



# 智能计算系统

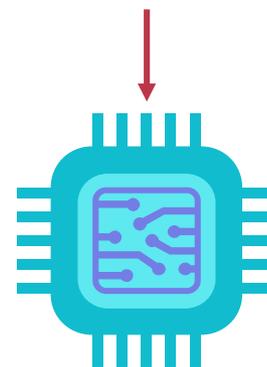
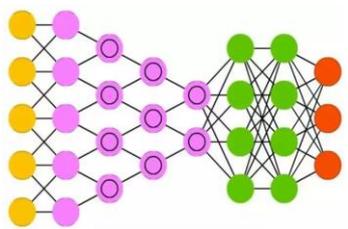
## 第六章 深度学习处理器原理

中国科学院计算技术研究所

陈云霄 研究员

[cyj@ict.ac.cn](mailto:cyj@ict.ac.cn)

# 运行



# 目录

- ▶ 深度学习处理器概述
- ▶ 目标算法分析
- ▶ 深度学习处理器DLP结构
- ▶ 优化设计
- ▶ 性能评价
- ▶ 其他加速器

# 目录

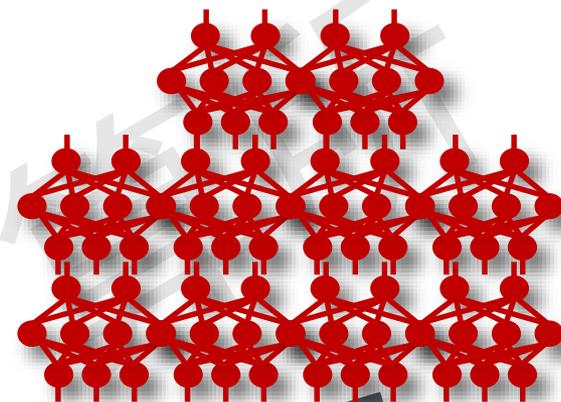
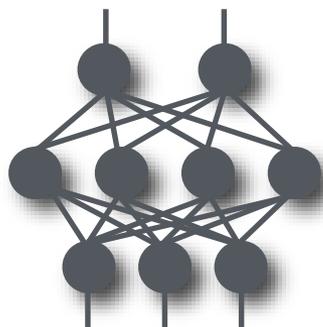
- ▶ 深度学习处理器概述
  - ▶ 研究意义
  - ▶ 发展历史
  - ▶ 设计思路

中科院计算所

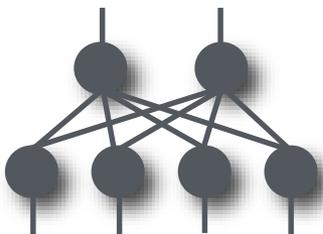
# 大而深的网络

深度神经网络

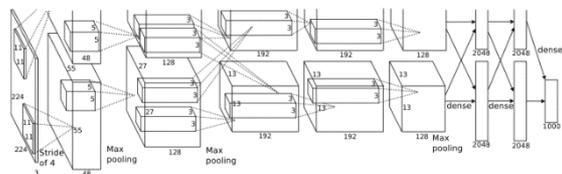
MLP



感知机



# 大而深的网络



6千万参数

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (pp. 1–9).

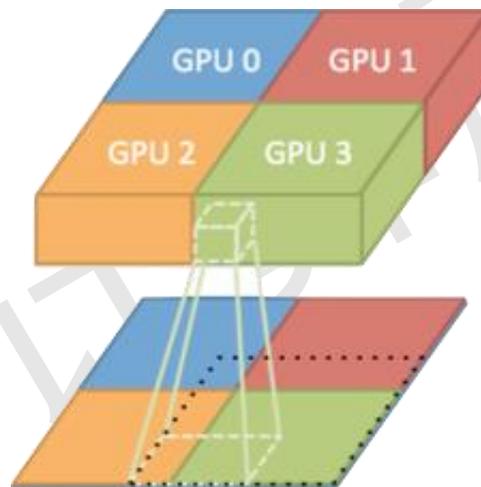
陈云霁 & 李玲 & 李威 et al.



十亿参数

Le, Q. V., Ranzato, M. A., Monga, R., Devin, M., Chen, K., Corrado, G. S., ... Ng, A. Y. (2012). Building High-level Features Using Large Scale Unsupervised Learning. In *International Conference on Machine Learning*.

<http://novelict.ac.cn/aics>



ResNet:  
152 层

商业某些  
网络:  
512层

110亿参数

Coates, A., Huval, B., Wang, T., Wu, D. J., & Ng, A. Y. (2013). Deep learning with cots hpc systems. In *International Conference on Machine Learning*.

2020年春季

# 为什么需要深度学习处理器

- ▶ 深度学习应用广泛
  - ▶ 图像识别、语音处理、自然语言处理、博弈游戏等领域
  - ▶ 已渗透到云服务器和智能手机的方方面面
- ▶ 通用CPU/GPU处理人工神经网络效率低下
  - ▶ 谷歌大脑：1.6万个CPU核跑数天完成猫脸识别训练
  - ▶ AlphaGo：和李世石下棋用了1202个CPU和176个GPU

# 专门的深度学习处理器

- ▶ 图形处理→GPU
- ▶ 信号处理→DSP
- ▶ 智能处理→?
- ▶ **未来每台计算机可能都需要一个专门的深度学习处理器**
  - ▶ 从云服务器到智能手机
  - ▶ 应用面将超过GPU：每年**数十亿**片

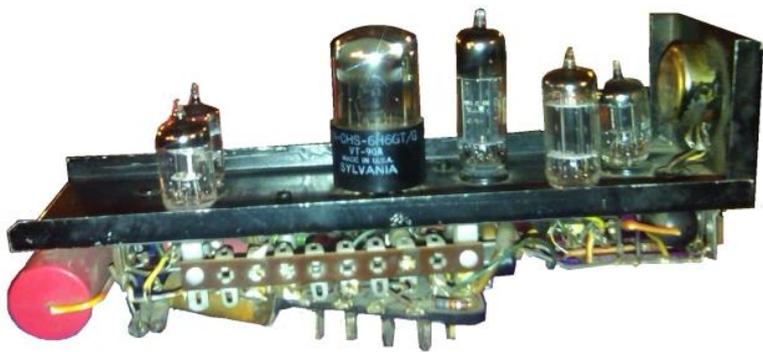
# 目录

- ▶ 深度学习处理器概述
  - ▶ 研究意义
  - ▶ 发展历史
  - ▶ 设计思路

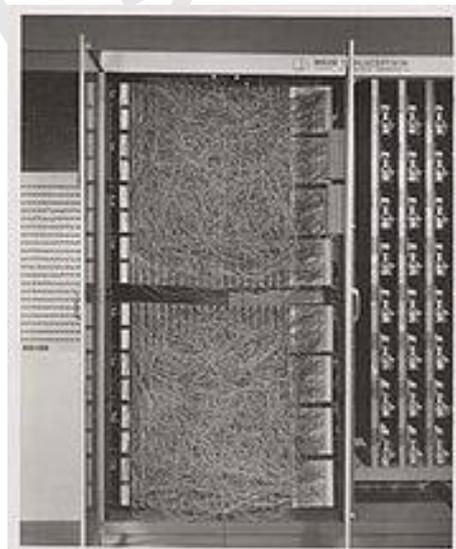
中科院计算所

# 神经网络计算机/芯片的发展历史

- ▶ 第一次热潮 (1950年代-1960年代)
  - ▶ 1951, M. Minsky研制了神经网络模拟器 SNARC
  - ▶ 1960, F. Rosenblatt研制了神经网络计算机Mark-I



SNARC



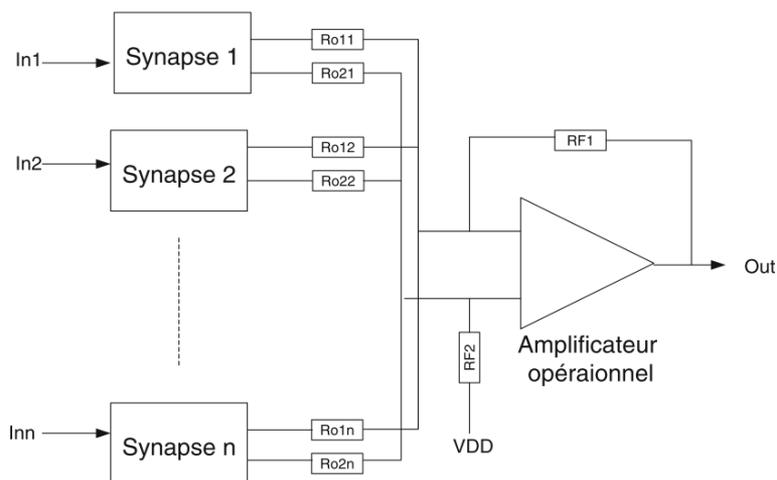
Mark-I

# 神经网络计算机/芯片的发展历史

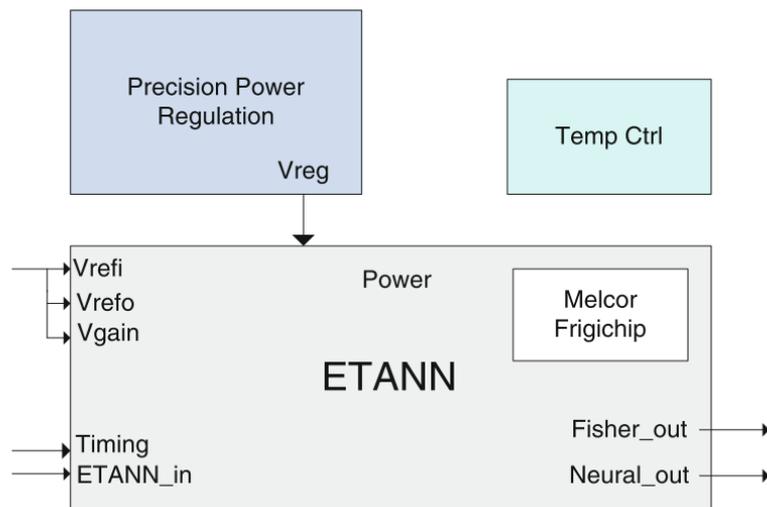
- ▶ 第一次热潮 (1950年代-1960年代)
- ▶ 第二次热潮 (1980年代-1990年代初)
  - ▶ 1989, Intel ETANN
  - ▶ 1990, CNAPS
  - ▶ 1993, MANTRA I
  - ▶ 1997, 预言神
  - ▶ .....

# 1990s的神经网络处理器

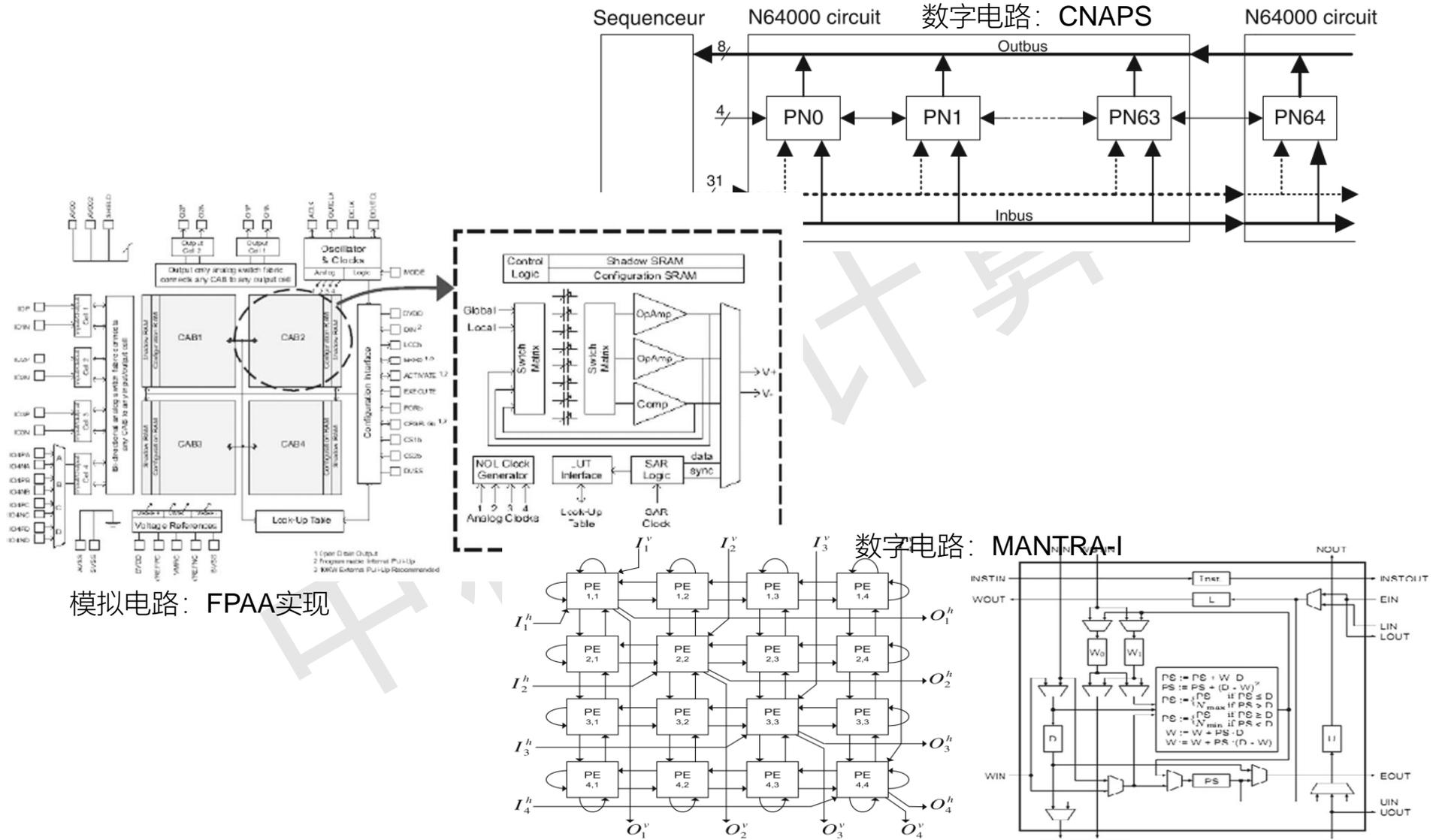
- ▶ 结构简单
- ▶ 规模小



模拟电路：Intel ETANN

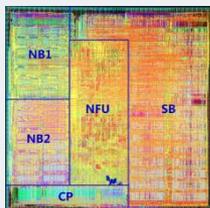


# 1990s的神经网络处理器

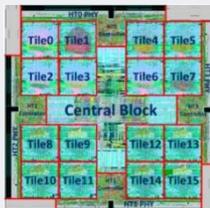


# 深度学习处理器发展

## 第三次热潮 (2006-至今)



国际首个深度学习处理器架构  
DianNao



国际首个多核深度学习处理器架构  
DaDianNao



国际首个深度学习处理器芯片



国际首个深度学习指令集奠定寒武纪生态基础



近亿台手机和服务器开始集成寒武纪处理器



国际上同期峰值速度最高的智能芯片MLU100



MLU270  
性能提升4倍

2008      2012      2013      2014      2015      2016      2017      2018      2019



可用于人工智能的GPU



Google Brain 猫脸识别  
1.6万个CPU核训练数天



首个面向深度学习的GPU架构Pascal



AlphaGo用1202个CPU+176个GPU战胜李世石



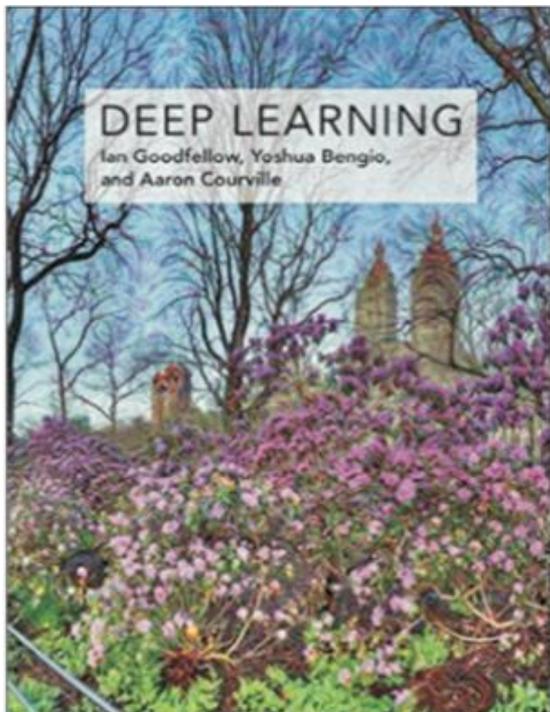
Google 公布其第一代深度学习处理器TPU



Nvidia 在其V100 GPU产品中加入深度学习加速器

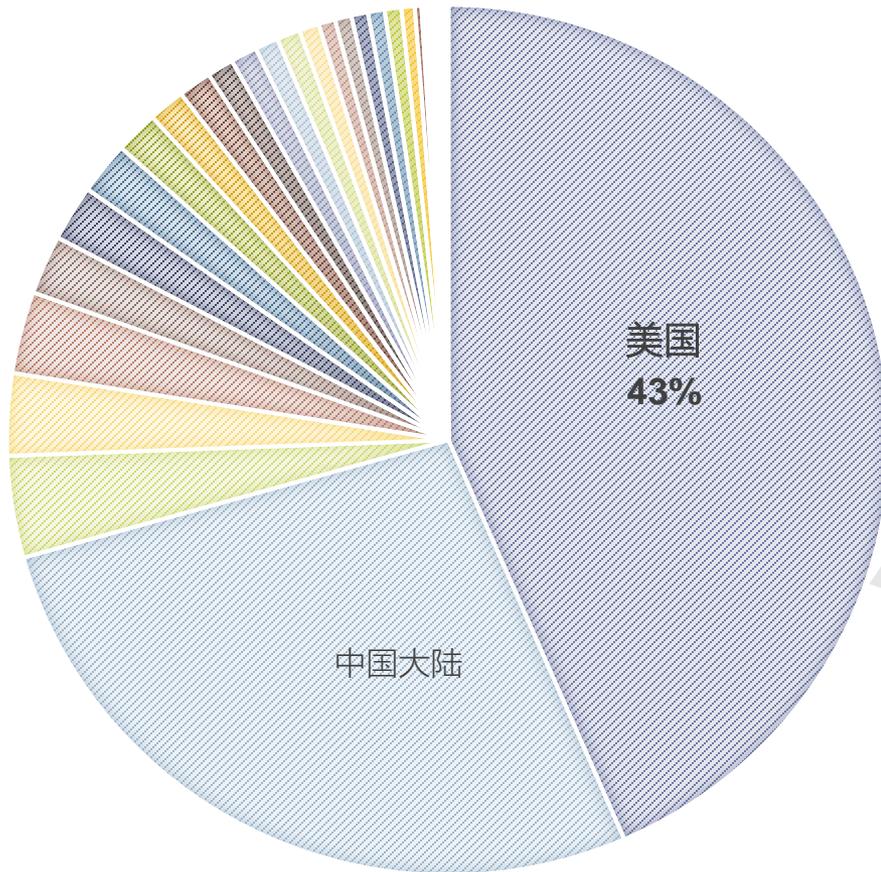
# 引领深度学习处理器学术方向

从2007起十年坚持，**开创深度学习处理器方向**，并使之成为计算机体系结构领域主要热点之一



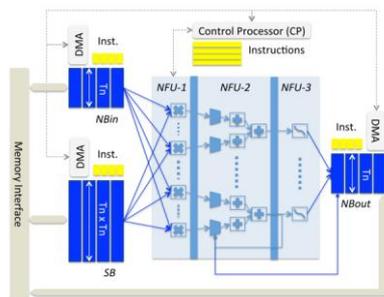
- 受2016-2018三年CCF A类会议ISCA（国际计算机体系结构年会）**近1/4论文引用**
- **约二百个**国际机构（哈佛、斯坦福、普林斯顿、哥伦比亚、MIT、UCLA、Intel、谷歌、微软等）跟踪
- **数百位**国际知名学者（2位图灵奖得主、10位中美院士、41位ACM会士、120位IEEE会士）引用
- 被深度学习开创者、图灵奖获得者Bengio的**深度学习教科书引用**

# 施引者遍及五大洲、三十个国家地区

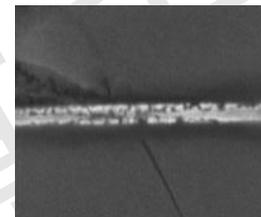


- 美国
- 中国大陆
- 韩国
- 英国
- 加拿大
- 德国
- 中国香港
- 印度
- 日本
- 以色列
- 意大利
- 俄罗斯
- 荷兰
- 中国台湾
- 芬兰
- 西班牙
- 土耳其
- 巴西
- 巴基斯坦
- 白俄罗斯
- 新加坡
- 比利时
- 丹麦
- 奥地利
- 葡萄牙
- 马来西亚
- 伊朗
- 法国
- 澳大利亚
- 塞浦路斯
- 南非

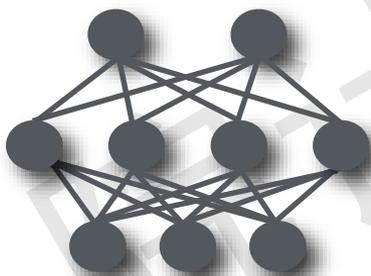
# 深度学习处理器发展的三个因素



architecture



Technology



Application

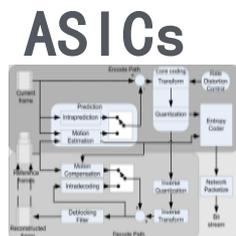
# 目录

- ▶ 深度学习处理器概述
  - ▶ 研究意义
  - ▶ 发展历史
  - ▶ 设计思路

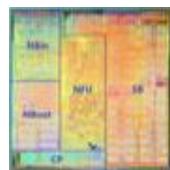
中科院计算所

# 深度学习处理器的定位

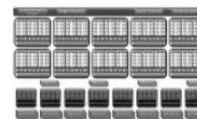
能效



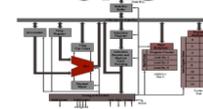
深度学习处理器



GPU



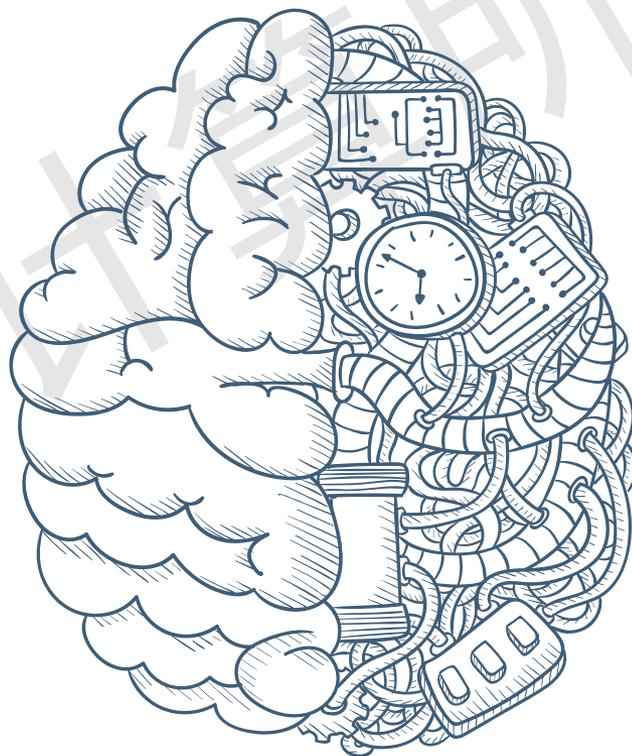
CPU



通用性

# 如何设计一个深度学习处理器DLP

- ▶ 目标?
- ▶ 体系结构?
- ▶ 微体系结构?
- ▶ 可编程性?



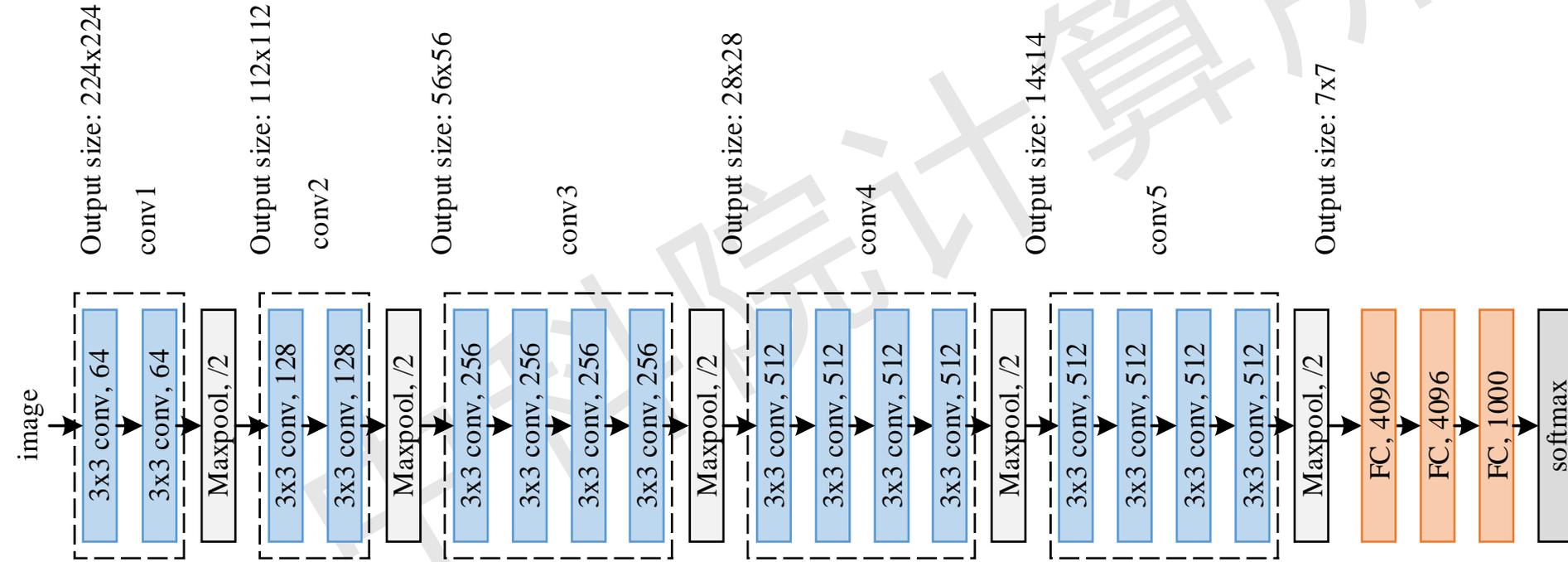
# 设计思路

- ▶ 整个体系结构最重要的问题
  - ▶ 算法范围界定
  - ▶ 算法分析（计算特性、访存特性）
- ▶ 谁是我们的“朋友”？
  - ▶ 自定制硬件，可利用算法特性
  - ▶ 高效率
- ▶ 谁是我们的“敌人”？
  - ▶ 阻碍高效率：带宽，访存速度，访存代价

# 目录

- ▶ 深度学习处理器概述
- ▶ 目标算法分析
- ▶ 深度学习处理器DLP结构
- ▶ 优化设计
- ▶ 性能评价
- ▶ 其他加速器

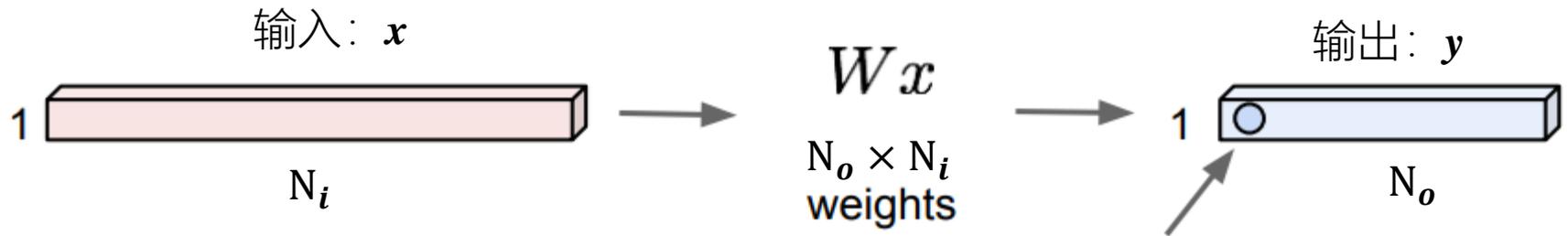
# VGG19卷积神经网络



# 分析什么

- ▶ 体系结构设计人员应该分析什么？
- ▶ 计算
  - ▶ 是否存在固定重复的计算模式
- ▶ 访存
  - ▶ 数据的局部性
  - ▶ 数据和计算的关系（对于带宽的需求）

# 全连接层



$$y[j] = G \left( b[j] + \sum_{i=0}^{N_i-1} W[j][i] \times x[i] \right)$$

\*Source from Feifei Li CS231N ([http://cs231n.stanford.edu/slides/2018/cs231n\\_2018\\_lecture05.pdf](http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf))

# 全连接层

## ▶ 代码实现

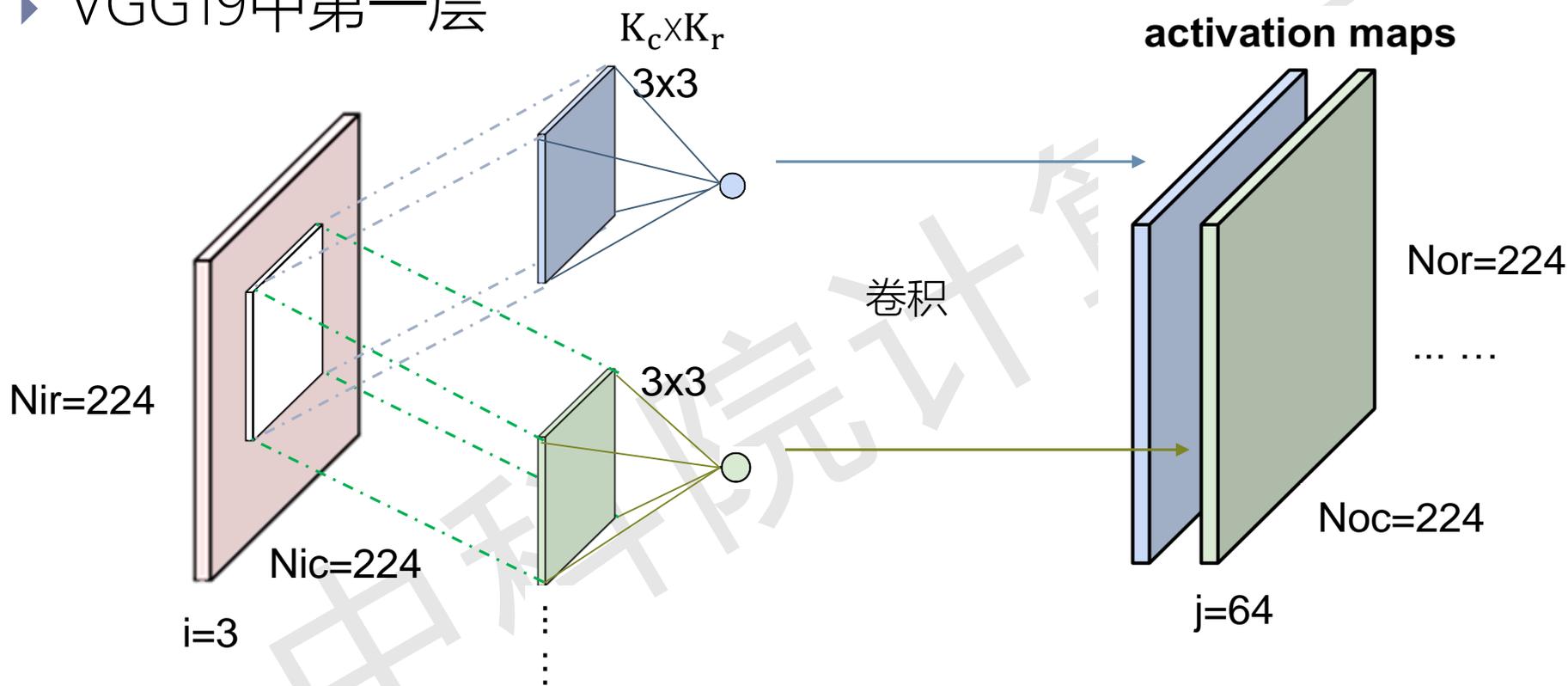
```
1 //x是输入神经元，y是输出神经元，W是权重
2 y( all ) = 0; //// 初始化所有输出神经元
3 for (j=0; j<No; j++)
4     for (i=0; i<Ni; i++){
5         y[j]+=W[j][i]*x[i];
6         if (i==Ni)
7             y[j]=G(y[j]+b[j]);
8     }
```

## ▶ 计算特点

- ▶ 向量内积、向量的元素操作
- ▶ 无复杂控制流

# 卷积层

## ► VGG19中第一层



$$Y[nor][noc][j] = G \left( \mathbf{b}[j] + \sum_{i=0}^{N_{if}-1} \sum_{k_c=0}^{K_c-1} \sum_{k_r=0}^{K_r-1} \mathbf{W}[k_r][k_c][j][i] \times \mathbf{X}[r+k_r][c+k_c][i] \right)$$

\*Source from Feifei Li CS231N ([http://cs231n.stanford.edu/slides/2018/cs231n\\_2018\\_lecture05.pdf](http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf))

# 卷积层

```
1 nor = 0;
2 for (r=0; r<Nir; r+=sr) { // sr是垂直方向的卷积步长
3     noc = 0;
4     for (c=0; c<Nic; c+=sc) { // sc是水平方向的卷积步长
5         for (j=0; j<Nof; j++)
6             sum[j]=0;
7         for (kr=0; kr<Kr; kr++)
8             for (kc=0; kc<Kc; kc++)
9                 for (j=0; j<Nof; j++)
10                    for (i=0; i<Nif; i++)
11                        sum[j]+=W[kr][kc][j][i]*X[r+kr][c+kc][i];
12         for (j=0; j<Nof; j++)
13             Y[nor][noc][j]=G(sum[j]+b[j]);
14         noc++;}
15     nor++;
16 }
```

# 卷积层

```
1 nor = 0;
2 for (r=0; r<Nir; r+=sr) { // sr是垂直方向的卷积步长
3     noc = 0;
4     for (c=0; c<Nic; c+=sc) { // sc是水平方向的卷积步长
5         for (j=0; j<Nof; j++)
6             sum[j]=0;
7         for (kr=0; kr<Kr; kr++)
8             for (kc=0; kc<Kc; kc++)
9                 for (j=0; j<Nof; j++)
10                    for (i=0; i<Nif; i++)
11                        sum[j]+=W[kr][kc][j][i]*X[r+kr][c+kc][i];
12     }
13 }
14
```

## ▶ 计算特点

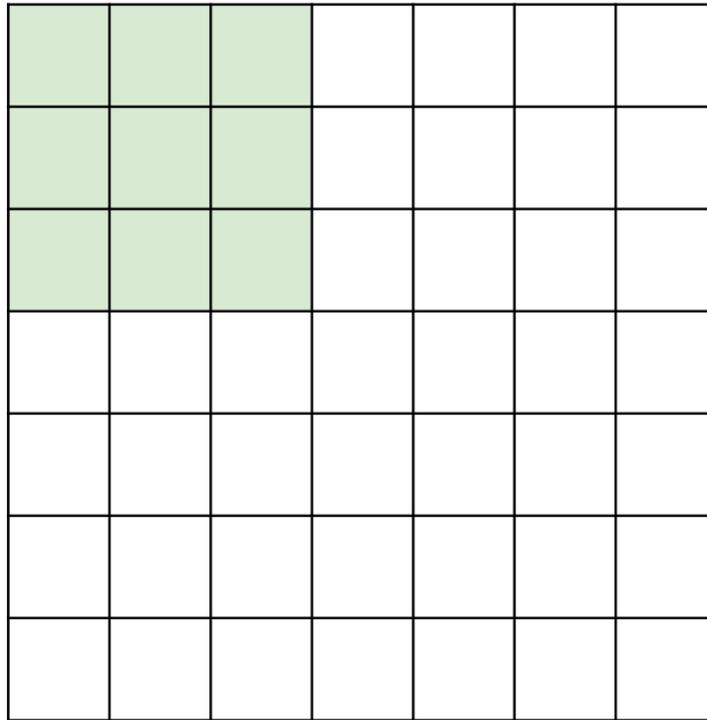
- ▶ 矩阵内积、向量的元素操作
- ▶ 无复杂控制流

# 卷积层

具体看看单个卷积是怎么做的

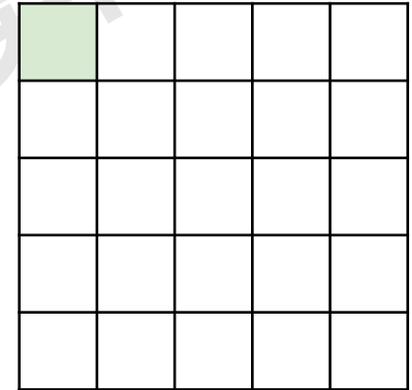
7

7x7输入图像  
3x3卷积核  
步长为1



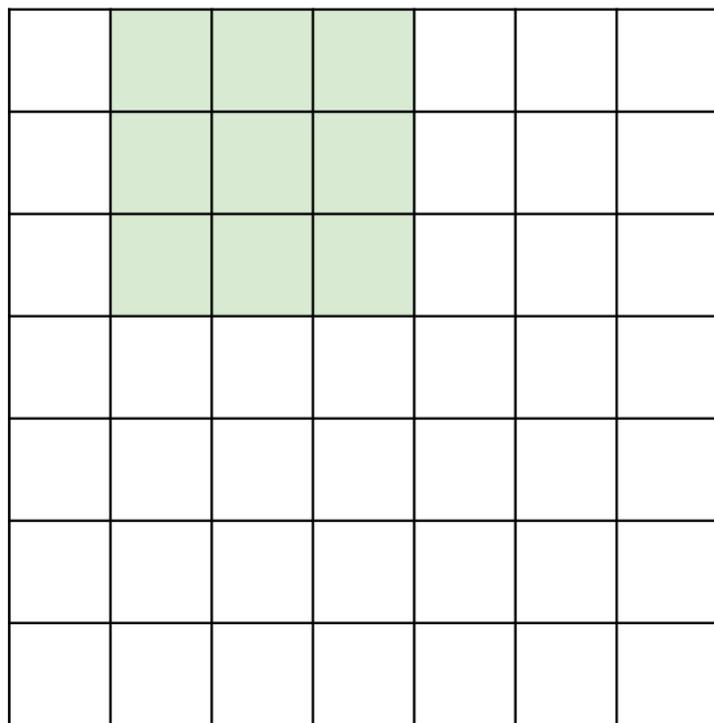
输出大小是5x5

7



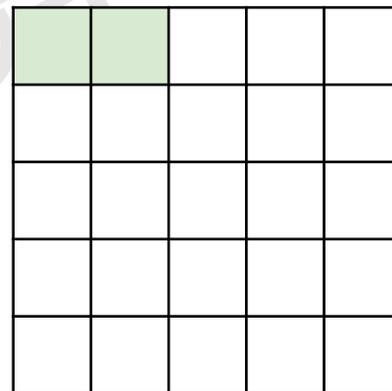
# 卷积层

7



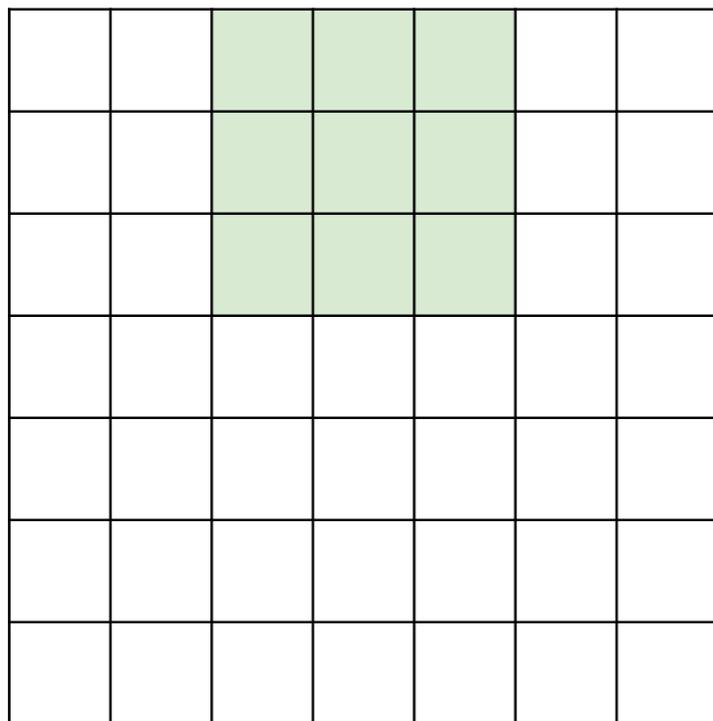
输出大小是5x5

7



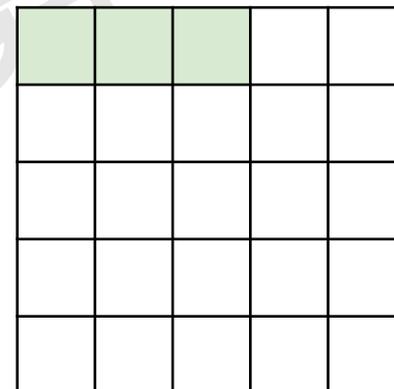
# 卷积层

7



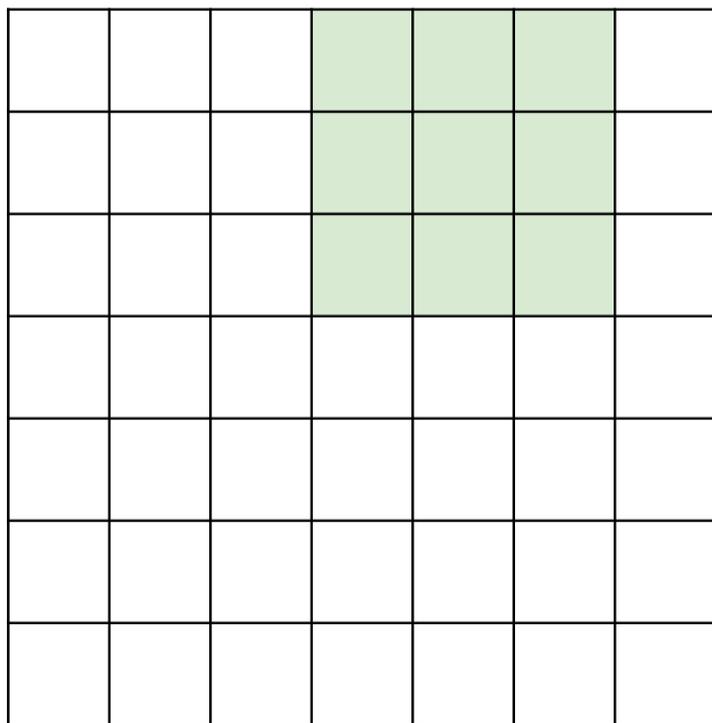
输出大小是5x5

7



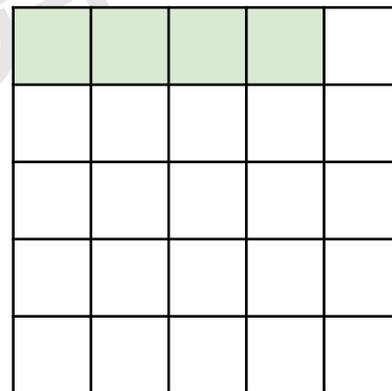
# 卷积层

7



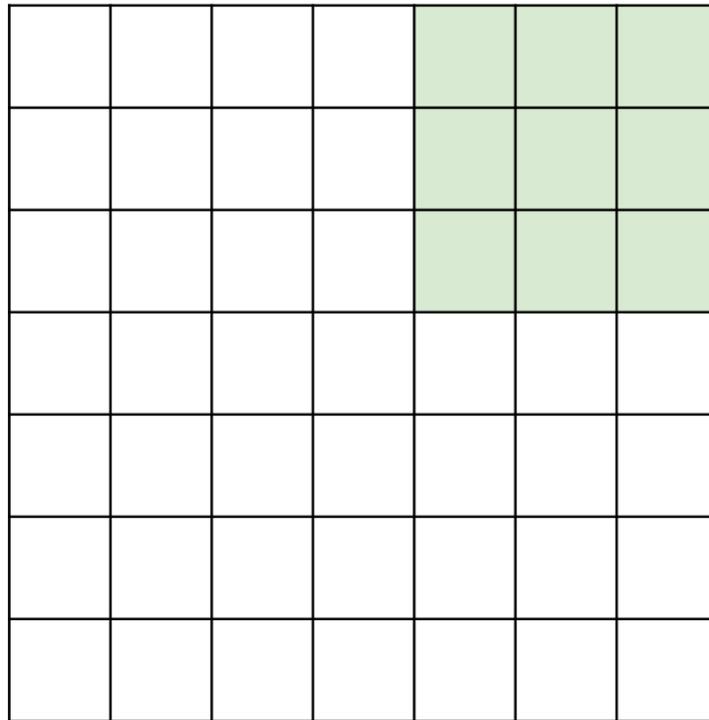
输出大小是5x5

7



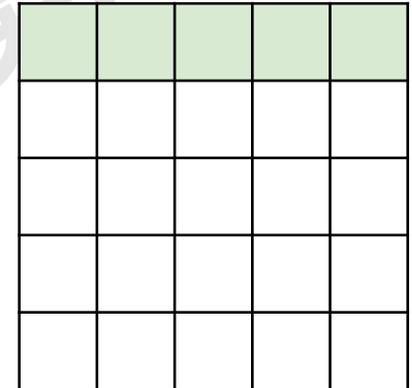
# 卷积层

7



输出大小是5x5

7

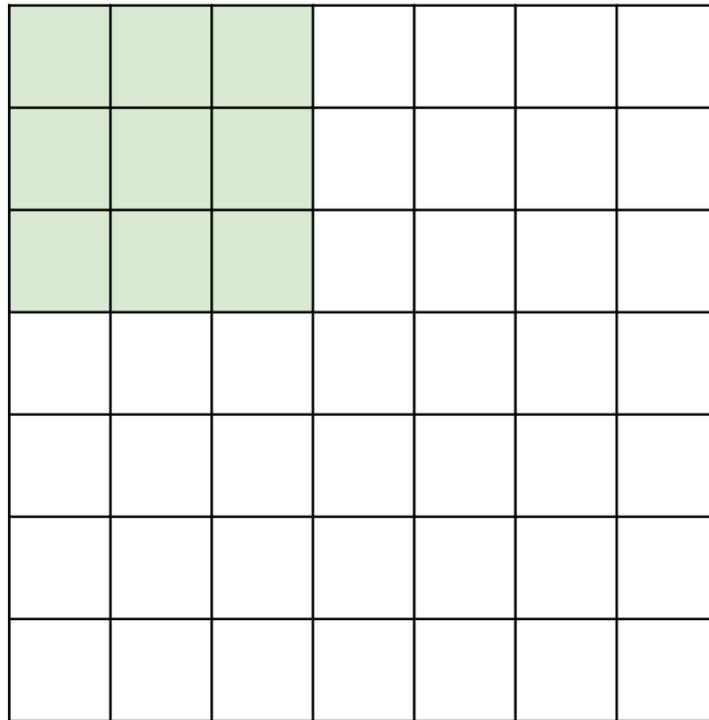


\*Source from Feifei Li CS231N ([http://cs231n.stanford.edu/slides/2018/cs231n\\_2018\\_lecture05.pdf](http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf))

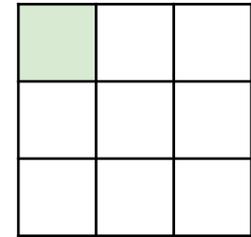
# 卷积层

7

7x7输入图像  
3x3卷积核  
步长为2



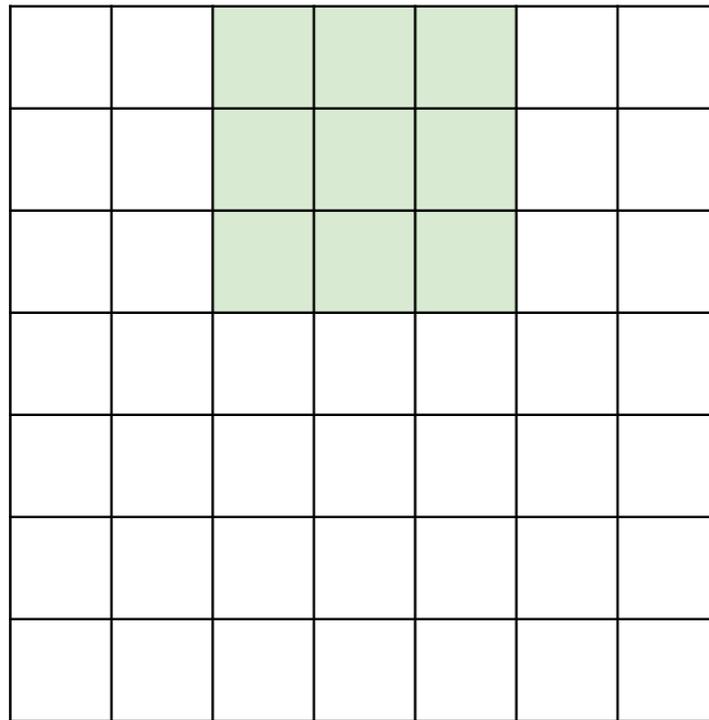
输出大小是3x3



# 卷积层

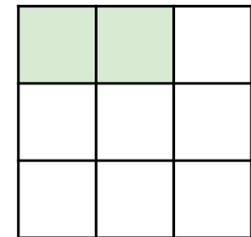
7

7x7输入图像  
3x3卷积核  
步长为2



7

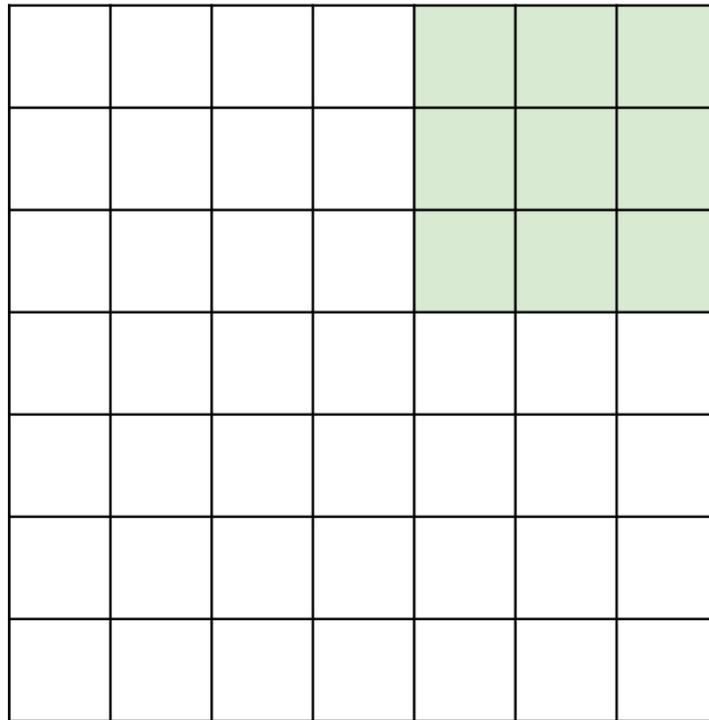
输出大小是3x3



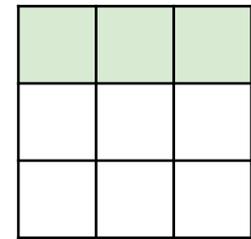
# 卷积层

7

7x7输入图像  
3x3卷积核  
步长为2



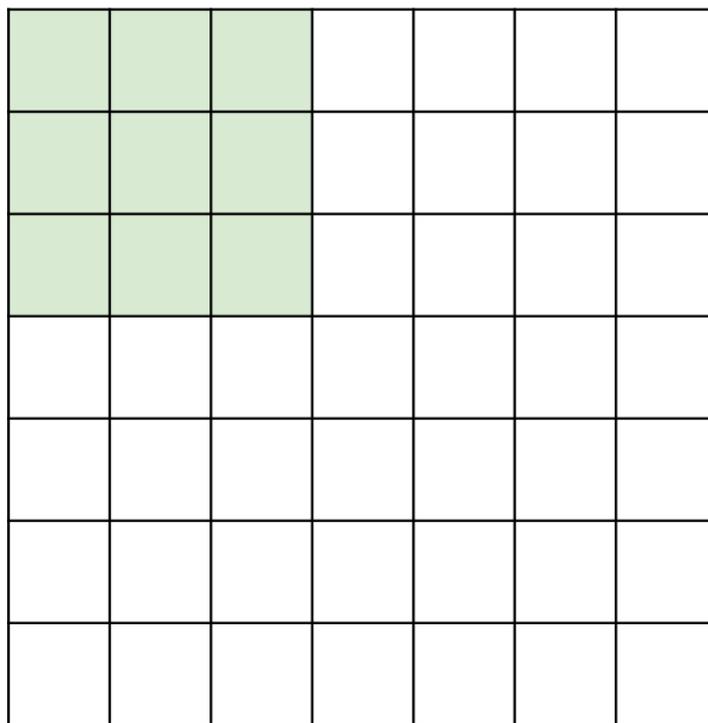
输出大小是3x3



7

# 卷积层

7

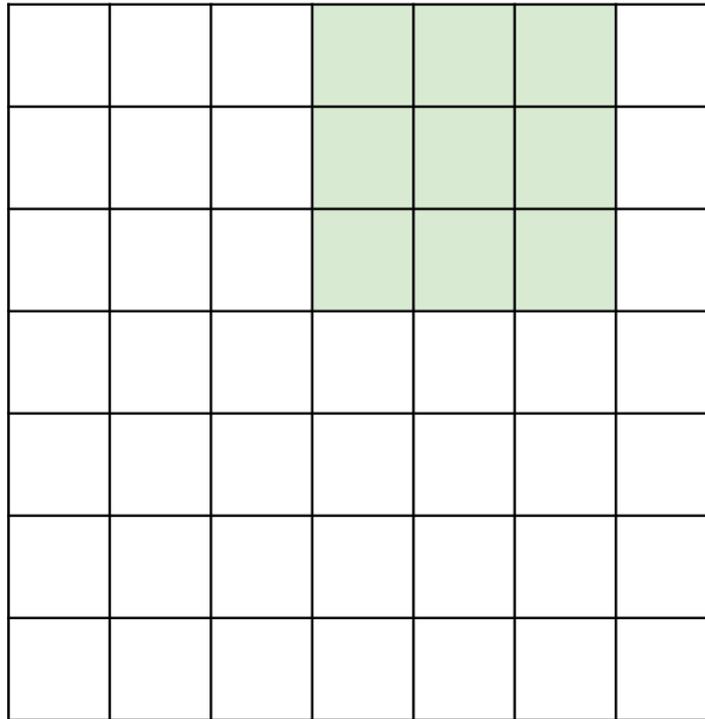


7x7输入图像  
3x3卷积核  
步长为3

7

# 卷积层

7

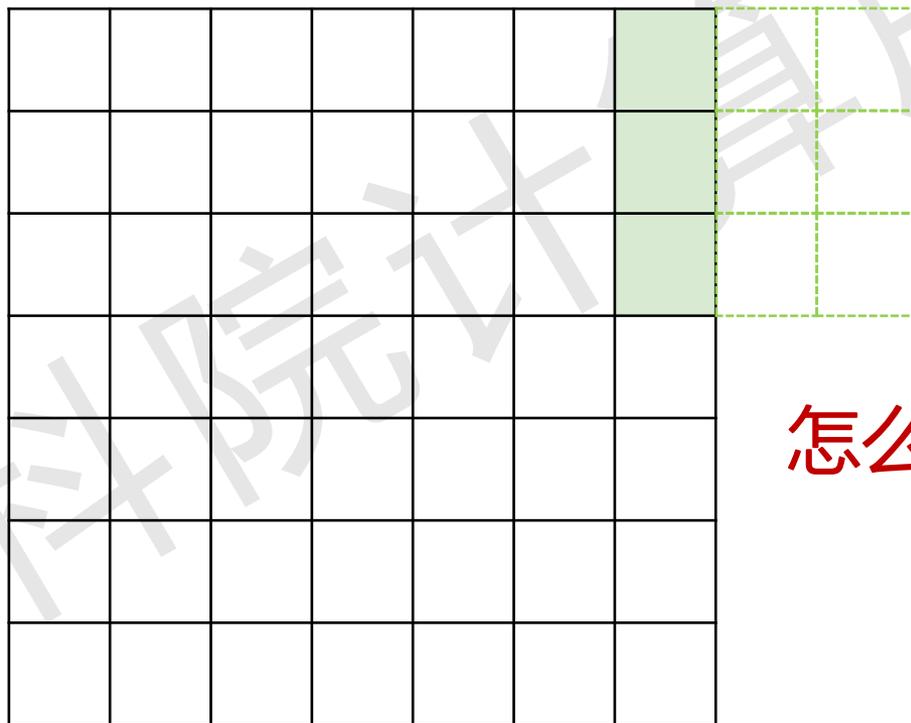


7x7输入图像  
3x3卷积核  
步长为3

7

# 卷积层

7x7输入图像  
3x3卷积核  
步长为3



# 卷积层

7x7输入图像  
3x3卷积核  
步长为3

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

**Zero pad**

# 卷积层

7x7输入图像  
3x3卷积核  
步长为3

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

**Zero pad**

# 卷积层

7x7输入图像  
3x3卷积核  
步长为3

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

**Zero pad**

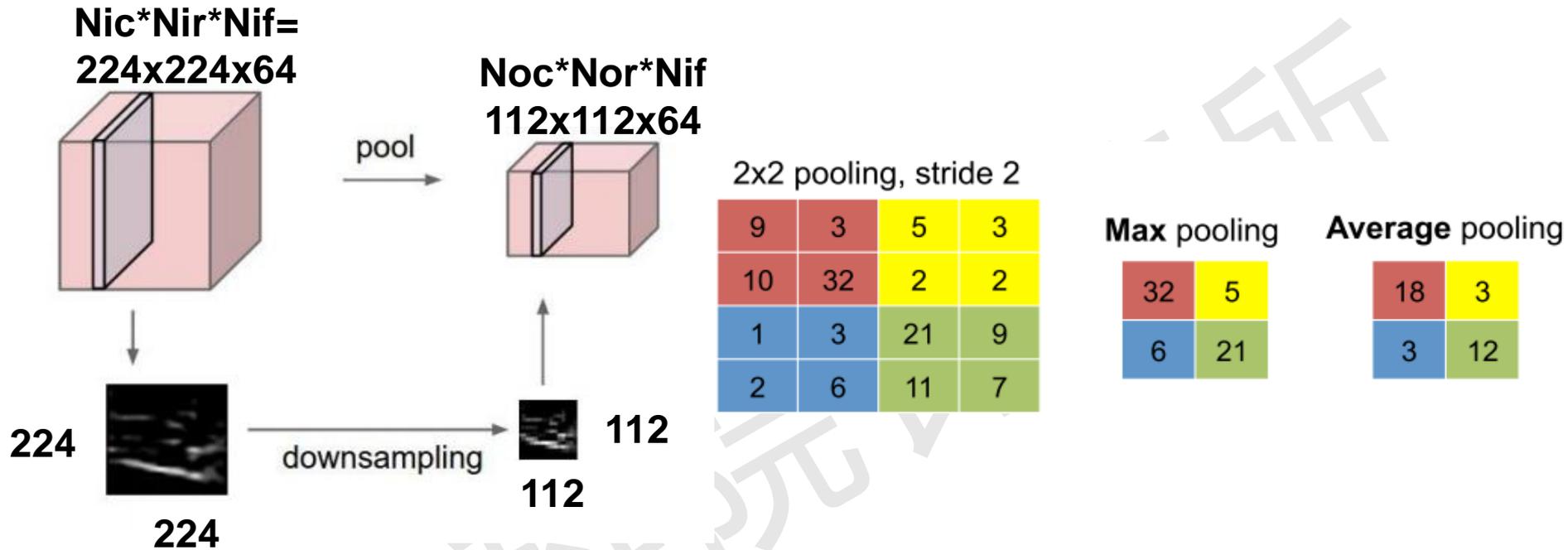
# 卷积层

7x7输入图像  
3x3卷积核  
步长为3

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

**Zero pad**

# 池化层



MAX pooling: 
$$Y[nor][noc][i] = \max_{0 \leq kc < K_c, 0 \leq kr < K_r} (X[r + kr][c + kc][i])$$

AVG pooling: 
$$Y[nor][noc][i] = \frac{1}{K_c \times K_r} \sum_{k_c=0}^{K_c-1} \sum_{k_r=0}^{K_r-1} X[r + kr][c + kc][i]$$

\*Source from Feifei Li CS231N ([http://cs231n.stanford.edu/slides/2018/cs231n\\_2018\\_lecture05.pdf](http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf))

# 池化层

```
1 nor = 0;
2 for (r=0; r<Nir; r+=sr) { // sr是垂直方向的池化步长
3     noc = 0;
4     for (c=0; c<Nic; c+=sc) { // sc是水平方向的池化步长
5         for (i=0; i<Nif; i++)
6             value[i]=0;
7         for (kr=0; kr<Kr; kr++)
8             for (kc=0; kc<Kc; kc++)
9                 for (i=0; i<Nif; i++) {
10                    // for average pooling
11                    value[i]+=X[r+kr][c+kc][i];
12                    // for max pooling
13                    value[i] = max(value[i], X[r+kr][c+kc][i]);}
14
15         for (i=0; i<Nif; i++)
16             // for average pooling
17             Y[nor][noc][i]=value[i]/Kr/Kc;
18             // for max pooling
19             Y[nor][noc][i]=value[i];
20     noc++;}
21 nor++;
}
```

# 池化层

```
1 nor = 0;
2 for (r=0; r<Nir; r+=sr) { // sr是垂直方向的池化步长
3     noc = 0;
4     for (c=0; c<Nic; c+=sc) { // sc是水平方向的池化步长
5         for (i=0; i<Nif; i++)
6             value[i]=0;
7         for (kr=0; kr<Kr; kr++)
8             for (kc=0; kc<Kc; kc++)
9                 for (i=0; i<Nif; i++) {
10                    // for average pooling
11                    value[i]+=X[r+kr][c+kc][i];
12                    // for max pooling
13                    value[i] = max(value[i], X[r+kr][c+kc][i]);}
14                for (i=0; i<Nif; i++)
```

## ▶ 计算特点

- ▶ 向量的元素操作
- ▶ 无复杂控制流

# 卷积神经网络

- ▶ VGG19: 在风格迁移算法中使用

	VGG19
参数	1.14 (亿)
层类型	卷积, 池化, 全连接
计算过程	简洁
层数	25 (19+6)
卷积层	16 (3x3卷积核, 图大小不变)
池化层	5 (Max Pooling)
全连接层	3

E	
19 weight layers	
input (224 × 224 RGB image)	
conv3-64	conv3-64
maxpool	
conv3-128	conv3-128
maxpool	
conv3-256	conv3-256
conv3-256	<b>conv3-256</b>
maxpool	
conv3-512	conv3-512
conv3-512	<b>conv3-512</b>
maxpool	
conv3-512	conv3-512
conv3-512	<b>conv3-512</b>
maxpool	
FC-4096	
FC-4096	
FC-1000	
soft-max	

# 卷积神经网络

## 计算特征

### 不同层的计算特点

层	计算类型	乘加操作个数	激活函数操作个数
卷积层	矩阵内积，向量的元素操作	$N_{if} \times N_{of} \times N_{or} \times N_{oc} \times K_r \times K_c$ 个乘加	$N_{of} \times N_{or} \times N_{oc}$
池化层	向量的元素操作	$N_{if} \times N_{or} \times N_{oc} \times K_r \times K_c$ 个加法或比较 $+N_{if} \times N_{or} \times N_{oc}$ 个除法操作（平均池化）	无
全连接层	矩阵乘向量，向量的元素操作	$N_o \times N_i$ 个乘加	$N_o$

计算指令  
硬件加速单元

# 卷积神经网络

## ▶ 访存特征

全连接层:

```
1 //x是输入神经元, y是输出神经元, W是权重
2 y(all) = 0; //// 初始化所有输出神经元
3 for (j=0; j<No; j++)          外循环
4     for (i=0; i<Ni; i++){      内循环
5         y[j]+W[j][i]*x[i];
6         if (i==Ni)
7             y[j]=G(y[j]+b[j]);
8     }
```

x[i] 外循环复用, 复用距离等于Ni

W[j][i] 内外循环无复用

y[j]+ 内循环复用, 复用距离等于1



可解耦性  
可复用性

# 卷积神经网络

## 访存特征

VGG19最后两层

maxpool
FC-4096
FC-4096
FC-1000
soft-max

权值参数数量:

FC-4096:  $4096 * 4096 = 16777216$

FC-1000:  $4096 * 1000 = 4096000$

参数大小 (浮点数):

FC-4096:  $16777216 * 4 \text{ Byte} = 64\text{MB}$

FC-1000:  $4096000 * 4 \text{ Byte} = 15.625\text{MB}$

Intel Xeon 6130的L3 cache大小: 22MB

Nvidia V100片上L2 cache: 6MB

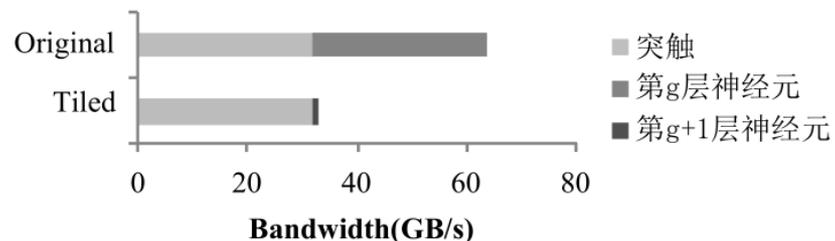
权值数据数据量大无复用, 带宽需求高, 需要进行循环分块

# 卷积神经网络

## 访存特征

循环分块 (tiling)

```
1 //T是循环分块大小
2 y(all) = 0; //初始化所有输出神经元
3 for (ii=0; ii<Ni; ii+=Ti)
4     for (j=0; j<No; j++)
5         for (i=ii; i<ii+Ti; i++){
6             y[j]+=W[j][i]*x[i];
7             if (i==Ni)
8                 y[j]=G(y[j]+b[j]);
9         }
```



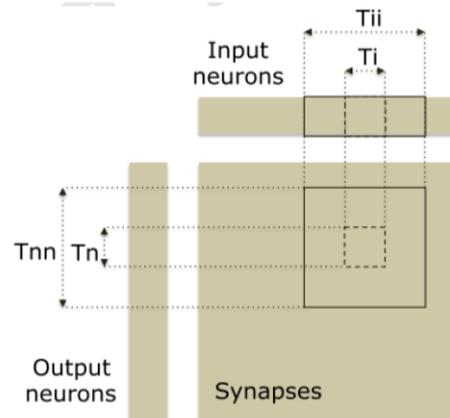
假设 $N_i=16384$   
循环分块能减少46.7%的访存带宽需求

# 卷积神经网络

## 访存特征

### 全连接层做两层循环分块 (tiling)

```
1 for (jjj=0; jjj <No; jjj +=Tjj) { //对输出神经元进行分块, Tjj和Tj是两层分块大小
2   for (jj=jjj; jj <jjj+Tjj; jj +=Tj) {
3     for (j=jj; j<jj+Tj; j++)
4       y[j]=0;
5     for (iii=0; iii <Ni; iii +=Tii) { //对输入神经元进行分块, Tii和Ti是两层分块大小
6       for (ii=iii; ii <iii+Tii; ii +=Ti)
7         for (j=jj; j<jj+Tj; j++)
8           for (i=ii; i<ii+Ti; i++)
9             sum[j] += W[j][i]*x[i];}
10    for (j=jj; j<jj+Tj; j++)
11      y[j]=G(sum[j]+b[j]);
12  }
```



# 卷积神经网络

## ▶ 访存特征

### 卷积层做循环分块 (tiling)

```
1 for (rr=0; rr<Nir; rr+=Tr) { //对输入特征图的垂直方向进行分块
2   for (cc=0; cc<Nic; cc+=Tc) { //对输入特征图的水平方向进行分块
3     for (jjj=0; jjj <Nof; jjj+=Tjj) { //对输出特征图的通道进行分块, Tjj为外层循环分块大小
4       nor = 0;
5       for (r=rr; r<rr+Tr; r+=sr) {
6         noc = 0;
7         for (c=cc; c<cc+Tc; c+=sc) {
8           for (jj=jjj; jj <jjj+Tjj; jj+=Tj) { //对输出特征图的通道进一步分块, Tj为内层循环分块大小
9             for (j=jj; j<jj+Tj; j++)
10              sum[j]=0;
11             for (kr=0; kr<Kr; kr++)
12              for (kc=0; kc<Kc; kc++)
13                for (ii=0; ii<Nif; ii+=Ti) //对输入特征图的通道进行分块
14                  for (j=jj; j<jj+Tj; j++)
15                    for (i=ii; i<ii+Ti; i++)
16                      sum[j]+=W[kr][kc][j][i]*X[r+kr][c+kc][i];
17             for (j=jj; j<jj+Tj; j++)
18               Y[nor][noc][j]=G(sum[j]+b[j]);
19             noc++;
20           nor++;
21         }
      }
    }
  }
}
```

# 卷积神经网络

## ▶ 访存特征

### 池化层做循环分块 (tiling)

```
1  for (rr=0; rr<Nir; rr+=Tr) { //对输入特征图的垂直方向进行分块
2      for (cc=0; cc<Nic; cc+=Tc){ //对输入特征图的水平方向进行分块
3          for (iii=0; iii<Ni; iii+=Tii){ //对输入特征图通道进行分块，Tii为外层循环分块
4              大小
5                  nor = 0;
6                  for (r=rr; r<rr+Tr; r+=sr){
7                      noc = 0;
8                      for (c=cc; c<cc+Tc; c+=sc){
9                          for (ii=iii; ii<iii+Tii; ii+=Ti){ //对输入特征图的通道进一步分块，Ti
10                             为内层循环分块大小
11                                 for (i=ii; i<ii+Ti; i++)
12                                     value[i]=0;
13                                 for (kr=0; kr<Kr; kr++)
14                                     for (kc=0; kc<Kc; kc++)
15                                         for (i=ii; i<ii+Ti; i++) {
16                                             // for average pooling
17                                             value[i]+=X[r+kr][c+kc][i];
18                                             // for max pooling
19                                             value[i] = max(value[i], X[r+kr][c+kc][i]);
20                                 for (i=ii; i<ii+Ti; i++)
21                                     // for average pooling
22                                     Y[noc][nor][i]=value[i]/Kr/Kc;
23                                     // for max pooling
24                                     Y[noc][nor][i]=value[i];
25                                 noc++;}
26                             nor++;
27                         }
28                     }
29                 }
30             }
31         }
32     }
```

# 卷积神经网络

## ▶ 访存特征

不同层的重用特性

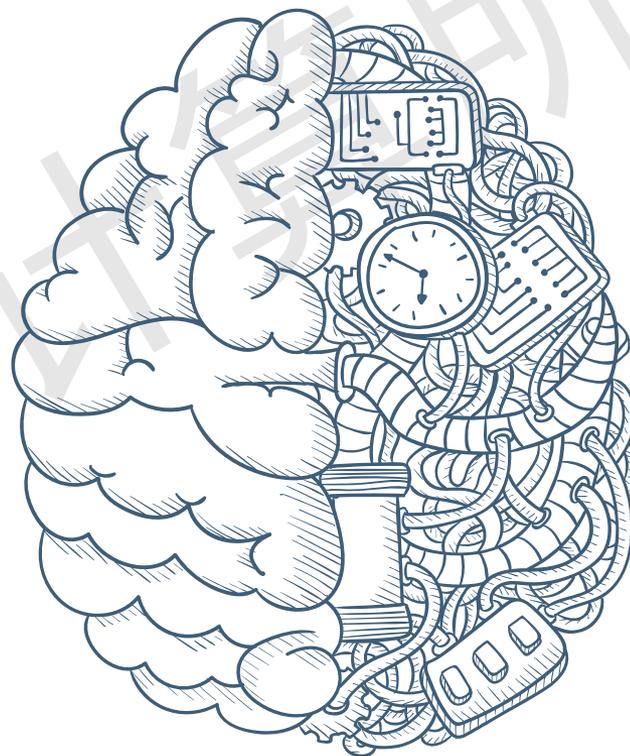
层	可重用	不可重用
卷积层	输入神经元、输出神经元、突触权重	无
池化层	当池化窗口大于步长，部分输入神经元可重用	池化窗口小于等于步长时，输入神经元、输出神经元都不可重用
全连接层	输入神经元、输出神经元	突触权重

# 目录

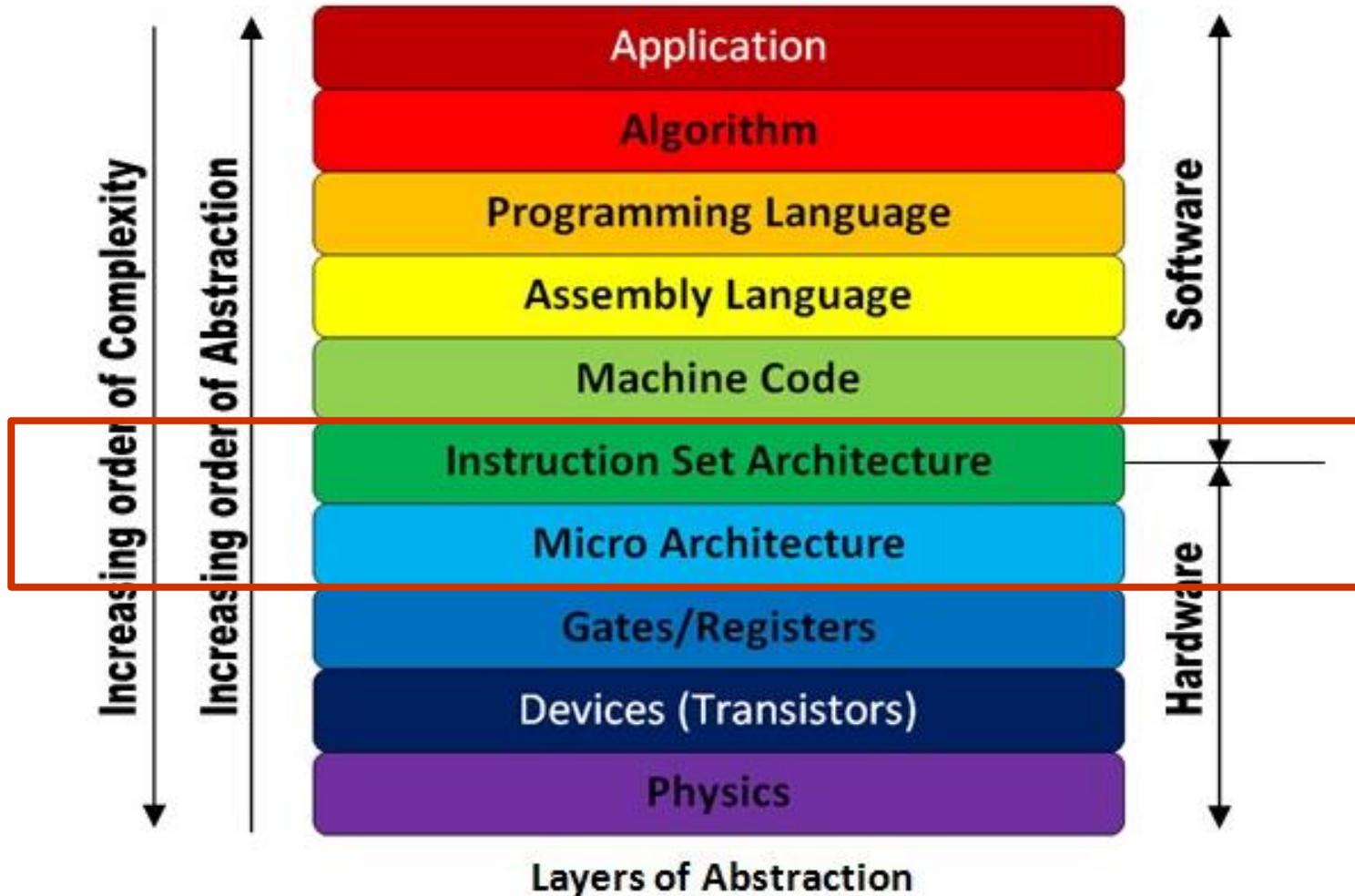
- ▶ 深度学习处理器概述
- ▶ 目标算法分析
- ▶ 深度学习处理器DLP结构
- ▶ 优化设计
- ▶ 性能评价
- ▶ 其他加速器

# 深度学习处理器DLP结构

- ▶ 深度学习专用
- ▶ Deep Learning Processor
  - ▶ i.e., DLP
- ▶ DLP结构
  - ▶ 指令集
  - ▶ 流水线
  - ▶ 运算部件
  - ▶ 访存部件
  - ▶ 算法映射



# 体系结构层次



# 指令集

- ▶ 计算机的抽象模型
- ▶ 定义了体系结构
- ▶ 软硬件的唯一接口
  
- ▶ 为什么采用指令集
  - ▶ 灵活性：支持未来可能出现的新的深度学习算法
  - ▶ 通用性：支持广泛的深度学习算法

# 指令集

## ▶ 设计原则

- ▶ Data-Level Parallelism
- ▶ 可向量化操作

## 不同层的计算特点

层	计算类型	乘加操作个数	激活函数操作个数
卷积层	矩阵内积, 向量的元素操作	$N_{if} \times N_{of} \times N_{or} \times N_{oc} \times K_r \times K_c$ 个乘加	$N_{of} \times N_{or} \times N_{oc}$
池化层	向量的元素操作	$N_{if} \times N_{or} \times N_{oc} \times K_r \times K_c$ 个加法或比较 + $N_{if} \times N_{or} \times N_{oc}$ 个除法操作 (平均池化)	无
全连接层	矩阵乘向量, 向量的元素操作	$N_o \times N_i$ 个乘加	$N_o$

# 指令集

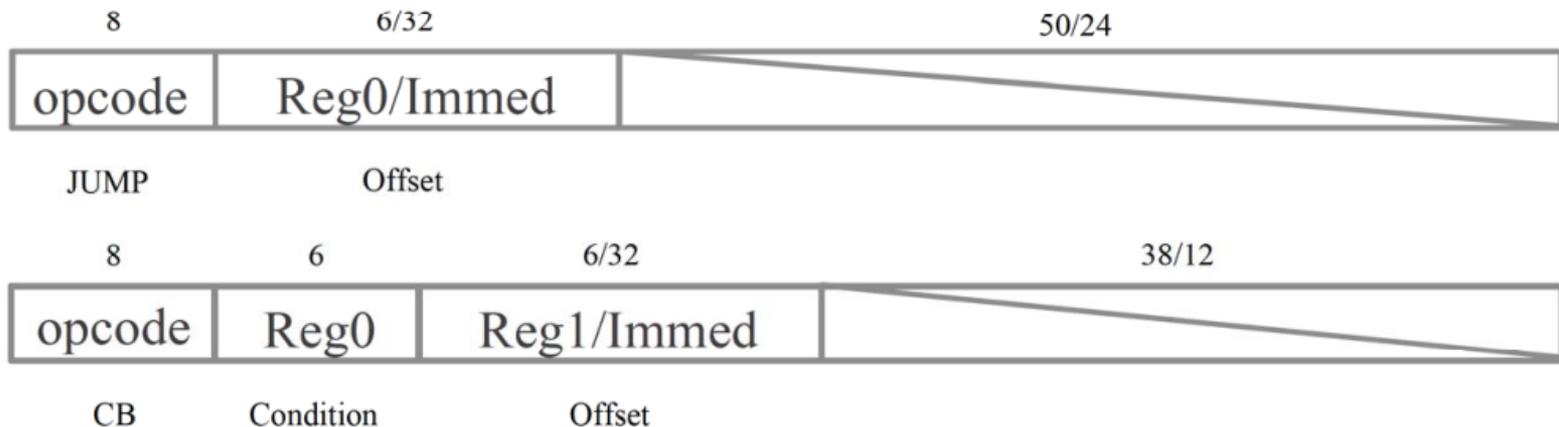
## ▶ DLP指令集

指令类型	例子	操作对象
控制指令	跳转(JUMP), 条件分支(CB)	寄存器(标量值), 立即数
数据转移指令	矩阵(Matrix)	矩阵 取(MLOAD)/存(MTORE)/移动(MMOVE)
	向量(Vector)	向量 取(VLOAD)/存(VSTORE)/移动(VMOVE)
	标量(Scalar)	标量 取(SLOAD)/存(SSTORE)/移动(SMOVE)
计算指令	矩阵(Matrix)	矩阵乘向量(MMV), 向量乘矩阵(VMM), 矩阵乘标量(MMS), 外积(OP), 矩阵相加(MAM), 矩阵相减(MSM)
	向量(Vector)	向量基本运算(加(VAV)、减(VSV)、乘(VMV)、除(VDV)), 向量超越函数(指数(VEXP)、对数(VLOG)), 内积(IP), 随机向量(RV), 向量最大值(VMAX), 向量最小值(VMIN)
	标量(Scalar)	标量基本运算, 标量超越函数
逻辑运算指令	向量(Vector)	向量比较(大于(VGT),等于(VE)),向量逻辑操作(与(VAND), 或(VOR), 取反(VNOT)), 向量最值归约(VGTM)
	标量(Scalar)	标量比较, 标量逻辑运算

Load-store结构: 只通过load和store指令访问主存  
64-bit定长指令, 变长操作数 (寄存器指定长度)

# 指令集

- ▶ 控制指令
  - ▶ JUMP: 立即跳转指令
  - ▶ CB: 条件分支指令



# 指令集

## ▶ 数据传输指令

▶ Load/Store指令：主存和片上存储交互

▶ MLOAD/MSTORE：矩阵数据（变长）

▶ VLOAD/VSTORE：向量数据（变长）

▶ SLOAD/SSTORE：标量数据

▶ MOVE指令：片上数据传输

▶ MMOVE, VMOVE, SMOVE

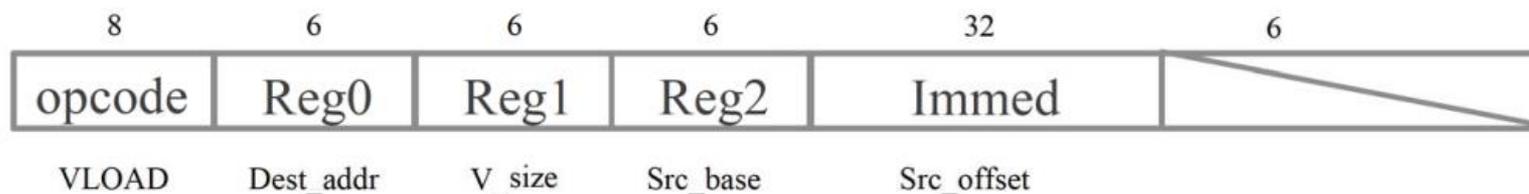
命名方式：

**M**LOAD

**Matrix**-Load

**V**LOAD

**Vector**-Load



# 指令集

## ▶ 计算指令

命名方式:

MMV

Matrix-Multiply-Vector

### ▶ 矩阵运算:

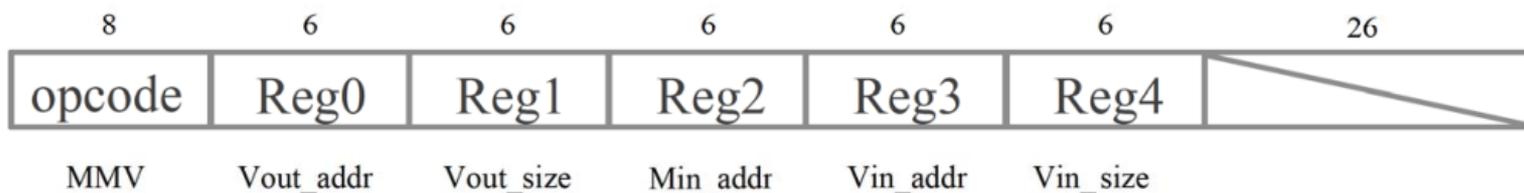
- ▶ MMV, VMM, MMS, OP (外积), MAM, MSM

### ▶ 向量运算:

- ▶ VAV, VSV, VMV, VDV, VEXP (向量指数), VLOG (向量对数), IP (内积), RV (随机向量生成), VMAX/VMIN (向量最值)

### ▶ 标量运算:

- ▶ 加减乘除基本运算, 标量超越函数



# 指令集

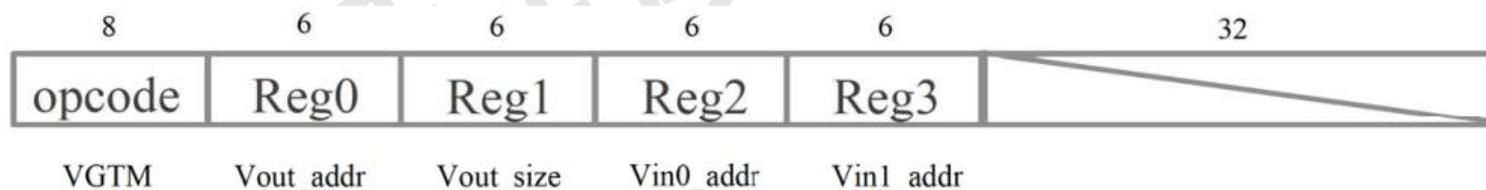
## ▶ 逻辑指令

### ▶ 向量逻辑:

- ▶ 比较 (VGT, VE) , 逻辑 (VAND, VOR, VNOT) , 最值归约VGTM

### ▶ 标量逻辑:

- ▶ 标量比较, 标量逻辑运算



最值归约:  $Vout[i] = (Vin0[i] > Vin1[i])?Vin0[i] : Vin1[i]$

# DLP代码示例

## ▶ 全连接层

```
1 Fully connection code:
2 // $0: 输入特征向量大小, $1: 输出特征向量大小, $2: 权重矩阵大小
3 // $3: 输入特征向量地址, $4: 权重地址
4 // $5: 偏置地址, $6: 输出特征向量地址
5 // $7-$10: 临时变量地址
6 VLOAD $3, $0, #100 //从地址 (100) 读取向量数据到片上存储地址 $3
7 MLOAD $4, $2, #300 //从地址 (300) 读取权重矩阵到片上存储地址 $4
8 MMV $7, $1, $4, $3, $0 //  $Wx$ 
9 VAV $8, $1, $7, $5 //  $t = Wx + b$ 
10 VEXP $9, $1, $8 //  $e^t$ 
11 VAS $10, $1, $9, #1 //  $1 + e^t$ 
12 VDV $6, $1, $9, $10 //  $y = e^t / (1 + e^t)$ 
13 VSTORE $6, $1, #200 // 输出向量存入地址 (200)
```

# DLP代码示例

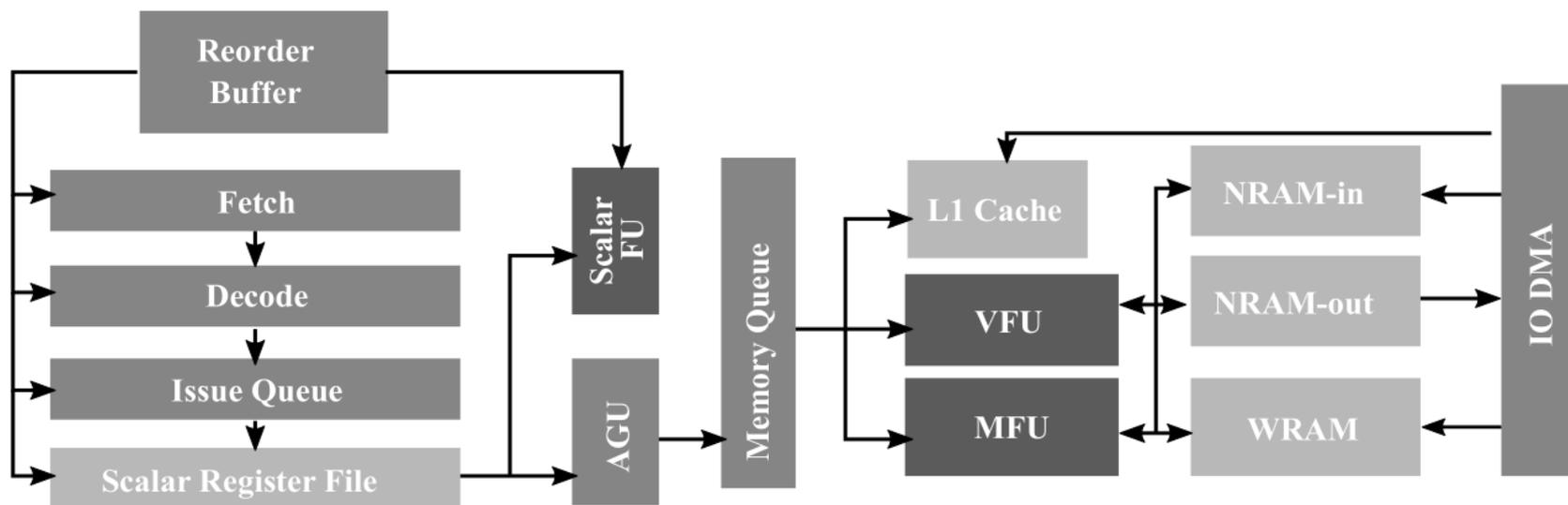
## ► 池化层

```
15 Pooling code:
16 // $0: 特征通道数  $N_i$ , $1: 输入特征图大小  $N_{ic} \times N_{ir} \times N_i$ 
17 // $2: 输出数据大小, $3: 池化窗口大小-1
18 // $4: 水平方向循环次数, $5: 垂直方向循环次数
19 // $6: 输入地址, $7: 输出地址
20 // $8: 输入特征图垂直方向步长
21     VLOAD    $6, $1, #100    //从地址 (100) 读取输入特征图到片上存储地址 $6
22     SMOVE    $5, $3          //初始化  $\$5 = K_r - 1$ 
23 L0: SMOVE    $4, $3          //初始化  $\$4 = K_c - 1$ 
24 L1: VGIM     $7, $0, $6, $7
25 // feature map m, output[m]=(input[c][r][m]>output[m])?
26 // input[c][r][m]:output[m]
27     SADD     $6, $6, $0    //更新输入向量地址
28     SADD     $4, $4, #-1   // c-
29     CB       #L1, $4      // if(c>0) goto L1
30     SADD     $6, $6, $8    //更新输入向量地址
31     SADD     $5, $5, #-1   // r-
32     CB       #L0, $5      // if(r>0) goto L0
33     VSTORE   $7, $2, #200 //输出向量存入地址 (200)
```

# 流水线

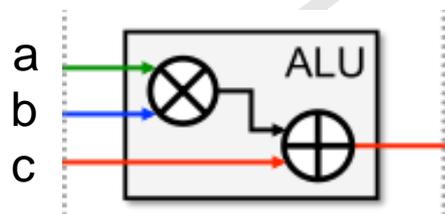
## ▶ 7段流水

- ▶ 取指、译码、发射、读寄存器、执行、写回、提交

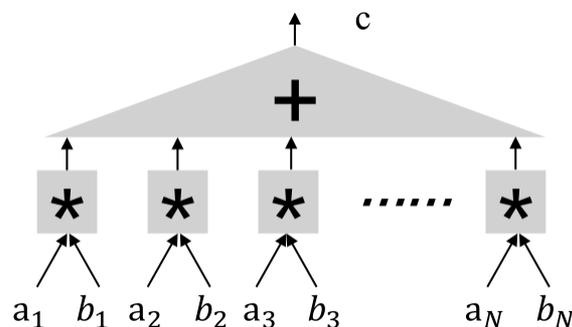


# 运算部件

- ▶ MAC (Multiply-Accumulator)
  - ▶ 标量MAC单元 vs. 向量MAC单元



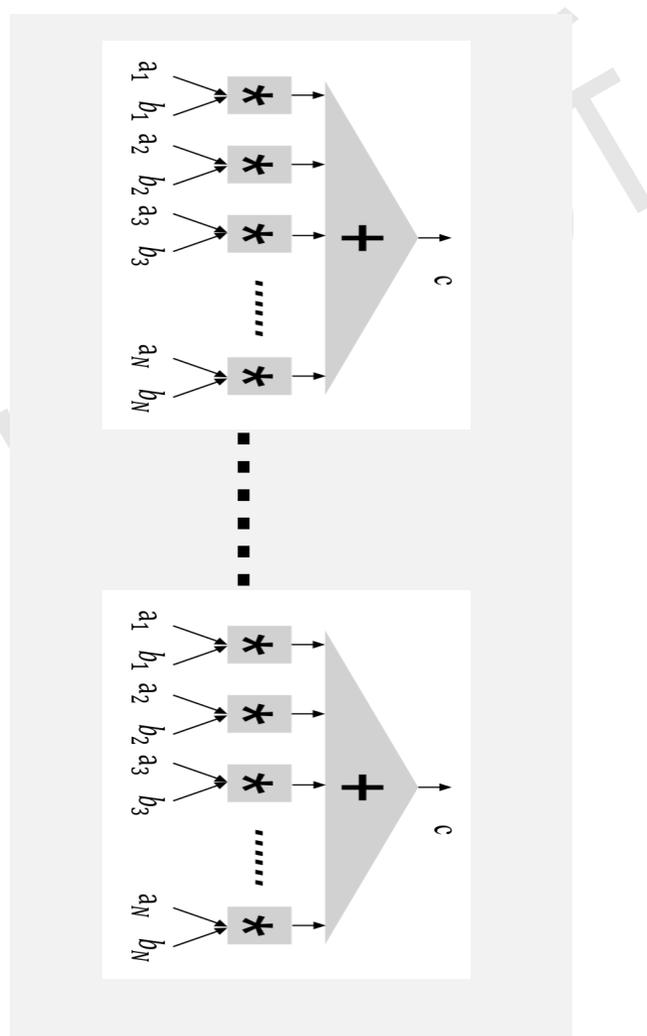
$$a \times b + c$$



$$\sum_{i=1}^N a_i \times b_i + c$$

# 运算部件

- ▶ N个向量MAC单元堆叠
- ▶ 能够支撑DLP指令集
  - ▶ 矩阵/向量/标量计算指令
- ▶ 可否完成
  - ▶ 全连接层?
  - ▶ 池化层?
  - ▶ 卷积层?



# 运算部件

## ▶ VGG19中三种层

全连接层

$$y[j] = G \left( \mathbf{b}[j] + \sum_{i=0}^{N_i-1} \mathbf{W}[j][i] \times \mathbf{x}[i] \right)$$

池化层:

$$Y[nor][noc][i] = \max_{0 \leq kc < K_c, 0 \leq kr < K_r} (\mathbf{X}[r + kr][c + kc][i])$$

$$Y[nor][noc][i] = \frac{1}{K_c \times K_r} \sum_{k_c=0}^{K_c-1} \sum_{k_r=0}^{K_r-1} \mathbf{X}[r + kr][c + kc][i]$$

卷积层:

$$Y[nor][noc][j] = G \left( \mathbf{b}[j] + \sum_{i=0}^{N_{if}-1} \sum_{k_c=0}^{K_c-1} \sum_{k_r=0}^{K_r-1} \mathbf{W}[k_r][k_c][j][i] \times \mathbf{X}[r + k_r][c + k_c][i] \right)$$

# 运算部件

## ▶ VGG19中三种层

全连接层

$$y[j] = G \left( b[j] + \sum_{i=0}^{N_i-1} W[j][i] \times x[i] \right)$$

池化层:

$$Y[nor][noc][i] = \max_{0 \leq kc < K_c, 0 \leq kr < K_r} (X[r + kr][c + kc][i])$$

$$Y[nor][noc][i] = \frac{1}{K_c \times K_r} \sum_{k_c=0}^{K_c-1} \sum_{k_r=0}^{K_r-1} X[r + kr][c + kc][i]$$

卷积层:

$$Y[nor][noc][j] = G \left( b[j] + \sum_{i=0}^{N_{if}-1} \sum_{k_c=0}^{K_c-1} \sum_{k_r=0}^{K_r-1} W[k_r][k_c][j][i] \times X[r + k_r][c + k_c][i] \right)$$

可支撑

# 运算部件

## ▶ VGG19中三种层

全连接层

$$y[j] = G \left( \mathbf{b}[j] + \sum_{i=0}^{N_i-1} \mathbf{W}[j][i] \times \mathbf{x}[i] \right)$$

不足以支撑

池化层:

$$Y[nor][noc][i] = \max_{0 \leq kc < K_c, 0 \leq kr < K_r} (X[r + kr][c + kc][i])$$

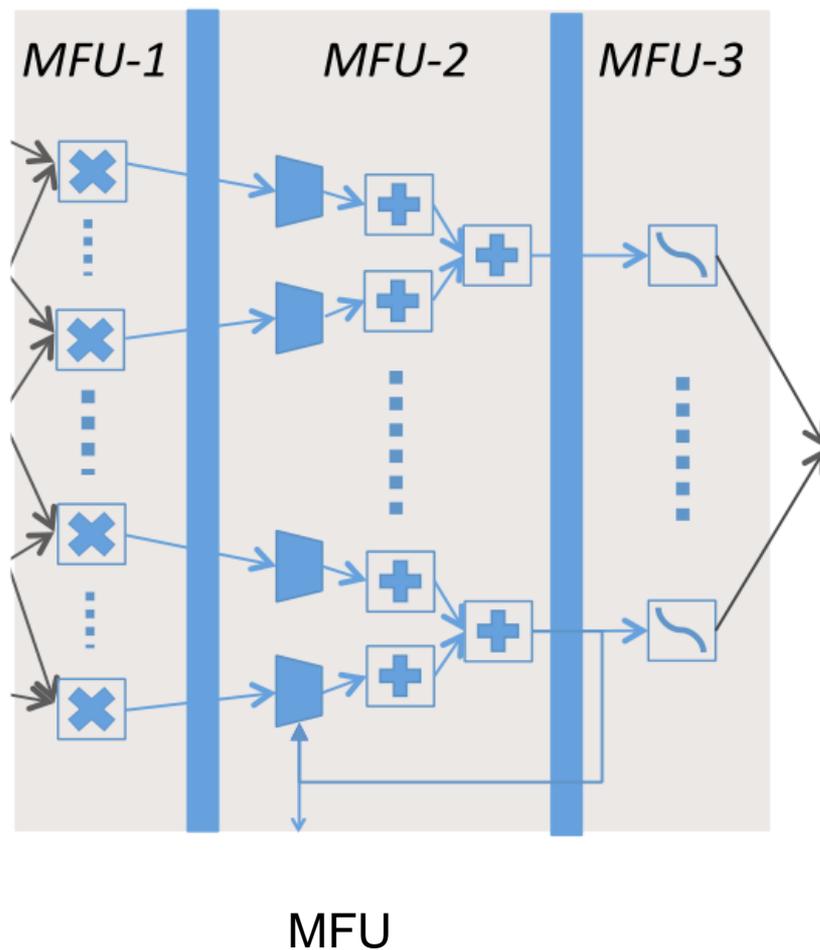
$$Y[nor][noc][i] = \frac{1}{K_c \times K_r} \sum_{k_c=0}^{K_c-1} \sum_{k_r=0}^{K_r-1} X[r + kr][c + kc][i]$$

卷积层:

$$Y[nor][noc][j] = G \left( \mathbf{b}[j] + \sum_{i=0}^{N_{if}-1} \sum_{k_c=0}^{K_c-1} \sum_{k_r=0}^{K_r-1} \mathbf{W}[k_r][k_c][j][i] \times X[r + k_r][c + k_c][i] \right)$$

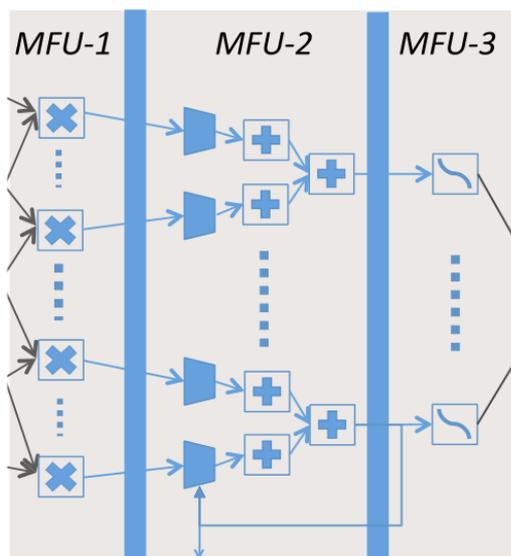
# 运算部件

- ▶ 激活函数处理单元
  - ▶ 非线性函数单元
- ▶ 池化操作
  - ▶ MFU的三个stage的退出通路
- ▶ 任意规模
  - ▶ 局部累加功能

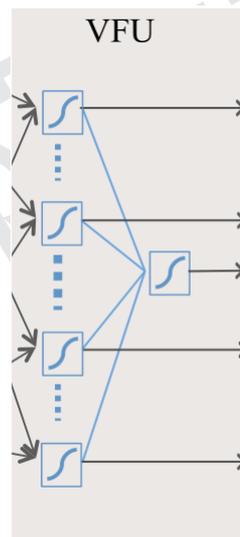


# 运算部件

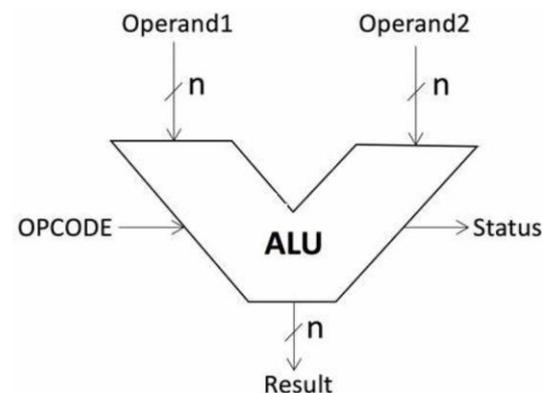
计算指令	矩阵(Matrix)	矩阵乘向量(MMV), 向量乘矩阵(VMM), 矩阵乘标量(MMS), 外积(OP), 矩阵相加(MAM), 矩阵相减(MSM)	寄存器(矩阵/向量 地址/大小, 标量值)
	向量(Vector)	向量基本运算(加(VAV)、减(VSV)、乘(VMV)、除(VDV)), 向量超越函数(指数(VEXP)、对数(VLOG)), 内积(IP), 随机向量(RV), 向量最大值(VMAX), 向量最小值(VMIN)	寄存器(向量 地址/大小, 标量值)
	标量(Scalar)	标量基本运算, 标量超越函数	寄存器(向量 地址/大小, 标量)



MFU: 矩阵运算单元



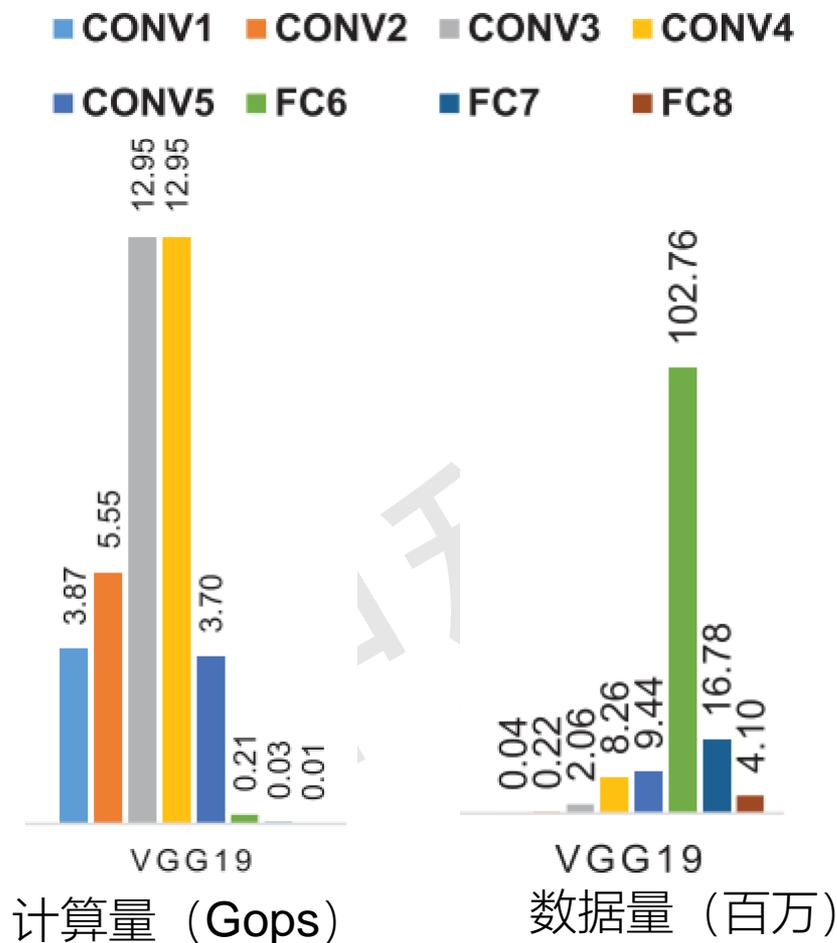
VFU: 向量运算单元



SFU: 标量运算单元

# 访存部件

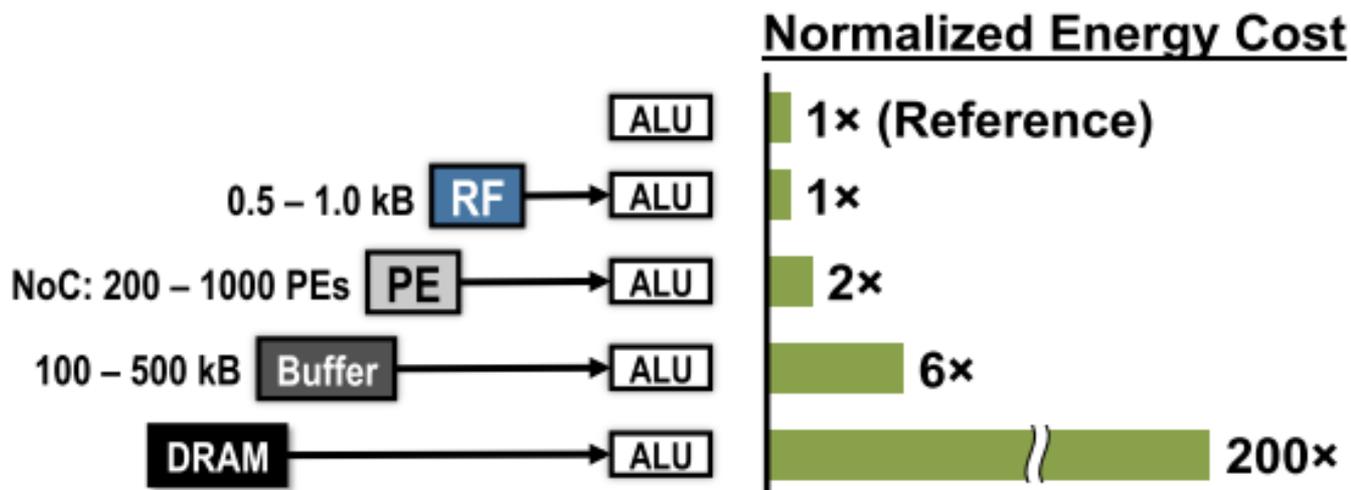
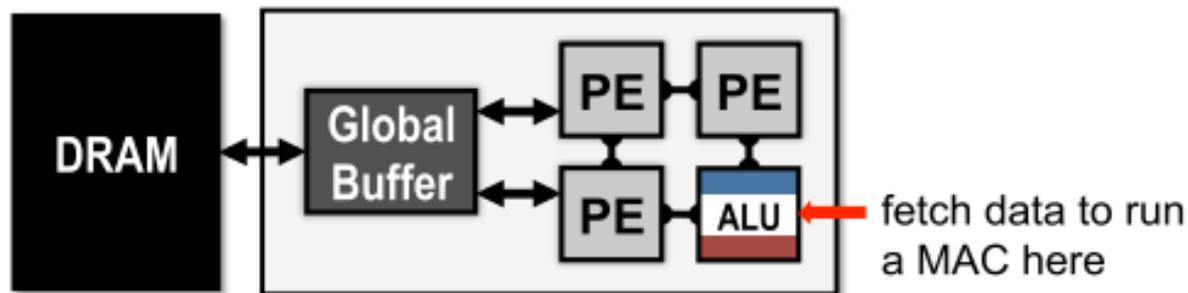
## ▶ VGG19中数据量大小



E	
19 weight layers	
input (224 × 224 RGB image)	
Conv1	conv3-64 conv3-64
	maxpool
Conv2	conv3-128 conv3-128
	maxpool
Conv3	conv3-256 conv3-256 conv3-256 conv3-256
	maxpool
	conv3-512 conv3-512 conv3-512 conv3-512
	maxpool
Conv4	conv3-512 conv3-512 conv3-512 conv3-512
	maxpool
	conv3-512 conv3-512 conv3-512 conv3-512
	maxpool
Conv5	FC-4096
	FC-4096
	FC-1000
	soft-max

# 访存部件

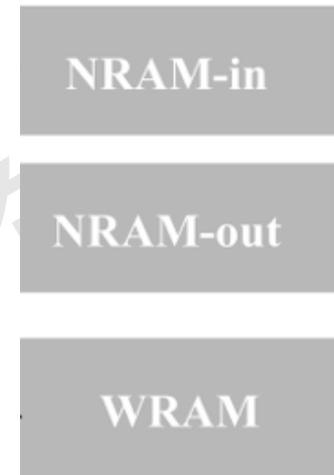
- ▶ 访存非常关键



# 访存部件

## ▶ 可解耦性

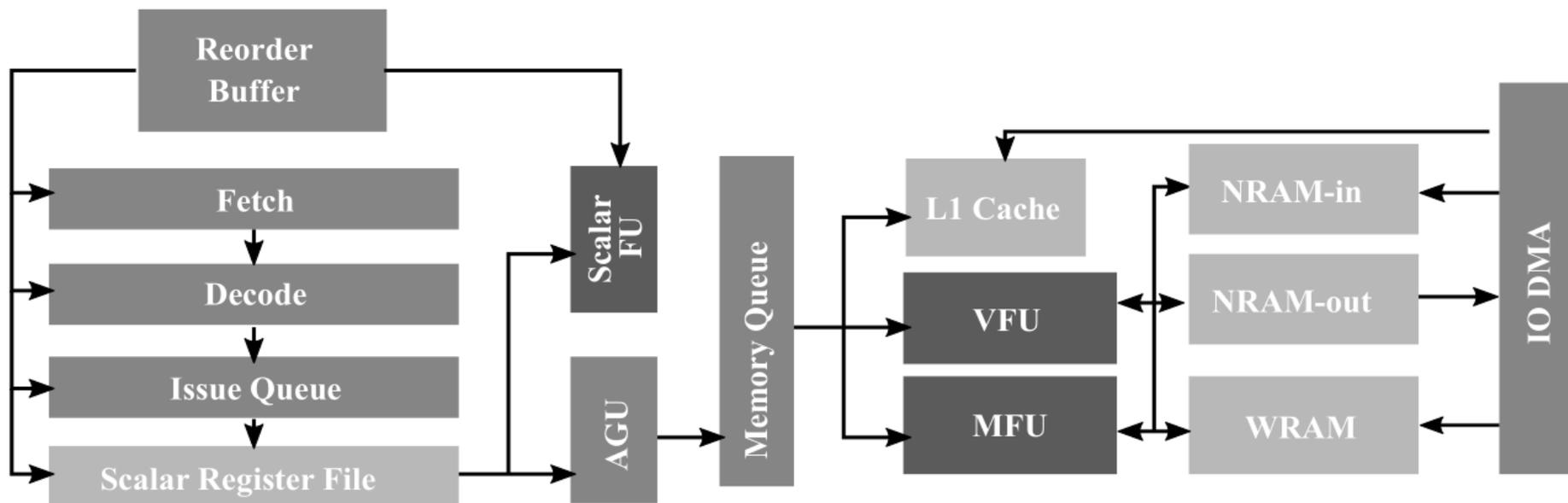
- ▶ 三个分离访存部件
- ▶ 有效避免访存流之间互相干扰



## ▶ 可复用性

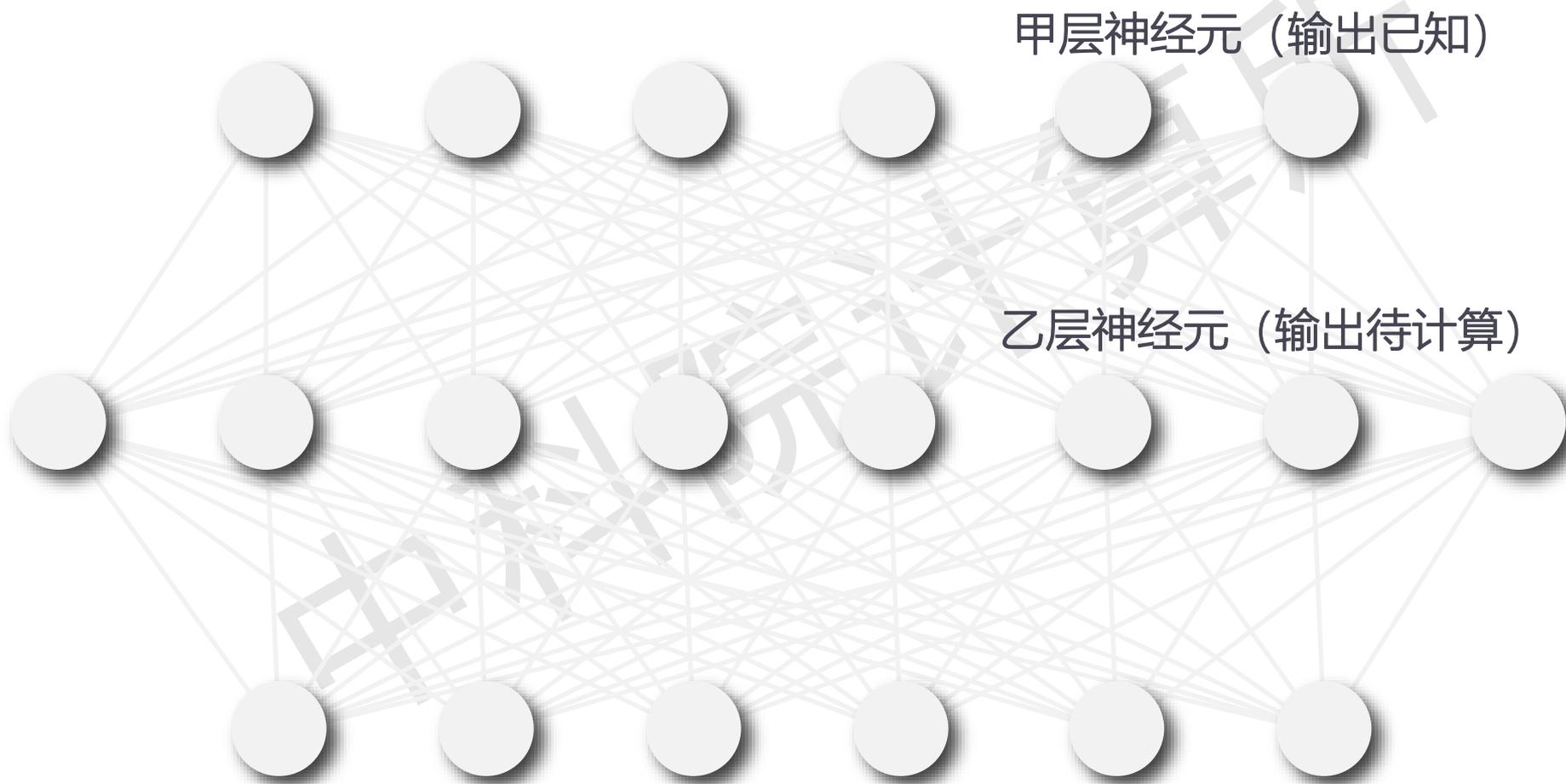
- ▶ 片上缓存：形成“运算单元-片上-片外”的存储hierarchy
- ▶ Scratchpad Memory管理
- ▶ 提高片上数据复用率

# 深度学习处理器DLP架构



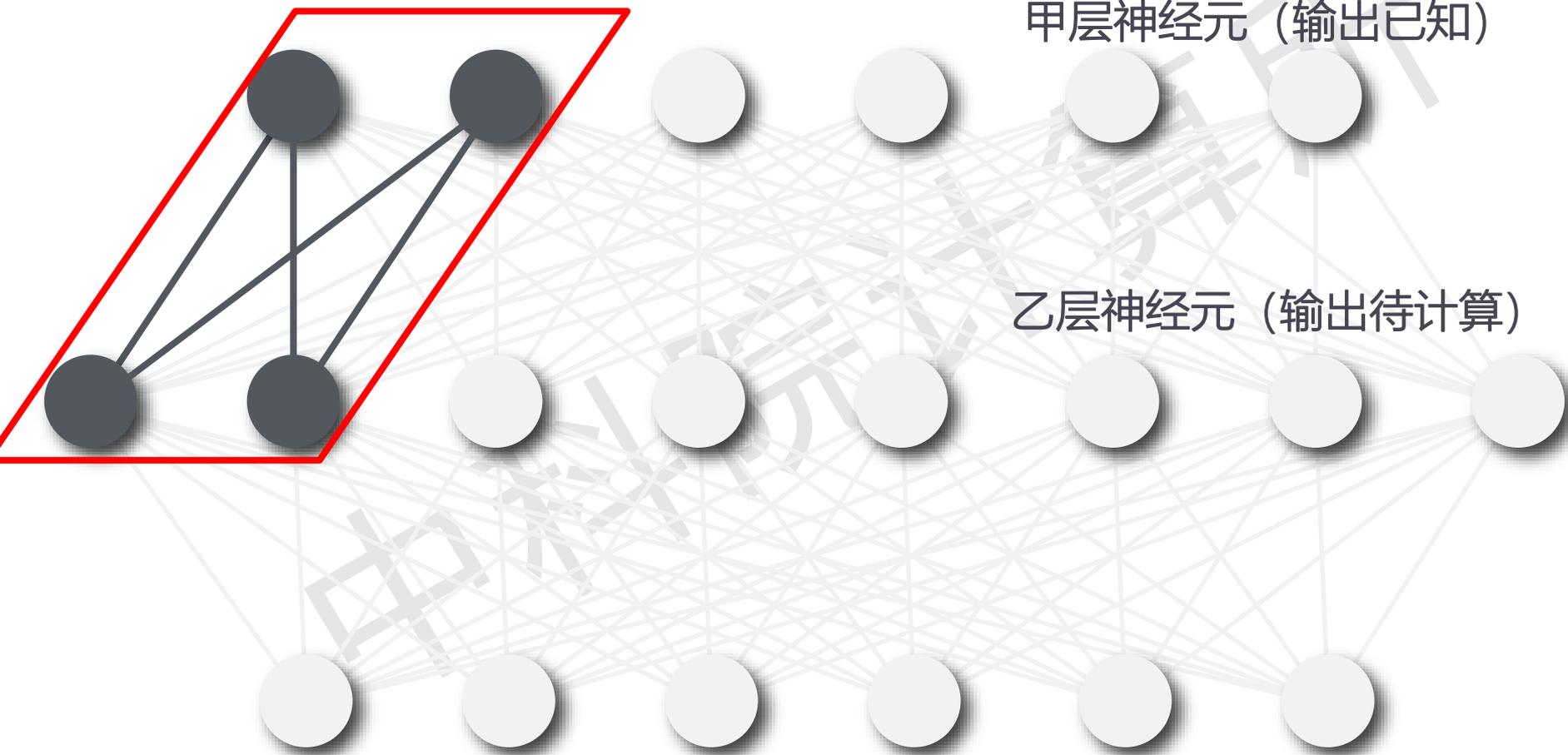
# 算法映射

- ▶ 基本思想：硬件的分时复用



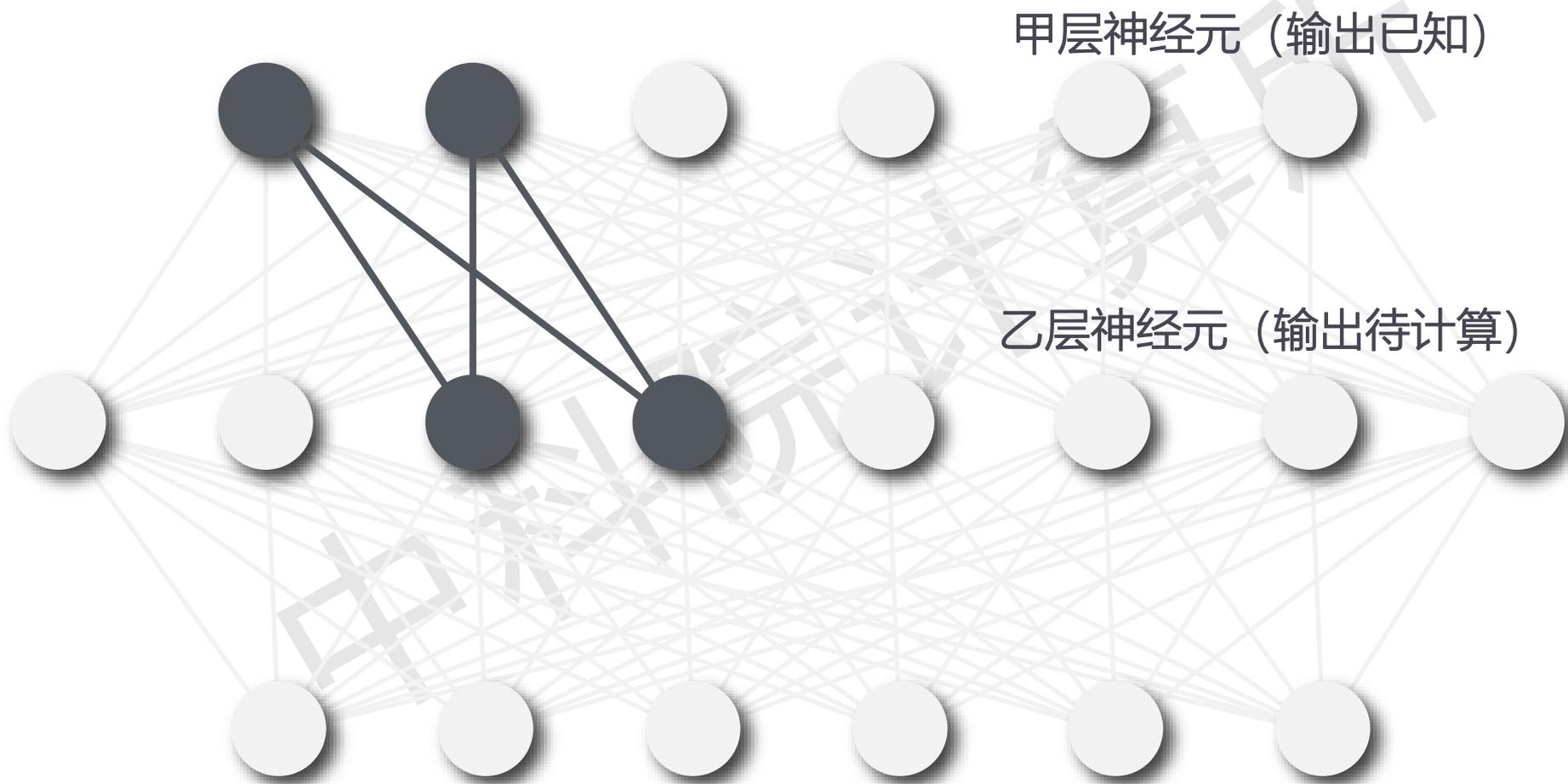
# 算法映射

- ▶ 基本思想 硬件运算单元单周期的处理能力



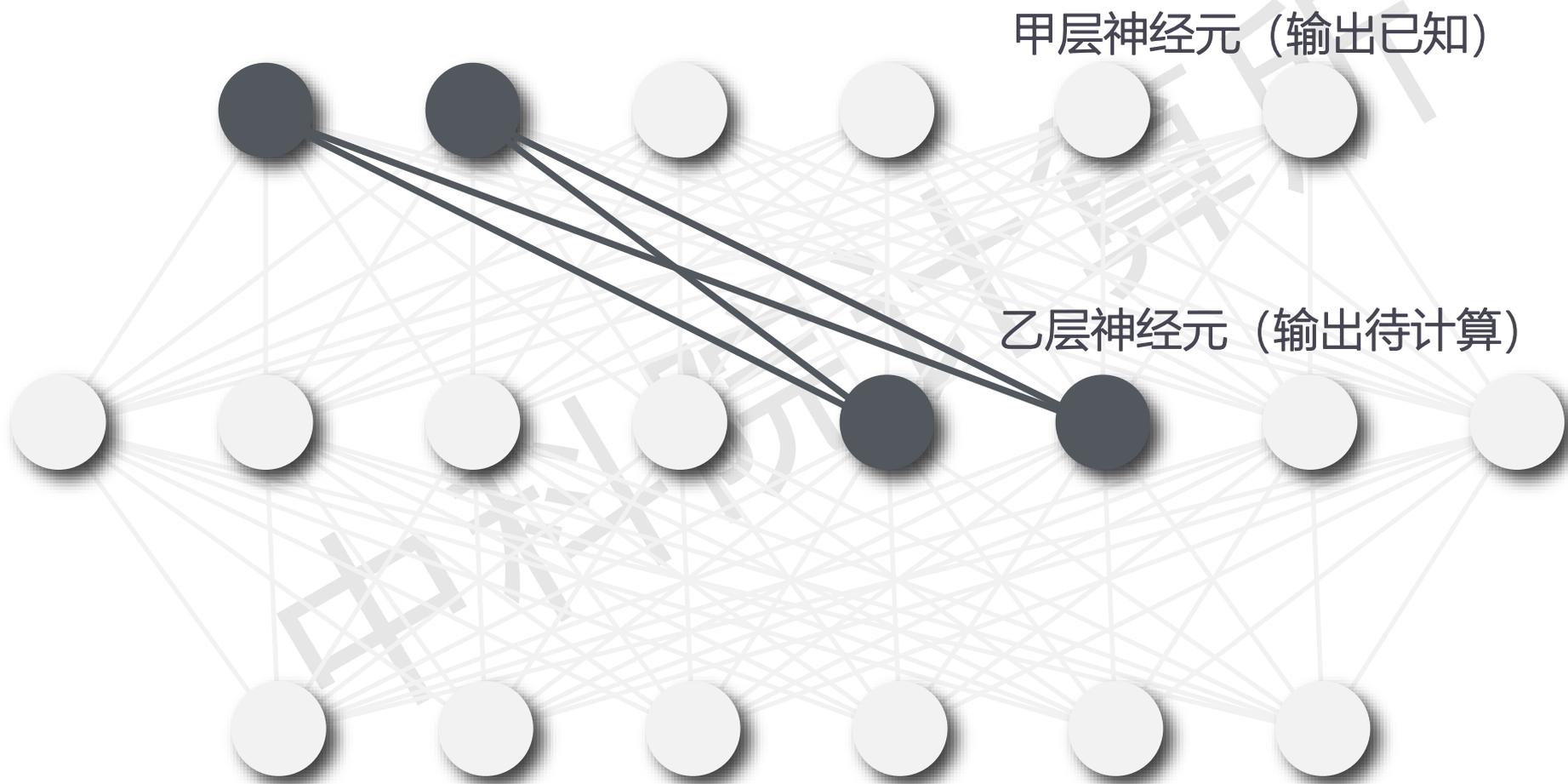
# 算法映射

## ▶ 基本思想



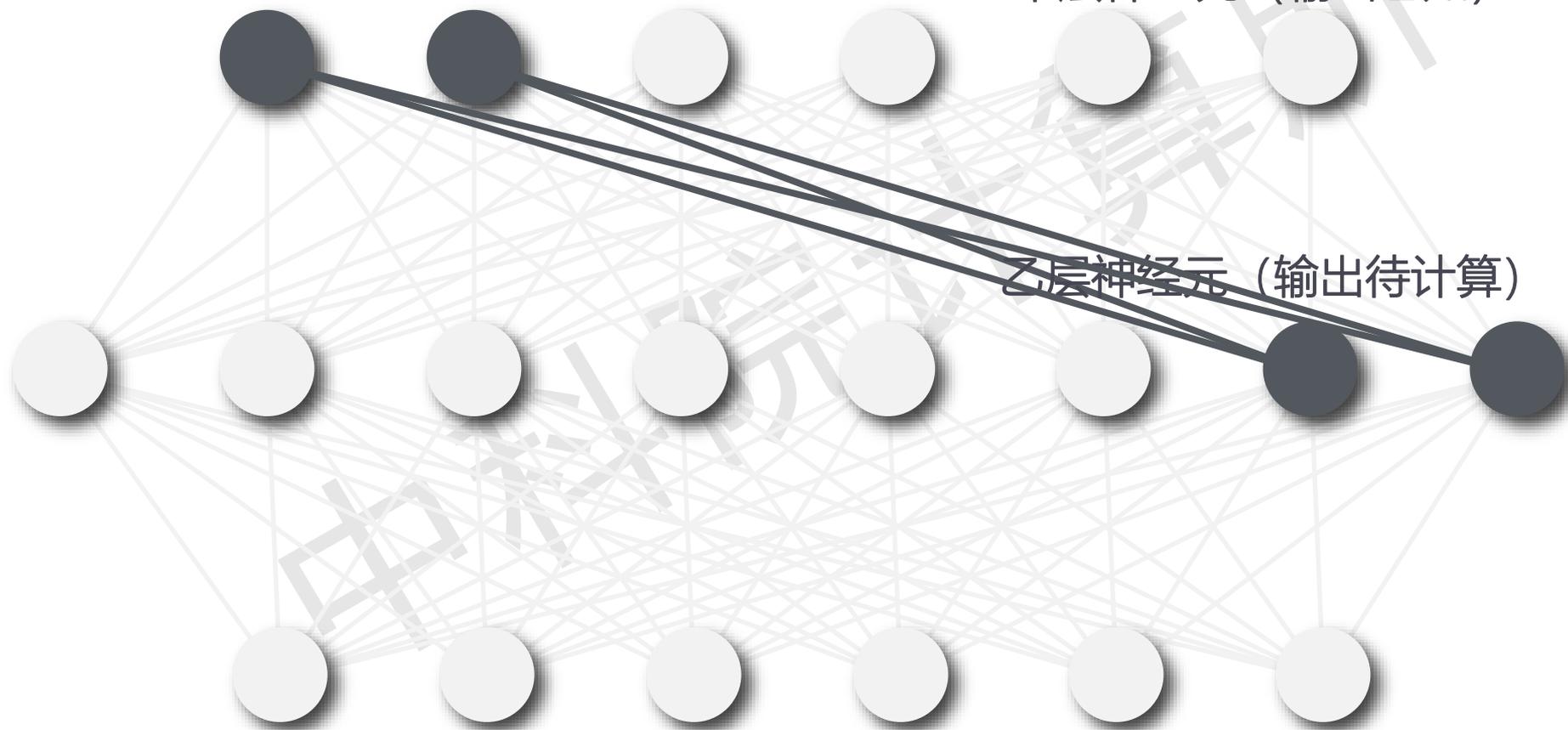
# 算法映射

## ▶ 基本思想



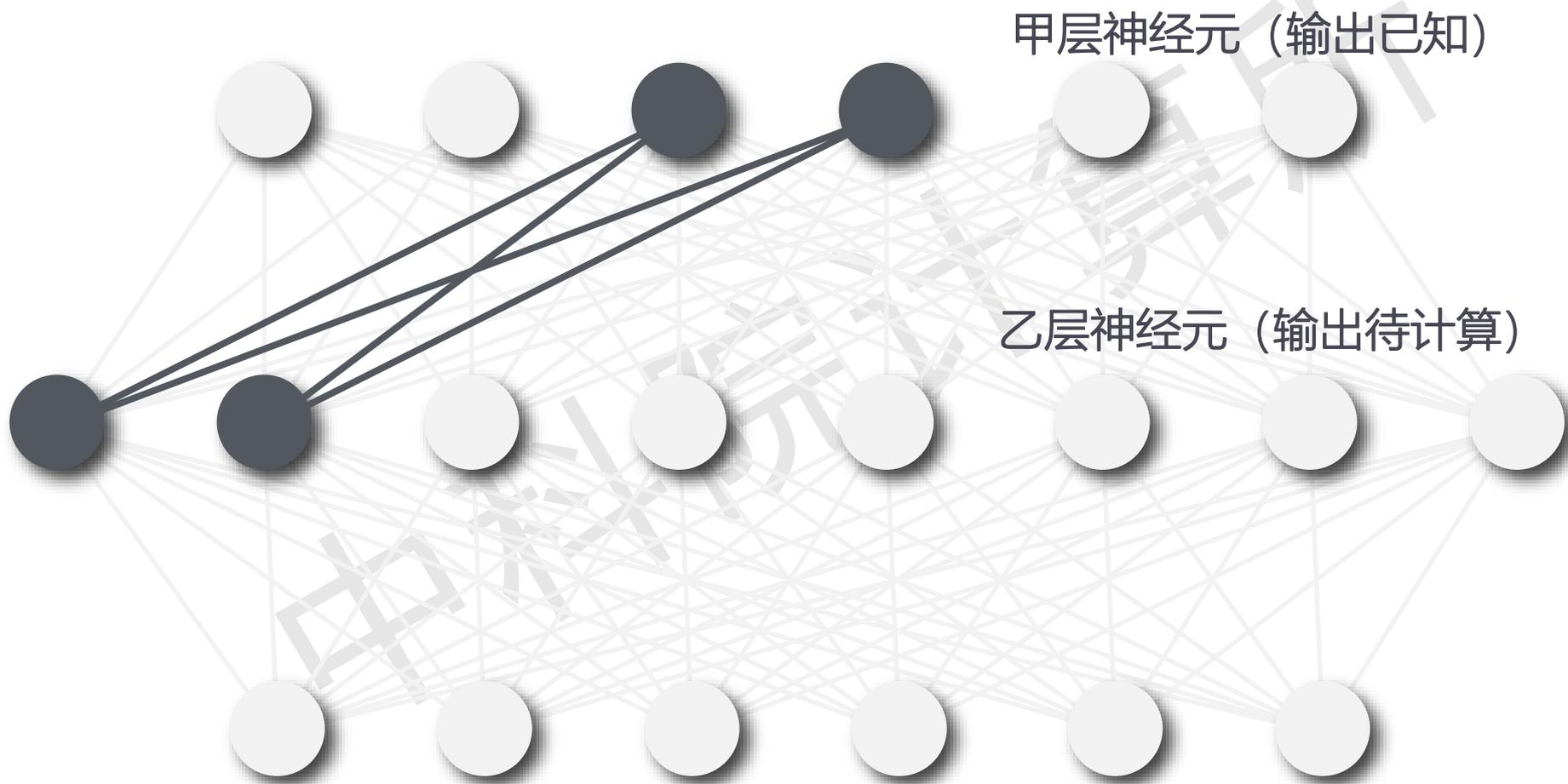
# 算法映射

- ▶ 基本思想  
这块神经元的输出数据不再能重用



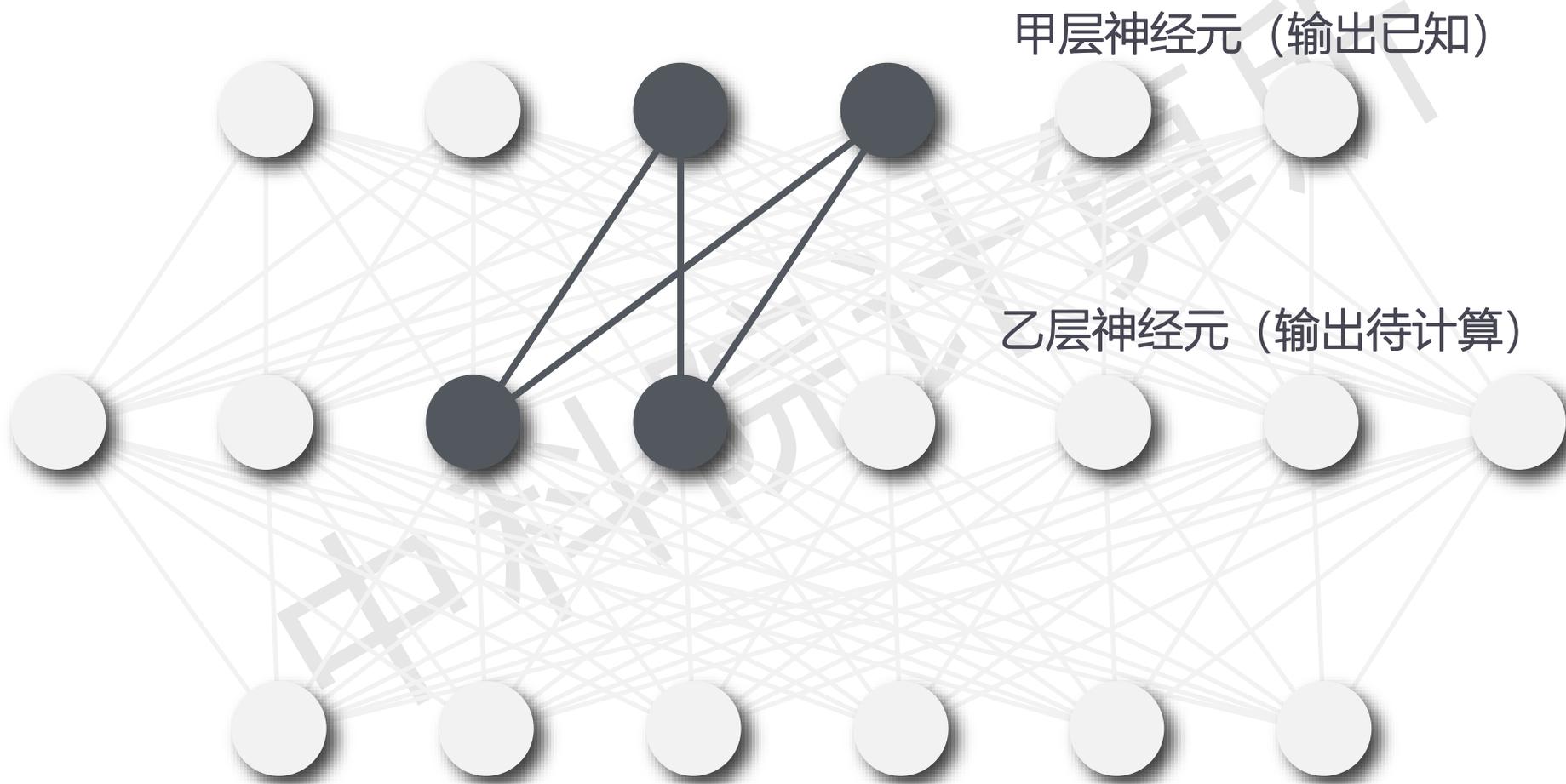
# 算法映射

## ▶ 基本思想



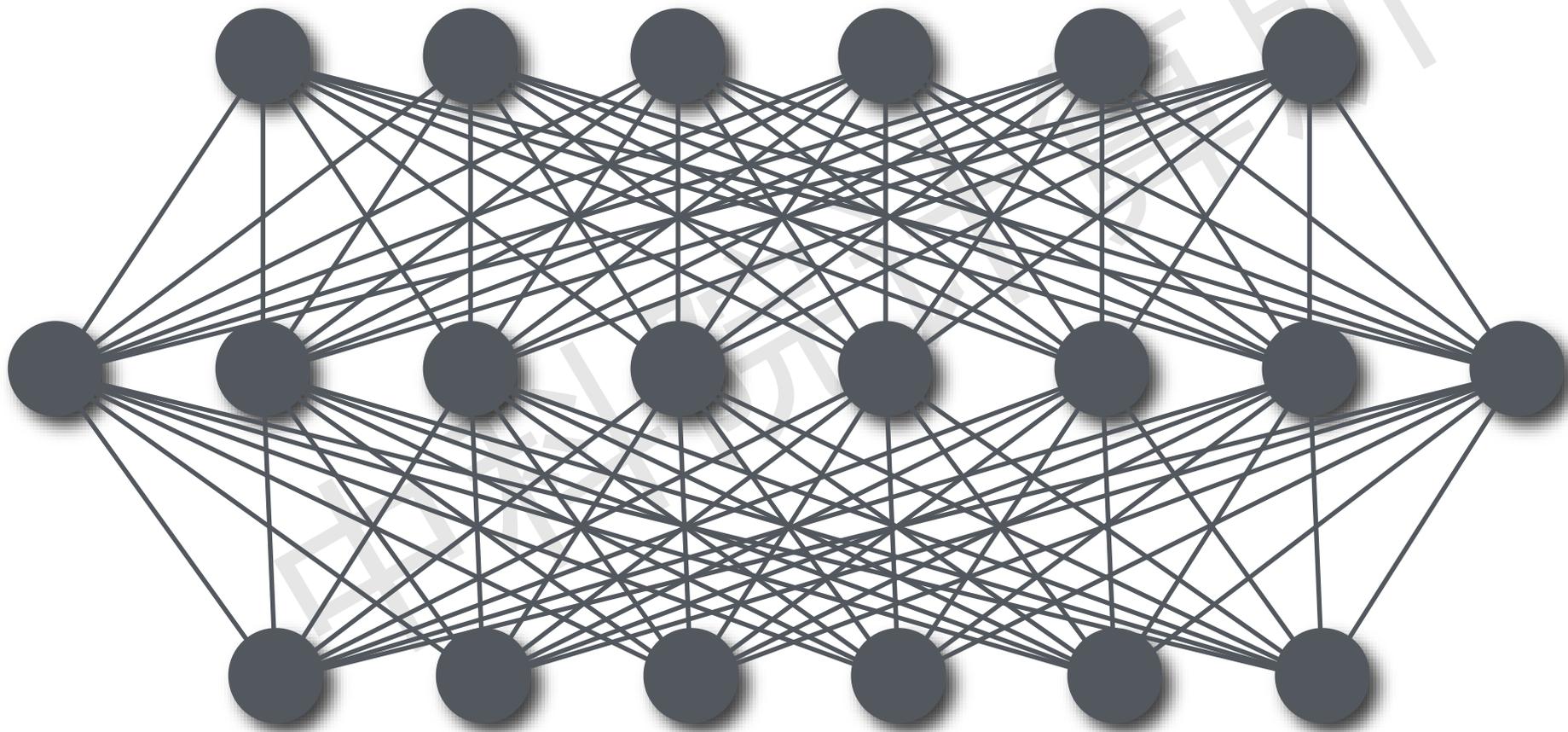
# 算法映射

## ▶ 基本思想



# 算法映射

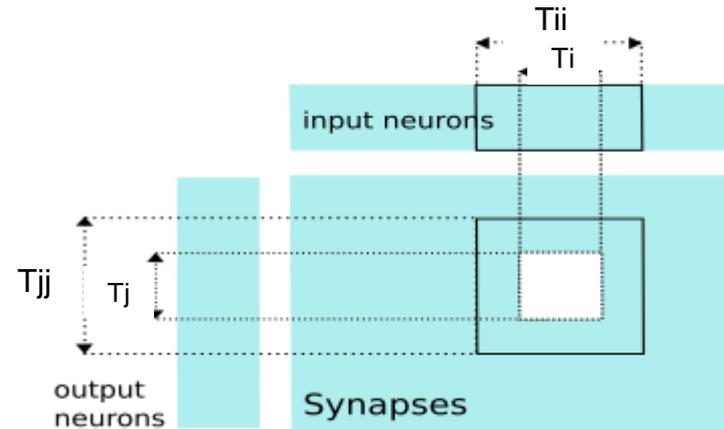
## ▶ 基本思想



# 算法映射

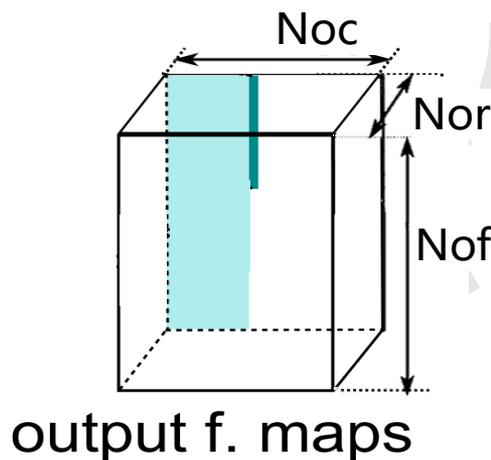
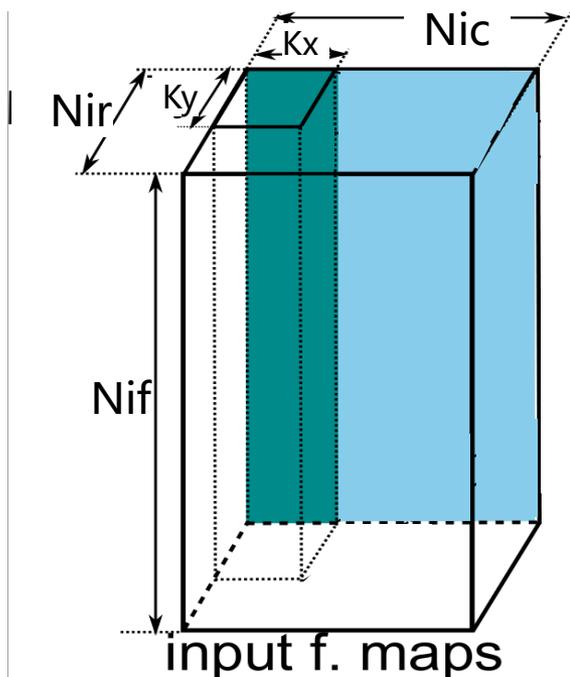
- ▶ 全连接层映射
  - ▶ 即具体计算指令的顺序

```
VLOAD $3, $0, #100 //从地
MLOAD $4, $2, #300 //从地
MMV $7, $1, $4, $3, $0
VAV $8, $1, $7, $5
VEXP $9, $1, $8
VAS $10, $1, $9, #1 //S
VDV $6, $1, $9, $10
VSTORE $6, $1, #200
```



# 算法映射

## 卷积层映射：一种方式



■ 计算当前输出所需要的输入

■ 缓存至片上的输入

■ 已经完成计算的输出神经元

■ 正在计算的输出神经元

$K_c$ 、 $K_r$ : 卷积核的宽、高

$N_{ic}$ 、 $N_{ir}$ : 输入feature map的宽、高

$N_{oc}$ 、 $N_{or}$ : 输出feature map的宽、高

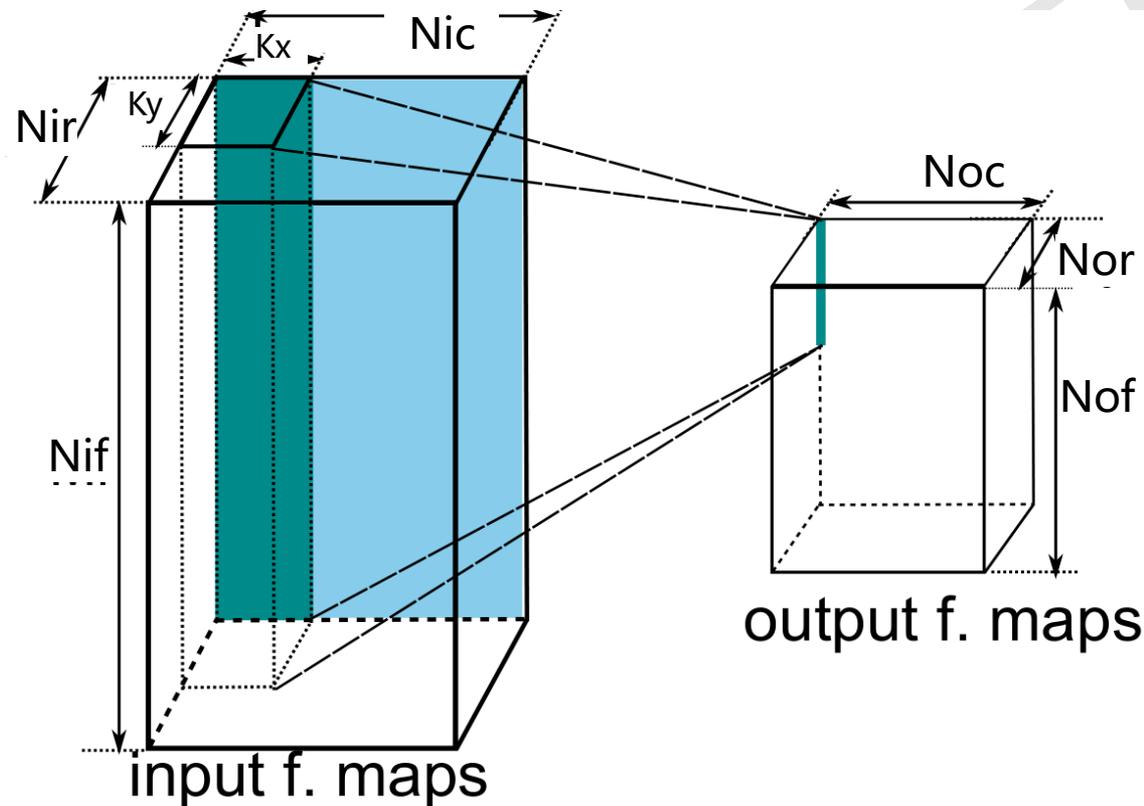
$N_{if}$ : 输入feature map个数

$N_{of}$ : 输出feature map个数

\*存在其他映射方式

# 算法映射

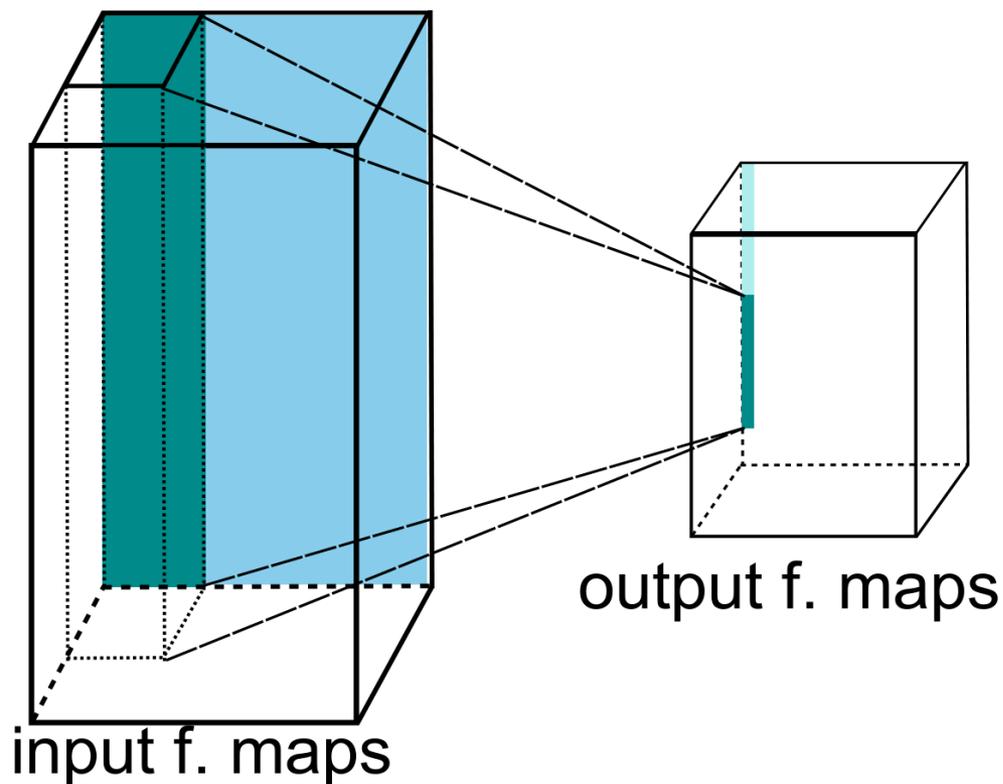
## 卷积层映射



$N$ 个向量MAC计算 $N$ 个输出神经元，其中这些输出神经元属于 $N$ 个输出feature map。片内载入数据是所有输入feature map的第一行 ( $N_{ic} \cdot N_{if}$ 个输入)，而正在计算的这 $N$ 个输出神经元只和 $K_c \cdot N_{if}$ 个输入相关，所需要的权值是 $K_c \cdot N_{if} \cdot N$ 个。

# 算法映射

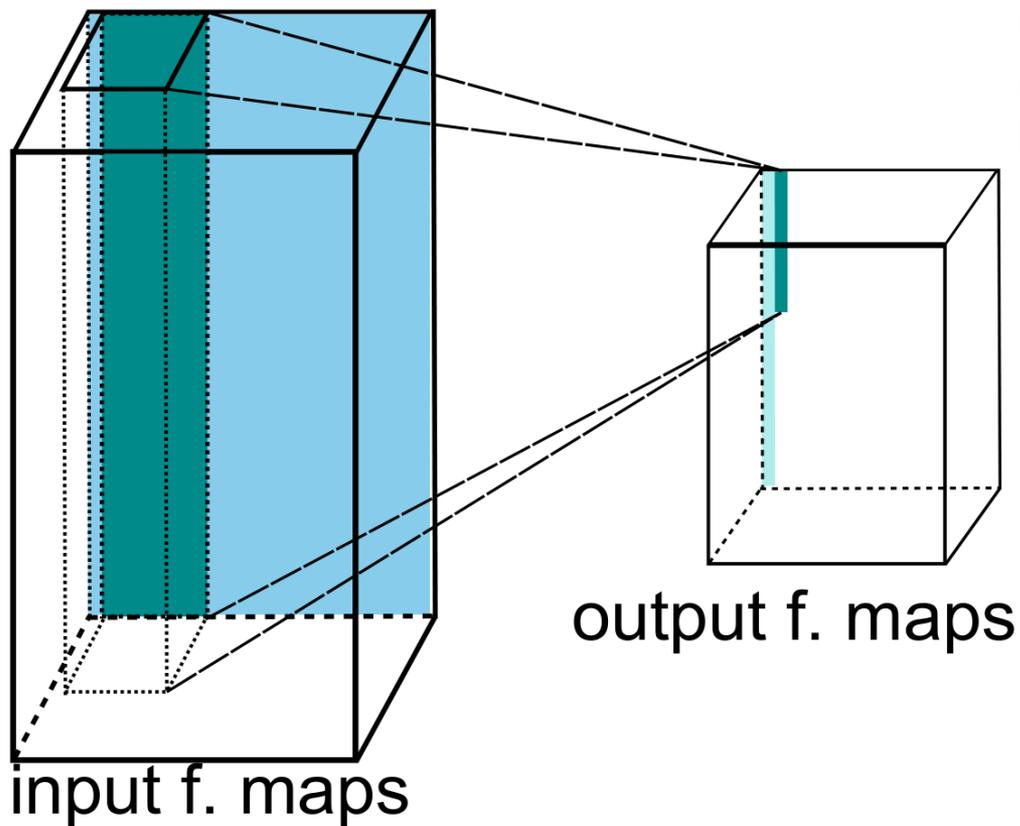
## 卷积层映射



当计算完成 $N$ 个输出神经元中与 $K_c \times N_{if}$ 个输入相关的部分，开始下面 $N$ 个输出神经元。同样的，这 $N$ 个神经元属于 $N$ 个不同的输出feature map。在这个计算过程中，上 $N$ 个输出神经元的 $K_c \times N_{if} \times N$ 个权值可以复用。

# 算法映射

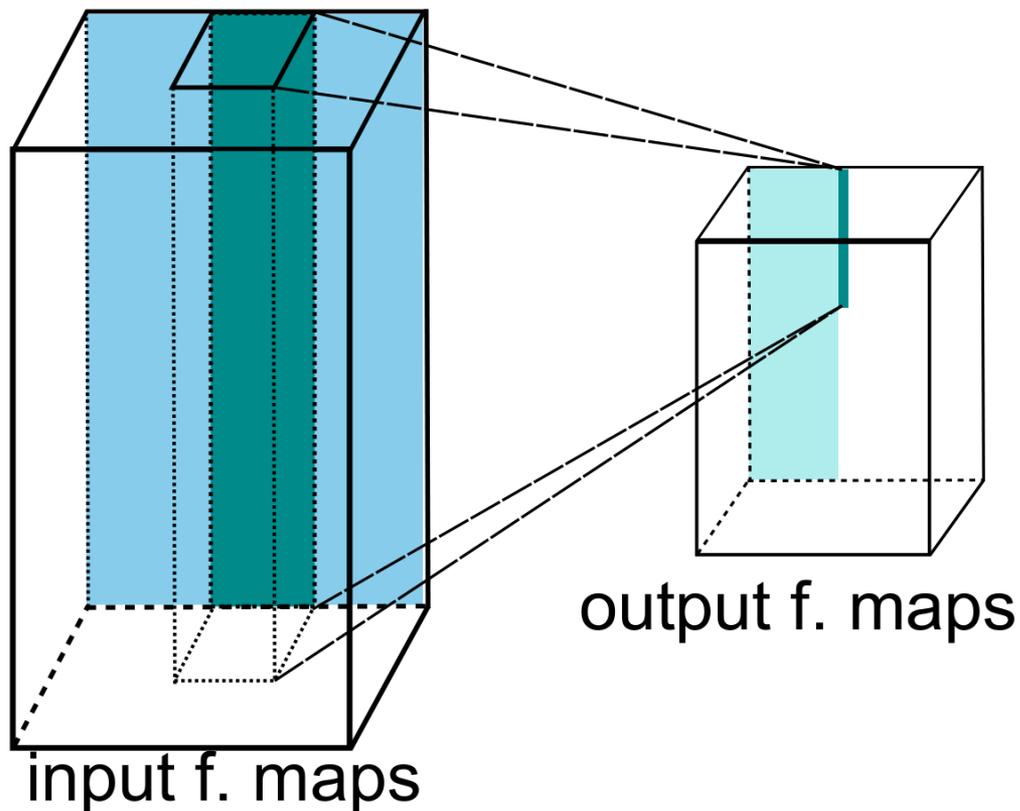
## 卷积层映射



当开始计算不同位置的 $N$ 个神经元时，所依赖的输入与之前输入重叠，而权值则可以完全复用。

# 算法映射

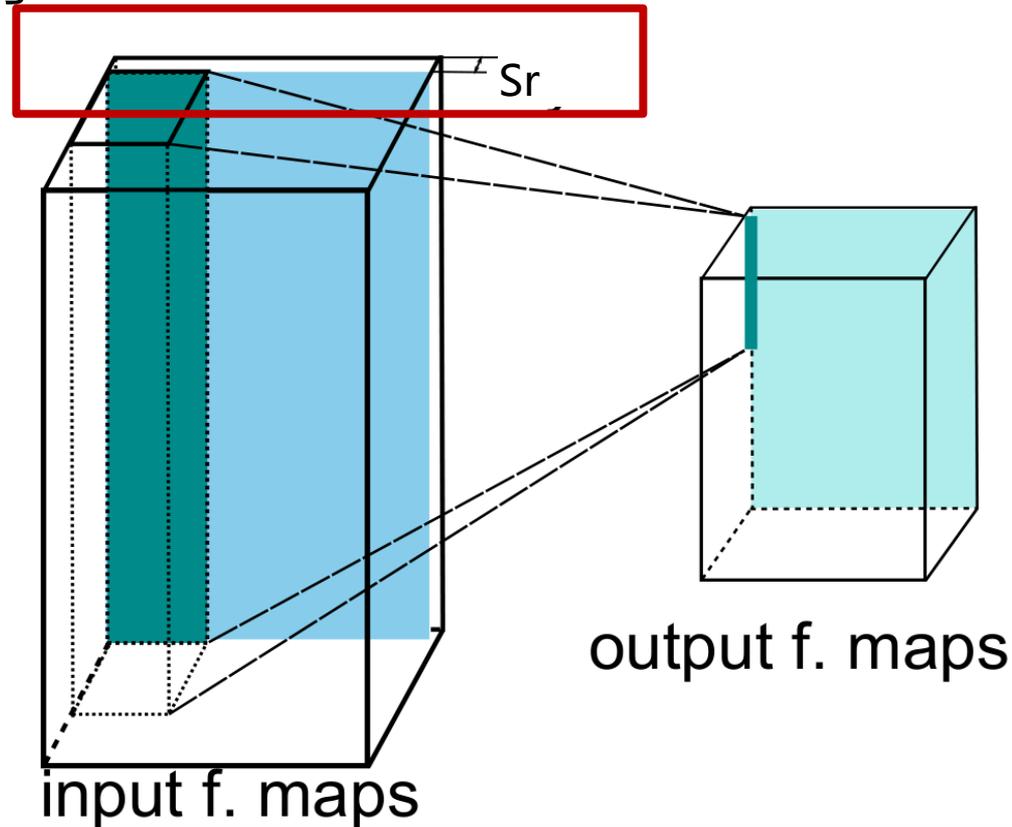
## 卷积层映射



重复以上过程，只要计算的输出神经元属于同一个面（ $N_{ic} \times N_{of}$ ），所依赖的权值就不会发生变化。

# 算法映射

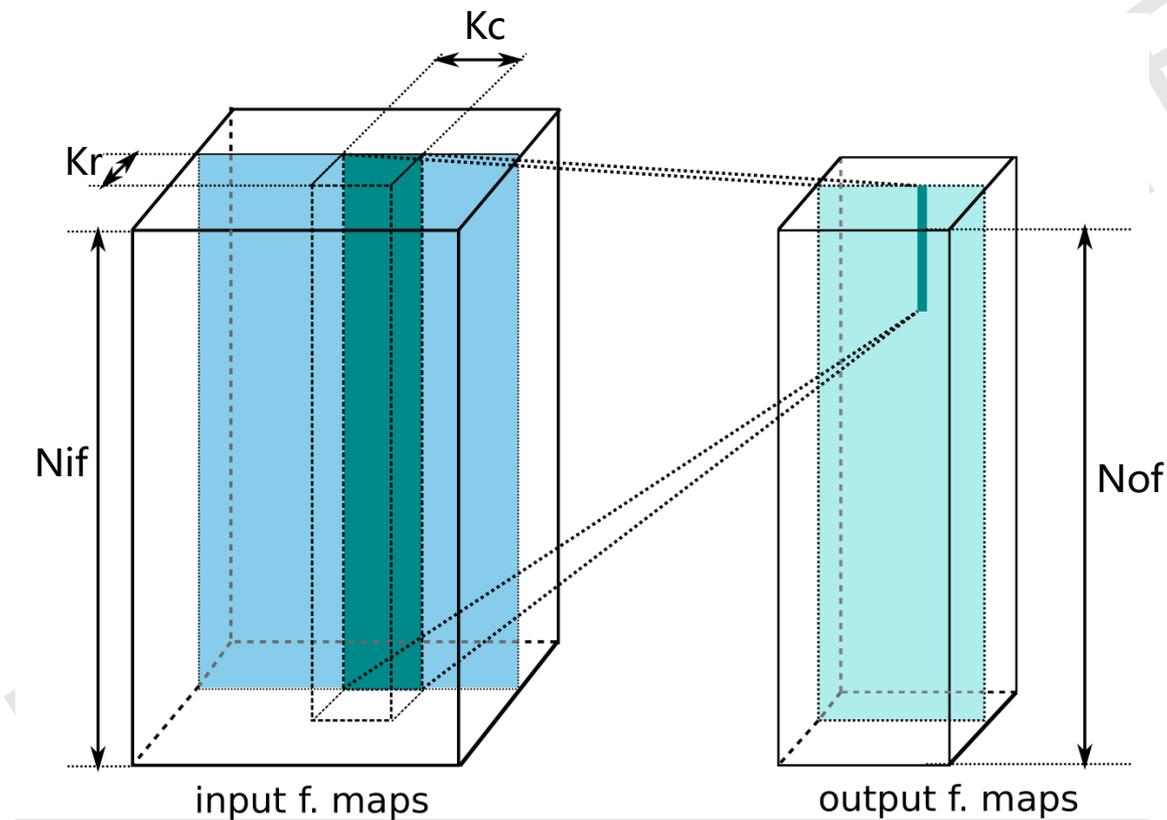
## 卷积层映射



当与一个面的输入神经元计算都已经完成（即所有输入feature map的第一行），后续计算已经再不需要这些输入神经元，片上可以重新载入第二个面（即所有输入feature map的第二行），开始进行新的输出神经元的计算。重复以上过程。

# 算法映射

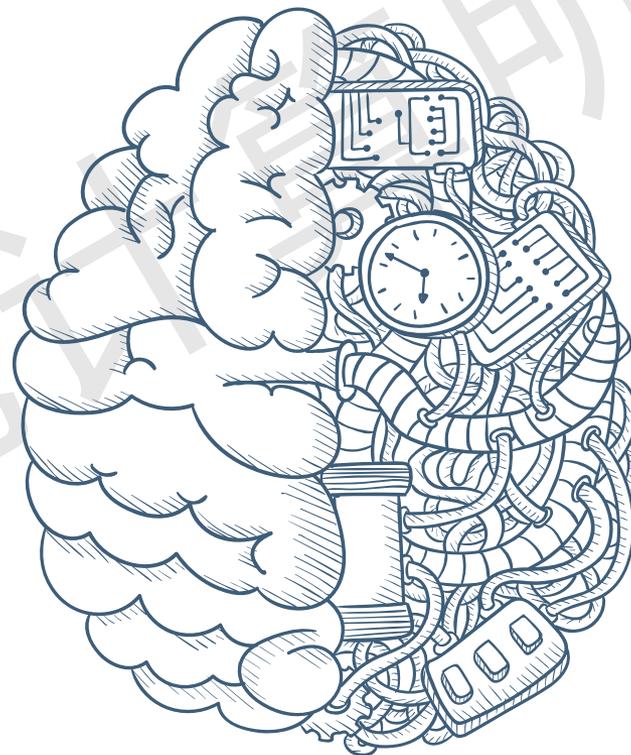
## 池化层映射



池化层的计算顺序与卷积层一致，只是计算是 $\max()$ 或者 $\text{avg}()$

# 深度学习处理器DLP

- ▶ 指令集
- ▶ 流水线
- ▶ 运算部件
- ▶ 访存部件
- ▶ 映射方式



# 目录

- ▶ 深度学习处理器概述
- ▶ 目标算法分析
- ▶ 深度学习处理器DLP结构
- ▶ 优化设计
- ▶ 性能评价
- ▶ 其他加速器

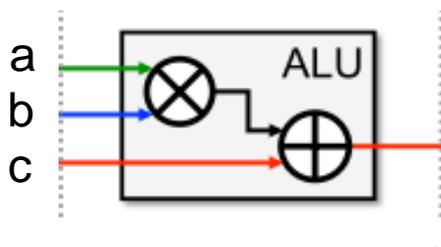
# 目录

- ▶ 优化设计
  - ▶ 基于标量MAC的运算部件
  - ▶ 稀疏化
  - ▶ 低位宽

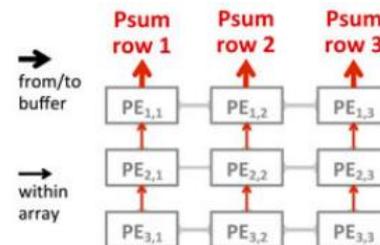
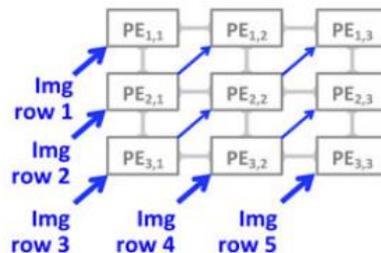
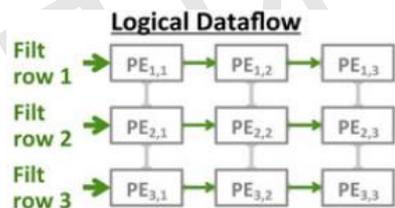
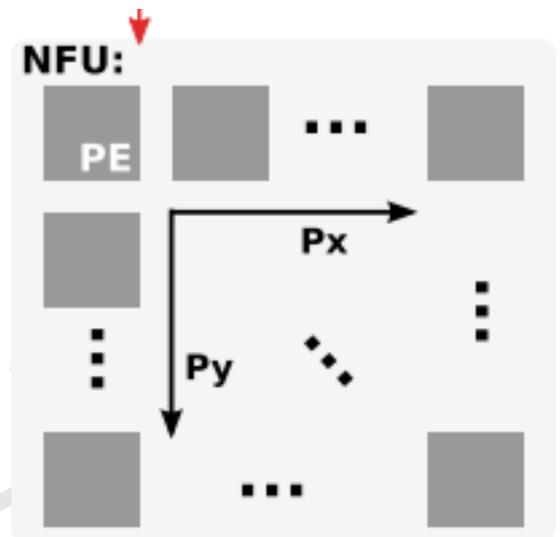
中科院计算所

# 基本运算单元

标量MAC计算单元

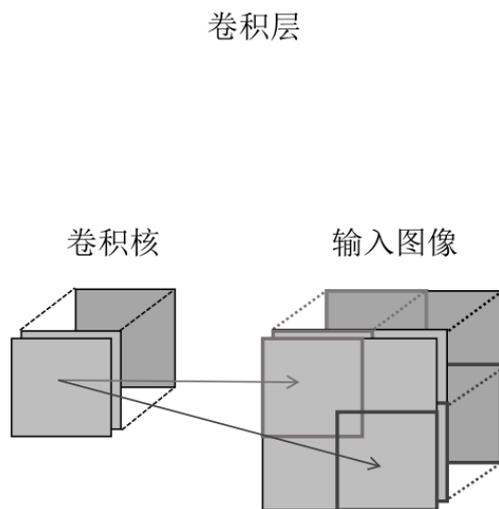


$$a \times b + c.$$



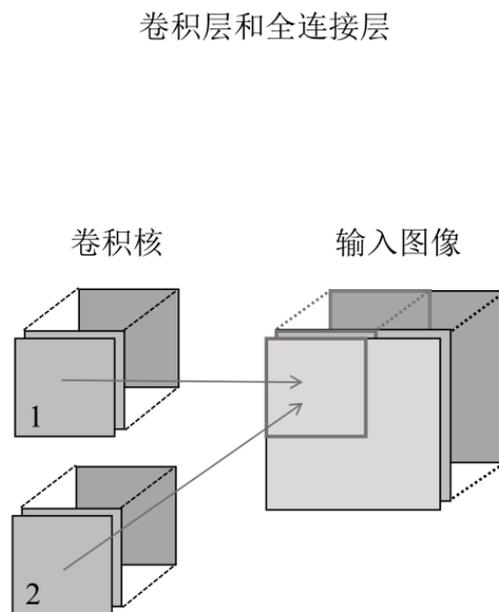
# 基本运算单元

## 卷积运算中的数据复用



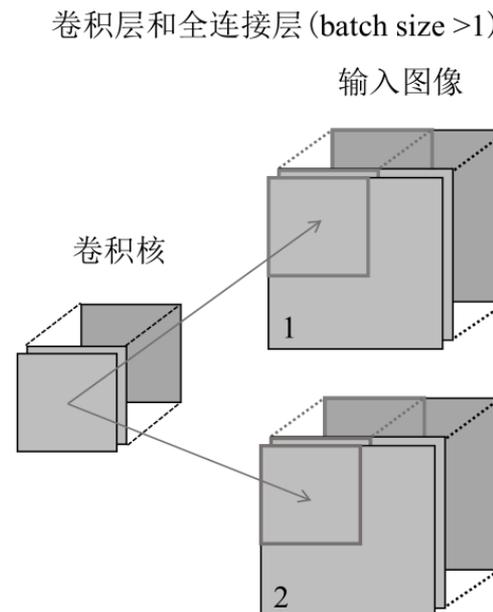
复用：激活 卷积核权值

(a) 卷积复用



复用：激活

(b) 输入特征图复用

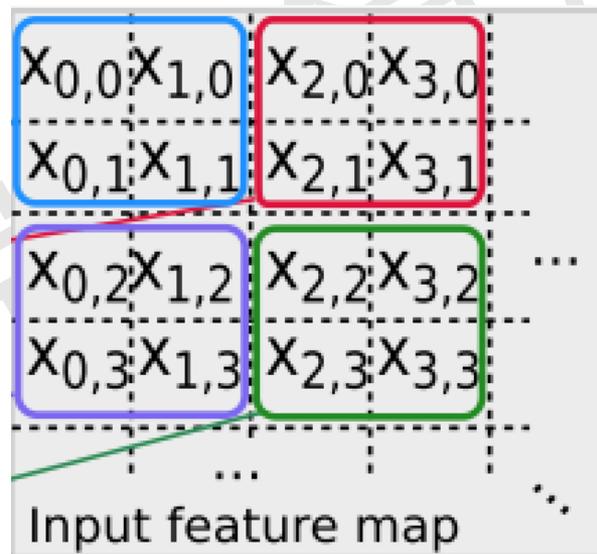
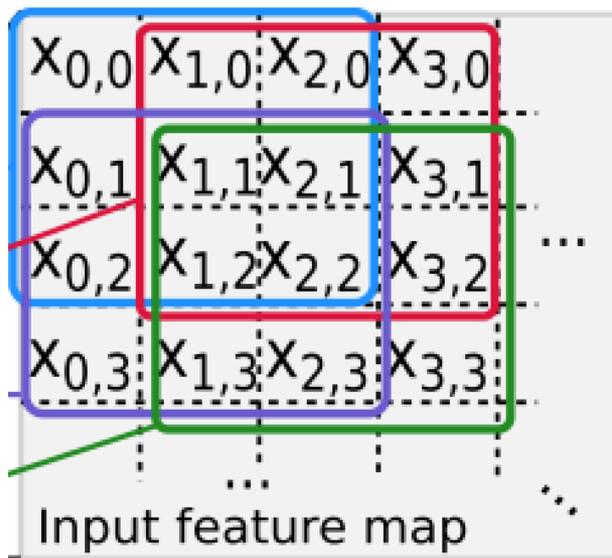


复用：激活

(c) 卷积核复用

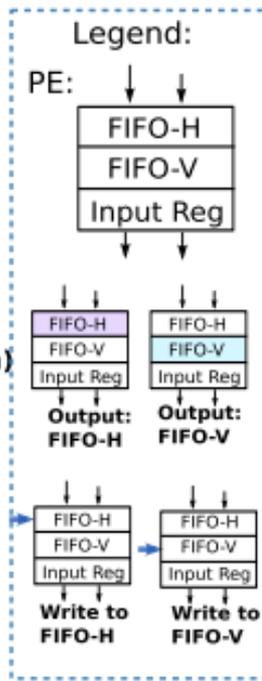
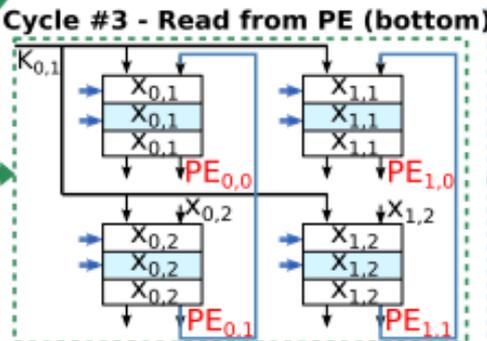
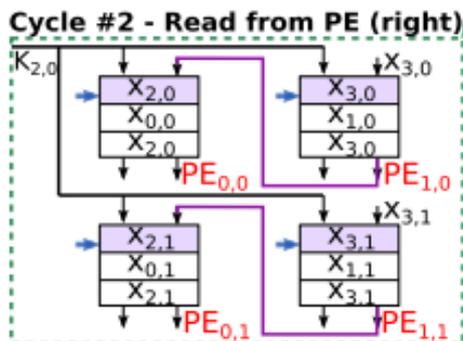
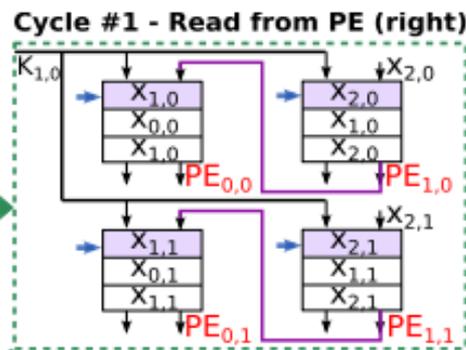
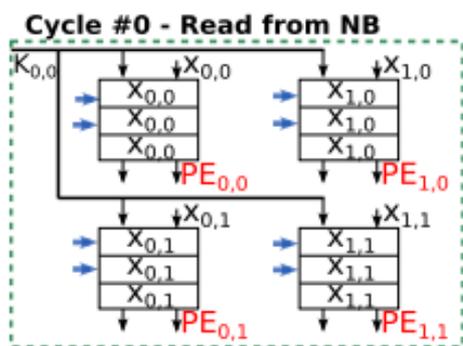
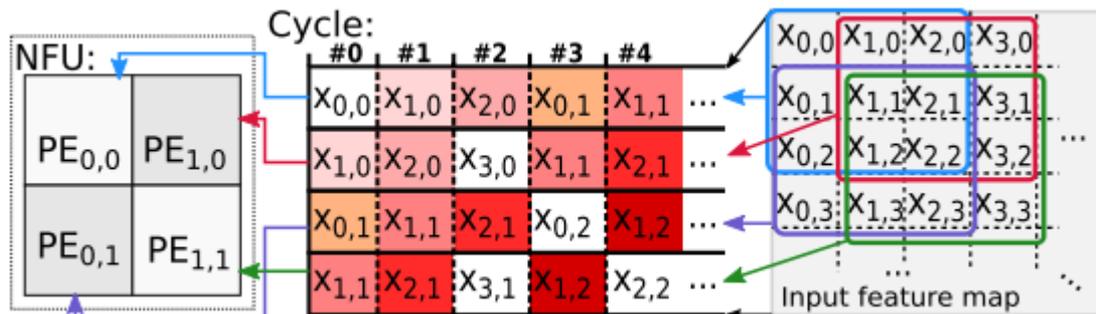
# 基本运算单元

- ▶ Recall卷积运算，对比于pooling运算



# 基本运算单元

## ► 一种实现



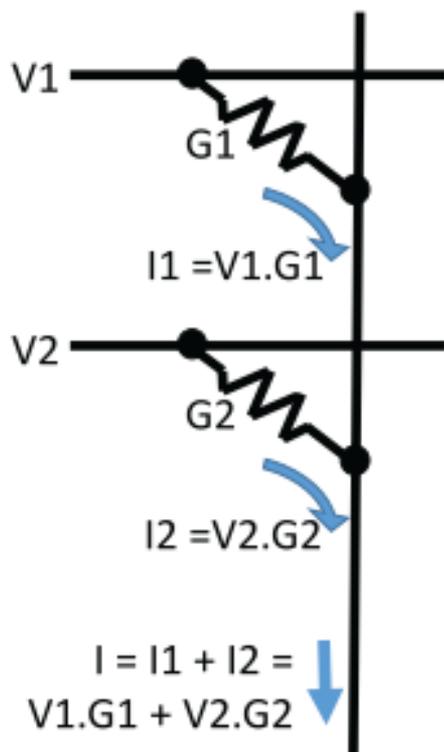
# 基本运算单元

## ▶ 向量MAC单元 vs. 标量MAC单元

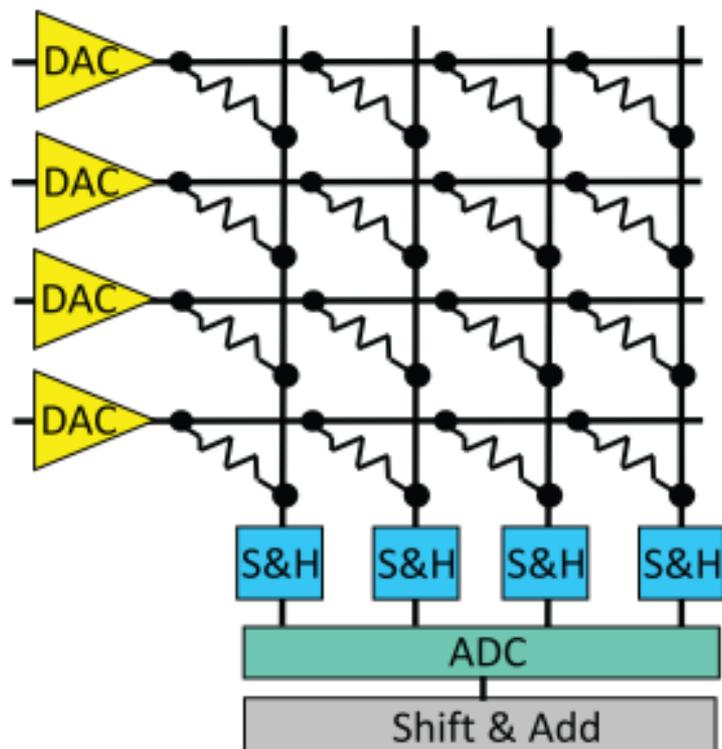
	基于向量 MAC 计算单元	基于标量 MAC 计算单元
大小	16 个 16 维向量 MAC	256 个 MAC, 16x16 阵列
乘法器个数	256	256
加法器个数	240 (16x15)	256
每拍需求外部操作数	512	32
操作粒度	向量、矩阵	向量、矩阵
卷积层映射	输入神经元复用、输出神经元复用	输入神经元复用、输出神经元复用、权重复用
优点	高效支持矩阵向量映射, 灵活性高	专用数据流高效支持卷积, 带宽需求降低
缺点	依赖外部数据排布, 带宽需求高	灵活性差, 支持其他算子其他特性困难

# 基本运算单元

## ▶ 非基于MAC运算单元的



(a) Multiply-Accumulate operation



(b) Vector-Matrix Multiplier

# 稀疏化

## ► 稀疏

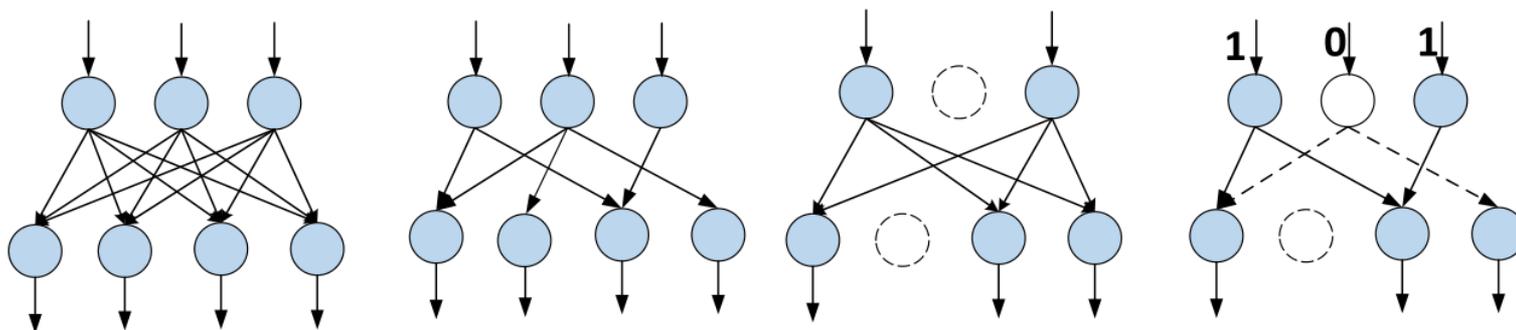
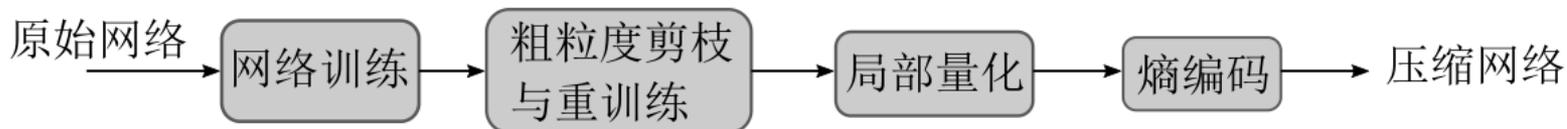
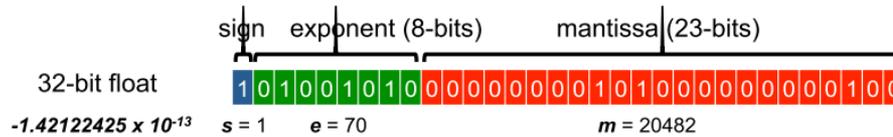


图 6.19 (a) 稠密神经网络 (b) 静态突触稀疏 (c) 静态神经元稀疏 (d) 动态稀疏

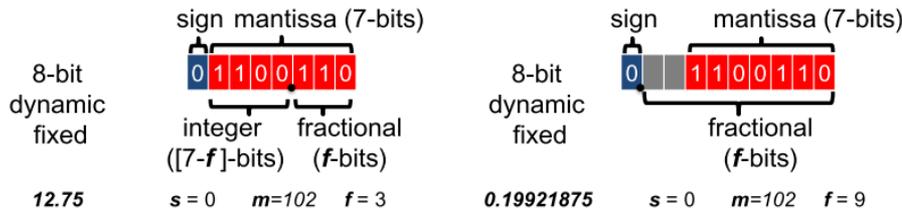


# 低位宽

## ▶ 低位宽



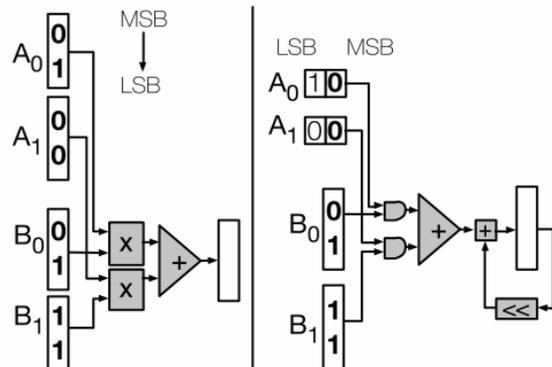
(a) 32-bit floating point example



(b) 8-bit dynamic fixed point examples

不同精度的数据格式

## ▶ 串行乘法器



# 目录

- ▶ 深度学习处理器概述
- ▶ 目标算法分析
- ▶ 深度学习处理器DLP结构
- ▶ 优化设计
- ▶ 性能评价
- ▶ 其他加速器

# 性能评价

- ▶ TOPS (Tera Operations Per Second)
- ▶ TOPS, 而不是TFLOPS

$$TOPS = f_c \times (N_{mul} + N_{add}) / 1000$$

处理器主频  $f_c$  的单位是 GHz,  $N_{mul}$  和  $N_{add}$  分别表示每个时钟周期执行多少乘法或加法操作

- ▶ 访存带宽

$$BW = f_m \times b \times \eta$$

访存带宽  $BW$  与存储器的主频  $f_m$ 、存储位宽  $b$ 、访存效率  $\eta$  的关系

# 性能评价

## ▶ 基准测试程序

### ▶ MLPerf

表 6.4 MLPerf Training 基准

应用程序	数据集	质量目标	用途
Resnet-50 v1.5	ImageNet(224 × 224)	75.9% Top-1 准确度	图像分类
SSD-ResNet34	COCO 2017	23% mAP	目标检测 (轻量级)
Mask R-CNN	COCO 2017	0.377 Box min AP, 0.339 Mask min AP	目标检测 (重量级)
GMNT	WMT 英语-德语	24.0 BLEU	循环翻译
Transformer	WMT 英语-德语	25.0 BLEU	非循环翻译
Mini Go		预训练的监测点	强化学习

表 6.5 MLPerf Inference 基准

应用程序	数据集	质量目标	用途
Resnet-50 v1.5	ImageNet(224 × 224)	达到单精度浮点 99% 的精度 (76.46% 的 Top-1 准确度)	图像分类
MobileNets-v1 224	ImageNet(224 × 224)	达到单精度浮点 98% 的精度 (71.68% 的 Top-1 准确度)	图像分类
SSD-ResNet34	COCO(1200 × 1200)	达到单精度浮点 99% 的精度 (0.20 mAP)	目标检测
SSD-MobileNets-v1	COCO(300 × 300)	达到单精度浮点 99% 的精度 (0.22 mAP)	目标检测
GMNT	WMT16	达到单精度浮点 99% 的精度 (23.9 BLEU)	机器翻译

# 影响性能的因素

$$T = \sum_i N_i \times C_i / f_c$$

$N_i$  表示该任务中第  $i$  类操作的数量  
 $C_i$  表示完成第  $i$  类操作所需要的时钟周期数  
 $f_c$  表示处理器的主频

- ▶ 减少  $C_i$
- ▶ 减少访存开销
- ▶ 多级并行

# 目录

- ▶ 深度学习处理器概述
- ▶ 目标算法分析
- ▶ 深度学习处理器DLP结构
- ▶ 优化设计
- ▶ 性能评价
- ▶ 其他加速器

# 其他加速器

## GPU

计算: SIMD (SIMT)

存储: 多层次

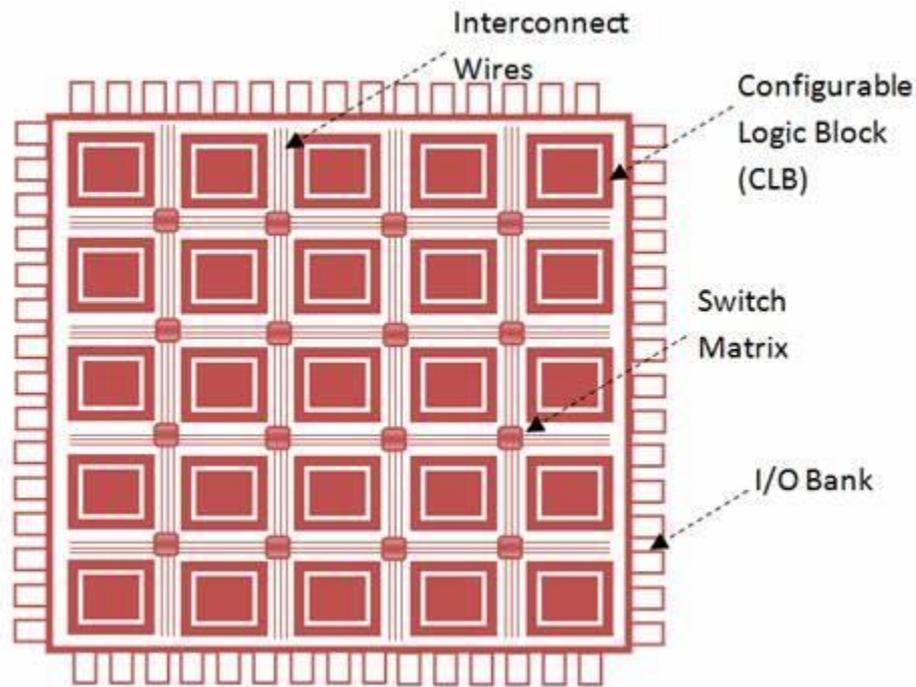
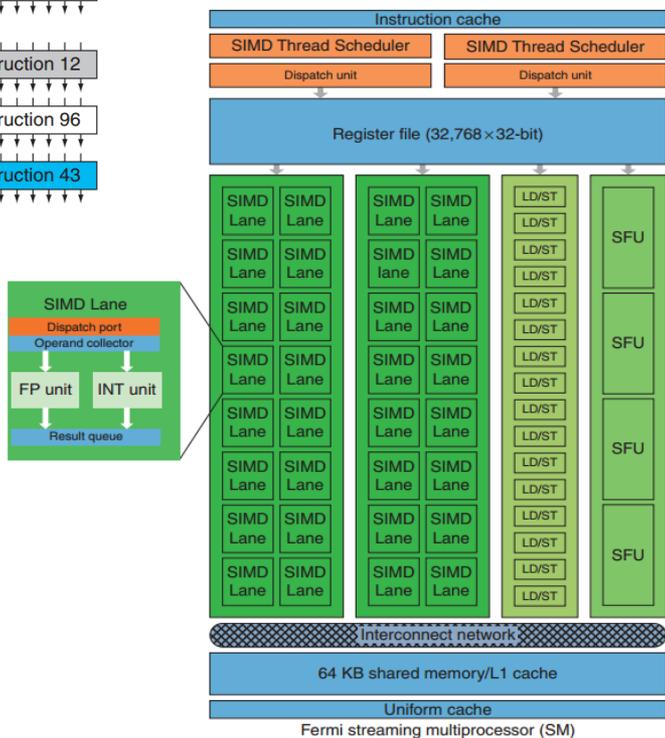
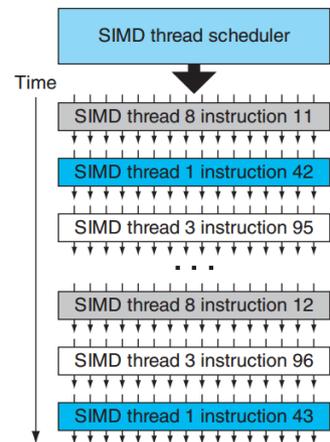
控制: SIMD指令

## FPGA

计算: 可配置CLB

存储: Block RAM

控制: 配置



# 其他加速器

类别	目标	速度	能效	灵活性
DLP	深度学习专用	高	高	深度学习领域通用
FPGA	通用的可编程电路	低	中	通用
GPU	SIMD 架构矩阵加速	中	低	矩阵类应用通用



# 谢谢大家!

欢迎关注课程公众号



中研大算所