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.

Семантическая сегментация изображений с использованием глубоких свёрточных нейронных сетей

(магистерская диссертация)

Обучающийся: Слободсков Игорь Олегович

_____
(подпись обучающегося)

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

_____
(подпись научного руководителя)

Москва 2018

Аннотация

Целью данной работы является исследование применимости глубоких свёрточных нейронных сетей для распознавания элементов графического интерфейса пользователя ПК. Нейронная сеть должна быть достаточно простой и быстрой для работы на телефонах с ОС Android и iOS - это необходимо для возможности редактировать область выделения текста при удалённом подключении к ПК при помощи элементов интерфейса телефона (перемещения пинов по краям области выделения).

В данной работе искались области выделения текста. Нейронные сети были реализованы с помощью библиотеки tensorflow. Были исследованы различные приёмы и архитектуры сетей, появившиеся в последние годы. Проведены эксперименты и сделано сравнение разных подходов с точки зрения применимости к данной задаче. Разработана простая и эффективная архитектура, позволяющая за небольшое время (200-500мс) распознавать и находить область выделения с помощью телефона.

Содержание

Введение **4**

1 Постановка задачи **8**

1.1 Ограничения 8

1.2 Семантическая сегментация 10

1.3 Критерии качества работы сети 12

2 Архитектура сети **14**

2.1 Более короткие пути внутри сети 18

2.2 Архитектура типа res net 18

2.3 Batch Normalization 20

2.4 Комбинирование свёрточных слоёв 21

2.5 Слои Xception 21

2.6 Сравнение max-пулинга и свёртки с шагом 22

2.7 Используемые функции активации 23

2.8 Уменьшение входного изображения 24

2.9 Вычисления с фиксированной точкой 26

2.10 Выбранная архитектура 28

3 Обучение сети **32**

3.1 Используемые изображения 32

3.2 Функция ошибки 33

3.3 Изменение шага обучения 34

4 Результаты **36**

4.1 Скорость работы 36

4.2 Качество распознавания 37

4.3 Применимость на мобильных устройствах 37

Заключение **41**

Приложение А. Слои свёрточной сети **46**

Введение

В последние несколько лет нейронные сети получили широкое распространение. Нейронные сети могут решать целый ряд различных задач: распознавание и поиск объектов на изображениях, перенос стилей с одной фотографии на другую, распознавание речи, работа с текстом, выявление неисправностей в сложных системах, оценка рисков в банковской сфере.

Нейронные сети активно развиваются - в последние несколько лет были предложены новые, более эффективные подоходы, касающиеся деталей реализации нейронных сетей и их архитектур в целом. Идея Residual neural network была предложена командой исследователей из Microsoft в 2015 году, предложенный Google слой, названный Xception - в 2016, являющийся экстремально упрощённым вариантом архитектуры Inception, предложенной опять же Google. В данной работе были проанализированы архитектуры сетей, появившихся в последние несколько лет.

Возросшие аппаратные возможности и наличие готовых библиотек и фреймворков для обучения нейронных сетей позволяют использовать нейронные сети даже на мобильных устройствах, не говоря уж о работе на ПК с современными GPU.

Целью данной работы является выбор архитектуры нейронной сети, позволяющей с минимальными вычислительными затратами распознавать области выделения текста на скриншотах пользовательского графического интерфейса ПК. Это необходимо для того, чтобы при удалённом доступе с телефона можно было редактировать область выделения текста с телефона с помощью привычных и удобных для пользователя телефона средств - перемещения пинов по краям области выделения. (К сожалению, далеко не всегда можно получить область выделения текста прямым способом)

Решение должно работать для любого размера экрана, причём для стандартных размеров типа 1366×768 иметь приемлемую скорость работы (порядка 100-500 мс). Кроме того, по некоторым причинам распознавание должно производиться на мобильном телефоне, который обладает значительно меньшей производительностью, чем ПК.

Большинство существующих нейронных сетей работают с небольшими изображениями: например, сеть VGG [5], обученная на изображениях датасета ImageNet [4], принимает на вход изображения, приведённые к размеру 224×224.

Сеть с аналогичным количеством слоёв (около 16) и каналов (вплоть до 512), наверняка способна распознавать область выделения, но её обучение и скорость работы на полноразмерных скриншотах будут крайне медленными.

Подходы типа "взять несколько предобученных слоёв от VGG, добавить к ним ещё несколько и обучить их"неприменимы из соображений производительности, так как сложность вычисления зависит квадратично от количества каналов в слое.

В данной работе рассматриваются максимально простые сети - с небольшим количеством слоёв (8-16) и каналов в слое (не больше 64). Задача тоже более простая - семантическая сегментация с двумя классами - распознать область выделения и её отсутствие. Такие сети быстро обучаются (на компьютере с видеокартой время обучения одной сети составляет полчаса-час, причём довольно неплохие результаты могут достигаться уже в первые 5-10 минут обучения, дальше улучшение результатов довольно медленное), что позволило провести большое количество экспериментов и сравнить различные подходы.

Пользовательский интерфейс на ПК отличается от обычных изображений - не требуется устойчивость распознавания к повороту, интерфейс содержит много строго вертикальных или горизонтальных элементов. Интерфейс большинства програм строится по схожим принципам. Впрочем, цвет элементов - фона, текста, кнопок и области выделения может быть произвольным. Данные особенности можно учесть при выборе архитектуры сети и получить лучшие результаты.

Для обучения использовались свободные библиотеки с открытым исходным кодом Tensorflow и Keras (через api на Python), для измерения скорости работы использовалась библиотека tensorflow mobile (более новая tensorflow lite ещё не поддерживает некоторых видов слоёв, используемых в сети) через api на Java и C++, а так же предоставляемый вместе с библиотекой бенчмарк для телефона, выдающий потребление памяти и точное время работы для каждого слоя сети.

Tensorflow mobile не содержит функциональности по обучению сети, она способна только загружать и запускать заранее обученные сети. Библиотека содержит api для использования из java. Сама tensorflow написана на C++ и может использовать GPU с помощью библиотеки CUDA.

При создании сеть инициализируется случайными весами, поэтому результаты её обучения могут быть немного различными. Для большей достоверности

результатов измерения производились по нескольку раз.

К сожалению, не удалось найти датасетов с изображениями пользовательского интерфейса, поэтому для данной работы был подготовлен датасет - около 500 изображений для каждой программы - браузера Chrome, Word, Excel, Outlook, среды разработки IntelliJ IDEA. Изначально небольшая часть изображений была подговлена вручную, но большая часть изображений была создана при помощи специальных скриптов и потом с помощью обученной нейронной сети отобраны и вручную проверены и удалены неудачные образцы.

Так как каждый пиксель изображения является маленьким обучающим примером и при обучении использовались максимально простые сети, переобучения не возникало. Коэффициенты обученных сетей занимали 100-500Кб. При обучении часть ($10\% - 20\%$) изображений использовались для валидации работы сети и оставшиеся ($80\% - 90\%$) в качестве обучающих примеров.

В работе сравниваются различные приёмы и подходы к построению сетей, появившиеся в последние несколько лет. Некоторые их сочетания являются довольно интересными и многообещающими.

Итоговое качество распознавания сильно зависит от входных данных. Если обучать и проверять только на скриншотах браузера, то сеть работает превосходно. К сожалению, область выделения в Excel выглядит принципиально иначе - это прямоугольник, обводящий несколько ячеек. В Word панель меню и пустое пространство слева и справа от текста - синего цвета, что усложняет задачу распознавания. Обученная сразу на образцах сразу для всех программ сеть работает значительно хуже.

Возможно, подходы типа использования Average Global Pooling [18] слоёв позволят сделать внутри сети ещё одну маленькую сеть, учитывающую глобальную информацию о происходящем на экране, но в данной работе этот подход не показал хороших результатов (впрочем, это направление достойно дальнейшего изучения). Для сети с такой архитектурой понадобится намного больше обучающих примеров, чтобы не возникало переобучения.

В результате была предложена архитектура сети, являющаяся сочетанием идей от Residual Network [13], слоёв Xception [10] и использования разрежённой свёртки для работы с признаками разного масштаба без уменьшения размера картинки. Предложенная сеть быстро работает и обучается, а так же легко масштабируется, что позволяет адапировать её архитектуру как для простых

задач, так и для более сложных.

Были проведены измерения скорости работы на PC и на телефоне с ОС Android, сравнены разные архитектуры с точки зрения качества распознавания, скорости работы и времени обучения.

1 Постановка задачи

Необходимо распознавать область выделения текста на скриншоте пользовательского интерфейса (рисунок 1). Эта задача не имеет корректного решения: различные программы могут рисовать эту область различным цветом, а так же содержать элементы интерфейса, похожие на область выделения - например, квадратные синие кнопки с текстом. Что хуже, на скриншоте может быть картинка текста с выделением - и без знания того, что это именно изображение, а не текст, принять корректное решение невозможно. Выделение может быть произвольной формы или даже состоять из нескольких отдельных областей (рисунки 2, 3).

Уже существующая реализация содержит список вручную составленных правил - такое решение требует ручной настройки под каждую программу. Поиск области выделения используется для того, чтобы можно было, используя удобный для телефона интерфейс, редактировать область выделения при удалённом подключении к ПК.

Целью данной работы является исследование использования глубоких свёрточных нейронных сетей для данной задачи. Задача не имеет абсолютно точного решения, и имеет смысл оценивать нейронную сеть с точки зрения скорости работы, потребляемых ресурсов и точности распознавания.

1.1 Ограничения

Нейронная сеть будет запускаться на ПК, а так же на телефонах с ОС Android и iOS. Требуется, обработать изображение размером со скриншот экрана (например, 1920×1080) за минимальное время (в идеале - порядка $100 - 500$ мс).

Большинство существующих сетей обучались на миллионах небольших изображений для распознавания большого количества различных классов. Проблема в том, что подобные сети имеют огромное количество коэффициентов (Например, коэффициенты сети VGG [5] занимают около 500мб) и работают относительно медленно.

Задача распознавания выделения намного проще: есть только два класса (выделение и его отсутствие). Распознаваемые данные обладают малым разнообразием - GUI различных программ организуется по схожим признакам. Цель -

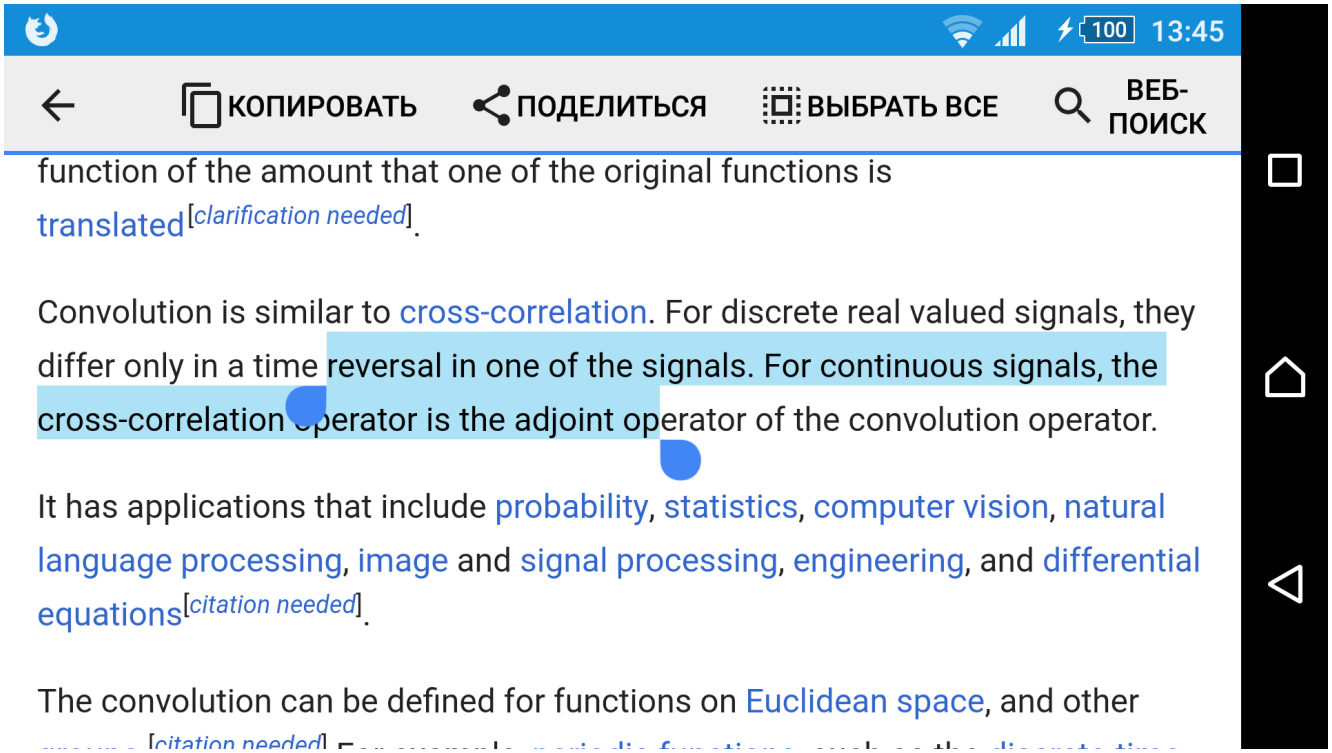

Рис. 1: Пример редактирования области выделения на телефоне

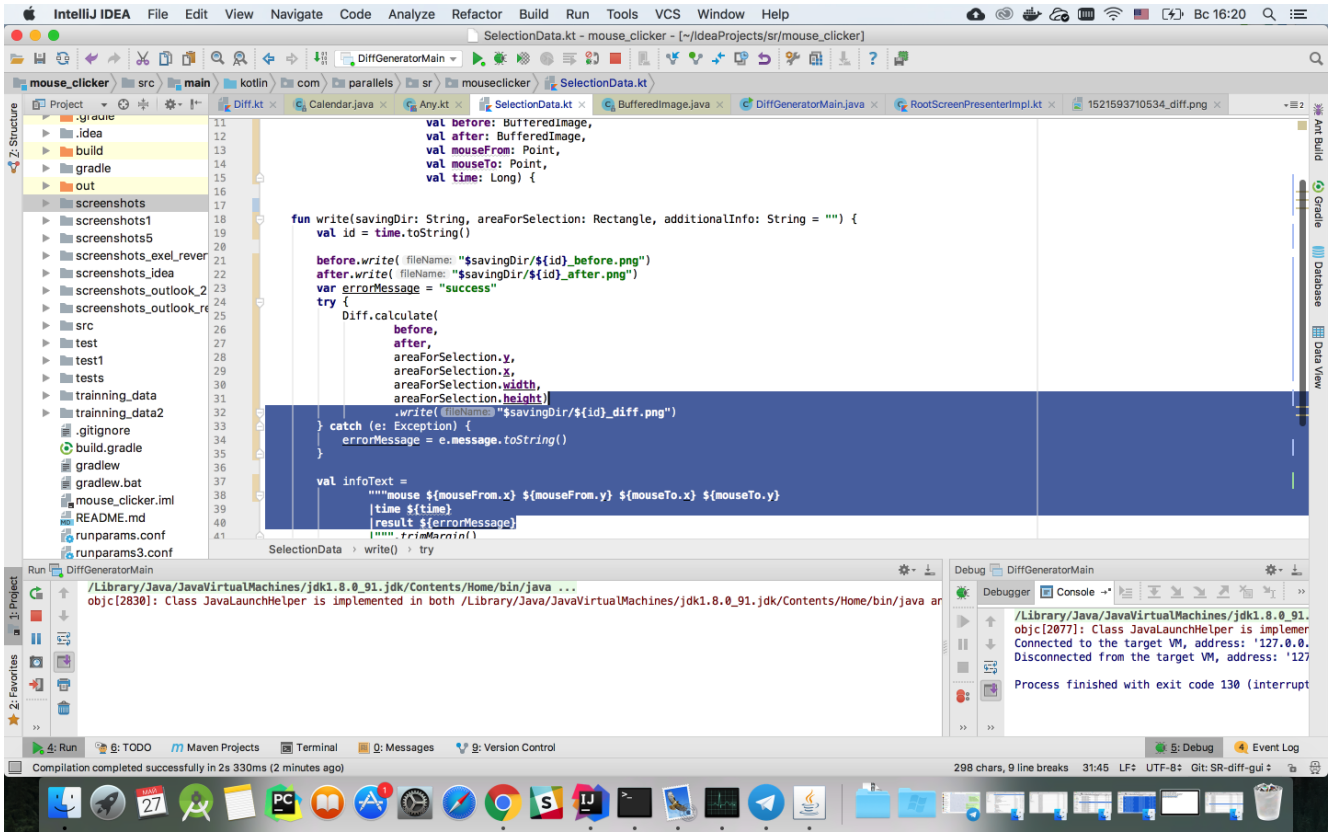

Рис. 2: Пример скриншота, на котором надо найти область выделения

Рис. 3: Эталонная область выделения

построить наиболее простую сеть с максимальной скоростью работы. В идеале, после обучения на большом количестве программ сеть должна усвоить общие прицпы построения пользовательсокого интерфейса и хорошо распознавать даже программы, на которых она не обучалась (подобно человеку).

Кроме того, не требуется инвариантность к повороту - линии интерфейса чётко горизонтальные и вертикальные с точностью до пикселя. Это позволяет использовать решения типа свёрток с ядром 1×7, которые в обычных архитектурах практически не используются.

В отличие от сети, которая даёт предсказание сразу по всей картинке, в задаче семантической сегментации каждый пиксель на выходе является маленьким обучающим примером, что позволяет на несколько порядков снизить количество изображений в обучающей выборке.

Использование более простой архитектуры позволяет не бояться переобучения сети и использовать меньшее количество входных данных.

1.2 Семантическая сегментация

Существуют четыре задачи распознавания изображений (рисунок 4):

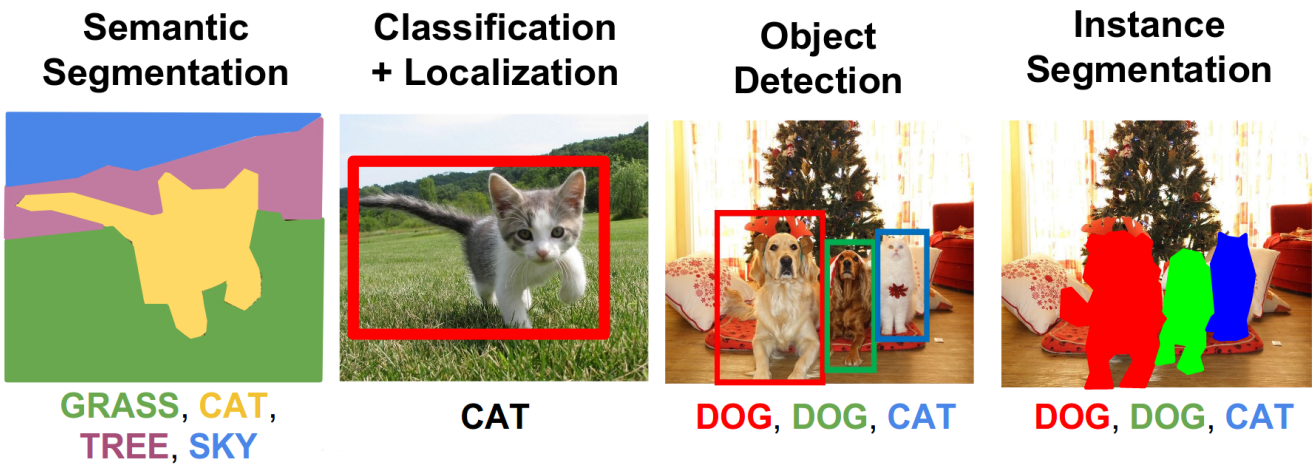

Рис. 4: Задачи распознавания изображений. Источник: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

1. Semantic segmentation: нейронная сеть на выходе для каждого пикселя выдаёт вероятность принадлежности к каждому классу.

2. Object recognition: исходная сеть получает на вход изображение с ровно одним объектом и возвращает координаты содержащего объект прямоугольника и его класс.

3. Object detection: На картинке есть несколько объектов, сеть должна обвести прямоугольником и предсказать класс для каждого объекта. Заслуживают внимания Faster R-CNN и YOLO. Но, к сожалению, они не очень применимы при распознавании областей выделения.

4. Instance segmentation: Самая сложная задача - объектов может быть несколько, причём надо отличать друг от друга разные объекты с совпадающим классом.

Итак, instance segmentation не требуется - даже если область выделения не односвязная, выделение всё равно одно и то же. Object recognition не подходит - область выделения может отсутствовать, вдобавок она может быть сложной формы.

Было бы заманчиво получить описание областей выделения в виде прямоугольников, но, к сожалению, сети типа YOLO [7] не подойдут. В сети YOLO изображение в результате свёрток и max-pooling слоёв уменьшается до маленького размера 7×7 пикселей. Каждый пиксель предсказывает координаты пяти прямоугольников и вероятности для объектов в каждом из них. Проблема в том,

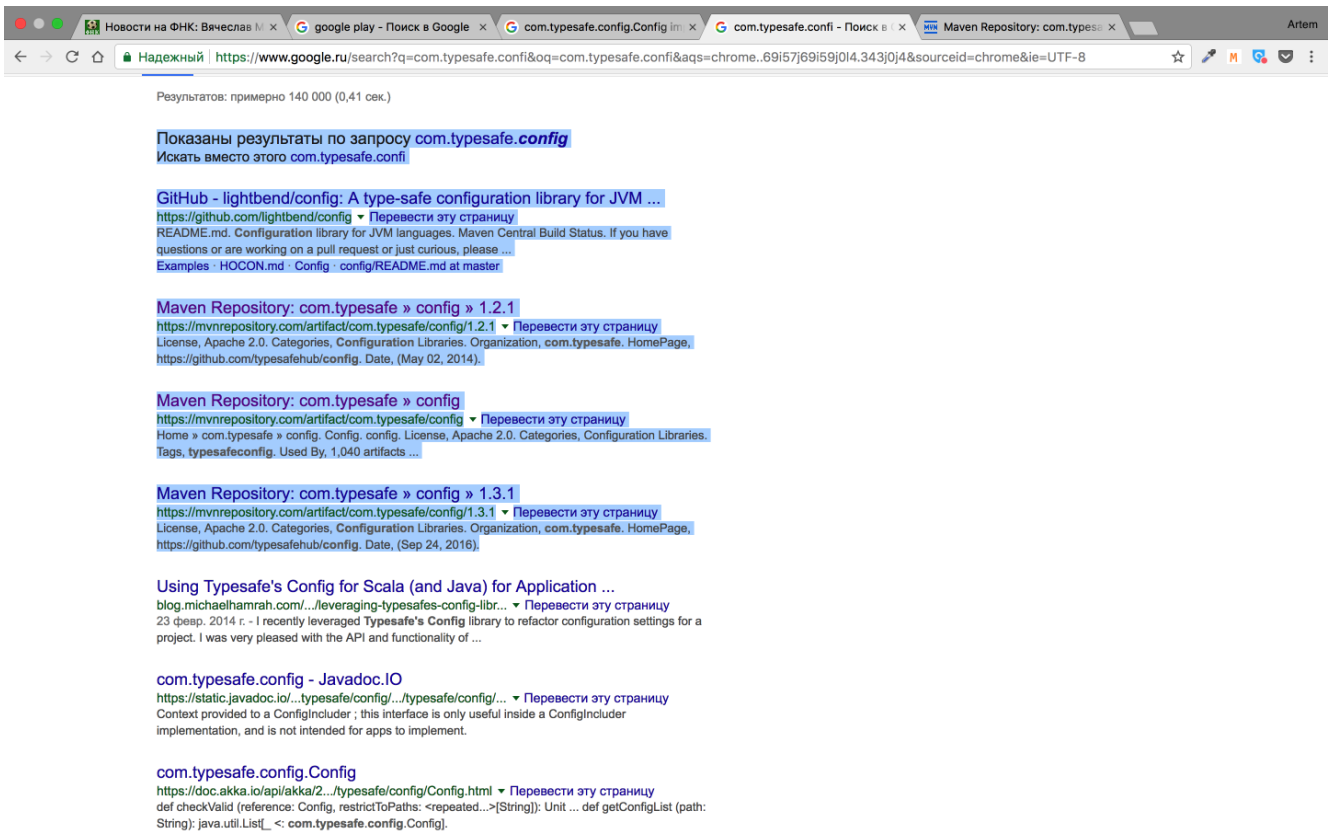

Рис. 5: Пример области выделения сложной формы

что каждый пиксель для своих предсказаний использует небольшую область исходного изображения. Это применимо к изображениям реальных объектов - когда по части объекта можно догадаться о его расположении (например, по изображению верхней части туловища человека легко предположить, как у него расположены ноги и как расположить прямоугольник, описывающий человека).

С областью выделения такое сделать проблематично. Нельзя по фрагменту области предсказать её форму и размер (рисунки 5, 6). Поэтому остаётся самый первый тип - semantic segmentation.

Он простой, но недостатком является тот факт, что надо получить предсказание сети для каждого пикселя изображения. В работе было применено несколько подходов для ускорения работы сети, про которые будет рассказано ниже.

1.3 Критерии качества работы сети

Нейронная сеть на выход подаёт предсказания для каждого квадрата 8×8 пикселей из исходного изображения. Как правило, область выделения занимает крайне небольшую область изоражения, и обучающая выборка является несба-

Рис. 6: Пример области выделения сложной формы

лансированной (область выделения в среднем занимала около 10% площади скриншота).

По этой причине метрика accuracy - доля правильно угаданных пикселей не является информативной. Нейронная сеть, которая на выход всегда выдаёт 0 (области выделения нет), будет давать правильные предсказания на 90% пикселей и иметь accuracy 0.9.

Вместо этого использовалась ф-мера.

$f\ measure = \frac{2 \cdot precision \cdot recall}{precision + recall}$

Где $precision$ - доля правильно пресдсказанных объектов среди общего количества срабатываний, а $recall$ - отношение количества правильно предсказанных объектов к объему количеству объектов, которые надо было найти.

Иными словами, precision зависит от того, как много правильных объектов среди всех найденных, а recall - как много объектов от общего количества мы смогли найти.

Чтобы f-мера была близка к единице, требуются высокие значения и для precision и для recall. Для оценки качества распознавания использовалась именно f-мера.

2 Архитектура сети

Типичная архитектура свёрточной сети выглядит примерно так

- свёрточные слои (16 каналов, 1-3 слоя)
- max-pooling(в 2 раза)
- свёрточные слои (32 канала, 1-3 слоя)
- max-pooling(в 2 раза)
- свёрточные слои (64 канала, 1-3 слоя)
- max-pooling(в 2 раза)

И так пока картинка не станет достаточно маленькой (в случае с YOLO) или даже не превратится в картинку 1×1 с большим количеством каналов (VGG, AlexNet).

Возможны вариации с заменой max-pooling на свёртку со stride.

Основные преобразования (различные варианты свёрток и max-pooling слой) приведены в Приложении А.

Первые слои получают на вход полноразмерное изображение, его считать долго, и поэтому начальные слои имеют малое количество каналов.

После уменьшения картинки в 2 раза её площадь уменьшается в 4 раза, мы увеличиваем количество каналов в 2 раза, из-за чего сложность вычисления для каждого пикселя вырастает в $2 \times 2 = 4$ раза. Таким образом получается, что все слои примерно равны по сложности вычисления. Это довольно логично, если бы время работы разных слоёв составляло бы прогрессию типа 1, 2, 4, то большую часть времени вычислялся бы один слой и возникла бы идея либо сделать его тоньше, либо увеличить количество каналов в других слоях, т.к. они вычислялись бы всё равно быстро.

Уменьшение картинки увеличивает скороть работы сети или позволяет использовать более "толстые" слои.

Из практических соображений было выбрано уменьшение выходной картинки в 8 раз - достаточно знать о наличии об области выделения с точностью до 8 пикселей, да и типичная область выделения будет больше этого размера.

Ещё важно количество занимаемой памяти. Для хранения хотя бы 8 каналов изоражения 1440×900 потребуется как минимум $1440 \times 900 \times 8 \times 4 = 41472000$ байт - более 40 мб памяти. При уменьшении размеров слоя в 2 раза и увеличении

количества каналов в 2 раза (т.е., при неизменной вычислительной сложности) мы получаем экономию памяти в 2 раза. Из-за этого максимум потребления памяти приходится на первые слои. Это довольно актуально для мобильных устройств - на них вычисление начальных слоёв получалось очень медленным. В дальнейшем будет рассказано, как удалось избавитья от этого эффекта практически без потери точности распознавания.

Обычно свёртка делается с размером ядра меньше или равным 3×3. Получается, что начальные слои выделяют какие-то локальные признаки (точки, чёрточки и т.п.), а последние слои обрабатывают уже сильно уменьшенную картинку и распознают какие-то высокоуровневые признаки типа "кнопка, текст и т.п."

Но так входное изображение уменьшается только в 8 раз, хотелось бы, чтобы область исходного изображения, на основании которой принимается решение о том, принадлежит ли пиксель выделению, была как минимум 100×100 пикселей в изначальном иображении. Чем больше эта область, тем больше обучающих примеров нужно, но зато сеть будет принимать во внимание какую-то информацию из довольно далёких частей изображения.

Существуют два подхода:

1. Уменьшить картинку и потом увеличить обратно (всё внутри одной сети)

2. Использовать свёртку с параметром dilation, позволяющую получать информацию от удаленных областей изображения.

Был проверены оба способа.

Первый подход использовался в сети U-net [8].

Кроме того, информация из слоёв до уменьшения подавалась слоям после увеличения до обратного размера, чтобы увеличенная картинка могла быть похожа на первоначальную.

Одни слои (по три свёрточных слоя для каждого уменьшения в два раза) учатся уменьшать изображение, другие (опять по три свёрточных слоя) учатся его увеличивать обратно, и всё это происходит одновременно.

В результате получается сеть с большим количеством слоёв, которая медленно обучается, так как слоям требуется как минимум научиться сжимать информацию и представлять её в большом количестве каналов, а потом уве-

личивать изображение обратно, что не очень быстро работает, так как в сети много слоёв.

Второй подход подошёл намного больше: в статье [9] используюся слои с dilations, равными степеням двойки: 2, 4, 8, 16. Каждый пиксель может получить информацию о каких-то далёких фрагментах изображения, и это не требует большого количества слоёв. Для примера, сеть из статьи содержит всего лишь восемь слоёв, в то время как в U-net их 28.

Кроме того, в качестве шага были испробованы числа Фибоначчи: $(2, 3, 5, 8, 13)$ - такая сеть работает примерно так же, как и со степенями двойки. Трудно утверждать, что что-то одно распознаёт лучше другого.

Итак, примерная начальная архитектура сети получилась следующей:

- convolution(8 channels)
- max-pooling(2)
- convolution(16 channels)
- max-pooling(2)
- convolution(32 channels)
- max-pooling(2)
- convolution(64 channels)
- convolution(64 channels, dilation $= 2$)
- convoltuion(64 channels, dilation $= 4$)
- convoltuion(64 channels, dilation $= 8$)
- convoltuion(64 channels)
- convolution(1 channel)

Т.е., сеть уменьшает изображение в восемь раз и дальше использует dilations вместо уменьшения. На цвет итогового пикселя влияет первоначальная область изображения размером более 200×200.

Дальнейшие эксперименты проводились с этой архитектурой - в неё вносились различные изменения.

В качестве функций активации использовалась leakyRelu с коэффициентом 0.3. Можно использовать либо последний слой с двумя каналами и softmax активацией, либо слой с одним каналом и сигмодиной активацией (с математической точки зрения это одно и то же). На выходе получим вероятность принадлежности к классу в интервале от 0 до 1.

Обычный свёрточный слой делает сразу две важные вещи: обрабатывает пространственную составляющую изображения (анализирует сразу несколько пикселей), а так же поканальную - анализирует сразу все каналы (со всех пикселей) и на основе их значений вычисляет новые. Ниже описана архитектура Xception слоя [10], в которых эти две функции разделены между слоями сети.

Если добавить промежуточные поточечные свёртки, то сеть работает лучше.

- convolution(8 channels)
- max-pooling(2)
- convolution(16 channels)
- max-pooling(2)
- convolution(32 channels)
- max-pooling(2)
- convolution(64 channels)
- pointwise convolution(32 channels)
- convolution(64 channels, dilation = 2)
- pointwise convolution(32 channels)
- convoltuion(64 channels, dilation = 4)
- pointwise convolution(32 channels)
- convoltuion(64 channels, dilation = 8)
- pointwise convolution(32 channels)
- convoltuion(64 channels)
- convolution(1 channel)

Что важно, сеть с промежуточными поточечными слоями лучше распознаёт, содержит меньше коэффициентов и работает быстрее.

Скорее всего, дело в том, что пара слоёв (свёртка и следующая за ней поточечная свёртка) содержат две нелинейных функции активации, что позволяет описывать более сложные функции.

Если в сети слишком много слоёв, возникают нежелательные эффекты типа затухающих или, наоборот, чрезмерно больших градиентов, из-за которых процесс обучения становится неустойчивым или очень долгим. Но, как правило, более глубокие сети работают лучше (теоретически любую функцию можно приблизить перцептоном с двумя слоями, но в этих слоях понадобится слишком большое количество нейронов).

При создании глубоких сетей используются следующие подходы:

1. Создание более коротких путей внутри сети (Fractal net [11], Dense net [12]).

2. Создание общего пути, из которого блоки будут брать входные значения и к нему же прибавлять выходные (resNet, позволяет использовать просто огромное количество слоёв - вплоть до тысячи).

3. Нормализация выходов от промежуточных слоёв (batch normalization).

2.1 Более короткие пути внутри сети

Наличие более коротких путей внутри сети позволяет ей намного быстрее обучаться.

Идея dense net - организовать сеть в блоки, каждый слой получает на вход выходы всех предыдущих слоёв внутри того же блока. В блоке содержится от трёх до шести слоёв. На рассматриваемых данных такая сеть значительно быстрее обучалась, но наилучшее качество распознавания было практически таким же, как и у предыдущей сети. Кроме того, не очень понятно, как масштабировать такую сеть - например, если собрать блок из восьми слоёв по 32 канала, то восьмой слой получит на вход целых $32 \times 7 = 224$ канала, из-за чего будет вычисляться довольно медленно. А при сборке сети из нескольких блоков наикратчайший путь в сети становится не таким уж и коротким.

Проблема масштабирования была решена во фрактальных сетях - но, как уже отмечалось выше, в данной работе используется довольно простая сеть, которая быстро обучается, и использование архитектур типа dense net или фрактальной сети не дало существенного увеличения качества распознавания или уменьшения времени работы. Более удачным оказался подход со сложением слоёв.

2.2 Архитектура типа res net

Residual Networks [13]. Эта архитектура позволяет обучать сети с огромным количеством слоёв - порядка нескольких сотен.

В обычной сети каждый слой учится передавать информацию дальше, идея residual блоков в сети в том, чтобы она изначально уже умела это делать. Есть

некий общий путь (например, картинка размером $height \times width \times channels$). Каждый residual block что-то вычисляет на основе входных данных. Выход блока суммируется с входными данными. Получается, что на вход каждого следующего блока поступает сумма непосредственных выходов всех предыдущих блоков - в этом плане архитектура похожа на dense net.

Внутри блока, как правило, содержится пара свёрток 3×3 и, возможно, поточечные свёртки для понижения или повышения колчества каналов.

Проблем с масштабируемостью нет, блоков может быть произвольное количество. Например, авторы пробовали экстремально глубокий вариант сети из 1202 слоёв, но отмечали, что качество работы сети ухудшается и сравнимо с сетью из 110 слоёв.

Напомню, в общем архитектура выбранной сети выглядит примерно так:

- convolution(8 channels)
- max-pooling(2)
- convolution(16 channels)
- max-pooling(2)
- convolution(32 channels)
- max-pooling(2)
- convolution(64 channels)
- pointwise convolution(32 channels)
- convolution(64 channels, dilation = 2)
- pointwise convolution(32 channels)
- convoltuion(64 channels, dilation = 4)
- pointwise convolution(32 channels)
- convoltuion(64 channels, dilation = 8)
- pointwise convolution(32 channels)
- convoltuion(64 channels)
- pointwise convolution(1 channel)

Есть часть сети после третьего max-pooling - слоя, в которой можно применить данный подход.

Можно сделать так:

- convolution(8 channels)

- max-pooling(2)
- convolution(16 channels)
- max-pooling(2)
- convolution(32 channels)
- max-pooling(2)
- block(dilation = 1)
- block(dilation = 2)
- block(dilation = 4)
- block(dilation = 8)
- pointwise convolution(1 channel)

Есть два способа применения функции активации

1. Применить функцию активации к выходу блока и потом сложить это со входными значениями.

2. Сложить выход блока со входными значеними и только потом применить функцию активации

До сих пор ведутся споры, какой способ лучше. На выбранных данных более удачным оказался второй вариант.

2.3 Batch Normalization

Этот метод был предложен Google в 2015 году [14].

При обучении сразу на большом количестве входных данных (батче) можно найти средние и среднеквадратическое отклонение и отнормировать выходы слоёв, чтобы, например, среднее стало 0, а среднеквадратичное отклонение: 1. Батч-нормализация ускоряет процесс обучения, а так же позволяет использовать больший шаг обучения.

Нормализация реализована как добавление промежуточных слоёв в сеть. В уже обученной сети можно избавиться от них, скорректировав веса остальных слоёв.

К сожалению, из-за того, что сеть работает с очень большими изображениями, из-за применения для обучения видеокарты с 2 гб памяти, размер батча не удавалось сделать больше 4-6 изображений. Для обратного распространения ошибки требуется, чтобы хранились все промежуточные значения сети.

Был опробовал подход вместо вычисления средних значений и отклонений по батчу вычислять скользящее среднее (так как батчи слишком маленькие и хочется усреднять по бОльшему количеству входных данных), но положительный эффект отсутствовал. Возможно, реализация была некорректной.

2.4 Комбинирование свёрточных слоёв

В сетях типа Inception [15] различным образом комбинируются слои со свёртками $1 \times 1, 3 \times 3, 7 \times 1, 1 \times 7$. Особенностью свёрток типа 1×7 является то, что они хорошо распознают пространственную составляющую, но менее устойчивы к повороту, чем свёртки 3×3. Свёртки с ядром 1×1 занимаются обработкой информации, содержащейся в различных каналах одного пикселя.

Стоит отметить, что одномерные свёртки типа 1×5 или 1×7 с вычислительной точки зрения быстрее сделать напрямую, а не как комбинацию двух или трёх свёрток 1×3. В двухмерном случае наоборот: свёртка 5×5 будет вычисляться дольше (по крайней мере теоретически), чем комбинация двух свёрток 3×3.

В рассматриваемом случае входными данными являются скриншоты, в которых большинство линий строго горизонтальные или вертикальные. Свёртки вдоль осей 1×7 и 7×1 показали очень хорошие результаты, так как на моих данных инвериантность к повороту вообще не требуется.

В сети inception поточечные свёртки обрабатывают информацию из разных каналов, а свёртки с ядром 3×3 или, например, 1×7 - пространственные паттерны на изображении.

2.5 Слои Xception

Можно развить идею дальше [10] и сделать свёртку 3×3 поканальной (channel-wise) - т.е., считать его отдельно и независимо для каждого канала. Это позволяет вычислять её максимально быстро.

Таким образом, слои сети выполняют две раздельные функции - поканальные свёртки обрабатывают только пространственную составляющую отдельно для каждого канала, поточечные - между каналами для каждого пикселя.

Что самое интересное - авторы утверждают, что сеть с активацией ReLU между поканальными и поточечными свёртками работает хуже, чем без проме-

жуточной активации. Таким образом, можно выделить экстремальный Xception слой, который состоит из последовательных поканальной и поточечной свёрток. В Tensorflow этот слой называется SeparableConv2D.

Время вычисления поканальной свёртки линейно зависит от размера входных данных, и самой медленной операцией в сети становится поточечная свёртка. Поточечная свёртка работает примерно в девять раз быстрее, чем обычная свёртка 3×3.

Эти слои были опробованы в выбранной сети - действительно, всё работает. К сожалению, прирост скорости в девять раз недостижим - получилось, что для достижения аналогичных результатов распознавания, как у сети с обычными свёртками 1×1 и 3×3, сети с xception слоями требуется больше каналов. Например, если увеличить количество каналов в два раза, время вычисления поточечной свёртки увеличится в четыре раза, и в целом сеть будет работать только в 2 - 2.5 раза быстрее.

В статье [16] отмечается, что такие сети содержат меньше коэффициентов, быстро работают и дают достаточно хорошие результаты. Их рекомендуется использовать на телефонах.

Путь дальнейших улучшений: так как самой медленной операцией становится поточечная свёртка (в mobile net её вычисление занимает 94.86% времени), надо ускорять именно её. Возможный путь - группировка каналов. Например, взять восемь групп по восемь каналов, сделать поточечные свёртки отдельно в каждой группе, а потом - перемешать каналы - чтобы в новых группах было по одному каналу от каждой предыдущей. Теоретически это будет в восемь раз быстрее, чем поточечная свёртка сразу на 64 каналах, но, к сожалению, операции "перемешивания"в tensorflow не реализованы, и реализация с группировкой и перемешиванием каналов вручную будет медленнее, чем реализация с обычным вычислением поточечной свёртки без группировки слоёв.

2.6 Сравнение max-пулинга и свёртки с шагом

В работе [2] было показано, что вместо max-pooling слоёв можно использовать свёртку с шагом 2 без ухудшения результатов. В работе рассматривается сеть, состоящая только из свёрток. Это утверждение было проверено:

Возможные варианты:

1. Первый вариант использовался изначально:

   - convolution(3×3)
   - max-pooling(2)
   - convolution(3×3)

2. Следующий вариант работает быстрее, но немного хуже. В результате получилось, что для достижения качества как в первом варианте необходимо увеличить в два раза количество каналов в первой свёртке. В итоге был выбран именно этот вариант.

   - convolution(3×3, stride $= 2$)
   - convolution(3×3)

3. Следующий вариант работает хорошо, но он вычислительно более сложный

   - convolution(3×3)
   - convolution(3×3, stride $= 2$)
   - convolution(3×3)

2.7 Используемые функции активации

Свёртка - линейная операция. Чтобы комбинация свёрток стала чем-то отличным от одной большой свёртки, между свёртками применяются нелинейные функции активации.

В данной работе использовались следующие функции:

1. ReLU (Rectifier Linear Unit) $relu(x) = max(x, 0)$ Авторы статьи [1] отмечают, что обучение с этой функцией активации происходит в шесть раз быстрее, чем аналогичной сети с использованием $tanh(x)$. В большинстве свёрточных сетей используют именно $relu$ или её вариации. Интересной особенностью relu является тот факт, что применённая несколько раз активация эквивалента её применению только один раз. Это позволяет в сетях типа ResNet использовать активацию прямо на "главном пути к которому прибавляются выходные значения блоков.

2. $Leaky\ relu(x, \alpha) = max(0, x) + min(0, \alpha x)$ При использовании $relu$ в сочетании с большим шагом обучения нейроны сети могут "умирать получать слишком большие (по модулю) коэффициенты и после этого уже не изменяться - так как нейрон будет всегда возвращать 0 на выходе. Для борьбы с этой проблемой была придумана $leaky\ relu$. В данной работе использовалась $leaky\ relu$ с $\alpha = 0.3$

3. CReLU (Concatenated ReLU) Как было отмечено в статье [3], в начальных слоях свёрточных нейронных сетей фильтры склонны образовывать пары: результаты применения "спаренных" фильтров практически противоположны. Возможно, это связано с особенностями relu. Получается, что можно вычислить в N фильтров, а потом добавить к ним ещё N фильтров, которые на самом деле являются просто первоначальными фильтрами, умноженными на минус 1, и только потом применить к $2N$ фильтрам активацию $relu$. Такой подход позволяет либо уменьшить количество реально вычисляемых фильтров в два раза, либо вдвое увеличить количество фильтров на выходе. Была опробована данная функция активации - нельзя утверждать, что в данном случае она приводит к существенному улучшению работы сети, но и хуже от неё не становилось.

4. $softmax(x)_i = \frac{e^{x_i}}{\sum_i e^{x_i}}$ Применяется для "нормализации"выходных значений - сумма значений по всем каналам станет равной единице, причём ближе всех к едининце будет именно максимальное значение. Чем-то похоже на определение вероятности. Используется в качестве последнего слоя.

5. Сигмоида: $\sigma(x) = \frac{1}{1+e^{-x}}$. Является частным случаем $softmax$ для задачи с двумя классами. Вместо свёртки с двумя фильтрами, применения softmax активации и взятием результатов одного слоя можно сразу взять свёртку с одним фильтром (с математической точки зрения - он будет разницей двух фильтров перед функцией softmax) и получить такой же результат.

2.8 Уменьшение входного изображения

После запуска бенчмарка на различных телефонах с ОС Android было выяснено, что первые слои являются самыми "тяжёлыми"по потреблению памяти,

а так же очень сильно замедляют сеть. До конца не понятны причины данного явления, так как по количеству операций умножения или сложения начальные слои сопоставимы с последующими. Возможно, это связано со скоростью доступа к памяти или с особенностями реализации умножения.

Был проведён эксперимент - из сети убраны начальные слои до первого max-pooling слоя включительно и заменены из на average-pooling. Это сделано исходя из предположения, что человек нормально видит текст и область выделения на экране, даже если не различает отдельные пиксели.

Были по-разному уменьшены входные изображения в 2,3,4,8 раз и проведено сравнение качества распознавания сети.

В таблице приведены результаты сравнения сетей с разными входными параметрами:

входные данные	максимальная ф-мера
оригинал	0.97617
av2	0.96224
max2	0.96370
min2, max2, av2	0.97608
min2, max2	0.97391
max2, av2	0.97416
av3	0.94768
max3	0.95453
min3, max3, av3	0.97288
av4	0.95888
max4	0.95234
min4, max4	0.97160
min4, max4, av4	0.96981
av8	0.91882
min8, max8, av8	0.96481

Запись типа min3, max3, av3 означает, что на вход подавалось уменьшенное в три раза изображение, причём для каждого из rgb каналов исходного изображения передавались минимальное, максимальное и среднее значение (в квадрате 3×3 на исходном изображении).

Установленно, что на выбранных данных оптимальным сочетанием является

уменьшение картинки в четыре раза при помощи max и min - pooling слоёв, а средние значения не влияли на качество.

Т.е, вместо RGB картинки $1440 \times 900 \times 3$ в финальной архитектуре подаётся на вход шестиканальную картинку размером $360 \times 250 \times 6$. Первые три канала - минимальные значения для rgb каналов в квадрате 4×4 на исходном изображении, и ещё три канала - максимальные значения. Уменьшение входных данных в восемь раз оказалось слишком радикальным и в дальнейшем работало не очень хорошо.

Это позволило избавиться от нескольких начальных слоёв и значительно ускорить сеть. Возможно, причиной эффекта является специфика данных: минимальные значения - цвет текста, максимальные значения - цвет фона или области выделения на этом фоне.

2.9 Вычисления с фиксированной точкой

В библиотеке tensorflow недавно был добавлен режим вычислений с фиксированной точкой. Его не очень удобно использовать в время обучения сети - за каждый шаг обучения веса сети изменяются на крайне небольшие значения, что будет невозможно при хранении весов с низкой точностью. Впрочем, в tensorflow есть механизм, использующий fake quantization - при вычислениях сети симулируется квантование значений, но изменения весов на малые значения всё ещё возможны. Альтернативным, и более удобным, способом является обучение сети с весами в формате float и последующая конвертация уже обученной сети в граф с весами вершин в формате "одно значение - один байт".

Для каждого слоя (массива весов в нём) при конвертации задаются максимальные и минимальные значения весов. (назовём их min, max) Веса внутри этого интервала линейно распределены от 0 до 255. Например, для веса x будет значение

$$x_{quantized} = round[\frac{255(x - min)}{max - min}]$$

Такой формат может представлять знаковые числа из любого диапазона. Впрочем, перемножение требует несколько операций, так как

$$(a_1x + b_1) \cdot (a_2y + b_2) = b_1b_2 + a_1b_2x + a_2b_1y + a_1a_2xy = c_0 + c_1x + c_2y + c_3xy$$

Коэффициенты a_1, a_2, b_1, b_2 известны заранее и их произведение можно вы-

числить заранее:

$$b_1b_2 + a_1b_2x + a_2b_1y + a_1a_2xy = c_0 + c_1x + c_2y + c_3xy$$

.

Вычисления с фиксированной точкой ориентированы на мобильные утройства:

1. Хранение весов в виде восьмибитных значений уменьшает размер сети.
2. По сравнению с float значениями в четыре раза снижается потребление памяти и количество читаемой-записываемой информации.
3. Свёрточные сети устойчивы к случайному шуму. Квантование весов сети можно рассматривать как добавленный шум. Качество распознавания практически не ухудшается.
4. Tensorflow на телефонах не поддерживает GPU на большинстве существующих моделей (не считая специально реализованных вариантов для устройств с Nvidia Tegra), и перемножения чисел типа float производятся процессором. В идеале, операции с векторами чисел uint8 должны выполняться быстрее. Хотя, как уже говорилось, вместо одного умножения придётся производить по несколько операций.

Для проверки данного подхода были сконвертированы несколько обученных сетей и сравнены скорость работы и точность. Точность практически не изменяется.

На PC квантованные вычисления выполняются медленнее. Это нормально, разработчики библиотеки отмечают, что у них нет цели достичь высокой скорости на PC, тем более обучение обычно происходит на GPU, очень хорошо поддерживающие вычисления для типа float.

К сожалению, резализация квантованных сетей в tensorflow mobile (по состоянию на март 2018 года) работает в два раза медленнее, чем аналогичная сеть с float. Этот режим был только-только добавлен начале 2018 года.

На момент проведения экспериментов реализация в tensorflow lite (ещё одна библиотека, которой в будущем планируют заменить tensorflow mobile) ещё находилась в тестовом режиме и не поддерживала многих слоёв, так что сравнить скорость работы не удалось.

Возможно, к концу 2018 года квантованные сети получат хорошую реализацию и появится смысл их использовать.

Ниже сравнение обычной сети с её квантованной версией на телефоне Samsung galaxy s6 edge, а так же время сравнивается с сетью, которая вместо свёртки и max-pooling слоя на входе имеет сразу average-pooling с масштабом 2:

	лучшее время, мс	среднее, мс.	худшее, мс
Обычная сеть с float весами	940	1072	1216
Та же сеть, но квантованная	2060	2245	2400
average pooling в начале	640	682	750
Та же сеть, но квантованная	960	1070	1260

Как можно видеть, квантованная версия сети работает раза в два медленее. Кроме того, выкидывание первых слоёв из сети позволяет очень сильно ускорить её. Или из-за высокого потребления памяти, или из-за ещё каких-то причин именно слои, работающие с полноразмерным изображением, получались очень медленными.

Аналогичные выводы можно сделать из результатов измерений на телефоне Sony Xperia ZL:

	лучшее время, мс	среднее, мс.	худшее, мс
Обычная сеть с float весами	2066	2200	2750
Та же сеть, но квантованная	5304	5505	5700
average pooling в начале	1250	1300	1475
Та же сеть, но квантованная	2510	2640	2950

2.10 Выбранная архитектура

В результате сравнения архитектур была выбрана следующая: изначально изображение уменьшается в четыре раза, для каждого канала (красный, зелёный, синий) находятся максимальные и минимальные значения (получится шесть каналов), после применяется свёртка 3×3 с 64 фильтрами и шагом в два (т.е., уменьшение изображения ещё в два раза). После в сети, как в ResNet, добавляются блоки.

Блоки выбраны упрощёнными - поточечная свёртка для уменьшения количества каналов до 32, поканальная свёртка и поточечная для увеличения

размерности до исходной. После того, как выход блока добавлен ко входу, применяется функция активации: $LeakyReLU$ с $\alpha = 0.3$

В блоках применяются свёртки dilation, увеличивающимися по степеням двойки. В конце добавлен блок с поточечными свёртками и после - поточечная свёртка для уменьшения количесва каналов до одного и сигмоидой в качестве функции активации:

- convolution: 3×3, 64 фильтра, $stride = 2$
- block: 3 поточечные свёртки, 1×1, 64
- block: 3×3, внутренняя размерность= 32, $dilation = 1$
- block: 3×3, внутренняя размерность= 32, $dilation = 2$
- block: 3×3, внутренняя размерность= 32, $dilation = 4$
- block: 3×3, внутренняя размерность= 32, $dilation = 8$
- block: 3×3, внутренняя размерность= 32, $dilation = 16$
- block: 1×7, внутренняя размерность= 16, $dilation = 5$
- block: 7×1, внутренняя размерность= 16, $dilation = 5$
- block: 3 поточечные свёртки, внутренняя размерность= 64
- convolution: 1×1, 1 фильтр, функция активации - сигмоид.

Устройство блока:

- convolution: 1×1, 32 фильтра (внутренняя размерность блока)
- channel-wise convolution: 3×3, dilation как указано в параметре для блока, функция активации отсутствует.
- convolution: 1×1, 32 фильтра (внутренняя размерность блока)
- channel-wise convolution: 3×3, dilation как указано в параметре для блока, функция активации отсутствует.
- convolution: 1×1, 64 фильтра, функция активации отсутствует

Выходные значения блока складываются с входными и только после этого применяется функция активации. Самыми медленными операциями является вычисление поточечных свёрток - входной и выходной. (из 64 каналов в 32 и из 32 обратно в 64).

Выбранную архитектуру при необходимости можно очень легко расширить или, наоборот, упростить. Если будет необходимо обрабатывать большие паттерны на изображении, можно добавить ещё пару блоков с большим парамет-

ром dilation. Если сеть недостаточно хорошо работает с паттерном с характерным размером, например, восемь, то можно увеличить внутренюю размерность у блока с параметром $dilation = 4$.

В данном случае добавлены слои со свёртками 1×7 и 7×1, позволяющими лучше обрабатывать вертикальные и горизонатльные паттерны на изображении.

Благодаря использованию функции активации relu, которая не изменяет положительные значения, фичи могут "просачиваться" от самых ранних слоёв к поздним, что положительно сказывается на скорости обучения сети.

График зависимости качества распознавания от количества шагов обучения показан на рисунке 7.

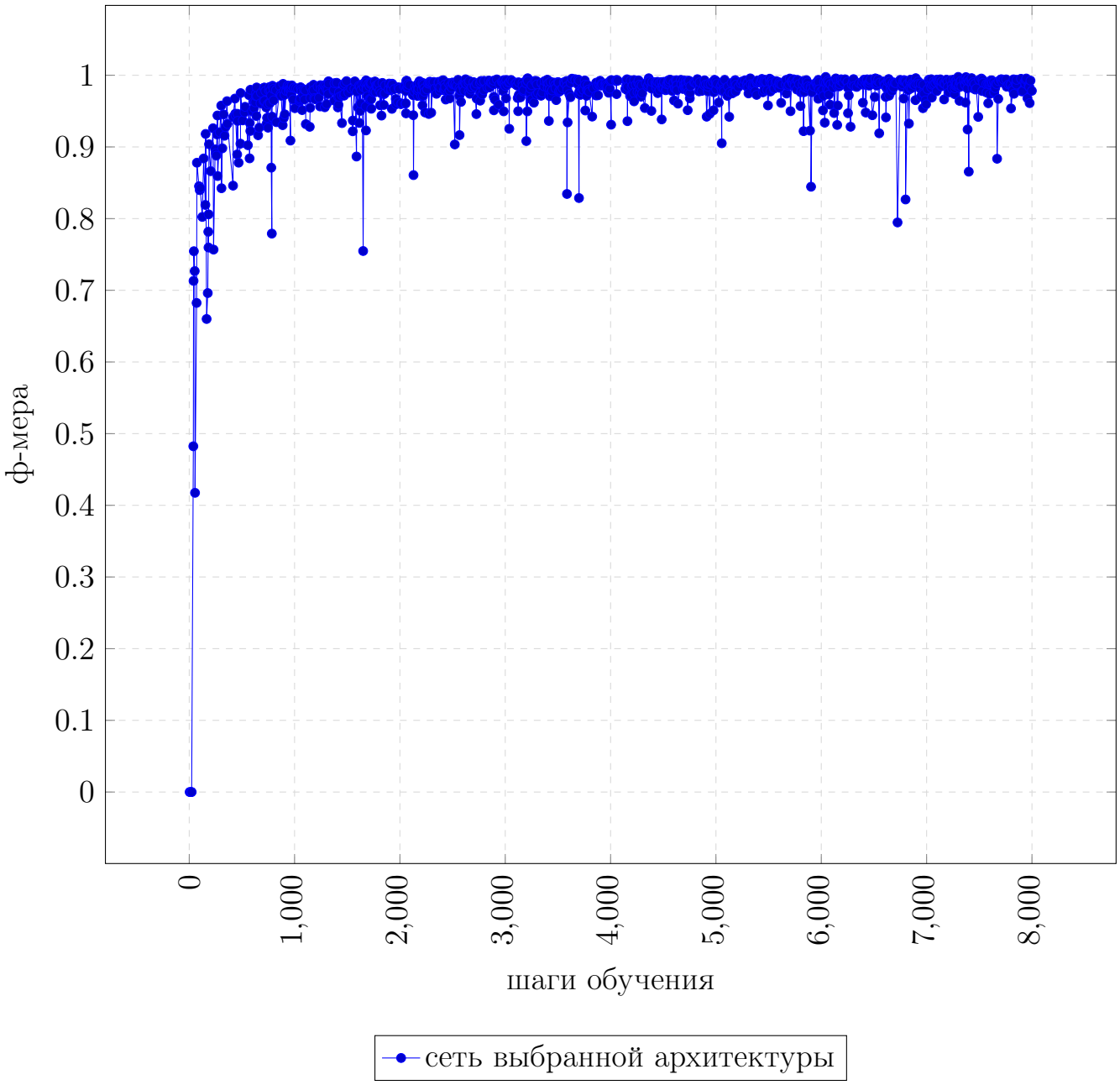

Рис. 7: Обучение сети

Как видно, наибольший прогресс достигается в первые 1000 шагов обучения, и в последующие 8000 сеть медленно обучается до оптимального состояния.

3 Обучение сети

3.1 Используемые изображения

В качестве данных были подготовлены изображения: скриншоты из браузера Google Chrome, Excel, Word, Outlook и среды разработки IntelliJ IDEA.

Использовались примерно по 500 изображений каждого класса в разрешении 1440×900.

Изображения были созданы при помощи программы, которая делала скриншот интерфейса, после чего выделяла случайную область мышкой, снова делала скриншот и по разнице изображений определяла область выделения. Несмотря на кажущуюся простоту, в процессе написания программы пришлось решить целый ряд проблем:

1. Интерфейс может изменяться сам по себе: часы в углу экрана, мигающая или анимированная реклама в браузере.

2. Мигающие курсоры, которые в разных программах могут выглядеть и вести себя совершенно по-разному.

3. При выделении случайной области программа могла воспринять клик и что-то сделать (например, в браузере случались переходы на другой сайт или клик на рекламный баннер)

Часть этих сложностей удалось решить с помощью ограничения областей: куда можно нажимать и в какой области экрана может быть выделение текста (изменения снаружи области игнорировались). Но далеко не все особенности было легко обнаружить и решить с помощью написания кода, поэтому полученные изображения проходили проверку.

Изначально изображения проверялись вручную, впоследствии с помощью нейронной сети были отобраны изображения из датасета, на которых сеть работала плохо, такие образцы были просмотрены вручную и некоторректные варианты были удалены. Автоматически удалять всё, плохо совпадающее с выходом обученной сети было нельзя, так как иногда сеть ошибается на корректных примерах - их ни в коем случае нельзя удалить, и наоборот, стоит добавить больше таких образцов, чтобы сеть научилась распознавать и их тоже.

Если обучать на более-менее однородных данных (на скриншотах из браузера), то даже простая сеть даёт очень хорошие результаты распознавания.

Для распознавания области выделения на разнородных изображениях требуется более "умная" сеть и результаты распознавания значительно хуже. Впрочем, область выделения в Excel- прямоугольник - совершенно не похожа на ту, что в браузере.

Несмотря на то, что датасет содержит не очень много изображений, их количество является достаточным, так как каждый пиксель выходного изоражения является маленьким "обучающим примером".

3.2 Функция ошибки

В качестве функции ошибки использовалась функция

$$logloss(x) = -y\log(x) + (1-y)\log(1-x)$$

(это частный случай перекрёстной энтропии). Так как функция log определена только на числах > 0 и около нуля даёт крайне малые значение, использовалась $eps = 10^{-8}$, и в качестве x:

$$x = clamp(predicted, eps, 1 - eps).$$

Это всё уже реализовано в стандарной функции Log loss в tensorflow.

Обучающая выборка имеет сильный дизбаланс - как правило, область выделения на изображении очень мала. На обучающих данных область выделения составляла в среднем 10% от площади изображения.

Из-за этого при начальных стадиях обучения сеть довольно часто уходила в состояние, когда она возвращала на выход только одно значение: отсутствие выделения. Кроме того, сеть намного лучше распознавала большие области выделения (в несколько строчек) и плохо маленькие: например, одна буква или одна строчка. Если же сделать много-много примеров с очень маленькими областями выделения, то несбалансированность выборки станет ещё сильнее - площадь области выделения может составлять 10^{-4} от общей площади.

Для компенсации этих эффектов использовались веса - на каждом изображении суммарный вес области выделения был 0.3 и остальной части изображения - 0.7. Изменение этих чисел, кроме влияния на процесс обучения сети в

самом начале, так же влияет на вероятность ошибок первого и второго рода.

3.3 Изменение шага обучения

Большой шаг обучения ускоряет обучения на начальных стадиях, но может приводить к неустойчивости при обучении - при наличии больших производных у функции ошибки сеть может внезано из близкого к оптимуму состояния "прыгнуть" в состояние с большой ошибкой. Чтобы избежать этого эффекта, использовались два подхода:

1. Использовался оптимизатор Adam [17], который вычисляет скользящее среднее от градиента (использовался коэффициент 0.9). Таким образом, случайный выброс на одном обучающем образце не очень сильно влияет на градиент. Кроме того, скользящее среднее позволяет избежать проблемы с седловыми точками (когда градиент вдоль одного направления очень большой и быстро меняется, а вдоль другого малый, слабо изменяющийся градиент). Градиент начинает сильно осциллировать, но скользящщее среднее вдоль направления малого, слабого изменяющегося градиента не уменьшается, а скользящее среднее от осциллирующих вдоль какого-то направления значений оказывается близким к нулю.

   С какой-то точки зрения на скользящее среднее можно смотреть как на наличие инерции у градиента, позволяющей почти не реагировать на его резкие изменения.

2. Кроме того, использовался изменяемый шаг обучения:

$$learning\ rate = \frac{0.001}{1 + 0.005 \cdot steps\ count}$$

   Таким образом, большой начальный шаг обучения (0.001) позволял сети быстрее сходиться к минимуму в начале обучения, а малый шаг на поздних стадиях позволял ближе подойти к оптимальному состоянию сети.

На графике 8 показан процесс обучения различных сетей.

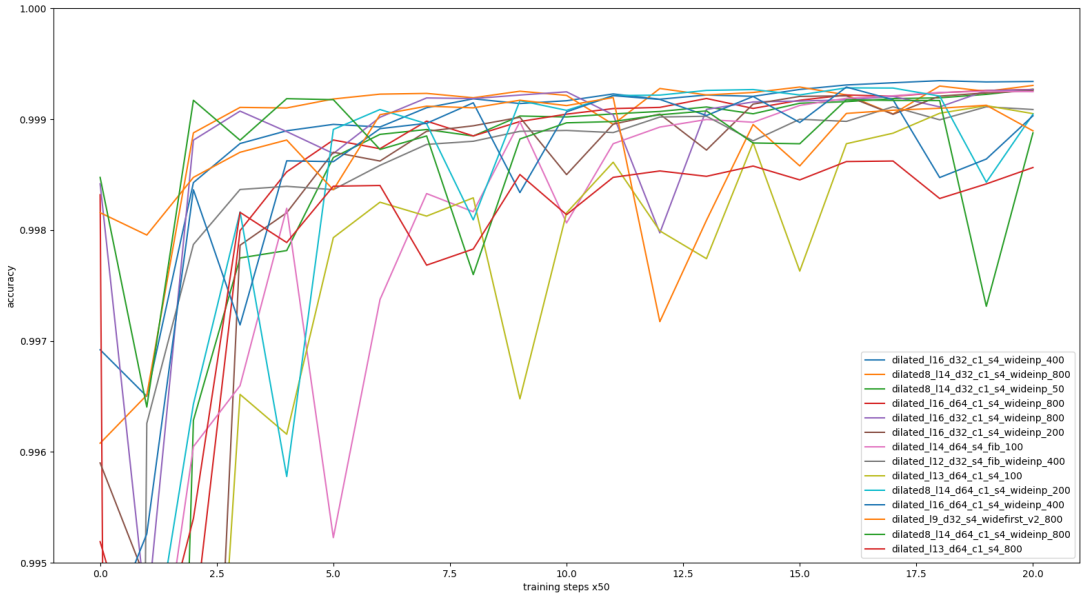

Рис. 8: Сравнение точности разных сетей в зависимости от количества шагов обучения

В данном случае сравнивались нейронные сети начальной архитектуры с различными параметрами.

4 Результаты

4.1 Скорость работы

Получена скорость распознавания на телефоне 150-500 мс на изображениях размером 1440×900. Время зависит от модели телефона, архитектуры сети и от каких-то случайных факторов: разброс времени работы даже при последовательных запусках сети может достигать 10%.

Время обучения нейронной сети на обычном GPU типа Nvidia Geforce 960 составляет порядка 10-30 минут. Последние 20 минут сеть уже довольно медленно улучшается, но для оптимального результата стоит подождать.

На CPU (core i5 третьего поколения, 3 ГГц) обучение получалось в 3-5 раз медленнее, чем на вышеописанном GPU.

Скорость распознавания на GPU огромна (меньше 10 мс), но накладные расходы типа сделать сриншот, преобразовать и положить в буфер для tensrflow, вытащить данные из буфера, нарисовать распознанное изображение, к сожалению, добавляют ещё порядка 200-300 мс (если реализовывать это на языке java, возможно, на C++ будет значительно быстрее).

В таблице ниже представлено время работы финальной архитектуры на телефоне Sony Xperia ZL (как одном из самых медленных из тех, на которых проводилось тестирование):

лучшее время, мс	среднее, мс.	худшее, мс
350	400	560

Также была измерена производительность на телефоне Samsung Galaxy s5:

лучшее время, мс	среднее, мс.	худшее, мс
260	270	350

Данные цифры показывают, что в данной работе удалось добиться приемлемого времени распознавания.

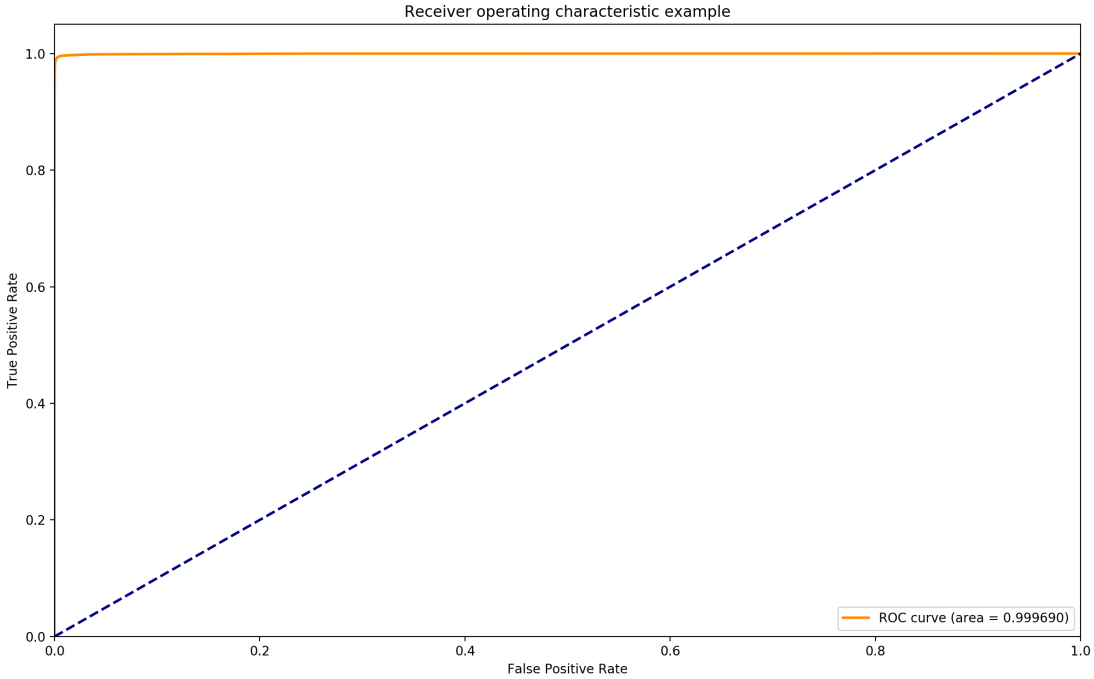

Рис. 9: ROC-кривая для сети, обученной и проверенной на скриншотах Google Chrome

4.2 Качество распознавания

Если обучать сеть и оценивать качество распознавания скриншотах какой-то одной программы, то сеть работает очень хорошо. Примеры приведены на рисунках 9 и 10.

При обучении и оценивании сети на очень разнообразных данных качество работы сети значительно хуже (рисунок 11). Возможно, стоит увеличить размер обучающей выборки и глубину сети. В коркретно данном случае наибольшие трудности распознавания у сети вызывали скриншоты программы Excel, так как в ней интерфейс и внешний вид области выделения кардинально отличаются от остальных программ.

4.3 Применимость на мобильных устройствах

Мобильные устройства пока что не обладают достаточной мощностью для realtime обработки полноразмерных изображений, приходится использовать различные приёмы типа максимального упрощения сети и предварительного уменьшения размера входных изображений, и всё равно время работы значительно

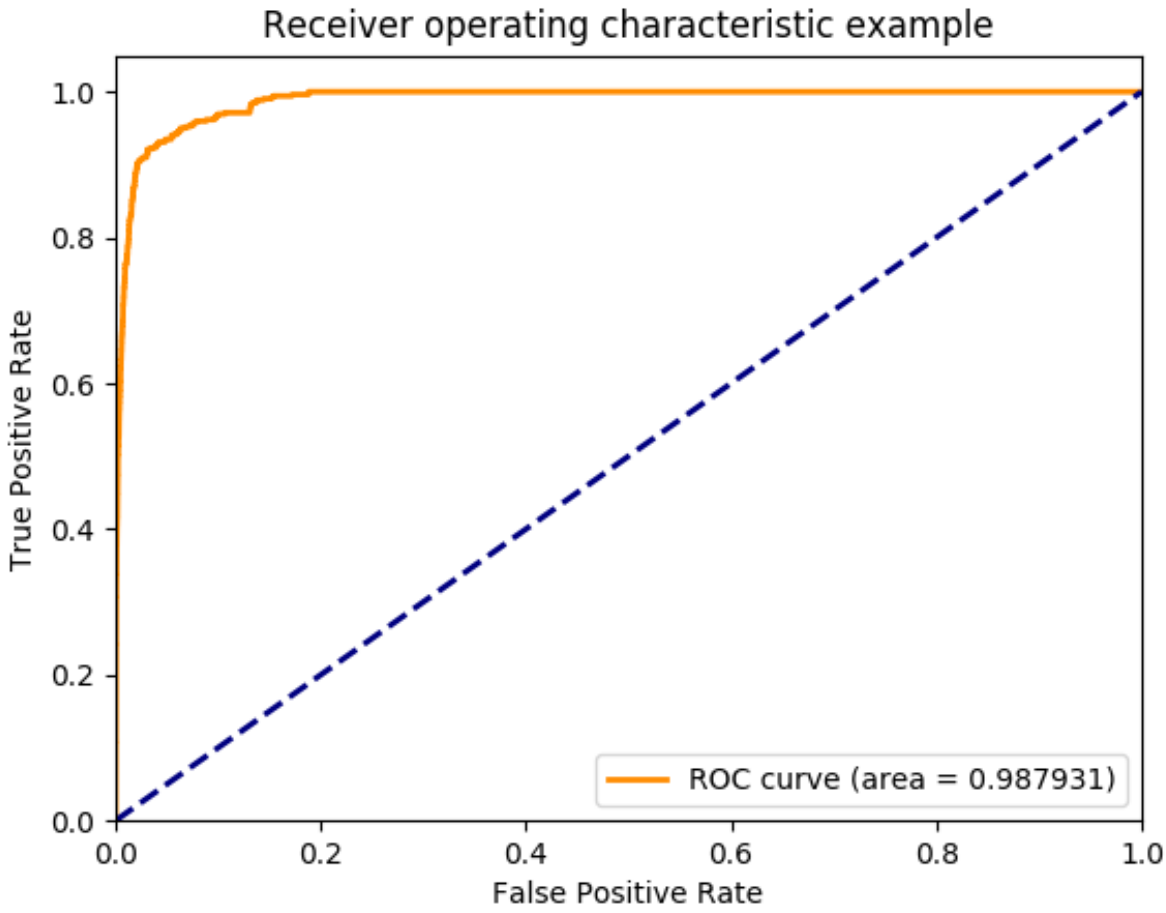

Рис. 10: ROC-кривая для сети, обученной и проверенной на скриншотах из среды разработки IntgelliJ IDEA

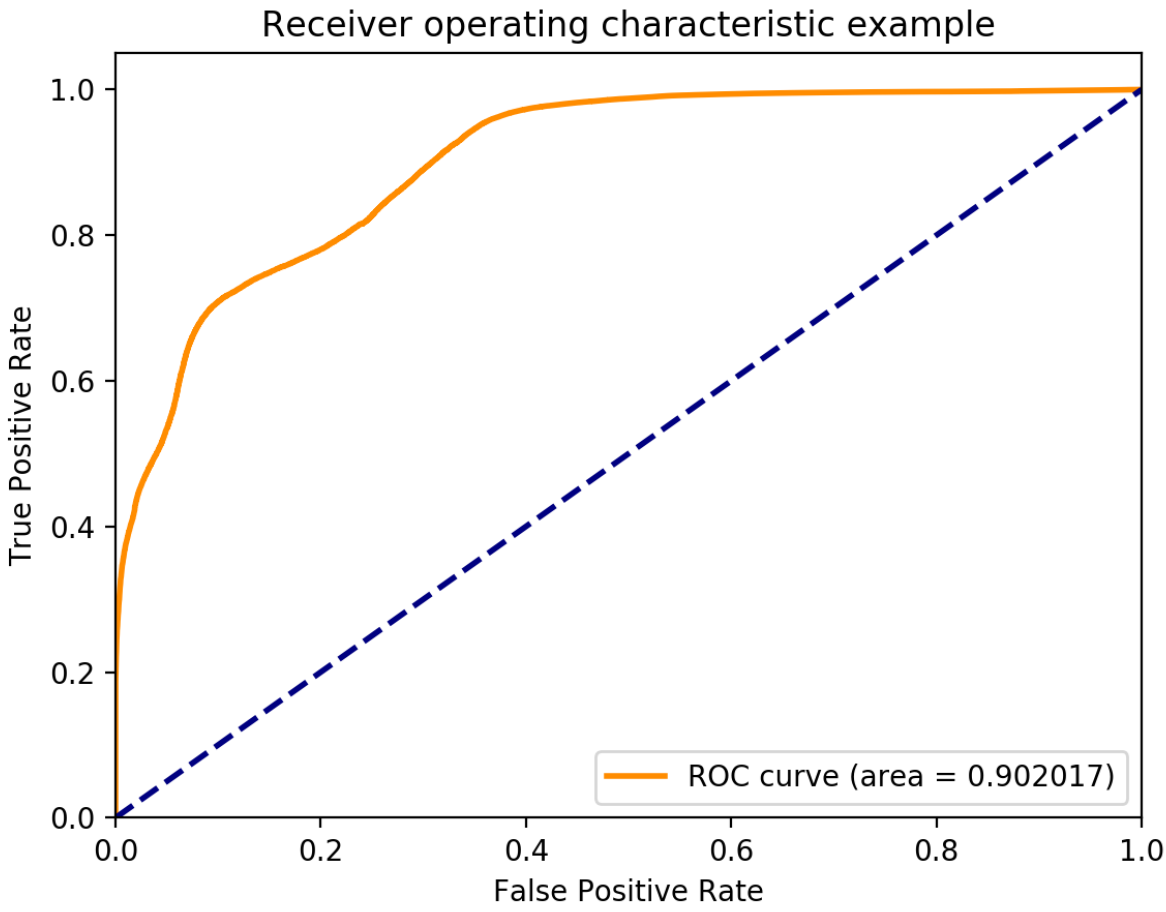

Рис. 11: ROC-кривая для сети, обученной сразу для всех программ

больше времени рисования кадра на экране (30-60 Гц). Но, тем не менее, при обработке более простых данных или при более слабых временных ограничениях нейронные сети можно успешно применять уже сейчас.

Существующие решения (например, tensorflow), позволяют обучать сети на PC и без больших сложностей конвертировать их для работы на мобильных устройствах.

Заключение

Нейронные сети являются довольно мощным средством при решении задач, не имеющих корректного, математически строгого решения. Всё, что нужно для нейронной сети - достаточное количество обучающих данных и функция ошибки. Сейчас собирается огромное количество информации, которую можно обрабатывать. Искать зависимости вручную может быть слишком дорого или бессмысленно.

Производительность современных устройств позволяет использовать нейронные сети даже на телефонах, что в будущем приведёт к широкому применению нейронных сетей в повседневной жизни. Кроме достижения максимального качества распознавания от современных нейронных сетей так же требуется минимальная вычислительная сложность и максимальная скорость работы. Даже на полноразмерных изображениях нейронная сеть может работать достаточно быстро, не говоря уж о задачах с меньшим объёмом данных - например, распознавании речи или работе с текстом.

Существующие решения типа библиотеки Tensorflow (хоть и находятся в состоянии активной разработки) позволяют уже сейчас использовать нейронные сети на ПК и мобильных устройствах с ОС Andoid и iOS.

В современных ОС добавляются встроенные поддержка и API для работы с нейронными сетями (например, в Android начиная с версии 8.1). Возможно, через пару лет в телефонах появятся специализированные вычислительные ядра, оптимизированные для работы нейронных сетей. Например, Google производят TPU (Tensor Processing Unit), которые используются для обучения нейронных сетей и с точки зрения энергопотребления работают на порядок эффективнее, чем GPU. (Первое поколение было анонсировано в 2016 году, 8 мая 2018 было анонсировано третье поколение TPU)

В данной работе были произведены различные эксперименты, изучены и опробованы разные подходы, предложена масштабируемая архитектура, хорошо подходящая для задачи семантической сегментации.

Выводы:

1. Особенности входных данных имеют большое значение. Для однородных данных даже очень простая сеть будет работать хорошо (например, 6-10 слоёв по 32 или даже 16 каналов).

2. Свёртка с использованием dilations позволяет при небольшом количестве слоёв у сети распознавать какие-то большие паттерны на изображении. Это значительно эффективнее, чем уменьшать изображение и потом увеличивать его обратно. Можно увеличивать dilation степенями двойки (1, 2, 4, 8), можно числами Фибоначчи (1, 2, 3, 5, 8) - качество распознавания будет примерно одинаковым. При выборе степеней двойки на выходных данных сети иногда можно заметить квадратные паттерны, при использовании чисел Фибоначчи визуально подобных паттернов не заметно.

3. Свёртки можно группировать - это почти не влияет на качество распознавания. Xception является экстремальным случаем, когда размер группы равен единице. Поканальная свёртка вычисляется очень быстро, и самой медленной операцией становится поточечная свёртка. Xception позволяют достичь большей производительности. Так как поканальная свёртка в Xcpetion слое выполняется очень быстро, можно практически без влияния на производительность использовать фильтры большего размера - например, 5×5.

4. Возможно, в будущем вместо обычной поточечной свёртки появится реализация поточечной свёртки с группировкой по слоям и последующим их перемешиванием. Это особенно актуально для сетей типа YOLO, у которых последние слои могут содержать 256 каналов и более.

5. Свёртка с шагом показывает себя не хуже, чем свёртка + max-pooling, для аналогичного качества работы свёртки с шагом лучше взять в два раза больше каналов, чем в max-pooling слое. Причём такая реализация будет быстрее работать, так как свёртка вычисляется только для одной чертвёртой части пикселей.

6. Аналогично, поканальные свёртки требуют больше каналов для работы, чем обычные свёртки, но зато они значительно быстее работают.

7. Если не требуется инвариантность к повороту, можно использовать свёртки 1×7 и 7×1 - они позволяют за один слой захватить сразу большую область изображения (особенно в сочетании с dilations). Кроме того, хорошо себя показывает комбинация свёрток 1×7 и 7×1 без функции

активации между ними - вычисляется быстро и при этом захватывает область размером 7×7.

8. Подход с добавлением более коротких путей внутри сети позволяет существенно ускорить процесс обучения и использовать значительно более глубокие сети. Делать конкатенацию слоёв не очень удобно, намного лучше масштабируется подход с добавлением входных данных блока к выходным. Это позволяет использовать практически любое количество блоков, причём необязательно одинаковых.

9. Разработанная архитектура сети позволяет распознавать элементы изображения на телефоне с ОС Android за время 200-500 мс в зависимости от модели телефона.

Список литературы

[1] *Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton* ImageNet Classification with Deep Convolutional Neural Networks, стр. 3 *(NIPS 2012)* URL https://papers.nips.cc/book/advances-in-neural-information-processing-systems-25-2012

[2] *Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller* Striving for Simplicity: The All Convolutional Net, *arXiv*:1412.6806v3

[3] *Wenling Shang, Kihyuk Sohn, Diogo Almeida, Honglak Lee* Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units, *arXiv*:1603.05201v2

[4] URL http://image-net.org

[5] *Karen Simonyan, Andrew Zisserman* Very Deep Convolutional Networks for Large-Scale Image Recognition *arXiv*:1409.1556v6

[6] URL https://www.tensorflow.org/

[7] *Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi* You Only Look Once: Unified, Real-Time Object Detection, textitarXiv:1506.02640v5

[8] *Olaf Ronneberger, Philipp Fischer, Thomas Brox* U-Net: Convolutional Networks for Biomedical Image Segmentation, *arXiv*:1505.04597v1

[9] *Fisher Yu, Vladlen Koltun* Multi-Scale Context Aggregation by Dilated Convolutions, *arXiv*:1511.07122v3

[10] *François Chollet* Xception: Deep Learning with Depthwise Separable Convolutions *arXiv*:1610.02357v3

[11] *Gustav Larsson, Michael Maire, Gregory Shakhnarovich* FractalNet: Ultra-Deep Neural Networks without Residuals *arXiv*:1605.07648v4

[12] *Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger* Densely Connected Convolutional Networks *arXiv*:1608.06993v5

[13] *Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun* Deep Residual Learning for Image Recognition, *arXiv*:1512.03385v1

[14] *Sergey Ioffe, Christian Szegedy* Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift *arXiv*:1502.03167v3

[15] *Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi* Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, *arXiv*:1602.07261v2

[16] *Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam* MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, *arXiv*:1704.04861v1

[17] *Diederik P. Kingma, Jimmy Ba* Adam: A Method for Stochastic Optimization *arXiv*:1412.6980v9

[18] *Min Lin, Qiang Chen, Shuicheng Yan* Network In Network *arXiv*:1312.4400v3

Приложение А. Слои свёрточной сети

Слой - тензор (массив чисел) с размерами $(height, width, channels\ count)$. Например, в чёрно-белой картинке один канал (яркость), а в цветной - три (красный, зелёный, синий)

Кроме того, в используемой библиотеке tensorflow добавлена ещё одна размерность - количество образцов в одном батче. В дальнейшем она почти не будет упоминаться, так как не влияет на рассуждения, будем считать, что работа производится только с одной картинкой.

Max-pooling

Используется для уменьшения изображения в несколько раз. Отдельно для каждого канала берётся максмальное значение. Параметры:

1. pooling size: размер области, в которой производится уменьшение (например, фрагмент изображния размером 2×2 или 3×3).

2. stride: шаг между выборками. Например, при шаге 2×2 исходная картинка будет уменьшена в 2 раза.

Pooling size и stride не обязательно должны совпадать - например, возможно использование перекрывающихся областей 3×3 при уменьшении изображения в два раза.

Существует вариация: average-pooling - нахождение среднего значения.

Сложность вычисления: $O(height \cdot width \cdot channels_count)$.

Пример преобразования показан на рисунке 12.

Сверточные слои

В общем случае - свёртка исходного изображения с фильтром конечного размера. При распознавании изображений преобразование Фурье не используется, размер фильтра делается по-возможности небольшим - например: 3×3, 3×1 или 1×3 (рисунок 13).

$$out[y, x, c_{out}] = bias[c_{out}] + \sum_{i,j,c_{in}} in[y + i, x + j, cin] \cdot filter[i, j, c_{in}, c_{out}]$$

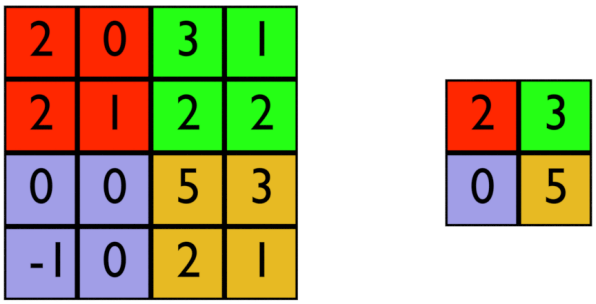

Рис. 12: Пример max-pooling преобразования с масштабом 2

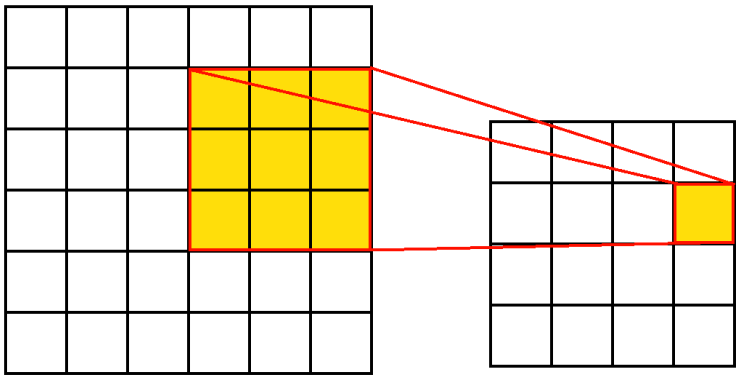

Рис. 13: Пример свёртки с фильтром 3×3

i, j - размеры фильтра, c_{in} - номер канала во входном изображении, c_{out} - в выходном.

Свёрточный слой на выходе может давать тензор с количеством каналов, отличающимся от количества каналов на входе.

Вычислительная сложность вычисления свёртки: $O(filter_width \cdot filter_height \cdot width \cdot height \cdot channels_in \cdot channels_out)$, добавления сдвига: $O(width \cdot height \cdot channels_out)$

Два фильтра 3×3 один за другим посчитать быстрее, чем один фильтр 5×5. Собственно, по этой причине сети обычно содержат свёртки размером не больше 3.

Выходная картинка будет с теми же высотой и шириной размера (если дополнять крайние значения нулями) либо меньшего на (размер фильтра минус единица), если не дополнять.

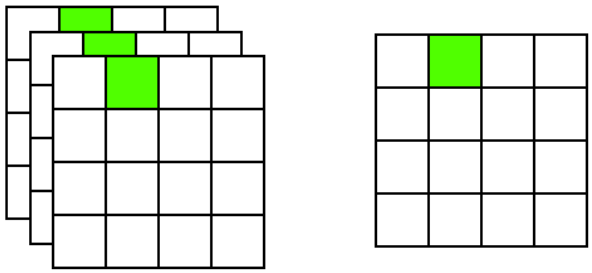

Рис. 14: Пример поточечной свёртки для входного изображения из трёх каналов

Поточечная свёртка (pointwise)

Свёртка, в которой фильтр "перемешивает"значения между каналами, но не заглядывает в соседние пиксели:

$$out[y, x, c_{out}] = bias[c_{out}] + \sum_{i,j,c_{in}} in[y, x, c_{in}] \cdot filter[c_{in}, c_{out}].$$

Вычисляется быстрее обычной свёртки, (примерно в девять раз быстрее свёртки 3×3) может использоваться для увеличения или уменьшения количества каналов. Для такой свёртки $filter_width = filter_height = 1$.

Поканальная свёртка (depthwise)

Каждый канал считается независимо от других:

$$out[y, x, c] = bias[c] + \sum_{i,j} in[y + i, x + j, c] \cdot filter[i, j, c].$$

Такая свёртка вычисляется значительно быстрее обычной свёртки - сложность $O(width \cdot height \cdot filter_width \cdot filter_height \cdot channels_count)$.

В некоторых сетях [10] вместо обычных свёрток используются пары слоёв - поканальная и поточечная свёртка, так как они вычисляются значительно быстрее и дают схожее качество.

Существует вариация, когда от каждого входного канала на выход вычисляется n разных каналов (обычно два или четыре).

Свёртка с разреженным фильтром (dilations)

Свёртки 3×3 используют информацию только от близлежащих пикселей. В тех случаях, когда уменьшение изображения по каким-то причинам не под-

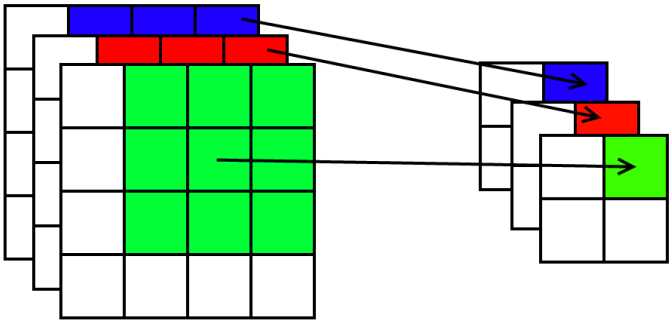

Рис. 15: Пример поканальной свёртки для входного изображения из трёх каналов: для каждого канала независимо от значений в других каналах вычисляется свёртка 3×3

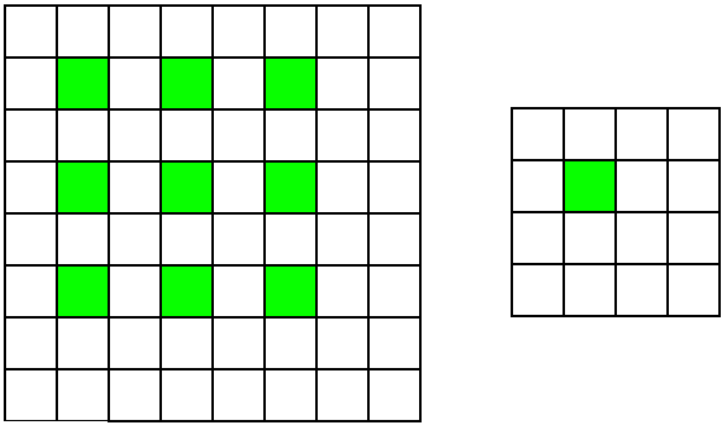

Рис. 16: Разреженная свёртка с $dilation = 2$ и ядром 3×3

ходит, и надо учитывать что-то далеко расположенное, можно использовать модификацию свёртки, которая идёт с шагом (обычно делают слои один за другим с dilations = 1, 2, 4, 8, 16) - в итоге уже после пяти слоёв сеть может учесть наличие каких-то признаков на большом расстоянии:

$$out[y, x, c_{out}] = bias[c_{out}] + \sum_{i,j,c_{in}} in[y + dilation_1 \cdot i, x + dilation_2 \cdot j, cin] \cdot filter[i, j, c_{in}, c_{out}].$$

Подробнее можно прочитать в [9].

Свёртка с шагом (stride)

По аналогии с max-pooling слоём можно вычислять свёртку не для каждого пикселя, а с каким-то шагом. Например, для шага $(2, 2)$ выходное изображение получится в два раза меньше по высоте и ширине. В статье [2] показано, что

сеть, уменьшающая изображение при помощи свёртки с шагом, работает не хуже сети, использующей для уменьшения max-pooling слои.

Композиция свёрток

Иногда фильтр можно предствавить в виде композиции двух более простых. Например, свёртку с фильтром Гаусса $f(x, y) = ae^{-\frac{(x^2+y^2)}{2c^2}}$ можно разложить в два фильтра - по вертикали и горизонтали. Применимо к свёрточным нейронным сетям: вместо свёртки 3×3 можно сделать пару свёрток: 3×1 и 1×3. Так можно разложить далеко не все фильтры. Эффект от разложения свёртки становится более очевидным, если увеличить фильтры вдоль каждой из осей: комбинация свёрток 5×1 и 1×5 вычисляется почти так же быстро, как и обычная свёртка 3×3.

Подобные свёртки используются в архитектуре Inception [15]