Document

Authors Info

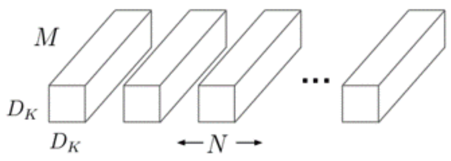
Group Number: 17

Members:

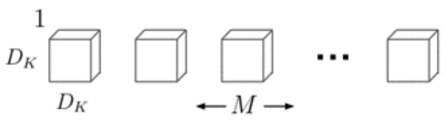
- 518030910408 周韧
- 518030910406 郑思榕
- 518030910388 胡家熙

Introduction

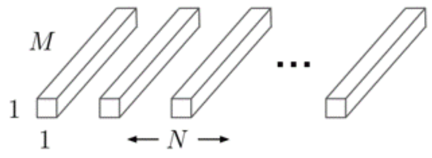
In this project we implements a inference model MobileNetV2 which was proposed by Google in 2018. This model has advantages on saving parameters and computation without losing accuracy, for it applied the depth-wise separable convolution layers. In fact, depth-wise separable convolution is composed by depth-wise convolution and point-wise convolution, which is used to replace the function of standard convolution.



(a) Standard Convolution Filters

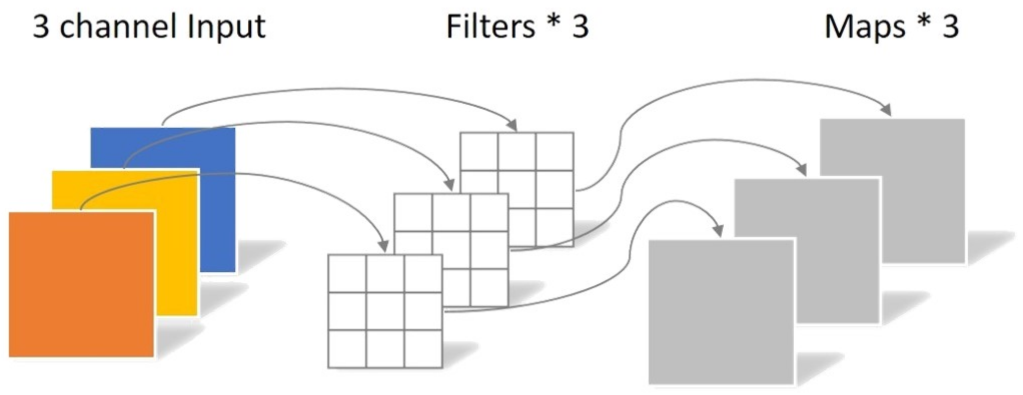


(b) Depthwise Convolutional Filters

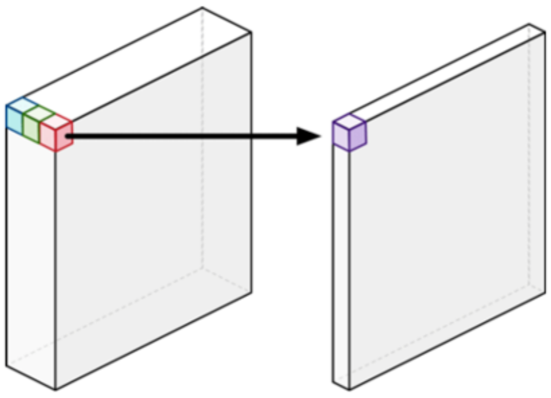


(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

For Depth-wise convolution, it just uses single channel filters. The number of filters is equal to the channels of input figures and one filter is responsible for a single channel. By depth-wise convolution layer, the information of nearby pixels on this channel can be extracted.

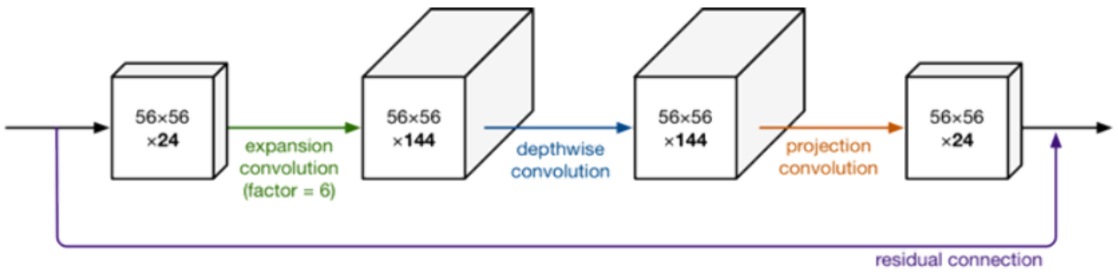


For point-wise convolution, it is actually the 1*1 standard convolution. The motivation of point-wise convolution is aggravating information of all channels.

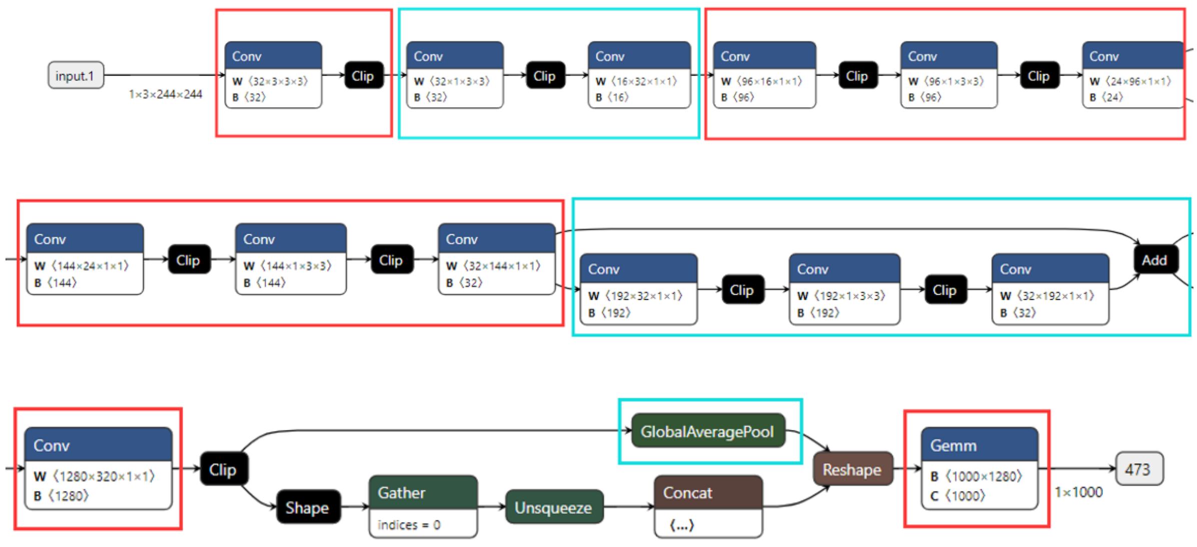


Model Structure

In our MobileNetV2 model, a common used structure is inverted residual block, which is shown as follows:



This block uses a structure that has a skip-connection layer as ResNet and contains 2 point-wise convolution layers and 1 depth-wise convolution layer. So we decomposed our model into several blocks.



Then we need to build the following layers in this model:

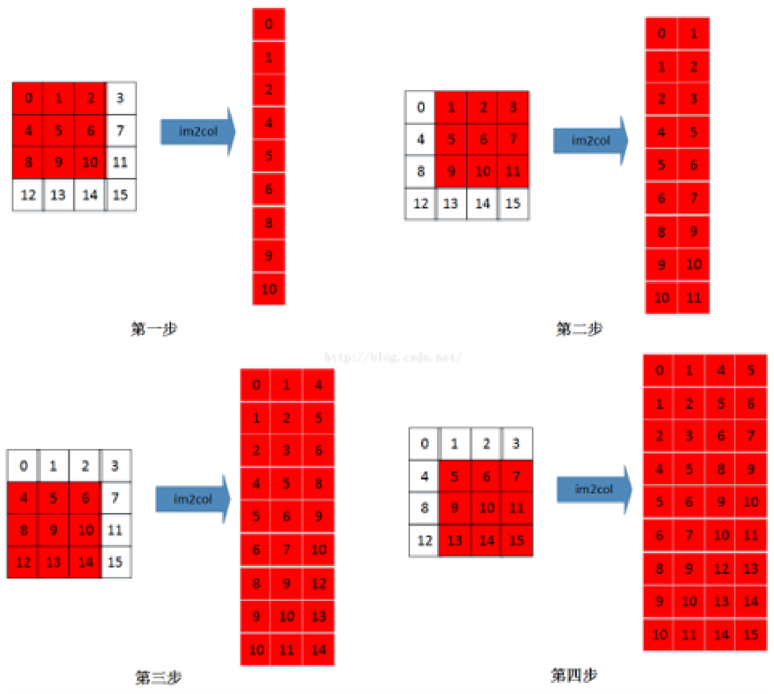
Basic Layers

- ☐ Standard Convolution 3D
- ☐ Depth-wise Convolution
- ☐ Point-wise Convolution
- ☐ Skip Connection Layer
- ☐ Global Average Pool
- ☐ Relu6
- ☐ Temporary Store Layer
- ☐ Full Connection Layer

Implementation Details

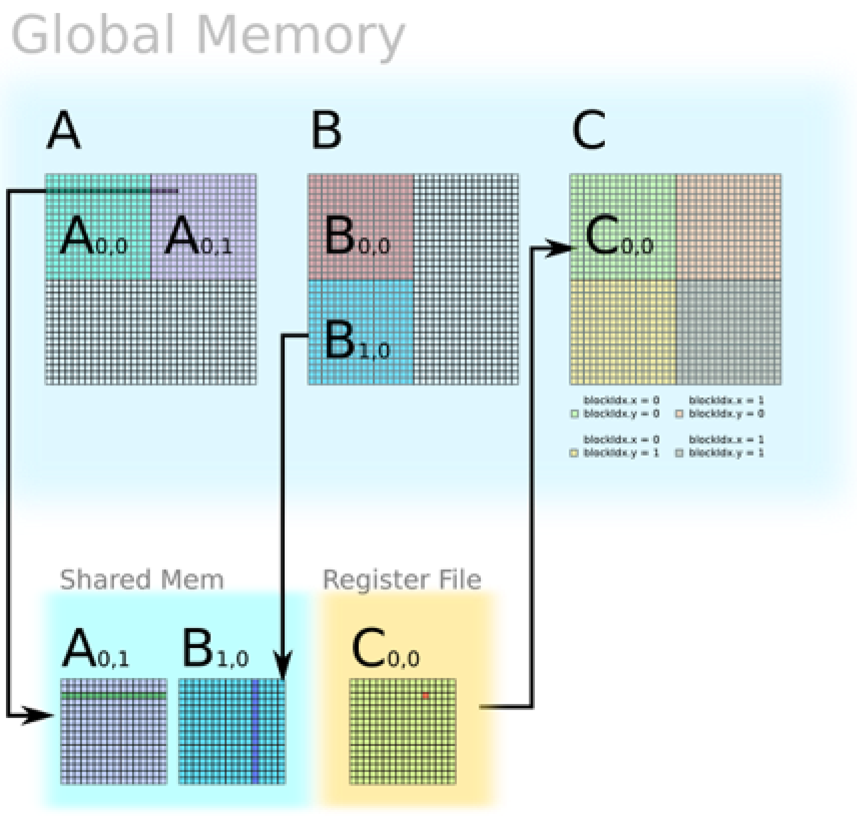
Standard Convolution

The first layer of MobileNetV2 is a standard convolution layer and the normal way is just to use multiple loops to multiply weight in filter and data in input tensor in the corresponding positions. However we found some methods to optimize this layer, such as img2col, FFT and winograd. We finally choose img2col as the optimization way.



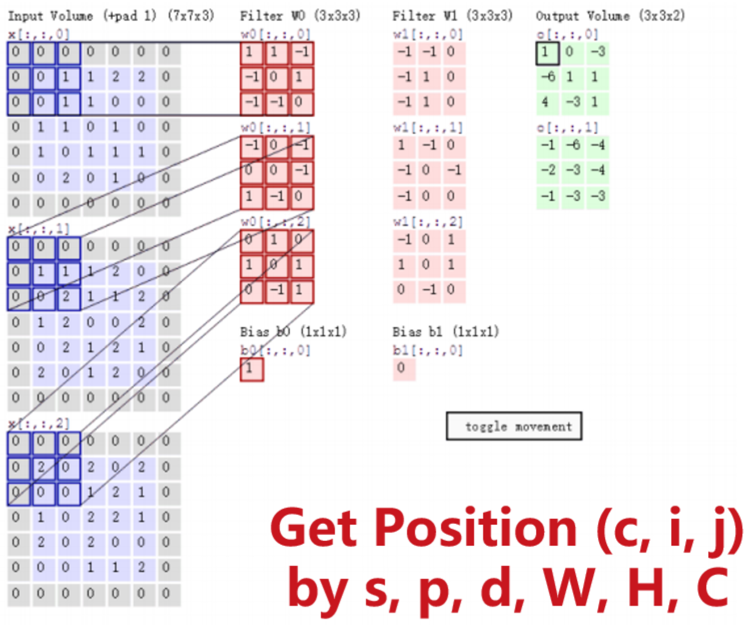
Img2col algorithm is a way to align the matrix which will be calculated in to a column and then converted convolution to matrix multiplication. So what we should in our implementation is actually first rearrange data and then apply matrix multiplication about filter and input tensor.

For matrix multiplication, we used 3 ways to calculate result. The first way is natively using a row to multiply a column. Because of the low loading-computation ratio, the performance is not satisfying. So we then applied tiling technique and loop unrolling by "#pragma unroll". Then one thread just loads one data and calculates twice, so we improve the workload of a thread. We arrange that one thread loads 4 numbers to calculate result, which enhances the performance well.


Global Memory
A
B
C
A0,0
A0,1
B0,0
B1,0
C0,0
blockIdx.x = 0 blockIdx.y = 0
blockIdx.x = 1 blockIdx.y = 0
blockIdx.x = 0 blockIdx.y = 1
blockIdx.x = 1 blockIdx.y = 1
Shared Mem
Register File
A0,1
B1,0
C0,0

Depth-wise Convolution

For depth-wise convolution, it is actually not easy to do some optimization. In original caffe implementation, it just uses loops to calculate each channel and brings too much unnecessary cost. In our inference, the max number of channels is 1280 so it not suitable to use loop for each channel. Thus, we apply point to point multiplication which is similar to the normal implementation of standard convolution.



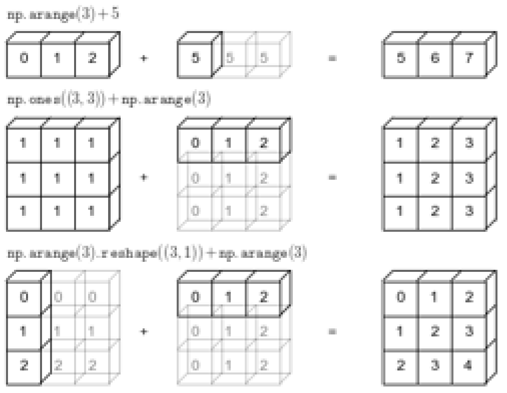
Point-wise Convolution

This layer is not difficult for point-wise convolution is actually 1*1 standard convolution. Then it can be easily arranged data and applied matrix multiplication. Besides, we have no necessary to consider pad or stride so then this layer can be simply built.

Other Optimizations

Linear Layer: This layer is also called full connection layer and it is just be used in the end of this model. And it can simply be converted to matrix multiplication.

Relu6 and Add Bias: When we add bias, we should know that it has a mechanism called broadcasting. For instance, a dimension of w is (32, 3, 3, 3), but the dimension of b is (32,). Then a number in b has been used many times in the process of adding bias. So we moved b to constant memory then it can be load more fast. In addition, if we apply relu6 layer, we just need to using min\max function of the add result, which means relu6 layer and adding bias can be merged into one layer.


np.arange(3)+5
np.ones((3,3))+np.arange(3)
np.arange(3).reshape((3,1))+np.arange(3)

Global Average Pool: In this layer, we just need to add the total sum of each channel and get the average number. So it can be calculated by reduction to reduce branch instructions. Besides, simply adding the numbers of single channel is also fast because the computation does not need to much numbers so that it can be summed up quickly.

Index conversion: When apply img2col algorithm, an important part is rearranging data. And there are two ways to calculate index: one is from image array index to column array index, another is just the reverse. For the former one, one position corresponds to multiple positions in column array, while the latter one is one position corresponding to a fix position in image array. The performance has no much difference but these are two vary ways to do that.

Code Document

Enter Main Function

```
int main() {
    /**
     * initialize model
     * read Input and Output Data
     * Run Inference Funtion
     * Calculate Average Time
     * Free Memory
     */
    ...
}
```

Initialize Model

```
void InitModel() {
    /**
     * @desc initialize parameters and memories
     * Allocate Memory
     * Read Binary Weights and Bias into Memory
     * Move Data to GPU Memory
     * Create Some Handles
     */
}
```

Infer Data by MobileNetV2 Model

```
void inference(float *input, float *output) {
    /**
     * @desc infer images data
     * move images data to cuda
     * pass Conv2d, DepthwiseConv, PointwiseConv, StoreBackup, AddLayer,
GlobalAvgPool and LinearLayer
     * move cuda result to cpu
     *
     * @param input : input Images, CHW format
     * @param output : inferece result
     */
}
```

Free Memory

```
void FreeMemory() {
    /**
     * @desc Free all Memory used on Cuda
     */
}
```

Layers

Standard Convolution

```
void Conv2d(float* in_tensor, float** out_tensor_p, float* w, float* b, int
in_shape, int in_c, int k_shape, int out_c, int stride, int pad, cublasHandle_t*
handle_p) {
    /**
     * @desc Standard Convolution Layer
     * rearrange data
     * matrix multiply
     * @param in_tensor : input data
     * @param out_tensor_p : the address which points to output data
     * @param w : weight data
     * @param b : bias data
     * @param in_shape : input h and w
     * @param in_c : input channel
     * @param k_shape : kernel h and w
     * @param out_c : output channel
     * @param stride : convolution stride
     * @param pad : convolution pad
     * @param handle_p : cublas handle
     */
}
```

Depth-wise Convolution

```
void DepthwiseConv(float* in_tensor, float** out_tensor_p, float* w, float* b,
int in_shape, int in_c, int k_shape, int out_c, int stride, int pad, bool
is_log) {
    /**
     * @desc Depth-wise Convolution Layer
     * group convolution
     * @param in_tensor : input data
     * @param out_tensor_p : the address which points to output data
     * @param w : weight data
     * @param b : bias data
     * @param in_shape : input h and w
     * @param in_c : input channel
     * @param k_shape : kernel h and w
     * @param out_c : output channel
     * @param stride : convolution stride
     * @param pad : convolution pad
     * @param is_log : whether print log
     */
}
```

Point-wise Convolution

```
void PointwiseConv(float* in_tensor, float** out_tensor_p, float* w, float* b,
int in_shape, int in_c, int out_c, bool is_relu, bool is_log, cublasHandle_t*
handle_p) {
    /**
     * @desc Point-wise Convolution Layer
     * point-wise rearrange data
     * matrix multiply
     * @param in_tensor : input data
     * @param out_tensor_p : the address which points to output data
     * @param w : weight data
     * @param b : bias data
     * @param in_shape : input h and w
     * @param in_c : input channel
     * @param k_shape : kernel h and w
     * @param out_c : output channel
     * @param is_relu : whether apply relu layer
     * @param is_log : whether print log
     * @param handle_p : cublas handle
     */
}
```

Add Layer

```
void AddLayer(float* A, float* B, float** C_p, int channels, int shape) {
    /**
     * @desc Skip Connection Layer
     * @param A : mat1
     * @param B : mat2
     * @param C_p : the address which points to the result mat
     * @param channels : input channels
     * @param shape : input h and w
     */
}
```

Global Average Pool

```
void GlobalAvgPool(float* in_tensor, float** out_tensor_p, int channels, int
in_shape) {
    /**
     * @desc calculate average value for each channel
     * @param in_tensor : input mat
     * @param out_tensor_p : the address which points to output data
     * @param channels : input channels
     * @param in_shape : input h and w
     */
}
```

Linear Layer

```
void LinearLayer(float* in_tensor, float** out_tensor_p, float* w, float* b, int
in_len, int out_len, cublasHandle_t* handle_p) {
    /**
     * @desc Full Connection Layer
     * matrix multiply
     * @param in_tensor : input data
     * @param out_tensor_p : the address which points to output data
     * @param w : weight data
     * @param b : bias data
     * @param in_len : input length
     * @param out_len : output length
     * @param handle_p : cublas handle
     */
}
```

Intermediate Result Storage

```
void StoreBackup(float* in_tensor, float** out_tensor_p, int out_lens) {
    /**
     * @desc Store Intermediate Result for Skip Connection Layer
     * @param in_tensor : input data
     * @param out_tensor_p : the address which points to output data
     * @param out_len : input and output length
     */
}
```