# Параллельные вычисления. Лабораторная работа 2

## Введение

Цель работы – проектирование и разработка свёрточного слоя нейронной сети на GPU.

Задание – ознакомиться с теоретическим материалом и особенностями реализации, оценить результаты работы программы при разных данных.

## 1. Теоретическая часть

*Свёрточная нейронная сеть*(англ. convolutional neural network, CNN) — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание изображений, входит в состав технологий глубинного обучения (англ. deep learning). Использует некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Название архитектура сети получила из-за наличия операции свёртки [1].

Свёртка (англ. convolution) — это математическая операция, показывающая «схожесть» одной функции с отражённой и сдвинутой копией другой.

Применительно к изображениям, можно дать следующее определение: свёртка – это операция, с помощью которой выполняется вычисление нового значения для пикселя, являющегося центром шаблона, учитывая значения окружающих, покрытых шаблоном пикселей. Обычно шаблон, называемый *ядром свертки* или *фильтром*, является квадратной матрицей n\*n, где n — нечетное число. Итак, для вычисления нового значения выбранного пикселя ядро свертки «накладывается» своим центром (именно поэтому важна нечетность размера матрицы) к данному пикселю. Окружающие пиксели также накрываются ядром. Затем высчитывается сумма, где слагаемыми являются произведения значений пикселей на значения ячейки ядра, накрывшей данный пиксель. Полученное значение является новым значением выбранного пикселя. В процессе свертки применяется как проход по всем пикселям изображения, так и со смещением (англ. stride) ядра свертки на несколько пикселей в зависимости от задачи. Таким образом, фильтры — это матрицы, которые позволяют находить на изображении специальные особенности, подобно тому, как работает зрительная кора головного мозга. Например, это могут быть прямые линии, линии под наклоном, а могут быть и сложные паттерны типа человеческого лица. На рисунке 1 представлена визуализированная операция свёртки.

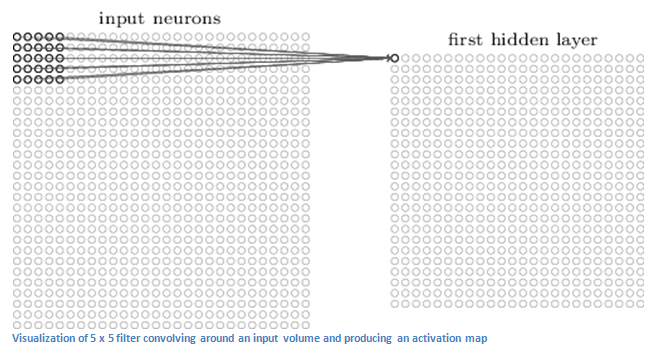


Рисунок 1 – Применение фильтра 5х5 к исходному изображению

Слой свёртки ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) convolutional layer) — это основной блок свёрточной нейронной сети. Слой свёртки включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты матричного произведения для каждого фрагмента).

Более подробно свёрточная нейронная сеть разобрана по ссылке: <https://ru.wikipedia.org/wiki/Свёрточная_нейронная_сеть>.

## 2. Практическая часть

В ходе выполнения лабораторной работы предлагается имплементировать сверточный слой на CUDA, а также разработать аналог для CPU, чтобы оценить и сравнить время выполнения программ.

## 2.1 Проектирование функции ядра

Рассмотрим подробнее основную программу kernel.cu и что представляет собой функция ядра. Ядро является потоковой функцией – большое количество потоков (threads) параллельно исполняют тело ядра. В данной лабораторной работе каждый поток отвечает за вычисление одного пикселя. Так как присутствует ограничение в количестве потоков в блоке, что не позволяет обработать целое изображение в 1 блоке даже сравнительно небольших изображений, то все потоки рассредоточены на множество блоков.

Итак, функция ядра работает следующим образом.

1. На вход подается изображение inImage. Это изображение имеет структуру трехмерного массива [[[r, g, b], [r, g, b], ..., ], [[r, g, b], [r, g, b], ..., ], ... ]. Здесь [r, g, b] - это пиксель (так как он состоит из трех цветов), [[r, g, b], [r, g, b], ..., ] - это строчка пикселов, [[[r, g, b], [r, g, b], ..., ], [[r, g, b], [r, g, b], ..., ], ... ] - контейнер таких строчек. У изображения есть ширина и высота. Массив имеет размерность [высота [ширина [3]]]. Об этом важно помнить!

Если развернуть это изображение в одномерный массив, то его размерность будет высота Х ширина Х 3, тогда [[[r, g, b], [r, g, b], ..., ], [[r, g, b], [r, g, b], ..., ], ... ] превратится в r, g, b, r, g, b, r, g, b, ...

Замечание: мы будем говорить о *пикселе* (или *точке*) и о *цвете*. Пиксель - это [r, g, b] (то есть иногда для удобства мы будем представлять, как будто мы работаем не с трехмерным массивом, а с двухмерным). Цвет - это конкретное значение (от 0 до 255) цвета пикселя. Когда мы будем говорить о цвете, никакой абстракции уже не останется.

1. На вход подаются фильтры. Это массив вида [ [[1, 1, 1,], [1, 1, 1], [1, 1, 1]], ... ]. В данном случае передается массив матриц-фильтров 3Х3.

Замечание:

* фильтры должны быть квадратными;
* размерность обязательно нечетная;
* все фильтры имеют одинаковую размерность.

Каждый фильтр после применения к изображению формирует новое изображение меньшей размерности. На этом изображении (в зависимости от того, насколько правильно подобраны веса) четко отображаются обнаруженные признаки. Все эти изображения формируют карту признаков или featureMaps. Эти изображения мы будем сохранять в одноименную переменную.

Таким образом, featureMaps будет по сути массивом изображений в виде [[[[r, g, b], [r, g, b], ..., ], [[r, g, b], [r, g, b], ..., ], ... ], ...]. Сколько фильтров, столько и изображений, причем идут они соответственно номерам, то есть первый фильтр формирует первое изображение в массиве, второй - второе и т.д.

В тот же самый момент, мы возьмем все карты признаков, попиксельно сложим, разделим на количество фильтров, и результат сохраним в outImage. По способу хранения он идентичен inImage (однако меньшего размера).

\_\_global\_\_ void gpuCNN(

const double \*inImage, // входное изображение

std::size\_t width, // ширина изображения

std::size\_t height, // высота изображения

double \*filters, // веса фильтров

std::size\_t filtersCount, // кол-во фильтров

std::size\_t filterLineSize,

std::size\_t stride, // величина смещения

double \*outImage, // выходное изображение

double \*featureMaps, // карта признаков

std::size\_t outWidth, // ширина вых. изображения

std::size\_t outHeight) // высота вых. изображения

Переменная halfLineSize нужна для того, чтобы рассчитывать смещение. Когда мы находимся на нужном нам пикселе с координатами (X, Y), нам нужно применить операцию свертки к квадратному фрагменту исходного изображения, середина которого - этот самый пиксель. При этом верхний угол этого фрагмента имеет координаты (X - filterLineSize / 2, Y - filterLineSize / 2), а правый - (X - filterLineSize / 2, Y - filterLineSize / 2). Чтобы постоянно не делить, мы будем использовать halfLineSize.

auto halfLineSize = filterLineSize / 2;

В CUDA функция ядра запускается параллельно на нескольких процессорах. При этом параллелится не все сразу, а по мере освобождения графических ядер. Но даже несмотря на это, у каждого "запуска функции" есть уникальные координаты: номер потока и номер блока, к тому же у блока есть размер (это может быть одномерная, двухмерная или трехмерная матрица). Учитывая номер потока и номер блока, можно получать индексы от 0 до ширины или высоты изображения и таким образом пройтись по всем пикселям.

auto outPixelX = threadIdx.x + blockIdx.x \* blockDim.x;

auto outPixelY = threadIdx.y + blockIdx.y \* blockDim.y;

Ниже представлены координаты пикселя входного изображения, которые нужны для формирования выходного. Умножив координату выходного изображения на страйд и прибавив половину размера ядра, мы получаем нужные координаты входного изображения.

auto pixelX = outPixelX \* stride + halfLineSize;

auto pixelY = outPixelY \* stride + halfLineSize;

Так как выходное изображение развернуто в одномерный массив, нам нужно учесть все размерности. В классическом виде a[i][j] = a[i \* size + j], но тут нужно помнить, что outWidth и outHeight - это длина и ширина в ПИКСЕЛЯХ, а на каждый пиксель приходится 3 цвета, то есть 3 элемента массива. Таким образом, outPixelY \* outWidth \* 3 + outPixelX \* 3 будет задавать индексы, кратные 3, то есть индексы начал пикселей. Чтобы достучаться до нужного цвета, надо прибавить индекс пикселя.

auto outColorPos = outPixelY \* outWidth \* 3 + outPixelX \* 3 + colorIdx;

Далее нужно написать основной цикл. В нем для каждого цвета применяем фильтры поочередно (еще циклы). Необходимо учесть, что мы должны прибавить к текущей координате ПИКСЕЛЯ (X, Y) смещение, рассчитанное таким образом, чтобы для (0, 0) матрицы ядра мы ходили на (X - halfLineSize, Y - halfLineSize) на изображении. Также нужно пересчитать координаты пикселя в одномерные координаты конкретного цвета. Результаты записываются в featureMaps (так как они все лежат в одномерном массиве друг за другом, мы должны пропустить filterIdx \* outWidth \* outHeight \* 3, чтобы попасть на нужный map) и в outImage.

В основной части программы для запуска функции ядра нужно посчитать количество блоков по Х и Y. Количество блоков по Х и Y считаем так: есть nThreads \* nThreads потоков, которые могут обрабатывать изображение в 1 блоке, один поток вычисляет значение одного выходного пикселя. Значит количество блоков (разное по X и Y) - это количество раз, которое нужно запускать nThreads потоков, чтобы обработать все пиксели. В программе будем использовать nThreads = 32 (т.е. в блоке статично используются все 1024 потока).

auto nBlocksX = Device::size.outImageWidth / threads.x;

nBlocksX += (0 == Device::size.outImageWidth % threads.x) ? 0 : 1;

auto nBlocksY = Device::size.outImageHeight / threads.y;

nBlocksY += (0 == Device::size.outImageHeight % threads.y) ? 0 : 1;

dim3 blocks(nBlocksX, nBlocksY);

Запуск функции ядра:

gpuCNN <<< blocks, threads >>> (

Device::inImage,

imageWidth,

imageHeight,

Device::filters,

filtersCount,

filterLineSize,

stride,

Device::outImage,

Device::featureMaps,

Device::size.outImageWidth,

Device::size.outImageHeight

);

## 2.2 Практическое задание

Необходимые материалы к лабораторной работе расположены в папке labs/lab2:

* img\_to\_text.py — утилита для преобразования JPEG в одномерный массив целых чисел. Принимает на вход имя файла (т.е. изображения), в качестве опционального аргумента – имя выходного файла с результатом работы (по умолчанию out.txt);
* kernel.cu — это главная программа и там же функция ядра. Запускается с помощью компилятора nvcc. Принимает на вход файл с одномерным массивом (out.txt), который выдает img\_to\_text.py, ширину и длину изображения, размер страйда. В результате обработки формируется два файла: out\_features.txt с полученными матрицами признаков и out\_image.txt с изображением, пропущенным через фильтры;
* text\_to\_img.py — это конвертер результата обработки (такого же одномерного массива, что выдает img\_to\_text.py) в изображение JPEG, которое можно посмотреть. На вход принимает имя файла с изображением в виде одномерного массива (out\_image.txt или out\_features.txt) и ширину изображения. В результате формируется файл result.jpg.

Порядок запуска:

1. Запуск утилиты img\_to\_text.py. Например,

python img\_to\_text.py lama.jpg

1. Скомпилировать и выполнить исполняемый файл, например:

nvcc kernel.cu -std=c++11 -arch=compute\_37 -code=sm\_37

./a.out out.txt 800 600 1

1. Получить результат в виде изображения:

python text\_to\_img.py out\_image.txt 798

python text\_to\_img.py out\_features.txt 798

## Задание

1. Разработать функцию ядра на GPU. Изменив количество фильтров, размер ядра, страйд, веса матриц ядер, запустить алгоритм для произвольных изображений. Количество прогонов алгоритма с разными параметрами свертки не менее 4. Привести результаты в отчете. Входное изображение брать такого разрешения, чтобы свертка выполнялась за разумное время: ~секунды.
2. Разработать функцию свертки на CPU.
3. Зафиксировать страйд, входное изображение, количество фильтров, заполнить таблицу ниже. Время выполнения считать усредненным по не менее 10 запускам алгоритмов. Входное изображение брать такого разрешения, чтобы свертка выполнялась за разумное время: ~секунды.

|  |  |  |
| --- | --- | --- |
| Размер ядра | Время выполнения, с (ускорение, ед) | |
| CPU | GPU |
| 3х3 |  |  |
| 5х5 |  |  |
| 7х7 |  |  |
| 9х9 |  |  |
| 17х17 |  |  |
| 21х21 |  |  |

1. Нарисовать 2 графика: время выполнения в зависимости от размера ядра и ускорение в зависимости от размера ядра. CPU и GPU представить на каждом графике линиями разных цветов.
2. Обосновать результаты.

## Список литературы

1. Свёрточные нейронные сети [Электронный ресурс] / URL: <https://ru.wikipedia.org/wiki/Свёрточная_нейронная_сеть> (Дата обращения 16.12.2017)
2. Фильтрация изображений с помощью свёртки [Электронный ресурс] / URL: <https://habrahabr.ru/post/62738/> (Дата обращения: 16.12.2017)