Papers I have read

宋明辉

May 23, 2018

Contents

List of Figures

Usage Instructions:

- This book include all papers I have read from 2017.05.28.
- The magenta represent the online link.
- The red represents the links in this book including reference, figure, table and others.
- The purple represents the emphasize.

Chapter 1

Image processing based on CUDA

This chapter include the Image processing acceleration based on CUDA.

1.1 Novel multi-scale retinex with color restoration on graphics processing unit

1.1.1 Abstract

In this paper, a parallel application of the MSRCR+AL algorithm on a GPU is presented. For the various configurations in our test, the GPU-accelerated MSRCR+AL shows a scalable speedup as the resolution of an image increases. The up to $45\times$ speed up (1024×1024) over the single-threaded CPU counterpart shows a promissing direction of using the GPU-based MSRCR+AL in large scale, time-critical applications. We also achieved 17 frames per second in video processing (1280×720).

1.1.2 Content

In our implementation, the CUFFT provides a simple interface for computing FFTs. After the plans of both forward and inverse FFTs are created according to the CUFFT requirements, the image data and the Gaussian filters can be parallel transformed to frequency domain. The multiplication between image data and Gaussian filters in frequency domain is finished by ModulateandNormal- ize() function, which is also provided by the CUFFT library.

The atomic function atomicAdd() provided by CUDA is used in the kernel histogram function to guar- antee to be performed without interference from other threads.

1.1.3 Parallel optimization strage

size of thread block and grid

For example, the maximum number of threads on the lower capability version of CUDA is 512, but newer CUDA-enabled GPUs with 2.x computecapability can reach 1,024. But, each streaming multi- processer (SM) can only execute 1,536 threads simul-

taneously. Therefore, we set the number of threads per block at 192, which means each SM can fully execute eight blocks to maximize resources.

Memory access optimization

A thread needs 400–600 clock cycles to access the global memory, but only needs about 4 clock cycles to access fast memory units such as register and shared memory due to lower access latency. Therefore, taking full advantage of the multi-level GPU memory storage components can obtain quick data access to improve the execution performance effectively.

Loop unrolling

After loop unrolling, the code only needs to run one time to write the result. The number of write processes decreases 66.7% compared with the serial code. Also, it only needs one time to read the result, compared with three times had we used in the serial code. The number of read processes decreases 66.7% as well. Furthermore, this loop unrolling strategy can be applied to sum different scale results together for the Multi-scale Retinex. More importantly, in the reduction algorithm for the summation process, this strategy is used to greatly increase the calculation speed and reduce the instruction overhead.

1.1.4 Conclusion

see the Abstract subsection.

1.2 Image Convolution

This section includes all articles I have read relating to Image Convolution using CUDA. Image convolution is usually used in image filtering, like gaussian filter.

1.2.1 Naïve Implementation

From the idea of convolutio filter itself, the most naive approach is to use global memory to send data to device and each thread accesses this to compute convolution kernel. Our convolution kernel size is radius 8 (total 17×17 multiplicaiton for single pixel value). In image border area, reference value will be set to 0 during computation. This naive approach includes many of conditional statements and this causes very slow execution. The code is as shown below:

Listing 1.1: Image Convolution based on global memory

```
__global__ void gaussfilterGlo_kernel(float *d_imgOut, float *
    d_imgIn, int wid, int hei,
```

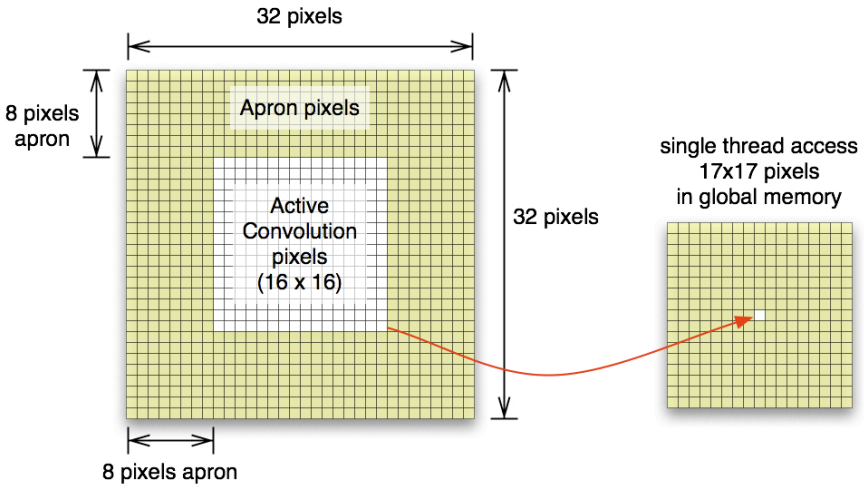


Fig 1.1. Image Convolution based on Global Memory

```
{
    int idx = threadIdx.x + blockDim.x * blockIdx.x;
    int idy = threadIdx.y + blockDim.y * blockIdx.y;

    if(idx > wid || idy > hei)
        return ;

    int filterR = (filterW - 1) / 2;

    float val = 0.f;

    for(int fr = -filterR; fr<= filterR; ++fr)            // row
        for(int fc = -filterR; fc <= filterR; ++fc)       // col
        {
            int ir = idy + fr;
            int ic = idx + fc;

            if((ic >= 0) && (ic <= wid - 1) && (ir >= 0) && (ir <=
                hei - 1))
                val += d_imgIn[INDX(ir, ic, wid)] * d_filter[INDX(fr
                    +filterR, fc+filterR, filterW)];
        }
    d_imgOut[INDX(idy, idx, wid)] = val;
```

```
}
```

For $396 \star 396$ input image, the time is 1.6ms. When the input filter is stored in constant memory or specified by '__restrict__', the time is 1.7 or 1.8 ms.

1.2.2 Naïve Shared Memory Implementation

The simplest approach to implement convolution in CUDA is to load a block of the image into a shared memory array, do a point-wise multiplication of a filter-size portion of the block, and then write this sum into the output image in device memory. Each thread block processes one block in the image. Each thread generates a single output pixel.

The algorithm itself is somewhat complex. For any reasonable filter kernel size, the pixels at the edge of the shared memory array will depend on pixels not in shared memory. Around the image block within a thread block, there is an *apron* of pixels of the width of the kernel radius that is required in order to filter the image block. Thus, each thread block must load into shared memory the pixels to be filtered and the apron pixels.

Note: The apron of one block overlaps with adjacent blocks. The aprons of the blocks on the edges of the image extend outside the image – these pixels can either be clamped to the color of pixels at the image edge, or they can be set to zero.

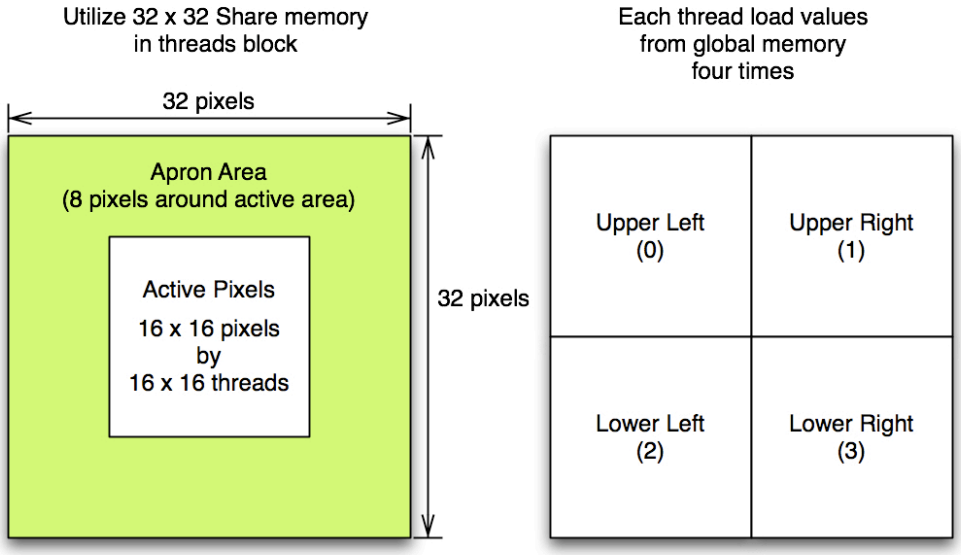


Fig 1.2. Image Convolution based on shared memory

The first attempt was to keep active thread size as same as previous and increase block size for apron pixels. This did not work since convolution kernel radius is 8 and it make block size to 32×32 (1024). This is bigger than G80 hardware limit (512 threads max per block).

Therefore, I changes scheme as all threads are active and each thread loads four pixels and keep the block size 16×16. Shared Memory size used is 32×32 (this includes all necessary apron pixel values for 16×16 active pixels). Below shows quite a bit of performance improve. This is almost $\times 2.8$ speed up over naive approach (in 2048 resolution).

Listing 1.2: Image colvotion based on Shared memory and Apron

```
__global__ void convolutionGPU(
..............................float *d_Result,
..............................float *d_Data,
..............................int dataW,
..............................int dataH
..............................)
{
....// Data cache: threadIdx.x , threadIdx.y
....__shared__ float data[TILE_W + KERNEL_RADIUS * 2][TILE_W +
    KERNEL_RADIUS * 2];

....// global mem address of this thread
....const int gLoc = threadIdx.x +
.......................IMUL(blockIdx.x, blockDim.x) +
.......................IMUL(threadIdx.y, dataW) +
.......................IMUL(blockIdx.y, blockDim.y) * dataW;

....// load cache (32x32 shared memory, 16x16 threads blocks)
....// each threads loads four values from global memory into shared
     mem
....// if in image area, get value in global mem, else 0
....int x, y;   // image based coordinate

....// original image based coordinate
....const int x0 = threadIdx.x + IMUL(blockIdx.x, blockDim.x);
....const int y0 = threadIdx.y + IMUL(blockIdx.y, blockDim.y);

....// case1: upper left
....x = x0 - KERNEL_RADIUS;
....y = y0 - KERNEL_RADIUS;
....if ( x < 0 || y < 0 )
........data[threadIdx.x][threadIdx.y] = 0;
....else
........data[threadIdx.x][threadIdx.y] = d_Data[ gLoc -
    KERNEL_RADIUS - IMUL(dataW, KERNEL_RADIUS)];

....// case2: upper right
....x = x0 + KERNEL_RADIUS;
....y = y0 - KERNEL_RADIUS;
....if ( x > dataW-1 || y < 0 )
........data[threadIdx.x + blockDim.x][threadIdx.y] = 0;
....else
........data[threadIdx.x + blockDim.x][threadIdx.y] = d_Data[gLoc +
    KERNEL_RADIUS - IMUL(dataW, KERNEL_RADIUS)];

....// case3: lower left
....x = x0 - KERNEL_RADIUS;
....y = y0 + KERNEL_RADIUS;
....if (x < 0 || y > dataH-1)
........data[threadIdx.x][threadIdx.y + blockDim.y] = 0;
....else
........data[threadIdx.x][threadIdx.y + blockDim.y] = d_Data[gLoc -
    KERNEL_RADIUS + IMUL(dataW, KERNEL_RADIUS)];
```

```
....// case4: lower right
....x = x0 + KERNEL_RADIUS;
....y = y0 + KERNEL_RADIUS;
....if ( x > dataW-1 || y > dataH-1)
........data[threadIdx.x + blockDim.x][threadIdx.y + blockDim.y] =
    0;
....else
........data[threadIdx.x + blockDim.x][threadIdx.y + blockDim.y] =
    d_Data[gLoc + KERNEL_RADIUS + IMUL(dataW, KERNEL_RADIUS)];

....__syncthreads();

....// convolution
....float sum = 0;
....x = KERNEL_RADIUS + threadIdx.x;
....y = KERNEL_RADIUS + threadIdx.y;
....for (int i = -KERNEL_RADIUS; i <= KERNEL_RADIUS; i++)
........for (int j = -KERNEL_RADIUS; j <= KERNEL_RADIUS; j++)
............sum += data[x + i][y + j] * d_Kernel[KERNEL_RADIUS + j]
    * d_Kernel[KERNEL_RADIUS + i];

....d_Result[gLoc] = sum;
}
```

Note : the value "gLoc - KERNEL_RADIUS - IMUL(dataW, KERNEL_RADIUS)" is the shift address of the image data on the upper left corner. 在本方法中，主要是索引的问题，所以又分为 Share Memory 的索引以及图像数据的索引。在具体实现过程中，选择是固定 thread Block 的大小，同时在 share memory 中添加边界。所以将图像数据拷贝到 Share Memory 中时，分了四次，分别对应：左上角，右上角，左下角、右下角。也就是图1.2中的对应关系，其处理过程就是将小的图像块映射到大的 Share memory 中，从这个方面进行理解。

1.2.3 Separable Gaussian Filtering

Separable Convolution

A tow-dimensional filter s is said to be separable if it can be written as the convolution of tow one-dimesional filters ***v*** and ***h***:

$$s = v * h$$

"How to determine if a matrix is an outer product of two vectors?"
"Go look at the **rank** function. ". Of course. If a matrix is an outer product of two vectors, its rank is 1.

So the test is this: The rank of A is the number of nonzero singular values of A, with some numerical tolerance based on eps and the size of A.

So how can we determine the outer product vectors? The answer is to go back to the svd function. Here's a snippet from the doc:

$[U, S, V] = svd(X)$ produces a diagonal matrix S of the same dimension as X, with nonnegative diagonal elements in decreasing order, and unitary matrices ***U*** and ***V*** so that $X = U * S * V'$

A rank 1 matrix has only one nonzero singular value, so $X = U * S * V'$ becomes $U(:,1) * S(1,1) * V(:,1)$. This is basically the outer product we were seeking. Therefore, we want the first columns of ***U*** and ***V***. (We have to remember also to use the nonzero singular value as a scale factor.)

Seperate Gaussian filter

First chose, somewhat artitrarily to split the scale factor, S(1, 1), "equally" between ***v*** and ***h***. Except for normal floating-point roundoff differences, gaussian and v*h are equal. Just show as following :

$$[U, S, V] = svd(X)$$
$$v = U(:,1) * sqrt(S(1,1))$$
$$h = V(:,1)`\times sqrt(S(1,1))$$
$$GassianFilter = v * h$$

More details can be found at :Separable Convolution.

1.2.4 Optimizing for memory coalescence

Base read/write addresses of the warps of 32 threads also must meet half-warp alignment requirement in order to be coalesced. If four-byte values are read, then the base address for the warp must be 64-byte aligned, and threads within the warp must read sequential 4-byte addresses. If the dataset with apron does not align in this way, then we must fix it so that it does.

The approach used in the row filter is to have additional threads on the leading edge of the processing tile, in order to make threadIdx.x == 0 always reading properly aligned address and thus to meet global memory alignment constraints for all warps. This may seem like a waste of threads, but it is of little importance when the data block, processed by a single thread block is large enough, which decreases the ratio of apron pixels to output pixels.

Each image convolution pass in both row and column pass is separated into two sub stages within corresponding CUDA kernels. The first stage loads the data from global memory into shared memory, and the second stage performs the filtering and writes the results back to global memory. We mustn’ t forget about the cases when row or column processing tile becomes clamped by image borders, and initialize clamped shared memory array indices with correct values. Indices not lying within input image borders are usually initialized either with zeroes or with values, corresponding to clamped image coordinates. In this sample we opt for the former.

In between the two stages there is a **__syncthreads( )** call to ensure that all threads have written to shared memory before any processing begins. This is necessary because threads are dependent on data loaded by other threads.

For both the loading and processing stages each active thread loads/outputs one pixel. In the computation stage each thread loops over a width of twice the filter radius plus 1, multiplying each pixel by the corresponding filter coefficient stored in constant memory. Each thread in a half-warp accesses the same constant address and hence there is no penalty due to constant memory bank conflicts. Also, consecutive threads always access consecutive shared memory addresses so no shared memory bank conflicts occur as well.

The column filter pass operates much like the row filter pass. The major difference is that thread IDs increase across the filter region rather than along it. As in the row filter pass, threads in a single half-warp always access different shared memory banks, but the calculation of the next/previous addresses involves increment/decrement by COLUMN_TILE_W, rather than simply 1. In the column filter pass we do not have inactive "coalescing alignment" threads during the load stage, because we assume that the tile width is a multiple of the coalesced read size. In order to decrease the ratio of apron to output pixels we want image tile to be as tall as possible, so to have reasonable shared memory utilization we shoot for as thin image tiles as possible: 16 columns.

1.3 CUDA C Best Practice Guide

1.3.1 Performance Metrics

Timing

- Using CPU Timers
  Should call *cudaDeviceSynchronize()* immediately before starting and stopping the CPU timer.

- Using CUDA GPU Timers
  The device will record a timestamp for the event when it reaches that event in the stream. This value is expressed in milliseconds and has a resolution of approximately half a microsecond.

Bandwidth

Bandwidth - the rate at which data can be transferred - is one of the most important gating factors for performance.

1.4 Npp Library Image Filters

1.4.1 Image Data

Line Step

All image data passed to NPPI primitives requires a line step to be provided. It is important to keep in mind that this line step is always specified in terms of bytes, not pixels.

1.5 PTX ISA 6.0

1.5.1 PTX Machine Model

The *Multiprocessor* maps each thread to one *scalar processor* core, and each scalar thread executes independently with its own instruction address and register state.

Individual threads composing a SIMT warp start together at the same program address but are otherwise free to branch and execute independently. A warp executes one common instruction at a time, so full efficiency is realized when all threads of a warp agree on their execution path. If threads of a warp diverge via a data-dependent conditional branch, the warp serially executes each branch path taken, disabling threads that are not on that path, and when all paths complete, the threads converge back to the same execution path.

Each multiprocessor has **on-chip memory** of the four following types :

- Local 32-bit registers per processor;
- Shared memory (parallel data cache);
- Read-only *constant cache* that is shared by all scalar processor cores;
- Read-only *texture cache*

The local and global memory spaces are read-write regions of **device memory** and are not cached.

If there are not enough registers or shared memory available per multiprocessor to process at least one block, the kernel will fail to launch.

Syntax

PTX programs are a collection of text source modules(files). PTX source modules have an assembly-language style syntax with instruction operation codes and operands. Pseudo-operations specify symbol and addressing management. The *ptxas* optimizing backend compiler optimizes and assembles PTX source modules to produce corresponding binary object files.

Source Format

PTX is case sensitive and uses lowercase for keywords.

Each PTX module must begin with a **.version** directive specifying the PTX language version, followed by a **.target** directive specifying the target architecture assumed.

Statements

A PTX statement is either a **directive** or an **instruction**. Statements begin with an optional label and end with a semicolon.

- Directive Statements
  Directive keywords begin with a dot, so no conflict is possible with user-defined identifiers. 如图1.3所示。
- Instruction Statements
  Instructions are formed from an instruction opcode followed by a **comma-separated** list of zero or more operands, and terminated with a semicolon.

Table 1 PTX Directives

.address_size	.file	.minnctapersm	.target
.align	.func	.param	.tex
.branchtargets	.global	.pragma	.version
.callprototype	.loc	.reg	.visible
.calltargets	.local	.reqntid	.weak
.const	.maxnctapersm	.section	
.entry	.maxnreg	.shared	
.extern	.maxntid	.sreg	

Fig 1.3. PTX Directives

Operands may be register variables, constant expressions, address expressions, or label names. The guard predicate follows the optional label and precedes the opcode, and is written as **@p** , where p is a predicate register. The guard predicate may be optionally negated, written as **@!p**.

The destination operand is first, followed by source operands. 如图1.4所示。

Table 2 Reserved Instruction Keywords

abs	div	or	sin	vavrg2, vavrg4
add	ex2	pmevent	slct	vmad
addc	exit	popc	sqrt	vmax
and	fma	prefetch	st	vmax2, vmax4
atom	isspacep	prefetchu	sub	vmin
bar	ld	prmt	subc	vmin2, vmin4
bfe	ldu	rcp	suld	vote
bfi	lg2	red	suq	vset
bfind	mad	rem	sured	vset2, vset4
bra	mad24	ret	sust	vshl
brev	madc	rsqrt	testp	vshr
brkpt	max	sad	tex	vsub
call	membar	selp	tld4	vsub2, vsub4
clz	min	set	trap	xor
cnot	mov	setp	txq	
copysign	mul	shf	vabsdiff	
cos	mul 24	shfl	vabsdiff2, vabsdiff4	
cvt	neg	shl	vadd	
cvta	not	shr	vadd2, vadd4	

Fig 1.4. Reserved Instruction Keywords

Chapter 2

FPGAs

Every section is arranged like follows:

- Abstract
  The abstract part of paper.
- Content
  The main idea of paper.
- Results
  The experiment implementation.
- Conclusion
  The conclusion part of paper.

2.1 Performance Comparison of FPGA, GPU and CPU in Image Processing 2009

2.1.1 Abstract

Many applications in image processing have high inherent parallelism. FPGAs have shown very high performance in spite of their low operational frequency by fully extracting the parallelism. In recent micro processors, it also becomes possible to utilize the parallelism using multi-cores which support improved SIMD instructions, though programmers have to use them explicitly to achieve high performance. Re- cent GPUs support a large number of cores, and have a po- tential for high performance in many applications. **However, the cores are grouped, and data transfer between the groups is very limited.** Programming tools for FPGA, SIMD in- structions on CPU and a large number of cores on GPU have been developed, but it is still difficult to achieve high perfor- mance on these platforms. In this paper, we compare the per- formance of FPGA, GPU and CPU using three applications in image processing; two-dimensional filters, stereo-vision and k-means clustering, and make it clear which platform is faster under which conditions.

Fig 2.1. 传统提升小波计算过程

2.1.2 Content

Compared three applications: two-dimension filters, stereo-vision, k-means clustering.

The high performance of FPGA comes from its flexibility which makes it possible to realize the fully optimized circuit for each application, and a large number of on-chip memory banks which supports the high parallelism.**FPGA can achieve extremely high performance in many applications in spite of its low op- erational frequency**.

GPU cores are grouped, the data transfer between groups is very slow.

GPU Analysis

It consists of 10 thread processor clus- ters. A thread processor cluster has three streaming mul- tiprocessors, eight texture filtering units, and one level-1 cache memory. Each streaming multiprocessor has one in- struction unit, eight stream processors (SPs) and one local memory (16KB). Thus, GTX280 has 240 SPs in total. Eight SPs in a stream-

ing multiprocessor are connected to one in- struction unit. This means that the eight SPs execute the same instruction stream on different data.

Two-Dimensional Filters

The computational complexity of filters is $O(w \times w)$, and w is radius of filter.

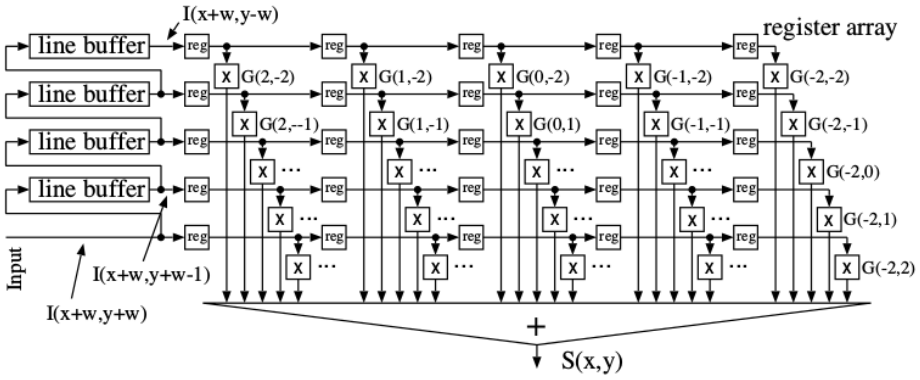


Fig 2.2. Circuits for non-separable filters

Fig2.2 shows the filter is 5×5 case.

Stereo Vision

This application is to get the distance to the location obtained from the two camera's disparity.The sum of absolute difference (SAD) is widely used to compared the windows because of its simplicity. More details can be found in paper[1]. **K-means Clustering**

More details can be found in paper [1].

2.1.3 Results

- Xilinx XC4VLX160.
- GeForce 280GTX, 1 GB DDR3, CUDA version 2.1.
- Intel Core2 Extreme QX6859.

The time to download images from main memory is not included. CPU has four cores, FPGA is fixed to 100MHz. Fig is the performance of two-dimension filters.

GPU is the fastest for all tested filter size.In this problem, filters can be applied to each pixel in the image in- dependently without using shared variables. So, GPU can show its best performance.

In the later two applications, the performance FPGA is much better than GPU.

2.1.4 Conclusion

We have compared the performance of GPU with FPGA and CPU (quad-cores) using three simple problems in im- age processing. GPU has a potential for achieving almost the same performance with FPGA. The number of cores in GTX280 is 240.Considering the trade-offs between the operational frequency of GPU (more than 10 times faster), and

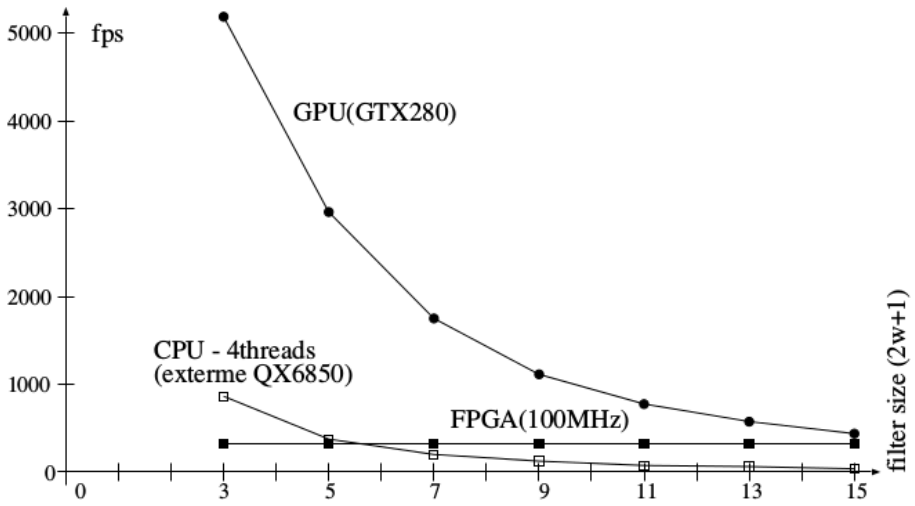


Fig 2.3. Performance of two-dimensional filters

the fine-grained parallelism in FPGA, this seems to be a natural consequence. However, GPU can show its potential only for naive computation methods, in which all pixels can be processed independently. For more sophisticated algorithms which use shared arrays, GPU can not execute those algorithms because of its very small local memory, or can not show good performance because of the memory access limitation caused by its memory architecture. GPU is slower than CPU in those algorithms (it may be possible to realize much better performance if we can find algorithms which can get around the limitations, but we could not find them). The performance of CPU is 1/12 - 1/7 of FPGA, which means that CPU with quad-cores can executes about 1/10 operations of FPGA in a unit time (the same algorithms are executed on CPU and FPGA). The performance of FPGA is limited by the size of FPGA and the memory bandwidth. With a latest FPGA board with DDR-II DRAM and a larger FPGA, it possible to double the performance by processing twice the number of pixels in parallel.

We have the following issues which have to be consid- ered. We have compared the performance using only three problems. The performances of the programs on GPU and CPU are not fully tuned up. In the comparison, power con- sumption and costs are not considered.

2.2 Fast FPGA Prototyping for real-time image processing with very high-level synthesis 2017

2.2.1 Abstract

Programming in high abstraction level can facilitate the development of digital signal processing systems. In the recent 20 yeahs, HLS has made significantly progress. However, due to the high complexity and computational intensity, image processing algorithms usually necessitate a higher abstraction environment than C-synthesis, and the current HLS tools do not have the ability of this kind. This paper presents a conception of

very high-level synthesis method which allows fast prototyping and verifying the FPGA-based image processing designs in the MATLAB environment. We build a heterogeneous development flow by using currently available tool kits for verifying the proposed approach and evaluated it within two real-life applications. Experiment results demonstrate that it can effectively reduce the complexity of the development by automatically synthesizing the algorithm behavior from the user level into the low register transfer level and give play to the advantages of FPGA related to the other devices.

2.2.2 Content

Advanced Digital Sciences Center(ADSC) of the University of Illinois reported that FPGA can achieve a speedup to 2-2.5x and save 84-92% of the energy consumption comparing to Graphics Processing Units(GPUs). ADSC indicates also that a manual FPGA design may consume 6-18 months and even years for a full custom hardware, while the GPUs(CUDA) based designed only 1-2 weeks.

Fig2.4 show the Gasjki-Kuhn's Y-chart comparing the conventional RTL with the HLS-based design flows.
The challenges of MATLAB-to-RTL synthesis include:

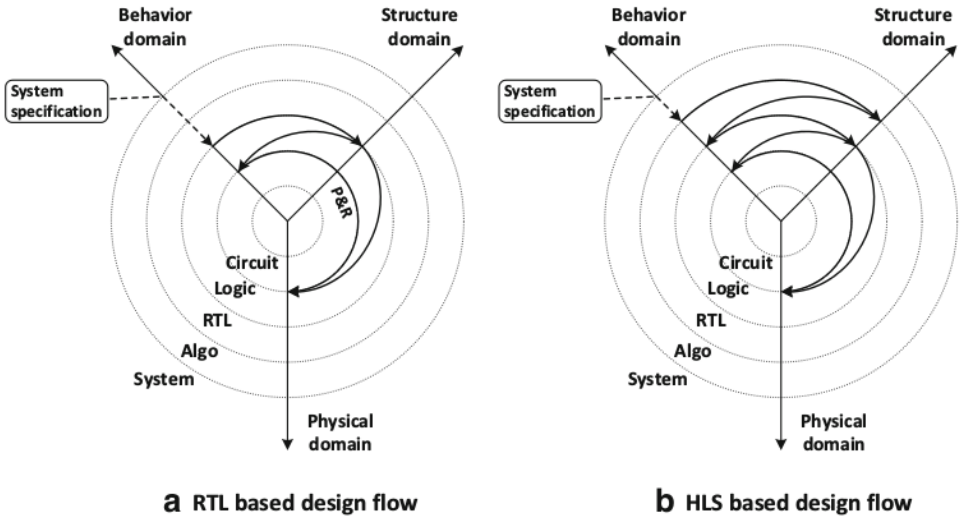


Fig 2.4. Comparison of RTL- and HLS-based design flows by using Gasjki-Kuhn's Y-chart: full lines indicate the automated cycles, while dotted lines the manual cycles.

- Operators in MATLAB perform different operations depending on the type of the operands, whereas the functions of the operators in RTL are fixed.

- MATLAB includes very simple and powerful vector operations such as the concatenation ``[ ]" and column operators ``$x(:)$" or ``end" construct, which can be quite hard to map to RTL.

- MATLAB supports ``polymorphism'' whereas RTL does not. More precisely, functions in MATLAB are genetic and can process different types of input parameters. In the behaviors of RTL, each parameter has only a single given type, which cannot change.
- MATLAB supports dynamic loop bounds or vector size, whereas RTL requires users to initialize explicitly them and cannot do any changes during the synthesis.
- The variables in MATLAB can be reused for different contents (different types), whereas RTL does not, as each variable has one unique type.

Two complex image processing applications: Kubelka-Munk genetic algorithm(KMGA) for the multispectral image based skin lesion assessments & level set method(LSM)-based algorithm for very high-resolution(VHR) satellite image segmentation..

Chapter 3

Image Fusion

3.1 Guided Image Filter 2013

3.1.1 Abstract

In this paper, we propose a novel explicit image filter called guided filter. Derived from a local linear model, the guided filter computes the filtering output by considering the content of a guidance image, which can be the input image itself or another different image. The guided filter can be used as an edge-preserving smoothing operator like the popular bilateral filter, but it has better behaviors near edges. The guided filter is also a more generic concept beyond smoothing: It can transfer the structures of the guidance image to the filtering output, enabling new filtering applications like dehazing and guided feathering. Moreover, the guided filter naturally has a fast and nonapproximate linear time algorithm, regardless of the kernel size and the intensity range. Currently, it is one of the fastest edge-preserving filters. Experiments show that the guided filter is both effective and efficient in a great variety of computer vision and computer graphics applications, including edge-aware smoothing, detail enhancement, HDR compression, image matting/feathering, dehazing, joint upsampling, etc.

3.1.2 Content

A general linear translation-variant filtering process, which involves a guidance image *I*, an filtering input image *p* and an output image *q*. Both I and p are given beforehand according to the application, and they can be **identical**! The filtering output at a pixel *i* is expressed as a weight average:

$$q_i = \sum_j W_{ij}(I) p_j \tag{3.1}$$

where *i, j* are pixel indexes. The filter kernel W_{ij} is a function of the guidance image I and independent of p.

3.1.3 Conclusion

3.2 Multiscale Image Fusion Through Guided Filtering

3.2.1 Abstract

We introduce a multiscale image fusion scheme based on GUided Filtering. Guided filtering can effectively reduce noise while preserving details boundaries. While restoring larger scale edges. The proposed multi-scale fusion scheme achieves optimal spatial consistency by using guided filtering both at the decomposing and at the recombination stage of the multiscale fusion process. First, size-selective iterative guided filtering is applied to decompose the source images into base and detail layers at multiple levels of resolution. Next, at each resolution level a binary weighting map is obtained as the pixelwise maximum of corresponding source saliency maps. Guided filtering of the binary weighting maps with their corresponding source images as guidance images serves to reduce noise and to restore spatial consistency. The final fused image is obtained as the weighted recombination of the individual detail layers and the mean of the lowest resolution base layers. Application to multiband visual (intensified) and thermal infrared imagery demonstrates that the proposed method obtains state-of-the-art performance for the fusion of multispectral nightvision images. The method has a simple implementation and is computationally efficient[16].

3.2.2 Contents

To data, a variety of image fusion algorithms have been proposed. A popular class of algorithms are the multi-scale image fusion schemes, which decompose the source images into spatial primitives at multiple spatial scales, then integrate these primitives to form a new multi-scale transform-based representation, and finally apply an inverse multi-scale transform to reconstruct the fused image. However, most of these techniques are computationally expensive and tend to oversharpen edges, which makes them less suitable for application in multiscale schemes

Bilateral Filter: It can reverse the intensity gradient near sharp edges.

Guided Filter:

The two filtering conditions are:

- the local filter output is a linear transform of the guidance image *G*
- as similar as possible to the input image *I*.

The first condition implies that:

$$O_i = a_k G_i + b_k \qquad \forall i \in \omega_k$$

where the ω_k is a square window of size $(2r+1) \times (2r+1)$. **The local linear model ensures that the output image O has an edge only at locations where the guidance image G also has one.** Linear coefficients a_k and b_k are constant in ω_k. They can be

estimated by minimizing the squared difference between the output image O and the input image I in the window ω_k, i.e. minimizing the cost function E:

$$E(a_k, b_k) = \sum_{i \in \omega_k} \left((a_k G_i + b_k - I_i)^2 + \epsilon a_k^2 \right)$$

where ϵ_k is a regularization parameter penalizing large a_k. The coefficients a_k and b_k can directly be solved by linear regression. Since pixel i is contained in several different window ω_k, the value of O_i depends on the window over which it is calculated:

$$O_I = \overline{a}_i G_i + \overline{b}_i$$

The abrupt intensity changes in the guiding image G are still largely preserved in the output image O. The guided filter is a computationally efficient, edge-preserving operator which avoids the gradient reversal artefacts of the bilateral filter.

In Iterative guided filtering: In such a scheme the result G^{t+1} of the t-*th* iteration is obtained from the joint bilateral filtering of the input image I using the result G^t of the previous iteration step:

$$G_i^{t+1} = \frac{1}{K_i} \sum_{j \in \omega} I_j \cdot f(\parallel i - j \parallel) \cdot g(\parallel G_i^t - G_j^t \parallel)$$

Note that the initial guidance image G^1 can simply be a constant (e.g. zero) valued image since it updates to the Gaussian filtered input image in the first iteration step.

Proposed Method:

- Iterative guided filtering is applied to decompose the source images into base layers (representing large scale variations) and detail layers (containing small scale variations).
- Frequency-tuned filtering is used to generate saliency maps for the source images.
- Binary weighting maps are computed as the pixelwise maximum of the individual source saliency maps.
- Guided filtering is applied to each binary weighting map with its corresponding source as the guidance image to reduce noise and to restore spatial consistency.
- The fused image is computed as a weighted recombination of the individual source detail layers.

Visual saliency refers to the physical, bottom-up distinctness of image details. It is a relative property that depends on the degree to which a detail is visually distinct from its background. **Since saliency quantifies the relative visual importance of image details saliency maps are frequently used in the weighted recombination phase of multiscale image fusion schemes**. Frequency tuned filtering computes bottom-up saliency as local multiscale luminance contrast. The saliency map S for an image I is computed as

$$S(x, y) = \parallel I_\mu - I_f(x, y) \parallel$$

where

I_μ is the arithmetic mean image feature vector

I_f represents a Gaussian blurred version of the original image, using a 5 * 5 separable binomial kernel

$\| \cdot \|$ is the L_2 norm(Euclidian distance), and x, y are the pixel coordinates.

We compute saliency using frequency tuned filtering since a recent and extensive evaluation study comparing 13 state-of-the-art saliency models found that the output of this simple saliency model correlates more strongly with human visual perception than the output produced by any of the other available models.

Binary weight maps BW_{X_i} and BW_{Y_i} are then computed by taking the pixelwise maximum of corresponding saliency maps S_{X_i} and S_{Y_i}:

$$BW_{X_i}(x,y) = \begin{cases} 1 & \text{if } S_{X_i}(x,y) > S_{Y_i}(x,y) \\ 0 & \text{otherwise} \end{cases}$$

$$BW_{Y_i}(x,y) = \begin{cases} 1 & \text{if } S_{Y_i}(x,y) > S_{X_i}(x,y) \\ 0 & \text{otherwise} \end{cases}$$

The resulting binary weight maps are noisy and typically not well aligned with object boundaries, which may give rise to artefacts in the final fused image. Spatial consistency is therefore restored through guided filtering (GF) of these binary weight maps with the corresponding source layers as guidance images

$$W_{X_i} = GF(BW_{X_i}, X_i)$$
$$W_{Y_i} = GF(BW_{Y_i}, Y_i)$$

Fused detail layers are then computed as the normalized weighted mean of the corresponding source detail layers:

$$dF_i = \frac{W_{X_i} \cdot dX_i + W_{Y_i} \cdot dY_i}{W_{X_i} + W_{Y_i}}$$

The fused image F is finally obtained by adding the fused detail layers to the average value of the lowest resolution source layers:

$$F = \frac{X_3 + Y_3}{2} + \sum_{i=0}^{2} dF_i$$

By using guided filtering both in the decomposition stage and in the recombination stage, this proposed fusion scheme optimally benefits from both the multi-scale edge-preserving characteristics (in the iterative framework) and the structure restoring capabilities (through guidance by the original source images) of the guided filter. The method is easy to implement and computationally efficient.

3.2.3 Conclusion

We propose a multiscale image fusion scheme based on guided filtering. Iterative guided filtering is used to decompose the source images into base and detail layers. Initial binary weighting maps are computed as the pixelwise maximum of the individual

source saliency maps, obtained from frequency tuned filtering. Spatially consistent and smooth weighting maps are then obtained through guided filtering of the binary weighting maps with their corresponding source layers as guidance images. Saliency weighted recombination of the individual source detail layers and the mean of the lowest resolution source layers finally yields the fused image. The proposed multi-scale image fusion scheme achieves spatial consistency by using guided filtering both at the decomposition and at the recombination stage of the multiscale fusion process. Application to multiband visual (intensified) and thermal infrared imagery demonstrates that the proposed method obtains state-of-the-art performance for the fusion of multispectral nightvision images. The method has a simple implementation and is computationally efficient.

3.3 Image Fusion With Guided Filtering

3.3.1 Abstract

A fast and effective image fusion method is proposed for creating a highly informative fused image through merging multiple images. The proposed method is based on a two-scale decomposition of an image into a base layer containing large scale variations in intensity, and a detail layer capturing small scale details. A novel guided filtering-based weighted average technique is proposed to make full use of spatial consistency for fusion of the base and detail layers. Experimental results demonstrate that the proposed method can obtain state-of-the-art performance for fusion of multispectral, multifocus, multimodal, and multiexposure images.

3.3.2 Contents

Guided Filter

The filtering output O is linear transformation of the guidance image I in a local window ω_k centered at pixel k:

$$O_i = a_k I_i + b_k \qquad \forall i \in \omega_k$$

where ω_k is a square window of size $(2r+1) \times (2r+1)$. The linear coefficients a_k and b_k are contant in ω_k by minimizing the squared difference between the output image O and the input iamge P:

$$E(a_k, b_k) = \sum_{i \in \omega_k} \left((a_k I_i + b_k - P_i) + \epsilon a_k^2 \right)$$

where ϵ is a regularization parameter given by the user. The coefficients can be directly solved by linear regression as follows:

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i P_i - \mu_k \overline{P}_k}{\delta_k + \epsilon}$$
$$b_k = \overline{P}_k - a_k \mu_k$$

where μ_k and δ_k are the mean and variance of I in ω_k respectively. $|\omega|$ is the number of pixels in ω_k, and $\overline{P}_k$ is the mean of P in ω_k. Then the output image can be calculated according to above equation.

$$O_i = \overline{a}_i I_i + \overline{b}_i$$

where $\overline{a}_i = \frac{1}{|\omega|}\sum_{k\in\omega_i} a_k, \overline{b}_i = \frac{1}{|\omega|}\sum_{k\in\omega_i} b_k$.

The color image situation, the a_i and other calculators become vector version. See [8].

3.3.3 Fusion Frame

See figure 3.1.

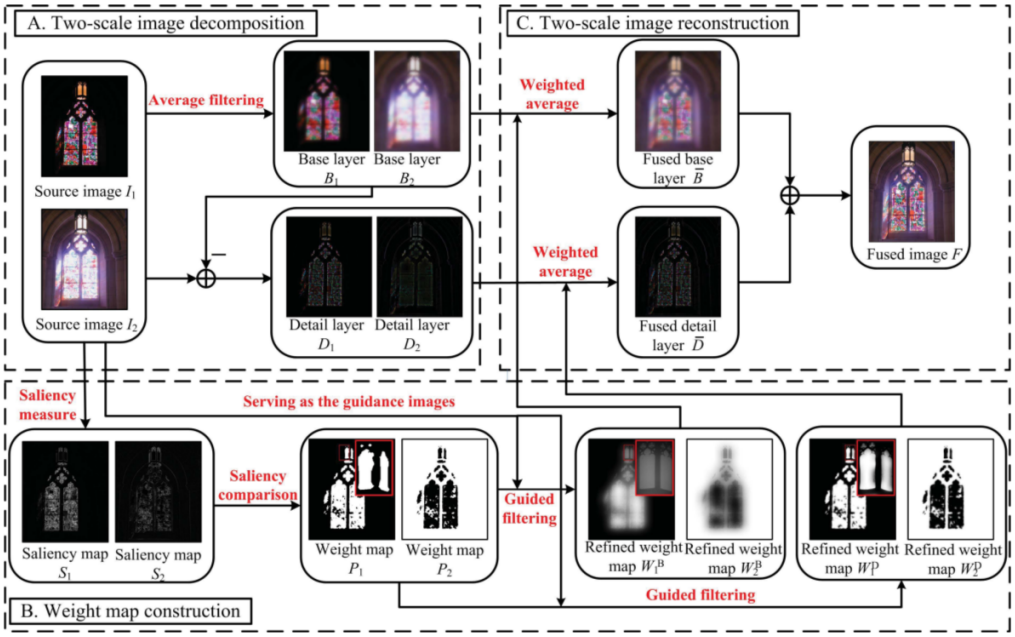


Fig 3.1. Schematic diagram of the proposed image fusion method based on guided filtering.

3.3.4 Conclusion

Chapter 4

Saliency Detection

This chapter includes papers about saliency detection.

4.1 Frequency-tuned Salient Region Detection

4.1.1 Abstract

In this paper, we introduce a method for salient region detection that outputs full resolution saliency maps with well-defined boundaries of salient objects. These boundaries are preserved by retaining substantially more frequency content from the original image than other existing techniques. Our method exploits fea- tures of color and luminance, is simple to implement, and is computationally efficient. We compare our algorithm to five state-of-the-art salient region detection methods with a frequency domain analysis, ground truth, and a salient object segmentation application. Our method outperforms the five algorithms both on the ground-truth evaluation and on the segmentation task by achieving both higher precision and better recall.

4.1.2 Contents

Related work

Saliency estimation methods can broadly be classified as:

- biologically based
- purely computational
- combination of above two approaches

Itti base their method on the biologically plausible architecture proposed by Koch and Ullman. They determine center-surround contrast using a **Difference of Gaussians** (DoG). Frintrop presetn a method inspired by Itti's method, but they compute **center-surround differences** with square filters and use integral images to speed up the calculations.

Other methods are purely computational and are not based on biological vision principles. Ma and Zhang and Achanta et al. estimate saliency us- ing center-surround feature

distances. Hu et al. es- timate saliency by applying heuristic measures on initial saliency measures obtained by histogram thresholding of feature maps. Gao and Vasconcelos maximize the mu- tual information between the feature distributions of center and surround regions in an image, while Hou and Zhang rely on frequency domain processing.

The third category of methods are those that incorporate ideas that are partly based on biological models and partly on computational ones. For instance, Harel et al. create feature maps using Itti' s method but perform their normal- ization using a graph based approach. Other methods use a computational approach like maximization of information that represents a biologically plausible model of saliency detection.

Limitations

The saliency maps generated by most methods have low resolutioni. Itti's method produces saliency maps that are just $1/256^{th}$

Frequency-tuned Saliency Detection

DoG

DoG : Difference of Gaussians. DoG filter is widely used in edge detecion since it closely and efficiently approximates the Laplacian of Gaussian (LoG) filter, cited as the most satisfactory operator for detecting intensity changes when the standard deviations of the Gaussians are in the ratio $1 : 1.6$. The DoG has also been used for interest point detection and saliency detection. The DoG filter is given by :

$$\begin{aligned} DoG(x,y) &= \frac{1}{2\pi}\left[\frac{1}{\delta_1^2}e^{-\frac{x^2+y^2}{2\delta_1^2}} - \frac{1}{\delta_2^2}e^{-\frac{x^2+y^2}{2\delta_2^2}}\right] \\ &= G(x,y,\delta_1) - G(x,y,\delta_2) \end{aligned} \tag{4.1}$$

where δ_1 and δ_2 are the standard deviatioins of the Gaussian $(\delta_1 > \delta_2)$.

A DoG filter is a simple band-pass filter whose passband width is controlled by the ratio $\delta_1 : \delta_2$.

4.1.3 Conclusion

Chapter 5

Semantic SLAM

5.1 DeLS-3D: Deep Localization and Segmentation with a 2D Semantic Map[18]

5.1.1 Abstract

Sensor fusion scheme: Integrates camera videos, Motion sensors (GPS/IMU), and a 3D semantic map.

步骤：

- Initial Coarse camera pose obtained from consumer-grade GPS/IMU
- A label map can be rendered from the 3D semantic map.
- Rendered label map and the RGB image are jointly fed into a pose CNN, yielding a corrected camera pose.
- A multi-layer RNN is further deployed improve the pose accuracy
- Based on pose from RNN, a new label map is rendered
- New label map and the RGB image is fed into a segment CNN which produces per-pixel sematnic label.

从结果可以看出，Scene Parsing 以及姿态估计两者可以相互改善，从而提高系统的鲁棒性以及精确度。

5.1.2 Introduction

在 Localization 中，传统的做法是基于特征匹配来做，但这样的坏处是，如果纹理信息较少，那么系统就不稳定，会出错。一种改进办法是利用深度神经网络提取特征。实际道路中包含大量的相似场景以及重复结构，所以前者实用性较差。

在 Scene Parsing 中，深度神经网络用的很多，最好的基于 (FCN + ResNet) 的途径。在视频中，可以借助光流信息来提高计算速度以及时间连续性。对于静态场景，可以借助 SfM 技术来联合 Parse 以及 Reconstruction. 但这些方法十分耗时。

相机的姿态信息可以帮助 3D 语义地图与 2D 标签地图之间的像素对应。反过来，场景语义又会帮助姿态估计。

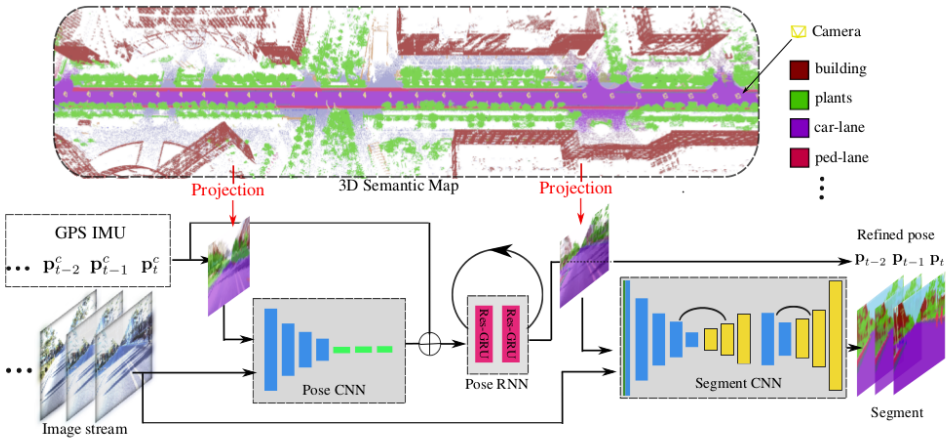


Figure 1: System overview. The black arrows show the testing process, and red arrows indicate the rendering (projection) operation in training and inference. The yellow frustum shows the location of cameras inside the 3D map. The input of our system contains a sequence of images and corresponding GPS/IMU signals. The outputs are the semantically segmented images, each with its refined camera pose.

Fig 5.1. DeLS Framwork

5.1.3 Framework

总的工作流程，如图5.1所示：

从图中可以看出，RGB Images 以及根据 GPS/IMU 获得的 semantic label map 被输入到 Pose CNN，然后输出的 Pose 信息输入到 Pose RNN 来对 Pose 进一步提高，**这里用 RNN 来获得前后帧的一致性**！然后在利用新得到的 Pose 来获取更精确的 Semantic Label Map，最后，这个 label Map 以及 RGB 图像输入到 Segment CNN 来进一步提高语义地图的精度。这里标签地图被用于提高语义地图的空间精度以及时间一致性。

网络的训练是基于非常精确地相机姿态以及语义分割，所以可以采用监督学习。

5.1.4 Related Work

- Camera Pose Estimation
  - PnP

    在大的范围内，可能需要提供姿态的先验信息。但对于城市环境中存在大量的 Points，这种方法不适用，且不适用于纹理少、结构重复、以及重叠的区域。
  - Deep learned features

    PoseNet, LSTM-PoseNet, Bi-Directional LSTM, or Kalman filter LSTM. 但实际中由于存在植被等重复性的场景，所以十分有必要加入 GPS/IMU 等信息来获得鲁棒的定位结果。而在这里，我们采用结合 RGB 图像与 Online Rendered label map 的方式来提供更好的结果。

    这里问题来了，首先是 label map 的精度如何？其次，随着时间的变化，label map 与实际 RGB 图像可能完全不同，如季节改变了，这应该如何？

- Scene Parsing

FCN, Multi-scale context module with dilated convolution, Pooling, CRF, or Spatial RNN with hundreds of layers. 这些方法都太耗时了。

一些方法是利用小模型或者模型压缩来加速，但会降低精度。

当输入是 Video 时，需要考虑时空信息。当前，存在利用光流来帮助 label 以及 semantic 在相邻帧之间的传递。借助 3D 信息以及相机姿态把相邻帧联系起来，可以更好的处理静态背景下的表示。具体的，是使用 DeMoN 来提高推理效率。

- Joint 2D-3D for video parsing

  CNN-SLAM 把传统的 3D 重建模块替换为深度预测网络，且借助语义分割网络来获取场景语义。同样比较耗时、仅适合静态背景，重建效果也不好。

5.1.5 Dataset

- Data collection

  Mobile LIDAR to collect point clouds of the static 3D map. Cameras' resolution: 2018 * 2432.

- 2D and 3D Labeling

  - Over-segment the point clouds into point clusters based on spatial distances and normal directions, then label each point cluster manually.
  - Prune out the points of static background, label the remaining points of the objects.
  - After 3D-2D projection, only moving object remain unlabeled.

- 使用图形学中的 *Splatting techniques* 来优化未被标签的像素。

5.1.6 Localizing camera and Scene Parsing

Render a label map from a camera pose

初始的相机的姿态来自于 GPS/IMU 等传感器。

6-DOF 相机姿态：$\mathbf{b} = [\mathbf{q}, \mathbf{t}] \in SE(3)$. 其中 $\mathbf{q} \in SO(3)$ 是四元数表示的旋转，$\mathbf{t} \in \mathbb{R}^3$ 表示 Translation。

在由 Point 经 Spaltting 获取其面时，面积大小 s_c 根据 Point 所属的类别来决定，且与该类别与相机的平均距离的比例有关。

$$s_c \propto \frac{1}{|\mathcal{P}_c|} \sum_{x \in \mathcal{P}_c} \min_{\mathbf{t} \in \tau} d(x, t)$$

其中，P_c 是属于类别 c 的 3D 点云，τ 是精确地相机姿态。如果面积过大，则会出现 Dilated edges, 而如果面积过小，则会形成 Holes。

Camera Localization rectification with road prior

CNN-GRU pose Network Architecture

文中的 Pose Network 包含一个 Pose CNN 以及一个 Pose GRU-RNN。其中 Pose CNN 的输入是 RGB 图像 I 以及一个标签地图 L。输出是一个 7 维的向量，表示输入图像 I 与输入标签地图 L(由较粗糙的姿态 $\mathbf{p}_i^c$ 得到) 之间的位姿关系，从而得到一个在 3D Map 中更精确的姿态：$\mathbf{P}_i = \mathbf{p}_i^c + \hat{\mathbf{p}}_i$。

CNN 结构借鉴 DeMoN, 利用打的 Kernel Size 来获取更大的内容，同时保证运行效率，减少参数量。

由于输入的是图像流，为了保证时间一致性，所以在 Pose CNN 之后又加上一个多层的 GRU 网络，且该网络具有 Residual Connection 的连接结构。结果表明，RNN 相比于卡尔曼滤波可以获得更好的运动估计。

Pose Loss

类似于 PoseNet 的选择，使用 Geometric Matching Loss 来训练。

Video Parsing with Pose Guidance

上一步得到的 Pose 估计，不是完美的，因为存在 light poles 存在。由于反光，很多点消失了。此外，由于存在动态物体，这些物体可能在原来的标签地图中不存在，所以这些区域可能发生错误。因此，利用额外的一个 Segment CNN 来出来这些问题。且利用标签地图来指导分割过程。

Segment Network Architecture

首先基于 RGB 图像对该网络训练，然后加入标签地图数据进行微调 (Fine Tune). 具体结构如图5.2所示。

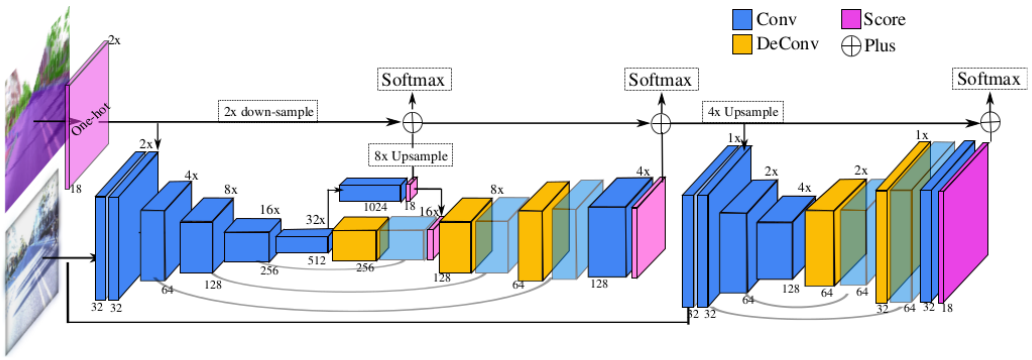


Figure 4: Architecture of the segment CNN with rendered label map as a segmentation priori. At bottom of each convolutional block, we show the filter size, and at top we mark the downsample rates of each block w.r.t. the input image size. The softmax' text box indicates the places a loss is calculated. Details are in Sec. 4.3.

Fig 5.2. Segment Network in DeLS

需要注意的是，当标签地图加入框架时，需要经过编码，即每一个像素经 One-hot 操作得到一个 32 维的 Feature Representation。然后得到的 32 维特征加入到 RGB 图像的第一层卷积输出中，且该层的 Kernel 数也是 32 个，从而平衡了两种数据的通道数 (Channel Number)。

5.1.7 Experiment

- Adopt OpenGL to efficiently render a label map with the z-buffer handling.

- Implement all the networks by adopting the MXNet platform.
- 使用 RNN 可以提高 Pose 的精度，也可以提高 Segment 的精度，尤其对于纤细的物体。

5.1.8 Conclusion

基于已有的 3D 语义地图以及视频数据，实现相机的姿态、场景语义任务的实现。算法融合了多种传感器信息。实验表明，相机位姿估计与场景语义两类任务可以相互促进、提高。

5.2 PAD-Net: Multi-Task Guided Prediction-and-Distillation Network for Simultaneous Depth and Scene Parsing [21]

5.2.1 Abstract

利用同一个网络，完成深度估计与场景解析两个任务。具体来说，通过神经网络学习一系列的中间辅助任务 (Intermediate Auxiliary Tasks)，然后基于中间任务的输出，作为多模式数据 (Multi-modal input) 输入到下一层网络中，完成最终的深度估计以及场景解析两个任务。

其中，一系列的中间任务包括低层任务和高层任务。低层任务包括：Surface Normal, Contour; 高层任务包括：Depth Estimation, Scene Parsing.

5.2.2 Analysis

Effect of Direct Multi-task Learning

It can be observed that on NYUD-v2, the Front-end + DE + SP slightly outperforms the Front-end + DE, while on Cityscapes, the performance of Front-end + DE + SP is even decreased, which means that using a direct multi-task learning as traditional is probably not an effective means to facilitate each other the performance of different tasks. (DE: Depth Estimation, SP: Scene Parsing)

Effect of Multi-modal Distillation

这种 Multi-Modal Distillation 对结果十分有效。且：

> By using the attention PAD-Net (Distillation C + SP) guided scheme, the performance of the module C is further improved over the module B.

Importance of Intermediate Supervision and Tasks

测试了选择不同的中间任务类型，如:(Multi-Task Deep Network)MTDN + inp2(depth + semantic map)，MTDN+3inp3(depth + semantic + surface normal), MTDN + all(depth + semantic + surface + contour).

其中 MTDN + all 比 MTDN + 0-3 都好。

5.3 RNN for Learning Dense Depth and Ego-Motion from Video

参考文献：[19]
时间：2018 年 05 月 19 日

5.3.1 Abstract

现在基于学习的单目深度估计，在 Unseen Dataset 上泛化较差，可以利用连续两帧之间的特征匹配来解决。本文提出了基于 RNN 的多目视觉深度估计以及运动估计。结果表明，在远距离上的深度估计，表现很好。本文方法可使用 both static and deformable scenes with either constant or inconsistent light conditions.

5.3.2 Introduction & Related Works

由于 CNN 只能捕获单帧内部的空间特征，所以即使输入两帧图像，实际效果也比较差。相关的 SLAM 框架有：

- ORB-SLAM, DSO: 稀疏 SLAM
- LSD-SLAM: semi-dense SLAM，利用光强来实现跟踪以及构建地图
- DTAM: dense SLAM

上述框架仅适用于静态、光照恒定、足够的 camera motion baseline 的情景。
进来，CNN 可被用于 SLAM 中的以下部分：

- Correspondence matching
- Camera Pose estimation
- Stereo

输出包括：

- Depth maps
- point clouds and voxels

CNN 的优势在于可以适用于纹理较少、表面覆盖、很细等场景下，这些场景对纯使用几何技术来说很难实现。

Newcombe 研究表明，多目视觉有助于提高深度估计的精度，而本文作者认为采用相邻几帧可以实现类似的效果。

有作者利用一个非监督生成模型来学习复杂的 3D 到 2D 的投影。但这些手段不太适用于 Deformable Objects.

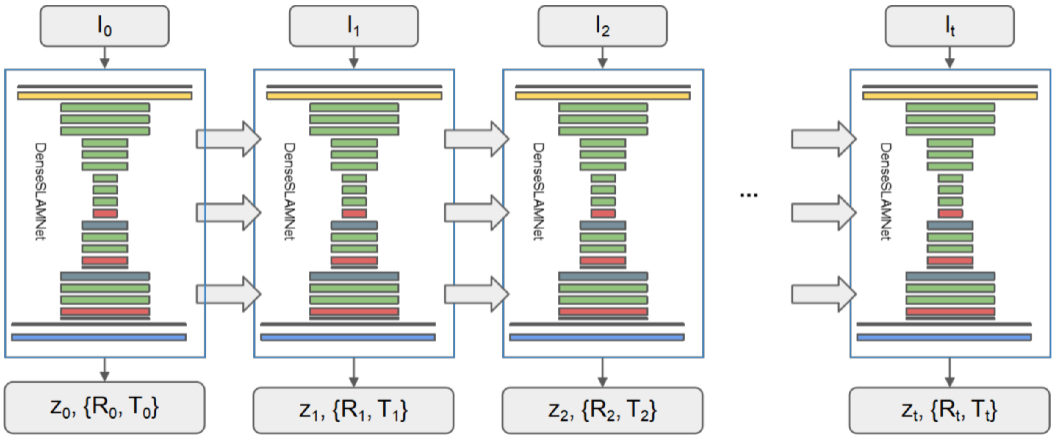


Fig 5.3. Dense SLAM 框架

5.3.3 Network Architecture

从图5.3可以看出，它接受当前 RGB 图像 I_t 以及来自迁移级的隐藏状态 h_{t-1}。h_{t-1} 通过 LSTM 单元进行内部转移。网络的输出是稠密地图 z_t 以及相机的姿态 R_t, T_t 等。有没在每一时刻，只输入一帧图像，且输出该帧对应的深度以及姿态，所以相比于 CNN 具有更高的灵活性。

具体的每一块的细节结构如下：

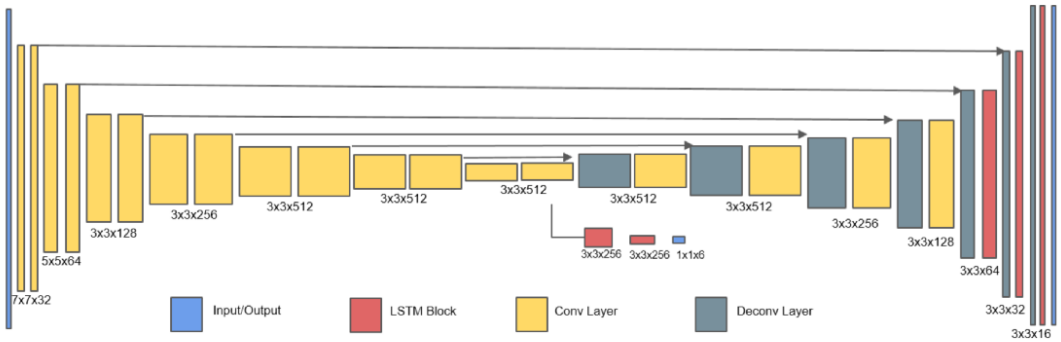


Fig. 4: (Best viewed in color) Our network architecture at a single time step. We use the DispNet architecture. The width and height of each rectangular block indicates the size and the number of the feature map at that layer. Each increase and decrease of size represents a change factor of 2. The first convolutional layer has 32 feature maps. The kernel size for all convolution layers is 3, except for the first two convolution layers, which are 7 and 5, respectively.

Fig 5.4. Dense SLAM 中每一级的细节框架

网络结构的另一个参数是时间窗口的大小 N。本文中 $N = 10$，也就是说，图5.4中的结构被重复十次。

5.3.4 Training

During training, we feed frames to the network and compute losses from all frames in a temporal window. However, there is no input length constraint at test time.

Loss Function

由三部分组成：

- A Point-wise depth loss

$$L_{depth} = \sum_{t}^{N} \sum_{i,j} \left\| \xi_t(i,j) - \hat{\xi}_t(i,j) \right\|_{L1}$$

其中，i, j 为索引，t 为 time step. 使用 $L1 - Norm$ 的原因是，它对噪声更鲁棒。

- a camera pose loss

Use the Euler angle R 和平移向量 T。

$$L_{rot} = \sum_{t} \|r_t - \hat{r}_t\|_2$$
$$L_{trans} = \sum_{t} \left\|t_t - \hat{t}_t\right\|_2$$

- a scale-invariant gradient loss

为了保证深度图的 Smoothness 和 Sharpness, 所以增加了这个 loss。具体如下：

$$L_{grad} = \sum_{t} \sum_{h \in \{1,2,4,8,16\}} \sum_{i,j} \|g_{h,t}(i,j) - \hat{g}_{i,j}(i,j)\|_2$$

其中，h 代表不同的尺度。$g_{h,t}$is a scale-normalized, discretized measurement of the local changes of ξ_t。定义如下：

$$g_{h,t} = \left(\frac{\xi_t(i+h,j) - \xi_t(i,j)}{|\xi_t(i+h,j) + \xi_t(i,j)|}, \frac{\xi_t(i,j+h) - \xi_t(i,j)}{|\xi_t(i,j+h) + \xi_t(i,j)|} \right)$$

L_{grad}emphasizes the depth discontinuities, such as occlusion boundaries and sharp edges, as well as the smoothness in homogeneous regions. This property encourages the estimated depth map to preserve more details and reduce noise. Therefore, we put highly weight on this component of the loss. 这一项在所有的 Loss 中比重较大。

最终的 Loss 由上面几项的和构成：

$$L = \alpha * L_{depth} + \beta * L_{pose} + \gamma * Lgrad$$

其中，α, β, γ 是系数，由经验决定。

注意，下面解释的是为什么使用 Disparity 而不是直接使用深度。

Caution! Disparity, the reciprocal of depth (深度的倒数), $\xi = \frac{1}{z}$ as our direct estimation because it can represent points at infinity and account for the localization uncertainty of points at increasing distance.

Different datasets have different camera intrinsic parameters, so we explicitly crop and resize images to ensure uniform intrinsic parameters. This step assures that the non-linear mapping between color and depth is consistent across all training datasets.

5.3.5 Experiments

We evaluate DenseSLAMNet using five error metrics。具体可以参考文章。

- Sc-inv
- Abs-rel, Abs-inv
- RMSE, RMSE-log

5.3.6 Ablation Studies

比较了三种不同的网络结构。

1. CNN-SINGLE, 去掉图5.4中 LSTM 单元
2. CNN-STACK, 使用与 CNN-SINGLE 相同的网络，但是输入 stack of ten images as input.

实验结果表明，LSTM 可有效保留时间域的信息，从而深度预测的更好。

Conclusions

引入了 LSTM 单元。Our method effectively utilizes the temporal relationships between neighboring frames through LSTM units, which we show is more effective than simply stack- ing multiple frames together as input.

比几乎所有的单帧 CNN-based 都好。Our DenseSLAMNet outperformed nearly all of the state-of-the-art CNN-based, single-frame depth estimation methods on both indoor and outdoor scenes and showed better generalization ability.

5.4 DA-RNN: Semantic Mapping with Data Associated Recurrent Neural Networks

参考文献：[20]

评语看得出来，RNN 在帮助建立相邻帧之间的一致性方面具有很大的优势。

本文利用可以实现 Data Associate 的 RNN 产生 Semantic Label。然后作用于 KinectFusion，来插入语义信息。

所以本文利用了 RNN 与 KinectFusion 来实现语义地图的构建！

5.4.1 Related Works

本文提出的 DA-RU (Data Associated Recurrent Unit) 可以作为一个单独的模块加入到已有的 CNN 结构中！

5.4.2 Methods

需要注意到是，本文采用了 DA-RNN 与 KinectFusion 合作的方式来产生最终的 Semantic Mapping，而文章采用的则是 ORB-SLAM 与 CNN 结合，他们有一些共同的结构，如 Data Associate，虽然我现在还不明白这个 DA 具体得怎么实现！?

5.4.3 Experiments

实验表明，针对不同的实验输入场景，RGB 图像和 Depth 图像可能发挥不同的作用，有一些时候甚至让结果更差，而本文采用 Contatenated 把来自于 RGB 和 Depth 的 Feature 进行组合起来，在这里，是否可以采取 Attention 或者 [21] 中采用的 Knowledge Distillation 的方式进行多模态数据融合呢！?

此外，实验中也发现，有时候 RGB 图像的 Color 会影响实验结果，所以，是否可以利用一些其它的不那么 Confusing 的数据呢，比如 Shape 等，这也类似于 [21] 中多任务中的 Contour 类似吧！

5.4.4 Conclusion

- ⋆ 将来的可以借助 Optical Flow 来生成图像帧之间的 Data Associate，来代替 KinectFusion 的作用！

5.5 SemanticFusion: Dense 3D Semantic Mapping with CNNs

参考文献：[11]
主要框架，用到了 ElasticFusion 和 CNN 来产生稠密的 Semantic Mapping.

5.5.1 Introduction & Related Works

本文的算法，利用 SLAM 来实现 2D Frame 与 3D Map 之间的匹配 (Corresponding)。通过这种方式，可以实现把 CNN 的语义预测以概率的方式融合到 Dense semantically annotated map.

为什么选择 ElasticFusion 呢，是因为这种算法是 surfel-based surface representation，可以支持每一帧的语义融合。

比较重要的一点是，通过实验说明，引入 SLAM 甚至可以提高单帧图像的语义分割效果。尤其在存在 wide viewpoint variation，可以帮助解决单目 2D Semantic 中的一些 Ambiguations.

之前存在方法使用 Dense CRF 来获得语义地图。

本文的主要不同是：利用 CNN 来产生语义分割，然后是在线以 Incremental 的方式生成 Semantic Map。即新来一帧就生成一帧的语义地图。

5.5.2 Method

系统由三部分组成：

- A real-tiem SLAM system ElasticFusion
  用于提供帧间的关联、全局一致的地图
- CNN
  产生语义分割
- Bayesian update scheme
  基于 SLAM 提供的 Pose 关系、CNN 提供的每个像素的 Class Probability 对 Map 中的每一个 surfel 的 Class Probability 进行更新！

此外，作者还尝试了用 CRF 来利用 SLAM 提供的 Geometry 帮助语义分割。
非常粗糙的流程图如下图所示：

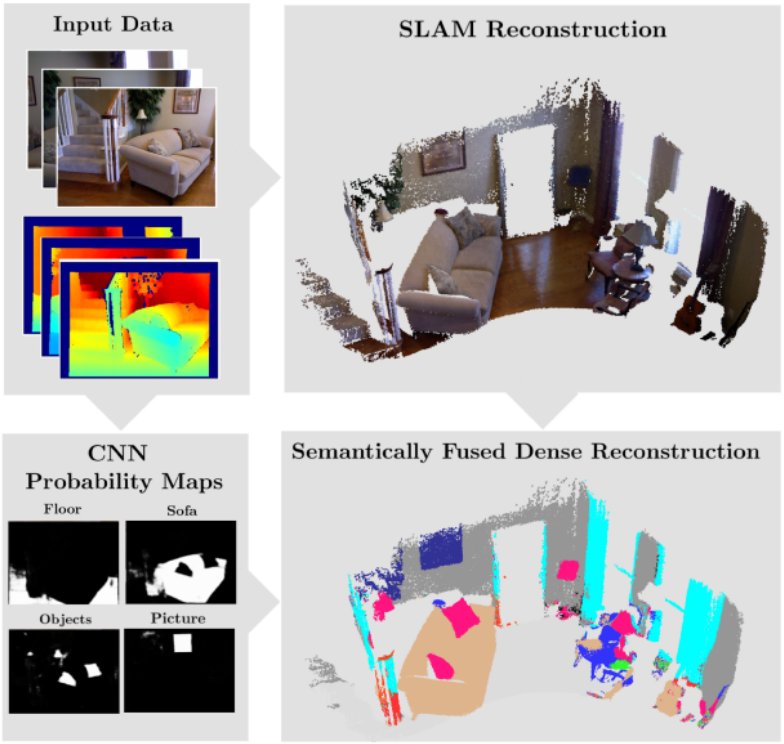


Fig. 2: **An overview of our pipeline**: Input images are used to produce a SLAM map, and a set of probability prediction maps (here only four are shown). These maps are fused into the final dense semantic map via Bayesian updates.

Fig 5.5. 粗糙的数据流图

SLAM Mapping

就是使用 ElasticFusion.

CNN Architecture

使用 Deconvolutional Semantic Segmentation network，与 FCN 同年提出的另一个分割网络。

在本实验中，输入由 RGB 变为 RGBD，多了一个 Channel. 然而，深度相关的训练数据比较少，为了提高利用率，作者利用其它三个输入的光强的平均值对 Depth Filter 进行初始化。

对输入数据，作者还进行了 Scaling，RGB 是用 Bilinear Interpolation, Depth 数据是 Nearest enighbour。

Incremental Semantic Label Fusion

在生成的地图 $\mathcal{M}$ 中，每一个 Surfel(ElasticFusion 的结果) 不仅代表了 Location、Normal 等信息，还包含一个类别 ($\mathcal{L}$) 的分布。

输入数据为 $\mathbf{I}_k$ 表示 RGB 图像与 Depth 等。

在对每一个 Surfel 进行类别分布概率更新时，采用的 Recursive Bayesian update，这个算法就是 Probability Robotics 里面最基础的算法，也就是分为 Prediction 和 Correlation 两个步骤。

$$P(l_i|I_{1,\dots,k}) = \frac{1}{Z}P(l_i|I_{1,\dots,k-1})P(O_{u(s,k)=l_i|I_k})$$

其中，第一个 P 表示根据过去的信息的预测，在这里也就是已有的 $\mathcal{M}$ 里面的一个 surfel 的 class probability，如果，是一个全新的 Surfel，则初始化它为均匀分布，因此这样熵最大；而第二个 P 表示来自 CNN 输入是 I_k 时输出的 class probability，然后对已有地图中的 surfel 的 class probability 进行 Correlation!

Map Regularisation

作者尝试了利用 CRF 来借助 map geometry 对预测的 Semantic surfel 进行 Regularise。

在这里，把每一个 Surfel 当做 CRF 的一个 Node。

这一部分，参考论文吧。

5.5.3 Experiments

本文用到的 Semantic 分割的方法 (CNN + SLAM) 与普通的单 CNN 相比，具有很大的优势。本文的方法是，得到 3D 的 Semantic Map 后，重投影到 2D 图像中。

时间方面，SLAM 需要 29.3ms，CNN 的前向传播用了 51.2ms，Bayesian 更新用了 41.1ms。

5.5.4 总结

未来也还是 SLAM 与语义分割相互促进吧，达到 Semantic SLAM。

5.6 Meaningful Maps with Object-Oriented Semantic Mapping

参考文献：[15]
主要的框架是，利用 CNN 与 ORB-SLAM2 来实现 Semantic Mapping。**但是都用到了 Data Association 的操作**！而且本文还是 Object-oriented (Instance level)！

5.6.1 Introduction & Related Works

重要：
与已有的方式不同的是，本文的算法不仅分割独立的 3D Point，也就是把语义信息投影到 3D Point 中，而是投影到 3D Structure。这样会更有利于场景理解。
已有的 SLAM 算法得到的结果都是一些几何上的概念，比如: 点、面、表面等。另一方面，为了实现与环境交互，必须基于语义地图才行。

Semantic Mapping

语义信息与地图构建可能属于两个不同的过程得到。
有一些算法用到了 HMM 或 Dense CRF 等。

Object Detection and Semantic Segmentation

FCN 的缺点是，形成的语义地图一般缺少 Notion of independent object instances. 如果只是 Pixel-level 的 labels 不能辨别有重叠时物体的身份。
也有一些 Instance-level 的分割算法，但精度、速度都有待提高。

5.6.2 Object Oriented Semantic Mapping

算法的主要步骤：
算法使用 RGB-D 作为输入，算法是用 ORB-SLAM2 提供相机的姿态、地图等。

Object Detection for Semantic Mapping

本文使用 SSD 来完成 Object 的 Location 和 Recognition。

3D Segmentation

由于需要非常精确的图像分割，所以本文利用 Depth 图来帮助分割。所以需要对 Depth 进行分割的算法，本文采用了文章 [3][13] 等人的算法。也是涉及到基于图的分割的过程。

Data Association

重点。
当完成了把 3D Point 投影到识别的物体后，数据关联要做的事：判断检测到的物体是否已经在已经构建的地图中存在了呢，如果不存在的话，就需要新增这个物体。

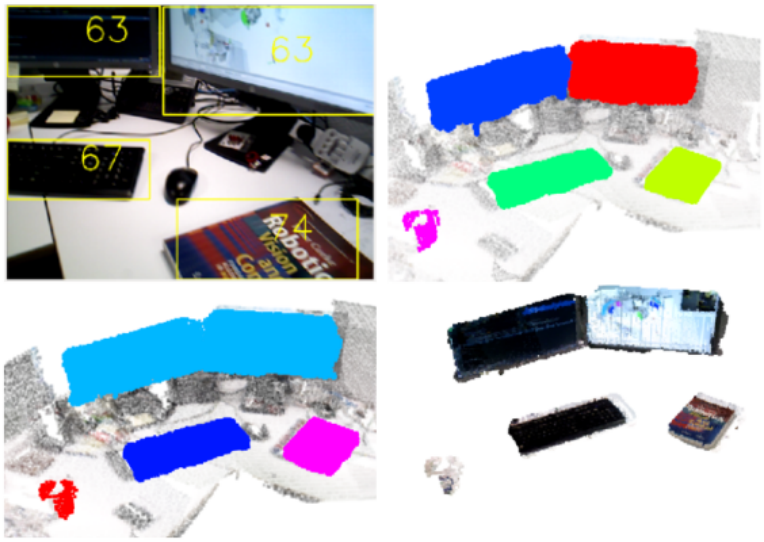


Fig. 2: Illustration of key steps in our proposed approach: (top-left) SSD [23] generates object proposals consisting of bounding boxes, class labels, and confidence scores. (top-right) Our unsupervised 3D segmentation algorithm creates a 3D point cloud segment for each of these objects detected in the current RGB-D frame. (bottom row) We obtain a map that contains semantically meaningful entities: objects that carry a semantic label, confidence, as well as geometric information. The semantic label is color coded in the bottom left image. light blue: monitor, pink: book, red: cup, dark blue: keyboard.

Fig 5.6. 算法的几个主要步骤

通过一个二阶的流程来实现：

- 对于每一个检测到的 Object，根据点云的欧式距离，来选择一系列的 Landmarks(已经检测到并在地图里面已经有的 Object)
- 对 Object 和 Point Cloud of landmark 进行最近邻搜索，使用了 k-d tree 来提高效率，其实这一步也就是判断当前图像检测到的 Object 与已有的地图中的 Landmark(Object) 是否相匹配。

在第二步中，如果多余 50 % 的 Points 的都距离小于 2cm 的话，就说明这个检测到的 Object 已经存在了。

这个也就是把 CNN 的分割结果与地图中的 Object 进行关联起来，并用颜色表示 Map 中 Object 的类别。

Object Model Update

地图中的目标保存：

- 与目标相关联的分割的 3D 点云
- ORB-SLAM 中因子图的各个位姿的索引
- 有 SSD 提供的各个目标的置信度

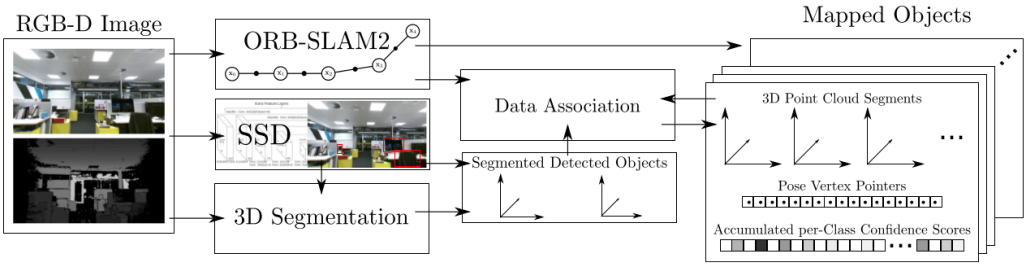


Fig. 3: Overview of our semantic mapping system. While ORB-SLAM2 performs camera localisation and mapping on every RGB-D frame, SSD [23] detects objects in every RGB *keframe*. Our own adapted 3D unsupervised segmentation approach assigns a 3D point cloud segment to every detection. Data association based on an ICP-like matching score decides to either create a new object in the map or associate the detection with an existing one. In the latter case, the 3D model of the map object is extended with the newly detected 3D structure. Every object stores 3D point cloud segments, pointers into the pose graph of ORB-SLAM and per-class confidence scores that are updated on the fly whenever new observations are available.

Fig 5.7. Semantic Mapping 系统概览

上图中可以看出，

评语：在上一篇 SematicFusion 中，采用的 Recursive Bayesian 的更新规则来完成地图更新的！看来，这个是 CNN 与传统的 SLAM 框架结合的时候的一个需要解决的问题，那就是如何把新来的物体与已有的地图中的物体相互关联起来，并更新！

Map Generation

通过保存 Keyframe 中物体的 3D poin clouds、每一个物体的 3D 分割以及执行位姿图中的一个指针。

5.6.3 总结

未来可以的发展方向：

- 语义 Landmark 如何提高 SLAM 的精度, 从而实现 Semantic SLAM
- 把稀疏的语义地图改进为稠密地图

5.7 LiteFlowNet: A Lightweight Convolutional Neural Nework for Optical Flow Estimation

参考文献：[6]，CVPR 2018

5.7.1 背景知识

FlowNet2 是基于 CNN 进行光流估计的 SOTA 算法 (?), 需要 160M 的参数量。LiteFlowNet 实现了 30 倍的轻量化，并且比 FlowNet2 块 1.36 倍。

主要的贡献：

- 在每一层金字塔 (Pyramid Level) 预测光溜更高效的轻量级联网络。通过早前矫正 (Early Correction) 提高了精度，同时无缝支持网络中的描述子匹配。
- 提出了一个新型的光流正则化层，可以改善野值点、光流边界模糊等问题，是基于特征驱动的区域卷积实现
- 网络结构可以高效提取 Pyramidal Feature 以及 Feature Warping，而不是像 FlowNet2 中的 Image Warping。

FlowNet2 通过级联 FlowNet，来不断调优光流场, by contributing on the flow increment between the first image and the warped second imaeg?

后来，SPyNet 通过在每一 Pyramid level 采用 Imae Warping 实现了只有 1.2M 大小的网络，但精度只有 FlowNet 大小，而达不到 FlowNet2 的水平。

Image Warping: 图像扭转，是一种数字图像处理过程，在任何图像中所描述的任何形状都会产生显著有损。扭曲可用于矫正图像有损，同时可用于某种创意目的。纯粹的图像扭曲意味着点到点的映射，而不改变其颜色。

提高 FlowNet2 以及 SPyNet 的两种准则 (Principles)：

- Pyramid feature extraction

  Consists of an encoder and a decoder. Encoder 把输入的图像对分别映射到多尺度高维特征空间中；Decoder 以 Coarse-to-fine 的框架估计光流场。这比 FlowNet2 采用 U-Net 更轻量化。相比于 SPyNet，我们的模型把特征提取与光流估计两个过程分离，可以更好的处理精度与复杂度之间的矛盾。

- Feature Warping

  FlowNet2 与 SPyNet 将输入图像对中的第二幅图像基于先前估计的光流进行 Warp，然后使用 Warped 的第二幅图像与第一幅图像的特征图谱 Refine 估计的光流。

所以这个过程中，首先把第二幅图像进行 Warp, 然后提取 Warp 图像的特征，这个过程十分繁琐。所以，本文提出直接对 Feature Map 进行 warp。保证模型更精确以及高效。

更详细的细节一定要看原文 [6]。

此外，除了上面两个主要改进的 Principle, 作者还提出第三个比较重要的改进措施，那就是 Flow Regularization.

- Flow Regularization

  级联的光流估计类似于能量最小化方法中的保真度 (Data Fidelity) 的作用。为了消除边界的模糊以及野值点，Regularize flow Field 的常用 Cues：

  - Local flow consistency
  - Co-occurrence between flow boundaries
  - Co-occurrence between intensity edges

  对应的代表方法包括：

  - Anisotropic image-driven
  - image- and flow-driven
  - Complementary regularizations

  在本文中，提出的是 Feature-driven local convolution layer at each pyramid level. 该方法对 Flow-以及 Image-敏感。

  评语：看起来，一个是提高精度，一个是提高效率，这是两个最终的目的。提高精度可以通过设计网络结构以及增加其它考虑来实现；提高效率的一个重要表现是降低模型的参数数量，提高运算速度。不过，本文虽然参数数量减少了 30 倍，但速度却只提高了 1.36 倍，这里面的原因是什么？

5.7.2 Related Works

Variational Methods

Address illumination changes by combining the brightness and gradient constancy assumptions.

DeepFlow, propose to correlate multi-scale patches and incorporate this as the matching term in functional.

PatchMatch Filter, EpicFlow.

本文提出的网络结构，是受 Variational methods 中的 Data Fidelity 以及正则化启发。

Machine Learning Methods

PCA-Flow. 这些参数化模型可以通过 CNN 高效的实现。

CNN-based Methods

FlowNet, 使用能量最小化作为后处理步骤，来降低在光流边界的平滑效应。不能端到端训练？

FlowNet2，通过 FlowNet 的级联实现。虽然提高了精度，但模型更大，计算更复杂。

SPyNet，受 Spatial Pyramid 启发，模型更紧凑，但效果远不如 FlowNet2。

InterpoNet，借助第三方系数光流但需要 off-the-shelf 的边缘检测。

DeepFlow, 使用 Correlation 而不是真正意义的 CNN，参数不能训练。

Establish Point Correspondence

Establishing point correspondence, 一种方式是 Match Image Patches

CNN-Feature Matching, 首先被 Zagoruyko 等人提出 (Learning to compare image patches via CNN, 2015 CVPR)。

MRF-based，Güney 等人提出利用 Feature Representation 以及利用 MRF 来估计光流。

Bailer 使用多尺度 Feature，然后以类似于 Flow Fields 的方式进行特征匹配。

Fischer 和 Ilg 等人为了提高计算效率，仅在稀疏空间维度进行特征匹配。

本文中，We reduce the computational burden of feature matching by using a short-ranged matching of warped CNN features at sampled positions and a sub-pixel refinement at every pyramid level.

类似于 Spatial Transformer, 本文利用 f-warp layer 来区分不同个 Channel. 本文的决策网络是一个更普用的 Warping Network，可以用来 Warp 高层次的 CNN Features，而不仅仅是对 Image 进行 Warpping.

5.7.3 LiteFlowNet

整体结构如下：

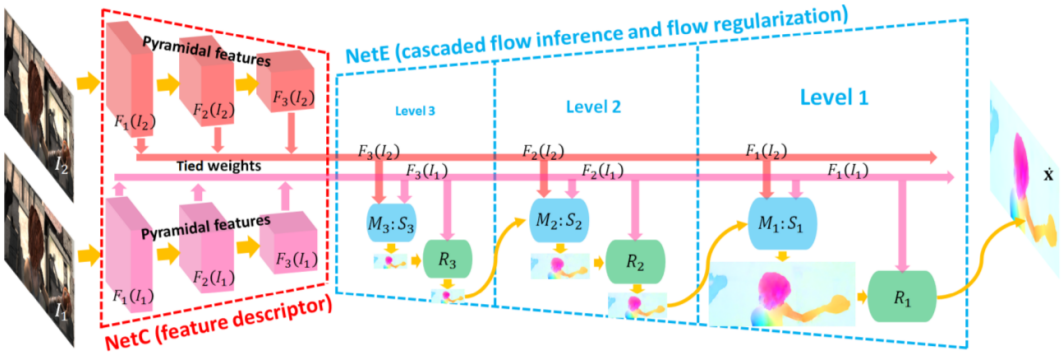


Figure 2: The network structure of LiteFlowNet. For the ease of representation, only a 3-level design is shown. Given an image pair (I_1 and I_2), NetC generates two pyramids of high-level features ($\mathcal{F}_k(I_1)$ in pink and $\mathcal{F}_k(I_2)$ in red, $k \in [1, 3]$). NetE yields multi-scale flow fields that each of them is generated by a cascaded flow inference module M:S (in blue color, including a descriptor matching unit M and a sub-pixel refinement unit S) and a regularization module R (in green color). Flow inference and regularization modules correspond to data fidelity and regularization terms in conventional energy minimization methods respectively.

Fig 5.8. LiteFlowNet 结构框图

Pyramid Feature Extraction

进行 stride-s 的卷积操作，得到的 Feature Map 表示为：$\mathcal{F}_k(I_i)$，即第 i 个图像的第 k 层 Feature Map。简化写成 $\mathcal{F}_i$。

Feature Warping (f-warp)

假设 ẋ 为预测的光流，则 Feature Warping 是指：

$$\tilde{\mathcal{F}}_2(x) \triangleq \mathcal{F}_2(x + \dot{x}) \sim \mathcal{F}_1(x)$$

注意，Warping 是作用于 $\mathcal{F}$ 上的，而不是输入图像上的。

为了使上述过程可以支持 end-to-end 训练，这里采用 Bilinear Interpolation 进行插值的技术实现 Warping。Bilinear Interpolation 是支持后向传播训练的！修改后 Warping 实现公式如下：

$$\tilde{\mathcal{F}} = \sum_{x_s^i \in N(x_s)} \mathcal{F}(x_s^i)(1 - |x_s - x_s^i|)(1 - |y_s - y_s^i|)$$

其中，$x_s = x + \dot{x} = (x_s, y_s)^T$ 是输入的源 Feature Map 中的坐标。$x$denotes the target coordinates of the regular grid in the interpolated feature map $\mathcal{F}$，$N(x)$ 代表 the four pixel neighbors of x_s.

自己的理解：首先 x 是插值后的图像索引，而 $x_s = x + \dot{x}$ 是插值前 Feature Map 中对应 Object Feature 的 x 索引处的索引。x_s^i 是在 x_s 周围的四个紧邻像素。也就是个 Bilinear Interpolated.

Cascaded Flow Interface

评语：这一块看的比较吃力。为什么会吃力？因为新的概念么？那么作者为什么提出这么麻烦的概念呢？实际效果又怎么样呢？

通过计算高层特征向量的 Correlation 来实现输入图像的点对应。

$$c(x, d) = \mathcal{F}_1(x) \cdot \mathcal{F}_2(x + d)/N$$

这个公式跟 FlowNet 中的计算方式区别不大。只不过这里的 N 是指 Feature 的长度。

M Module 在 Descirptor Matching unit M, Residual flow $\Delta\dot{x}_m$. A complete flow field $\dot{x}_m$ is computed as follows, 这句话没懂

$$\dot{x}_m = \underbrace{M(C(\mathcal{F}_1, \tilde{\mathcal{F}}_2; d))}_{\Delta\dot{x}_m} + s\dot{x}^{\uparrow s}$$

其中，$\dot{x}$ 是上一层的最开始的光流估计！

S Module 为了进一步提高 flow estimate $\dot{x}_m$ 的精度，即达到亚像素级别。作者引入了 Second flow inference。这可以防止错误光流的放大并传到下一级。Sub-pixel refinement unit S，会产生一个更准确的光流场，这通过最小化 $\mathcal{F}_1$ 和 $\tilde{\mathcal{F}}_2$ 之间的距离来实现。

$$\dot{x}_s = \underbrace{S(C(\mathcal{F}_1, \tilde{\mathcal{F}}_2, \dot{x}_m))}_{\Delta\dot{x}_s} + \dot{x}_m$$

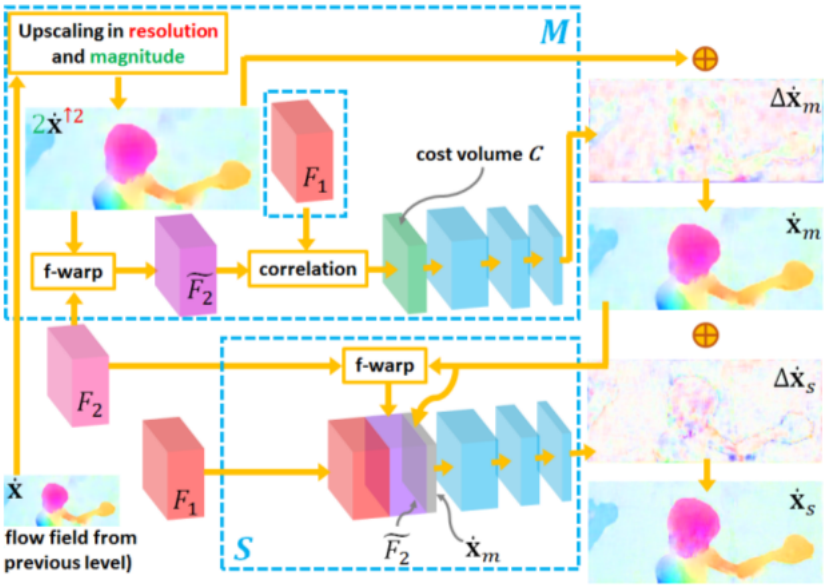


Figure 3: A cascaded flow inference module M:S in NetE. It consists of a descriptor matching unit M and a sub-pixel refinement unit S. In M, f-warp transforms high-level feature $\mathcal{F}_2$ to $\widetilde{\mathcal{F}}_2$ via upscaled flow field $2\dot{\mathbf{x}}^{\uparrow 2}$ estimated at previous pyramid level. In S, $\mathcal{F}_2$ is warped by $\dot{\mathbf{x}}_m$ from M. In comparison to residual flow $\Delta\dot{\mathbf{x}}_m$, more flow adjustment exists at flow boundaries in $\Delta\dot{\mathbf{x}}_s$.

Fig 5.9. 在 NetE 中的级联光流推理模块,M:S

所以总的来说 M 模块是为了计算 $\Delta\dot{x}_m$，而 S 模块是为了计算 $\Delta\dot{x}_s$。对最开始 $\dot{x}$ 估计的光流进行两次的 Refinement.

Flow Regularization

这一部分主要消除光流边界的模糊、存在的 artifacts 等。提出用 feature-driven local convolution (f-lcon)

假设 Feature Map (F) 的尺寸为：$M * N * c$，定义 f-lcon 的滤波器为 $\mathbf{G} = g$

对于输入为 $\dot{x}_s$，Flow Regularization 值的是：

$$\dot{x}_r = R(\dot{x}_s; G)$$

输出的是正则化后的光流估计 $\dot{x}_r$

下面的关键是如何生成这个用于正则化的卷积核。为此，作者定义了一个 feature-driven 的距离度量 $\mathcal{D}$，总的来说，该度量由一个 CNN unit R_D 来计算：

$$\mathcal{D} = R_D(\mathcal{F}_1, \dot{x}_s, O)$$

基于这个度量，可以计算得到卷积核：

$$g(x, y, c) = \frac{\exp(-\mathcal{D}(x, y, c)^2)}{\sum_{(x_i, y_i) \in N(x,y)} \exp(-\mathcal{D}(x_i, y_i, c)^2)}$$

其中 $N(x)$ 表示一个 $\omega * \omega$ 的近邻。

5.7.4 Ablation Study

算法的结果是优于 FlowNet2, SPyNet 等。

Feature Warping

没有 Warping, 光流更 Vague. 通过计算 residual flow (M:S 两个模块的功能) 可以提高估计效果。

M: Matching

S: Sub-pixel refinement

R: Regularization units in NetE

Descriptor Matching

Sub-Pixel Refinement

The flow field generated from WMS is more crisp and contains more fine details than that generated from WM with sub-pixel refinement dis- abled.

更小的 flow artifacts.

5.7.5 Regularization

In comparison WMS with regularization disabled to ALL, undesired artifacts exist in homogeneous regions

Flow bleeding and vague flow boundaries are observed.

表明，Feature-driven local convolution 对于光滑光流场、保持 crisp flow boundaries 非常重要！

5.7.6 Conclusion

Pyramidal feature extrac- tion and feature warping (f-warp) help us to break the de facto rule of accurate flow network requiring large model size. To address large-displacement and detail-preserving flows, LiteFlowNet exploits short-range matching to gener- ate pixel-level flow field and further improves the estimate to sub-pixel accuracy in the cascaded flow inference. To result crisp flow boundaries, LiteFlowNet regularizes flow field through feature-driven local convolution (f-lcon). With its lightweight, accurate, and fast flow computation, we ex- pect that LiteFlowNet can be deployed to many applications such as motion segmentation, action recognition, SLAM, 3D reconstruction and more.

5.8 小结

2018.05.23 小结

在生成 Semantic Map 的时候，看样子现在的趋势，是利用 RNN 保证时间一致性；利用多模态数据提高精度，但如何融合多模态数据的 Feature 有待研究，现有的有一些是直接 Concatenate、Knowledge Distillation、Attnetion(?) 等机制。

5.9 ExFuse: Enhancing Feature Fusion for Semantic Segmentation

参考文献：ExFuse 简介 -知乎

5.9.1 要解决的问题

在语义分割领域中，经常需要融合多层的 Feature。然而，底层的 Feature 含有的语义信息较少，但分辨率较高，噪声也比较少，这是由于卷积层比较浅；而高层的 Feature 语义信息多，但空间分辨率很小。

所以本文就提出了：1) 增加底层特征的语义；2) 在高层中增加空间信息

5.9.2 Method

使用了 ResNet、GCN(Global Convolution Net) 的思想。

其中，SS, SEB, ECRE, DAP 是文章作者提出的算法。

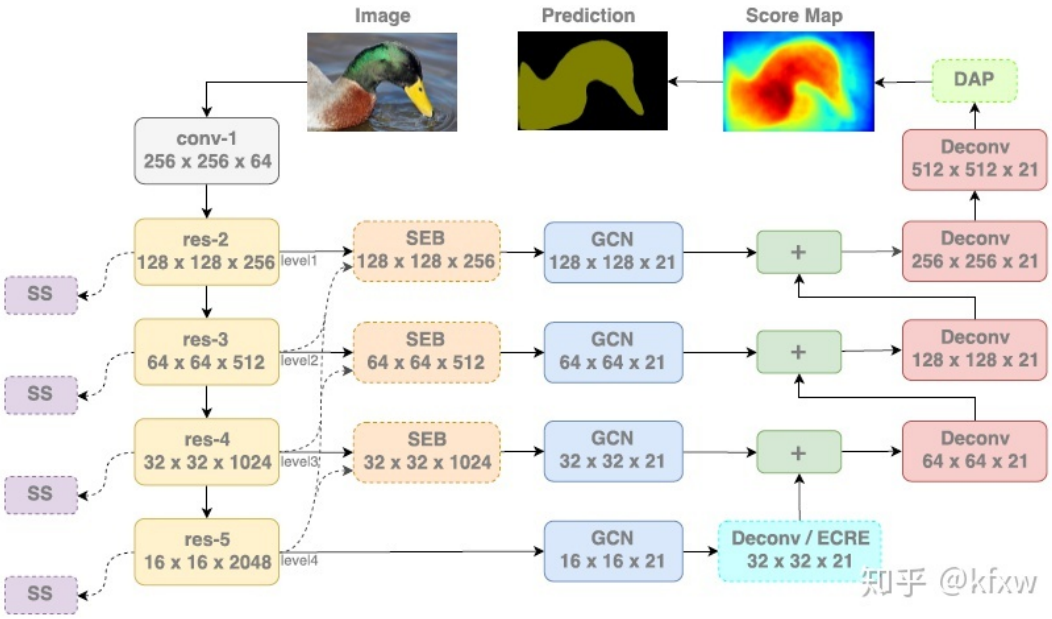


Fig 5.10. ExFusion 的实现框图

在底层中加入更多的语义信息

具体是三个方面的改进：

- Layer Rearrengement

  ResNeXt 网络结构中，各级的网络包含的残差单元个数为 3,4,23,3。为了提高底层特征的语义性，一个想法便是让低层的两级网络拥有的层数更多。因此作者将残差单元个数重排为 8,8,9,8，并重新在 ImageNet 上预训练模型。重排后网络的分类性能没有明显变化，但是分割模型可以提高约 0.8 个点 (mean intersection over union) 的性能。

- Semantic Supervision(SS)

  深度语义监督其实在其他的一些工作里 (如 GoogLeNet，边缘检测的 HED 等等) 已经使用到了。这里的使用方法基本上没有太大变化，能够带来大约 1 个点的提升。

- Semantic Embedding branch(SEB)

  其做法是将高层特征上采样后，与低层特征逐像素相乘，用在 GCN 之前。该部分能带来大约 0.7 个点的提升。

在高层中加入更多的空间信息

用两种方法来把更多的空间信息融入到高层特征中：

- 通道分辨率嵌入 (Explicit Channel resolution embedding, ECRE)

  其思路是在上采样支路中使用 [2,3,4] 工作中都使用到的子像素上采样模块 (sub-pixel upsample)。作者的出发点并不是前人工作中强调的如速度快、消除反卷积的棋盘效应等等，而是通过这个结构能够让和空间信息相关的监督

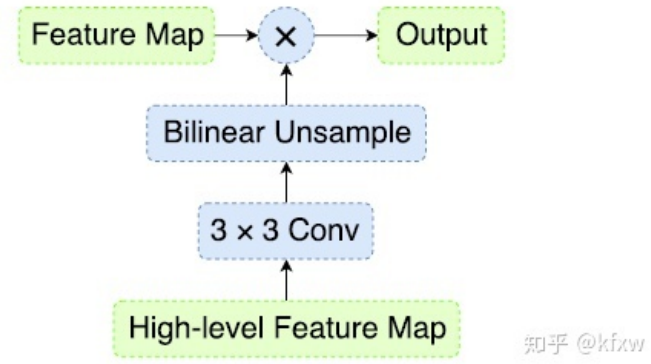


Fig 5.11. 语义嵌入分支的结构图

信息回传到各个通道中，从而让不同通道包含不同空间信息。该模块和原有的反卷积一起使用才能显示出更好的性能。同单独使用反卷积相比，性能可以提高约 0.6 个点。

- 稠密邻域预测 (Densely adjacent prediction, DAP)

  DAP 模块只使用在输出预测结果的时候。其想法也是通过扩展通道数来增加空间信息。举一个例子来描述其功能，假设 DAP 的作用区域为 3x3，输出结果的通道数为 21，则扩展后的输出通道数为 21x3x3。每 3x3 个通道融合成一个通道。如在最终结果中，第 5 通道 (共 21 通道) 的 (12,13) 坐标上的像素，是通过 DAP 之前的第 5+0 通道 (11,12)、5+1 通道的 (11,13)、5+2 通道的 (11,14)、5+3 通道的 (12,12)、5+4 通道的 (12,13)、5+5 通道的 (12,14)…平均得到的。DAP 能带来约 0.6 个点的提升。

5.10 Multi-View Deep Learning for Consistent Semantic Mapping with RGB-D Cameras

时间：2018.05.24

Chapter 6

Open Set Recognition

Open set recognition with GAN & OpenMax.

6.1 GAN

6.1.1 GAN 原理笔记

参考文献：GAN 原理学习笔记 -知乎

传统的生成模型，如自编码机 (Auto-Encoder)，通常采取 MSE 作为 Loss Function，这样的弊端是学习得到的 Decoder 模块性能不太令人满意。

GAN 原理

首先，真实数据的分布已知,$P_{data}(x)$，我们需要做的就是生成一些也在这个分布内的图片，但无法直接利用这个分布。

刚才讨论过了，MSE 的损失函数可能效果较差，改进办法是用交叉熵 (Cross Entroy) 来计算损失，从下面的推导可以看出，**交叉熵的最大化等价于 KL 散度的最小化**。KL 散度衡量的是模型分布与真实数据分布之间的差异。

模型分布用 P_G 表示，数据分布用 P_{data} 表示。

$$\begin{aligned}
\hat{\theta} &= \arg\max_{\theta} \prod_{i=0}^{m} P_G(y_i|x_i,\theta) && (6.1)\\
&= \arg\max_{\theta} \sum_{i=0}^{m} \log P_G(y_i|x_i,\theta) \\
&= \arg\max_{\theta} E_{x\sim P_{data}} \log\log P_G(y_i|x_i,\theta) \\
&= \arg\max_{\theta} \int_x P_{data}(x)\log P_G(y_i|x_i,\theta)dx - \int_x P_{data}(x)\log P_G(y_i|x_i,\theta)dx && (6.2)\\
&= \arg\min_{\theta} \int_x P_{data}(x)\log\frac{P_{data}(x)}{P_G(x;\theta)}dx && (6.3)\\
&= \arg\min_{\theta} KL\left(P_{data}(x), P_G(x;\theta)\right) && (6.4)
\end{aligned}$$

其中,m 表示从训练数据中采样的样本数。主要难理解的地方在于公式（6.2）中的后一项，由于这一项与 θ 无关，所以加上之后也不会影响 $\arg\max_\theta$ 运算的取值。

GAN 公式

$$V(G,D) = E_{x\sim P_{data}}[\log D(X)] + E_{x\sim P_G}[1-\log D(X)] \tag{6.5}$$

优化目标是：

$$G^\star = arg\min_G \max_D V(G,D) \tag{6.6}$$

下面解释上面的两个式子。

D 的作用是让这个式子尽可能的大。对于第一项，在输入 x 来自于真实数据时为了使 V 最大，$D(x)$ 应该接近于 1；对于第二项，在输入 x 来自于 G 的生成时，则应该使 $D(x)$ 尽可能的接近于 0。

训练过程

根据公式6.5与6.6，GAN 的训练过程是 G 与 D 相互**迭代更新**的过程。具体如下：注意，可能更新多次 D 次之后才更新一次 G。

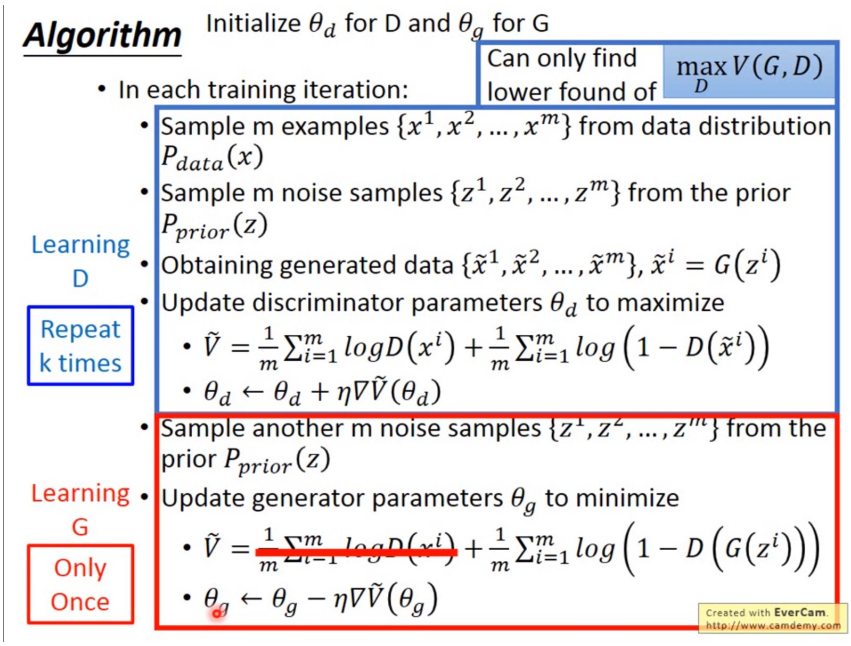


Fig 6.1. GAN 训练过程

给定 G，首先推导 D 的最优解。

$$\begin{aligned} V(G,D) &= E_{x\sim P_{data}}[\log D(X)] + E_{x\sim P_G}[1-\log D(X)] \\ &= \int_x P_{data}(x)\log D(x)dx + \int_x P_G \log(1-D(x))dx \\ &= \int_x P_{data}(x)\log D(x) + P_G(x)\log(1-D(x))dx \end{aligned} \tag{6.7}$$

在给定的 x 时，对上式中 D 的最大化，等价于：

$$\arg\max_D \left[\underset{a}{P_{data}(x)} \log D(x) + \underset{b}{P_G(x)} (1 - \log D(x)) \right]$$

另

$$f(D) = a \log D + b \log(1 - D) \tag{6.8}$$

对式6.8进行求导并另其为 0 得到最优 D 的表达式：

$$\begin{aligned} D^\star(x) &= \frac{a}{a+b} \\ &= \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \end{aligned}$$

把上式关于 D 的最优解代入6.5可以得到以下公式：

$$\begin{aligned} \max_D V(G, D) &= V(G, D^\star) \\ &= \int_x P_{data}(x) \log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} dx + \int_x P_G(x) \log \frac{P_G(x)}{P_{data}(x) + P_G(x)} dx \\ &= -2log2 + KL(P_{data}(x) \parallel \frac{P_{data}(x) + P_G(x)}{2}) + KL(P_G(x) \parallel \frac{P_{data}(x) + P_G(x)}{2}) \\ &= -2\log 2 + 2JSD(P_{data}(x) \parallel P_G(x)) \end{aligned}$$

其中，JS 散度是 KL 散度的平滑版本，表示两个分部之间的差异。所以固定 G 时，$\max_D V(G, D)$ 表示两个分布之间的差异，最小值是 $-2\log 2$，最大值是 0。当 $P_G(x) \equiv P_{data}(x)$，$G$ 是最优的。

Loss Function 中的两个小问题

- 修改 G 的 Loss Function

  现有的 G 的 Loss Function 中的 $\log(1 - D(x))$ 在 $D(x)$ 趋近于 0 时，梯度非常小。所以在开始训练时，十分缓慢，一种改进办法是将其更改为：$\min_V = -\frac{1}{m} \sum_{i=1}^m \log(D(x^i))$

- Mode Collapse

  这是由于 KL 散度的不对称引起的，一种办法是将 KL 求解的顺序取反。即：$KL(P_G \parallel P_{data})$ 更改为 $KL(P_{data} \parallel P_G)$

$$KL(P_{data} \parallel P_G) = E_{x \sim P_{data}} \log P_G(x)$$

Time: 2018.05.21

6.2 从头开始 GAN

参考文献：从头开始 GAN 知乎

6.2.1 定义

Ian Goodfellow 自己的话说 GAN[5]:

> The adversarial modeling framework is most straightforward to apply when the models are both multilayer perceptrons. To learn the generator's distribution pg over data x, we define a prior on input noise variables $p_z(z)$, then represent a mapping to data space as $G(z;_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters $_g$. We also define a second multilayer perceptron $D(x;_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $log(1-D(G(z)))$.

简单的说，GAN 包含以下三个主要元素：

- 两个网络：一个生成网络 G，一个判别网络 D
- 训练误差函数：
  - G Net: $\log(1 - D(G(z)))$
    希望 $D(G(z))$ 趋近于 1。
  - D Net: $-(\log D(x) + \log(1 - D(G(z))))$
    D 网络是一个二分类，希望真实数据的输出趋近于 1，而生成数据的输出即 $D(G(z))$ 趋近于 0。
- 数据输入：G Net 的输入是随机噪声，D Net 的输入是混合 G 的输出与真实数据样本的数据

值得注意的地方在于，训练 D 时，输入数据同时来自于真实数据 x 以及生成数据 $G(z)$。且不用 Cross Entropy 的原因是，如果使用 CE, 会使 $D(G(z))$ 变为 0，导致没有梯度，而 GAN 这里的做法是让 $D(G(z))$ 收敛到 0.5。

实际训练中，G Net 使用了 ReLU 和 Sigmoid，而 D Net 中使用了 MaxOut 和 DropOut，并且修改了 G Net 的 Loss Function, 后一点可参考上一节。

但作者指出, 此时的 GAN 不容易训练。

6.2.2 DCGAN：Deep Convolution GAN

DCGAN 的几点改造：

- 去掉了 G 网络和 D 网络中的 Pooling Layer
- 在 G 网络和 D 网络中都是用 BN
- 去掉全连接的隐藏层
- 在 G 网络中去除最后一层 ReLU，改用 Tanh
- 在 D 网络中每一层使用 LeakyReLU

G 网络使用了 4 层反卷积，而 D 网络使用了 4 层卷积。基本上，G 网络和 D 网络的结构正好反过来的。在使用 DCGAN 生成图像的研究线上，最新到了 BEGAN(十个月以前)，达到了以假乱真的效果。

6.2.3 CGAN: Conditional Generative Adversarial Nets

此时，输入不再仅是随机的噪声。就是在 G 网络的输入在 z 的基础上连接一个输入 y，然后在 D 网络的输入在 x 的基础上也连接一个 y：

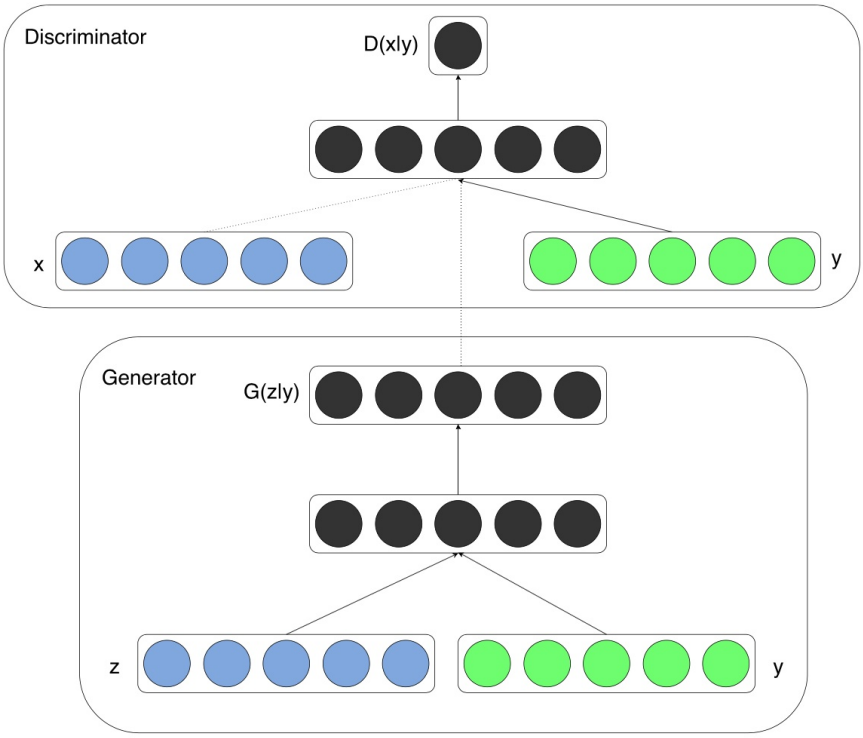


Fig 6.2. CGAN 示意图，在 G、N 网络中新增了数据 y

相应的目标函数变为：

$$arg\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{data}}[\log D(x|y)] + \mathbb{E}_{x\sim P_G}[\log(1-D(G(z|y)))]$$

训练方式几乎就是不变的，但是从 GAN 的无监督变成了有监督。只是大家可以看到，这里和传统的图像分类这样的任务正好反过来了，图像分类是输入图片，然后对图像进行分类，而这里是输入分类，要反过来输出图像。显然后者要比前者难。

6.2.4 InfoGAN

在 CGAN 的基础上，将其变为无监督学习过程。要实现无监督的 CGAN，意味着需要让神经网络不但通过学习提取了特征，还需要把特征表达出来。

怎么做呢？作者引入了信息论的知识，也就是 mutual information 互信息。作者的思路就是 G 网络的输入除了 z 之外同样类似 CGAN 输入一个 c 变量，这个变量一开始神经网络并不知道是什么含义，但是没关系，我们希望 c 与 G 网络输出的 x 之间的互信息最大化，也就是让神经网络自己去训练 c 与输出之间的关系。

Mutual Information 在文章中的定义如下：

$$I(c, G(z, c)) = \mathbb{E}_{c\sim P(c), x\sim G(z,c)} \left[\log Q(c|X) + H(c)\right]$$

其中，H 为熵运算。Q 网络则是反过来基于 X 输出 c。基于上式定义的 I，则整个 GAN 的训练目标变为：

$$\min_G \max_D V(D, G) - \lambda I(c, G(z, c))$$

相比于 CGAN，InfoGAN 又做了如下改变：

- D 网络的输入只有 x，不加 c
- Q 网络和 D 网络共享同一个网络，只是到最后一层独立输出

6.3 Generative Adversarial Nets

参考文献：[5]
评语：需要重点关注，提出一个新的网络后，如何在数学上证明是可行的呢？

6.4 Towards Open Set Deep Networks

引入了一种新型的新型的神经网络层结构，OpenMax，可以评估输入来自于一个未知类的概率。其中一个重要的组成部分是采用 Meta-Recognition 来对网络的 Activation patterns 进行处理。

比对 SoftMax 的结果进行 Thresholding 要好很多。(为什么会好？)

这是因为，实际使用中，即使输入是非常奇怪的，也就是说"fooling" or "Rubbish" images 时，网络也可能会在某一类上产生很高的输出概率，所以这时候通过设置 Threshod 的方式是不合适的。

> They strongly suggests that thresholding on uncertainty is not sufficient to determine what is unknown.

6.4.1 Introduction & Related Works

In Sec. 3, we show that extending deep networks to threshold SoftMax probabil- ity improves open set recognition somewhat, but does not resolve the issue of fooling images.

Thresholding 可能会在某些时候会帮助 Open Set Recognition，但对于 Fooling images，结果可能比较差。

Chapter 7

MXNet

参考文献：MXNet Architecture
Time: 2018.05.18

7.1 Optimizing Memory Consumption in DL

Over the last ten years, a constant trend in deep learning is towards deeper and larger networks. Despite rapid advances in hardware performance, cutting-edge deep learning models continue to push the limits of GPU RAM. So even today, it’s always desirable to find ways to train larger models while consuming less memory. Doing so enables us to train faster, using larger batch sizes, and consequently achieving a higher GPU utilization rate.

7.1.1 Computation Graph

A computation graph describes the (data flow) dependencies between the operations in the deep network. The operations performed in the graph can be either fine-grained or coarse-grained.

The concept of a computation graph is explicitly encoded in packages like Theano and CGT. In other libraries, computation graphs appear implicitly as network configuration files. The major difference in these libraries comes down to how they calculate gradients. There are mainly two ways: performing back-propagation on the same graph or explicitly representing a backwards path to calculate the required gradients.

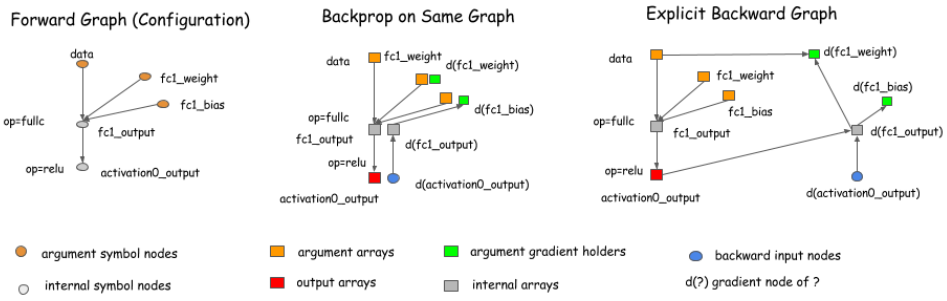


Fig 7.1. The implicitly & explicitly back-propagation on Graph

Libraries like Caffe, CXXNet, and Torch take the former approach, performing backprop on the original graph. Libraries like Theano and CGT take the latter approach, explicitly representing the backward path. In this discussion, we adopt the explicit backward path approach because it has several advantages for optimization.

We adopt the explicit backward path approach because it has several advantages for optimization.

Why is explicit backward path better? Two reasons:

- The explicit backward path clearly describes the dependency between computations. Like the following case, where we want to get the gradient of **A** and **B**. As we can see clearly from the graph, the computation of the $d(C)$ gradient doesn't depend on F. This means that we can free the memory of F right after the forward computation is done. Similarly, the memory of F can be recycled.

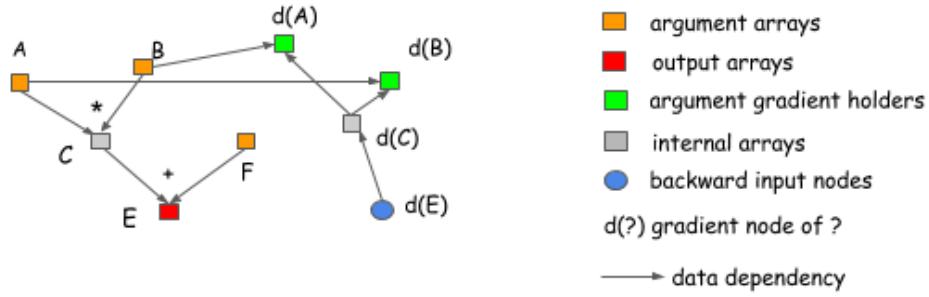


Fig 7.2. Dependencies can be found quickly.

- Another advantage of the explicit backward path is the ability to have a different backward path, instead of a mirror of forward one.
  A common example is the split connectioncase, as shown in the following figure. In this example, the output of **B** is referenced by two operations. If we want to do

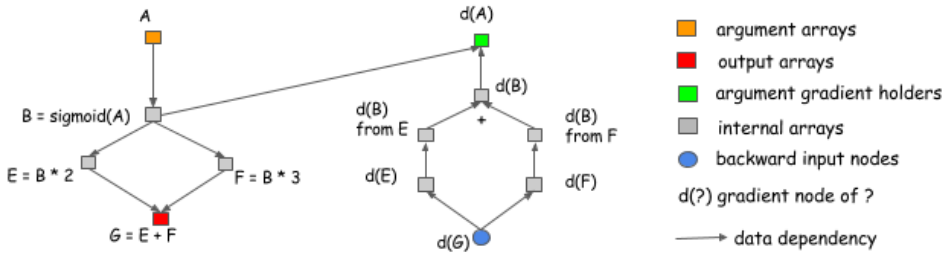


Fig 7.3. Different backward path from forward path.

the gradient calculation in the same network, we need to introduce an explicit split layer. This means we need to do the split for the forward pass, too. In this figure, the forward pass doesn't contain a split layer, but the graph will automatically insert a gradient aggregation node before passing the gradient back to **B**. This helps us to save the memory cost of allocating the output of the split layer, and the operation cost of replicating the data in the forward pass.

7.1.2 What Can be Optimized?

As you can see, the computation graph is a useful way to discuss memory allocation optimization techniques. Already, we’ ve shown how you can save some memory by using the explicit backward graph. Now let’ s explore further optimizations, and see how we might determine reasonable baselines for benchmarking.

Assume that we want to build a neural network with n layers. Typically, when implementing a neural network, we need to allocate node space for both the output of each layer and the gradient values used during back-propagation. This means we need roughly $2n$ memory cells. We face the same requirement when using the explicit backward graph approach because the number of nodes in a backward pass is roughly the same as in a forward pass.

In-place Operations

One of the simplest techniques we can employ is *In-place memory sharing* across operations. For neural networks, we can usually apply this technique for the operations corresponding to activation functions.

"In-place" means using same memory for input and output. But you should be careful about that the result is used by more than one operation!

Standard Memory Sharing

In-place operations are not the only places where we can share memory. In the following example, because the value of B is no longer needed after we compute E, we can reuse B’ s memory to hold the result of E.

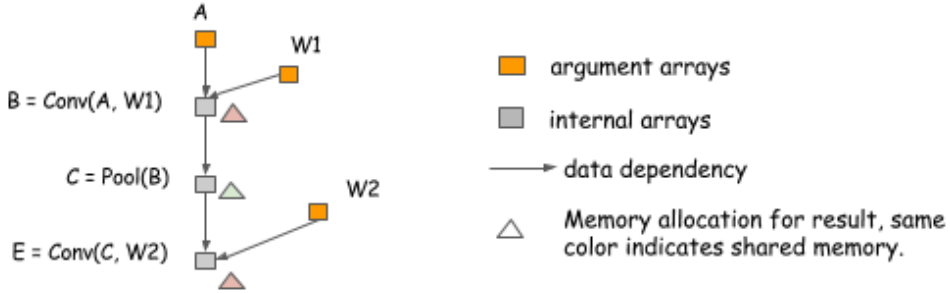


Fig 7.4. Standard Memory sharing between **B** & the result of **E**.

Memory sharing doesn't necessarily require the same data shape. Note that in the preceding example, the shapes of **B** and **E** can differ. To handle such a situation, we can allocate a memory region of size equal to the maximum of that required by **B** and **E** and share it between them.

7.1.3 Memory Allocation Algorithm

Based on the " In-Place Operatioins", how can we allocate memory correctly?

The key problem is that we need to place resources so that they don’t conflict with each other. More specifically, each variable has a **life time** between the time it gets computed

until the last time it is used. In the case of the multi-layer perceptron, the life time of *fc1* ends after *act1* get computed. See below figure:

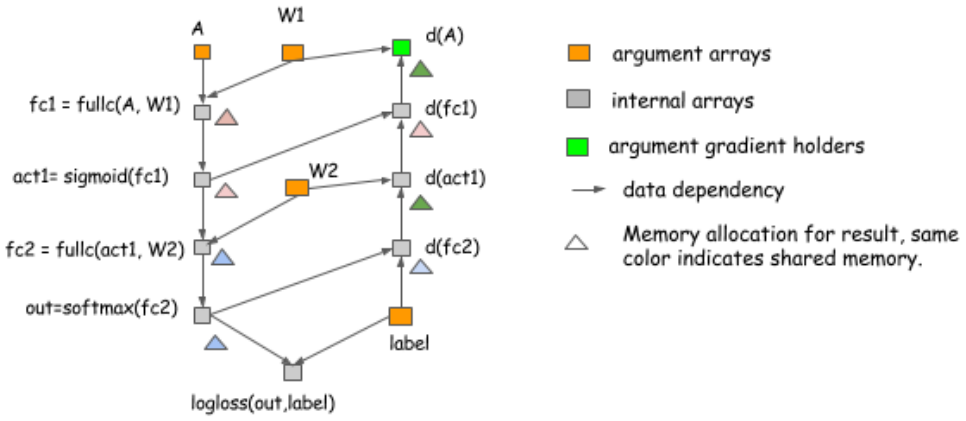


Fig 7.5. Standard Memory sharing between **B** & the result of **E**.

The principle is to allow memory sharing only between variables whose lifetimes don't overlap. There are multiple ways to do this. You can construct the conflicting graph with each variable as a node and link the edge between variables with overlapping lifespans, and then run a graph-coloring algorithm. This likely has $O(n^2)$ complexity, where n is the number of nodes in the graph. This might be too costly.

Let's consider another simple heuristic. The idea is to simulate the procedure of traversing the graph, and keep a count of future operations that depends on the node.

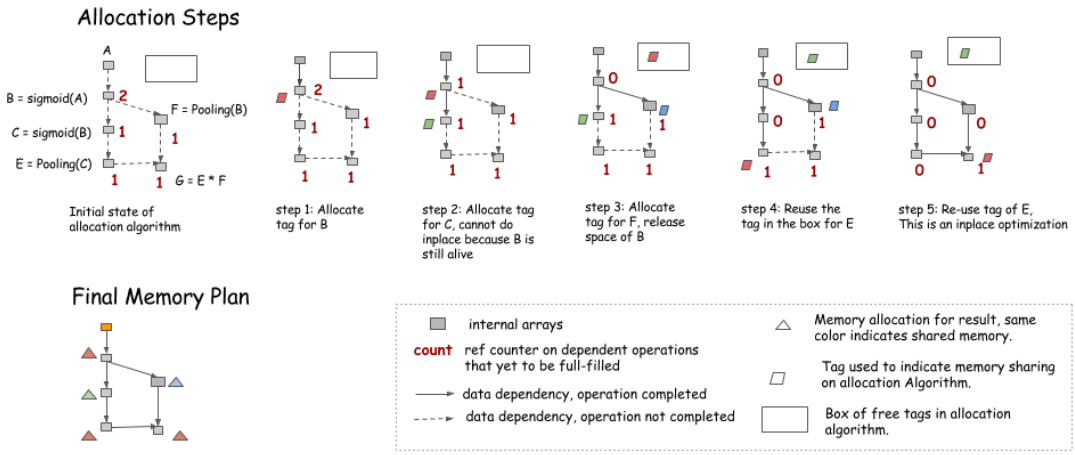


Fig 7.6. Standard Memory sharing between **B** & the result of **E**.

- An in-place optimization can be performed when only the current operation depends on the source (i.e. count == 1).
- Memory can be recycled into the box on the upper right corner when the *count* goes to 0.
- When we need new memory, we can either get it from the box or allocate a new one.

Noet: During the simulation, no memory is allocated. Instead, we keep a record of how much memory each node needs, and allocate the maximum of the shared parts in the final memory plan.

7.1.4 Static vs. Dynamic Allocation

The major difference is that static allocation is only done once, so we can afford to use more complicated algorithms. For example, we can search for memory sizes that are similar to the required memory block. The Allocation can also be made graph aware. We’ll talk about that in the next section. Dynamic allocation puts more pressure on fast memory allocation and garbage collection.

There is also one takeaway for users who want to rely on dynamic memory allocations: do not unnecessarily reference objects. For example, if we organize all of the nodes in a list and store then in a Net object, these nodes will never get dereferenced, and we gain no space. Unfortunately, this is a common way to organize code.

7.1.5 Memory Allocation for Parallel Operations

In the previous section, we discussed how we can simulate running the procedure for a computation graph to get a static allocation plan. However, optimizing for parallel computation presents other challenges because resource sharing and parallelization are on the two ends of a balance. Let’s look at the following two allocation plans for the same graph:

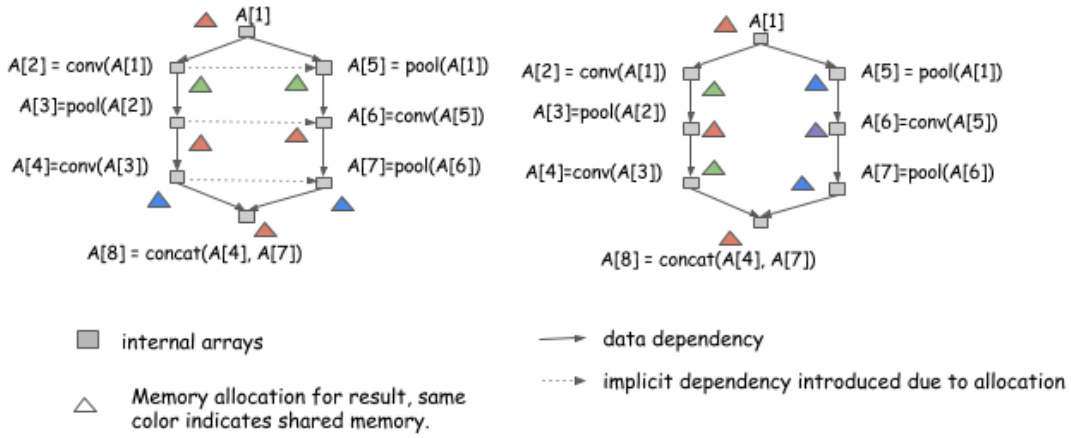


Fig 7.7. Standard Memory sharing between **B** & the result of **E**.

Both allocation plans are valid if we run the computation serially, **from** $A[1]$ **to** $A[8]$**.** However, the allocation plan on the left introduces additional dependencies, which means we can't run computation of $A[2]$ and $A[5]$ in parallel. The plan on the right can. To parallelize computation, we need to take greater care.

Be Correct and Safe First

Being correct is our first principle. This means to execute in a way that takes implicit dependency memory sharing into consideration. You can do this by adding the implicit

dependency edge to the execution graph. Or, even simpler, if the execution engine is mutation aware, as described in our discussion of dependency engine design, push the operation in sequence and write to the same variable tag that represents the same memory region.

Always produce a safe memory allocation plan. This means never allocate the same memory to nodes that can be parallelized. This might not be ideal when memory reduction is more desirable, and we don't gain too much when we can get benefit from multiple computing streams simultaneously executing on the same GPU.

Try to Allow More Parallelization

Now we can safely perform some optimizations. The general idea is to try and encourage memory sharing between nodes that can't be parallelized. You can do this by creating an ancestor relationship graph and querying it during allocation, which costs approximately $O(n^2)$ in time to construct. We can also use a heuristic here, for example, color the path in the graph. As shown in the following figure, when you try to find the longest paths in the graph, color them the same color and continue.

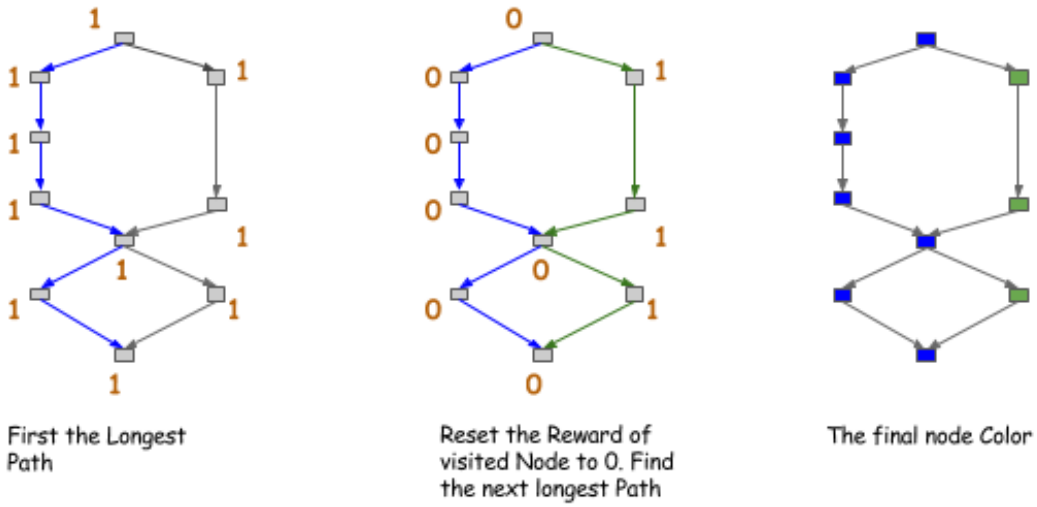


Fig 7.8. Color the longest paths in the Graph.

After you get the color of the node, you allow sharing (or encourage sharing) only between nodes of the same color. This is a stricter version of the ancestor relationship, but is costs only $O(n)$ of time if you search for only the first k path.

7.1.6 How Much Can we Save ?

On coarse-grained operation graphs that are already optimized for big operations, you can reduce memory consumption roughly by half. You can reduce memory usage even more if you are optimizing a fine-grained computation network used by symbolic libraries, such as Theano.

7.1.7 References

More details can be found in: Opimizing the Memory Consumption in DL(MXNet).

7.2 Deep Learning Programming Style

Two of the most important high-level design decisions

- Whether to embrace the symbolic or imperative paradigm for mathematical computation
- Whether to build networks with bigger or more atomic operations

7.2.1 Symbolic vs. Imperative Program

即：符号式编程 vs. 命令式编程

Symbolic programs are a bit different. With symbolic-style programs, we first define a (potentially complex) function abstractly. When defining the function, no actual numerical computation takes place. We define the abstract function in terms of **placeholder values**(占位符). Then we can compile the function, and evaluate it given real inputs.

This operation generates a computation graph (also called a symbolic graph) that represents the computation.

Most symbolic-style programs contain, either explicitly or implicitly, a compile step.

真正的计算只发生在传入数值之时，在这之前，都没有任何计算发生。

The defining characteristic of symbolic programs is their clear separation between building the computation graph and executing it.For neural networks, we typically define the entire model as a single compute graph.

7.2.2 Imperative Programs Tend to be More Flexible

使用命令式编程，那么任何 Python 语法都可以使用 (Nearly anything), 但使用符号式编程时，一些 Python 特性可能无法使用，如迭代。

当使用 Python 的符号式编程时，实际实在一个 Domain-Specific-Language(DSL) 定义的空间中进行编程。

Intuitively, you might say that imperative programs are more native than symbolic programs. It’s easier to use native language features.

7.2.3 Symbolic Programs Tend to be More Efficient

命令式编程与原生 Python 的相差不大，所以灵活性很高。但符号式编程更有利于速度、存储优化。

Symbolic programs are more restricted. When we call *Compile* on d, we tell the system that only the value of d is needed. The intermediate values of the computation, is then invisible to us.

- We benefit because the symbolic programs can then safely reuse the memory for in-place computation.

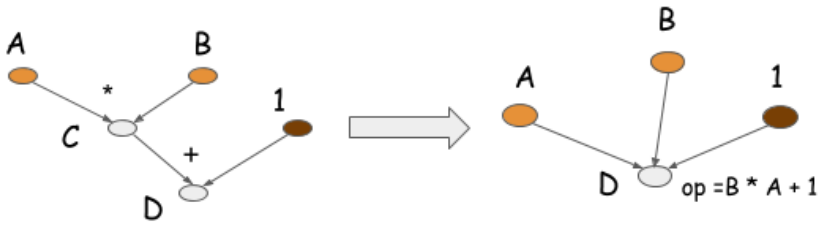


Fig 7.9. Operation Folding 示意图。

- Symbolic programs can also perform another kind of optimization, called operation folding(图7.9). In fact, this is one way we hand-craft operations in optimized libraries, such as CXXNet and Caffe. Operation folding improves computation efficiency.

  Note, you can't perform operation folding in imperative programs, because the intermediate values might be referenced in the future. Operation folding is possible in symbolic programs because you get the entire computation graph, and a clear specification of which values will be needed and which are not.

7.2.4 Case Study: Backprop and AutoDiff

toy model

```
import numpy as np
    a = np.ones(10)
    b = np.ones(10) * 2
    c = b * a
    d = c + 1
    ...
```

基于命令式编程的自动求导

循环调用求导函数，直至最开始的输入变量。
利用 grad 闭包 (Closure) 来隐含的保存后向计算图。
但一个坏处是，必须保存所有中间变脸的 Grad 闭包。

基于符号式编程的自动求导

可以实现的优化力度更大。

Analysis

可以实现的优化的程度，依赖于可以允许的操作 (Restrictions on what you can do)。使用符号式编程时，必须明确提供这些约束，所以可进行优化也就更多。

对于命令式编程，可以通过其它的一些方式添加明确的约束。比如一种方法是 Context Variable。如：

```
with context.NoGradient()
    ...
```

可以关闭梯度的计算。但这样也不能利用 In-Place Calculation 来对存储空间进行复用。

其实是一种 trade-off between restriction and flexibility.

7.2.5 Model Checkpoint

保存和加载模型。保存模型的时候，需要保存两类变量：网络的结构配置、网络的权重系数。

符号式编程有利于配置的检查。而对于命令式编程，需要保存所有的代码，或者利用符号式指令进行能够顶层封装。

- Parameter Updates

  大部分符号是编程，都是基于数据流图 (Data Flow Graphs, DFG) 实现的。DFG 描述的是计算。但这种方式下，参数的更新不是很方便。一些做法是将更新过程转化为基于命令式的方式实现，而梯度的计算是基于计算图的方式计算。

- There is No Strict Boundary

  这两种风格的框架其实没有很明显的区分。如在命令式编程时，可以借助 Just-in-Time(JIT) Compiler 来实现一些符号式编程里面的全局优化等好处。

7.2.6 Big vs. Small Operations

- Big Operations

  主要是一些经典的神经网络层在用，如 Fully Connected and Batch Normalize.

- Small Operations

  一些数学上的计算，如矩阵乘、Element-wise Addition.

CXXNet, Caffe 支持 Layer 一级的计算，而 Theano, Minerva 支持更精细的计算。

- Smaller Operations Can Be More Flexible (小运算更灵活)

  可实现的东西多，且建立新的 Layer 比较简单，直接添加部件就行。

- Big Operations Are More Efficient (大运算更高效)

  可能引起计算、存储上的开支。

- Compilation and Optimization

  对于小运算的优化，计算图支持一下两种优化：

  - Memory Allocation Optimization
    重用中间结果的存储空间。也可用于大运算。
  - Operator Fusion
    Fuse several small operations into big one.

这些优化对小操作十分重要。小操作对编译器也增加了负担。

- Expression Template and Statically Typed Language

  借助 Expression Template 来产生具体的 Kernels.Template Expression, 其实底层是基于 C++ Template 实现的。

7.2.7 Mix The Approaches

Amdahl's Law:

> If you are optimizing a non-performance-critical part of your problem, you won’t get much of a performance gain

实际考虑编程 Style 时，需要综合考虑：性能、灵活性、工程复杂度等。实践表明，混合使用多个 Style 可以得到更好的性能。

- Symbolic and Imperative Programs

  有两种方式可以实现这种混用：

  - Use imperative programs within symbolic programs as callbacks
  - Use symbolic programs as part of imperative programs

  如在参数更新中的讨论。如果代码中，混合了 Symbolic 和 Imperative，那么结果是 Imperative. 但更好的选择是，用支持 GPU 计算、参数更新的符号式编程框架来开发。

- Small and Big Operations

- Choose Your Own Approach

7.3 Dependency Engine for Deep Learning

7.3.1 Problems in Dependency Scheduling

- Data Flow Dependency

  Data flow dependency describes how the outcome of one computation can be used in other computations.

- Memory Recycling

- Random Number Generation

  A pseudo-random number generator (PRNG) is not thread-safe because it might cause some internal state to mutate when generating a new number. Even if the PRNG is thread-safe, it is preferable to serialize number generation, so we can get reproducible random numbers.

Design a Generic Dependency Engine

目标是建立一个轻量级、普用的依赖引擎。目标如下：

- 识别有效的操作
- 可以调度 GPU、CPU 存储的依赖，以及处理随机发生器的依赖
- 引擎不应分配资源，而仅处理依赖

步骤如下：

1. At the beginning, the user can allocate the variable tag, and attach it to each of the objects that we want to schedule.

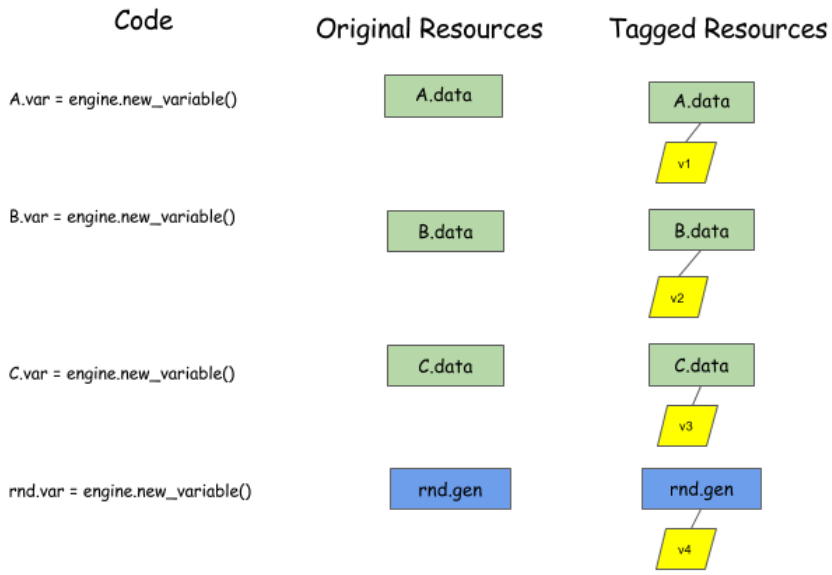


Fig 7.10. 第一步，给变量分配 tag

2. 然后调用 *push* 来告知引擎需要运行的函数、参数等，需要区分读取、写入的参数，分别输入。引擎通过识别上一步的 tag 来识别变量，这样的好处是不涉及 tag 具体指向什么，所以引擎可以处理包括变量、函数之类的 Everything。

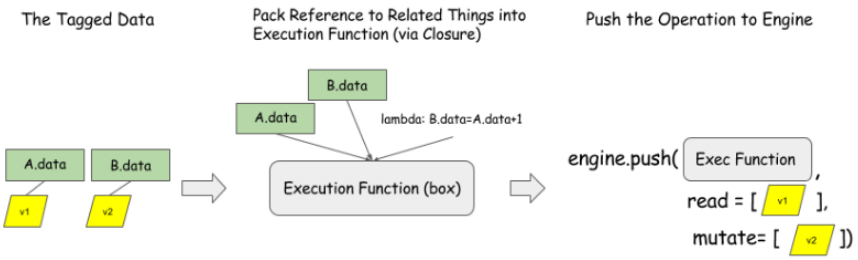


Fig 7.11. 把相应的 Function Closure push 进依赖分析引擎

7.3.2 Implementing the Generic Dependency Engine

基本思想

- Use a queue to track all of the pending dependencies on each variable tag
- Use a counter on each operation to track how many dependencies are yet to be fulfilled
- When operations are completed, update the state of the queue and dependency counters to schedule new operations

一个例子如图所示：

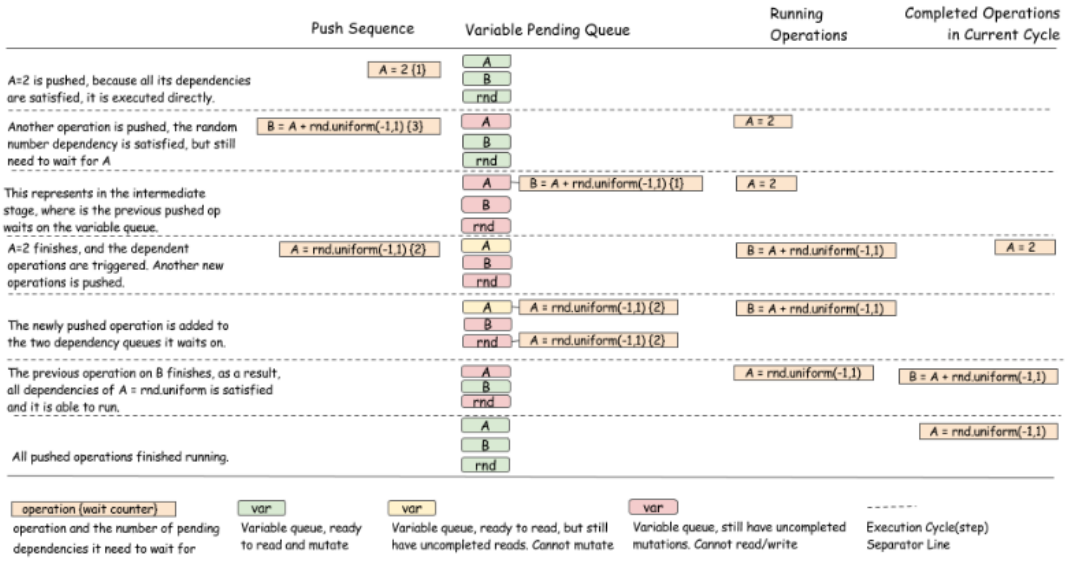


Fig 7.12. 一个具体的例子

7.3.3 Discussion

- Dynamic vs. Static
- Mutation vs. Immutable

7.4 Designing Efficient Data Loaders for DL

几点重要的考虑：

- Small File Size
- Parallel (Distributed) packing of data
- Fast data loading and online augmentation
- Quick reads from arbitrary parts of the dataset in the distributed setting

7.4.1 Design Insight

为了设计好的 IO 系统，需要解决两类任务：Data Preparation, Data Loading.

Data Preparation

Data preparation describes the process of packing data into a desired format for later processing.

- Pack the dataset into small numbers of files
- Do the packing once
- Process the packing in parallel to save time
- Be able to access arbitrary parts of the data easily

Data Loading

The next step to consider is how to load the packed data into RAM.

- Read Continuously
- Reduce the bytes to be loaded，可以借助于数据压缩等
- Load and train in different threads
- Save RAM

7.4.2 Data Format

需要选择既高效又方便的数据结构。为了实现这个目标，把二进制数据包装成可以分离的结构。具体，MXNet 采用 DMLC-Core 里面的 recordIO 类型。

通过连续读，来避免随机读引入的延时。

这种数据结构的一个好处是，每一个 record 的长度可以改变。从而支持更好的数据压缩。

其中，*resize* 把输入图像变为 256 * 256 大小。

Access Arbitrary Parts of Data

The packed data can be logically sliced into an arbitrary number of partitions, no matter how many physical packed data files there are.

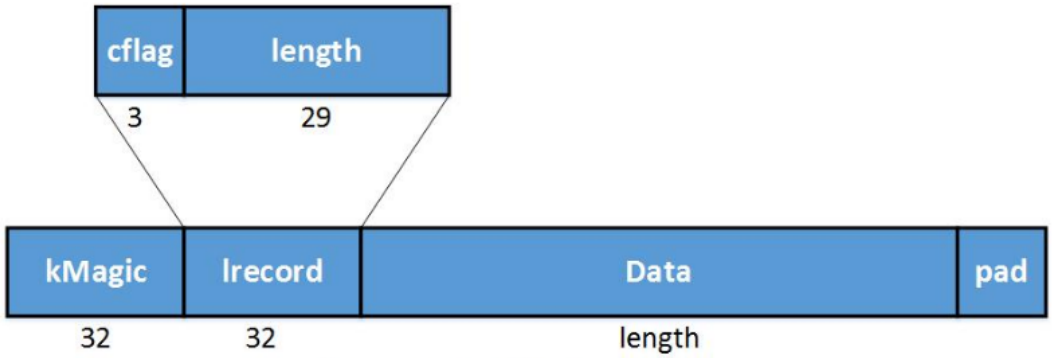
在 recordIO 中借助 Magic Number 来实现上述目的，具体是使用 dmlc-core 中的 *InputSplit* 函数来实现。这个函数极大的帮助了并行实现，因为每一个节点只处理一个 *Part*.

7.4.3 Data Loading and Preprocessing

Loading and Preprocessing on the Fly

In service of efficiency, we also address multi-threading techniques.

在加载了大量图像数据后，利用多线程工具 (OpenMP) 进行并行处理。



Binary recordIO Basic Format

In MXNet's binary RecordIO, we store each data instance as a record. **kMagic** is a *magic number* indicating the start of a record. **Lrecord** encodes length and a continue flag. In lrecord,

- cflag == 0: this is a complete record
- cflag == 1: start of a multiple-records
- cflag == 2: middle of multiple-records
- cflag == 3: end of multiple-records

Data is the space to save data content. **Pad** is simply a padding space to make record align to 4 bytes.

Fig 7.13. Binary recordIO 数据结构

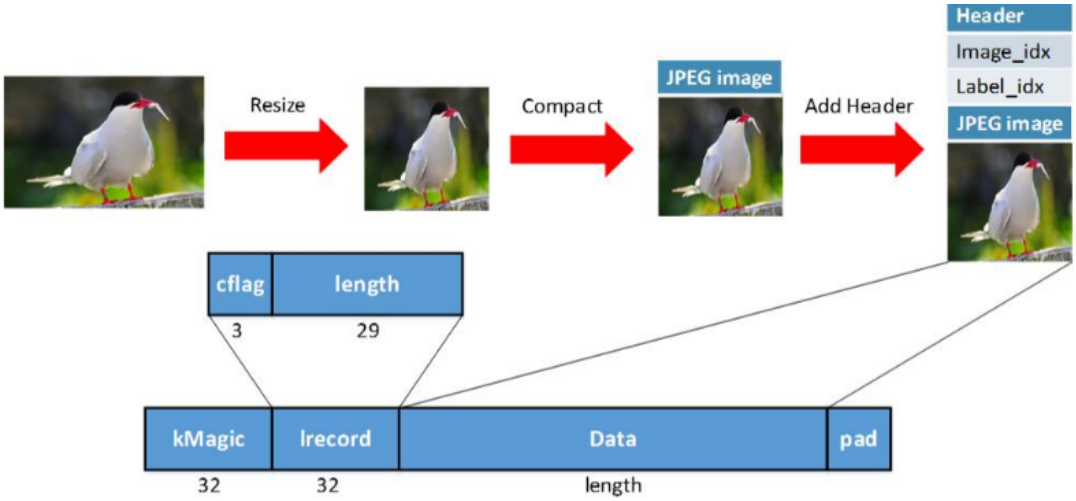


Fig 7.14. Binary recordIO 的一个例子

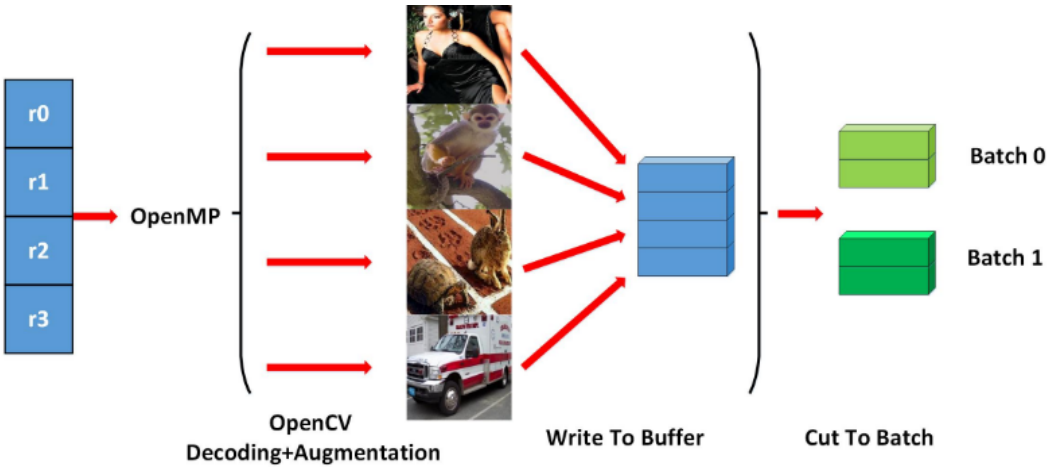


Fig 7.15. 并行预处理例子

Hide IO Cost Using Threadediter

一种降低 IO 影响的办法是：数据预取。具体使用 dmlc-core 提供的 *threadediter* 来处理 IO. The key of threadediter is to start a stand-alone thread that acts as a data provider, while the main thread acts as a data consumer as illustrated below.

The threadediter maintains a buffer of a certain size and automatically fills the buffer when it’s not full. And after the consumer finishes consuming part of the data in the buffer, threadediter will reuse the space to save the next part of data.

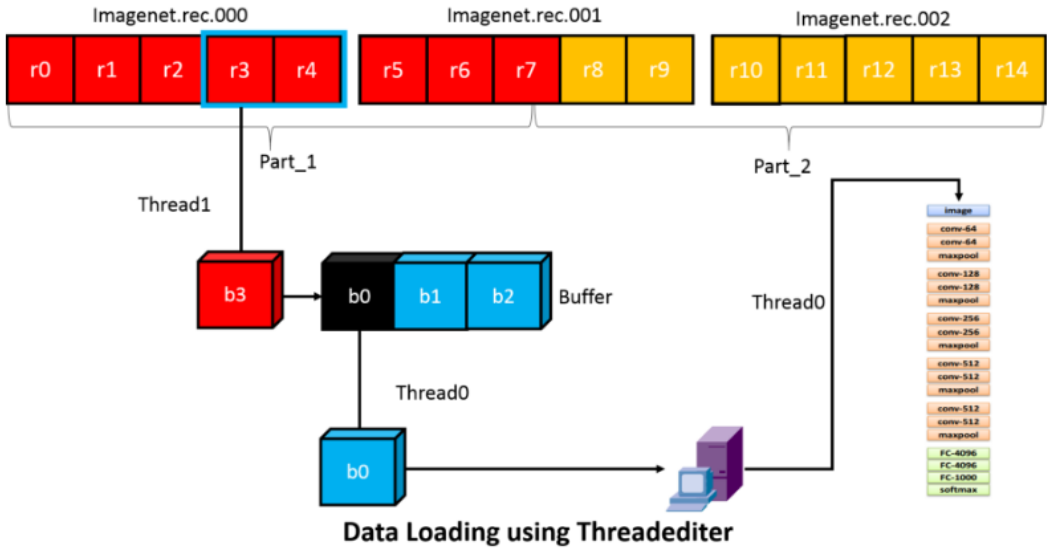


Fig 7.16. 数据预取的示意图，借助 Buffer 来实现

7.4.4 MXNet IO Python Interface

We make the IO object as an iterator in numpy. By achieving that, the user can easily access the data using a for-loop or calling next() function. Defining a data iterator is very similar to defining a symbolic operator in MXNet.

为了创建一个数据迭代器，需要提供五种类型的参数：

- Dataset Param: 如路径、输入的尺寸等
- Batch Param: Batch Size
- Augmentation Param: 确定数据增广的类型，如翻转等
- Backedn Param: 控制后台线程，来隐藏数据读取延时
- Auxiliary Param: 帮助 Debug 的信息等。

通常必须确定 **Dataset Param** 和 **Batch Param**。MX Data IO 也支持模块化，如以下两种：

- 自己的高效数据预取。allows the user to write a data loader that reads their customized binary format that automatically gets multi-threaded prefetcher support.
- 数据转换。image random cropping, mirroring, etc. Allows the users to use those tools, or plug in their own customized transformers

7.5 Except Handling in MXNet

MXNet 在两类情况下可以抛出异常：

- MXNet main thread. For *e.g.* Infershape and InferType。在主线程中处理
- Spawned threads (生成线程？)
  - By dependency engine for operator execution in parallel。会被 rethrown 到主线程，进行处理
  - By the iterators, during the data loading, text parsing phase *etc.*

Exception Handling for Iterators

CVIter 使用 PrefetcherIter 来加载和解析数据。PrefetcherIter 会生成一个 Producer 线程并后台运行，在主线程 (Consumer) 中使用这些数据。在 Producer 线程中可能抛出异常，该异常被发送到主线程。

可能引起线程之间的竞争 (Race).To avoid this situation, you should try and iterate through your full dataset if you think it can throw exceptions which need to be handled.

Except Handling for Operators

For the operator case, the dependency engine spawns a number of threads if it is running in the **ThreadedEnginePool** or **ThreadedEnginePerDevice** mode. The final operator is executed in one of the spawned threads.

If an operator throws an exception during execution, this exception is propagated down the dependency chain. Once there is a synchronizing call i.e. **WaitToRead** for a variable in the dependency chain, the propagated exception is rethrown.

注意：Please avoid waitalls in your code unless you are confident about your code not throwing exception in any scenario. 因为 **mx.nd.waitall** 不支持 rethrowing 异常。

Chapter 8

Tips in DL

8.1 Enlarge the FOV

增加网络的感受野。目前看到的主要方法如下：

- CRFs[10]
- Global Graph-reasoning module[2]
- Pooling
- Dilated conv[22]
-

8.2 Upsampling

在卷积以及 Pooling 之后保持分辨率。目前看到的主要方法如下：

- Padding
- Deconvolution
- Uppooling
-

8.3 Multiscale Ability

在目标检测 (Object Detection) 中加入多尺度信息。记得的有以下几个方法：

- Pyramid Network
- Stacked CNN ?
-

8.4 Dilated Convolution

主要原理如下。

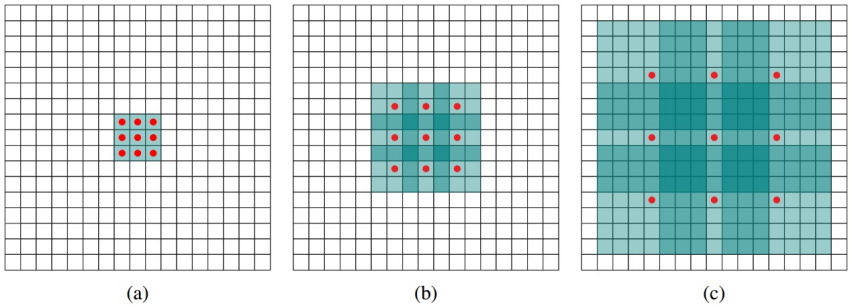


Fig 8.1. Dilated Convolution 示意图

注意，下文提到的 N-Dilated Conv 中的 $N = 1, 2, 3, \ldots$ 是指图中相邻红点之间的间隔。

图8.1中，(a) 图对应 3x3 的 1-dilated conv，和普通的卷积操作一样，(b) 图对应 3x3 的 2-dilated conv，实际的卷积 kernel size 还是 3x3，但是空洞为 1，也就是对于一个 7x7 的图像 patch，只有 9 个红色的点和 3x3 的 kernel 发生卷积操作，其余的点略过。也可以理解为 kernel 的 size 为 7x7，但是只有图中的 9 个点的权重不为 0，其余都为 0。可以看到虽然 kernel size 只有 3x3，但是这个卷积的感受野已经增大到了 7x7（**如果考虑到这个 2-dilated conv 的前一层是一个 1-dilated conv 的话，那么每个红点就是 1-dilated 的卷积输出，所以感受野为 3x3，所以 1-dilated 和 2-dilated 合起来就能达到 7x7 的 conv**）,(c) 图是 4-dilated conv 操作，同理跟在两个 1-dilated 和 2-dilated conv 的后面，能达到 15x15 的感受野。对比传统的 conv 操作，3 层 3x3 的卷积加起来，stride 为 1 的话，只能达到 (kernel-1)*layer+1=7 的感受野，也就是和层数 layer 成线性关系，而 dilated conv 的感受野是指数级的增长。

Dilated 的好处是不做 Pooling 算是信息的情况下，加大了感受野，让每个卷积核输出都包含较大范围的信息。在图像需要全局信息或者语音文本需要较长的 Sequence 信息依赖的问题中，都能很好的应用 Dilated Convolution, 比如图像分割、语音合成 WaveNet、机器翻译 ByteNet。

WaveNet 的例子。

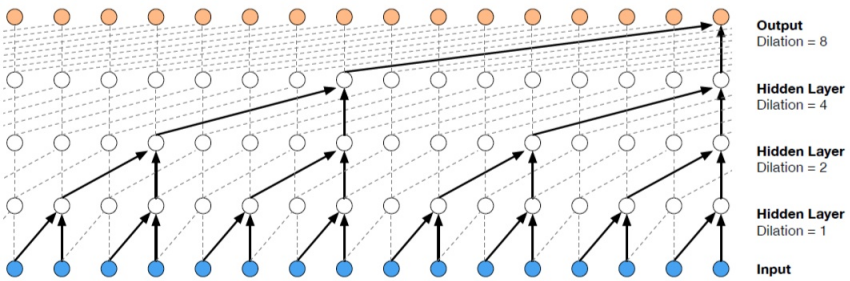


Fig 8.2. Dilated Convolution 在 WaveNet 中的应用示意图

参考文献：Dilated Conv 知乎

8.5 Deconvolutional Network

参考文献：Deconvolution Networks

可能应用的领域：Visualization, Pixel-wise Prediction, Unsupervised Learning, Image Generation.

大致可分为以下几个方面：

- Unsupervised Learning

  其实是 Covolutional Sparse Coding. 这里的 Deconv 只是观念上和传统的 Conv 反向，传统的 conv 是从图片生成 feature map，而 deconv 是用 unsupervised 的方法找到一组 kernel 和 feature map，让它们重建图片。

- CNN Visualization

  通过 deconv 将 CNN 中 conv 得到的 feature map 还原到像素空间，以观察特定的 feature map 对哪些 pattern 的图片敏感，这里的 deconv 其实不是 conv 的可逆运算，只是 conv 的 transpose，所以 tensorflow 里一般取名叫 transpose_conv。

- Upsampling

  在 pixel-wise prediction 比如 image segmentation[4] 以及 image generation[5] 中，由于需要做原始图片尺寸空间的预测，而卷积由于 stride 往往会降低图片 size，所以往往需要通过 upsampling 的方法来还原到原始图片尺寸，deconv 就充当了一个 upsampling 的角色。

下面主要介绍这三个方面的论文。

8.5.1 Convolutional Spare Coding

第一篇：Deconvolutional Netwoks

主要用于学习图片的中低层级的特征表示，属于 Unspervised Feature Learning。

更多内容参考本小节的参考文献。

8.5.2 CNN 可视化

ZF-Net 中利用 Deconv 来做可视化，它是将 CNN 学习到的 Feature Map 的卷积核，取转置，将图片特征从 Feature Map 空间转化到 Pixel 空间，用于发现哪些 Pixel 激活了特定的 Feature Map，达到分析理解 CNN 的目的。

8.5.3 Upsampling

用于 FCN[9] 和 DCGAN。

8.6 Dilated Network 与 Deconv Network 之间的区别

Dilated Convolution 主要用于增加感受野，而不是 Upsampling；Deconv Network 主要用于 Upsample，即增加图像分辨率。

对于标准的 $k \times k$ 的卷及操作，stride 为 s，分为一下几种情况：

- $s > 1$

  即卷积的同时做了降采样，输入 Feature Map 的分辨率[1]下降。但这一般也会增加感受野。

- $s = 1$

  普通的步长为 1 的卷积，输入与输出分辨率相同。

- **0 < s < 1**

  Fractionally strided convolution. 相当于图像做 upsampling。比如 $s = 0.5$ 时，意味着图像像素之间 padding 一个空白的像素 (像素值为 0) 后，stride 改为 1 进行卷积，达到一次卷积看到的空间范围变大的目的。

8.7 Uppooling

In the convnet, the max pooling operation is non-invertible, however we can obtain an approximate inverse by recording the locations of the maxima within each pooling region in a set of switch variables. In the deconvnet, the unpooling operation uses these switches to place the reconstructions from the layer above into appropriate locations, preserving the structure of the stimulus.

也就是说用一组开关变量保存最大值在 Pooling Region 中的位置。

参考文献：Quora Answer

8.8 目标检测中的 mAP 的含义

- 对于类别 C, 在一张图像上

  首先计算 C 在一张图像上的精度。

$$Precision_C = \frac{N(TP)_C}{N(Total)_C}$$

  其中，$Precision_C$ 为类别 C 在一张图像上的精度。$N(TP)_C$ 为算法检测正确 (True Positive) 的 C 的个数，检测是否正确按照 $IoU > 0.5$ 算，同理，$T(Total)_C$ 为这一张图像所有 C 类的个数。所以则一步，仅涉及一个类别 C 以及一张图像。

- 对于类别 C，在多张图像上

  这一步计算的是类别 C 的 AP 指数。

$$AveragePrecision_C = \frac{\sum Precision_C}{N(TotalImage)_C}$$

  其中，$AveragePrecision_C$ 是类别 C 的 AP 指数，$Precision_C$ 为上文计算得到的类别 C 的在一张图像上的精度，然后对所有包含类别 C 的图像上的 C

[1]分辨率是指像素的多少，而尺度是指模糊程度的大小，即 Gaussian Filter 中的方差 δ

的精度 $Precision_C$ 求和；$N(TotalImage)_C$ 为包含类别 C 的图像的数量，也对应于分子中求和所涉及的图像。

- 在整个数据集上，多个类别

  mAP 在上一步的计算结果的基础上，计算所有类别的 AP 和 / 总的类别数。

$$meanAveragePrecision = \frac{\sum_C AveragePrecision_C}{N(Class)}$$

也就是相当于计算所有类别的 **AP** 的平均值，是对应于类别总数的平均值。

参考文献：知乎文章

8.9 统计学习方法

一个比较好的总结：机器学习常见算法个人总结

8.10 Distillation Module

文献来源：[21][12]
在 [21] 中，同时完成深度估计以及场景解析两个任务。
Distillation Module 的目的：

- Deep-model distilation modules fuses information from the intermediate predictions for each specific final task[21]. 高效的利用中间任务的信息互补。文章 [21] 提出的三种不同的实现方式如图8.3所示。

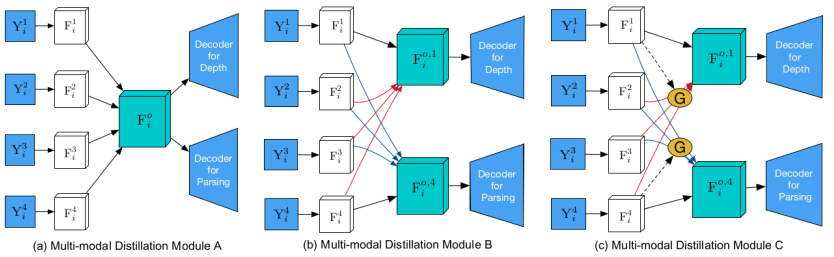


Figure 3. Illustration of the designed different multi-modal distillation modules. The symbols Y_i^1, Y_i^2, Y_i^3, Y_i^4 represent the predictions corresponding to multiple intermediate tasks. The distillation module A is a naive combination of the multiple predictions; the module B proposes a mechanism of passing message between different predictions; the module C shows an attention-guided message passing mechanism for distillation. The symbol G denotes a generated attention map which is used as guidance in the distillation.

Fig 8.3. 三种不同的 Distillation Module

- Distillation loss function[12]

  利用 Distillation 帮助将 Teacher Network(精度更高) 的知识迁移到的 Student Network.

8.10.1 Knowledge Distillation

什么是 Distilling the knowledge

一句话总结，就是用 teacher network 的输出作为 soft label 来训练一个 student network.

Disilling the knowledge in a Neural Network

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

其实就是一个 Softmax，值得注意的是 T 是 Temperature，T 越大，概率分布就越 Soft。在训练 Student Network 时，该概率分布就是 Student Network 的 soft label.

Student Network 的训练策略：

- 先用 Teacher Network 的概率分布训练
- 再用 Real Label 训练

8.10.2 Recurrent Knowledge Distillation [14]

8.11 光流估计中的 Average end-point error

貌似就是类似于均方误差类似。具体定义还没查到。

8.12 待续

Chapter 9

Image Processing

9.1 Feature Extraction

9.1.1 SIFT

详细信息可以参考：SIFT CSND 博客
我的未验证实现：SIFT Triloo Github

Chapter 10

Feature Extraction

总结文章：Object Detection 总结 Two Stage
总结文章：Object Detection 总结 One Stage

10.1 Selective Search

10.1.1 Efficient Graph-Based Image Segmentation

Selective Search 基于文章 [3] 中基于图分割的图像分割技术。

论文 [3] 提出的是一种基于贪心选择的图像分割算法，论文中把图像中的每个像素表示图上的一个节点，每一条连接节点的无向边都具有一个权重（weights），以衡量其连接的两个节点之间的不相似度，这篇论文的创新点在于该算法能够根据相邻区域在特征值上变化速度的大小动态调整分割阈值。这个特征值就是类间距离和类内距离，如果类间距离大于类内距离就认为是两个区域。定义类内距离为对应区域的最小生成树（因为把图像看做一个连接图，所以每个区域可以用最小生成树来表示）中权重最大的边的权重值，类间距离定义为两个区域内相邻点的最小权重边，如果两个区域没有相邻边则取无穷大。但是这样其实还是有问题，比如一个类中只有一个点的时候，它的类内距离为 0，这样就没法搞了（每个点都变成了一类，此时类内距离变为 0），所以作者又引入了一个阈值函数，用来表示两个区域的区域的类间距离至少要比类内距离大多少才能认为是两个区域。

判断两个点 C_1 与 C_2 是否属于同一类的判定：

$$D(C_1, C_2) = \begin{cases} true & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ false & \text{otherwise} \end{cases}$$

其中，$MInt(C_1, C_2)$ 为判断类间间距 Dif 的阈值，由类内间距计算得到：

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2))$$

其中,τ 即为控制当类间的最短距离。一般取为 C 的负相关函数，k 为常数：

$$\tau(C) = k/|C|$$

当 k 的取值大时，类间距要求大，所以分割后的区域面积也会较大，否则区域面积较小。

具体的分割过程，可以参考文章 [3] 的 **Algorithm 1**。

10.1.2 Selective Search

主要参考文献：[17]

接下来介绍 Selective Search 算法，该算法利用 [3] 的图分割算法获得的分割区域结果，再一次根据一些搜索策略（相似度）做了一个聚类。也是和上面的思路一致，首先根据获得的区域，算出每个区域和其他区域的相似度，不相邻的和自身与自身的相似度都设置为 0，得到一个 N*N 的矩阵，然后将相似度最大的合并，再次计算相似度矩阵（这里是增量更新，只需要计算新生成的区域和其他区域的相似度就可以了），这样合并一次较少一个区域，对于 N 个区域需要执行 N-1 次合并，最终得到一个区域。对于相似度的度量，作者主要选取了颜色和区域两大块，颜色作者比较了 8 种模型, 最终选择了 HSV，区域主要考量大小、纹理和吻合度（相交包含关系）这三个因素。

10.2 Region CNN

10.2.1 概述

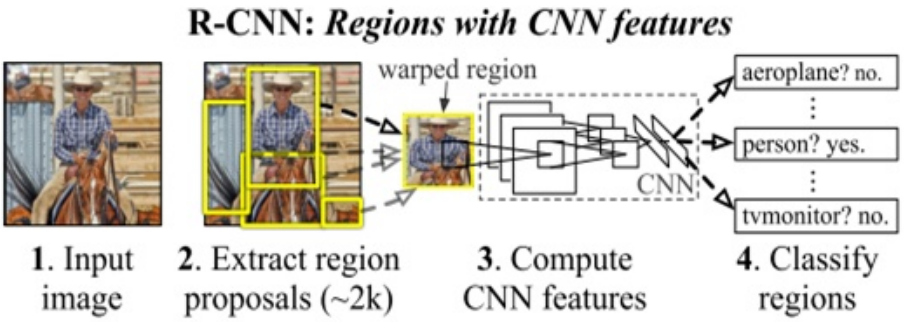


Fig 10.1. RCNN 整体思想

这里的 Extract region proposals 是基于 Selective Search 实现的, 然后将这些候选框输入到 CNN 进行特征提取，最后利用 SVM 对提取的特征进行分类。

10.3 SPP Net

10.4 Fast RCNN

10.5 Faster RCNN

10.6 R FCN

10.7 FPN

10.8 Mask RCNN

10.9 YOLO

10.10 YOLO v2

10.11 YOLO v3

10.12 SSD

10.13 DSSD

10.14 Retina Net (Focal Loss)

References

[1] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. Performance comparison of fpga, gpu and cpu in image processing. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 126--131. IEEE, 2009.

[2] Xinlei Chen, Li-Jia Li, Li Fei-Fei, and Abhinav Gupta. Iterative visual reasoning beyond convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[3] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167--181, 2004.

[4] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167--181, 2004.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672--2680. Curran Associates, Inc., 2014.

[6] T.-W. Hui, X. Tang, and C. Change Loy. LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation. *ArXiv e-prints*, May 2018.

[7] Chao Li, Yanjing Bi, Franck Marzani, and Fan Yang. Fast fpga prototyping for real-time image processing with very high-level synthesis. *Journal of Real-Time Image Processing*, pages 1--18, 2017.

[8] Shutao Li, Xudong Kang, and Jianwen Hu. Image fusion with guided filtering. *IEEE Transactions on Image Processing*, 22(7):2864--2875, 2013.

[9] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. *ArXiv e-prints*, November 2014.

[10] Roberto Cipolla Marvin T.T. Teichmann. Convolutional crfs for semantic segmentation. *arXiv:https://arxiv.org/pdf/1805.04777.pdf*, 2018.

[11] J. McCormac, A. Handa, A. Davison, and S. Leutenegger. SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks. *ArXiv e-prints*, September 2016.

[12] R. Mehta and C. Ozturk. Object detection at 200 Frames Per Second. *ArXiv e-prints*, May 2018.

[13] Trung T. Pham, Markus Eich, Ian Reid, and Gordon Wyeth. Geometrically consistent plane extraction for dense indoor 3d maps segmentation. In *Ieee/rsj International Conference on Intelligent Robots and Systems*, pages 4199--4204, 2016.

[14] Jan C. van Gemert Silvia L. Pintea, Yue Liu. Recurrent knowledge distillation. *ArXiv e-prints*, May 2018.

[15] N. Sünderhauf, T. T. Pham, Y. Latif, M. Milford, and I. Reid. Meaningful Maps With Object-Oriented Semantic Mapping. *ArXiv e-prints*, September 2016.

[16] Alexander Toet and Maarten A Hogervorst. Multiscale image fusion through guided filtering. In *SPIE Security+ Defence*, pages 99970J--99970J. International Society for Optics and Photonics, 2016.

[17] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154--171, Sep 2013.

[18] P. Wang, R. Yang, B. Cao, W. Xu, and Y. Lin. DeLS-3D: Deep Localization and Segmentation with a 3D Semantic Map. *ArXiv e-prints*, May 2018.

[19] R. Wang, J.-M. Frahm, and S. M. Pizer. Recurrent Neural Network for Learning DenseDepth and Ego-Motion from Video. *ArXiv e-prints*, May 2018.

[20] Y. Xiang and D. Fox. DA-RNN: Semantic Mapping with Data Associated Recurrent Neural Networks. *ArXiv e-prints*, March 2017.

[21] D. Xu, W. Ouyang, X. Wang, and N. Sebe. PAD-Net: Multi-Tasks Guided Prediction-and-Distillation Network for Simultaneous Depth Estimation and Scene Parsing. *ArXiv e-prints*, May 2018.

[22] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

Index