AI ASIC: Design and Practice (ADaP)

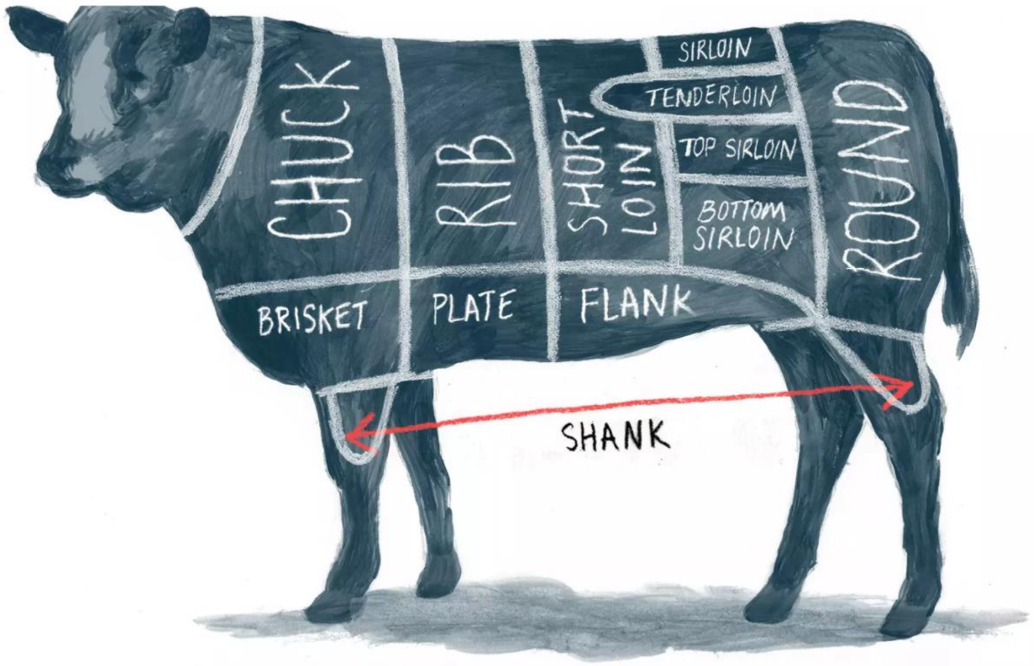
Fall 2024

Deep Learning Basics

燕博南

北京大学 PEKING UNIVERSITY

计算机工程中常用的深度学习算法基础



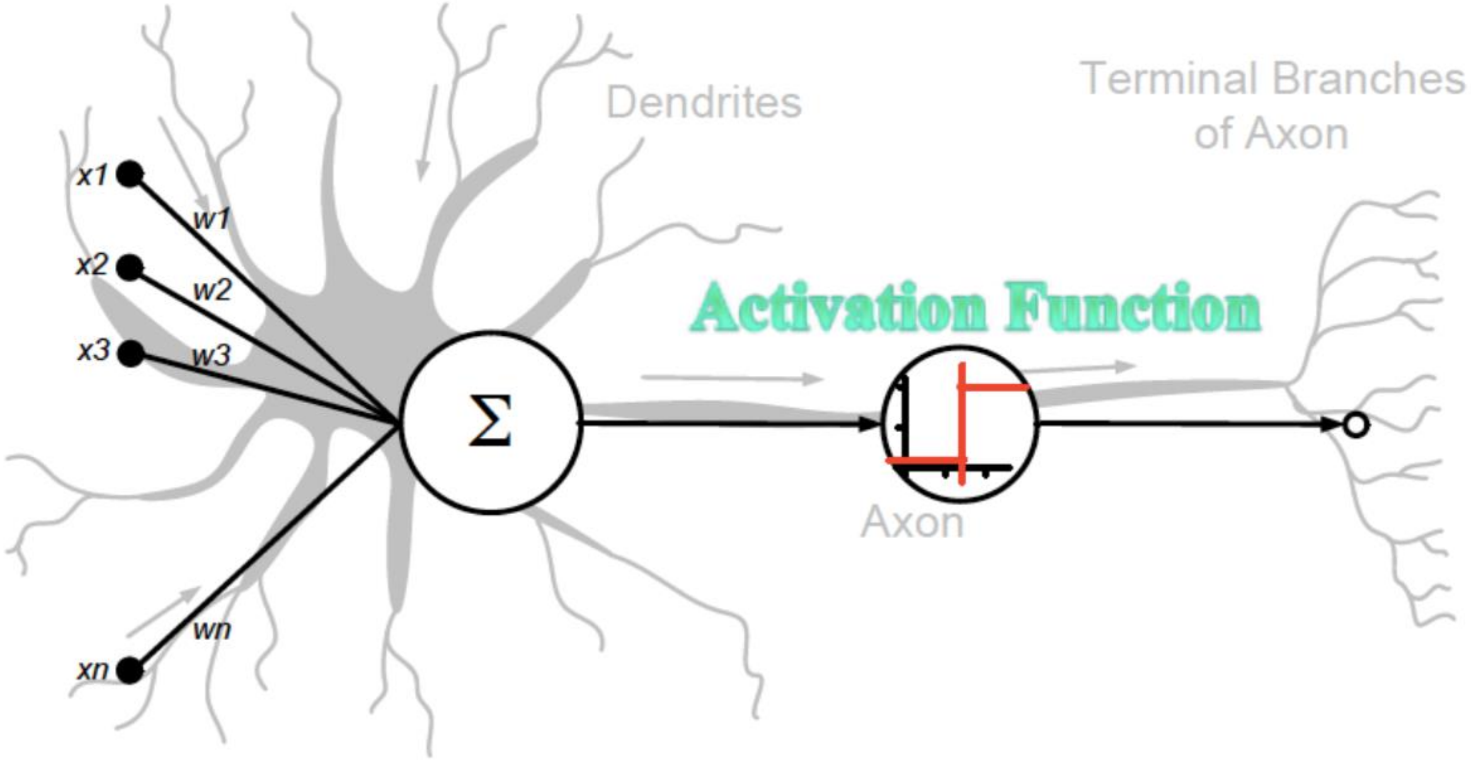
- Fully
- Convolution
- 其它算子
- Residual Network
- Depthwise Separable
- …

以“用”的角度去学，
ASIC定下目标去优化

Neural Networks

Human brain is made up of >100 billion **neurons**

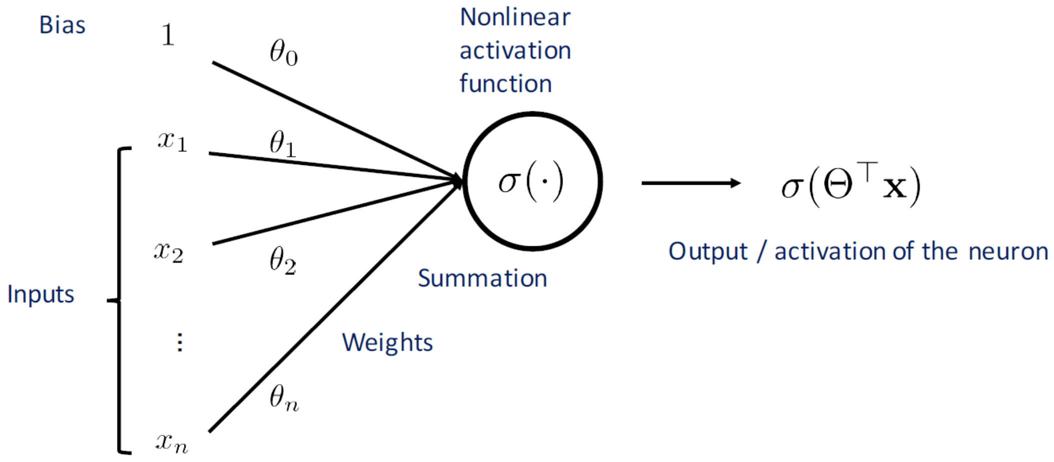
- Neurons **receive** electric signals at the dendrites and **send** them to the axon
- Dendrites can perform complex **non-linear** computations
- Synapses are not a single weight but a **complex** non-linear dynamical system



Artificial Neural Networks to Deep Neural Networks

Artificial neural networks

- A **simplified** version of biological neural network



(Synaptic) Weighting

First Layer
$$\boldsymbol{z} = \boldsymbol{W}^{(1)}\boldsymbol{x}$$
$$\boldsymbol{h} = \phi(\boldsymbol{z})$$

Activation

Second Layer
$$\boldsymbol{o} = \boldsymbol{W}^{(2)}\boldsymbol{h}$$
$$\boldsymbol{o}' = \phi(\boldsymbol{o})$$

...

Forward propagation in inference 推理

主要的计算量：　乘加 (multiply-accumulate, MAC)

Deep Learning

Forward propagation:

(Synaptic) Weighting

First Layer

$$\boldsymbol{z} = \boldsymbol{W}^{(1)}\boldsymbol{x}$$

$$\boldsymbol{h} = \phi(\boldsymbol{z})$$

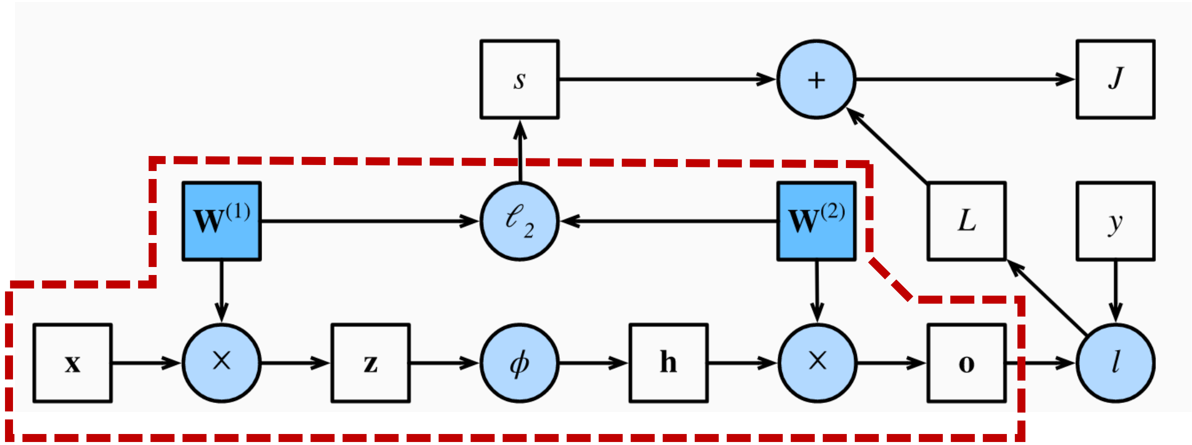
Activation

Second Layer

$$\boldsymbol{o} = \boldsymbol{W}^{(2)}\boldsymbol{h}$$

$$\boldsymbol{o}' = \phi(\boldsymbol{o})$$

Computational Graph



Inference

Evaluate the network:

$$L = l(\boldsymbol{o}, y)$$

Loss function

$$s = \frac{\lambda}{2}(||W^{(1)}||^2 + ||W^{(2)}||^2)$$

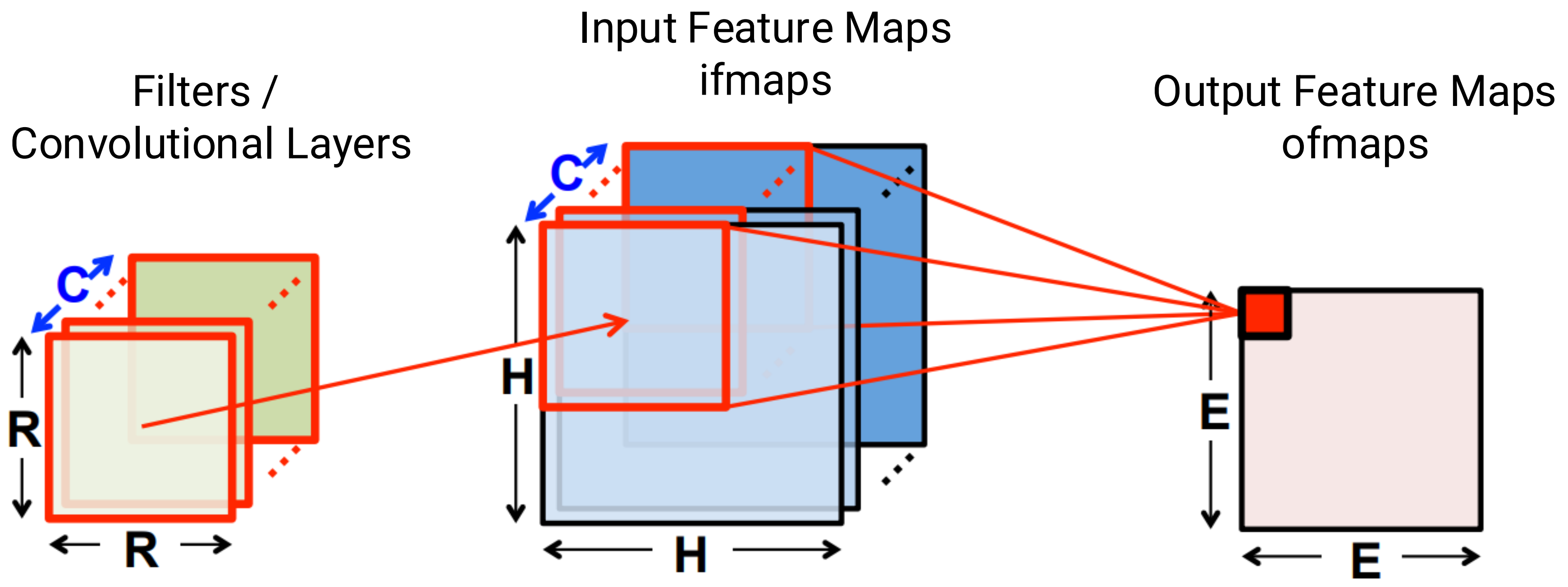
Regularization

Training: Minimize (L+s)

Convolutional Layer

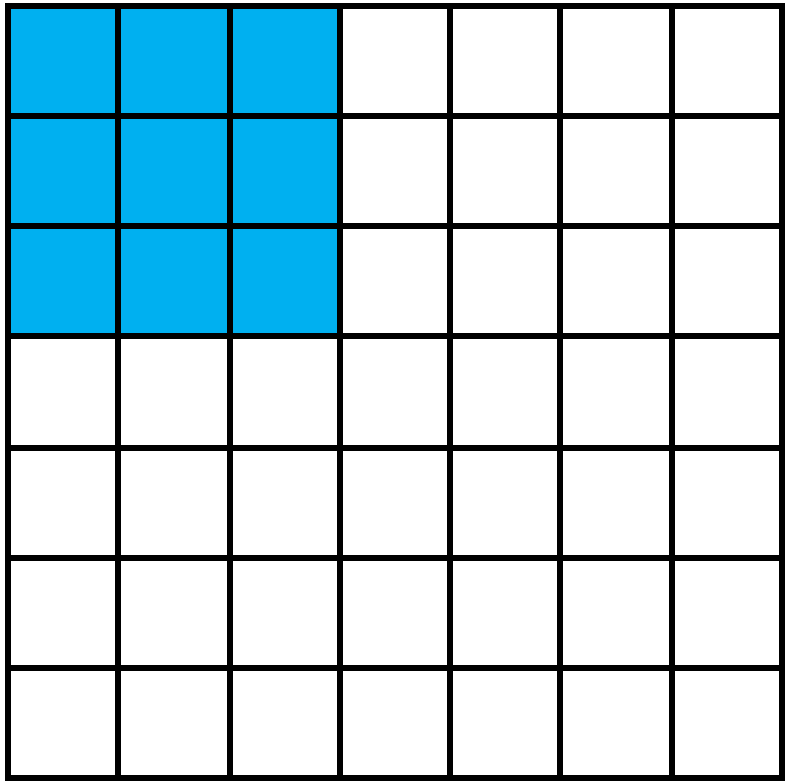
$$\mathbf{O}[z][u][x][y] = \mathbf{B}[u] + \sum_{k=0}^{C-1}\sum_{i=0}^{R-1}\sum_{j=0}^{R-1} \mathbf{I}[z][k][Ux+i][Uy+j] \times \mathbf{W}[u][k][i][j],$$

$$0 \le z < N, 0 \le u < M, 0 \le x, y < E, E = (H - R + U)/U.$$



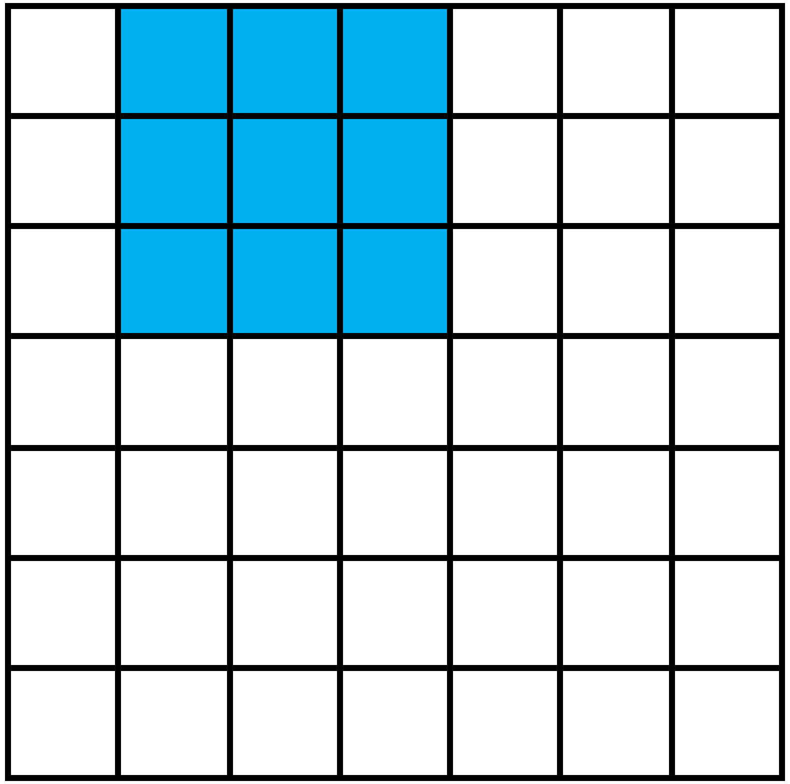
N: batch size
M: # of ofmap channels
C: # of ifmap filter channels
H: ifmap height/width
R: filter plan height/width
E: ofmap height/width

Convolutional Layer - Example



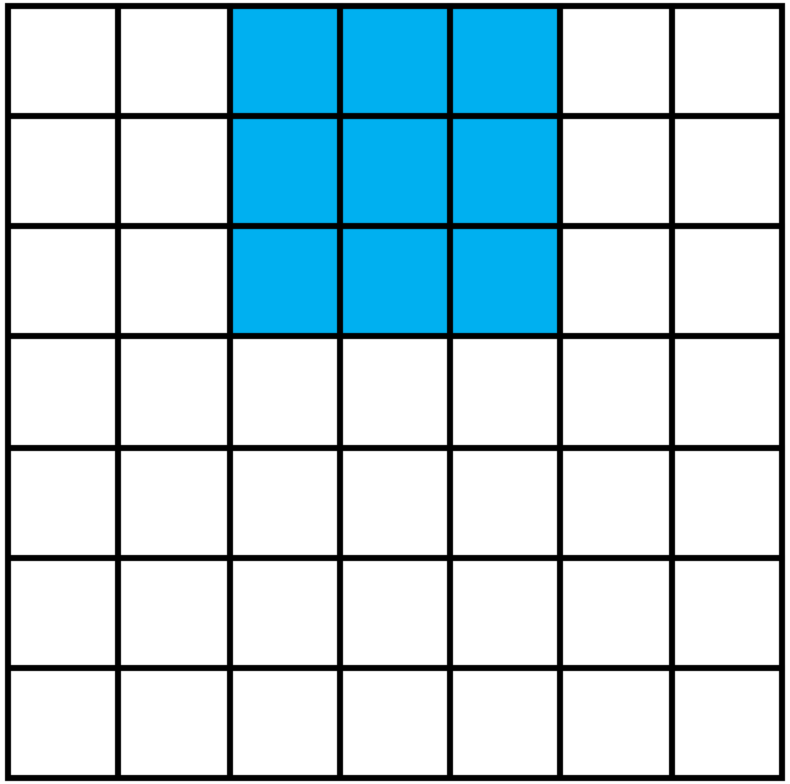
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Convolutional Layer - Example



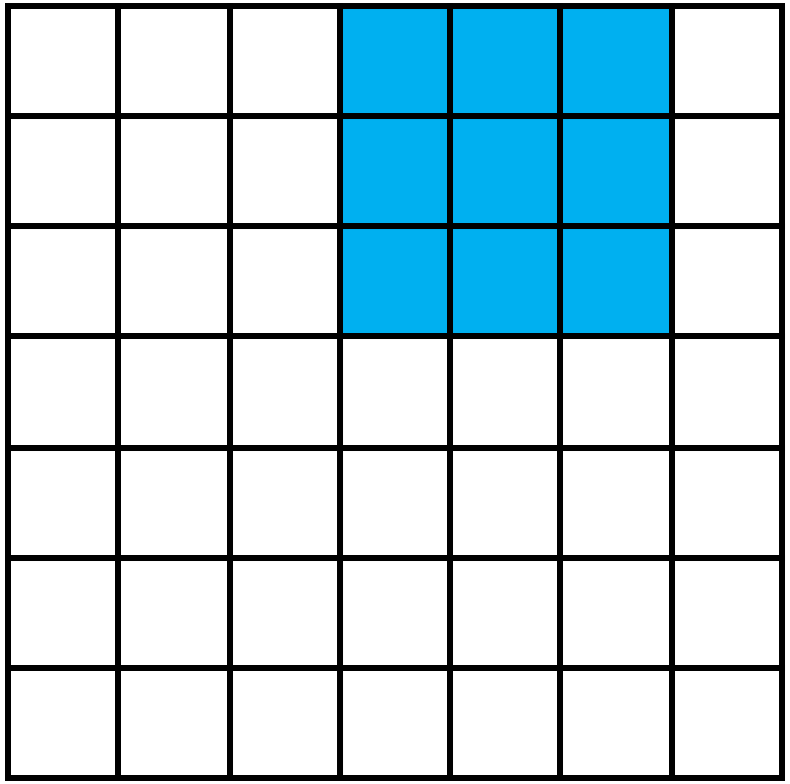
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Convolutional Layer - Example



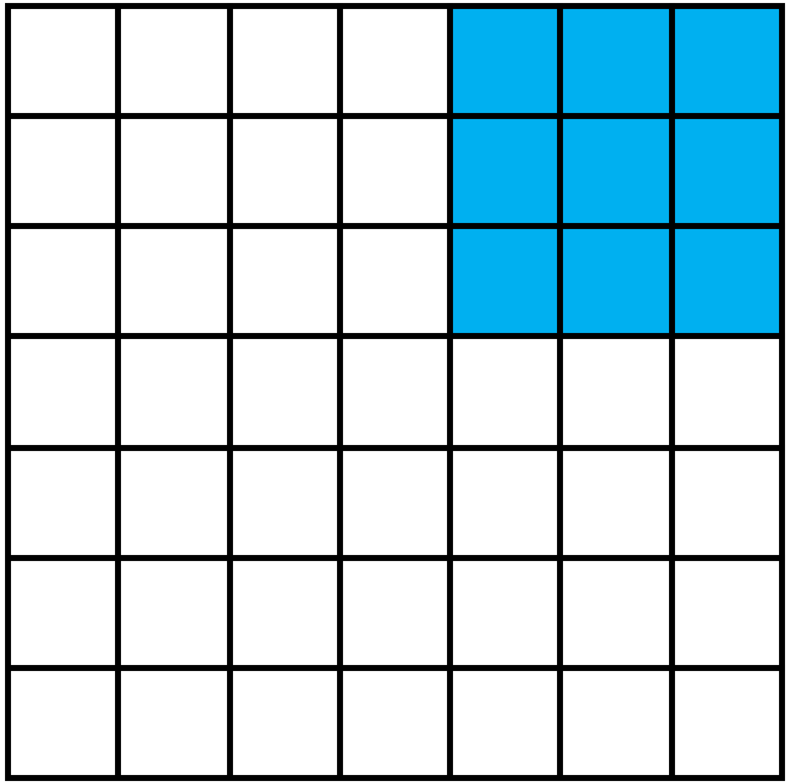
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Convolutional Layer - Example



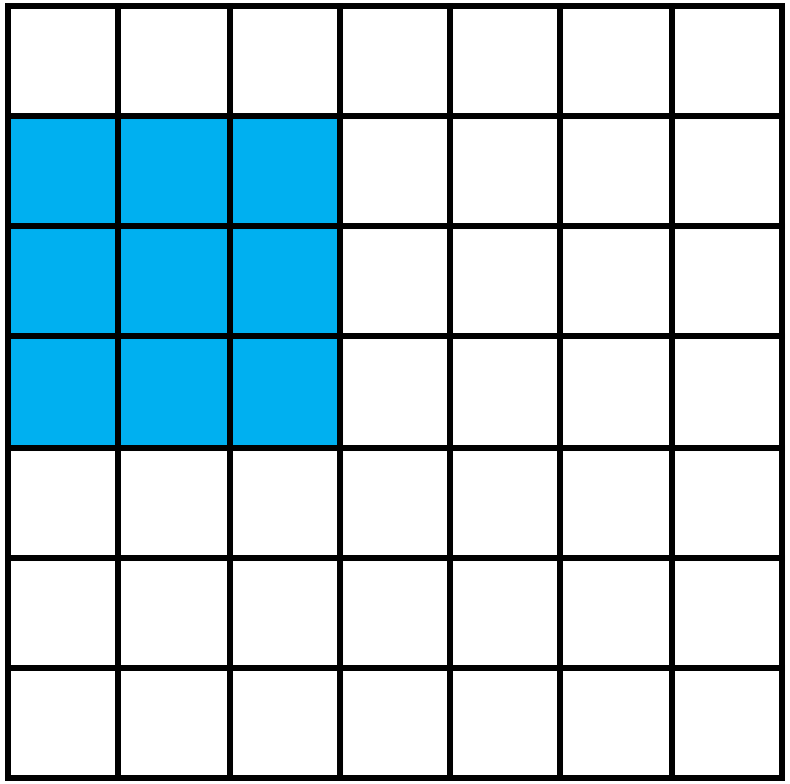
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Convolutional Layer - Example



7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Convolutional Layer - Example



7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Output: 5×5

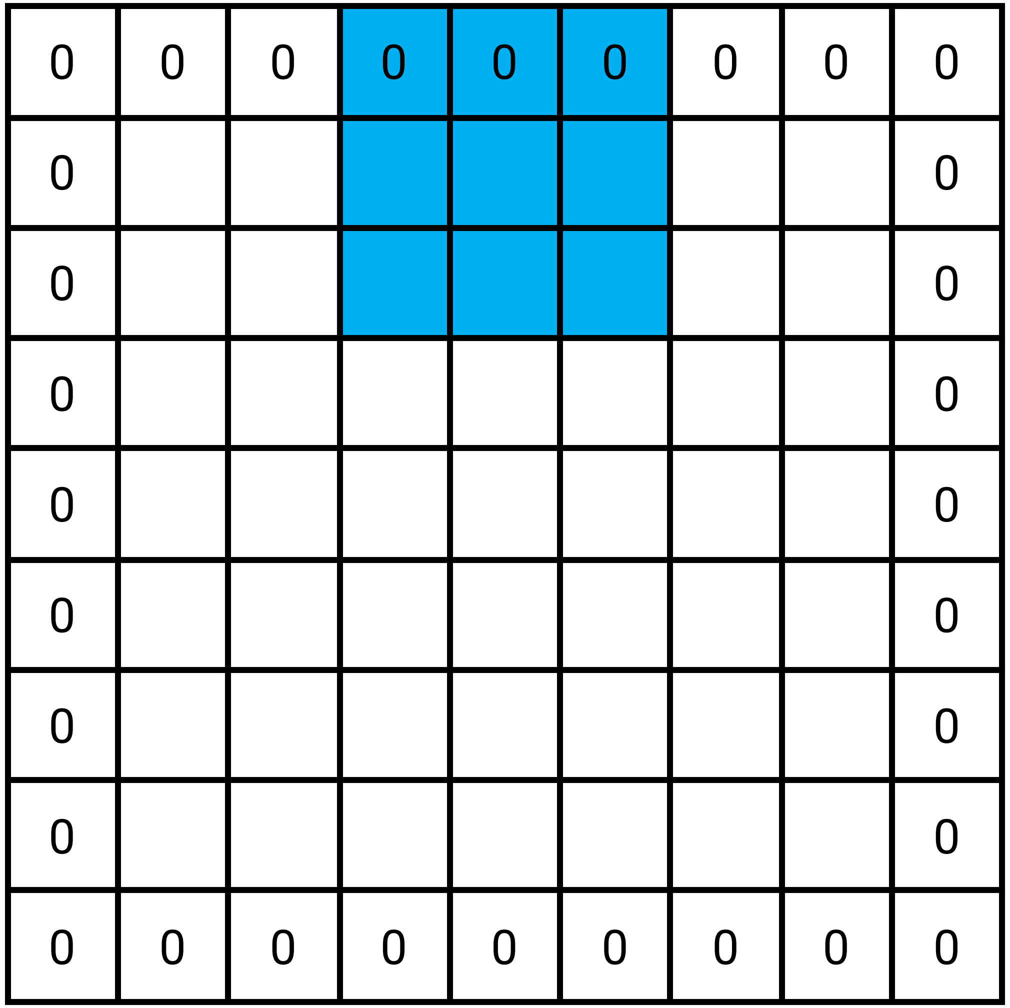
Convolutional Layer - Example

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

Padding

7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 3**

Convolutional Layer - Example



7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 3**

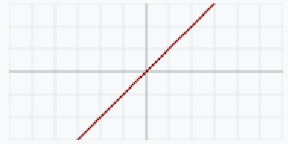
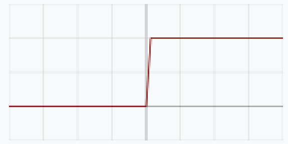
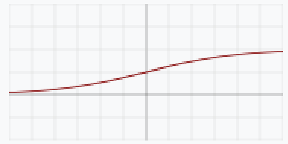
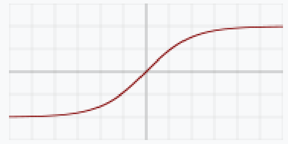
Convolutional Layer - Example

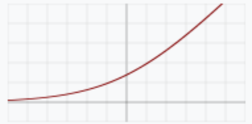
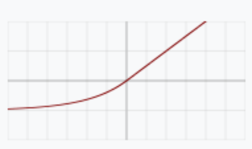
0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 3**

其它的算子-1

- Activation
  - Relu
  - Tanh
  - Sigmoid
  - Softmax
  - …

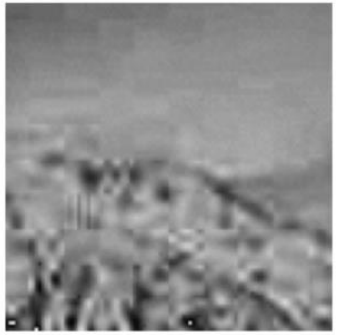
Identity	
Binary step	
Logistic, sigmoid, or soft step	
Hyperbolic tangent (tanh)	

Rectified linear unit (ReLU)[7]	
Gaussian Error Linear Unit (GELU)[4]	
Softplus[8]	
Exponential linear unit (ELU)[9]	

硬件上如何最简单地实现？

其它的算子-2

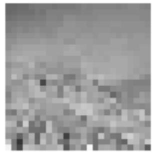
- Pooling
  - Max pooling
  - Average pooling
  - Stochastic pooling
  - …



12	13	30	1
9	12	2	0
33	98	37	4
100	73	25	12

Example of max pooling:

13	30
100	37



Example of average pooling:

12	8
76	20

其它的算子-3

- Normalization
  - Batch Normalization (BN)
  - Group Normalization (GN)
  - …

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

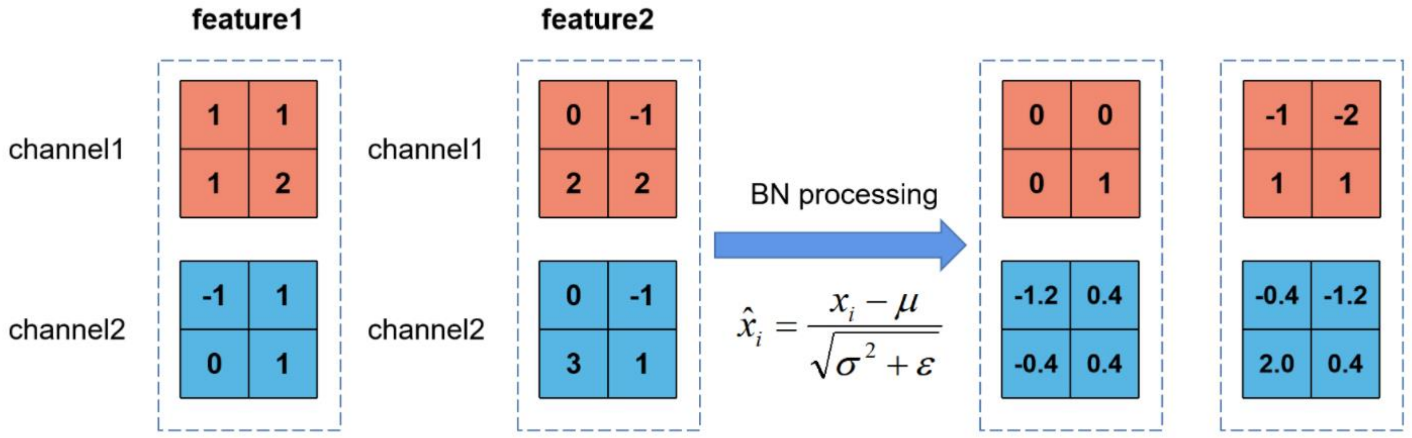
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Example of BN:



$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$x^{(1)} = \{1,1,1,2,0,-1,2,2\}$$

$$x^{(2)} = \{-1,1,0,1,0,-1,3,1\}$$

$$\mu_1 = \frac{1}{m}\sum_{i=1}^{m} x_i^{(1)} = 1 \qquad \sigma_1^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i^{(1)} - \mu_1)^2 = 1$$

$$\mu_2 = \frac{1}{m}\sum_{i=1}^{m} x_i^{(2)} = 0.5 \qquad \sigma_2^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i^{(2)} - \mu_2)^2 = 1.5$$

$$\Rightarrow \quad \mu = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} \qquad \sigma^2 = \begin{bmatrix} 1 \\ 1.5 \end{bmatrix}$$

Some Famous and Basic NN

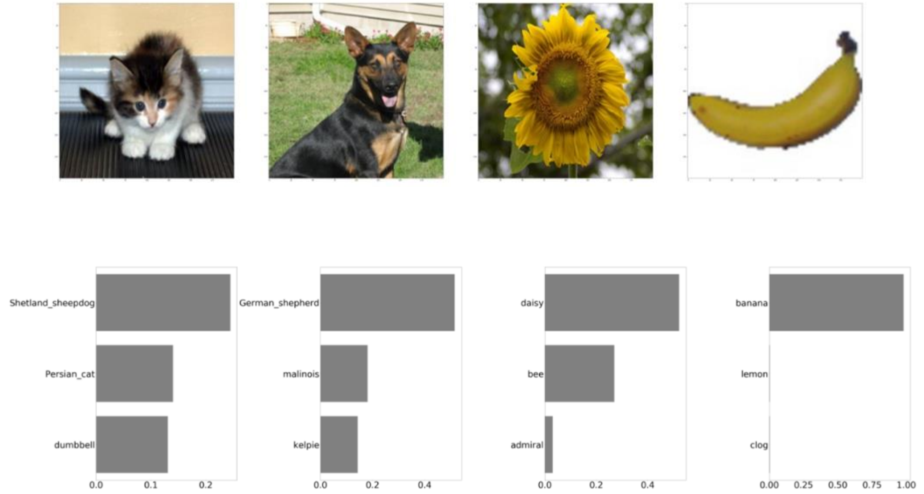
- VGG16/19
- Resnets
- Depthwise Convolution Group
  - InceptionNets
  - XceptionNets
  - MobileNet

VGG16

北京大学 PEKING UNIVERSITY

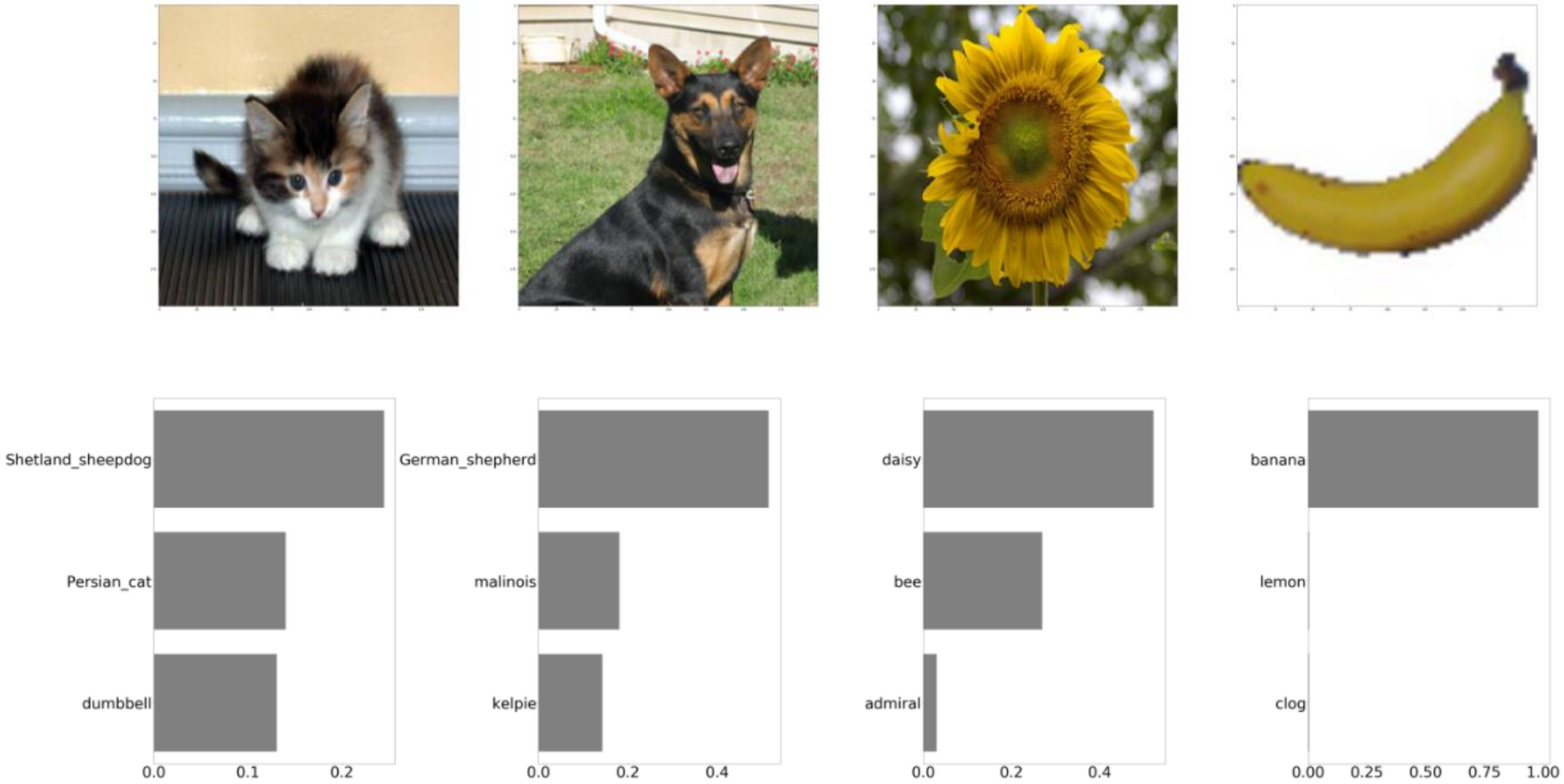
Network Architecture

```
_input = Input((224,224,1))

conv1  = Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(_input)
conv2  = Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(conv1)
pool1  = MaxPooling2D((2, 2))(conv2)

conv3  = Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(pool1)
conv4  = Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(conv3)
pool2  = MaxPooling2D((2, 2))(conv4)

conv5  = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(pool2)
conv6  = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(conv5)
conv7  = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(conv6)
pool3  = MaxPooling2D((2, 2))(conv7)

conv8  = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(pool3)
conv9  = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv8)
conv10 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv9)
pool4  = MaxPooling2D((2, 2))(conv10)

conv11 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(pool4)
conv12 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv11)
conv13 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv12)
pool5  = MaxPooling2D((2, 2))(conv13)

flat   = Flatten()(pool5)
dense1 = Dense(4096, activation="relu")(flat)
dense2 = Dense(4096, activation="relu")(dense1)
output = Dense(1000, activation="softmax")(dense2)
```



VGG16

Performance

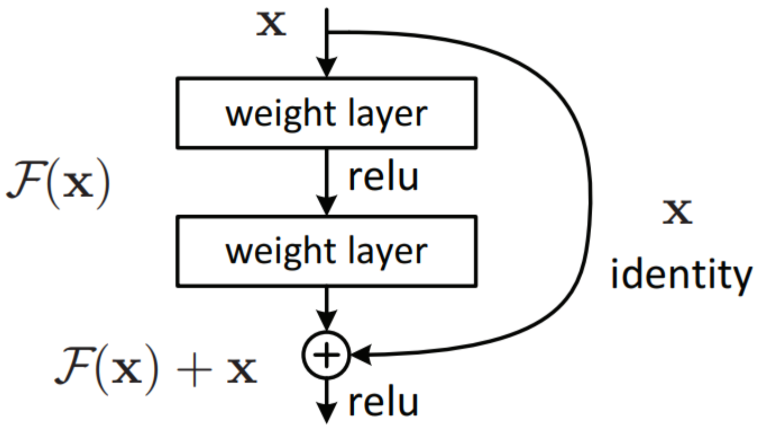


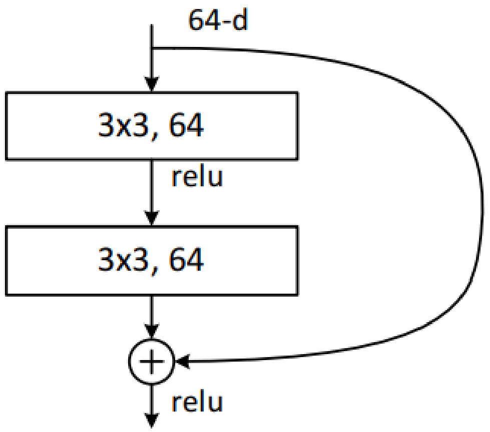
Residual Network

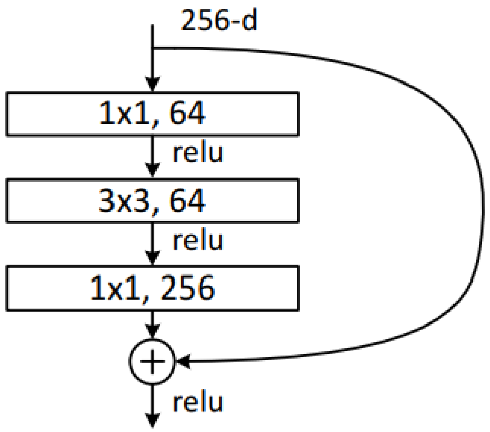
When DNN goes deeper:

- Network becomes difficult to optimize
- Vanishing / Exploding Gradients
- Degradation Problem (accuracy first saturates and then degrades)

Examples:







注意BN的位置!

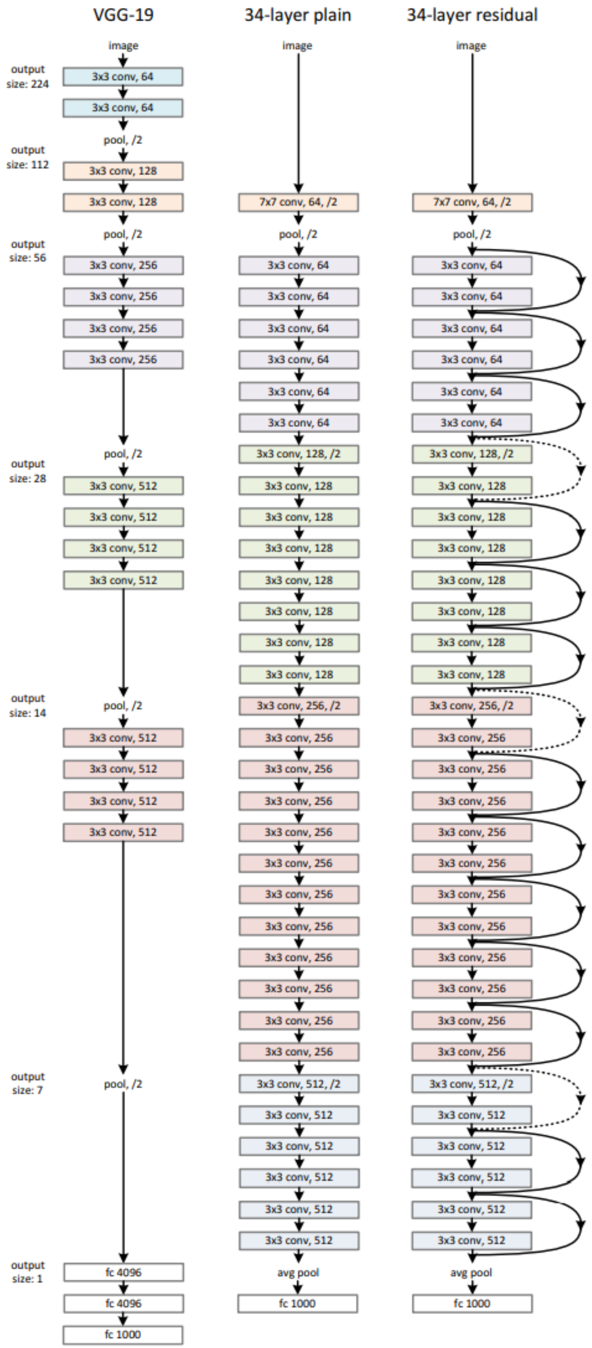
Bottleneck block

He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

ResNet

北京大学 PEKING UNIVERSITY

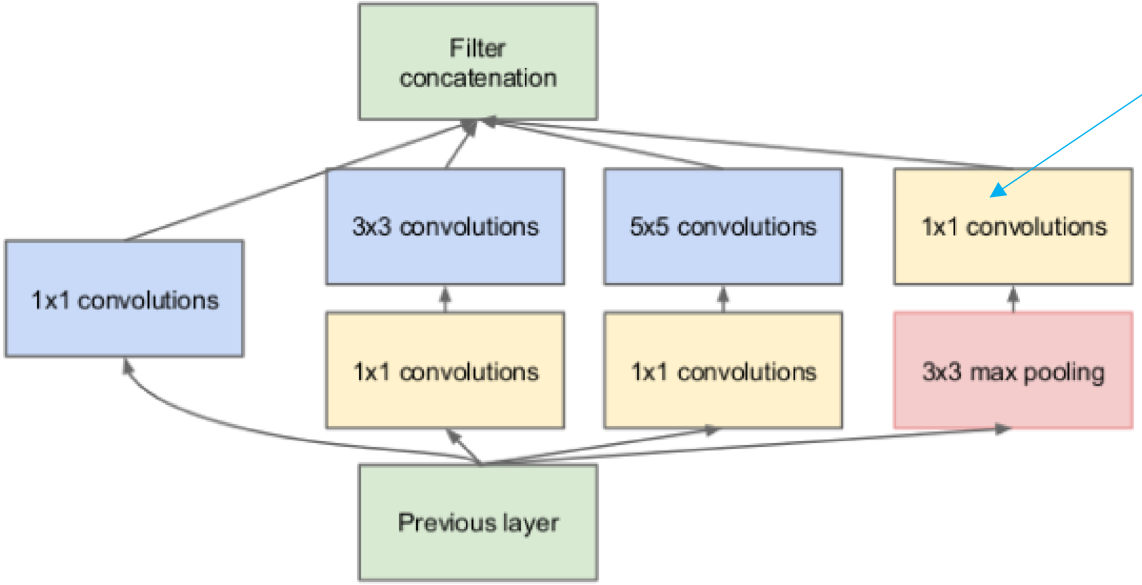
Dotted lines means
Increased dimension



InceptionNets

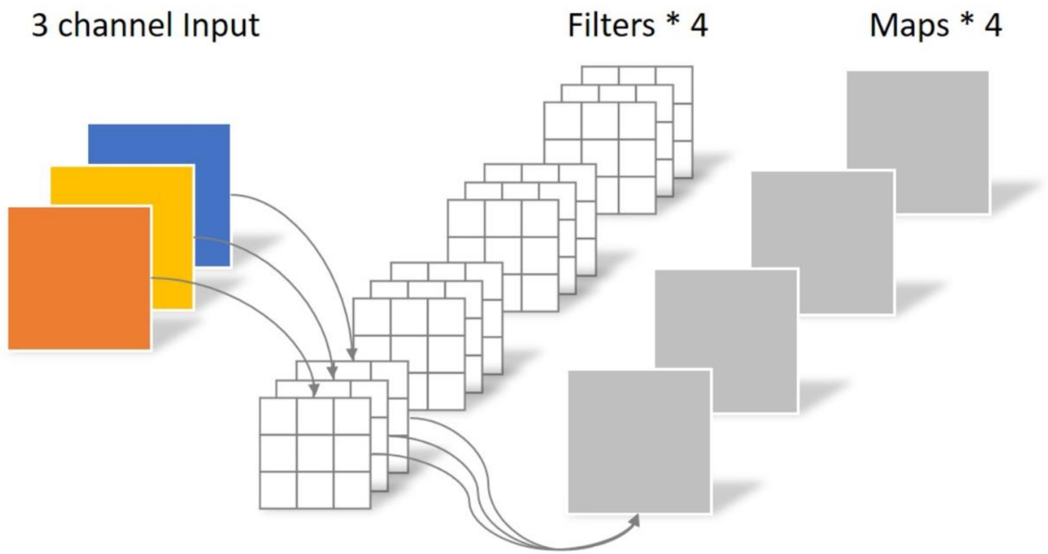
- Also known as GoogleNet consists of total 22 layers and was the winning model of 2014 image net challenge.

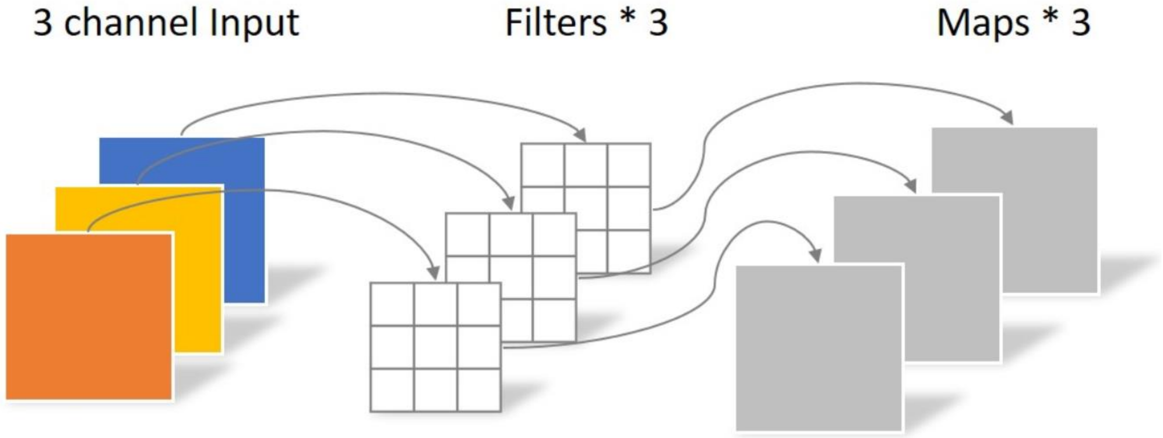
Inception Module:



perform dimensionality reduction

Depthwise Separable Convolution





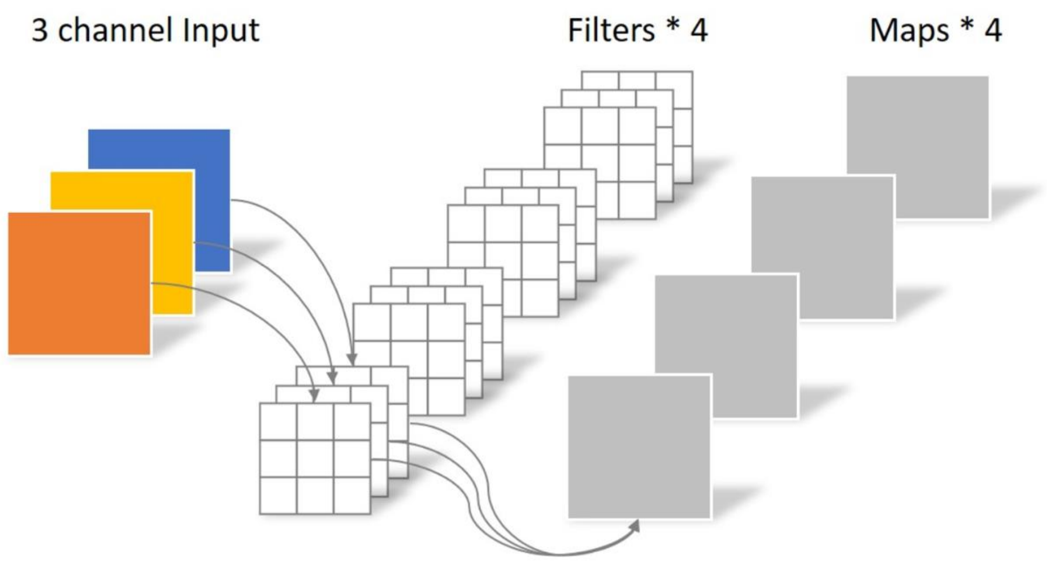
Parameter Number： 4*3*3*3=108

Parameter Number： 3*3*3=27

arXiv:1704.04861 (2017)

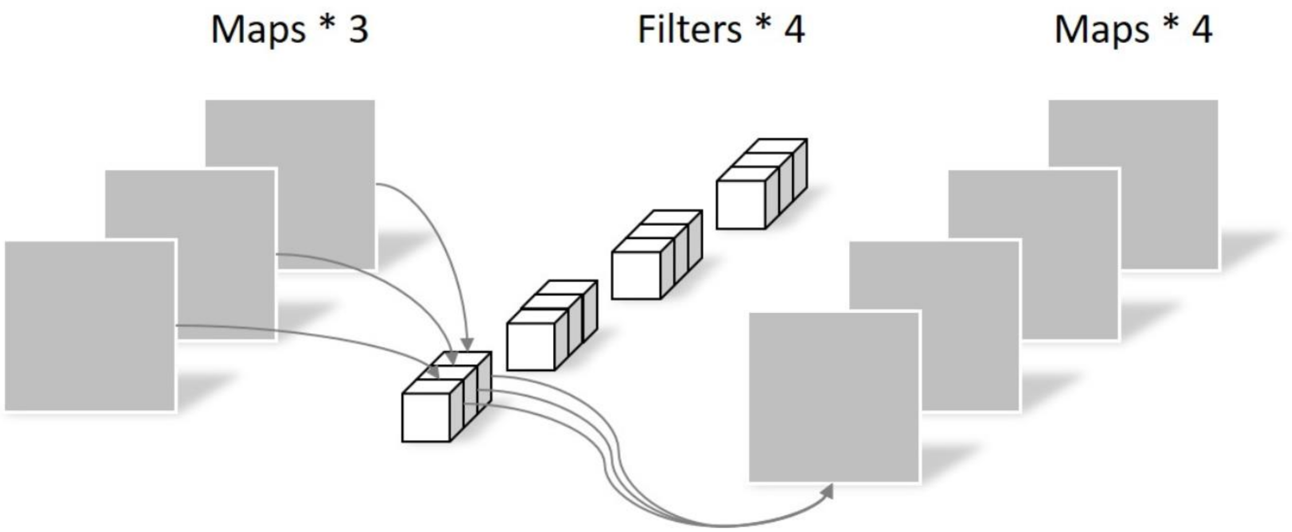
Depthwise Separable Convolution

Normal Convolution



Parameter Number： 4*3*3*3=108

Pointwise Convolution

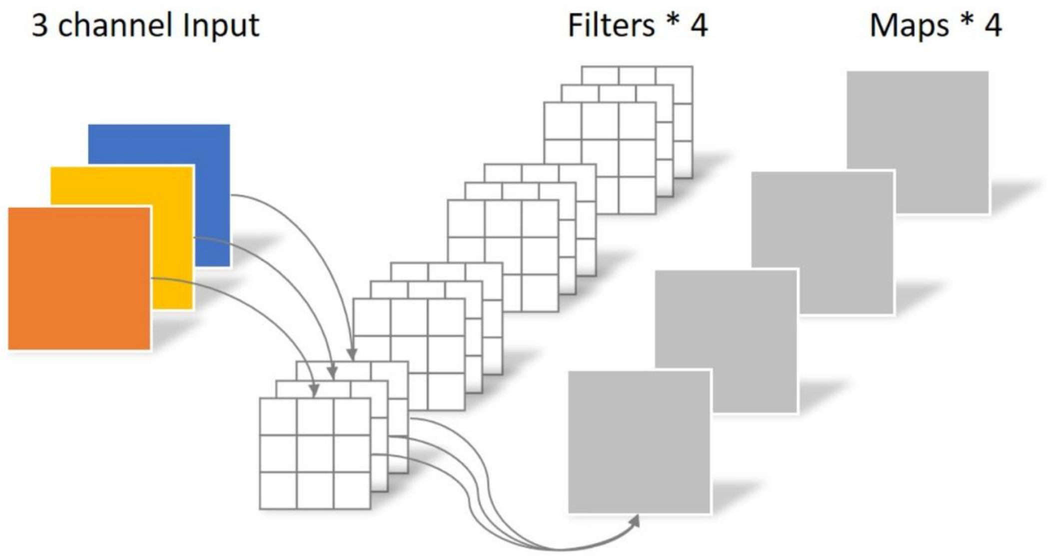


Parameter Number： 4*3*1*1=12

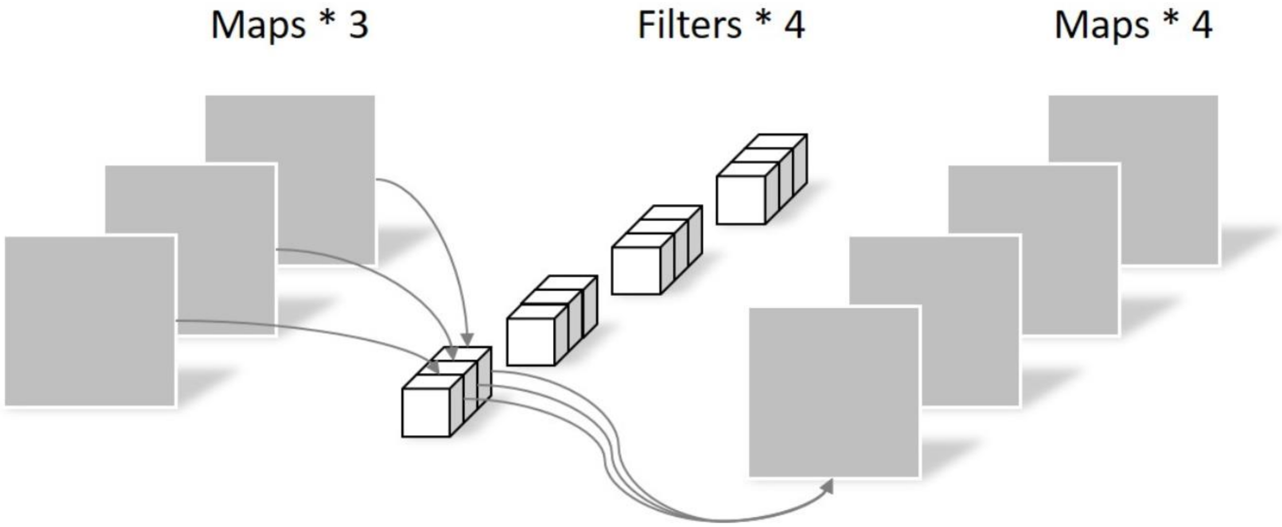
Depthwise Separable Convolution

PEKING UNIVERSITY

Normal Convolution



Pointwise Convolution

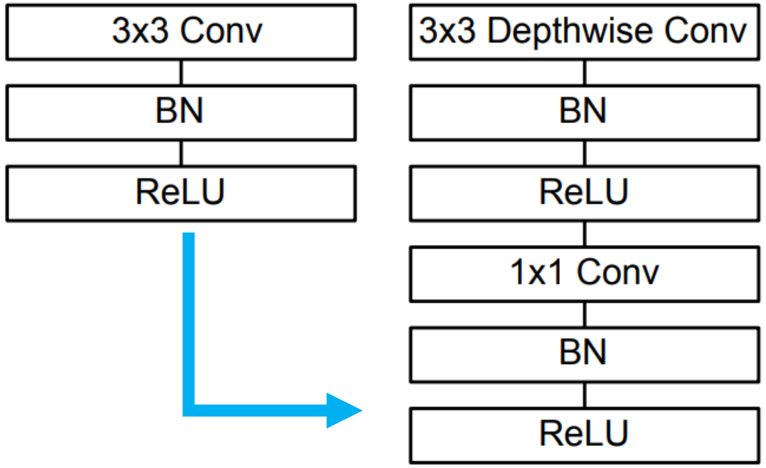


Parameter Number： 4*3*3*3=108

Parameter Number： 4*3*1*1=12

arXiv:1704.04861 (2017)

MobileNet



Parameter Number： 4*3*3*3=108

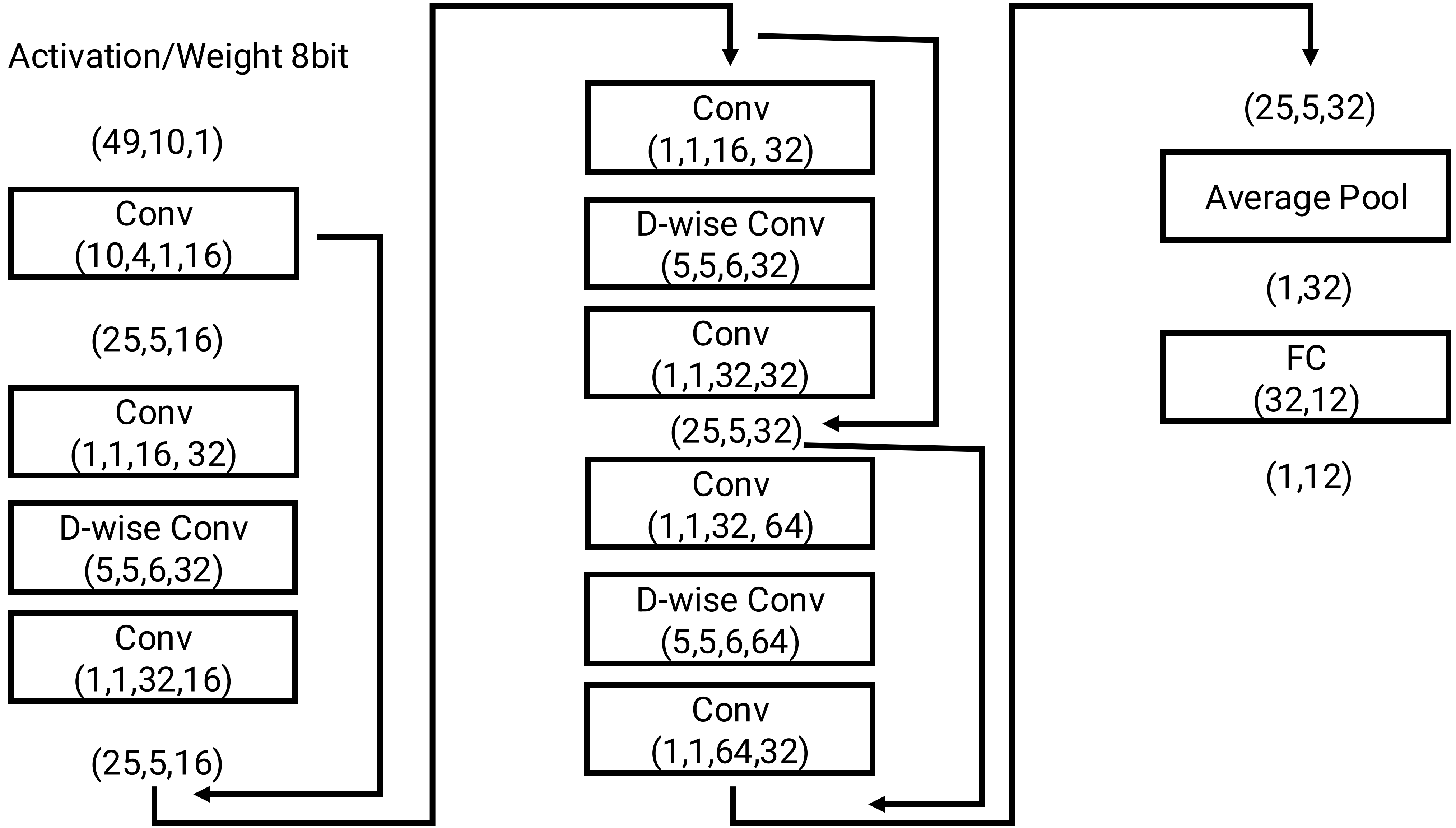
Parameter Number： Depthwise+Pointwise=27+12=39

减少计算量!

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

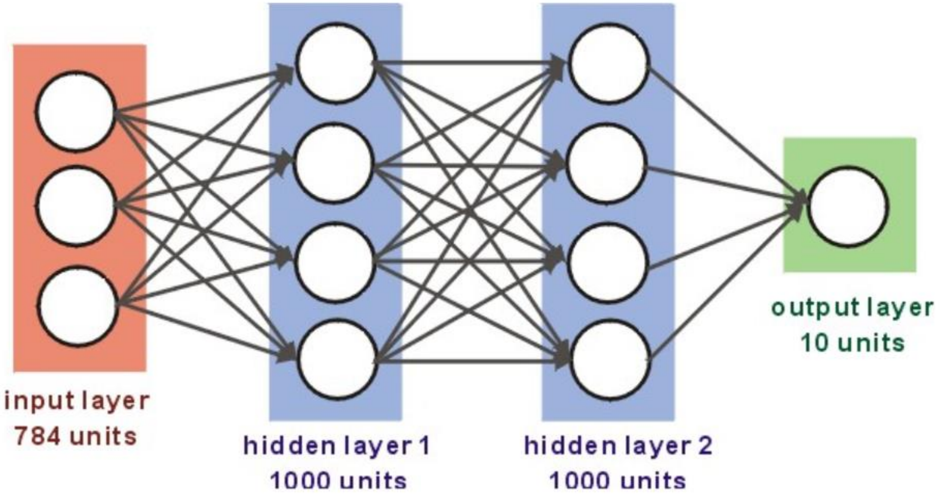
arXiv:1704.04861 (2017)

Example NN Architecture



NN Coding Example … in C

- C/C++ is the reliable contract between software and hardware
- Implement in CPU
- Ref: [MNIST - CNN coded in C - [0.995] | Kaggle]



Initial Setup

```
int layerSizes[10] = {0,0,0,0,0,0,784,1000,1000,10};
float* layers[10] = {0};
float* errors[10] = {0};
float*  weights[10] = {0};
// INITIALIZATION
void initNet(){
    // ALOCATE MEMORY
    layers[0] = (float*)malloc((layerSizes[0]+1) * sizeof(float));
    errors[0] = (float*)malloc(layerSizes[0] * sizeof(float));
    for (i=1;i<10;i++){
        layers[i] = (float*)malloc((layerSizes[i]+1) * sizeof(float));
        errors[i] = (float*)malloc(layerSizes[i] * sizeof(float));
        weights[i] = (float*)malloc(layerSizes[i] * (layerSizes[i-1]+1) * sizeof(float));
    }
    // RANDOMIZE WEIGHTS AND BIAS
    float scale;
    for (i=0;i<10;i++) layers[i][layerSizes[i]]=1.0;
    for (j=1;j<10;j++){
        scale = 2.0 * sqrt(6.0/(layerSizes[j] + layerSizes[j-1]));
        if (j!=9 && activation==1) scale = scale * 1.41; // RELU
        else if (j!=9) scale = scale * 4.0; // TANH
        for (i=0;i<layerSizes[j] * (layerSizes[j-1]+1);i++)
            weights[j][i] = scale * ( (float)rand()/RAND_MAX - 0.5 );
        for (i=0;i<layerSizes[j];i++) // BIASES
            weights[j][(layerSizes[j-1]+1)*(i+1)-1] = 0.0;
    }
}
```

Forward Pass (Inference)

北京大学 PEKING UNIVERSITY

```
int activation = 1; //ReLU
// FORWARD PROPAGATION
int forwardProp(int x){
    int i,j,k,imax;
    float sum, esum, max;
    // INPUT LAYER - RECEIVES 28X28 IMAGES
    for (i=0;i<784;i++) layers[10-numLayers][i] = trainImages[x][i];
    // HIDDEN LAYERS - RELU ACTIVATION
    for (k=11-numLayers;k<9;k++)
    for (i=0;i<layerSizes[k];i++){
        sum = 0.0;
        for (j=0;j<layerSizes[k-1]+1;j++)
            sum += layers[k-1][j]*weights[k][i*(layerSizes[k-1]+1)+j];
        if (activation==1) layers[k][i] = ReLU(sum);
        else if (activation==2) layers[k][i] = TanH(sum);
    }
    // OUTPUT LAYER - SOFTMAX ACTIVATION
    esum = 0.0;
    for (i=0;i<layerSizes[9];i++){
        sum = 0.0;
        for (j=0;j<layerSizes[8]+1;j++)
            sum += layers[8][j]*weights[9][i*(layerSizes[8]+1)+j];
        if (sum>30) return -1; //GRADIENT EXPLODED
        layers[9][i] = exp(sum);
        esum += layers[9][i];
    }
    // SOFTMAX FUNCTION
    max = layers[9][0]; imax=0;
    for (i=0;i<layerSizes[9];i++){
        if (layers[9][i]>max){
            max = layers[9][i];
            imax = i;
        }
        layers[9][i] = layers[9][i] / esum;
    }
    return imax;
}
```

Here The Accelerator Comes!

```
int activation = 1; //ReLU
// FORWARD PROPAGATION
int forwardProp(int x){
    int i,j,k,imax;
    float sum, esum, max;
    // INPUT LAYER - RECEIVES 28X28 IMAGES
    for (i=0;i<784;i++) layers[10-numLayers][i] = trainImages[x][i];
    // HIDDEN LAYERS - RELU ACTIVATION
    for (k=11-numLayers;k<9;k++)
    for (i=0;i<layerSizes[k];i++){
        sum = 0.0;
        for (j=0;j<layerSizes[k-1]+1;j++)
            sum += layers[k-1][j]*weights[k][i*(layerSizes[k-1]+1)+j];
        if (activation==1) layers[k][i] = ReLU(sum);
        else if (activation==2) layers[k][i] = TanH(sum);
    }
```

```
    // OUTPUT LAYER - SOFTMAX ACTIVATION
    esum = 0.0;
    for (i=0;i<layerSizes[9];i++){
        sum = 0.0;
        for (j=0;j<layerSizes[8]+1;j++)
            sum += layers[8][j]*weights[9][i*(layerSizes[8]+1)+j];
        if (sum>30) return -1; //GRADIENT EXPLODED
        layers[9][i] = exp(sum);
        esum += layers[9][i];
    }
    // SOFTMAX FUNCTION
    max = layers[9][0]; imax=0;
    for (i=0;i<layerSizes[9];i++){
        if (layers[9][i]>max){
            max = layers[9][i];
            imax = i;
        }
        layers[9][i] = layers[9][i] / esum;
    }
    return imax;
}
```

```
Sum = AcceleratorVMM(*inputs, *weights, *outputs)
```

C Function Access PIM Function

```
Sum = AcceleratorVMM(*inputs, *weights, *outputs)
```

C Program: XXX.c

```
#include <stdio.h>

extern int AcceleratorVMM(*inputs, *weights, *outputs);

Int function () {
…
Sum = AcceleratorVMM(*inputs, *weights, *outputs)
…
}
```

C Function Access PIM Function

PEKING UNIVERSITY 北京大学

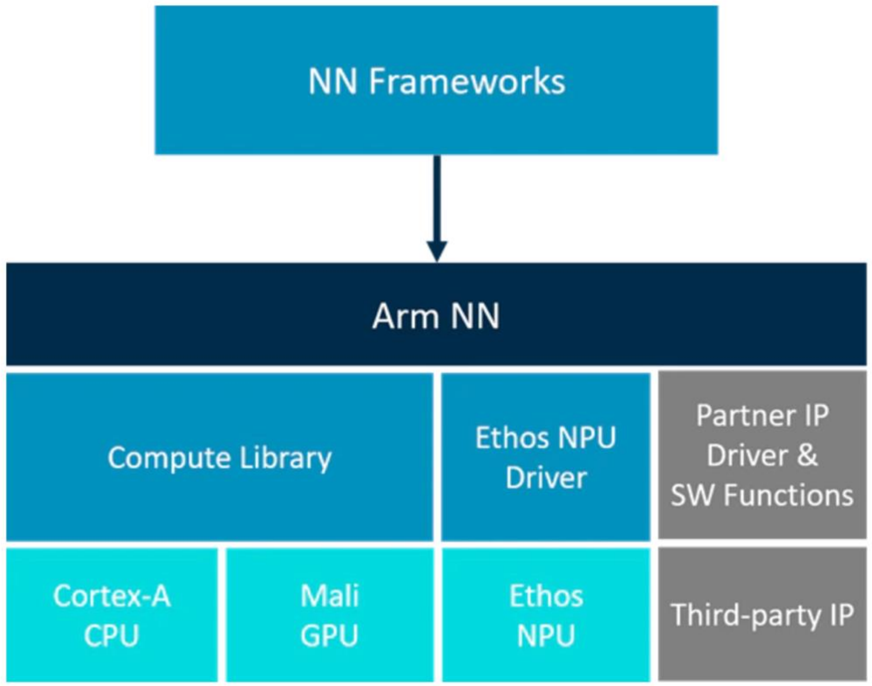
```
Sum = AcceleratorVMM(*inputs, *weights, *outputs)
```

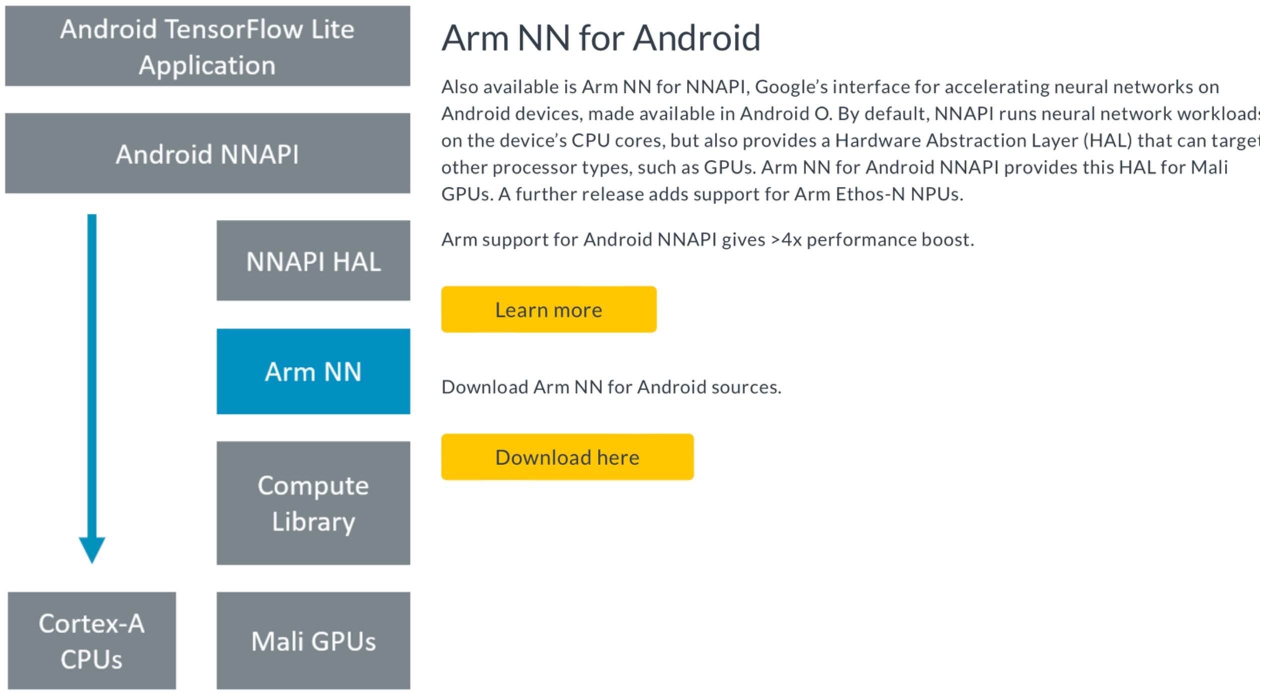
C Intrinsic: XXX.s

```
.section text

.global AcceleratorVMM
.type AcceleratorVMM, @function

AcceleratorVMM:
        sw a0, a1
        ld a0, a1
        …
```

Is this real?

- Yes!

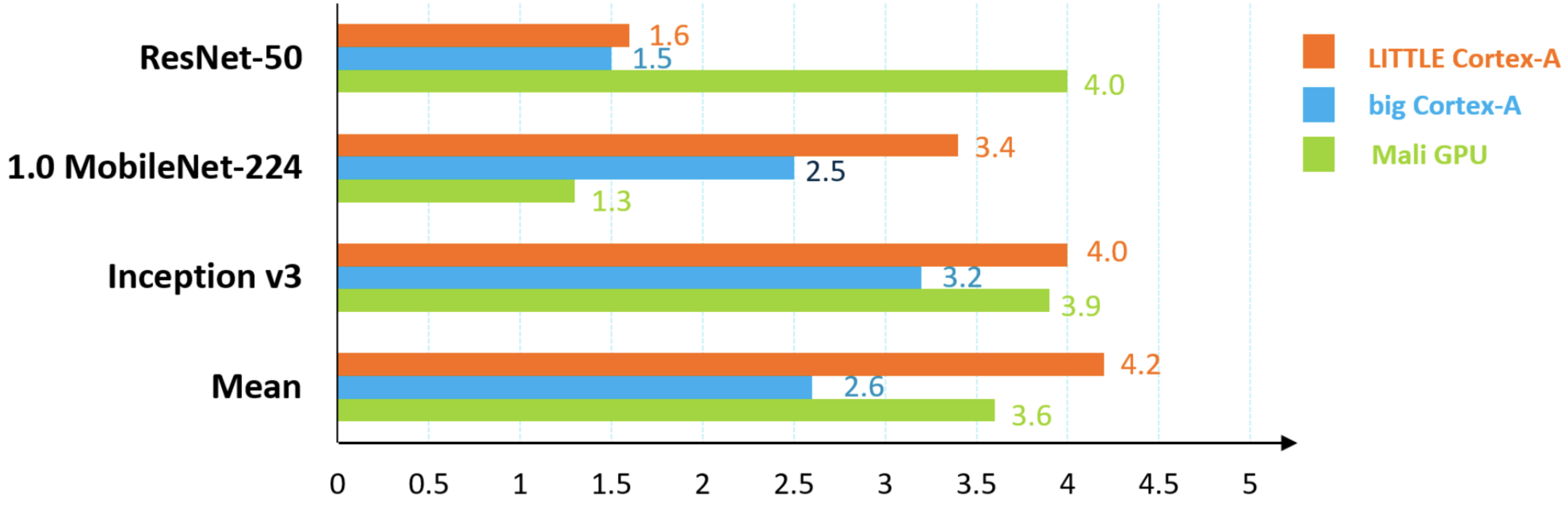




https://developer.arm.com/ip-products/processors/machine-learning/arm-nn.

Real products

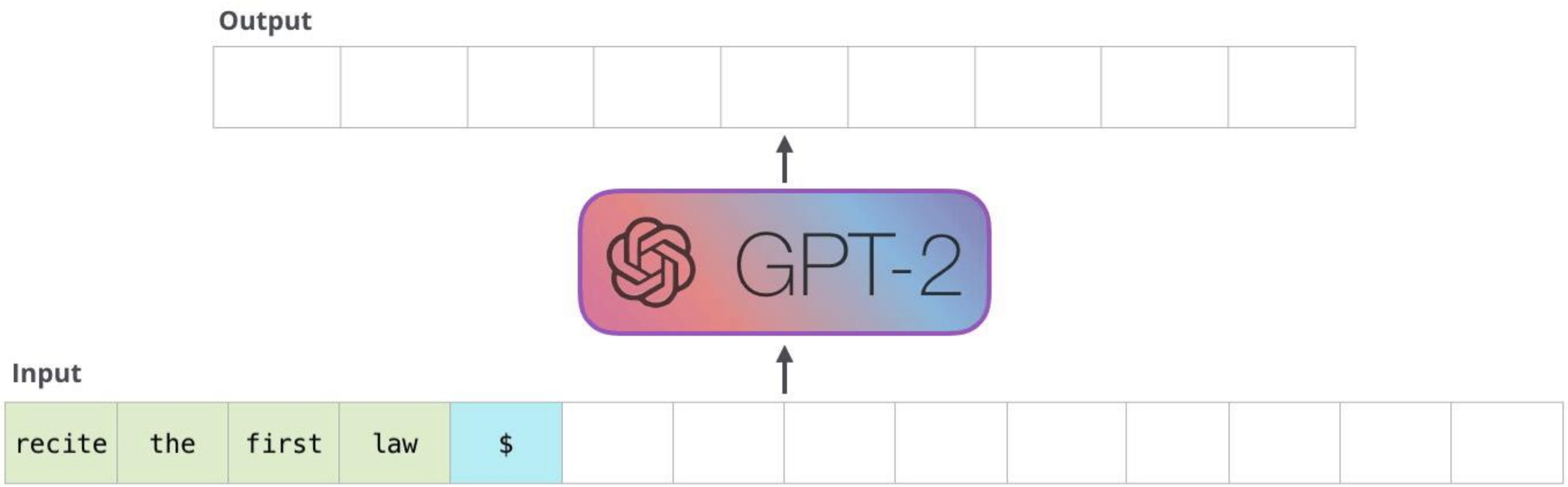
Arm NN performance relative to other NN frameworks



Mean improvements of Arm NN relative to multiple industry inference engines on CPU

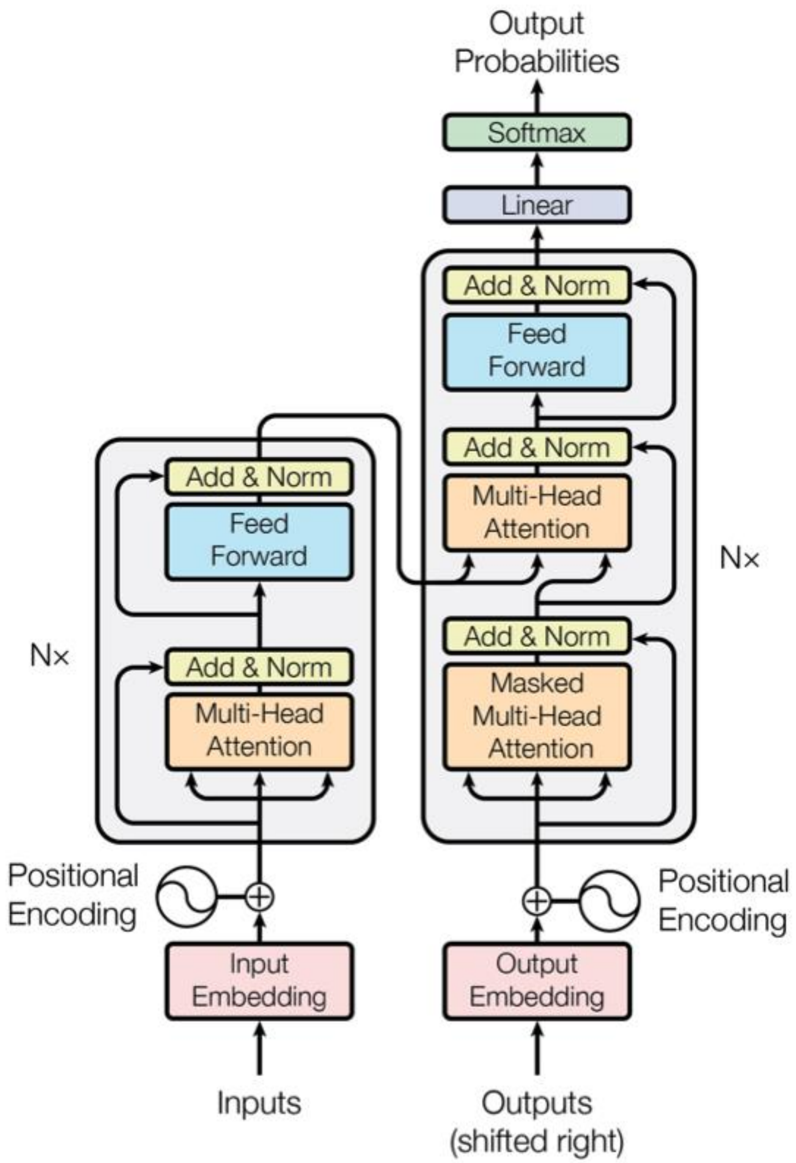
- Arm NN open-source collaboration enables optimal third-party implementations
- Deployed in multiple production devices (>250Mu)

Transformer & Large Language Model (LLM)

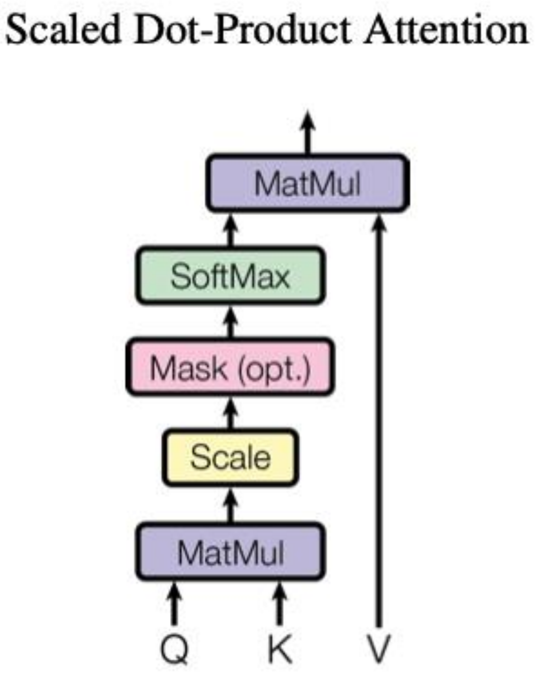


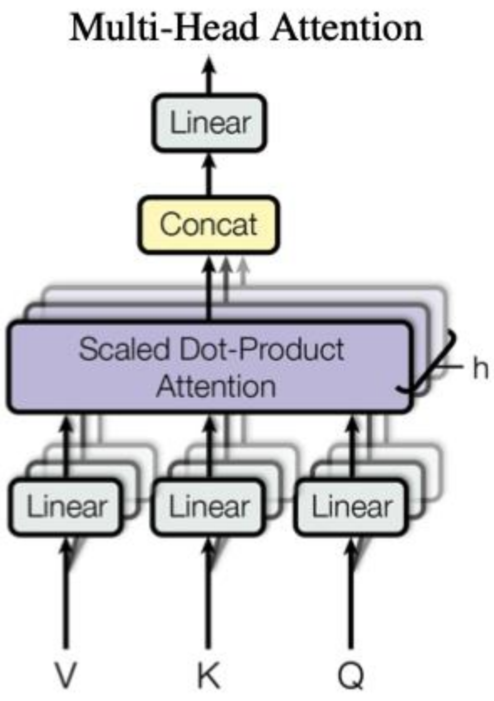
- LLM is a regressive model that generates text.
- Thousands of attention modules construct the entire GPT models.

Additional: Attention in Transformer



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).



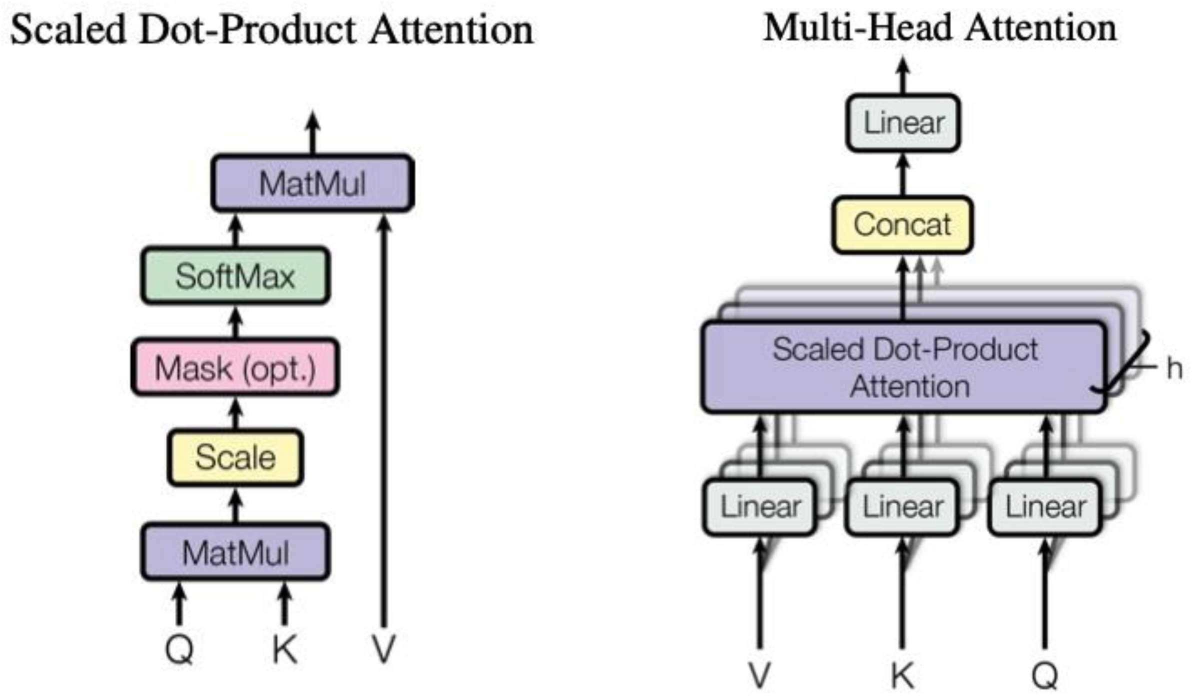


Additional: Attention in Transformer

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

$$Q = xW_Q, K = xW_K, V = xW_V$$

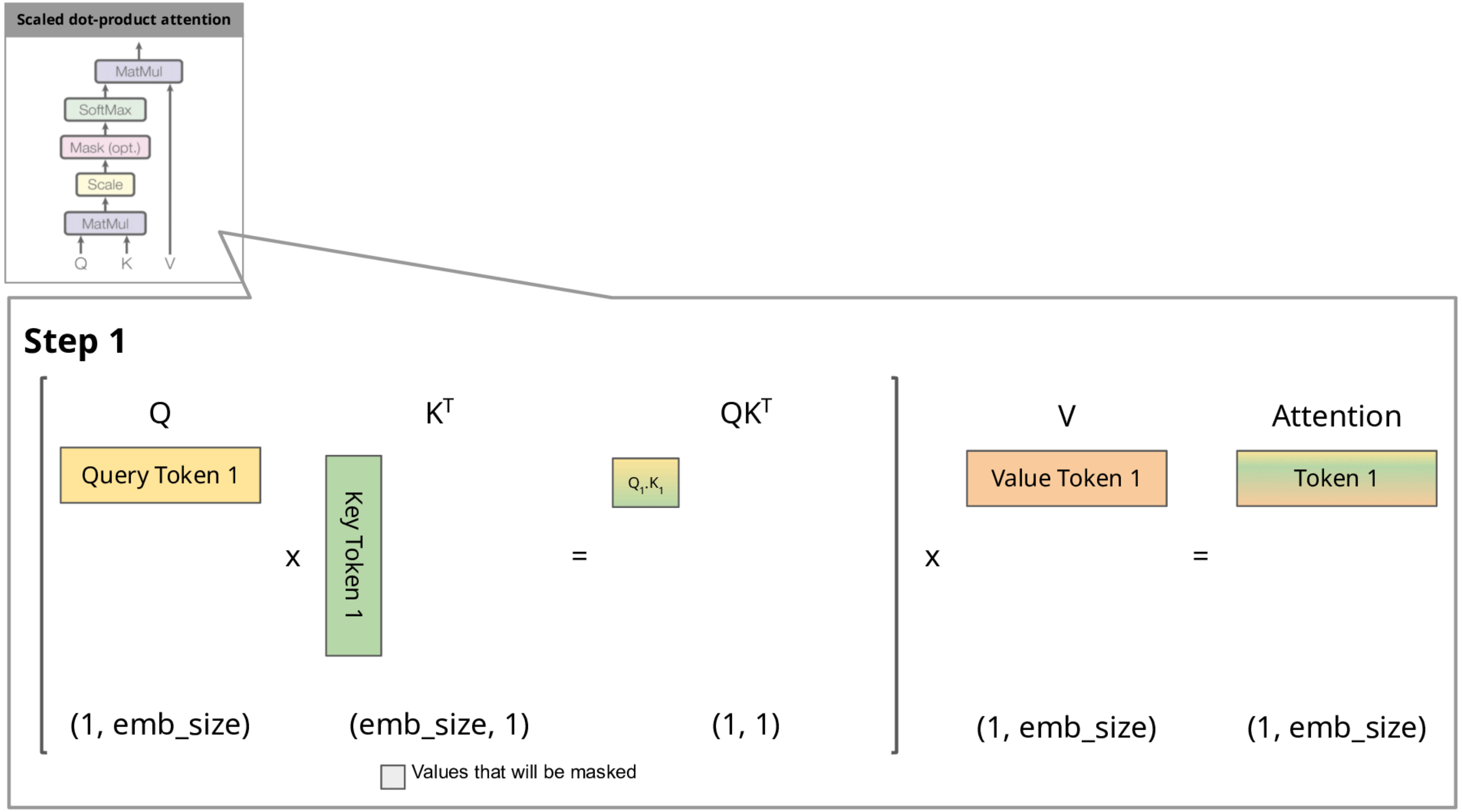
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://zhuanlan.zhihu.com/p/624740065

Transformer & Large Language Model (LLM)

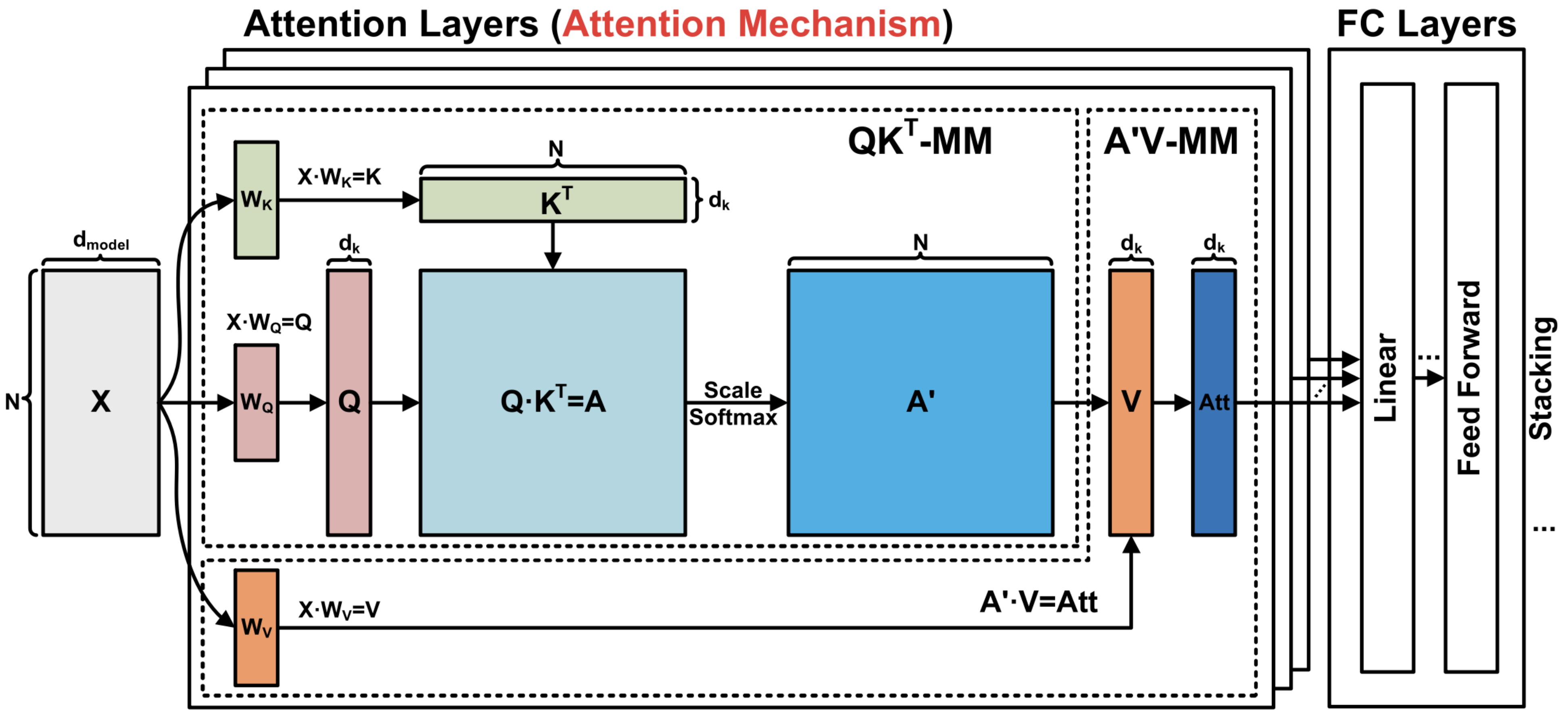
北京大學 PEKING UNIVERSITY



Zoom-in! (simplified without Scale and Softmax)

https://medium.com/@joaolages/kv-caching-explained-276520203249

Accelerator Computing Task



Courtesy: TranCim Paper

Check Related Transformer Code

- https://github.com/tinygrad/tinygrad
- tinygrad/examples/transformer.py
- tinygrad/examples/llama.py