

**МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)**

**Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»**

**Лабораторная работа №3
по курсу «Программирование графических процессоров»**

Классификация и кластеризация изображений на GPU.

Выполнил: М.А. Бронников

Группа: 8О-407Б

Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

Условие

Цель работы: Научиться использовать GPU для классификации и кластеризации изображений. Использование константной памяти.

Вариант 4. Метод максимального правдоподобия.

Программное и аппаратное обеспечение

Device: GeForce GT 545

Размер глобальной памяти: 3150381056

Размер константной памяти : 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров : 3

OS: Linux Mint 20 Cinnamon

Редактор: VSCode

Метод решения

Для начала необходимо посчитать значения средних по каждому из каналов и значения обратной матрицы ковариации для каждого из заданных признаков. Далее для каждого из пикселей изображения определяется его класс по формуле:

$$jc = \arg \max_j \left[- (p - \text{avg}_j)^T * \text{cov}_j^{-1} * (p - \text{avg}_j) - \log(\|\text{cov}_j\|) \right]$$

Описание программы

Для выполнения программы я реализовал собственный класс изображения в методе которого и вызывался kernel. Этот класс не потерпел значительных изменений со времени выполнения второй лабораторной работы.

Для выполнения операции я инициализировал на этапе компиляции массив константной памяти необходимого размера, а именно максимального количества возможных классов, умноженного на размер, необходимый для хранения вычислительной информации для каждого из классов.

```
struct class_data{  
    float avg_red;  
    float avg_green;  
    float avg_blue;  
    float cov11;  
    float cov12;  
    float cov13;
```

```

float cov21;
float cov22;
float cov23;
float cov31;
float cov32;
float cov33;
float log_det;
};
// constant memory
constant class_data computation_data[MAX_CLASS_NUMBERS];

```

Для копирования данных с host в этот участок памяти у меня есть следующий код:

```

throw_on_cuda_error(
    cudaMemcpyToSymbol(computation_data, cov_avg,
        MAX_CLASS_NUMBERS*sizeof(class_data), 0, cudaMemcpyHostToDevice)
);

```

После чего Сами же данные для вычисления я рассчитываю на CPU, поскольку расчет на GPU не даст значимого прироста к производительности.

После чего я вызываю kernel с заданным количеством блоков и потоков, где я преобразую изображение в кластеризованное по описанному алгоритму:

```

dim3 threads = dim3(MAX_X, MAX_Y);
dim3 blocks = dim3(BLOCKS_X, BLOCKS_Y);

classification<<<blocks, threads>>>(d_data, height, width, indexes.size());
throw_on_cuda_error(cudaGetLastError());

```

В самом kernel мы вычисляем правдоподобие для пикселя по каждому из классов и записываем наиболее вероятный номер класса.

```

global void classification(uint32_t* picture, uint32_t h, uint32_t w, uint8_t classes){
    uint32_t idx = blockIdx.x * blockDim.x + threadIdx.x;
    uint32_t idy = blockIdx.y * blockDim.y + threadIdx.y;

    uint32_t step_x = blockDim.x * gridDim.x;
    uint32_t step_y = blockDim.y * gridDim.y;

    // run for axis y
    for(uint32_t i = idy; i < h; i += step_y){
        // run for axis x
        for(uint32_t j = idx; j < w; j += step_x){

```

```

// init very big num
float min = INT32_MAX;

uint32_t pixel = picture[i*w + j];
uint8_t ans_c = 0;

for(uint8_t c = 0; c < classes; ++c){
    float red = RED(pixel);
    float green = GREEN(pixel);
    float blue = BLUE(pixel);
    float metric = 0.0;

    red -= computation_data[c].avg_red;
    green -= computation_data[c].avg_green;
    blue -= computation_data[c].avg_blue;

    float temp_red = red*computation_data[c].cov11 +
    green*computation_data[c].cov21 + blue*computation_data[c].cov31;

    float temp_green = red*computation_data[c].cov12 +
    green*computation_data[c].cov22 + blue*computation_data[c].cov32;

    float temp_blue = red*computation_data[c].cov13 +
    green*computation_data[c].cov23 + blue*computation_data[c].cov33;

    // dot + log(|cov|)
    metric = (temp_red*red + temp_green*green + temp_blue*blue + computation_data[c].log_det);
    if(metric < min){
        ans_c = c;
        min = metric;
    }
}

#ifdef WITH_IMG
pixel ^= ((uint32_t) ans_c) << 24;
#endif

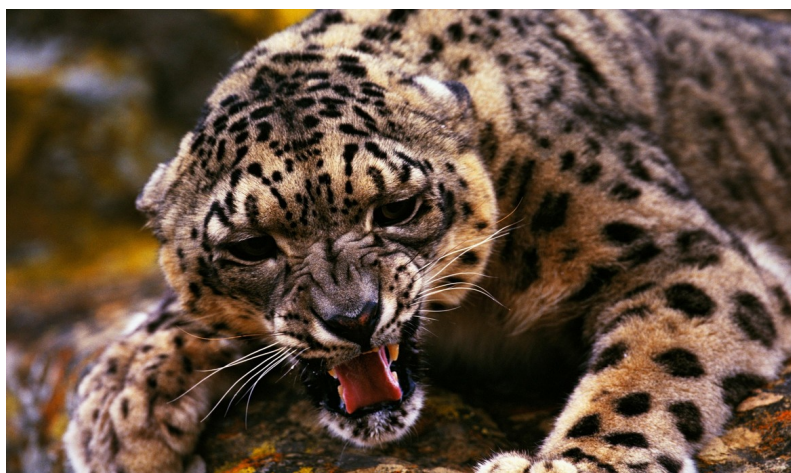
picture[i*w + j] = pixel;
}
}
}

```

После вызова kernel я копирую данные в массив и освобождаю выделенную память.

Результаты

Для иллюстрации результатов работы алгоритма я выбрал 2 изображения:



После чего я применил конвертер в заданный в задании формат, который я взял из материалов от преподавателя. После чего я применил к ним свою программу, только вместо номера класса на место α канала я записываю avg по наиболее вероятному классу. Полученные результаты:





В обоих изображениях я выделял несколько точек из трех классов. На лицо некоторые неточности, которые устраняются с добавлением новых классов и новых точек для них.

Таблица замеров времени выполнения

CPU или конфигурация	Размер изображения	Время выполнения
CPU	800 1200	99.141
CPU	1600 2560	290.946
<<<(4, 4), (4, 4)>>>	800 1200	7.38314
<<<(4, 4), (4, 4)>>>	1600 2560	29.7029
<<<(4, 4), (16, 16)>>>	800 1200	1.37098
<<<(4, 4), (16, 16)>>>	1600 2560	5.76128
<<<(4, 4), (32, 32)>>>	800 1200	1.22864
<<<(4, 4), (32, 32)>>>	1600 2560	5.04371
<<<(16, 16), (4, 4)>>>	800 1200	5.1625
<<<(16, 16), (4, 4)>>>	1600 2560	21.9541
<<<(16, 16), (16, 16)>>>	800 1200	2.1607
<<<(16, 16), (16, 16)>>>	1600 2560	6.9639
<<<(16, 16), (32, 32)>>>	800 1200	1.24474
<<<(16, 16), (32, 32)>>>	1600 2560	4.74365
<<<(32, 32), (4, 4)>>>	800 1200	5.01814
<<<(32, 32), (4, 4)>>>	1600 2560	21.536
<<<(32, 32), (16, 16)>>>	800 1200	1.11478
<<<(32, 32), (16, 16)>>>	1600 2560	4.46362
<<<(32, 32), (32, 32)>>>	800 1200	1.50403
<<<(32, 32), (32, 32)>>>	1600 2560	5.12384

Выводы

В ходе лабораторной работы я познакомился с одним из методов сегментации изображения. Такие методы сейчас набирают особенную популярность с развитием нейронных сетей и компьютерного зрения в частности.

Реализованный мной алгоритм, несмотря на то, что неплохо классифицирует пиксели на изображении, имеет некоторые недостатки, главный из которых — необходимость вручную задавать классы и некоторое количество пикселей, которое к нему относиться. Так как зачастую требуется просто выделить некоторые кластеры, вне зависимости от начального разбиения, имеет смысл обратиться к одному из методов «без учителя», такому как k-means.

В ходе выполнения работы возникла трудность в том, что я более суток не мог понять, что логарифм в формуле функции правдоподобия берется не от нормы матрицы, а модуля ее определителя. Этот момент был для меня не очевидным и потратил много моих сил и нервов.