяет контролировать пространственный размер выходных объемов.

4.2.2 Слой субдискретизации

В архитектуре сверточных нейронных сетей обычной практикой является вставка слоя субдискретизации между последовательностями сверточных слоев. Его функция состоит в постепенном уменьшении пространственные габариты изображения с целью уменьшения количества параметров и вычислений в сети, а также контроля переобучения. Слой пулинга работает независимо от среза глубины входных данных и масштабирует объем пространственно, используя функцию максимума. Чаще всего используется слой с фильтрами размера 2×2 с шагом 2; подобный слой снижает дискретизацию каждого среза глубины входа в 2 раза как по ширине, так и по высоте, откидывая при этом 75% активация. Каждая

операция MAX в этом случае будет выбирать максимальное значение из 4 чисел. Размер по глубине при этом остается неизменным.

Основные гиперпараметры:

1. Размер
2. Шаг

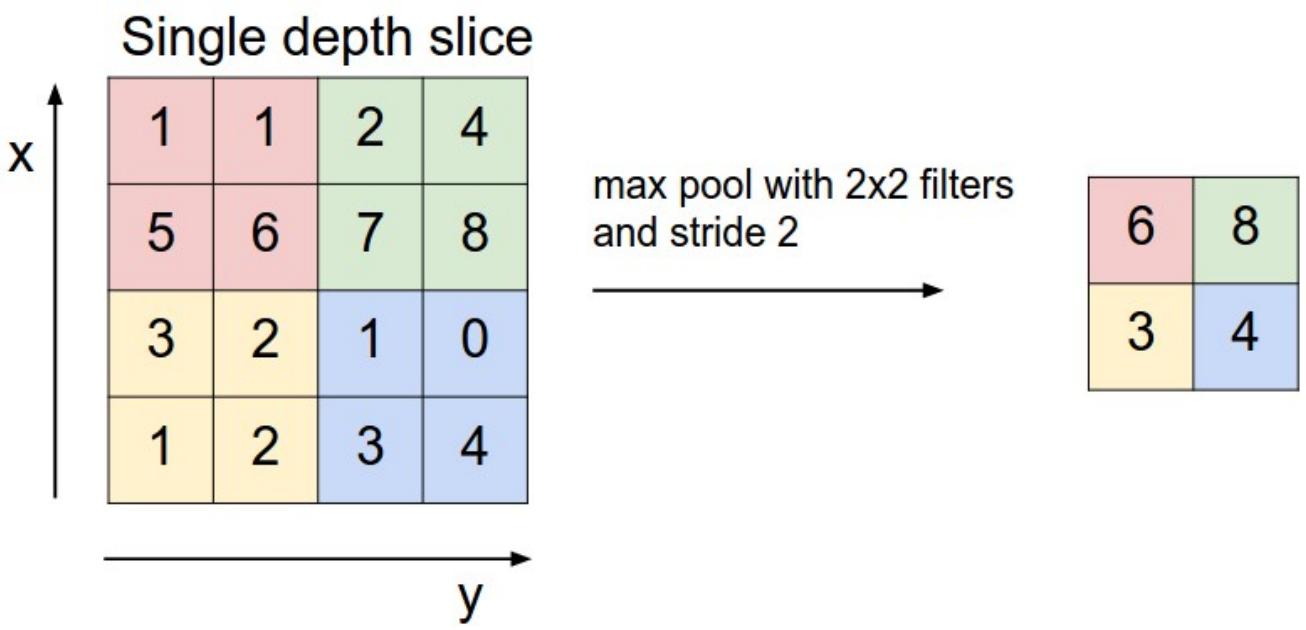


Рис 4.2 Слой субдискретизации

4.2.3 Полносвязный слой

Точно так же, как и в обычных нейросетях, в полносвязном слое нейроны обладают полными соединениями со всеми активациями в предыдущем слое. Их активации могут быть вычислены посредством умножения матриц, сопровождающегося смещением.

4.3. Подбор архитектуры сверточной нейронной сети

Определив основные структурные компоненты сверточной нейронной сети можно приступать к подбору ее архитектуры.

4.3.1. Метрики

Однако, прежде чем начать подбор архитектуры нейронной сети необходимо ввести метрики по которым будет определяться качество ее работы. Поскольку сверточная нейронная сеть должна каждый пиксель входящего изображения должна определить принадлежит ли он области выделения или нет, то в этих терминах введем следующие понятия:

- TP (True Positive) — истиноположительный. Нейронная сеть решила, что пиксель принадлежит области выделения и он действительно ей принадлежит.

- FP (False Positive) — ложноположительный. Нейронная сеть решила, что что пиксель принадлежит области выделения, однако на самом деле он не является ей. Это так называемая ошибка первого рода. Она не так страшна, как ошибка второго рода, особенно в тех случаях, когда нейронная сеть — тест на какое-нибудь заболевание.

- FN (False Negative) — ложноотрицательный. Нейронная сеть решила, что пиксель не принадлежит области выделения, однако он принадлежит ей. Это так называемая ошибка второго рода. Обычно при создании модели желательно минимизировать ошибку второго, даже увеличив тем самым ошибку первого рода.

- TN (True Negative) — истиноотрицательный. Нейронная сеть решила, что пиксель не принадлежит области выделения и он действительно ей не принадлежит.

Соответственно метрики, которые базируются на этих терминах:

$$Accuracy = \frac{tp+tn}{tp+fp+fn+tn}$$

Самая простая метрика - метрика достоверности. Показывает сколько пикселей было определено к области и не к области выделения суммарно правильно по отношению ко всему количеству пикселей. В данном случае эта метрика может очень сильно вводить в заблуждение, так как обычно области выделения занимают относительно небольшую площадь на изображении и , допустим, нейросеть начнет просто считать, что на изображении нет областей выделения и закрашивать его черным цветом, то из-за малости области выделения эта метрика все равно будет показывать хороший результат, что может вводить в заблуждение.

Однако более показательными в данном случае являются другие метрики: точность и полнота.

Точность показывает отношение верно угаданных объектов класса ко всем объектам, которые мы определили как объекты класса. Или в данном случае: точность - это отношение правильно угаданных пикселей определенных как область выделения ко всем пикселям, которые были определены как область выделения.

$$\text{Precision} = \frac{tp}{tp + fp}$$

Полнота показывает отношение верно угаданных объектов класса ко всем представителям этого класса. Или в данном случае: полнота - это отношение правильно угаданных пикселей определенных как область выделения ко всем пикселям, которые правильно были указаны как область и как не область выделения.

$$\text{Recall} = \frac{tp}{tp + fn}$$

Есть ещё F1-мера, которая показывает среднее гармоническое точности и полноты. Она помогает сравнить модели, используя только одну числовую меру.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

4.3.2. Подбор базовой архитектуры сети

Поскольку сверточная сеть будет работать в качестве фильтра исходного изображения, то полносвязный слой в подборе архитектуры участвовать не будет.

Будут применяться только сверточные слои и слои субдескретизации размером 2 на 2.

Существуют разные способы подбора архитектуры нейронной сети. Суть их заключается в том, что берутся гиперпараметры нейронной сети и варьируются различным образом, потом сеть обучается и выбираются те, на которых нейронная сеть показывает лучшее качество. Соответственно, способом варьированием этих гиперпараметров и отличаются разные подходы.

Был взят метод случайного поиска, суть которого в том, что гиперпараметры подбираются случайным образом[6]. Потом находятся наиболее оптимальный набор параметров и уже в небольшом радиусе происходит поиск новых гиперпараметров.

За основу было взято то, что будет использоваться 2 слоя субдискретизации, а количество слоев свертки, их размер и количество фильтров будет распределяться следующим образом:

И, соответственно, размер слоя свертки:

	1	3	5
до 1го слоя	0.4	0.4	0.2
между 1м и 2м	0.4	0.4	0.2
после 2го	0.4	0.4	0.2

Значения в ячейках таблицы - вероятность того, что будет выбрано столько фильтров для конкретного слоя свертки.

Количество фильтров:

	4	8	16	32	64
до 1го слоя	0.4	0.4	0.1	0.05	0.05
между 1м и 2м	0.1	0.3	0.3	0.15	0.05

после 2го	0.05	0.2	0.35	0.25	0.15

Значения в ячейках таблицы - вероятность того, что будет выбран такой размер фильтра для конкретного слоя свертки.

Количество размер фильтров выбирался по принципу, чем глубже слой, тем больше размер фильтра, по аналогии с архитектурами AlexNet[[10] и VGG.

Количество слоев свертки между слоями субдискретизации выбиралось случайным образом от 2 до 5.

Поскольку в распоряжении имелся только MacBook Pro 2014, на котором процесс обучения занимал очень много времени и при этом блокировал основную работу, то необходимо был арендован сервер на базе Google Cloud Platform с видеокартой NVIDIA Tesla K80.

Кроме того важно отметить, что в нейросети обучались только на изображениях полученных для Google Chrome. Это связано с тем, что при попытках в обучающую выборку конкретных нейросетей добавить дополнительно другие изображения, то нейронные сети начинают чаще ошибаться и их качество заметно деградирует. Это связано с тем, что в других программах по-другому выделяется текст: он может быть другого текста, другой структуры(в Excel выделяется строка и столбец, а так же название строки и столбца). Вероятно, эту проблему можно решить, если параллельно понимать какая программ сейчас запущена для более глубокой подстройки нейронной сети под конкретную область выделения.

Всего было сгенерировано 30 сетей:

	до 1го слоя	между 1м и 2м	после 2го	F1-мера

1	(4, 1), (4, 1), (4, 1), (16, 1)	(4, 1), (16, 1), (16, 3), (64, 1)	(32, 5), (32, 1), (32, 1), (32, 1)	0,4006
2	(4, 1), (32, 5), (8, 3)	(16, 3), (16, 5), (32, 3), (16, 3)	(32, 3), (16, 3), (64, 3), (32, 5)	0.9892
3	(4, 1)	(32, 1), (64, 5), (4, 1)	(16, 3), (16, 1), (4, 3)	0.9632
4	(4, 3), (4, 3)	(8, 1), (64, 5), (32, 5), (32, 3)	(16, 1)	0.9631
5	(4, 3), (8, 1), (16, 3)	(4, 1), (8, 5)	(8, 1)	0.9562
6	(4, 3), (8, 3)	(8, 1), (8, 5), (4, 1)	(32, 1), (16, 3), (64, 5)	0.9910
7	(4, 3), (16, 1)]	(64, 1), (16, 3), (16, 1)	(16, 3)	0.8986
8	(4, 3)	(16, 3), (8, 3)	(64, 5), (32, 3)	0.9719
9	(4, 5), (8, 3), (32, 1)	(64, 5), (4, 3), (32, 1), (32, 3)	(32, 3), (32, 3)	0.9867
10	(4, 5)	(16, 3), (32, 3), (16, 1)	(64, 1), (16, 1)	0.9021
11	(8, 1), (4, 1), (4, 1)	(16, 5), (16, 5), (64, 3), (8, 3)	(32, 1), (16, 1), (64, 3)	0.8519
12	(8, 1), (4, 3)	(8, 1), (4, 3), (8, 3), (4, 1)	(64, 3), (16, 3), (8, 3)	0.9599
13	(8, 1), (4, 5), (8, 5), (4, 5)	(32, 5), (8, 5), (8, 3)	(8, 1), (32, 1)	0.8670
14	(8, 1), (8, 1)	(16, 1)	(16, 1), (64, 3), (16, 1)	0.6565
15	(8, 1), (32, 3), (8, 1)	(16, 3), (32, 3)	(16, 3), (32, 1), (8, 5)	0.8739
16	(8, 3), (4, 3), (4, 1), (4, 1)	(32, 1)	(8, 1), (16, 3)	0.9099
17	(8, 3), (4, 3), (4, 3)	(16, 3), (4, 3)	(32, 1), (16, 3)	0.9621
18	(8, 3), (4, 3)	(4, 3), (16, 3), (16, 1)	(64, 5)	0.9710
19	(8, 3)	(8, 5), (32, 5), (32, 1), (16, 3)	(32, 5), (8, 3)	0.9956
20	(8, 3)	(16, 1), (4, 3), (4, 1)	(64, 1)	0.6764
21	(8, 3)	(16, 3)	(32, 1), (4, 3), (16, 1), (4,	0.8756

			1)	
22	(8, 5), (4, 1), (4, 1), (4, 3)	(8, 3), (4, 3), (4, 5), (32, 1)	(64, 3), (16, 1), (8, 3)	0.7128
23	(8, 5), (8, 1)	(8, 1), (8, 3), (16, 1)	(32, 1), (64, 1), (16, 3)	0.9942
24	(8, 5), (16, 1), (8, 1), (8, 1)	(8, 1), (4, 3), (32, 5)	(16, 3), (16, 1), (16, 3), (16, 1)	0.9940
25	(8, 5)	(64, 5)	(16, 1), (8, 3), (16, 3), (8, 3)	0.9911
26	(16, 3), (4, 1)	(64, 1), (4, 5)	(16, 1), (16, 5)	0.9767
27	(16, 5)	(8, 3)	(32, 1), (16, 3), (8, 1)	0.9914
28	(32, 1), (8, 3)	(16, 1)	(16, 5), (32, 3), (16, 3), (64, 3)	0.9850
29	(32, 3), (4, 1), (4, 1), (4, 1)	(4, 1)	(8, 3), (8, 5), (32, 5), (64, 5)	0.9908
30	(64, 1), (8, 5), (4, 3), (4, 5)	(32, 1), (32, 3)	(16, 5), (8, 1), (32, 1)	0.9084

(n,m) - пара обозначает:

n - число фильтров, m - размер фильтра, столбец F1-меры заполнен значениями на 2000 шаге.

График зависимости значения F1-меры от количества шагов приведен ниже.

Хотелось бы отметить, что значения на графики для читаемости усреднены скользящим средним, кроме того видны выбросы, которые возникают из-за неидеальности обучающей выборки.

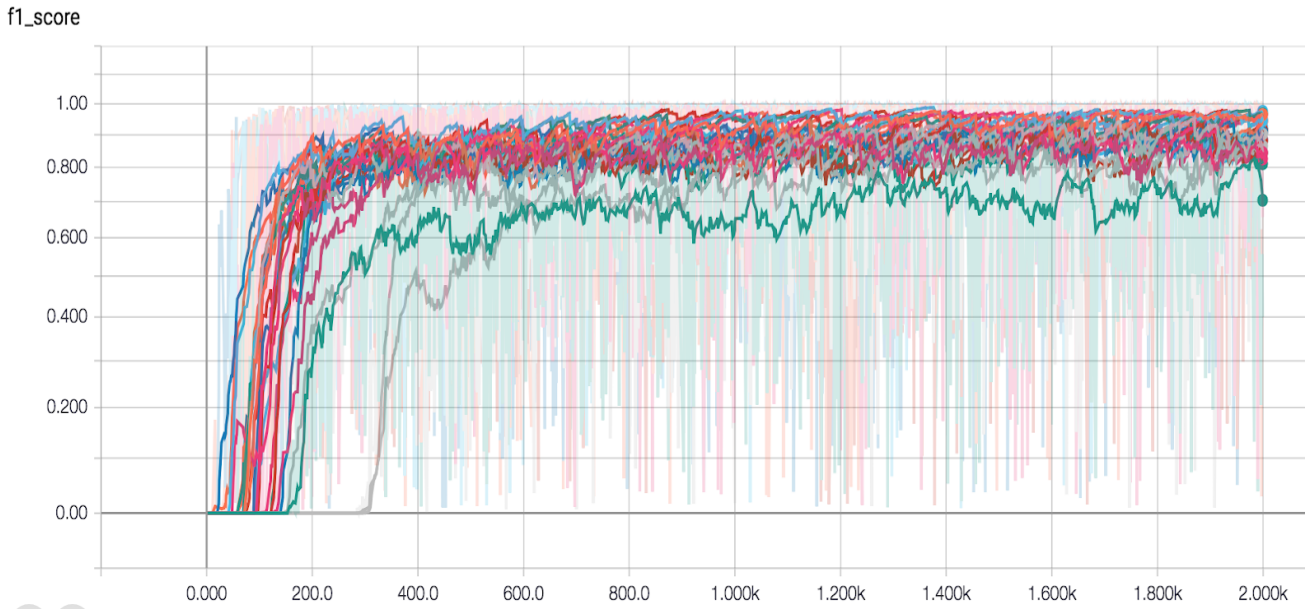


Рис 4.3 Подбор параметров. Архитектуры

Видно, что довольно-таки у многих сетей получилось значение F1-меры большее 0.99:

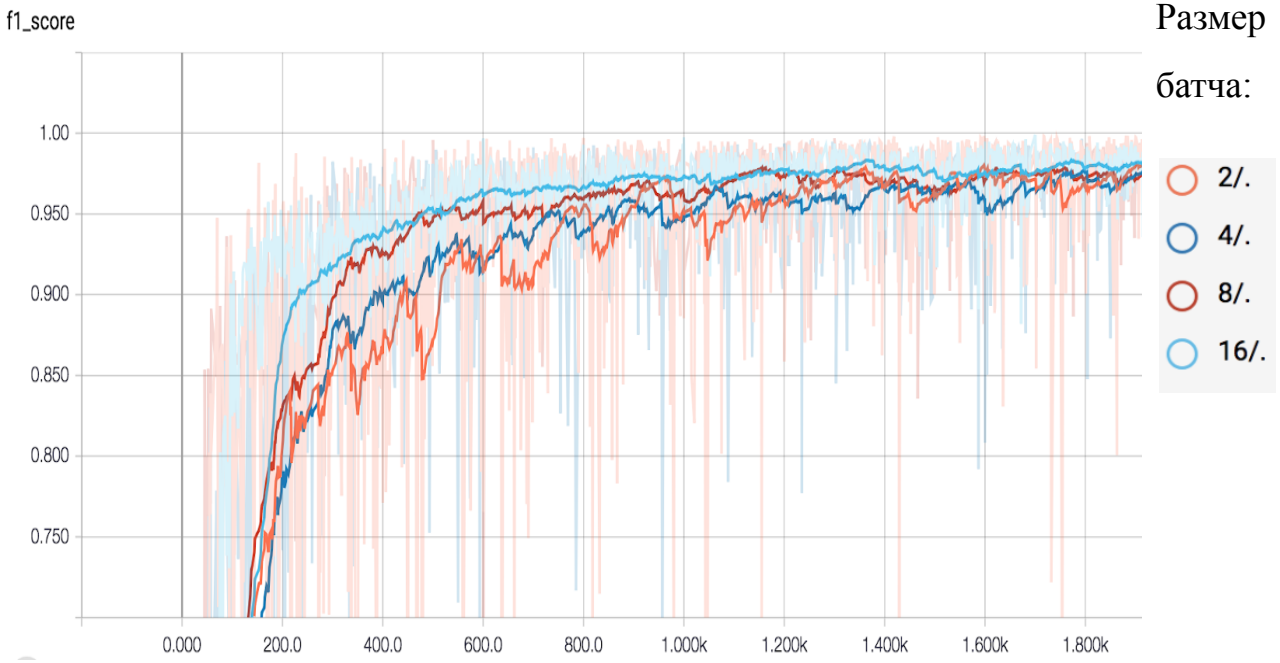
6	(4, 3), (8, 3)	(8, 1), (8, 5), (4, 1)	(32, 1), (16, 3), (64, 5)	0.9910
19	(8, 3)	(8, 5), (32, 5), (32, 1), (16, 3)	(32, 5), (8, 3)	0.9956
23	(8, 5), (8, 1)	(8, 1), (8, 3), (16, 1)	(32, 1), (64, 1), (16, 3)	0.9942
24	(8, 5), (16, 1), (8, 1), (8, 1)	(8, 1), (4, 3), (32, 5)	(16, 3), (16, 1), (16, 3), (16, 1)	0.9940
25	(8, 5)	(64, 5)	(16, 1), (8, 3), (16, 3), (8, 3)	0.9911
29	(32, 3), (4, 1), (4, 1), (4, 1)	(4, 1)	(8, 3), (8, 5), (32, 5), (64, 5)	0.9908

После того, как были опробованы разные архитектуры, были изучены влияние батч-нормализации и размер батча на обучение.

4.3.3. Влияние размера батча

Была испробована техника, когда во время обучения на вход нейронной сети подаются сразу несколько изображений, соответственно на каждом шаге обучения будет минимизироваться функция ошибки полученная сразу для нескольких изображений, что в теории должно увеличить скорость обучения.

Для тестирования была взята сеть под номером 19:



По графику видно, что чем больше количество одновременно подаваемых изображений, тем за меньшее количество шагов нейронная сеть обучается, однако если измерить суммарное время обучения для разных размеров батча:

размер батча	1	2	4	8	16
время,с	653	1 110	1 658	2 459	3 201

Видно, что время растет практически линейно, причем скорость роста времени значительно выше, чем уменьшение количества шагов и при этом поэтому в данном случае увеличение размера батча не зарекомендовало себя.

4.3.4. Влияние батч-нормализации

В начале 2015 года был предложен метод, позволяющий ускорить обучение нейронных сетей[. Метод решает следующую проблему, которая мешала эффективному обучению нейронных сетей: когда сигнал распространяется по нейроной сети, даже если он подавался нормализованным на входе, то когда он проходит через внутренние слои, то его математическое ожидание и дисперсия могут сильно измениться (данное явление называется внутренним ковариационным сдвигом), что чревато серьезными несоответствиями между градиентами на различных уровнях. Поэтому приходится использовать более сильные регуляризаторы, замедляя тем самым темп обучения.

Однако существует довольно-таки простое решение такой проблемы: нормализовать входные данные таким образом, чтобы их математическое ожидание было равно нулю, а дисперсия единице. Нормализация выполняется перед входом в каждый слой. Таким образом происходит нормализация n-примеров , где n- размер батча, а во время тестирования происходит нормализация статистики, которая берется на основе всего обучающего множества, так как увидеть заранее все тестовые данные невозможно. Так происходит получения значения математического ожидания и дисперсии для определенного батча[4] :

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$

Далее, функция активации преобразуется так, чтобы ее математическое ожидание было равно нулю, а дисперсия равна единице:

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

где ε > 0 — параметр, нужен для того, что оградить дробь от деления на ноль (в случае, если среднеквадратичное отклонение батча очень мало или даже равно нулю). Для того, чтобы вычислить окончательную функцию активации *y*, необходимо убедиться, что во время нормализации не была потеряна способность к обобщению, и так как к исходным данным была применена операция сдвига и масштабирования, то можно выбрать произвольные масштабирование и сдвиг нормализованных значений.

Тогда окончательная функция активации:

$$y_i = \gamma \hat{x}_i + \beta$$

Где β и γ — параметры батч-нормализации, которым системы можно обучить (их

можно оптимизировать методом градиентного спуска на обучающих данных). Это обобщение также означает, что батч-нормализацию может быть полезно применять непосредственно к входным данным нейронной сети.

Батч-нормализация почти всегда увеличивает скорость обучения глубоких нейронных сетей. Более того, она может случить отличным *регуляризатором*, позволяя не так осмотрительно выбирать темп обучения. Регуляризация здесь — следствие того факта, что результат работы сети для определенного примера больше *не детерминировано* (он зависит от всего батча, в рамках которого данный результат получен), что упрощает обобщение.

Аналогично, для сеть под номером 19 была реализована батч нормализация для разных размеров батчей.

Результаты:

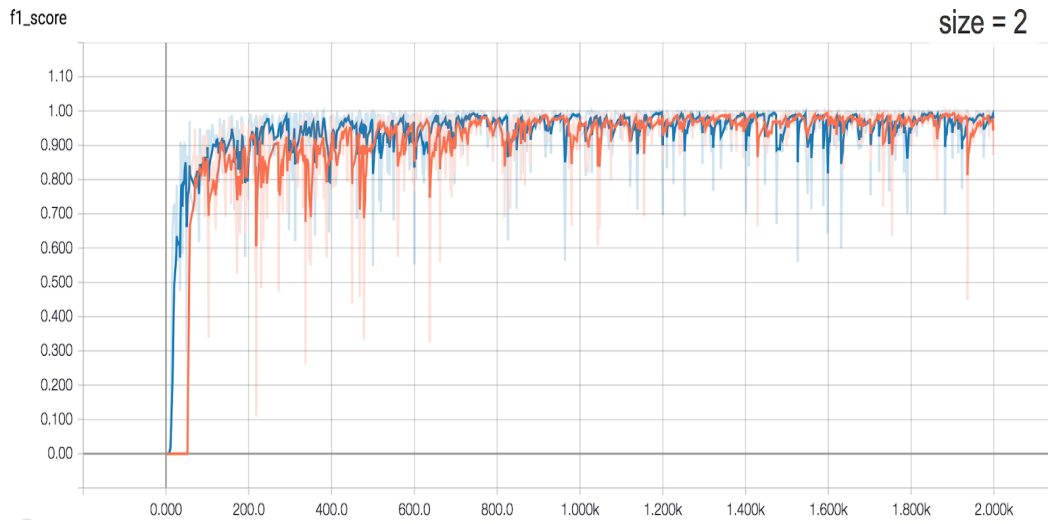


Рис 4.5 Размер батча 2

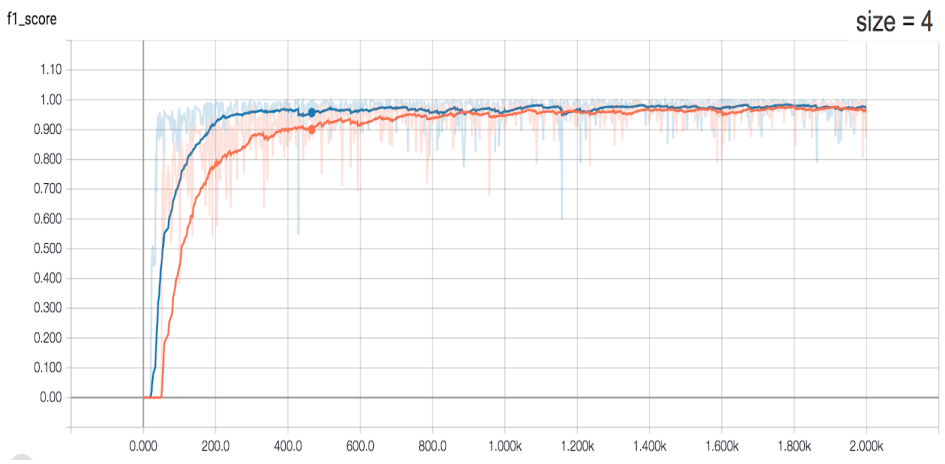


Рис 4.6 Размер батча 4

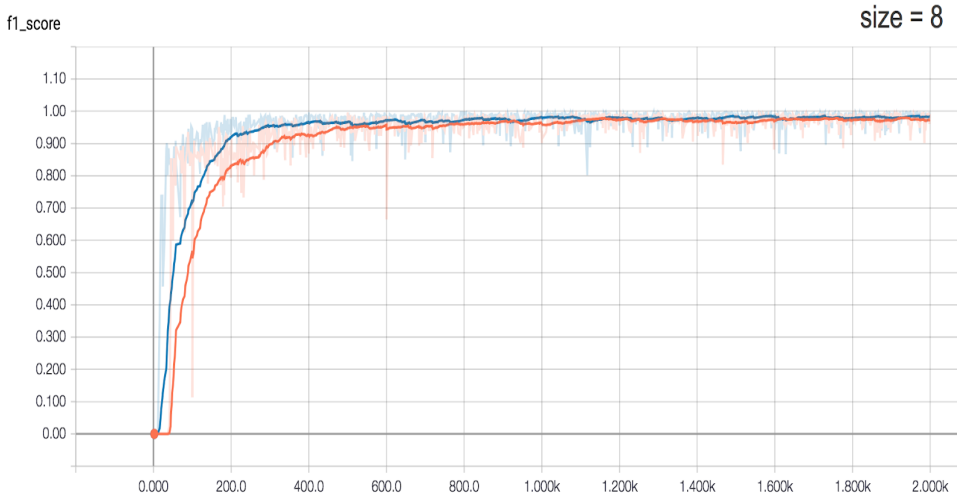


Рис 4.7 Размер батча 8

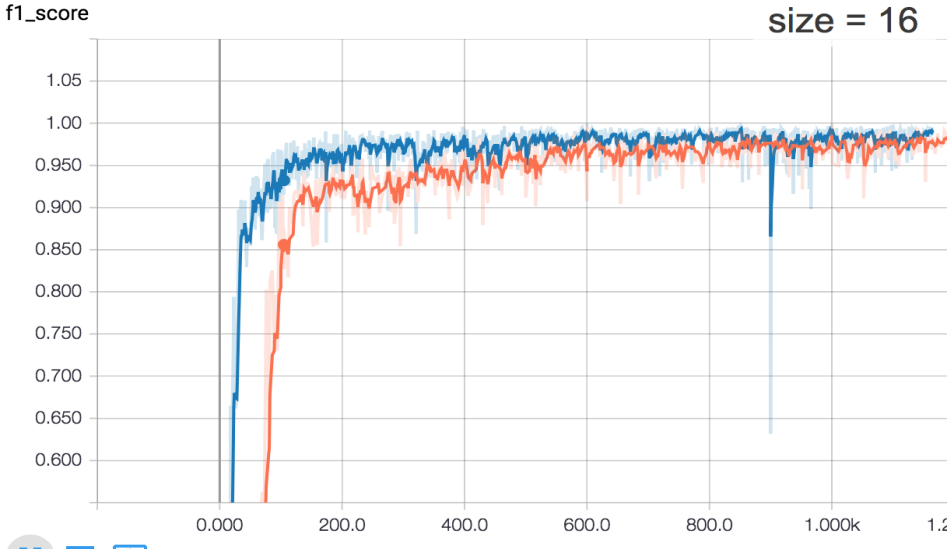
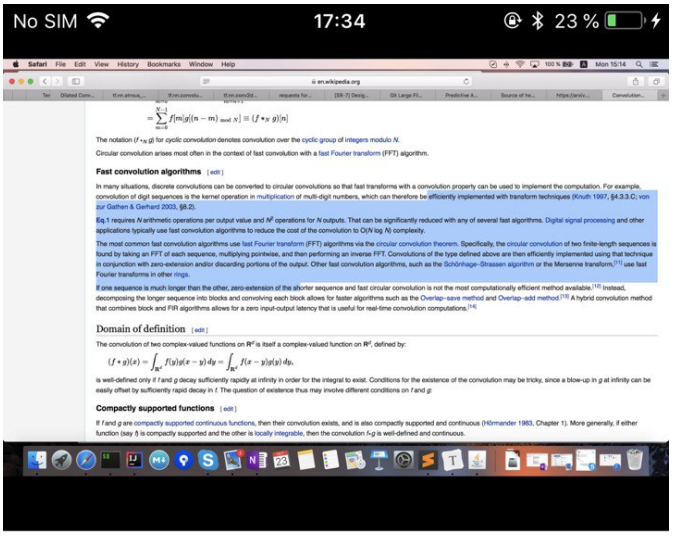


Рис 4.8 Размер батча 16

Видно, что в данном случае батч-нормализация работает хорошо только когда размер батча становится больше 4.

4.4 Пример изображения. Анализ производительности

Ниже изображен результат применения 19 нейросети на верхнее изображения из Google Chrome



3 Prev Next Run Run ∞ Stop Run All

Видно, что в принципе получилось приемлемое качество, однако время обработки заняло порядка одной секунды, что является недостаточным для обеспечения

непрерывного потока изображений и поиска областей выделения на них без дополнительных проверок.

Глава 5

Заключение

5.1. Выводы

В результате данной работы было сделано:

1. Разработан алгоритм генерации обучающей выборки
2. Реализована утилита, автоматизирующая генерацию обучающей выборки
3. Сгенерированы изображения для основных программ
   - Microsoft Office
   - Microsoft Outlook
   - Microsoft Excel
   - Google Chrome
4. Рассмотрены разные архитектуры нейронных сетей
5. Исследовано влияние размера батча и батч-нормализации на обучение

5.2. Планы на дальнейшую работу

Как было сказано ранее, что получилось добиться хорошего результата только для изображений, полученных для Google Chrome. Соответственно,

необходимо расширить поддерживаемые приложения, кроме того необходимо обучить сеть распознавать больше различных цветов, которые могут выступать в качестве выделения.

Также есть идеи попробовать дообучить уже предобученные модели, которые решают подобную задачу. В частности решение от компании Google - Tensorflow Object Detection API.

Дополнительно есть идеи добавить маленькую легковесную сеть, которая будет определять открытую программу, чтобы более тщательно подстраиваться под ее области выделения.

И, определенно, необходимо поработать над производительностью работы нейронной сети на мобильных устройствах.

Литература

[1] Гонка технологий. Как искусственный интеллект помогает бизнесу. http://www.forbes.ru/tehnologii/354727-gonka-tehnologiy-kak-iskusstvennyy-intellekt-pomogaet-biznesu

[2] Introduction to TensorFlow Lite. https://www.tensorflow.org/mobile/tflite/

[3] Najman, Laurent. Mathematical morphology: from theory to applications / Laurent Najman, Hugues Talbot // ISTE-Wiley. — 2010. — P. 520.

[4] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift / Sergey Ioffe, Christian Szegedy

[5] Going deeper with convolutions / Christian Szegedy, Wei Liu, Pierre Sermanet et al. // The Computer Vision Foundation. — 2014. — Pp. 1–12

[6] Random Search for Hyper-Parameter Optimization / James Bergstra, Yoshua Bengio

[7] Cross-platform software development for embedded & desktop / https://www.qt.io/

[8] The Python GUI Project / http://www.cosc.canterbury.ac.nz/greg.ewing/python_gui/

[9] Pyforms / https://pyforms.readthedocs.io/en/v3.0/

[10] ImageNet Classification with Deep Convolutional Neural Networks/ Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton