УДК 004.855
DOI: 10.25559/SITITO.14.201804.903-910

ВЕРТИКАЛЬНОЕ РАСПРЕДЕЛЕНИЕ НЕЙРОННОЙ СЕТИ МЕЖДУ МОБИЛЬНЫМ УСТРОЙСТВОМ И СЕРВИСАМИ ОБЛАЧНОЙ ИНФРАСТРУКТУРЫ

Ю.А Ушаков, П.Н. Полежаев, А.Е. Шухман, М.В. Ушакова
Оренбургский государственный университет, г. Оренбург, Россия

DISTRIBUTION OF THE NEURAL NETWORK BETWEEN MOBILE DEVICE AND CLOUD INFRASTRUCTURE SERVICES

Yury A. Ushakov, Petr N. Polezhaev, Alexandr E. Shukhman, Margarita V. Ushakova
Orenburg State University, Orenburg, Russia

Ключевые слова

Глубокие нейронные сети; облачные вычисления; контейнеры; виртуальные машины; параллельные вычисления.

Аннотация

Нейронные сети в некоторых областях становятся безальтернативным способом решения задач. Распознавание изображений, звуков, классификация – эти задачи требуют серьезной процессорной мощности и памяти для обучения и для функционирования сети. Современные мобильные устройства имеют довольно неплохие характеристики для первичных слоев глубоких нейросетей, но для полноценной работы не хватает ресурсов. Поскольку обучение нейросетей для мобильных устройств происходит отдельно на внешних ресурсах, был разработан метод распределённой работы нейросети с вертикальным распределением по наборам слоев с синхронизацией данных обучения. Для этого модель разделяется после сохранения ее состояния, все слои на мобильном устройстве конвертируются в формат для мобильного фреймворка и синхронизируются с устройством после обучения на распределенной платформе. Отдельно формируются массивы, связанные с переменными и коэффициентами, что позволяет существенно уменьшить размер файла данных нейросети, загружаемого на устройство. Предложен алгоритм автоматического выбора места разделения нейросети на основе количества передаваемых между слоями данных и нагрузки на ресурсы мобильного устройства. Подход позволяет в некоторых случаях использовать полноразмерные глубокие нейросети совместно с мобильным устройством. Как показало исследование производительности, не перегружая канал связи и ресурсы устройства возможно получить приемлемый отклик даже при нестабильном канале связи.

Об авторах:

Ушаков Юрий Александрович, кандидат технических наук, доцент, кафедра геометрии и компьютерных наук, Оренбургский государственный университет (460000, Россия, г. Оренбург, пр. Победы, д. 13), ORCID: http://orcid.org/0000-0002-0474-8919, unpk@mail.ru

Полежаев Петр Николаевич, старший преподаватель, кафедра компьютерной безопасности и математического обеспечения информационных систем, Оренбургский государственный университет (460000, Россия, г. Оренбург, пр. Победы, д. 13), ORCID: http://orcid.org/0000-0001-7747-646X, newblackpit@mail.ru

Шухман Александр Евгеньевич, кандидат педагогических наук, доцент, заведующий кафедрой геометрии и компьютерных наук, Оренбургский государственный университет (460000, Россия, г. Оренбург, пр. Победы, д. 13), ORCID: http://orcid.org/0000-0002-4303-2550, shukhman@gmail.com

Ушакова Маргарита Викторовна, старший преподаватель, кафедра геометрии и компьютерных наук, Оренбургский государственный университет, (460000, Россия, г. Оренбург, пр. Победы, д. 13), ORCID: http://orcid.org/0000-0003-4462-9946, m.v.ushakova@mail.ru

Keywords

Deep neural networks; cloud computing; containers; virtual machines; parallel computing.

Abstract

Neural networks become the only way to solve problems in some areas. Such tasks as recognition of images, sounds, classification require serious processor power and memory for training and functioning of the network. Modern mobile devices have quite good characteristics for primary layers of deep neural networks, but there are not enough resources for whole network. Since neural networks for mobile devices are trained separately on external resources, a method of distributed work of a neural network with vertical distribution over sets of layers with synchronization of training data was developed. The model is divided after saving its state, all layers on the mobile device are converted to the format for the mobile framework and synchronized with the device after training on a distributed platform. Variables and coefficients are formed separately, which allows to significantly reduce the size of the neural network data file uploaded to the device. An algorithm for automatic selection of a neural network separation point was proposed. It based on the data amount transferred between the layers and the load on the mobile device resources. The approach allows to use full-size deep neural networks with a mobile device. Performance experiment showed possibility of obtains an acceptable response even with an unstable communication channel without overloading communication channels and device resources.

Введение

Работа с визуальной информацией в современной научной практике и реализованных продуктах сводятся к задачам глубокого обучения и компьютерного зрения. Нейросетевая обработка изображений в практике используется для следующих задач: идентификация объекта, семантическая сегментация, распознавание лиц, распознавание частей тела человека, семантическое определение границ, выделение объектов внимания на изображении и выделение нормалей к поверхности (для определения направлений). Чаще всего нейросетевые технологии применяются к распознаванию и классификации объектов на изображении и видеоряде, причем для видеоряда характерно распознавание движений.

Большинство современных систем, основанных на нейросетевых технологиях и предназначенных для распознавания используют сверточные нейронные сети в связке с различными эвристическими алгоритмами или другими типами сетей, например, для анализа видеоряда дополнительно используются реккурентные сети.

Основным недостатком сверточных нейронных сетей является их размер и потребляемые ресурсы как при обучении, так и при работе. Например, распространенная для обучения и тестирования сеть VGGNet состоит из свёрточных слоев, которые выполняют свёртывание 3×3, и слоя пулинга, выполняющих максимальную подвыборку 2×2 имеет в сумме 138 миллионов весов (более 512 Мб памяти), на каждое изображение используется около 93 Мб памяти [1]. Использование фильтров большего размера и увеличение количества слоев еще больше увеличивает требования к памяти. Сверточные сети имеют большое количество разных слоев, которые имеют различную связность. В ноябре 2017 года вышла версия Google Tensorflow Lite для мобильных устройств и гаджетов, в связи с чем возник дополнительный спрос на реализацию нейросетевых вычислений под мобильные и переносные устройства, такие как смартфоны, планшеты, умные очки, небольшие ноутбуки, носимая электроника. Основной проблемой реализации вычислений и обучения на переносных устройствах является большой объем необходимых ресурсов. Существует несколько путей решения этой проблемы. Во-первых, в работе [2] показаны методы оптимизации самой модели, в которой сверточные слои и фильтры выстроены таким образом, чтобы минимизировать объем потребляемой памяти, кроме того используется эвристическая реккуретная связь между сверточными слоями. Второй подход - разнесение слоев на различные устройства, вынос основных вычислений в облако, ведутся давно и некоторые успехи в этом достигнуты для конкретных сетей, например в [3] используется механизм кеширования результатов на основе сохраненных вычислений на стороне мобильного устройства. Это позволяет при разделении слоев не передавать каждый раз все параметры, а делать ссылку на сохраненные части вычислений. Однако при использовании различных изображений такой подход не применим, как и при непрерывном обучении и при обучении с подкреплением. Одной из первых по-настоящему распределенных сетей является GoogleNet, которая поддерживает симметричное разделение внутри слоя, однако слои все также имеют требование к непрерывной связности. Изучение вопроса разделения нейронных сетей на части показано в [4]. В этой работе показаны методы симметричного деления, деления полносвязных топологий, методы объединения результатов, но разделение слоев горизонтально не рассматривается. Это происходит потому, что практически все современные фреймворки, даже основанные на Hadoop, MXNET или кластере TensorFlow, в основном используют вертикальное симметричное разделение слоев [4]. Это дает возможность контролировать на каждом узле всю цепочку вычислений и ускоряет их, но делает затруднительным вынос слоя на другое устройство [5].

Архитектура RedEye [6] сохраняет потребление энергии датчика изображения посредством выполнения слоев свёрточной нейронной сети на аналоговом уровне до квантования. Фреймворк DeepMon [7] предлагает набор методов оптимизации для эффективной разгрузки свёрточных нейронных сетей на мобильных графических процессорах. Миниатюрная камера DeepEye [8] позволяет выполнять мощный анализ изображений практически в режиме реального времени.

Облачная разгрузка глубокого обучения.

Работа глубоких нейронных сетей, особенно моделей глу-

боких сверточных нейронных сетей для задач мобильного компьютерного зрения, часто требует большого объема вычислительных ресурсов. Общеизвестно, что коммерческие смартфоны не могут поддерживать такие высокие вычислительные нагрузки с разумной достаточной степенью латентности и потреблением энергии. Следовательно, традиционный способ использования алгоритмов глубокого обучения – это разгрузка мобильных устройств посредством выполнения вычислений в облаках. Многие компании, такие как Google и Apple, уже создали мощные Web-сервисы для поддержки подобных задач разгрузки в виде приложений [9] и [10].

Не отстают от гигантов индустрии и разработки научных коллективов: сервисная инфраструктура DjiNN [11], специализирующаяся на обработке крупномасштабных задач глубокого нейронного обучения, выгруженных из удалённых мобильных приложений. Авторы работы [12] предлагают глубокую модель сохранения конфиденциальности с использованием BGV-схемы (Бракерски-Гентри-Вайкунтанатан) для шифрования персональных данных и использования облачных серверов для выполнения алгоритма обратного распространения высокого порядка для зашифрованных данных. Планировщик Neurosurgeon [13] может автоматически разбивать вычисления глубоких нейронных сетей между мобильными устройствами и центрами обработки данных при детализации слоёв нейронной сети. Подобная стратегия разделения основана на учете латентности и энергопотребления мобильных устройств и может адаптироваться к различным спецификациям оборудования. Кроме того, существует много работ, нацеленных на более общие идеи разгрузки: рефакторинг java-кода [14], миграция и выполнение приложений javascript [15], автоматическая модульная разбивка java-приложений [16].

В настоящий момент самой популярной считается библиотека TensorFlow, хотя она медленнее, чем Torch и Theano и не имеет такого большого набора моделей, как Caffe. В отличие от любой другой архитектуры, TensorFlow имеет возможность делать частичные вычисления подграфа, то есть получение выборки от общей нейронной сети, а затем ее обучение отдельно от остальной части. Это так называемое Model Parallelization, которое используется для распределенного обучения.

Таким образом, несмотря на преимущества прочих библиотек, для исследований в рамках данной работы был выбран фреймворк TensorFlow/TensorFlow-lite, как надежный и развивающийся проект, способный работать с разными типами нейронных сетей и поддерживающий параллельные вычисления с помощью графических процессоров, а также Keras в качестве высокоуровневого API для TensorFlow.

Цель исследования

Необходимо разработать систему распределенной интеллектуальной обработки визуальной информации с использованием мобильных устройств, включая алгоритм автоматического распределения частей нейронной сети для этапов ее обучения и эксплуатации между мобильным устройством и сервисами облачной инфраструктуры с учетом типа сети, возможностей мобильного устройства и канала связи.

Основной проблемой является именно оптимальное разделение сети на слои таким образом, чтобы минимизировать нагрузку на процессор мобильного устройства и на канал связи, при этом не выходя за рамки приемлемого времени отклика всей системы в целом. Основной проблемой является наличие в системе нестабильного канала связи с высокой вероятностью потери пакетов и разрывов соединений.

Пути решения проблем

Основной проблемой является автоматическое разделение нейросети на распределенные автономные слои и согласование части на мобильном устройстве с частью во внешней сети. При дополнительном обучении сети в серверной части необходимо согласовывать новые коэффициенты сети, особенно для ее автономной части. Кроме того, для мобильных устройств всегда необходимо учитывать высокую вероятность полностью автономной работы при плохом качестве связи.

Для использования мобильной библиотеки TensorFlow lite доступны несколько готовых моделей, для обработки изображений (классификация) используются Mobilenet, ResNet, NASNet и Inception, также для MobileNet доступны квантовые(Quantized) модели. Библиотека изначально рассчитана на офлайн режим работы с локальной сетью и классификатором, и ориентирована на реальное время. TF-lite изначально содержит данные модели в файле с расширением tflite в котором в формате FlatBuffers содержится прямая выгрузка структур данных обученной сети. Данный файл получается путем использования конвертера TensorFlow

Первым этапом является собвственно создание и обучение нейросети на основе Keras, Tensorflow или совместимых фреймворков. Затем модель сохраняется одним из совместимых способов, экспортируется в виде двух файлов – собственно модели FlatBuffer и файла .tflite с информацией о метках Labels. Файлы используются для инициализации модели, если модель уже инициализирована, пересоздается экземпляр объекта tensorflow (рисунок 1).

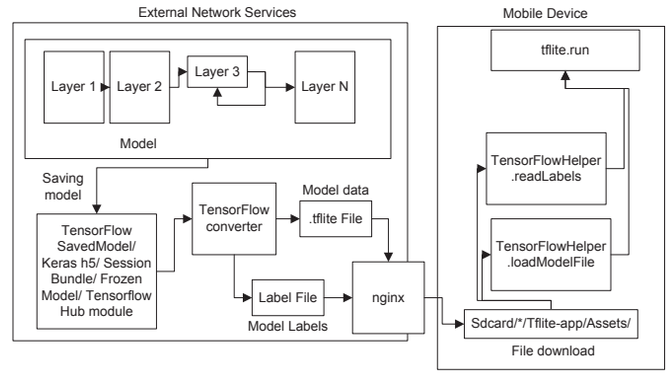

Рис. 1. Схема переноса нейросети на мобильное устройство

Fig. 1. The scheme of transferring a neural network to a mobile device

Основной проблемой такой схемы является большой размер нейронной сети (например, NASNet large занимает более 350 Мб, ResNet_V2_101 более 170 Мб) и отсутствие механизмов промежуточного взаимодействия со структурой сети. Базовая структура данных формата FlatBuffers для TF Lite показана на рисунке 2. Как видно из рисунка, структура данных содержит массивы информации с графом, состоящим из подграфов (слоев), которые состоят из списка тензоров, состоящих из операторов и их свойств.

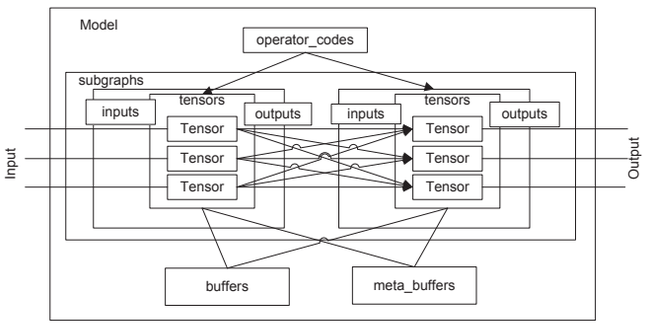

Рис. 2. Общая структура данных TFLite
Fig. 2. General TFLite data structure

Поскольку структура данных позволяет синтезировать уже обученную сеть за счет наличия буферов с параметрами, ее можно разделить на части, разделив также и буферы данных, оставив только те, что необходимы используемыми операторами. При этом можно сократить объем данных, передаваемых единовременно для инициализации модели, при обновлении модели можно передавать только разностный файл с параметрами. Однако при таком подходе есть и недостатки – количество передаваемых между слоями параметров определяется как n*m, где n-количество операторов с исходящими каналами, m – со входящими в следящем слое. При этом, если тип оператора подразумевает передачу int64 или complex64 – объем данных даже для 100 оператором может достигать 80000 байт для всех связей, кроме того все операции выполняются асинхронно и нет пакетизации на выходе.

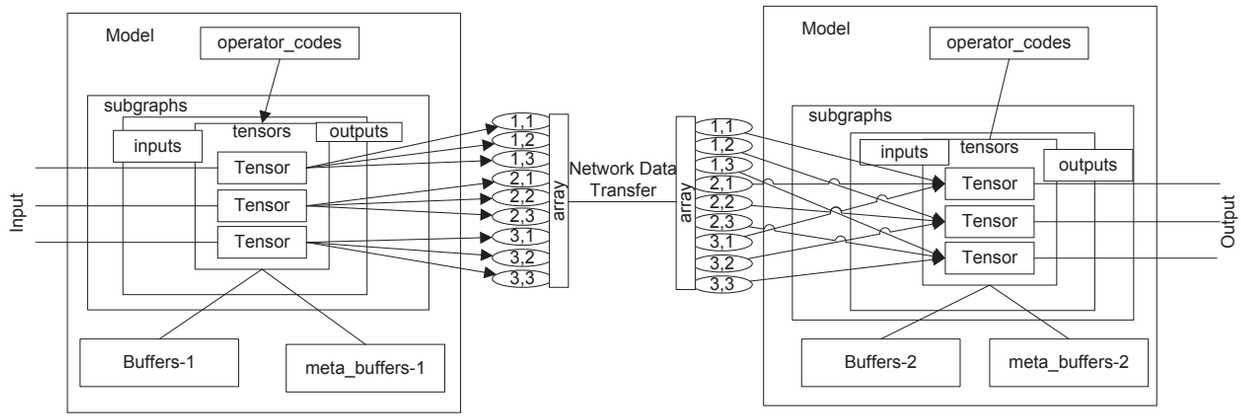

Рис. 3. Схема разделения модели
Fig. 3. Model separation scheme

Тензор разрывать нельзя, поскольку внутри существуют асинхронные операторы, которые должны ожидать выполнение всех ветвей кода для расчета выхода (пример на рисунке 4)

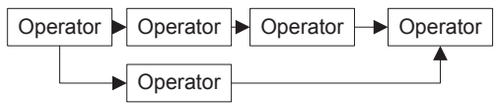

Рис. 4. Пример ветвления внутри
Fig. 4. Example of branching inside

Корректная работа TF Lite требует загрузки всей модели в память, и, например, для загрузки ResNet v2-101 исходным размером немного больше 100Мб, требуется более 500Мб свободной памяти. Процесс разбиения модели состоит из разбиения массива tensors на две части и последующего разбиения всех связанных массивов, таких как operators, quantization и прочих. Внутри массивов также могут содержаться массивы параметров. Разбивка буферов данных не производится.

Для определения места разделения необходимо провести эксперимент и выявить нагрузку, которую дает на процессор и память каждый слой. Для этого по очереди отключаются все слои после исследуемого и снимаются показания на тестовом наборе данных. Одновременно определяется объем данных, передаваемых между слоями как объем выходных данных последнего работающего слоя. Отключение слоев автоматизировано запускающим скриптом.

Затем производится расчет оптимального места разделения с учетом заданных граничных параметров. Пусть Cmax - максимальная доступная полоса пропускания, Dmax – максимально допустимая задержка при передаче данных, CPUmax – максимальное допустимое использование процессора (такты / proc/[PID]/stat) на удаленном устройстве, Mmax – максимальное допустимое использование памяти на удаленном устройстве (pmap –x [PID]). Тогда CPUi – суммарное использование процессора при расчете i слоев, Ci – выходной объем данных, Mi=max(Pmapi, i=1..n), где Pmapi – периодический дамп использования памяти процессами TF, n – количество слоев.

j: Mj = max(Mi: i=1..n, Mi<=Mmax) – максимальное потребление памяти, которое еще не превышает максимальное значение Mmax ;

k: CPUk = max(CPUi: i=1..n, CPUi<=CPUmax) – максимальное потребление процессорного времени, которое еще не превышает максимальное значение CPUmax;

q: Cq = max(Ci: i=1..n, Ci<=Cmax*Dmax) - максимальный объем данных, который еще можно передать, не нарушая параметры QoS по задержке и пропускной способности.

g = min (j, k, q) – наименьший вариант, подходящий под граничные требования.

При этом, вводится нечеткая граница для всех максималь-

ных граничных значений, при которой если 2 параметра из трех удовлетворяют имеют разницу не более, чем в 1 уровень, третий параметр принимается равным наименьшему оставшемуся параметру, если различия не превышают 20% от максимального уровня параметра. Это позволяет немного превышать пороговые значения для максимально оптимального с точки зрения загрузки удаленных ресурсов способа размещения.

Для завершения работы нейросети на выбранном уровне необходимо сохранять все выходные данные, параметры и переменные для передачи их на другое устройство для возобновления с того же места. Для этого используется тот же формат ProtocolBufer, что и для основного сохранения и распределения модели.

Планирование эксперимента

Для экспериментальных исследований разработанного метода была использована сеть ResNet_V2_101, которая должны была классифицировать изображения, полученные на мобильном устройстве Android. Для исключения влияния субъективных факторов в роли мобильного устройства был использован эмулятор Android 7 на Linux, выделенный канал Ethernet с ограничением пропускной способности и эмуляцией потери пакетов на базе Linux qdisk.

Для запуска второй части нейросети был использован кластер из 2 серверов Intel Xeon E5-2650/RAM 32G/GPU GTX-1080/11G с установленным TensorFlow. Сначала сеть была обучена на тестовом наборе ImageNet 11K с помошью PyTorch [17]. Она содержит 11221 классов и 12.4 млн. картинок, поэтому ее обучение заняло довольно продолжительное время даже при уменьшение размера картинок в 2 раза (около 2 недель), одна эпоха около 2 часов, 38 эпох. Результат обучения при выгрузке в формат SavedModel .pb занимал чуть больше 200 Мб.

Затем сеть была модифицирована таким образом, что бы начинать и прекращать выполнение сети на заданных в при запуске номерах слоев. Это позволяет проводить как измерения нагрузки, так и начинать выполнять сеть не с начала, а с заранее заданного слоя. Параметр задавался при начале эксперимента, после проведения измерений по потреблению памяти и процессора.

Полная сеть resnet_101 потребляет почти 6Гб GPU в пике при batch_size=8. Поскольку тест будет запускаться на эмуляторе Android с 2Гб ОЗУ (API 27, Nexus 5X), максимальное потребление нейросети ограничим 1Гб.

Для запуска на Andoid был использован модифицированный пример TFLite для Android Studio. К сожалению этот пример требует версию bazel, которая корректно не компилируется в Windows, поэтому сборка запускалась на Linux Ubuntu 16.04 в Android Studio 3.1.4. Данные обученной сети при помощи TensorFlow Lite Optimizing Converter (toco) экспортировались в приложение в папку Asset вместе с метками (labels.txt). Тестовое приложение не выводит на экран результат и посылает его на виртуальный сервер для измерений. Картинка поставляется также с виртуального сервера.

Поскольку ResNet использует типовые блоки для формирования всей структуры (conv 1, conv 2.x conv 3.x), которые чередуются, разделения будут проводиться в местах смены типа блоков.

Полученные результаты

Разделение по границам слоев разного состава дало следующие конфигурации:

а) 1-12 слой на мобильном устройстве (conv 1-2), 13-101 на сервере, выходной набор данных 56x56, потребление трафика 2.7Мб на изображение;

б) 1-24 слой на мобильном устройстве (conv 1-3), 25-101 на сервере, выходной набор данных 28x28, потребление трафика 1.3Мб на изображение;

в) все слои на мобильном устройстве, выходной набор данных 1x1, потребление трафика 100б на изображение;

г) один слой на мобильном устройстве (conv 1), 2-101 на сервере, выходной набор данных 112x112, потребление трафика 38Мб на изображение.

В потреблении трафика учтена вся сумма всех параметров и данных, необходимых для передачи на следующий уровень.

На рисунке 5 показаны графики потребления оперативной памяти для случая с запуском модели только на устройстве (1-101 layers), первых 24 (1-24), первых 12 (1-12), и одного уровня (1-1). Как видно из графика, даже запуск только первых 24 уровней снижает потребление памяти в несколько раз.

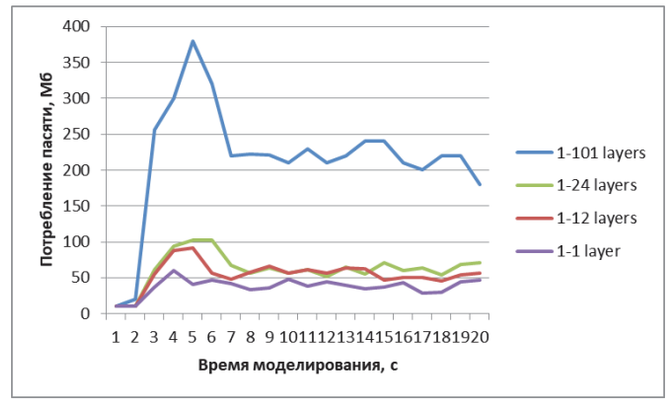

Рис. 5. Потребление памяти на Android при различных разделениях
Fig. 5. Android memory consumption under various partitions

На рисунке 6 показаны графики измерения производительности для случая с запуском модели только на устройстве (1-101 layers), первых 24 (1-24), первых 12 (1-12), и одного уровня (1-1). Производительность измерялась в атематическом режиме при получении изображения с сервера и выгрузки результатов обратно на сервер. Как видно из графиков, производительность вполне достаточная для использования в режиме фото и некритичного реального времени, хотя даже при 5 fps распознавание может выглядеть очень отзывчивым. Однако на реальном оборудовании скорость может быть существенно снижена из-за ограничений камеры, скорости фокусировки, первичной обработки изображения. При распознавании видео данной производительности будет недостаточно, требуется использовать платформы с поддержкой TFlite GPU API (Android 8). Вычисления проводились путем подсчета количества полностью обработанных изображений за интервал времени 1 сек.

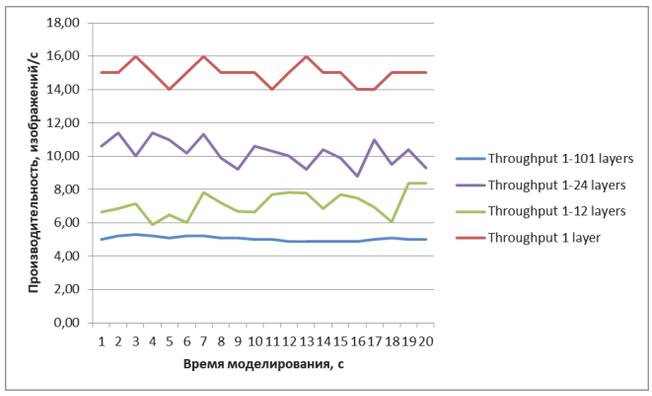

Рис. 6. Производительность при разделении слоев
Fig. 6. Performance when separating layers

Однако при использовании разделения сети необходимо учитывать потребление трафика сети и пропускную способность. На рисунке 7 показаны графики измерения потребления пропускной способности во всех случаях распределения кроме случая (1-1). В случае только с 1 слоем объем данных превысил возможности эмуляции. В случае с (1-12) и (1-24) требования к пропускной способности составили соответственно около 7 и 10 Мбит/с, что довольно много для мобильной сети, но немного для Wi-Fi и городских подключений LTE.

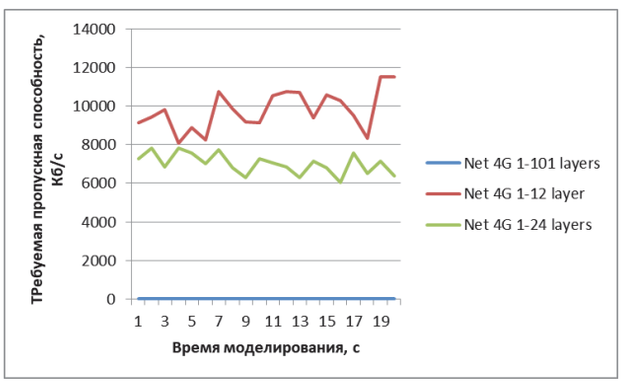

Рис. 7. Необходимая пропускная способность канала связи для максимально доступной производительности распознавания
Fig. 7. The required bandwidth of the communication channel for maximum recognition performance

Дополнитеьно был проведен эксперимент по производительности распознавания на сетях 3G/ Пропускная способность сети 3G была эмулирована на скорости 1Мбит/с с повышением до 1540, с потерями 1% и задержкой 100мс и стандартным отклонением 20мс с нормальным законом распределения, каждый 5 пакет будет послан вне очереди с задержкой 10мс, 0.1% пакетов будет испорчен:

- tc qdisc change dev eth0 root netem delay 100ms 20ms distribution normal
- tc qdisc change dev eth0 root netem loss 1%
- tc qdisc change dev eth0 root netem corrupt 0.1%
- tc qdisc change dev eth0 root netem gap 5 delay 10ms
- tc qdisc add dev eth0 root tbf rate 1024kbit latency 50ms burst 1540

Данное ограничение будет действовать только на исходящий из Android трафик. На рисунке 8 показана производительность при ограничениях сети. Пока не заканчивалась передача данных нейросети новые изображения для нее не передавались.

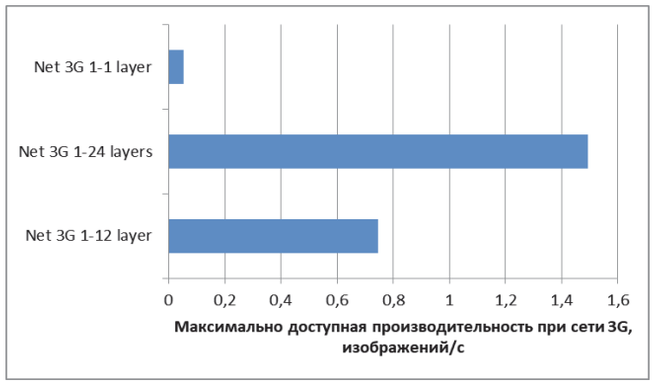

Рис. 8. Максимальная производительность на канале 3G 1024Кбит/с
Fig. 8. Maximum performance on the 3G channel 1024 Kbps

Как видно из графика, производительности сети 3G хватает для обработки 1 изображения в секунду, что вполне достаточно для задач статической обработки.

Заключение

В результате экспериментальных исследований возможностей вертикального разделения нейросети для использования в мобильных устройствах было выявлено:

Перевод нейросетевых вычислений на мобильные платформы возможен лишь отчасти, необходима глубокая оптимизация и использование специальных инструментов. Обучение нейросетей на мобильных платформах доступно лишь в виде отдельных экспериментов;

Нейросетевые структуры могут быть распределены как вертикально, так и горизонтально в режиме вычислений, но мобильные платформы сейчас не поддерживают такие режимы;

Разделение по слоям и распределение вычислений на последовательные блоки в разных устройствах возможно и в некоторых случаях является единственным способом запуска нейросети на мобильных устройствах с ограничением памяти;

Производительность распределенной сети повышается при наличии доступных сетевых ресурсов и необходимой скорости передачи данных.

Благодарность

Исследование выполнено при финансовой поддержке Правительства Оренбургской области и Российского фонда фундаментальных исследований (проекты №18-47-560017, №18-07-01446 и №18-37-00485).

Acknowledgement

The study was carried out with the financial support of the Government of the Orenburg region and the Russian Foundation for Basic Research (projects No. 18-47-560017, No. 18-07-01446 and No. 18-37-00485).

Список использованных источников

[1] Свёрточные нейронные сети: взгляд изнутри // DataSides. 2017. [Электронный ресурс]. URL: http://ru.datasides.com/code/cnn-convolutional-neural-networks/ (дата обращения: 23.08.2018).

[2] *Iandola F.N., Han S., W. Moskewicz M.W.* SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size // arXiv:1602.07360v4 [cs.CV], 2016. 13 p. URL: https://arxiv.org/pdf/1602.07360.pdf (дата обращения: 23.08.2018).

[3] *Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, Xuanzhe Liu.* DeepCache: Principled Cache for Mobile Deep Vision // Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18). ACM, New York, NY, USA, 2018. Pp. 129-144. DOI: 10.1145/3241539.3241563

[4] *Shankar S., Robertson D., Ioannou Y., Criminisi A., Cipolla R.* Refining Architectures of Deep Convolutional Neural Networks // Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, 2016. Pp. 2212-2220. DOI: 10.1109/CVPR.2016.243

[5] *Teerapittayanon S., McDanel B., Kung H.T.* Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices // Proceedings of 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). Atlanta, GA, 2017. Pp. 328-339. DOI: 10.1109/ICDCS.2017.226

[6] *LiKamWa R., Hou Y., Gao Y., Polansky M., Zhong L.* RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision // Proceedings of 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (*ISCA).* Seoul, 2016. Pp. 255-266. DOI: 10.1109/ISCA.2016.31

[7] *Loc H.N., Lee Y., Balan R.K.* DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications // Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17). ACM, New York, NY, USA, 2017. Pp. 82-95. DOI: 10.1145/3081333.3081360

[8] *Mathur A., Lane N.D., Bhattacharya S., Boran A., Forlivesi C., Kawsar F.* DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models using Wearable Commodity Hardware // Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17). ACM, New York, NY, USA, 2017. Pp. 68-81. DOI: 10.1145/3081333.3081359

[9] *Jouppi N.* Google supercharges machine learning tasks with TPU custom chip // Google Cloud, 2016. [Электронный ресурс]. URL: https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip (дата обращения: 23.08.2018).

[10] *Lovejoy B.* Apple moves to third-generation Siri back-end, built on open-source Mesos platform // 9to5mac. 2015. [Электронный ресурс]. URL: https://9to5mac.com/2015/04/27/siri-backend-mesos/ (дата обращения: 23.08.2018).

[11] *Hauswald J.* et al. DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers // Proceedings of 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA). Portland, OR, 2015. Pp. 27-40. DOI: 10.1145/2749469.2749472

[12] *Zhang Q., Yang L.T., Chen Z.* Privacy Preserving Deep Computation Model on Cloud for Big Data Feature Learning // IEEE Transactions on Computers. 2016. Vol. 65, no. 5. Pp. 1351-1362. DOI: 10.1109/TC.2015.2470255

[13] *Kang Y., Hauswald J., Gao C., Rovinski A., Mudge T., Mars J., Tang L.* Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge // ACM SIGARCH Computer Architecture News. 2017. Vol. 45, issue 1. Pp. 615-629. DOI: 10.1145/3093337.3037698

[14] *Zhang Y., Huang G., Liu X., Zhang W., Mei H., Yang S.* Refactoring android Java code for on-demand computation offloading // Proceedings of the ACM international conference on Object oriented programming systems languages and applications (OOPSLA '12). ACM, New York, NY, USA, 2012. Pp. 233-248. DOI: 10.1145/2384616.2384634

[15] *Wang X., Liu X., Zhang Y., Huang G.* Migration and execution of JavaScript applications between mobile devices and cloud // Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity (SPLASH '12). ACM, New York, NY, USA, 2012. Pp. 83-84. DOI: 10.1145/2384716.2384750

[16] *Zhang Y., Huang G., Zhang W., Liu X., Mei H.* Towards module-based automatic partitioning of java applications // Frontiers of Computer Science. 2012. Vol. 6, issue 6. Pp. 725-740. DOI: 10.1007/s11704-012-2220-x

[17] Full imagenet dataset // GitHub. 2017. [Электронный ресурс]. URL: https://github.com/tornadomeet/ResNet/blob/master/README.md#imagenet-11k (дата обращения: 23.08.2018).

Поступила 23.08.2018; принята в печать 10.10.2018;
опубликована онлайн 10.12.2018.

References

[1] Convolutional Neural Networks: a View from the Inside. DataSides. 2017. Available at: http://ru.datasides.com/code/cnn-convolutional-neural-networks/ (accessed 23.08.2018). (In Russian)

[2] Iandola F.N., Han S., W. Moskewicz M.W. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:*1602.07360v4 [cs.CV], 2016. 13 p. Available at: https://arxiv.org/pdf/1602.07360.pdf (accessed 23.08.2018).

[3] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, Xuanzhe Liu. DeepCache: Principled Cache for Mobile Deep Vision. *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18).* ACM, New York, NY, USA, 2018, pp. 129-144. DOI: 10.1145/3241539.3241563

[4] Shankar S., Robertson D., Ioannou Y., Criminisi A., Cipolla R. Refining Architectures of Deep Convolutional Neural Networks. *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Las Vegas, NV, 2016, pp. 2212-2220. DOI: 10.1109/CVPR.2016.243

[5] Teerapittayanon S., McDanel B., Kung H.T. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. *Proceedings of 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS).* Atlanta, GA, 2017, pp. 328-339. DOI: 10.1109/ICDCS.2017.226

[6] LiKamWa R., Hou Y., Gao Y., Polansky M., Zhong L. RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision. *Proceedings of 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA).* Seoul, 2016, pp. 255-266. DOI: 10.1109/ISCA.2016.31

[7] Loc H.N., Lee Y., Balan R.K. DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications. *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17).* ACM, New York, NY, USA, 2017, pp. 82-95. DOI: 10.1145/3081333.3081360

[8] Mathur A., Lane N.D., Bhattacharya S., Boran A., Forlivesi C., Kawsar F. DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models using Wearable Commodity Hardware. *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17).* ACM, New York, NY, USA, 2017, pp. 68-81. DOI: 10.1145/3081333.3081359

[9] Jouppi N. Google supercharges machine learning tasks with TPU custom chip. Google Cloud, 2016. Available at: https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip (accessed 23.08.2018).

[10] Lovejoy B. Apple moves to third-generation Siri back-end, built on open-source Mesos platform. 9to5mac. 2015. Available at: https://9to5mac.com/2015/04/27/siri-backend-mesos/ (accessed 23.08.2018).

[11] Hauswald J. et al. DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers. *Proceedings of 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA).* Portland, OR, 2015, pp. 27-40. DOI: 10.1145/2749469.2749472

[12] Zhang Q., Yang L.T., Chen Z. Privacy Preserving Deep Computation Model on Cloud for Big Data Feature Learning. *IEEE Transactions on Computers.* 2016; 65(5):1351-1362. DOI: 10.1109/TC.2015.2470255

[13] Kang Y., Hauswald J., Gao C., Rovinski A., Mudge T., Mars J., Tang L. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *ACM SIGARCH Computer Architecture News.* 2017; 45(1):615-629. DOI: 10.1145/3093337.3037698

[14] Zhang Y., Huang G., Liu X., Zhang W., Mei H., Yang S. Refactoring android Java code for on-demand computation offloading. *Proceedings of the ACM international conference on Object oriented programming systems languages and applications (OOPSLA '12).* ACM, New York, NY, USA, 2012, pp. 233-248. DOI: 10.1145/2384616.2384634

[15] Wang X., Liu X., Zhang Y., Huang G. Migration and execution of JavaScript applications between mobile devices and cloud. *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity (SPLASH '12).* ACM, New York, NY, USA, 2012, pp. 83-84. DOI: 10.1145/2384716.2384750

[16] Zhang Y., Huang G., Zhang W., Liu X., Mei H. Towards module-based automatic partitioning of java applications. *Frontiers of Computer Science.* 2012; 6(6):725-740. DOI: 10.1007/s11704-012-2220-x

[17] Full imagenet dataset. GitHub. 2017. Available at: https://github.com/tornadomeet/ResNet/blob/master/README.md#imagenet-11k (accessed 23.08.2018).

Submitted 23.08.2018; revised 10.10.2018;
published online 10.12.2018.

About the authors:

Yury A. Ushakov, Candidate of Engineering Sciences, Associate Professor at the Department of Geometry and Computer Science, Orenburg State University (13 Pobeda Av., Orenburg 460000, Russia), ORCID: http://orcid.org/0000-0002-0474-8919, unpk@mail.ru

Petr N. Polezhaev, Lecturer at the Department of Computer Security and Mathematical Maintenance of Information Systems, Orenburg State University (13 Pobeda Av., Orenburg 460000, Russia), ORCID: http://orcid.org/0000-0001-7747-646X, newblackpit@mail.ru

Aleksandr E. Shukhman, Candidate of Pedagogic Sciences, Associate Professor, Head of the Department of Geometry and Computer Science, Orenburg State University (13 Pobeda Av., Orenburg 460000, Russia), ORCID: http://orcid.org/0000-0002-4303-2550, shukhman@gmail.com

Margarita V. Ushakova, Lecturer of the Department of Geometry and Computer Science, Orenburg State University (13 Pobeda Av., Orenburg 460000, Russia), ORCID: http://orcid.org/0000-0003-4462-9946, m.v.ushakova@mail.ru