

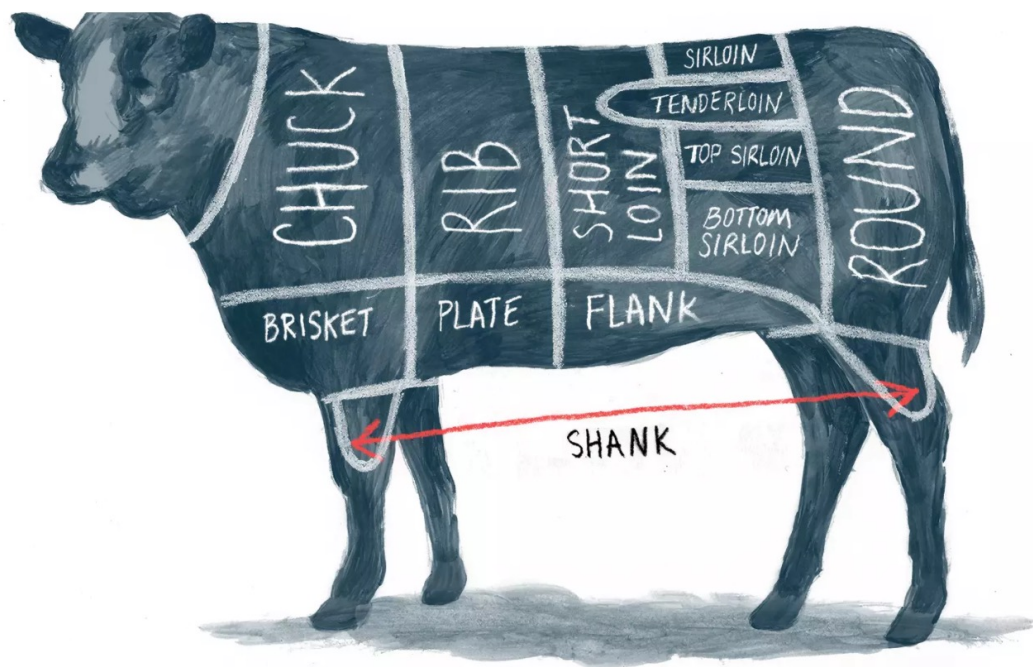
04835370

人工智能芯片设计导论

Fall 2023

Deep Learning Basics

燕博南



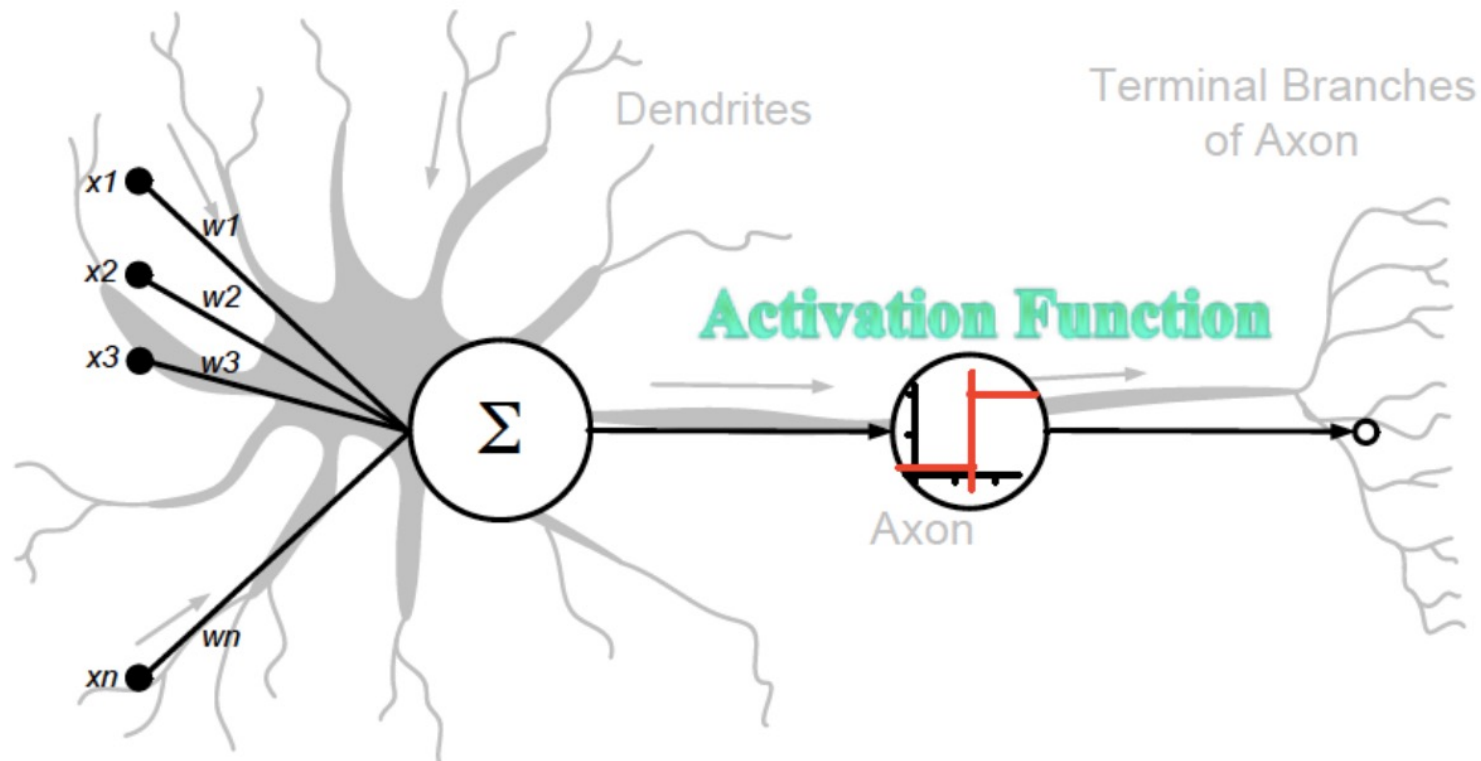
- Fully
- Convolution
- 其它算子
- Residual Network
- Depthwise Separable
- ...

以“用”的角度去学

Neural Networks

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

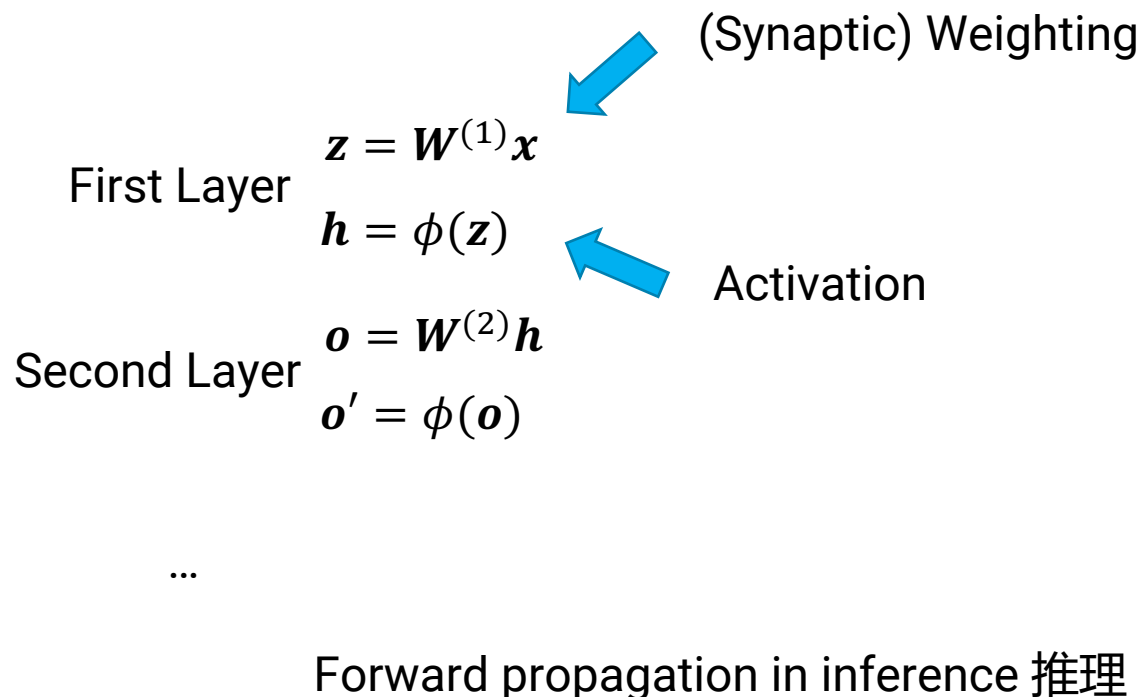
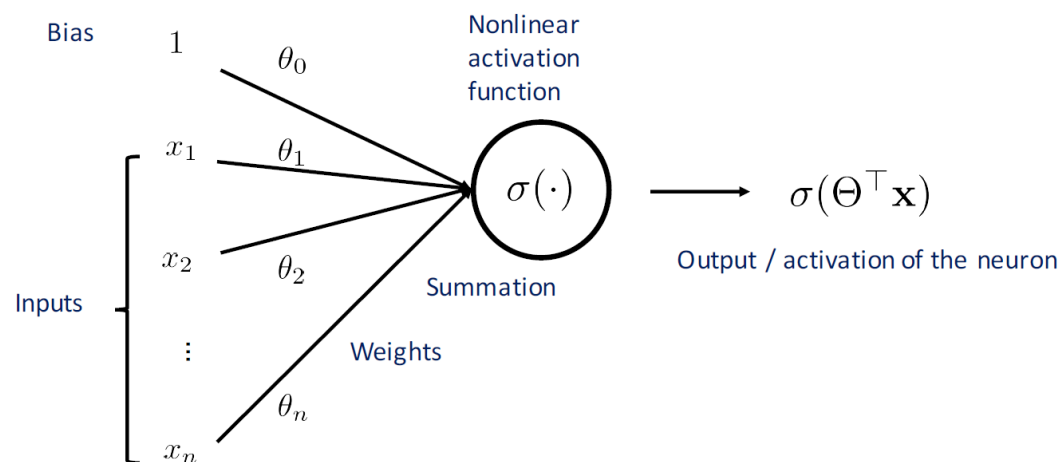
- Neurons **receive** electric signals at the dendrites and **send** them to the axon
- Dendrites can perform complex **non-linear** computations
- Synapses are not a single weight but a **complex** non-linear dynamical system



Artificial Neural Networks to Deep Neural Networks

Artificial neural networks

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



主要的计算量： 乘加 (multiply-accumulate, MAC)

Deep Learning

Forward propagation:

(Synaptic) Weighting

First Layer
 $z = W^{(1)}x$

$h = \phi(z)$
Activation

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

Evaluate the network:

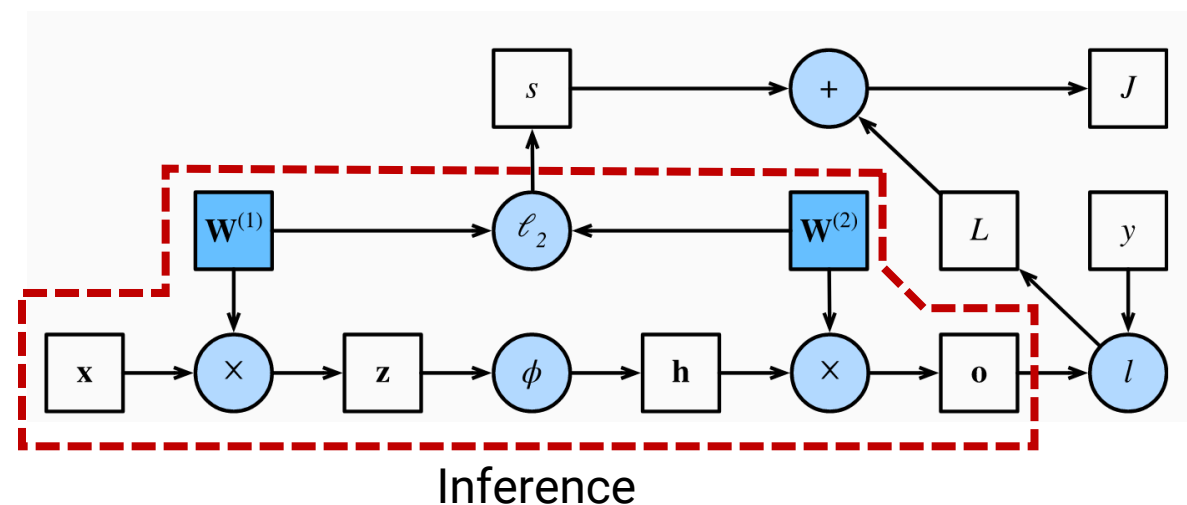
$L = l(o, y)$
Loss function

Regularization

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

Training: Minimize $(L+s)$

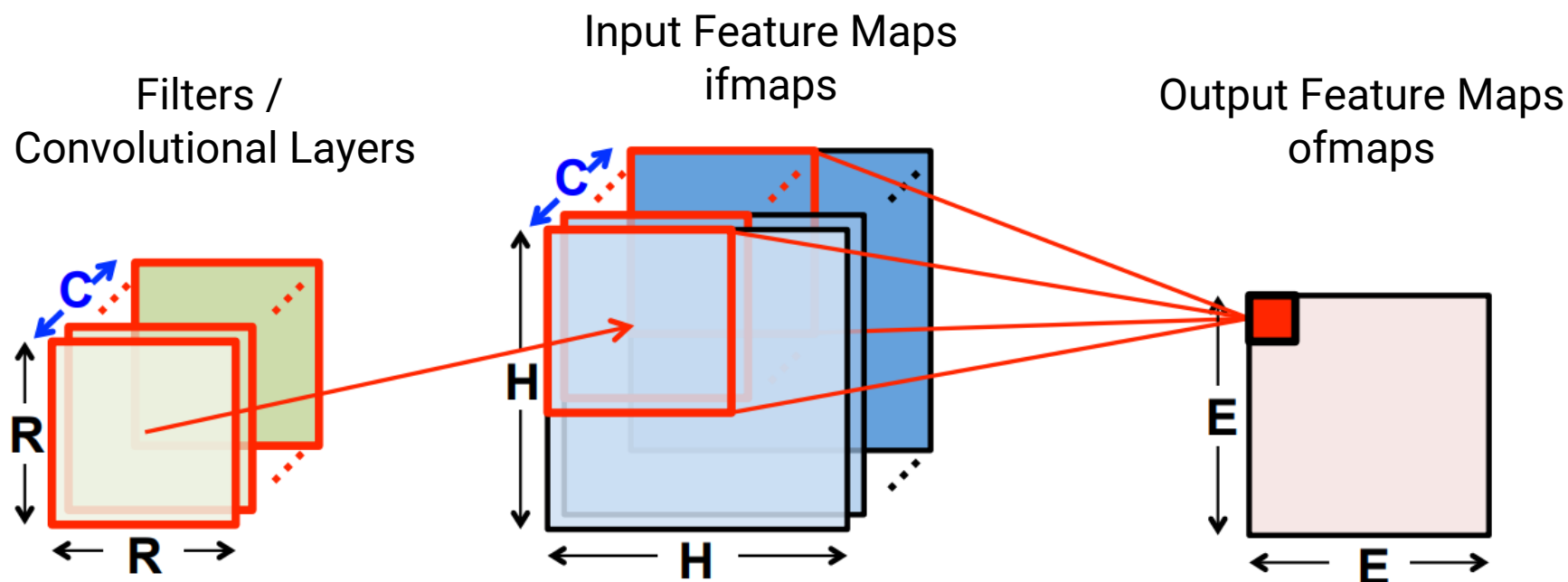
Computational Graph



Convolutional Layer

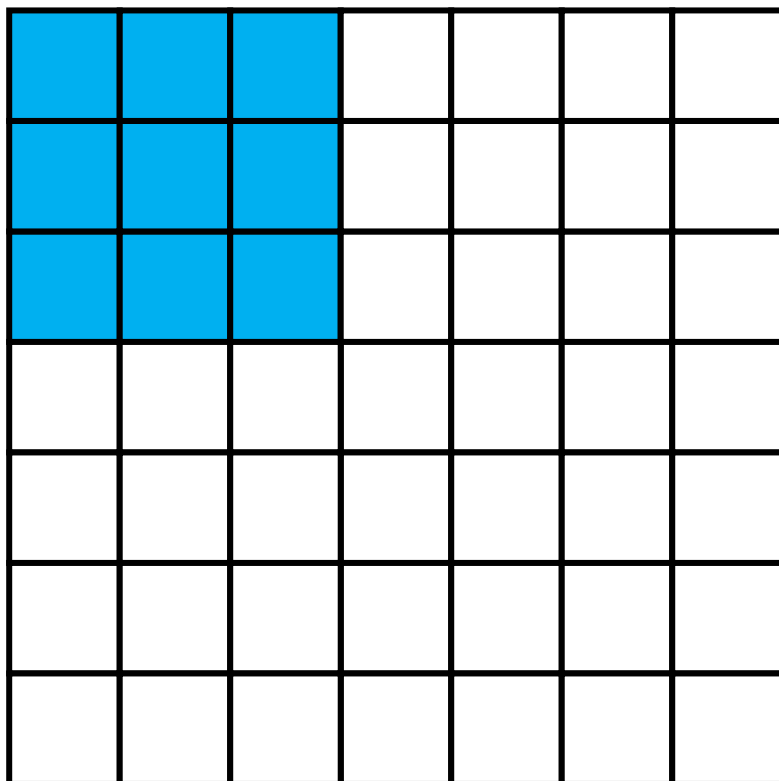
$$\mathbf{O}[z][u][x][y] = \mathbf{B}[u] + \sum_{k=0}^{C-1} \sum_{i=0}^{R-1} \sum_{j=0}^{R-1} \mathbf{I}[z][k][Ux+i][Uy+j] \times \mathbf{W}[u][k][i][j],$$

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



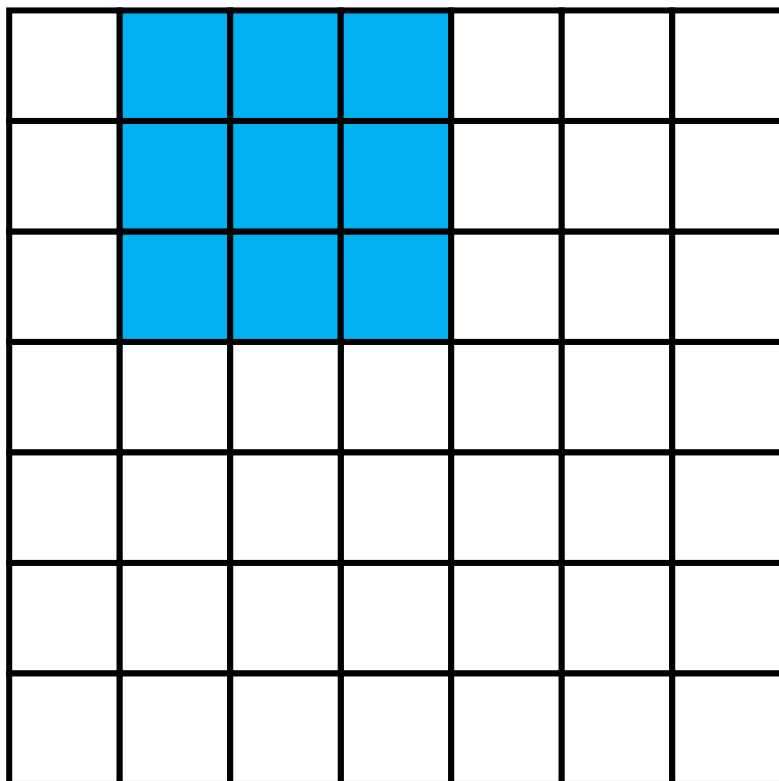
N : batch size
 M : # of ofmap channels
 C : # of ifmap filter channels
 H : ifmap height/width
 R : filter plan height/width
 E : ofmap height/width

Convolutional Layer - Example



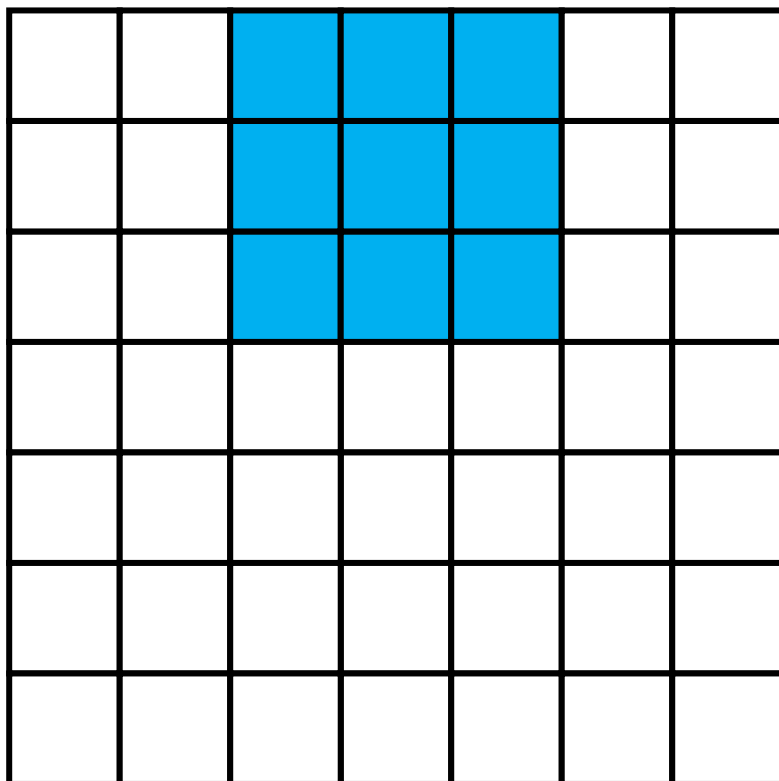
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Convolutional Layer - Example



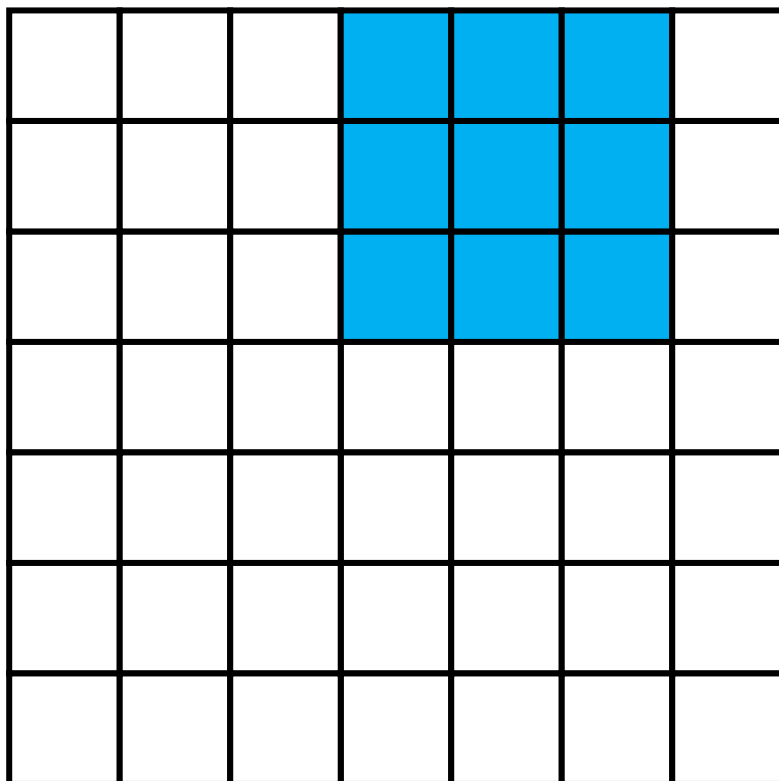
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Convolutional Layer - Example



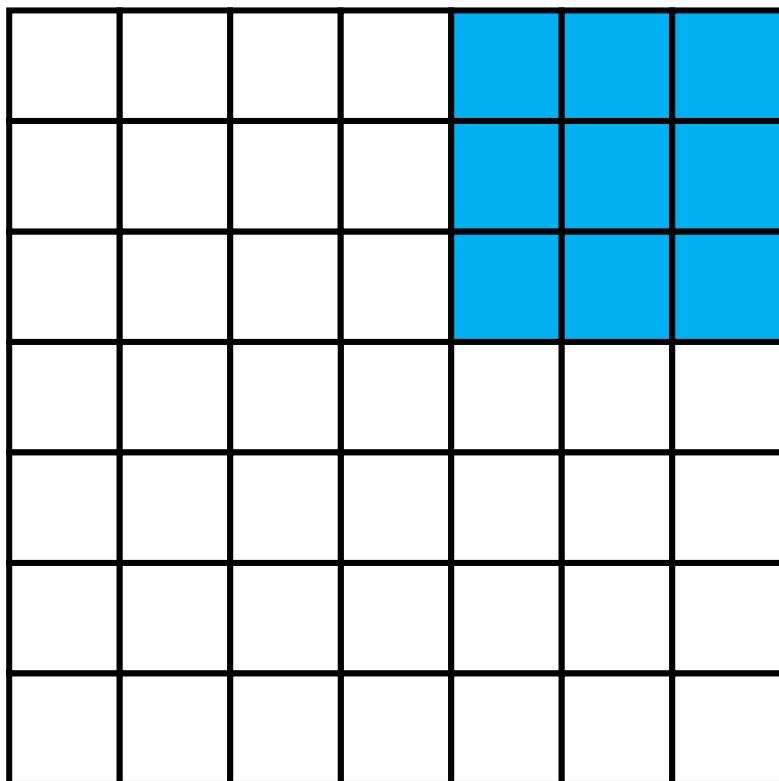
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

Convolutional Layer - Example



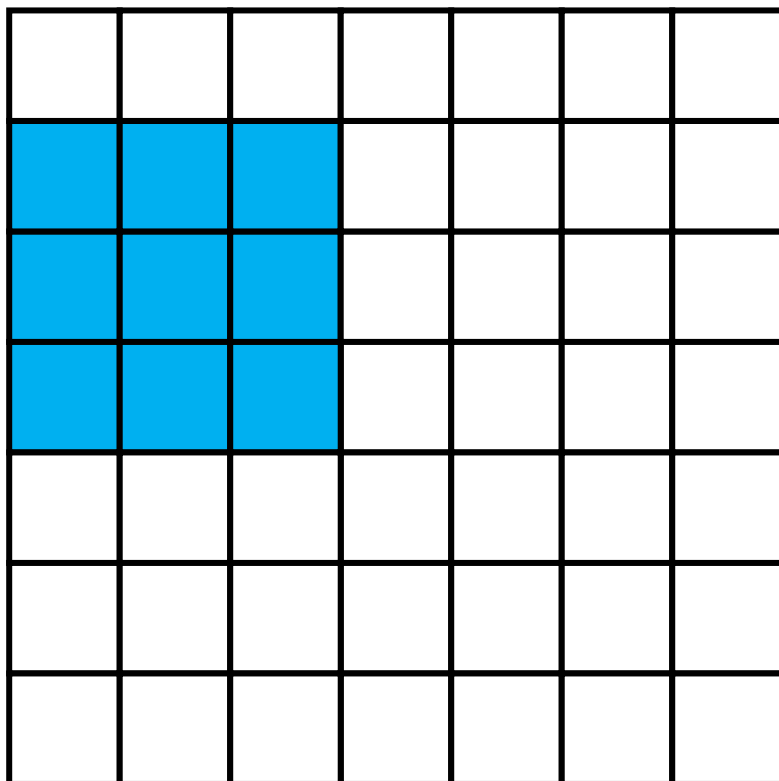
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

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

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**

Convolutional Layer - Example

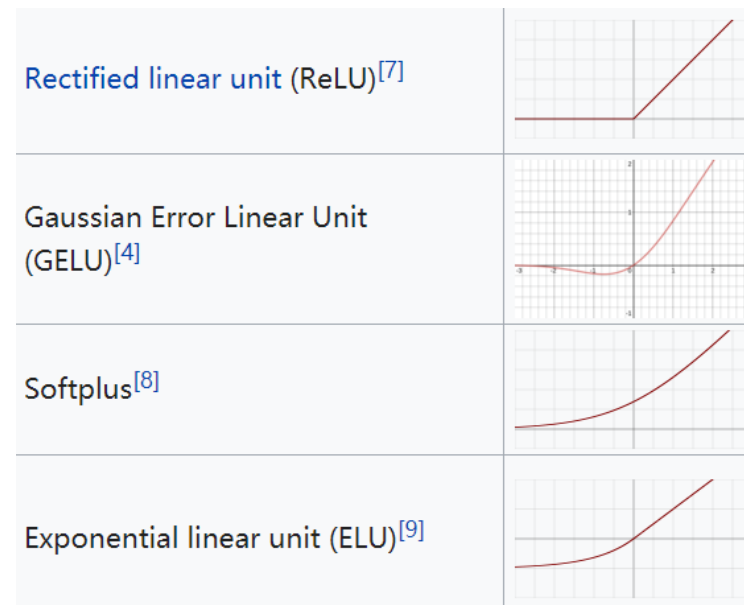
0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

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

其它的算子-1

- Activation

- Relu
- Tanh
- Sigmoid
- Softmax
- ...

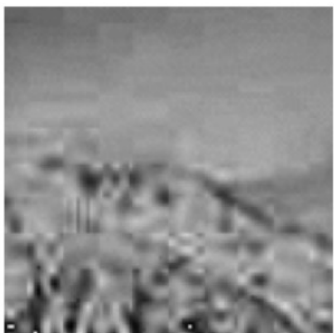


硬件上如何最简单地实现？

其它的算子-2

- Pooling

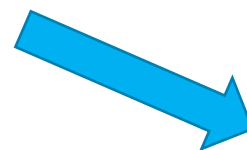
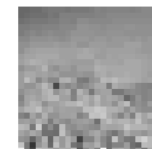
- Max pooling
- Average pooling
- Stochastic pooling
- ...



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)
 - ...

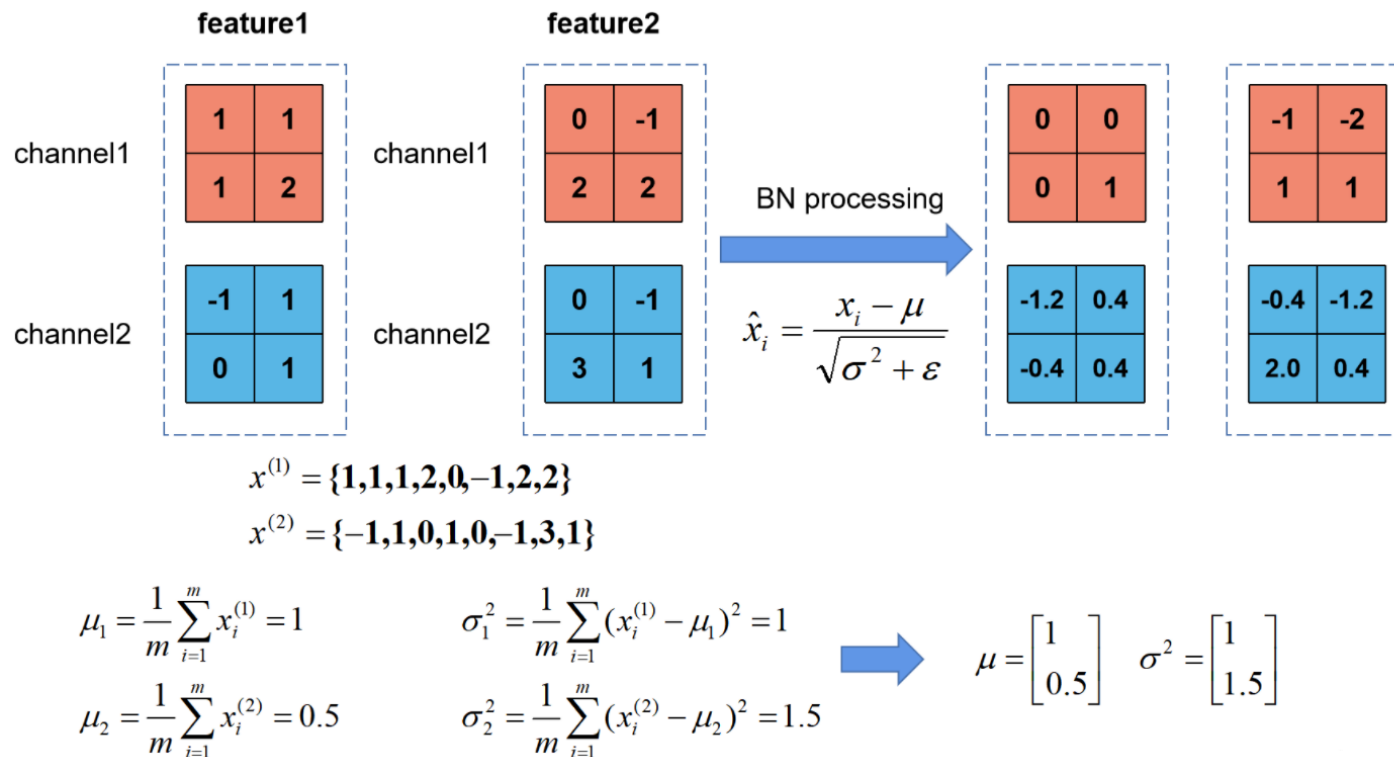
Example of BN:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;
 Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

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

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

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

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


Some Famous and Basic NN

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

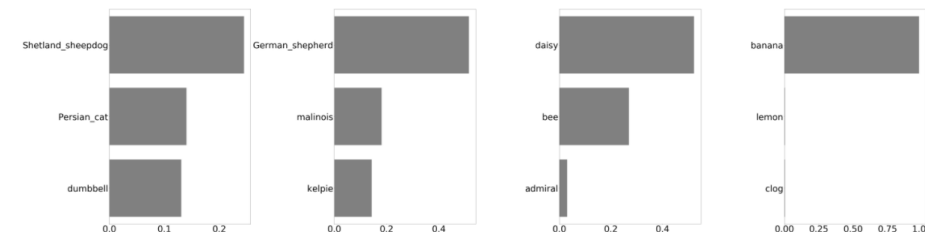
VGG16

Network Architecture

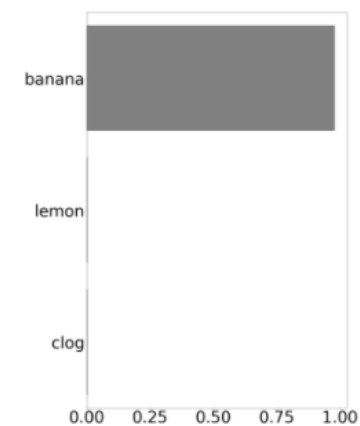
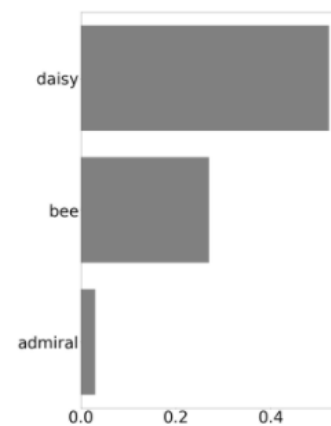
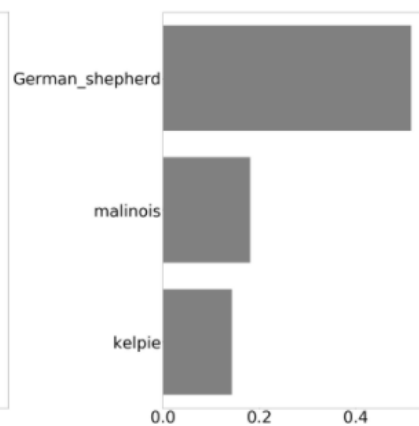
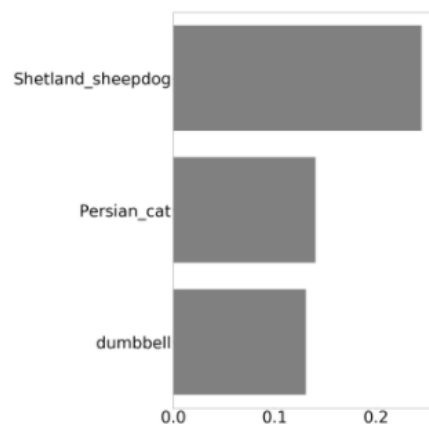
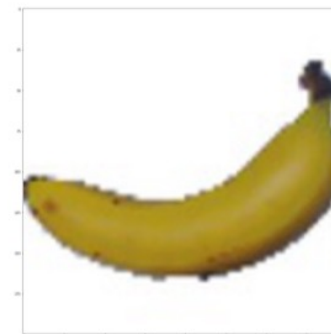
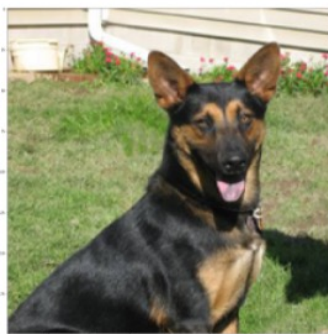
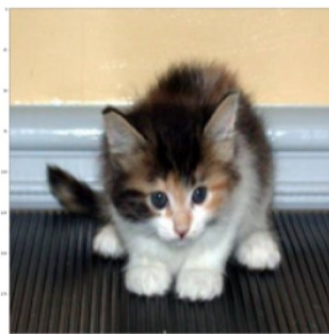
```

1  _input = Input((224,224,1))
2
3  conv1 = Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(_input)
4  conv2 = Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(conv1)
5  pool1 = MaxPooling2D((2, 2))(conv2)
6
7  conv3 = Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(pool1)
8  conv4 = Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(conv3)
9  pool2 = MaxPooling2D((2, 2))(conv4)
10
11 conv5 = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(pool2)
12 conv6 = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(conv5)
13 conv7 = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(conv6)
14 pool3 = MaxPooling2D((2, 2))(conv7)
15
16 conv8 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(pool3)
17 conv9 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv8)
18 conv10 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv9)
19 pool4 = MaxPooling2D((2, 2))(conv10)
20
21 conv11 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(pool4)
22 conv12 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv11)
23 conv13 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv12)
24 pool5 = MaxPooling2D((2, 2))(conv13)
25
26 flat = Flatten()(pool5)
27 dense1 = Dense(4096, activation="relu")(flat)
28 dense2 = Dense(4096, activation="relu")(dense1)
29 output = Dense(1000, activation="softmax")(dense2)

```



Performance

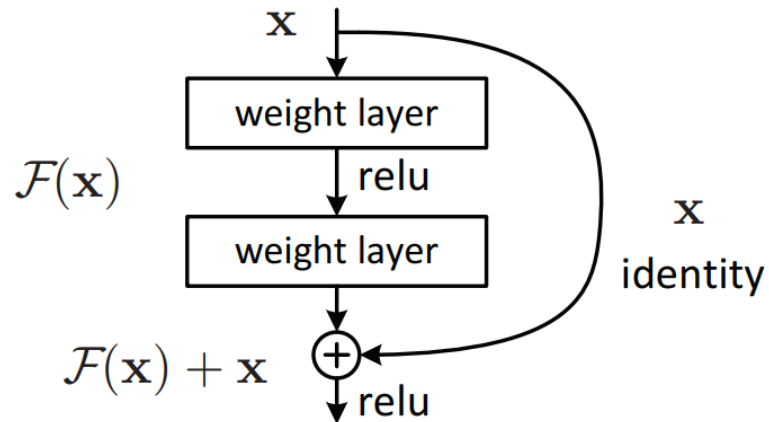


Residual Network

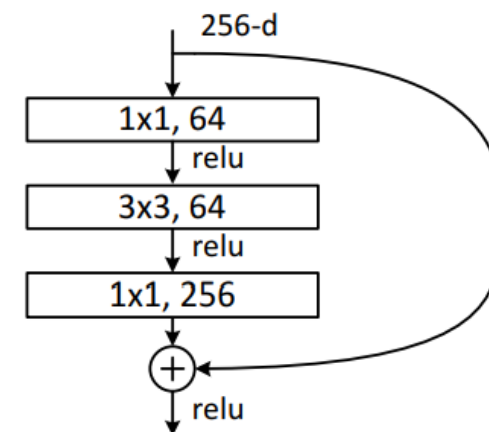
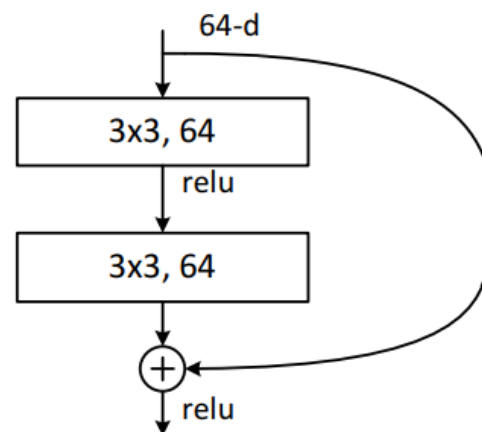
When DNN goes deeper:

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

Examples:



注意BN的位置!

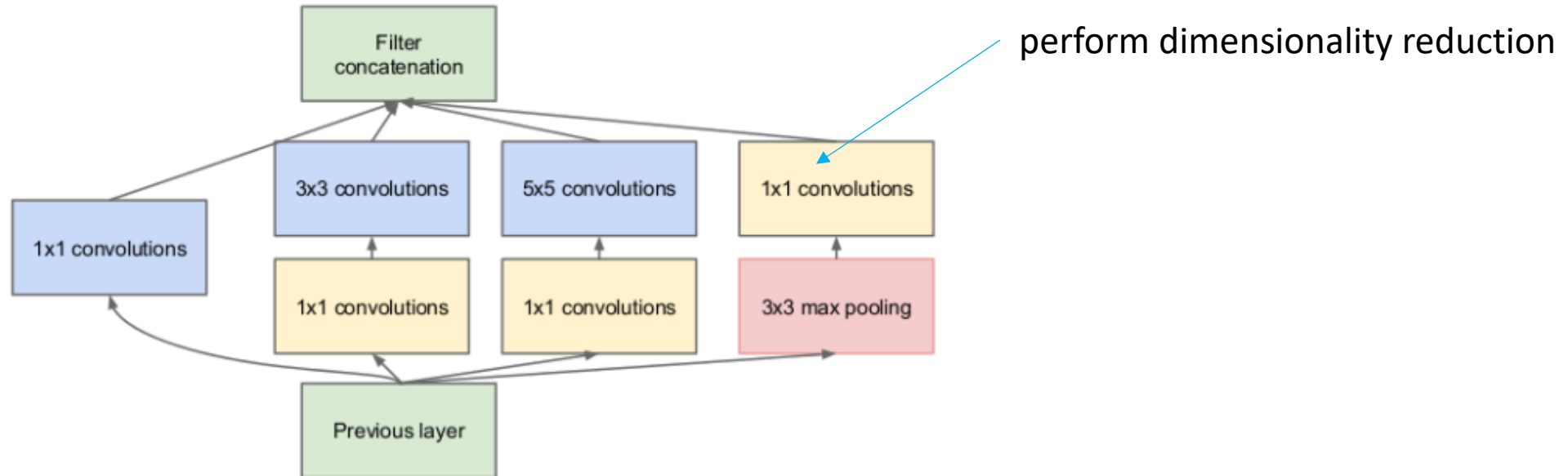


Bottleneck block

InceptionNets

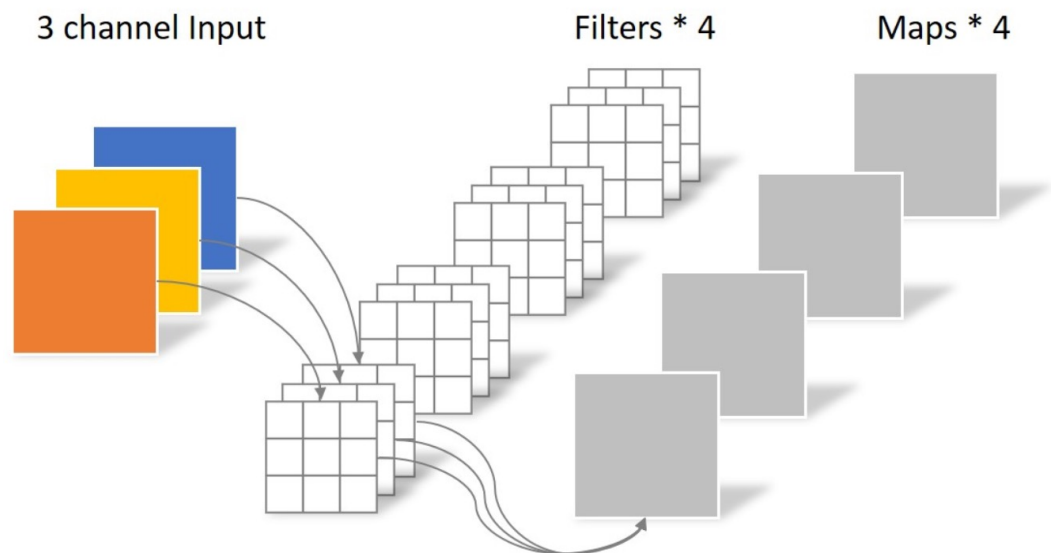
- Also known as GoogleNet consists of total 22 layers and was the winning model of 2014 image net challenge.

Inception Module:



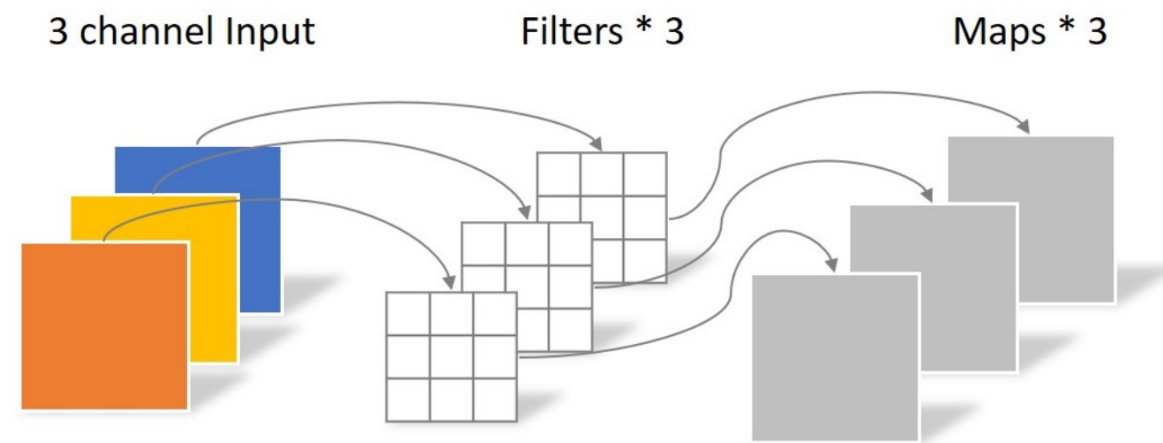
Depthwise Separable Convolution

Normal Convolution



Parameter Number: $4 \times 3 \times 3 \times 3 = 108$

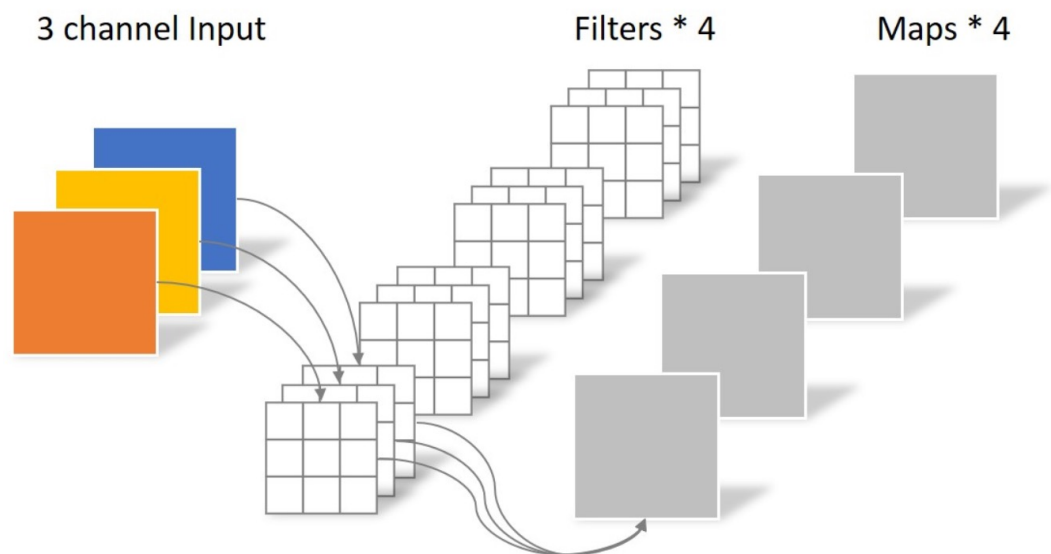
Depthwise Convolution



Parameter Number: $3 \times 3 \times 3 = 27$

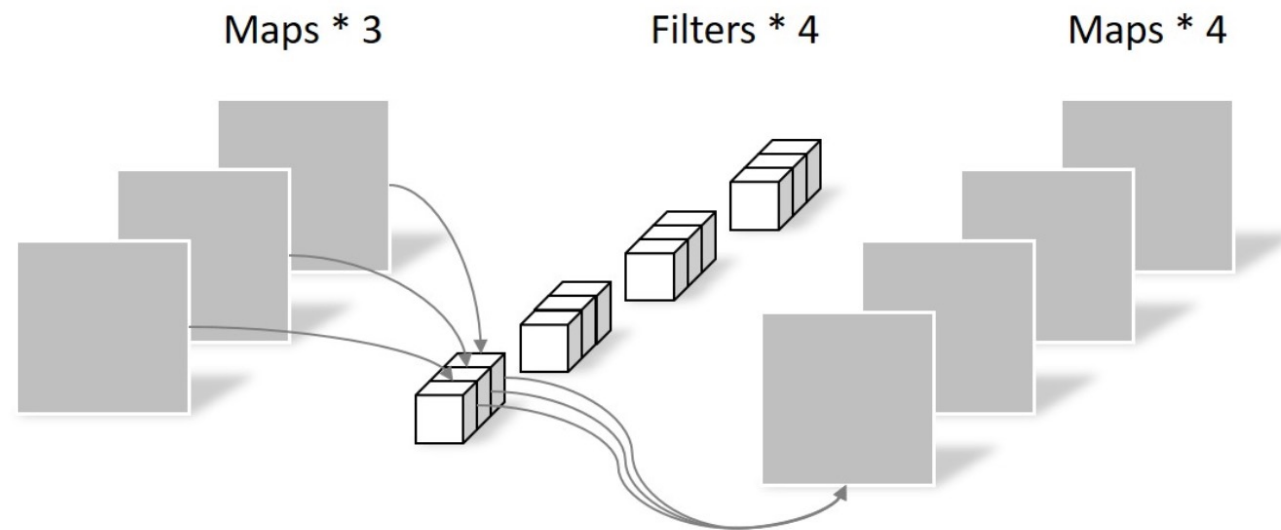
Depthwise Separable Convolution

Normal Convolution



Parameter Number: $4 \times 3 \times 3 \times 3 = 108$

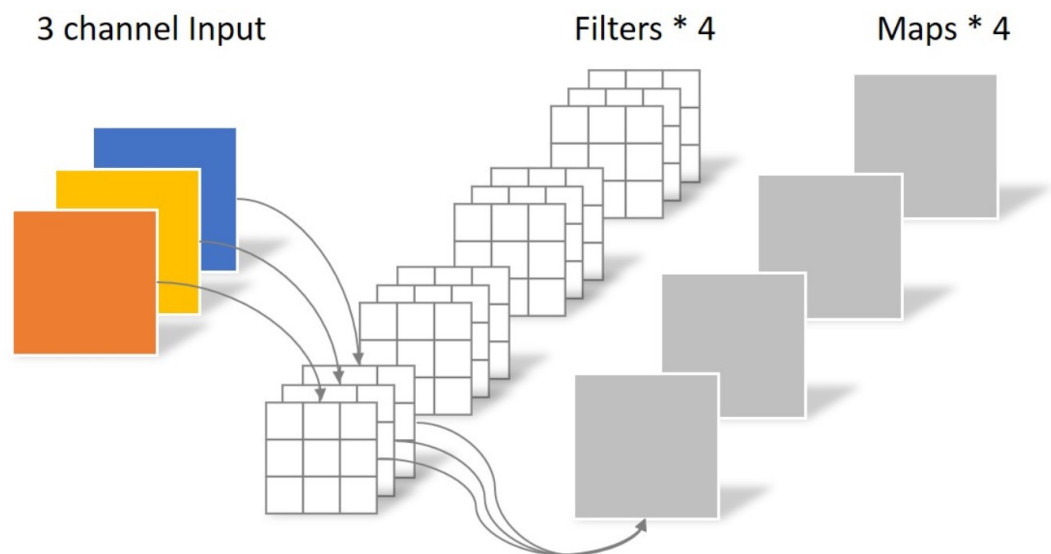
Pointwise Convolution



Parameter Number: $4 \times 3 \times 1 \times 1 = 12$

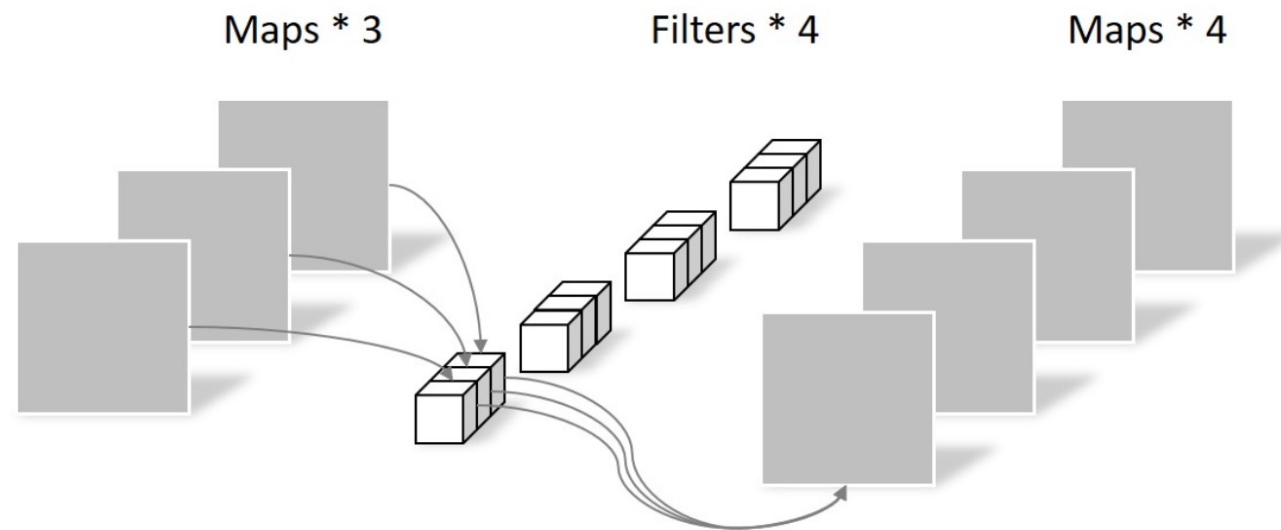
Depthwise Separable Convolution

Normal Convolution



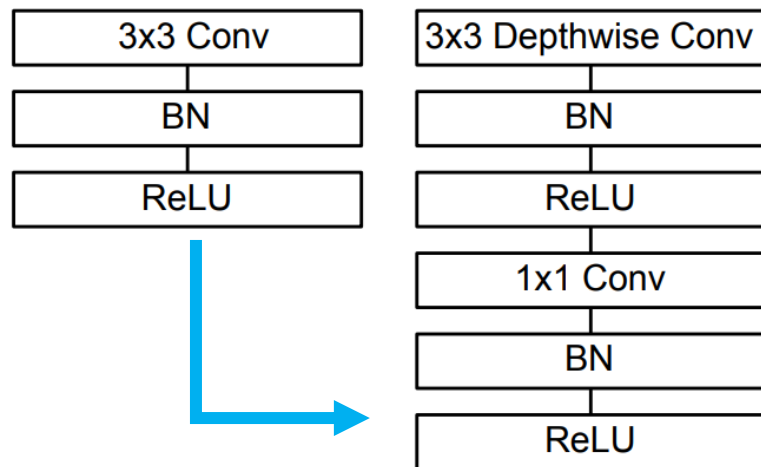
Parameter Number: $4 \times 3 \times 3 \times 3 = 108$

Pointwise Convolution



Parameter Number: $4 \times 3 \times 1 \times 1 = 12$

MobileNet



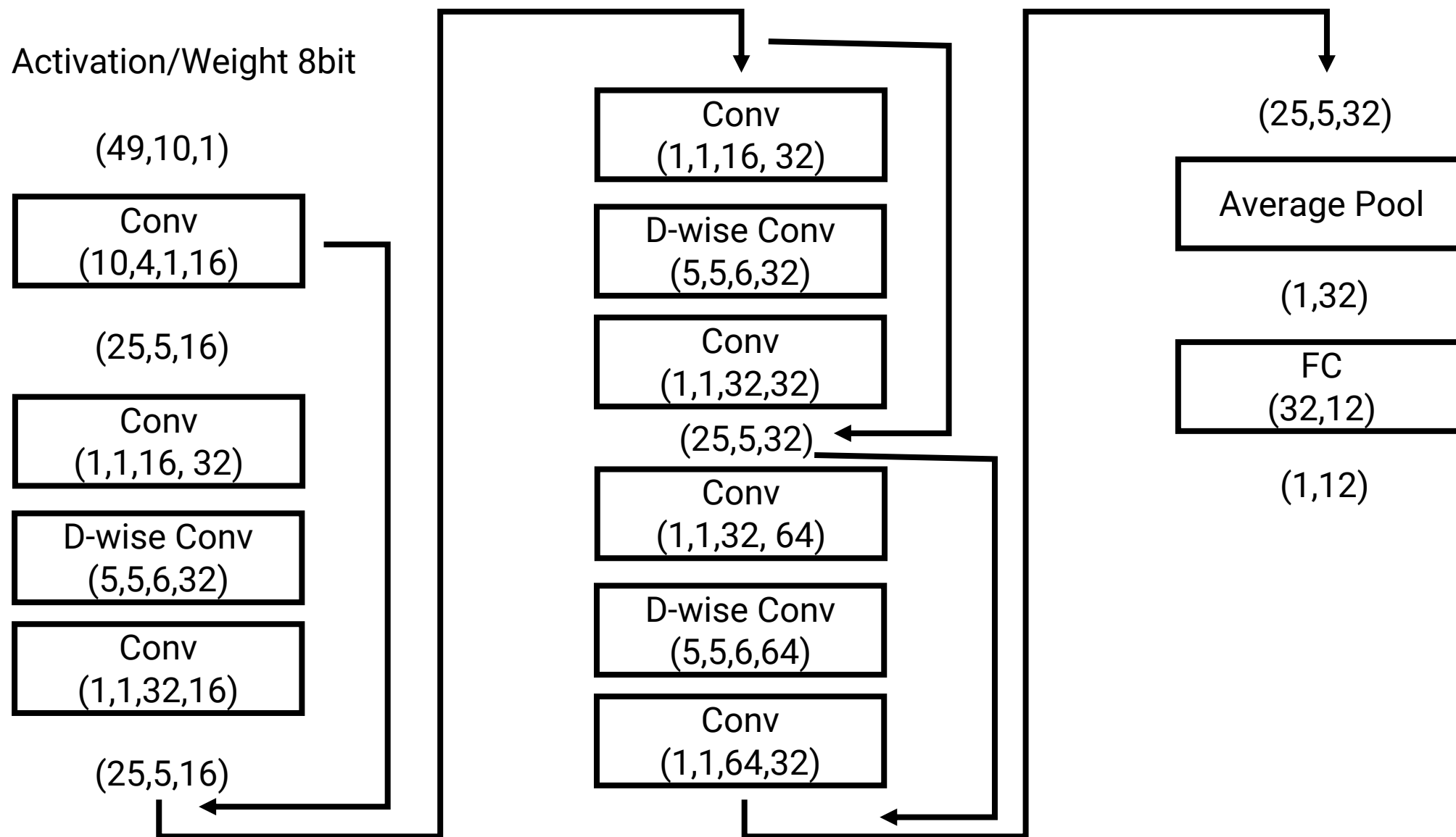
Parameter Number: $4 \times 3 \times 3 \times 3 = 108$

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

减少计算量!

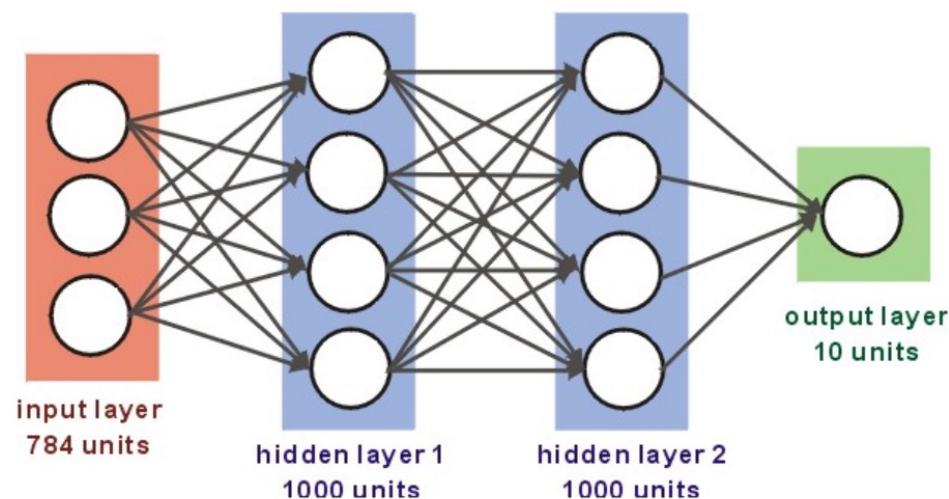
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32 \text{ dw}$	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64 \text{ dw}$	$112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

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



```
1  int layerSizes[10] = {0,0,0,0,0,0,784,1000,1000,10};
2  float* layers[10] = {0};
3  float* errors[10] = {0};
4  float* weights[10] = {0};
5  // INITIALIZATION
6  void initNet(){
7      // ALLOCATE MEMORY
8      layers[0] = (float*)malloc((layerSizes[0]+1) * sizeof(float));
9      errors[0] = (float*)malloc(layerSizes[0] * sizeof(float));
10     for (i=1;i<10;i++){
11         layers[i] = (float*)malloc((layerSizes[i]+1) * sizeof(float));
12         errors[i] = (float*)malloc(layerSizes[i] * sizeof(float));
13         weights[i] = (float*)malloc(layerSizes[i] * (layerSizes[i-1]+1) * sizeof(float));
14     }
15     // RANDOMIZE WEIGHTS AND BIAS
16     float scale;
17     for (i=0;i<10;i++) layers[i][layerSizes[i]]=1.0;
18     for (j=1;j<10;j++){
19         scale = 2.0 * sqrt(6.0/(layerSizes[j] + layerSizes[j-1]));
20         if (j!=9 && activation==1) scale = scale * 1.41; // RELU
21         else if (j!=9) scale = scale * 4.0; // TANH
22         for (i=0;i<layerSizes[j] * (layerSizes[j-1]+1);i++)
23             weights[j][i] = scale * ( (float)rand()/RAND_MAX - 0.5 );
24         for (i=0;i<layerSizes[j];i++) // BIASES
25             weights[j][(layerSizes[j-1]+1)*(i+1)-1] = 0.0;
26     }
27 }
```

Forward Pass (Inference)

```
1  int activation = 1; //ReLU
2  // FORWARD PROPAGATION
3  int forwardProp(int x){
4      int i,j,k,imax;
5      float sum, esum, max;
6      // INPUT LAYER - RECEIVES 28X28 IMAGES
7      for (i=0;i<784;i++) layers[10-numLayers][i] = trainImages[x][i];
8      // HIDDEN LAYERS - RELU ACTIVATION
9      for (k=11-numLayers;k<9;k++){
10         for (i=0;i<layerSizes[k];i++){
11             sum = 0.0;
12             for (j=0;j<layerSizes[k-1]+1;j++){
13                 sum += layers[k-1][j]*weights[k][i*(layerSizes[k-1]+1)+j];
14                 if (activation==1) layers[k][i] = ReLU(sum);
15                 else if (activation==2) layers[k][i] = TanH(sum);
16             }
```

```
17         // OUTPUT LAYER - SOFTMAX ACTIVATION
18         esum = 0.0;
19         for (i=0;i<layerSizes[9];i++){
20             sum = 0.0;
21             for (j=0;j<layerSizes[8]+1;j++){
22                 sum += layers[8][j]*weights[9][i*(layerSizes[8]+1)+j];
23                 if (sum>30) return -1; //GRADIENT EXPLODED
24                 layers[9][i] = exp(sum);
25                 esum += layers[9][i];
26             }
27         // SOFTMAX FUNCTION
28         max = layers[9][0]; imax=0;
29         for (i=0;i<layerSizes[9];i++){
30             if (layers[9][i]>max){
31                 max = layers[9][i];
32                 imax = i;
33             }
34             layers[9][i] = layers[9][i] / esum;
35         }
36         return imax;
37     }
```


Here The Accelerator Comes!

```
1  int activation = 1; //ReLU
2  // FORWARD PROPAGATION
3  int forwardProp(int x){
4      int i,j,k,imax;
5      float sum, esum, max;
6      // INPUT LAYER - RECEIVES 28X28 IMAGES
7      for (i=0;i<784;i++) layers[10-numLayers][i] = trainImages[x][i];
8      // HIDDEN LAYERS - RELU ACTIVATION
9      for (k=11-numLayers;k<9;k++){
10         for (i=0;i<layerSizes[k];i++){
11             sum = 0.0;
12             for (j=0;j<layerSizes[k-1]+1;j++){
13                 sum += layers[k-1][j]*weights[k][i*(layerSizes[k-1]+1)+j];
14             }
15             if (activation==1) layers[k][i] = ReLU(sum);
16             else if (activation==2) layers[k][i] = TanH(sum);
17         }
18     }
```

```
17 // OUTPUT LAYER - SOFTMAX ACTIVATION
18 esum = 0.0;
19 for (i=0;i<layerSizes[9];i++){
20     sum = 0.0;
21     for (j=0;j<layerSizes[8]+1;j++){
22         sum += layers[8][j]*weights[9][i*(layerSizes[8]+1)+j];
23     }
24     if (sum>30) return -1; //GRADIENT EXPLODED
25     layers[9][i] = exp(sum);
26     esum += layers[9][i];
27 }
28 // SOFTMAX FUNCTION
29 max = layers[9][0]; imax=0;
30 for (i=0;i<layerSizes[9];i++){
31     if (layers[9][i]>max){
32         max = layers[9][i];
33         imax = i;
34     }
35     layers[9][i] = layers[9][i] / esum;
36 }
37 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

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

C Intrinsic: XXX.s

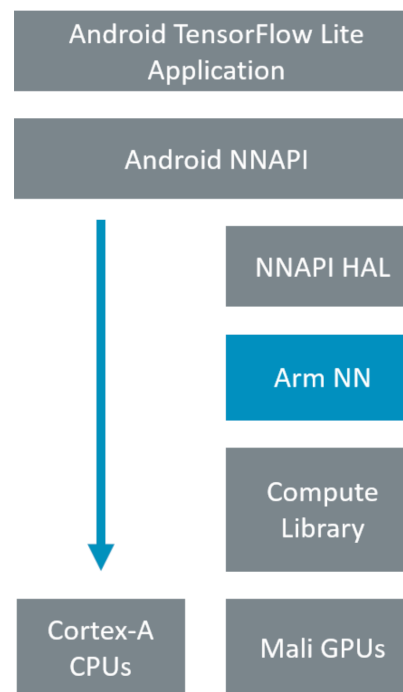
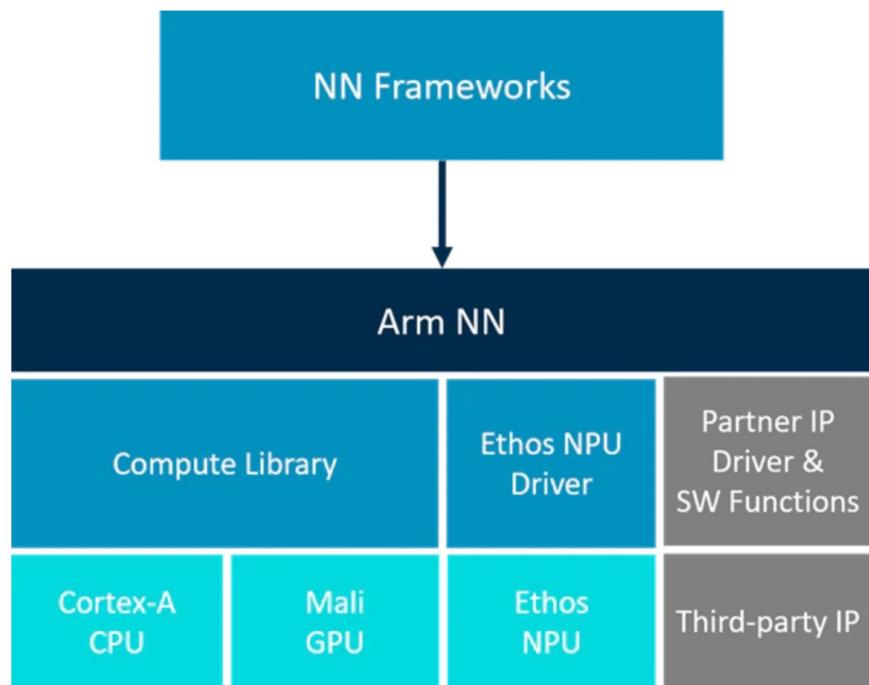
```
.section text

.global AcceleratorVMM
.type AcceleratorVMM, @function

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

Is this real?

- Yes!



Arm NN for Android

Also available is Arm NN for NNAPI, Google's interface for accelerating neural networks on Android devices, made available in Android O. By default, NNAPI runs neural network workload on the device's CPU cores, but also provides a Hardware Abstraction Layer (HAL) that can target other processor types, such as GPUs. Arm NN for Android NNAPI provides this HAL for Mali GPUs. A further release adds support for Arm Ethos-N NPUs.

Arm support for Android NNAPI gives >4x performance boost.

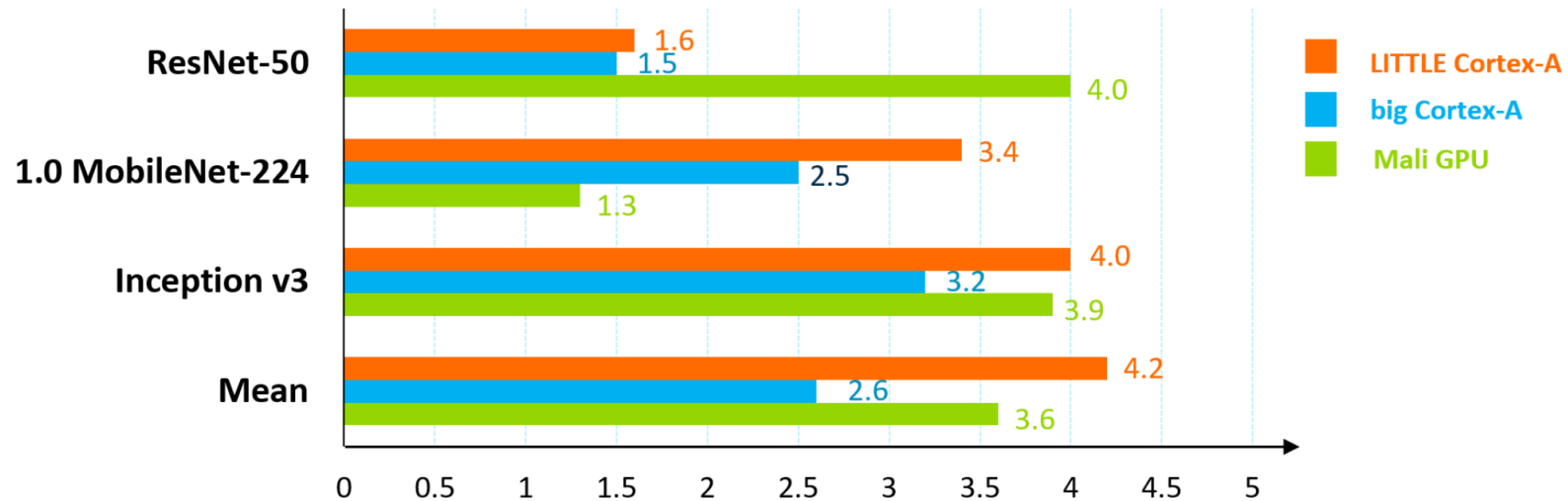
[Learn more](#)

Download Arm NN for Android sources.

[Download here](#)

Real products

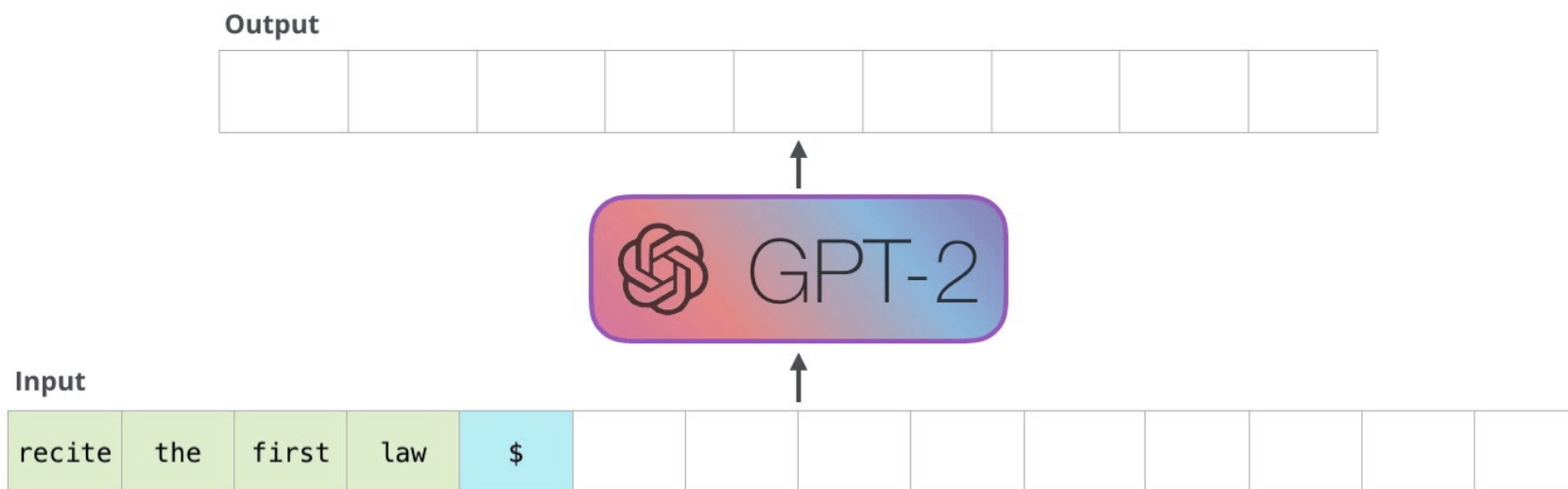
Arm NN performance relative to other NN frameworks



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)

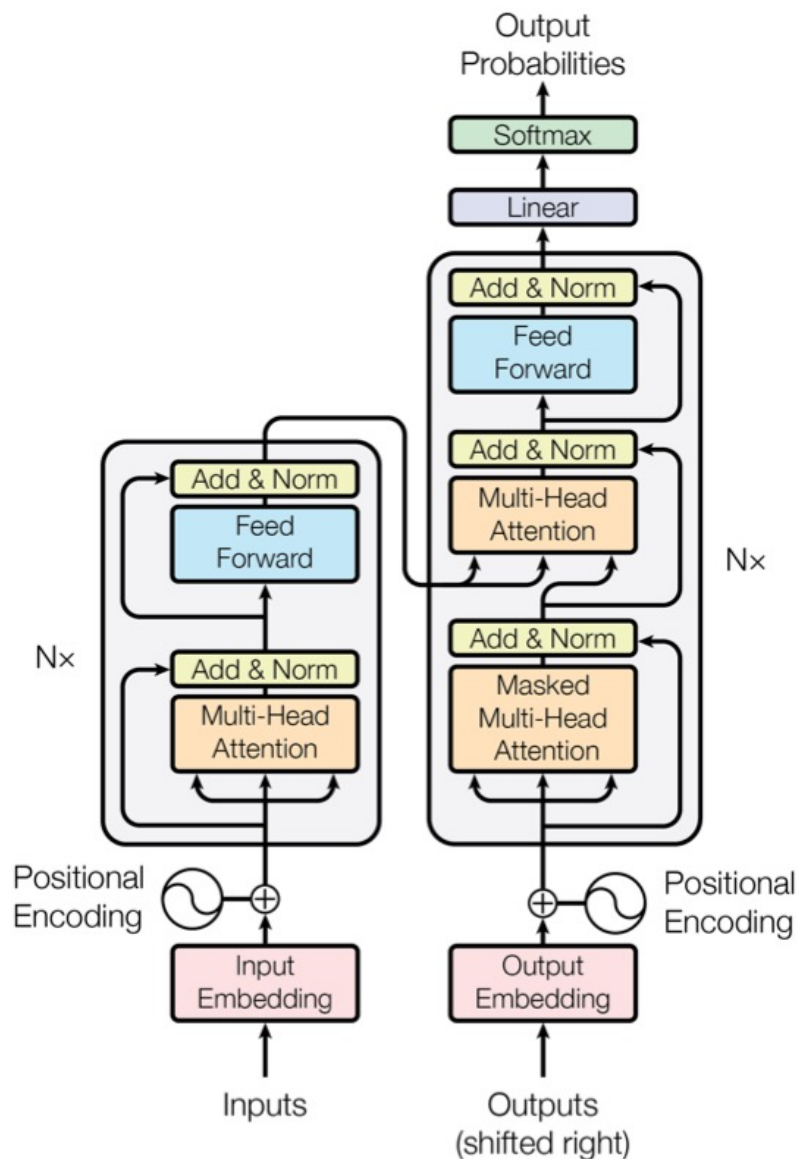
Transformer & Large Language Model (LLM)



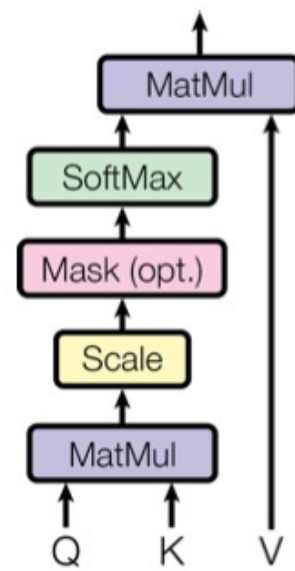
- LLM is a regressive model that generates text.
- Thousands of attention modules construct the entire GPT models.

Additional: Attention in Transformer

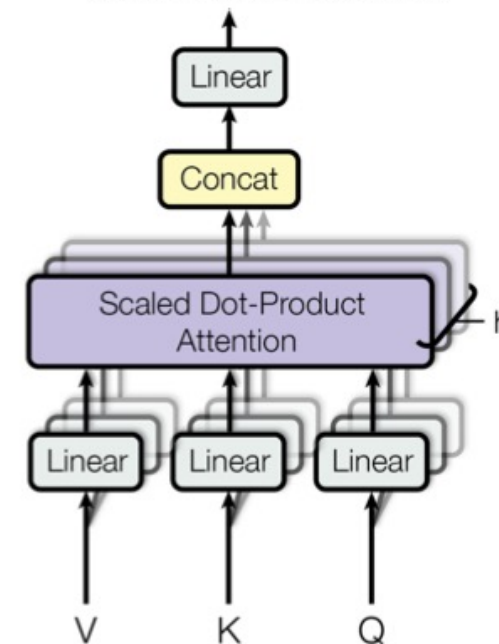
Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).



Scaled Dot-Product Attention



Multi-Head Attention



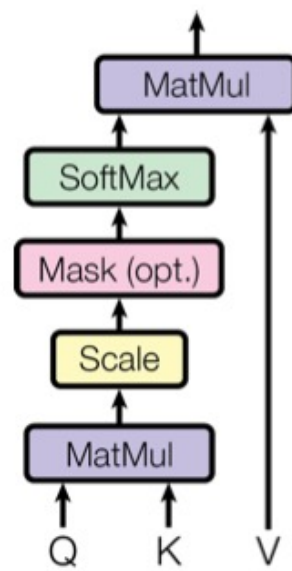
Additional: Attention in Transformer

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

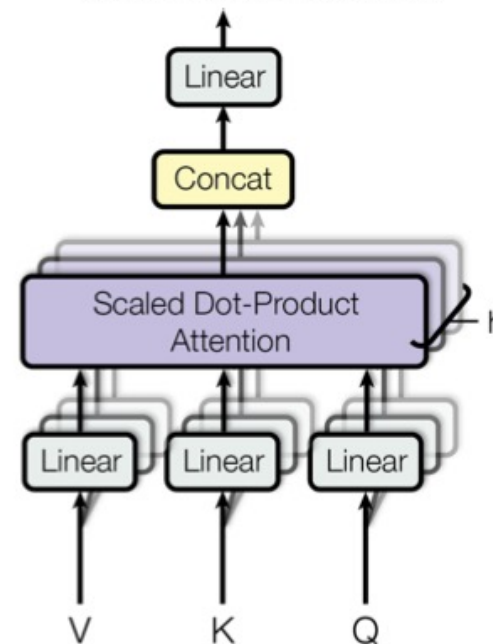
$$Q = xW_Q, K = xW_K, V = xW_V$$

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

Scaled Dot-Product Attention

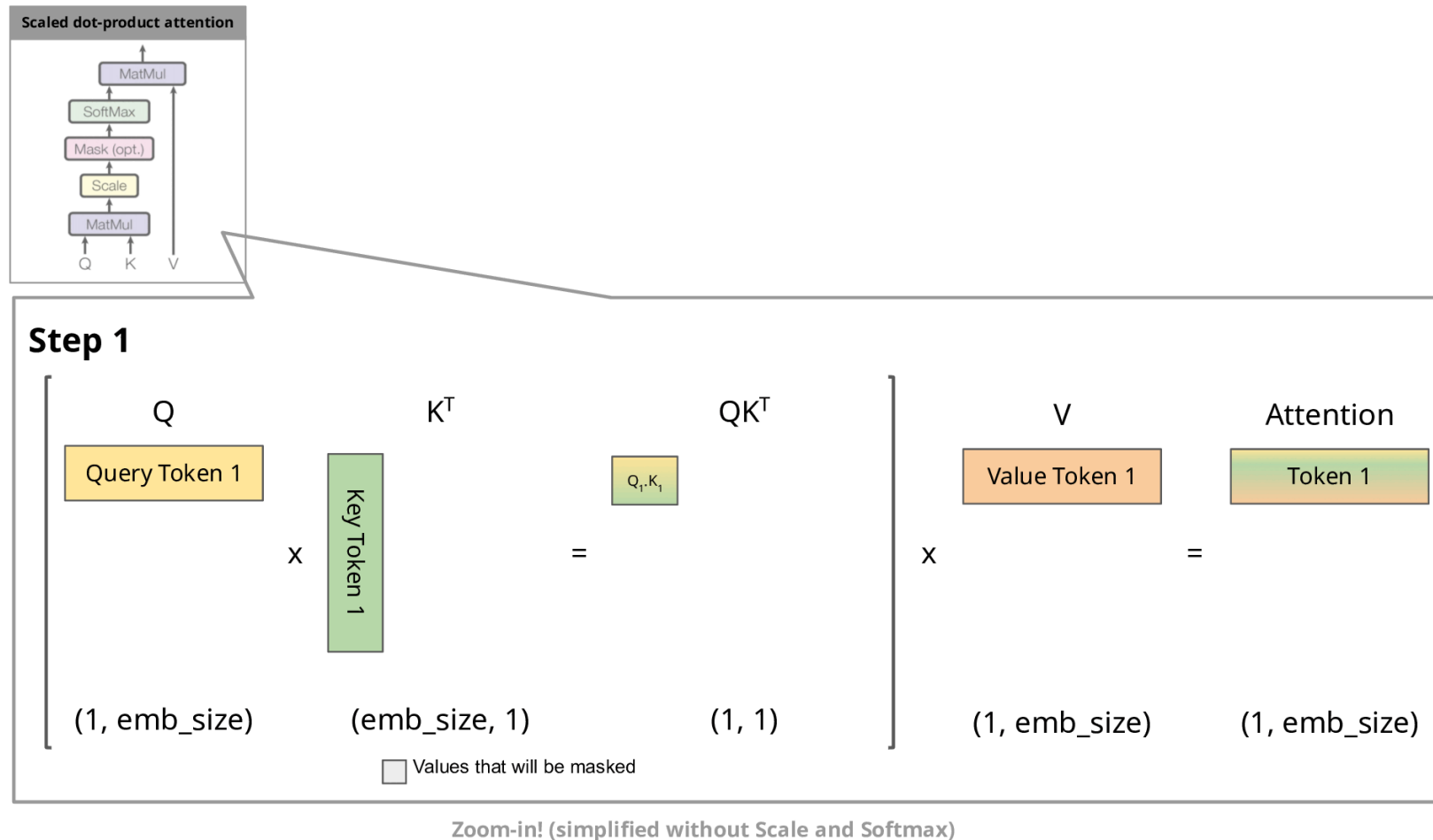


Multi-Head Attention

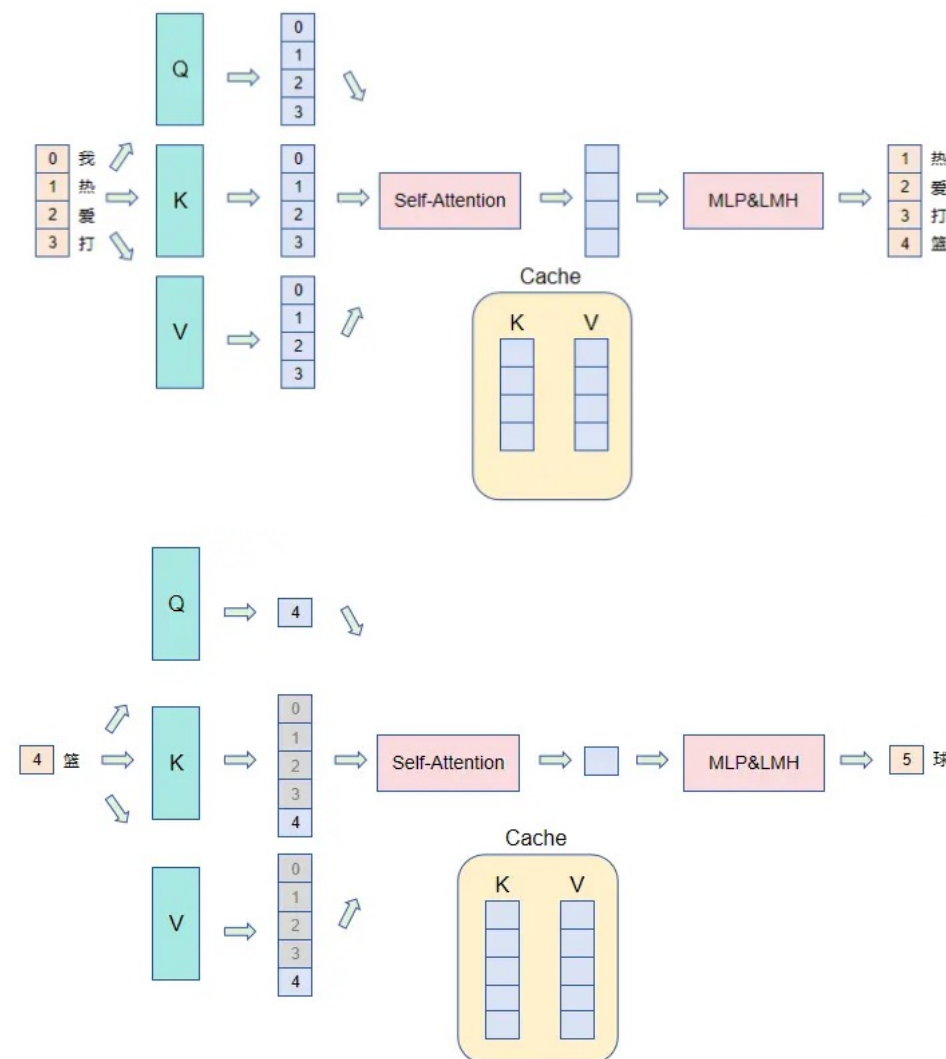
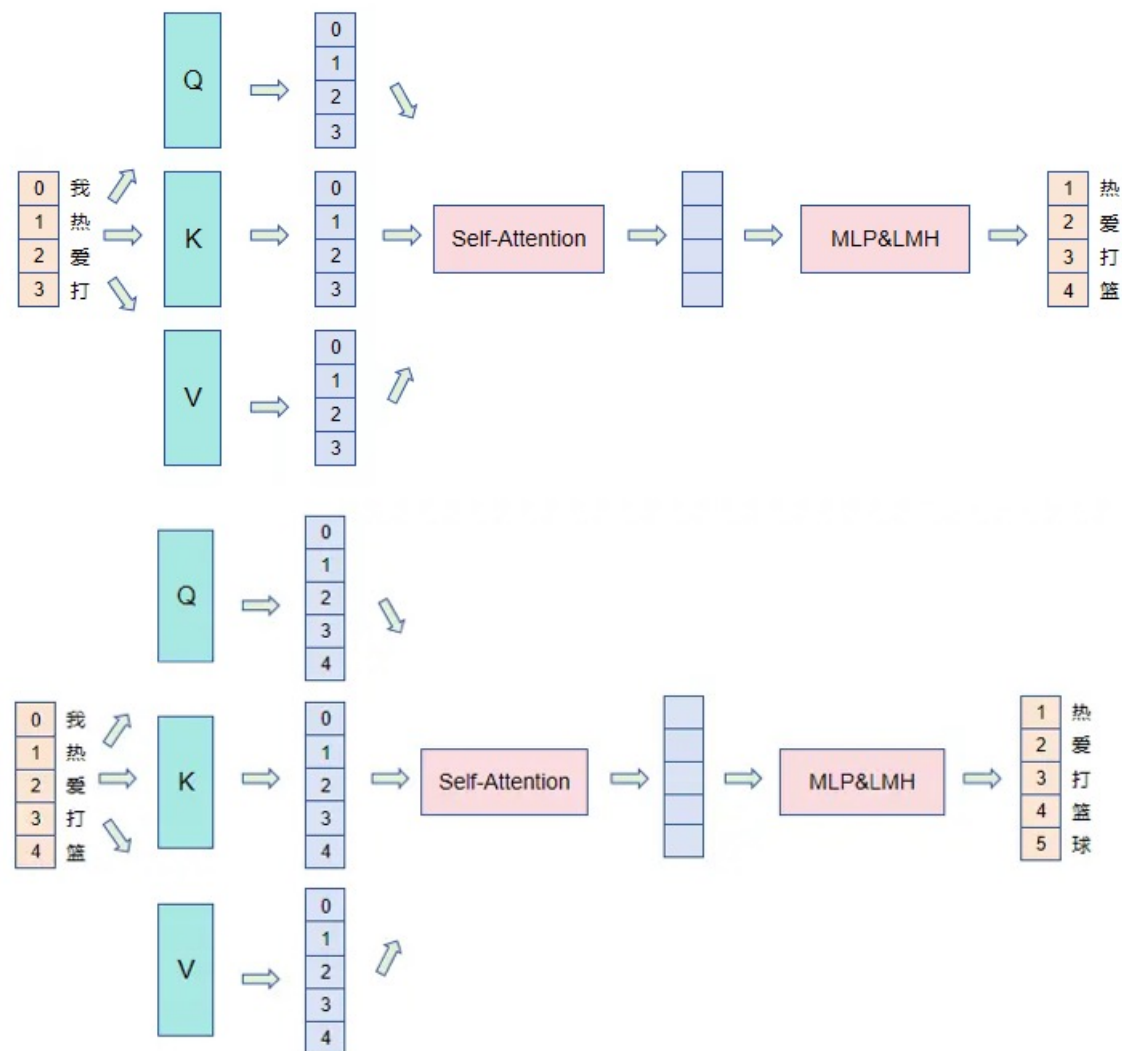


<https://zhuanlan.zhihu.com/p/624740065>

Transformer & Large Language Model (LLM)



KV-Cache Optimization



Check Related Code

- <https://github.com/tinygrad/tinygrad>
- `tinygrad/examples/transformer.py`

Homework for

- Get familiar with Attention model computation