



Accelerator Architectures for Machine Learning (AAML)

Lecture 1: Basics of AI Accelerator

Tsung Tai Yeh

Department of Computer Science
National Yang-Ming Chiao Tung University



Acknowledgements and Disclaimer

- Slides was developed in the reference with
Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019
tutorial
Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin
Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC
Berkeley, 2020
CS231n Convolutional Neural Networks for Visual Recognition,
Stanford University, 2020



Outline

- Dennard Scaling vs Dark Silicon
- Artificial Neural Network (ANN)
- Spiking Neural Network (SNN)
- Neuromorphic architectures
- Digital vs Analog Accelerators
- Convolution Neural Network
 - Residual network
 - Depthwise convolution



Why do we need accelerators ?

- Previously
 - We focused on designing general-purpose processors
- Why do accelerators have become attractive in recent years?
 - **Dennard Scaling** has ended
 - Dennard Scaling allowed voltage to shrink with transistor size
 - Without Dennard Scaling, we need to find other ways to keep **power** in check



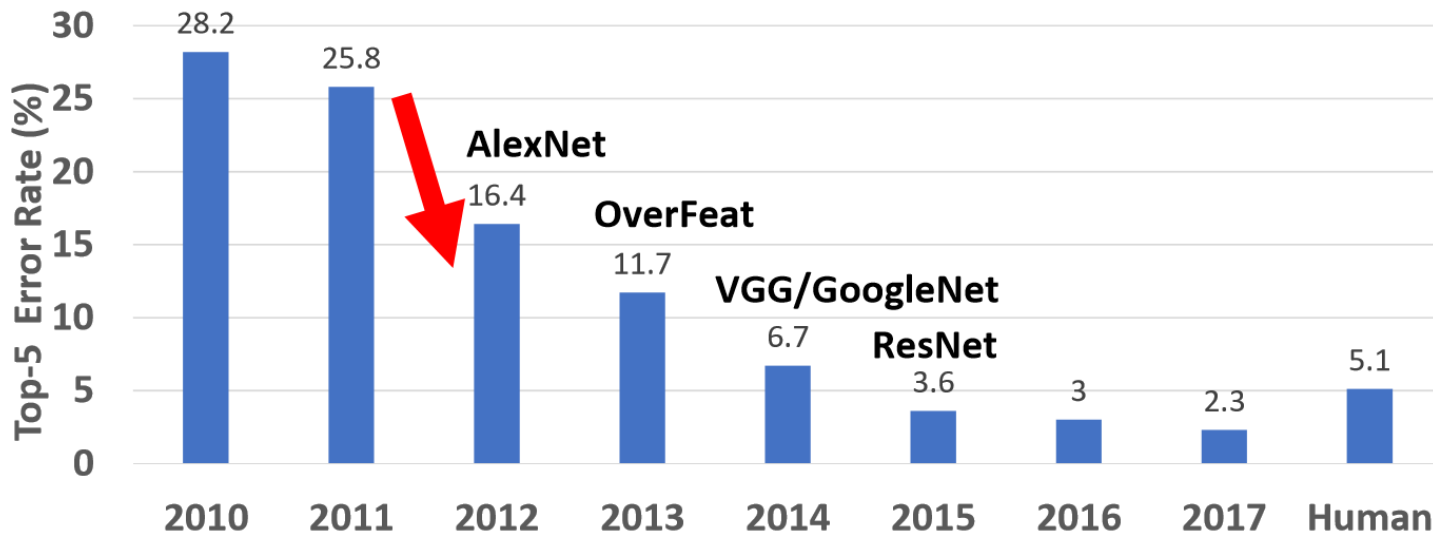
Why do we need accelerators ?

- Why do accelerators have become attractive in recent years?
 - **Dark Silicon**
 - Not turn on all transistors on the chip
 - The success of application's accelerators (encryption, compression ...)
 - Applications only use subset of processors/accelerators at a time, such a heterogeneous architecture meets dark silicon phenomenon



Why Deep Neural Network become popular?

- DNN model outperforms human-being on the ImageNet Challenge

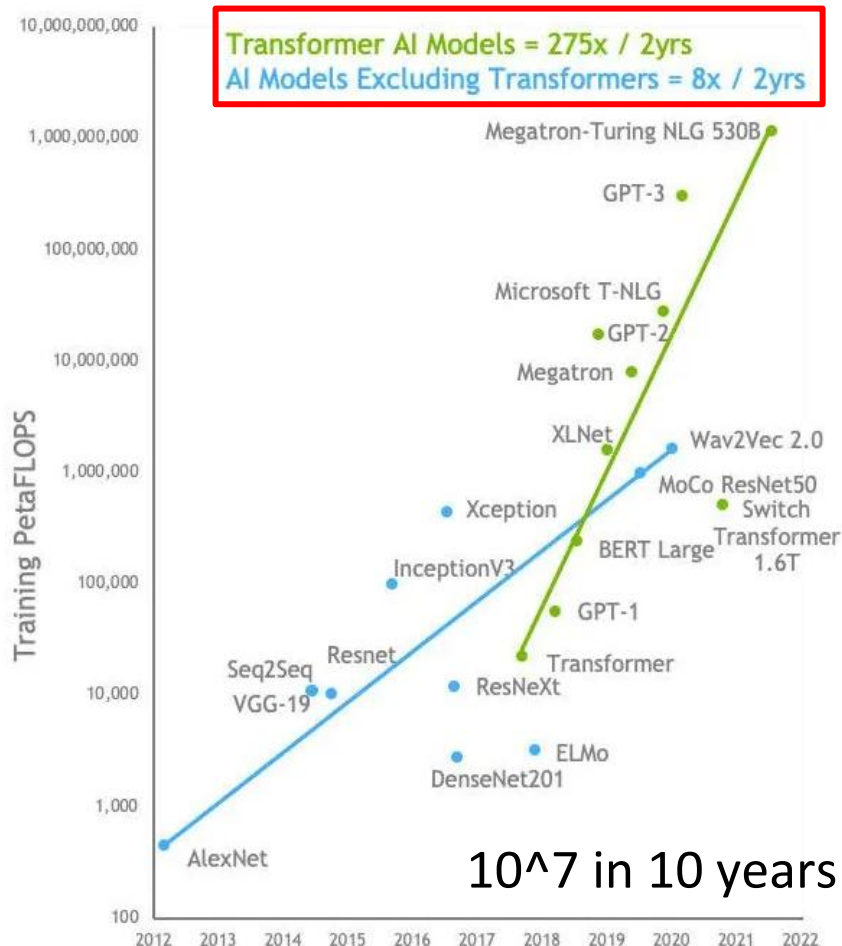




Computations of DNN

- **Deep Neural Network is getting large**
 - Large model parameters
 - Palm-E (540B)
 - GPT-MoE (1.8T)
 - Why do we need such a large model?

EXPLODING COMPUTATIONAL REQUIREMENTS





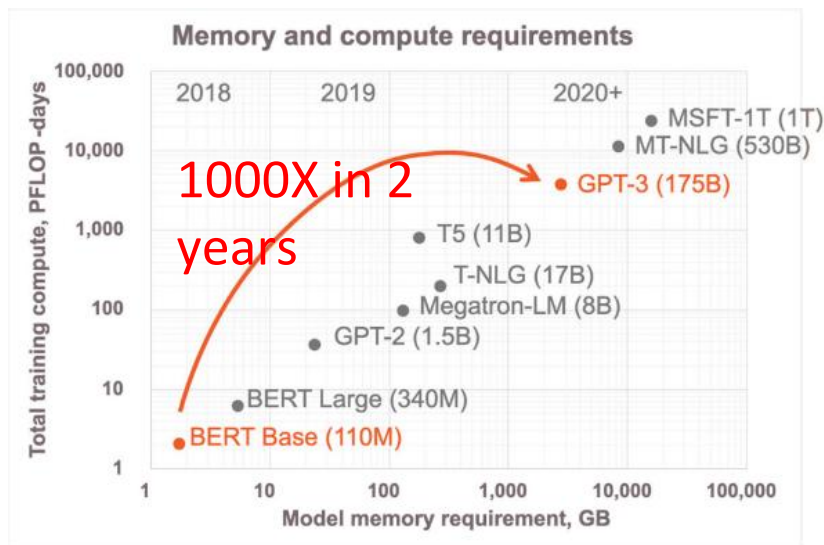
Large Language Model

- **A language model is a mathematical mapping**
 - Text $\rightarrow$ embedding vector (“representation”)
 - Embedding vectors encode meaning of texts i.e. dog [1,0,0]
 - Train such a model via **next-word prediction** on a large corpus of text data as the **lossy compression**
 - 1000B text token $\rightarrow$ 30B model parameters
 - Empirically, LLMs behaves as human as the model size increases



Unsustainable ML Model Growth

- We need a better way to grow models more efficiently
- Get the advantages of larger models but with substantially less compute and memory resources



Sparsity
might be one
of answer



Hardware trends

- **Stagnant single and multi-thread performance on general-purpose cores**
- **What do accelerators matter?**
 - Dark silicon (emphasis on power-efficient throughput)
 - End of scaling
- **Emergence of machine learning**
 - Facilitate the pervasive of hardware acceleration as machine learning emerges as a solution for “everything”.



Commercial Hardware for Machine Learning

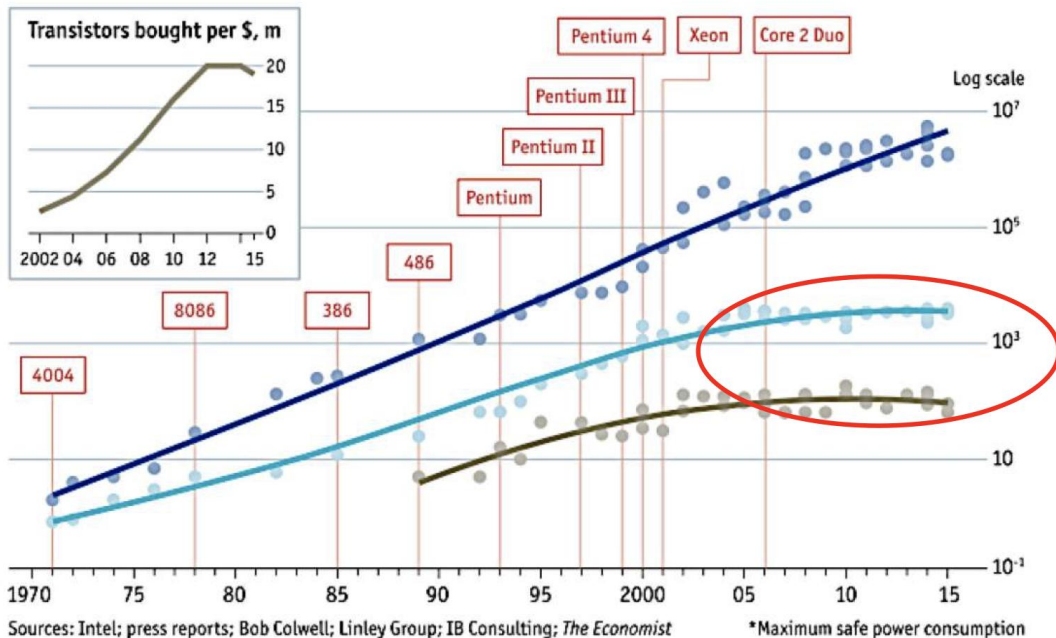
- Google TPU (inference and training)
- Nvidia Tensor/transformer cores (Ampere, Hopper)
- Microsoft Brainwave and Catapult
- Intel Loihi NPU
- Cambricon
- Graphcore (training)
- Cerebras (Training)
- Tesla (FSD, Dojo)
- ...



Increasing transistors is not getting efficient

Stuttering

● Transistors per chip, '000 ● Clock speed (max), MHz ● Thermal design power*, w □ Chip introduction dates, selected





Dennard Scaling

- Dennard scaling allowed voltage to shrink with transistor size
 - E.g. 180 nm -> 1.8 V, 130 nm -> 1.3 V
 - All 4 cores (45 nm) can be worked in full speed
 - Could all 8 cores (28 nm) be worked in full speed, too ? Why ?

Power = $\alpha \times CFV^2$

alpha: percent time
switched

C: capacitance

F: Frequency

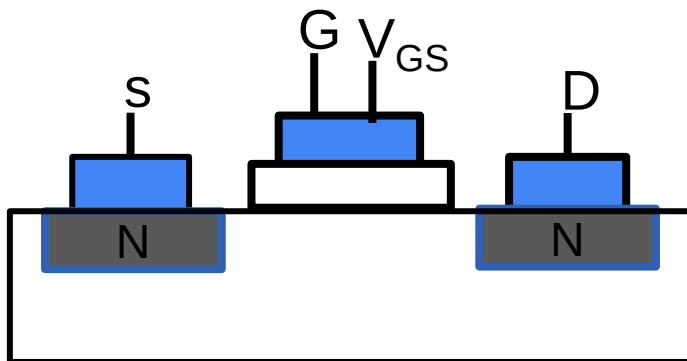
V: Voltage

1. Typically, the transistor size reduces K (~ 1.4) times
2. In the same chip area, **the number of transistor** increases K^2 times, the **frequency** increases K times
3. **The size of capacitance** shrinks K times as the reduction of transistor size, and **the voltage** reduces K^2 times
4. So, we can boost performance of the chip without any compensation of the power

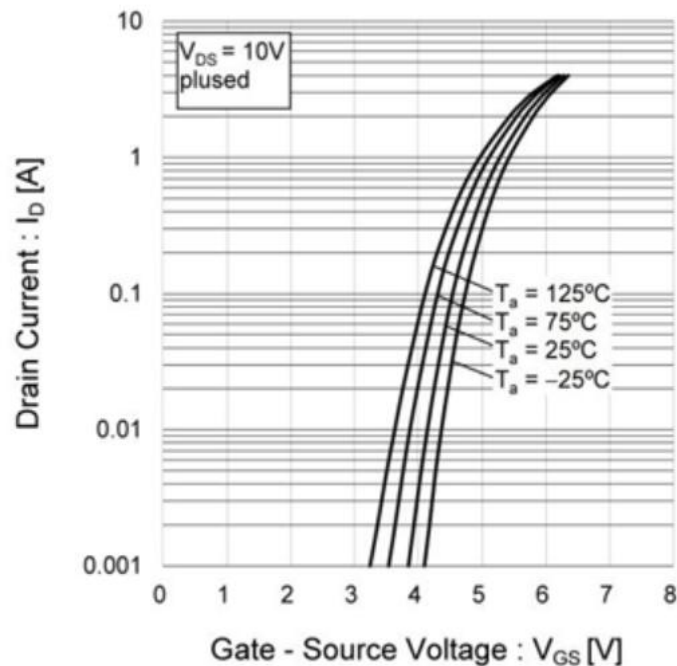


Voltage threshold of MOSFET

- Temperature affects the value of V_{GS} and I_D
 - $T_a = 25^\circ\text{C}$, $I_D = 1\text{A}$ and $T_a = 75^\circ\text{C}$, $I_D = 1.5\text{A}$ when fixing V_{GS}
 - Due to $V_{GS(TH)}$ constraint, difficult to keep reducing voltage to be proportional to the transistor size below 28 nm



Nch MOSFET





What can we do ?

- **Dark silicon**

- Below 28 nm, the voltage (V) is hard to be changed
- $K^2 = (\text{transistor size as capacitance size}) \times K \times \text{frequency (K)}$
- The power increases K^2 times
- Therefore, **not turn on all transistor on the chip**
- What is the percentage of inactive transistors ?
- 20 nm: 33%, 16 nm: 45%, 10 nm: 56%, 7 nm: 75%, 5 nm: 80%

- **Dim silicon**

- Turn all transistor on at low clock speeds

Power = $\alpha \times C F V^2$

alpha: percent time switched

C: capacitance

F: Frequency

V: Voltage



Heterogeneous SoC

- **Post-Moore era and dark silicon**

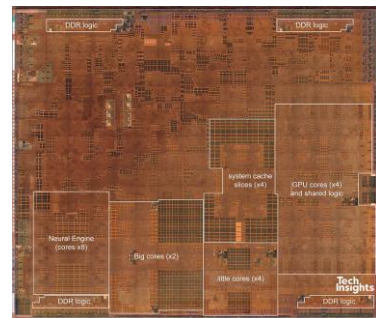
- A suite of accelerators on chip are rising
- Applications will only use a subset of processors/accelerators at a time
- Such a heterogeneous architecture is compatible with dark silicon



2010 Apple A4
65 nm TSMC 53 mm²
4 accelerators



2014 Apple A8
20 nm TSMC 89 mm²
28 accelerators



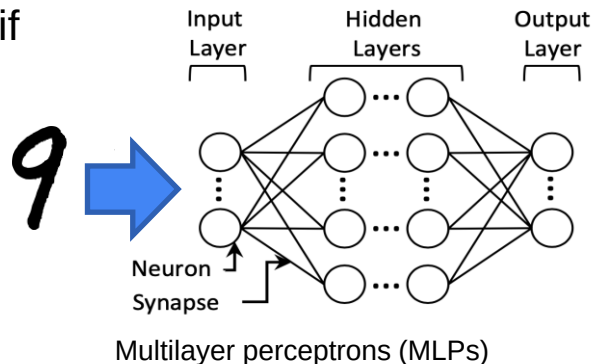
2019 Apple A12
7 nm TSMC 83 mm²
42 accelerators



Artificial Neural Network (ANN)

- **Most machine learning algorithms**

- Perceptron or artificial neuron
- Receiving synchronous inputs, and performs math, then produce outputs
- Measuring the “**strength**” (**z**) of weighted inputs
- $\mathbf{Z} = \mathbf{x1} * \mathbf{w1} + \mathbf{x2} * \mathbf{w2}$ where (x is the input of the neuron, w is the weight (determined by training))
- **Activation function** $a = f(z)$ to decide if a neuron should fire or not
- **Training performs back-propagation** with gradient descent



9
5

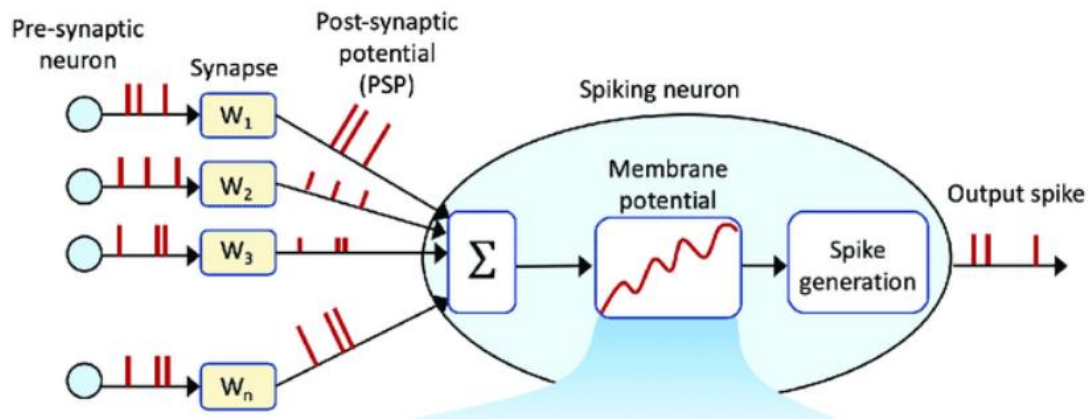
Probability:
0.8

Probability:
0.3



Spiking Neural Network (SNN)

- Spiking neurons resembles chemical reactions in our brains
 - A neuron has a **certain potential** that represents inputs received
 - The potential **rises and falls** depending on the relative importance of those inputs and leaks away when no receiving inputs
 - When the potential reaches a **threshold**, **the neuron fires**
 - All inputs/outputs are in the form of **binary spikes**





ANN vs. SNN

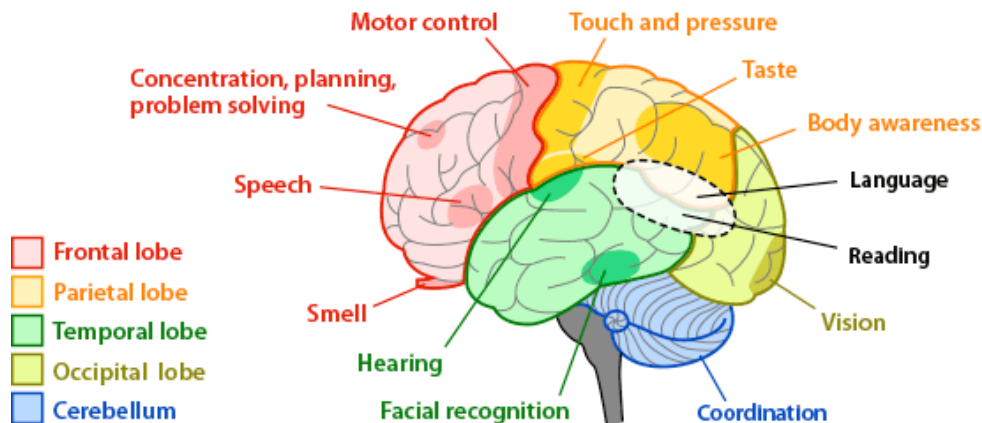
- **ANN**
 - Perceptron, 8-bit or 16-bit multiplications, complex activation functions
 - High accuracy, supervised learning (inference and training)
- **SNN**
 - Don't achieve very high accuracies, not well understood
 - A neuron has state that is a more powerful construct for applications that have a notion of time, e.g. video and language analysis
 - Carry a large amount of information in a few bits
 - Unsupervised learning



Uncover Your Brain

$2400 \text{ kcal}/24 \text{ hr} = 100 \text{ kcal/hr} = 27.8 \text{ cal/sec}$
 $= 116.38 \text{ J/s} = 116 \text{ W}$
 $20\% \times 116 \text{ W} = 23.3 \text{ W}$

- The computer as a brain that comprises **specialized accelerators**
- **Low power** – the brain consumes only about 20W
- **Fault tolerant** – the brain loses neurons all the time



Yang, Eric. [Think Dinner](#). Mac Evolution, 1998



Neuromorphic architectures

- Architectures inspired by neuron behavior
- **Two major flavors**
 - Artificial Neural Network (ANN)
 - Operations on perceptrons
 - Spiking Neural Network (SNN)
 - Mimic operations in the brain
- **Two major implementation styles**
 - Digital
 - Analog



Neuromorphic Hardware

- **Emulating the human brain**
 - Low power – the brain consumes only 20 W
 - Fault tolerant – the brain loses neurons all the time
 - No programming required – the brain learns by itself
- **Examples:**
 - SpiNNaker, Spikey, TrueNorth



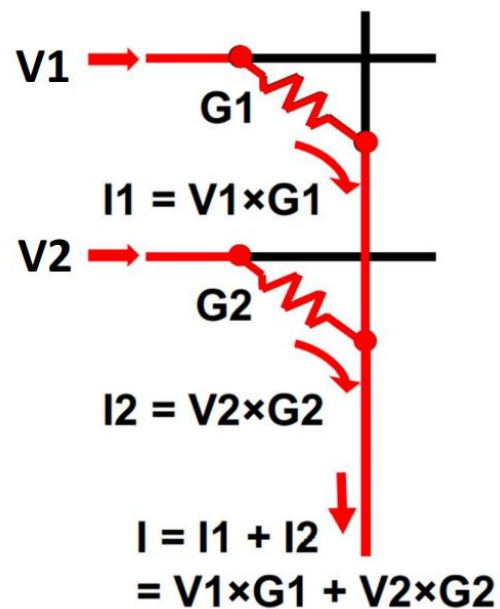
Digital vs. Analog

- **A single analog device**

- Perform multiple multi-bit operations
- Analog has **challenges, e.g., noise/precision**
- The **current** in a wire or the **charge** in a capacitor represent **a rational number**
- Perform **addition** by merging the currents in two wires
- **Multiplication** can be represented by the current that emerges when a voltage is applied to a conductor
- **Instability** as temperature changes, currents change

- **Digital device**

- Use CMOS transistors and gates, exclusively deal with 0s and 1s

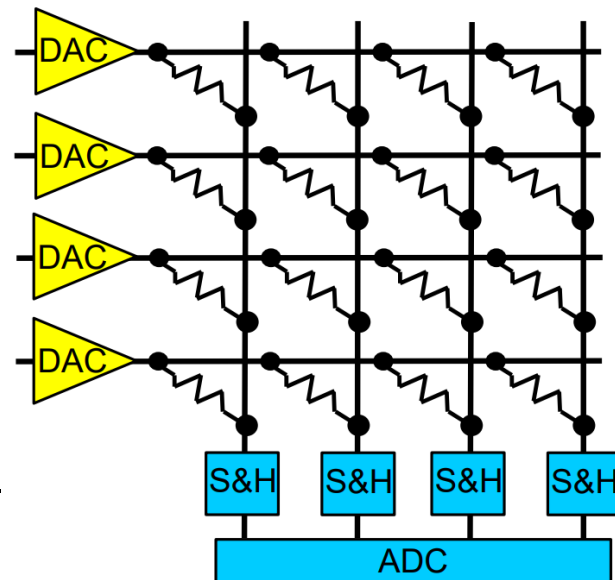


ISAAC, ISCA 2016



Crossbar for vector-matrix multiplication

- A grid of **resistances** and horizontal and vertical **wires**
 - The **input voltages** are provided on the horizontal wires (**word lines**)
 - Each **column** represents a different **neuron**
 - Each column computes a different **dot-product based on conductances** in that column
 - Analog current is sent through an analog-to-digital converter (**ADC**).
 - **S&H** is the sample-and-hold circuit that feeds signals sequentially to the ADC



ISAAC, ISCA 2016



Challenges of analog devices

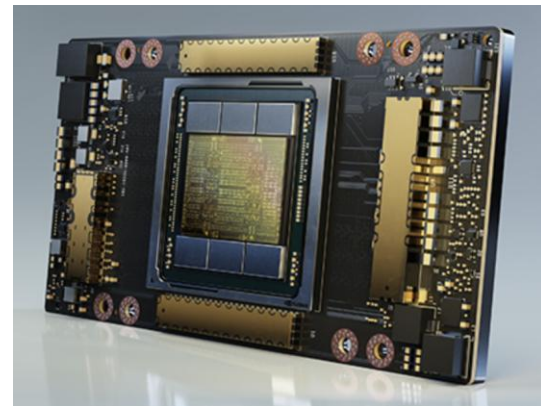
- **High ADC/DAC area/energy**
 - Long stay in analog needs expensive **analog buffering**, introduces **significant noise** that accumulates across network layers
 - Some ADC overheads increase exponentially with resolution
 - The number of bits coming out of a bitline is a function of **the bits of info in the voltage (v)**
 - **The bits of info in the weight (w)**
 - **The number of rows (R)** being added
 - To increase the parallelism and storage density – high v, w, and R
 - Demanding an expensive high-resolution ADC
 - SNN is amenable to analog, why ?



Digital (I) GPU

	Nvidia V100 GPU (2019)	Nvidia A100 GPU (2020)
Transistor count	21 billion	54 billion
FP32 performance	15.7 TFLOP/s	19.5 TFLOP/s
Tensor FP32	125 TFLOP/s	156 TFLOP/s
TDP	300 W	250 W
Die size	815 mm ²	862 mm ²
	TSMC 12 nm	TSMC 7 nm

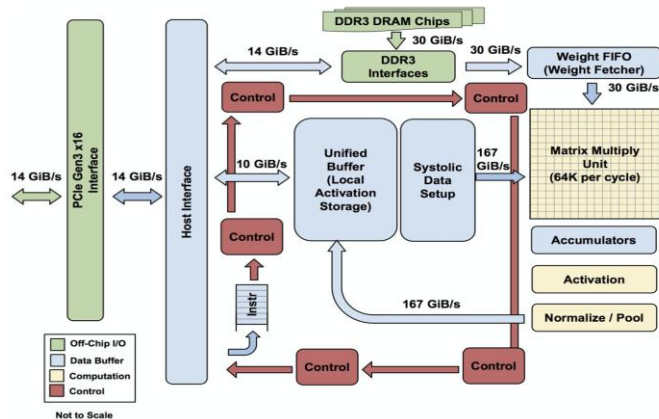
2.57 X
1.24 X
1.25 X





Digital (II) Google Tensor Processing Unit (TPU)

- Systolic-array accelerator
 - V1: Inference only
 - V2: Training with bfloat
 - V3: 2X powerful than v2
- Edge TPU
 - Coral Dev Board
 - 4 TOPS
 - 2 TOPS/Watt
 - Support TensorFlow Lite



<https://cloud.google.com/tpu/docs/tpus>

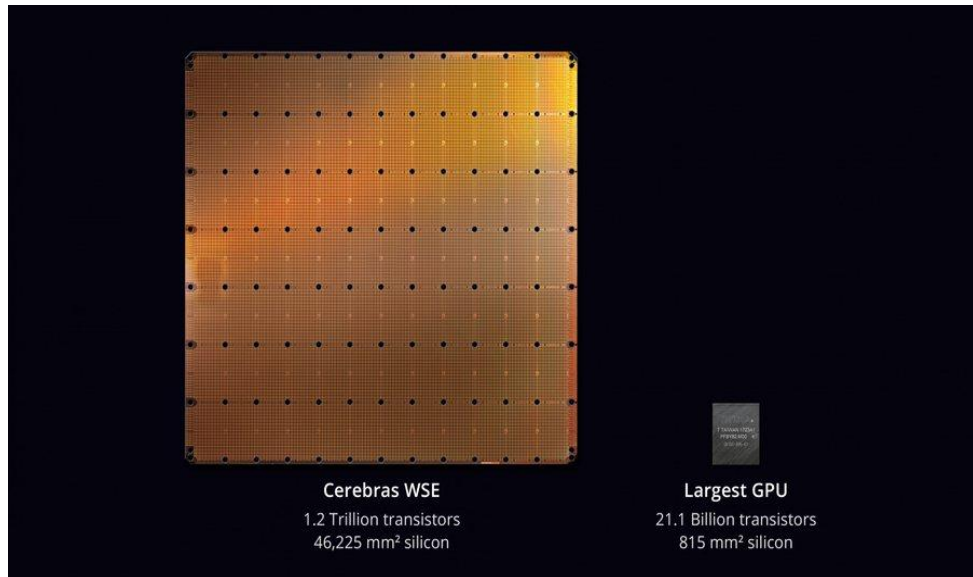


<https://coral.ai/products/>



Digital (III) Cerebras: Wafer-Scale DL Engine

- Largest DL Chip Ever Built!!
- 46225 mm² (WoW !!)
- 1.2 trillion transistor
- 400,000 optimized AI cores
- 18 GB on-chip memory
- TSMC 16 nm process





In summary

- **Learning from History**

- Neural network (NN) booms, but fades away when it ceases to be fashionable -> support vector machines (SVM) took over
- General-purpose processors and GPU quickly outpace ASICs

- **Today**

- NNs > SVM
- GPPs and GPUs will stagnate in performance, but ML is hot
- ML accelerators (hardware + ML software perspective) include many implementation operations
- Neuroscience + emerging technology



Takeaway Questions

- What does dark silicon tell us ?
 - (A) We should turn all transistor on at low clock speeds
 - (B) We cannot turn on all transistors on a chip
 - (C) Allowed voltage to shrink with transistor size
- Why does SNN have the potential for low-energy computations and communication ?
 - (A) Skipping connections
 - (B) Complex SNN computation
 - (C) Not involve in multiplications or complex activation functions



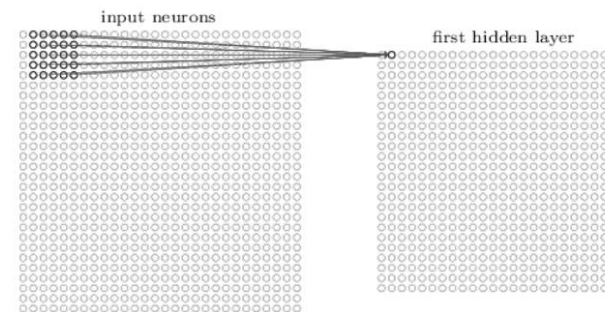
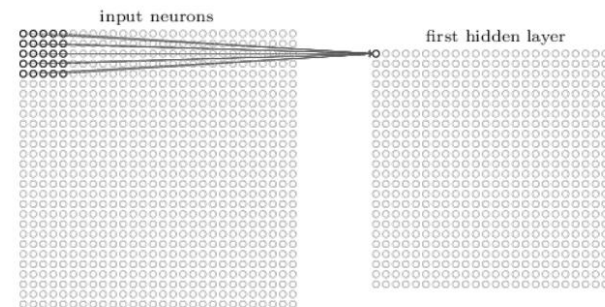
Takeaway Questions

- What are the challenges of analog accelerators ?
 - (A) High ADC/DAC area and energy
 - (B) Limited parallelism
 - (C)Non-programmable



Deep convolutional neural networks

- Each neuron only sees a “**local receptive field**”
 - 5 x 5 grid of neurons in this example
 - The first neuron is looking for feature in the top-left 5x5 corner of the image
 - Combines the 25 inputs with 25 synaptic weights to decide its output
 - The set of 5x5 weights as a “**filter**”

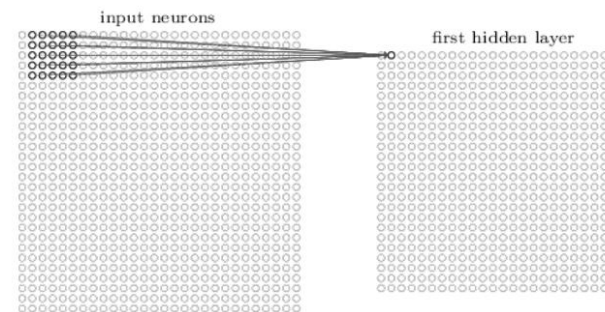
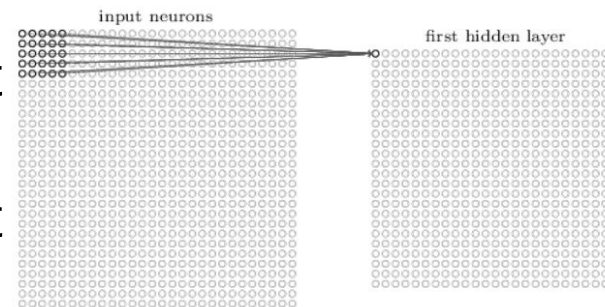




Deep convolutional neural networks

- **Convolution**

- Applying a 5x5 filter (kernel) to each part of the image
- All the neurons are sharing the same set of 25 weights (plus bias)
- Why do we create small size filter ?
 - The small local receptive field and the use of shared weights can help for slow learning rate in early layers of the network





Convolutional Computation Details

• Convolution

- Sliding dot product or cross-correlation
- Convoluting a 5x5x1 image with a 3x3x1 filter kernel to get a 3x3x1 convoluted feature

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x1}	1	0
0 _{x1}	0 _{x1}	1 _{x0}	1	1
0	0	1	1	0
0	1	1	0	0

Image



5		

Convolved
Feature

1	1 _{x1}	1 _{x0}	0 _{x1}	0
0	1 _{x0}	1 _{x1}	1 _{x1}	0
0	0 _{x1}	1 _{x1}	1 _{x0}	1
0	0	1	1	0
0	1	1	0	0



5	4	



CNN Dimension Parameters

- N – Number of **input fmaps/output fmaps** (batch size)
- C – Number of 2D **input fmaps/filters** (channels)
- H – Height of **input fmap** (activations)
- W – Width of **input fmap** (activations)
- R – Height of 2D **filter** (weights)
- S – Width of 2D **filter** (weights)
- M – Number of 2D **output fmaps** (channels)
- F – Width of **output fmap** (activations)
- E – Height of **output fmap** (activations)



CONV Layer Tensor Computation

Output fmaps (Y) **Bias (B)** **Input fmaps (X)** **Filter weights (W)**

$$Y[n][m][x][y] = \text{Activation}(B[m] + \sum_{i=0}^{R-1} \sum_{j=0}^{S-1} \sum_{k=0}^{C-1} X[n][k][Ux+i][Uy+j] \times W[m][k][i][j])$$

$$0 \leq n \leq N, 0 \leq m \leq M, 0 \leq y \leq E, 0 \leq x \leq F$$

$$E = (H - R + U)/U, F = (W - S + U)/U$$

Shape Parameter	Description
N	fmap batch size
M	# of filters or # of output fmap channels
C	# of input fmap or # of filter channels
U	Convolution stride



CONV Layer Implementation

```
for ( n = 0; n < N; n++) {  
    for ( m = 0; m < M; m++) {  
        for ( x = 0; x < F; x++) {  
            for ( y = 0; y < E; y++) {  
                Y[n][m][x][y] = B[m];  
                for ( i = 0; i < R; i++) {  
                    for ( j = 0; j < S; j++) {  
                        for ( k = 0; k < C; k++) {  
                            Y[n][m][x][y] += X[n][k][Ux+i][Uy+j] x W[m][k][i][j]  
                        }  
                    }  
                }  
                Y[n][m][x][y] = Activation(Y[n][m][x][y]);  
            }  
        }  
    }  
}
```

For each output fmap value

CONV & Activation

How to run CONV in parallel ?



CONV Layer Parallel Implementation

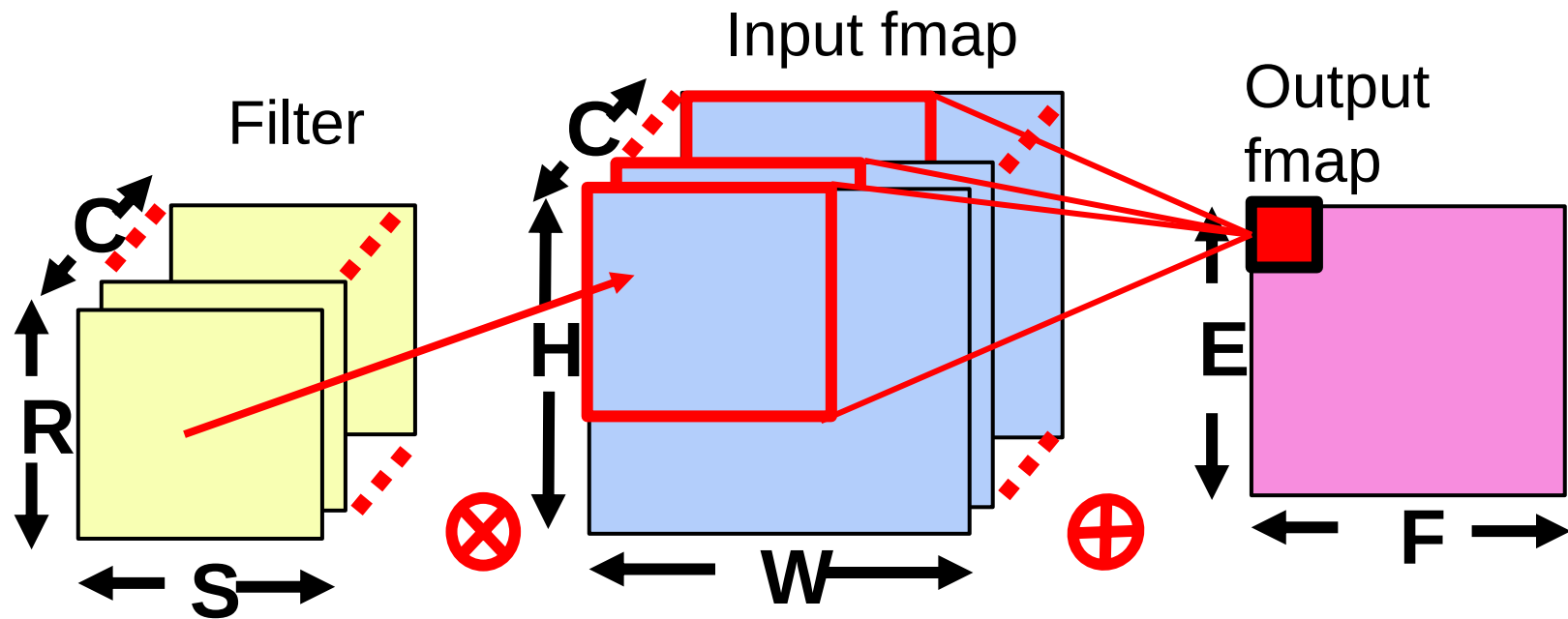
```
Parallel_for ( n = 0; n < N; n++ ) {  
  Parallel_for ( m = 0; m < M; m++ ) {  
    Parallel_for ( x = 0; x < F; x++ ) {  
      Parallel_for ( y = 0; y < E; y++ ) {  
        Y[n][m][x][y] = B[m];  
        for ( i = 0; i < R; i++ ) {  
          for ( j = 0; j < S; j++ ) {  
            for ( k = 0; k < C; k++ ) {  
              Y[n][m][x][y] += X[n][k][Ux+i][Uy+j] x W[m][k][i][j]  
            }  
          }  
        }  
        Y[n][m][x][y] = Activation(Y[n][m][x][y]);  
      }  
    }  
  }  
}
```

For each output fmap value

CONV & Activation



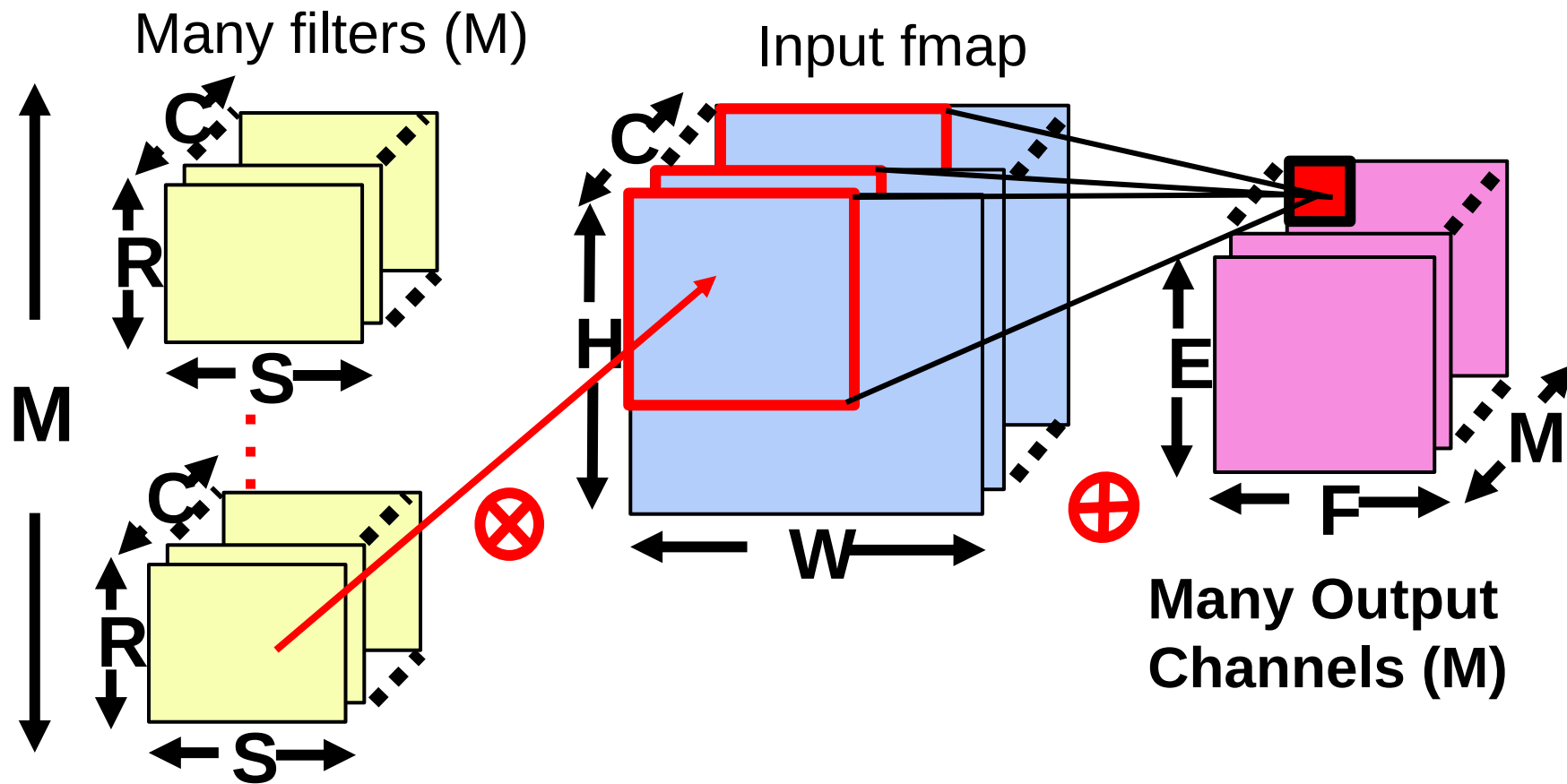
Convolution (CONV) Layer



Many Input Channels (C), e.g. RGB in an image

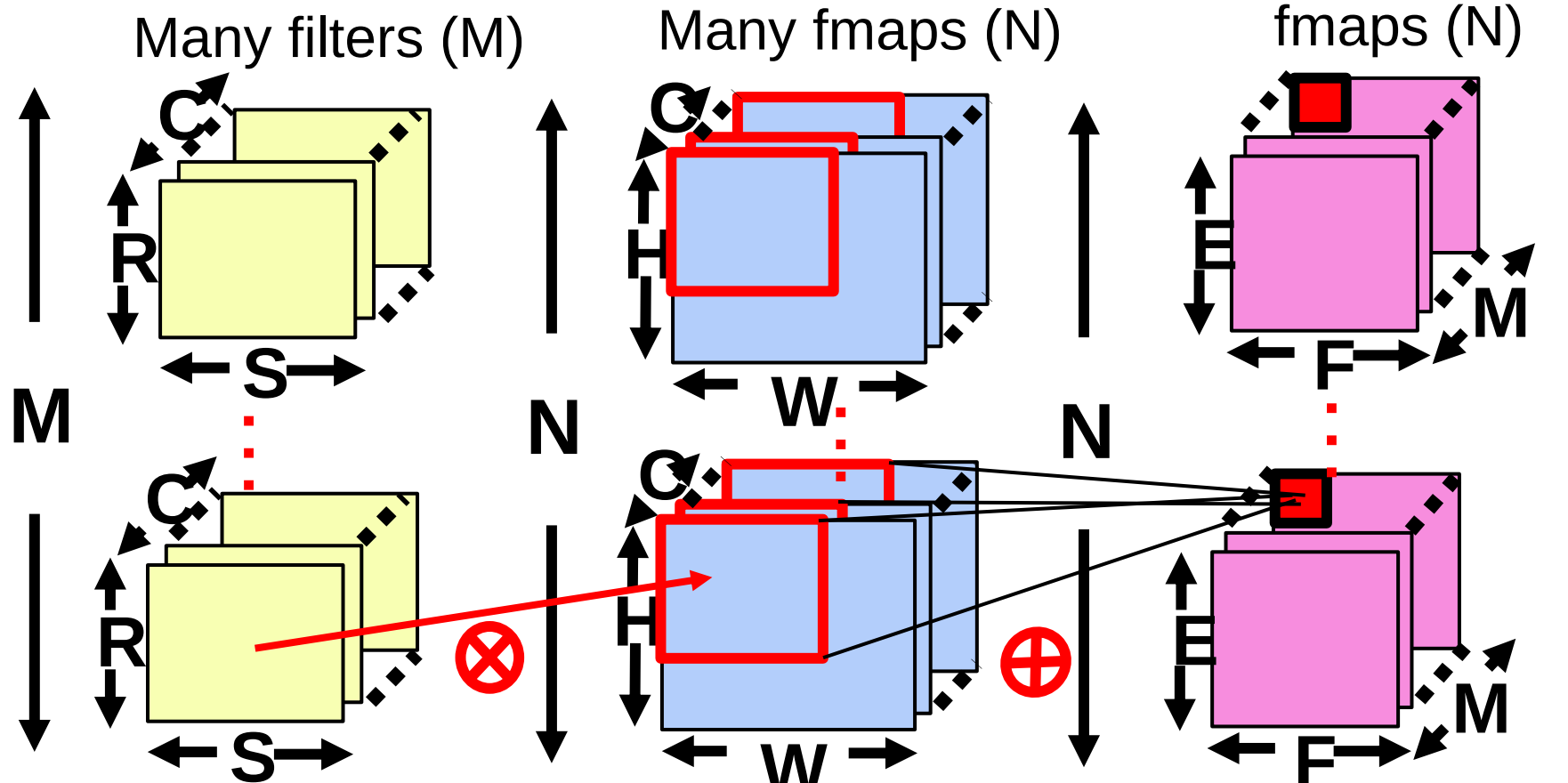


Convolution (CONV) Layer





Convolution (CONV) Layer

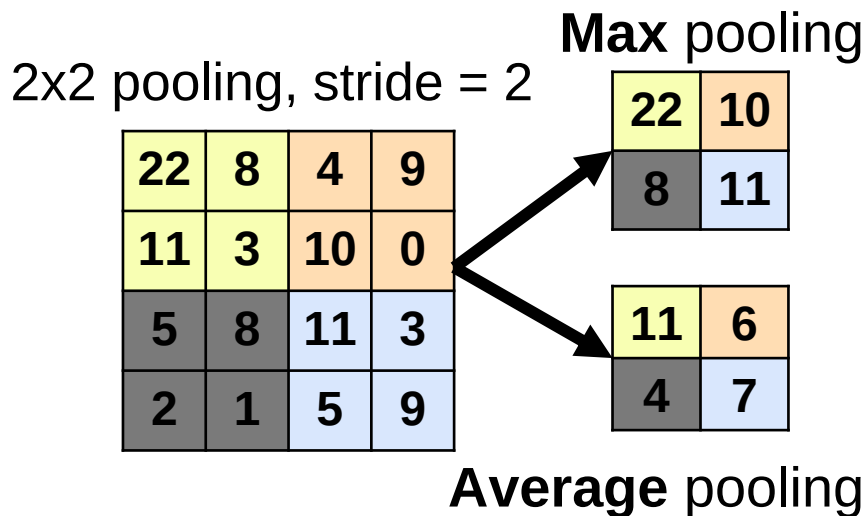




Pooling

• Pooling

- Once a feature has been found, its's exact location isn't as important as its relative location – help us reduce the parameters
- Further reduce the network, say reduce 4 neurons into a single one



Input fmap size: $W_1 \times H_1 \times C_1$

Spatial extent: F

Stride S

Output fmap after pooling: $W_2 \times H_2 \times$

C_2

$$W_2 = (W_1 - F) / S + 1$$

$$H_2 = (H_1 - F) / S + 1$$

$$C_2 = C_1$$

<https://cs231n.github.io/convolutional-networks/>



POOL Layer Implementation

```
for ( n = 0; n < N; n++) {  
    for (m = 0; m < M; m++) {  
        for (x = 0; x < F; x++) {  
            for (y = 0; y < E; y++) {  
                max = -Inf  
                for ( i = 0; i < R; i++) {  
                    for ( j = 0; j < S; j++) {  
                        if ( X[n][m][Ux+i][Uy+j] > max) {  
                            max = X[n][m][Ux+i][Uy+j];  
                        }  
                    }  
                }  
                Y[n][m][x][y] = max;  
            }  
        }  
    }  
}
```

} for each pooled
value

} Find the max in
each window



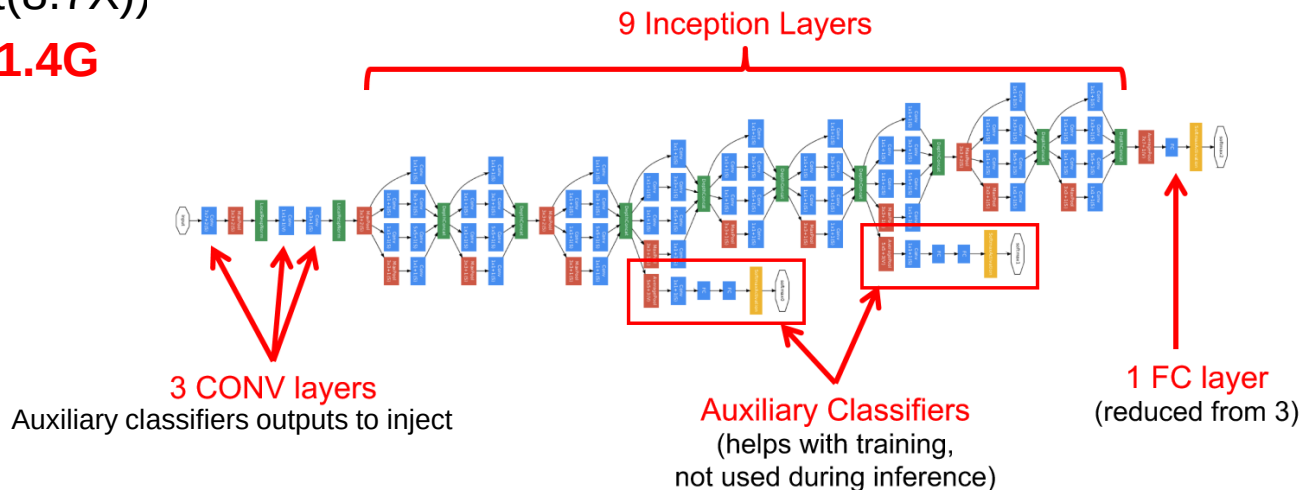
GoogleNet Inception Architecture

- 22 layers
- Fully-Connected Layers: 1
- **Weights: 7.0 M** (< VGG(19.7X)
AlexNet(8.7X))
- MACs: **1.4G**

ILSCVR14 Winner

GoogleNet is used to classify images

GoogleNet top-5 error rate is 6.67%
over VGG 7.3%

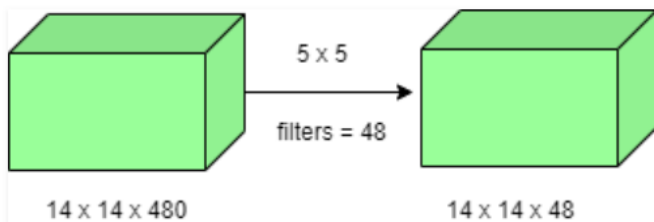




What's New in GoogleNet ?

- **1 x 1 CONV filter (why?)**

- Decrease the number of parameters (weights and biases)
- Increase the depth of the network

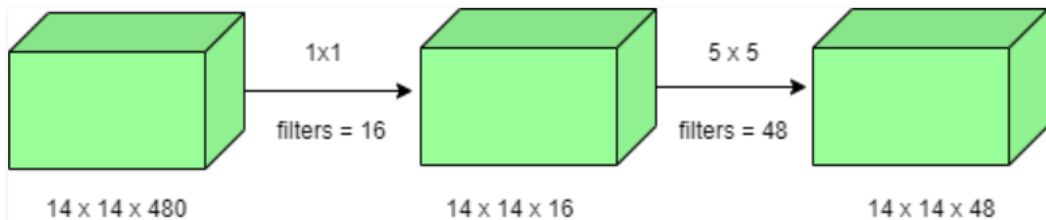


Case 1: 5x5 filter, # of filter = 48

Total MACs: $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = \mathbf{112.9M}$

Case 2: 1x1 filter, # of filter = 16 as intermediate

Total MACs: $(14 \times 14 \times 16) \times (1 \times 1 \times 480) + (14 \times 14 \times 48) \times (5 \times 5 \times 16) = \mathbf{5.3M}$



GoogleNet can be trained in a single machine such as (GPU with limit memory space) !!

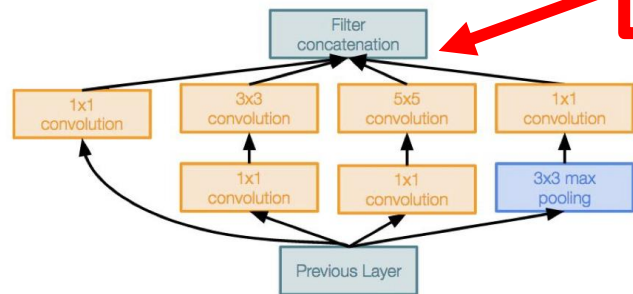


What's New in GoogleNet ?

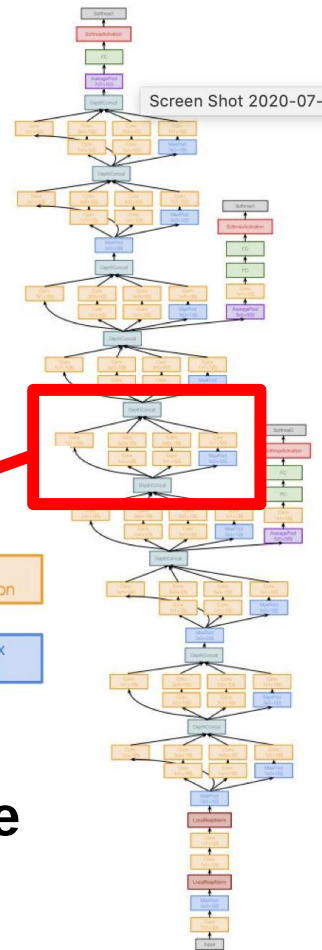
- **Inception module**

- A local network topology (network within a network)
- Stack modules on top of each other
- Multiple receptive field sizes for CONV (1x1, 3x3, 5x5)
- Pooling operation (3x3)
- Depth-wise filter concatenation

What's the problem of inception module?



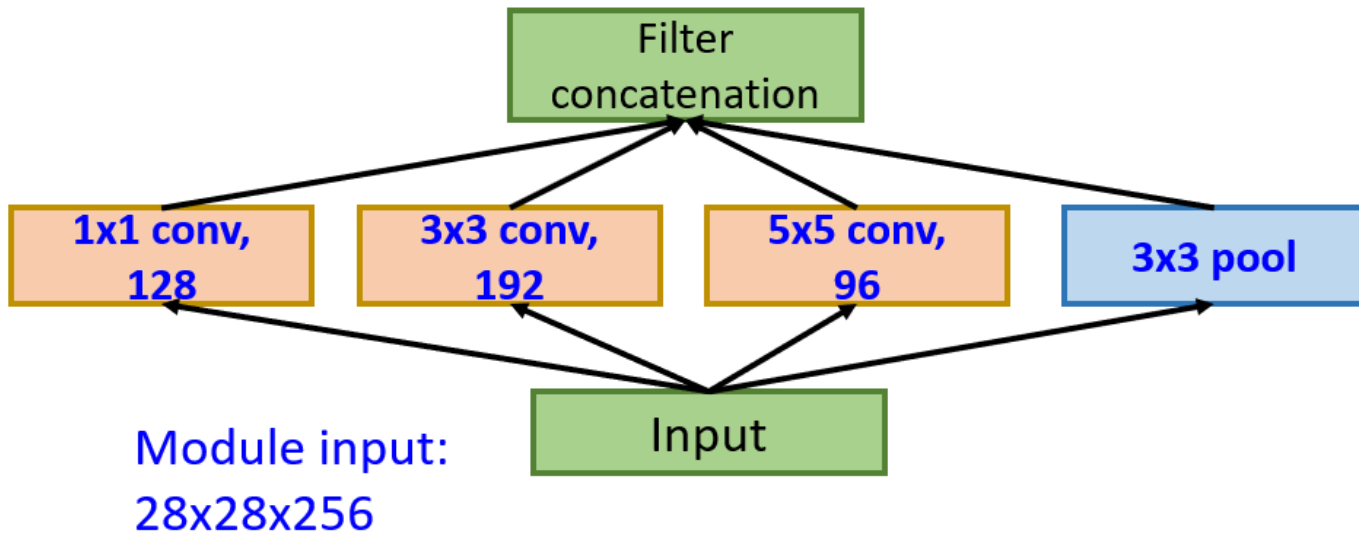
Inception module





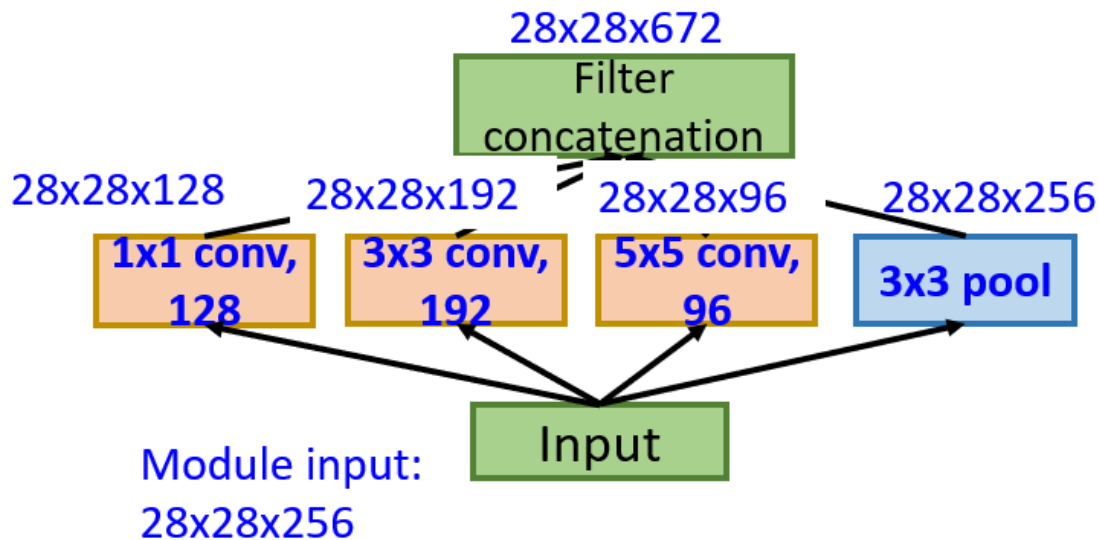
GoogleNet Inception Module Problems?

- What is the output size of 1x1 conv, with 128 filters ?





GoogleNet Inception Module Problems?



CONV Ops:

[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 256$

[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 256$

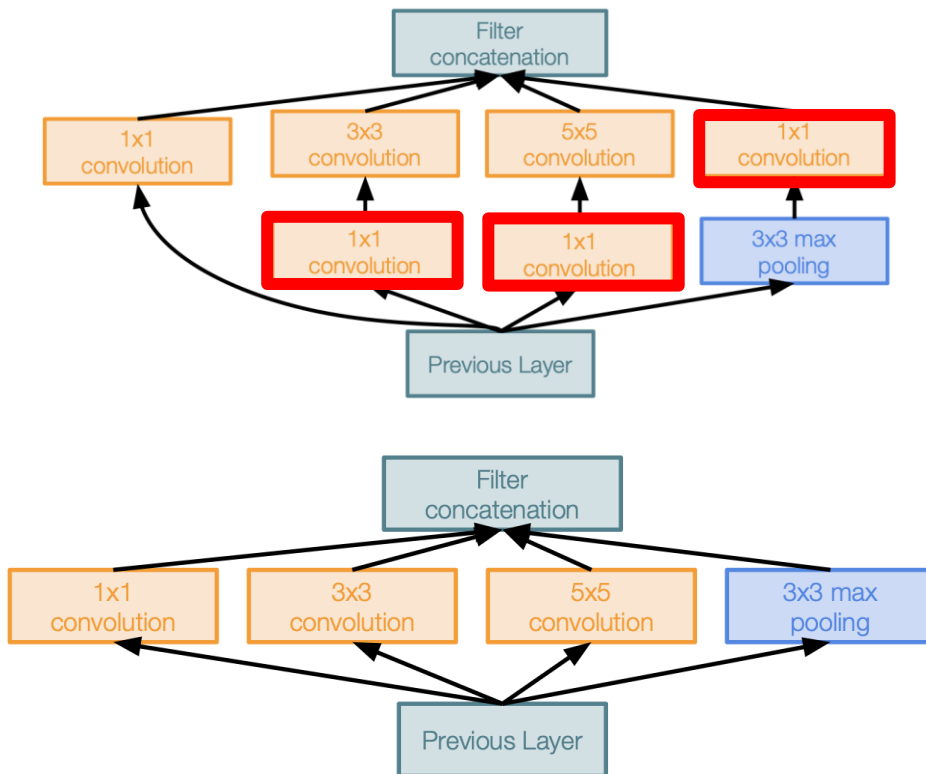
Total: 854 M ops

Very expensive compute

Solution: "bottleneck" layers
that use 1x1 convolutions to
reduce feature depth



Dimension Reduction on GoogleNet

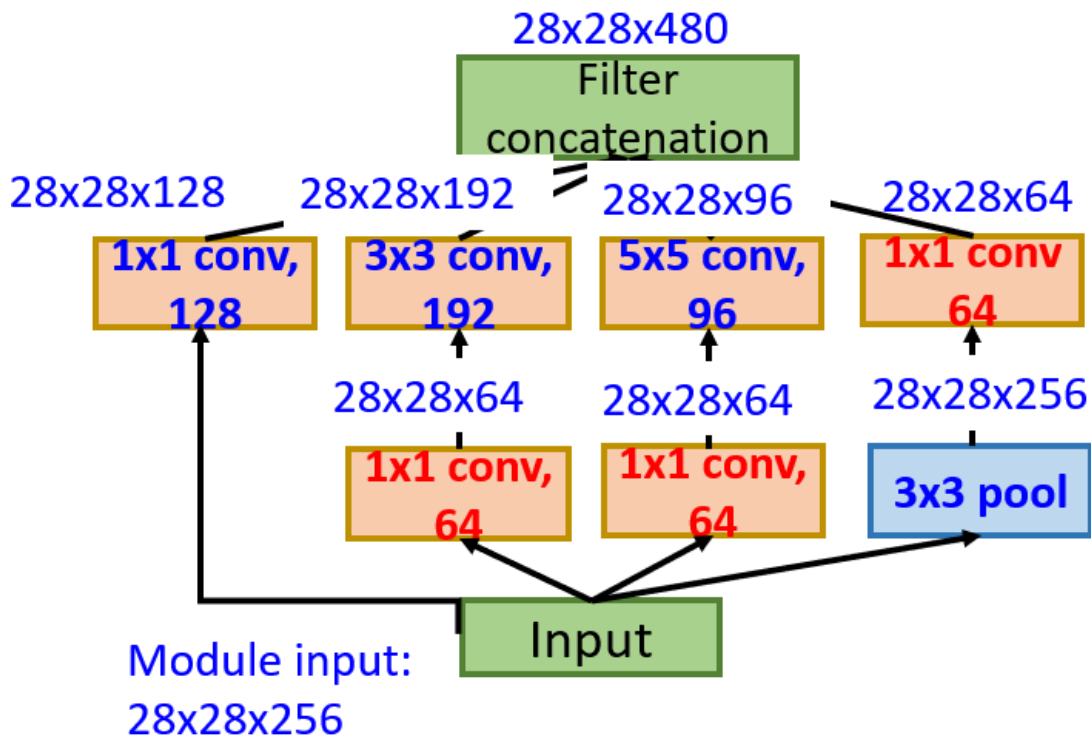


Inception module with
dimension reduction
using
**1x1 conv “bottleneck”
layers**

Naïve inception module



GoogleNet 1x1 Bottleneck Layer Ops



Conv Ops:

[1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
[1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$
[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 64$
[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 64$
[1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$

Total: 358 M ops

Naïve version has **854M ops**

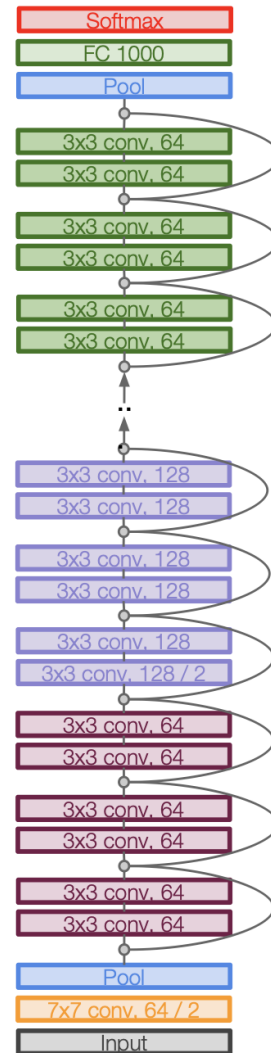
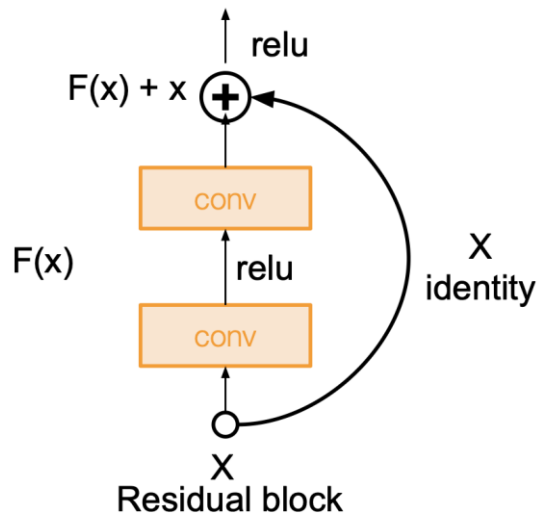
Bottleneck layer can reduce ops
using dimension reduction



ResNet Model Overview

- 152-layer model for ImageNet Classification
- ILSVRC'15 winner (3.57% top-5 error)
- Using residual blocks and connections

**How to train the data in
ULTRA-DEEP network
(over 1000 layers)?**





Deep Network Training Problems on ResNet

- Training and test error are increased with the length of networks



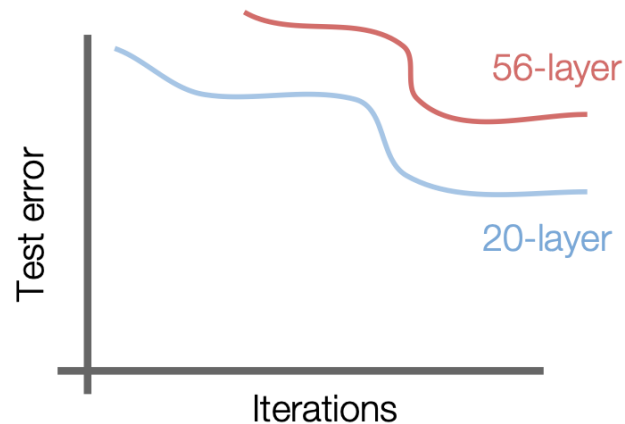
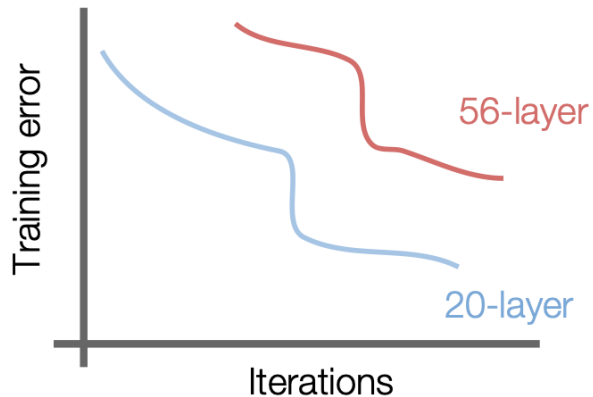
What is wrong when increasing the length of networks?

- Deeper model performs worse on both training and test error (overfitting?)



Deep Network Training Problems on ResNet

- Why overfitting isn't the main reason to increase error rate of 56-layer?
- Hypothesis: **vanishing gradient** raises error rate of ultra-deep networks?
 - Solution: Add layers to fit a residual mapping instead of fitting a desired underlying mapping directly (skipping connection)(What?)

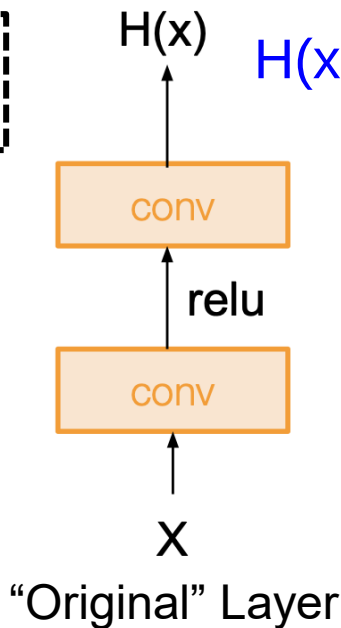




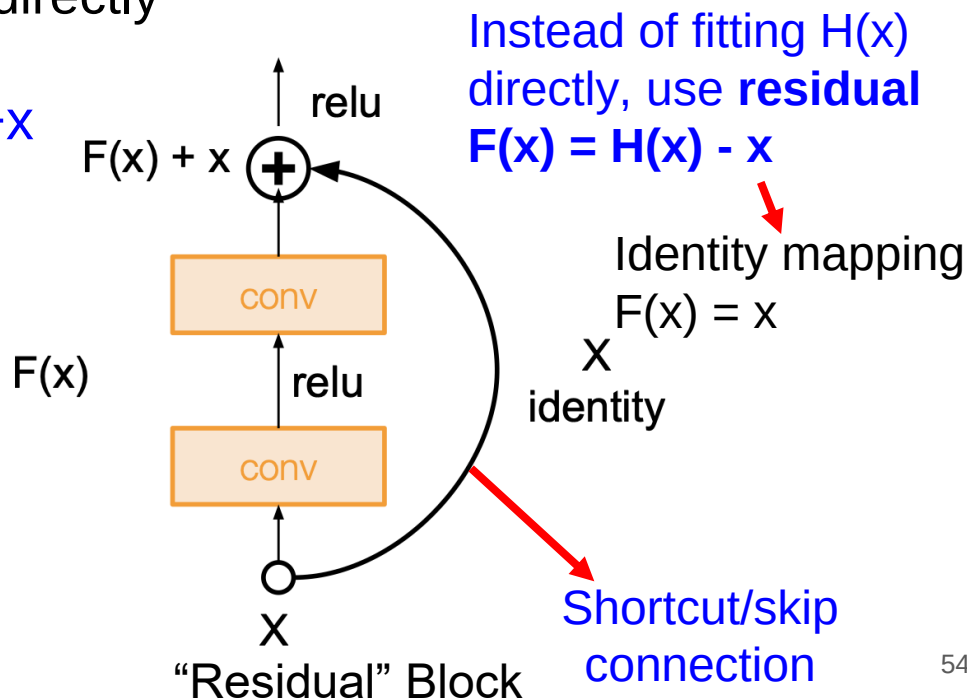
Deep Network Training Problems on ResNet

- Solution: Add layers to fit a residual mapping instead of fitting a desired underlying mapping directly

Why not use $H(x)$ directly?



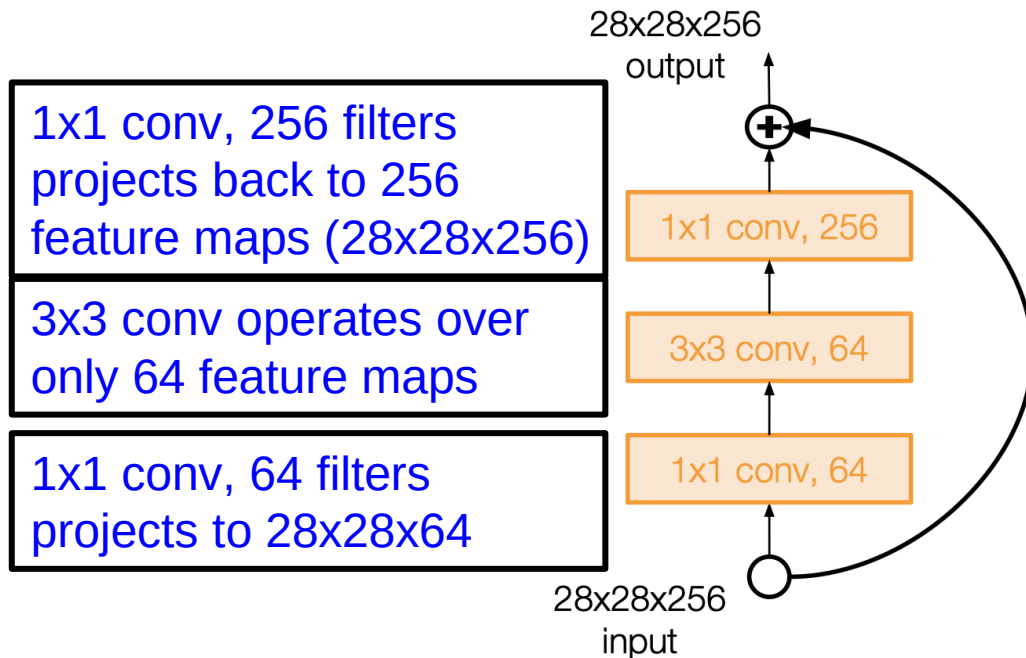
$$H(x) = F(x) + x$$





Bottleneck Layer on ResNet

- ResNet50+ also uses “bottleneck” layer to improve efficiency for deep networks (similar to GoogleNet)





- Full ResNet architecture

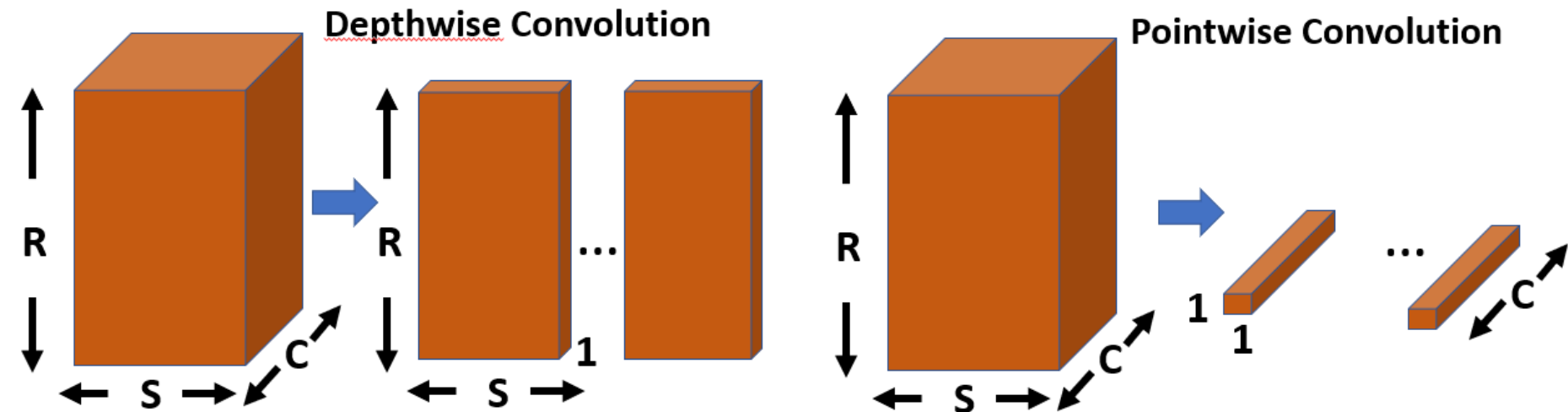
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double the number of filters and down-sample using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- Only FC 1000 to output class
- Total depths of 34, 50, 101 or 152 layers for ImageNet





MobileNet v1: Depthwise Separable CONV

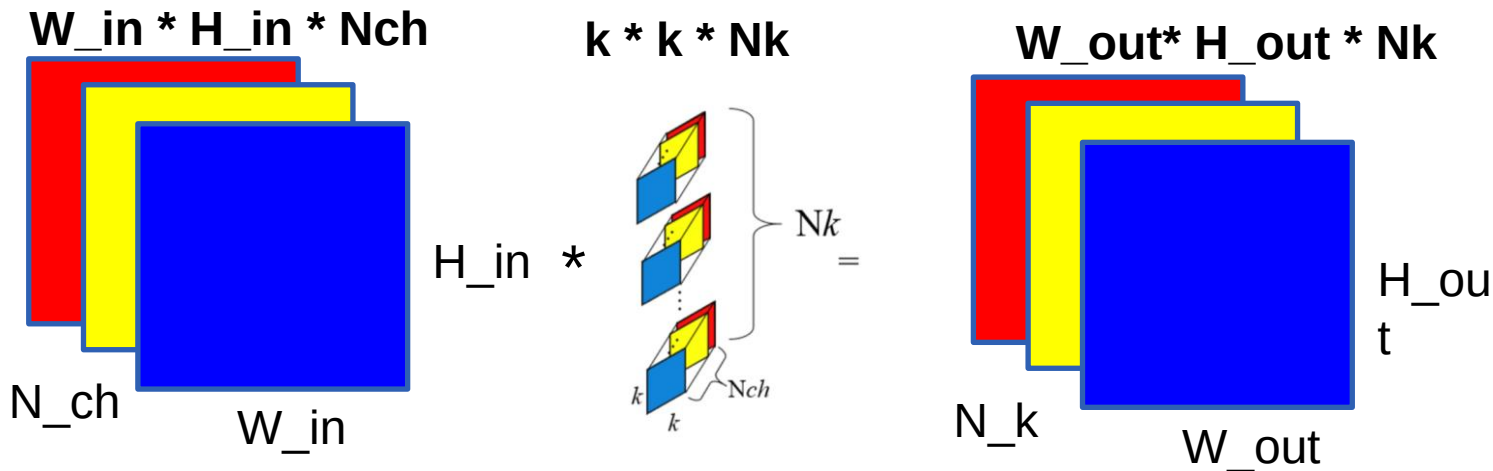
- Decouple **cross-channel correlations** and **spatial correlations** in the feature maps
- How to reduce # of parameters? [Andrew et. al. arxiv, 2017]





What is Depthwise Separable Convolution ?

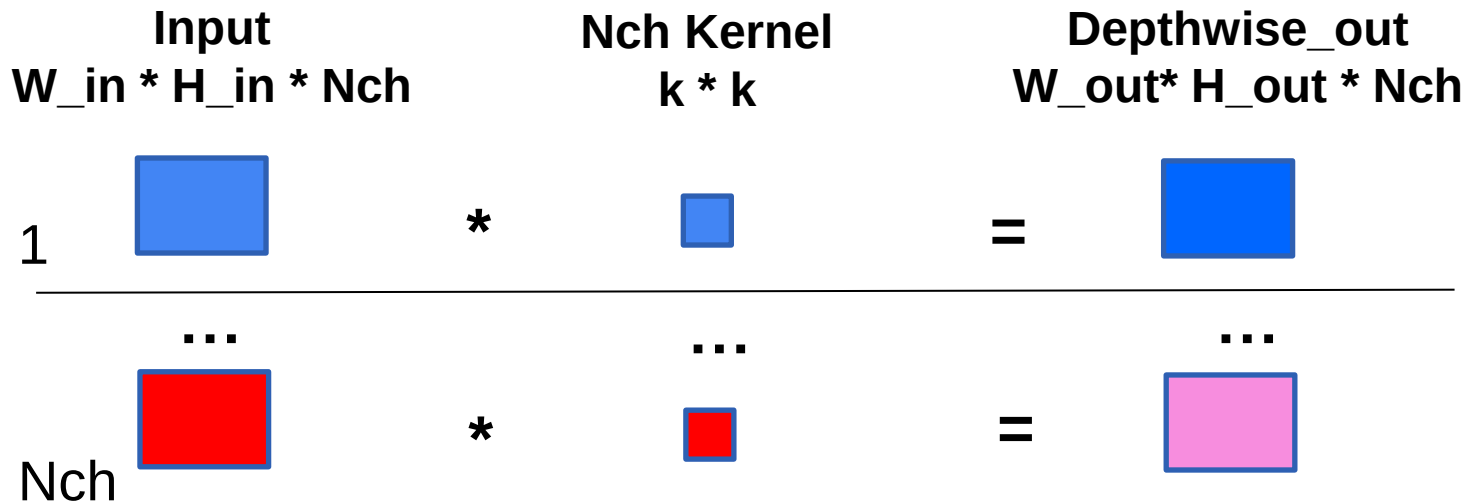
- **Purpose:** Reduce the amount of CONV computation
- **Input:** $W_{in} * H_{in} * N_{ch}$ (# of channels)
- **Kernel:** $k * k * N_k$ (# of kernels)
- **Output:** $W_{out} * H_{out} * N_k$ (# of kernels)





Depthwise Convolution

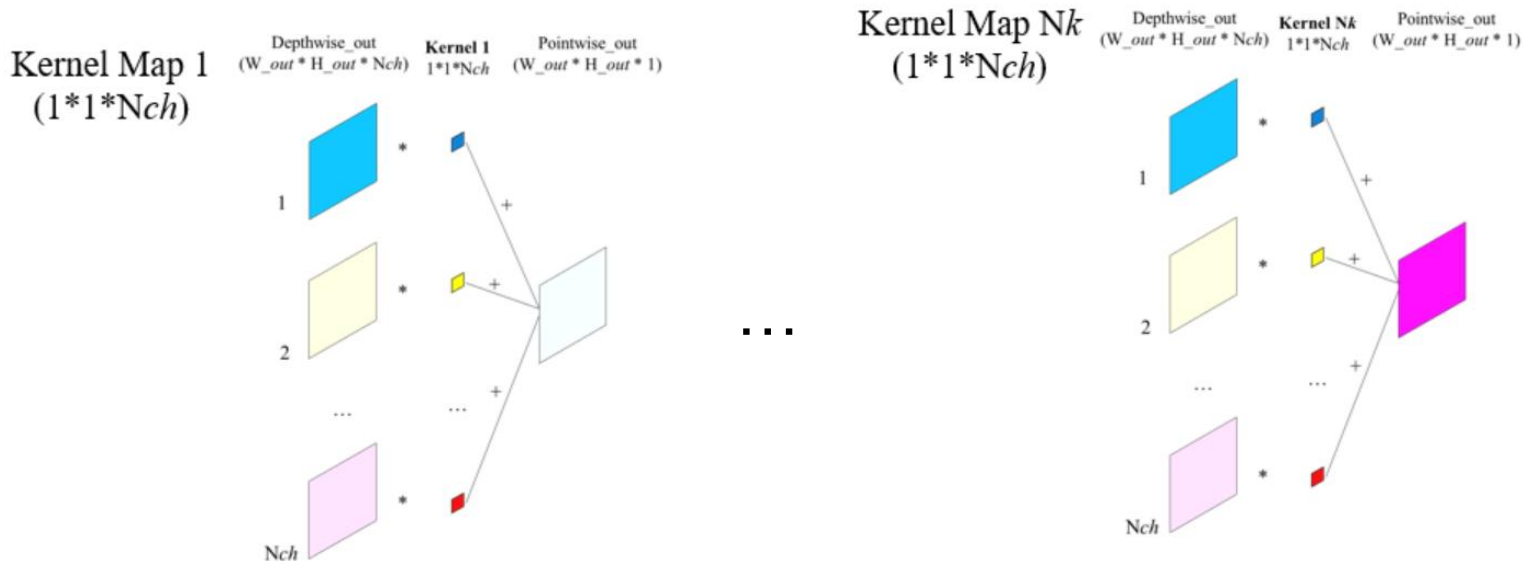
- Each channel of inputs has a $k * k$ kernel
- Separate the convolution of each channel
- **Difference:** Every kernel convolves with all channels in standard CONV





Pointwise Convolution

- The number of kernel: N_k with $(1 * 1 * N_{ch})$ size
- Do CONV on the outputs of depthwise convolution



Pointwise convolution



Depthwise + Pointwise Convolution

Depthwise convolution

Input: $W_{in} * H_{in} * N_{ch}$

N_{ch} Kernel ($k * k$)

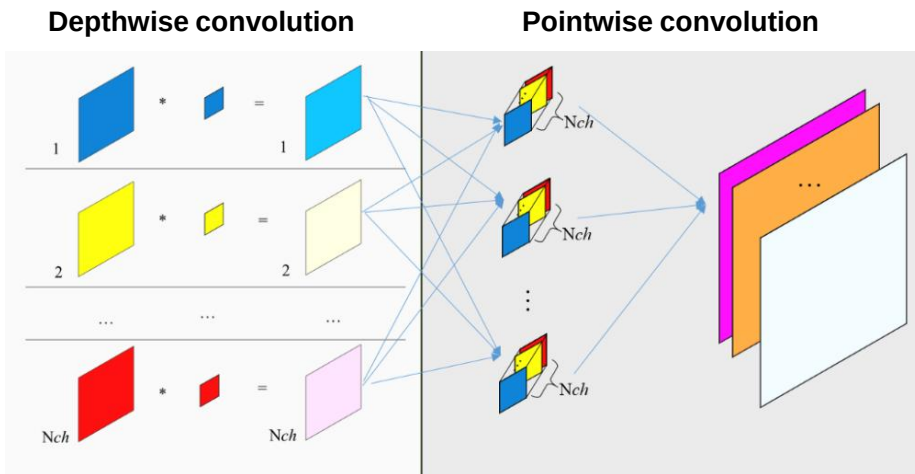
Output: $W_{out} * H_{out} * N_{ch}$

Pointwise convolution

Input: $W_{out} * H_{out} * N_{ch}$

N_k kernel = ($1 * 1 * N_k$)

Output = $W_{out} * H_{out} * N_k$





Depthwise Separable Convolution

- **Standard CONV**

- Input: $W_{in} * H_{in} * N_{ch}$
- Kernel: $k * k * N_k$
- Output: $W_{out} * H_{out} * N_k$
- Computation: $W_{in} * H_{in} * N_{ch} * k * k * N_k$

- **Depthwise separable convolution**

- Depthwise CONV computation: $W_{in} * H_{in} * N_{ch} * k * k$
- Pointwise CONV computation: $N_{ch} * N_k * W_{in} * H_{in}$

$$\frac{\text{Depthwise separable CONV}}{\text{Standard CONV}} = \frac{W_{in} * H_{in} * N_{ch} * k * k + N_{ch} * N_k * W_{in} * H_{in}}{W_{in} * H_{in} * N_{ch} * k * k * N_k}$$
$$= \frac{1}{N_k} + \frac{1}{K * k}$$



Depthwise Separable Convolution

- Depthwise separable convolution can save more computation when
 - kernel size is large
 - The number of kernel is increased
- Suppose input is $416 * 416 * 50$, # of filter is 10, its size is $3 * 3$.
- How much computation can be saved by depthwise separable convolution ?
 - $1/10 + 1/9 = 0.22$



Takeaway Questions

- What are problems in ultra-deep neural networks ?
 - (A) Over-fitting
 - (B) Gradient vanishing
 - (C) Low training accuracy
- Given a CNN model below, how many channels are in the second layer ?
 - (A) 4
 - (B) 8
 - (C) 16

	Input size	# of filter	Filter size	# of channel
Layer1	12 x 12	4	3x3	64
Layer2	12 x 12	16	3x3	



Takeaway Questions

- A standard CONV layer
 - Input: $W_{in} * H_{in} * N_{ch} = (32 * 32 * 16)$
 - Kernel: $k * k * N_k = (3 * 3 * 8)$
 - Computation: $W_{in} * H_{in} * N_{ch} * k * k * N_k = (32 * 32 * 16 * 3 * 3 * 8)$
- What is the amount of computation that is carried out by **depthwise separable convolution** ?
 - (A) $(32 * 32 * 16 * 3 * 3) + (3 * 8 * 8)$
 - (B) $(32 * 32 * 3 * 3 * 8) + (16 * 3 * 3)$
 - (C) $(32 * 32 * 16 * 3 * 3) + (16 * 8 * 32 * 32)$