6.5930/1
Hardware Architectures for Deep Learning

Memory and Evaluation Metrics

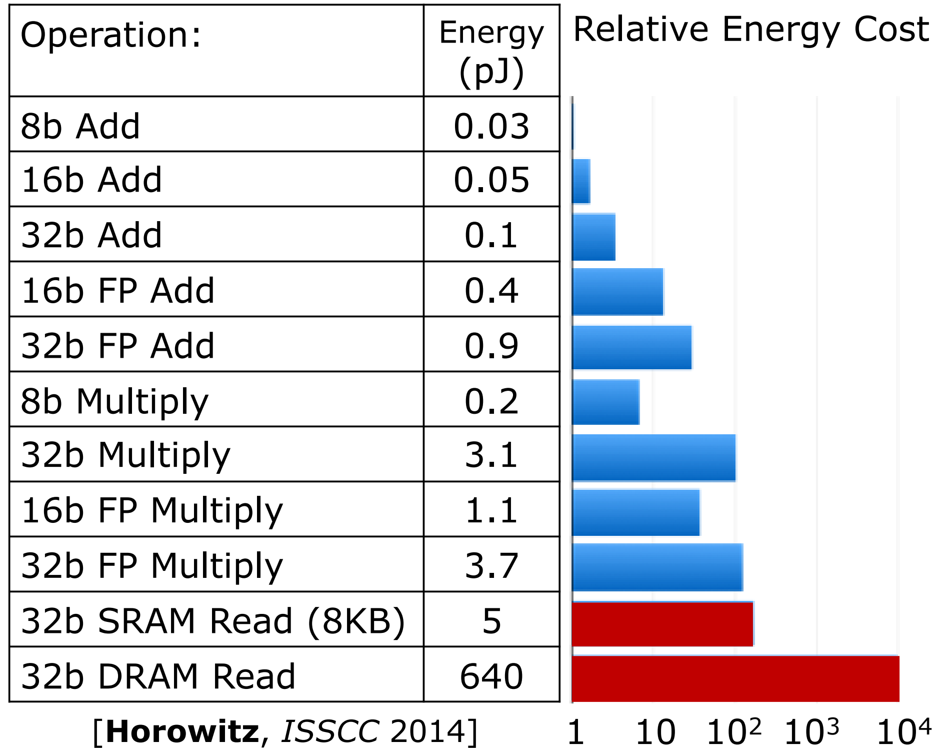
February 9, 2026

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Memories

Energy Consumption Dominated by Data Movement



Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Multiply	0.2
32b Multiply	3.1
16b FP Multiply	1.1
32b FP Multiply	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

[**Horowitz**, *ISSCC* 2014]

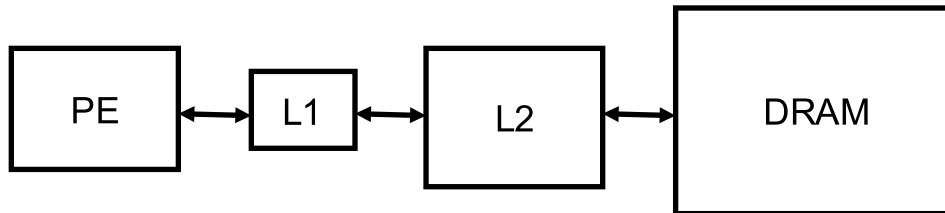
Energy = ***Capacitance*** *x Voltage*^{2}

Memory access is **orders of magnitude** higher energy than compute

Note that speed is often limited by data movement as well

Memory Hierarchy

- Memories cannot be fast **and** large
- Goal: Reduce access to large memories
  - Build hierarchy with smaller memories (local buffers) closer to compute (e.g., PE)
  - Design processing order to exploit data reuse via spatial and temporal locality to reduce access from large memories **(Section 5.2 in book)**
    - Size of memory limits amount of reuse that can be exploited (use processing order to reduce reuse distance and increase reuse)



Overview of Memories

Memory consist of arrays of cells that hold a value.

- Types of Memories/Storage
  - Latches/Flip Flops (Registers)
  - SRAM (Register File, Caches)
  - DRAM (Main Memory)
  - Flash (Storage)

Elements of Memory Operation

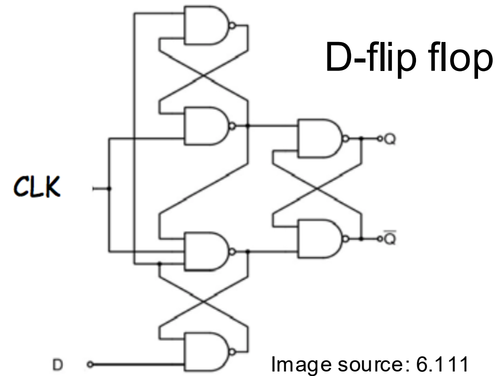
Implementations vary based on:

- How a memory cell holds a value?
- How to read a value from a memory cell?
- How to write a value to a memory cell?
- How is array constructed out of individual cells?

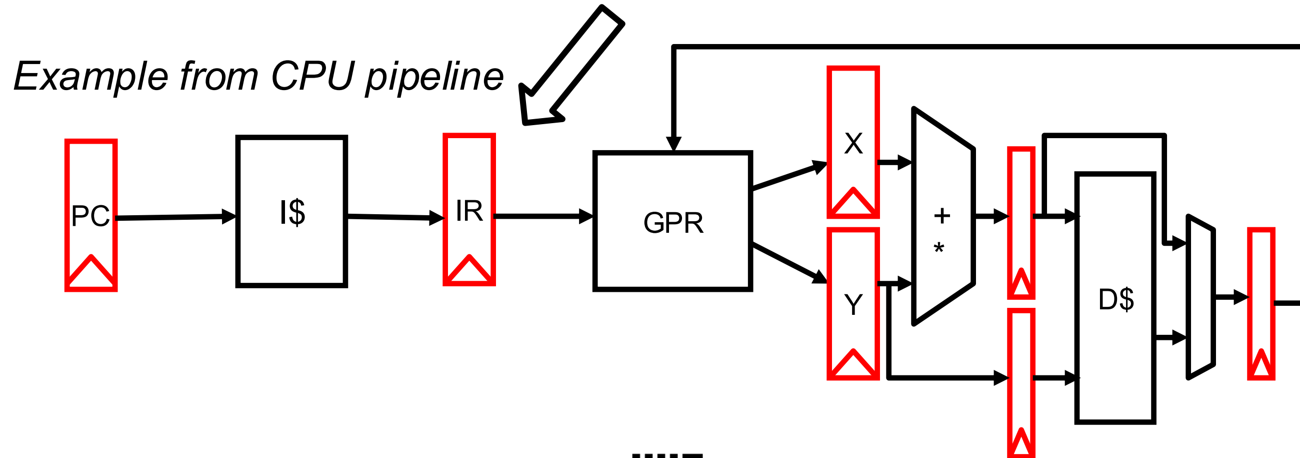
- Results in tradeoffs between cost, density, speed, energy and power consumption

Latches/Flip Flops

- Fast and low latency
- Located with logic



Example from CPU pipeline



Latches/Flip Flops (< 0.5 kB)

- Fast and low latency
- Located with logic
- Not very dense
  - 10+ transistors per bit
  - Usually use for arrays smaller than 0.5kB

D-flip flop

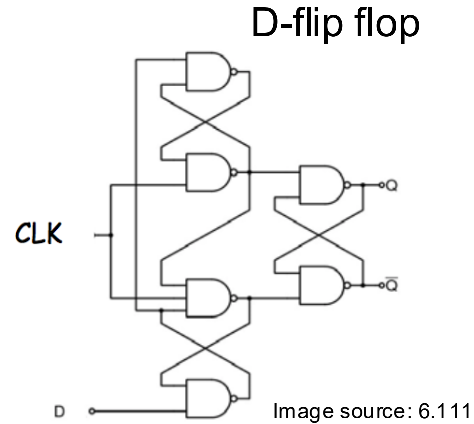
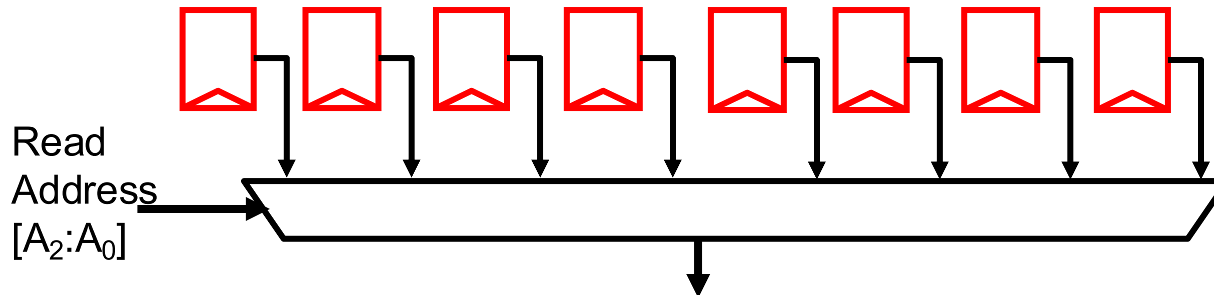
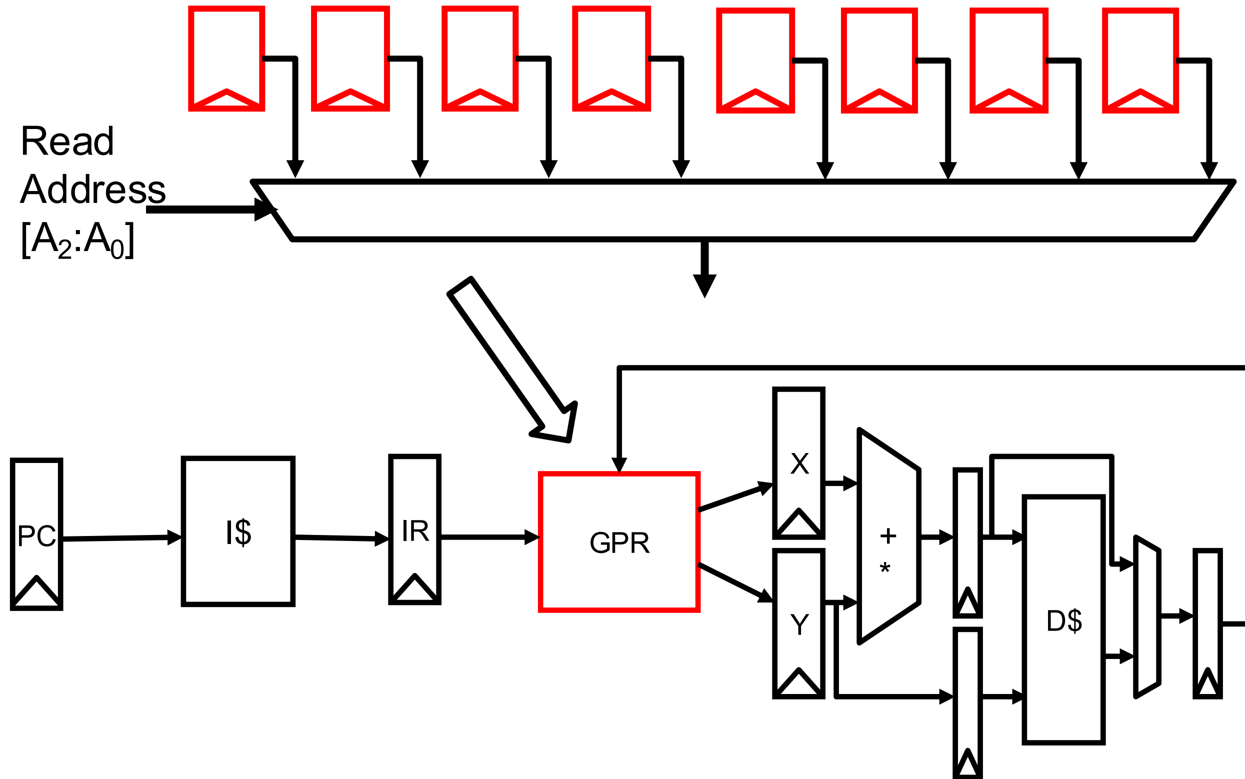


Image source: 6.111

Array of Flip flops

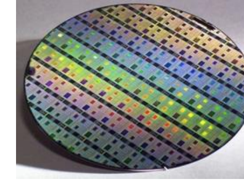


Latches/Flip Flops (< 0.5 kB)

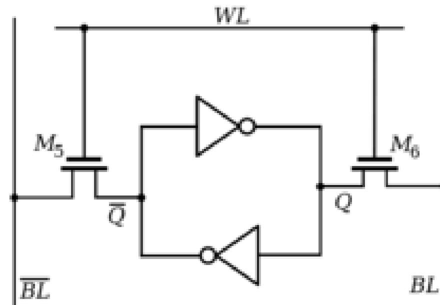


SRAM

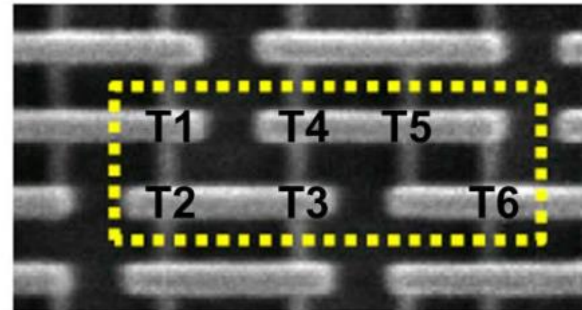
IC wafer



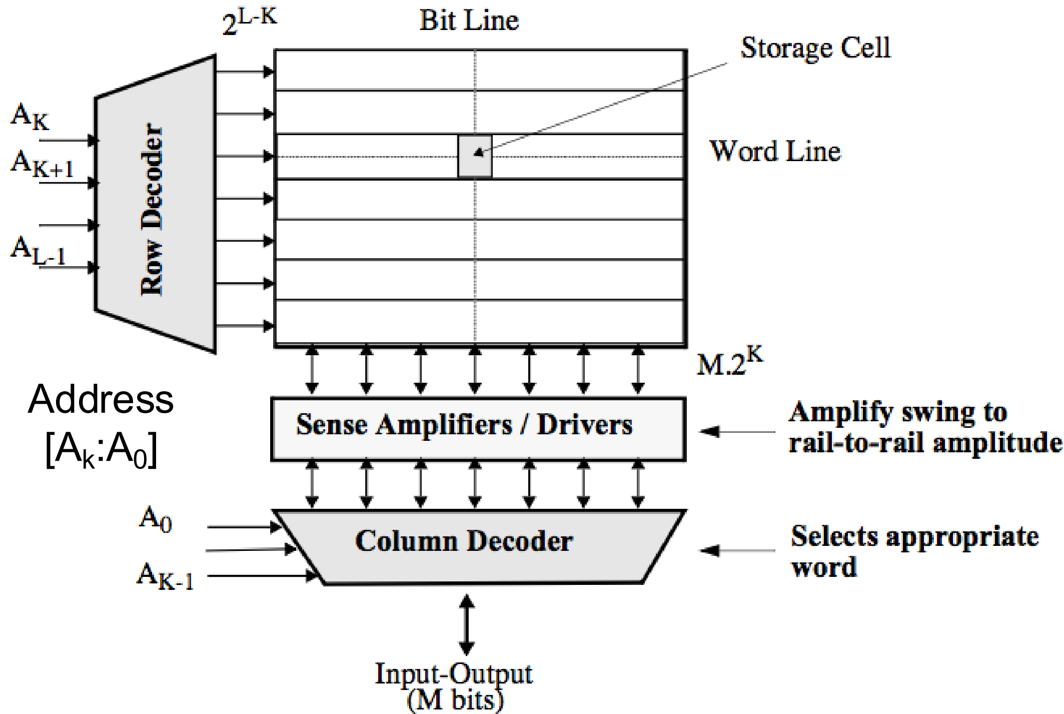
- Higher density than register
  - Usually, 6 transistors per bit-cell
- Less robust and slower than latches/flip-flop



Bit cell size 0.75um^2 in 14nm



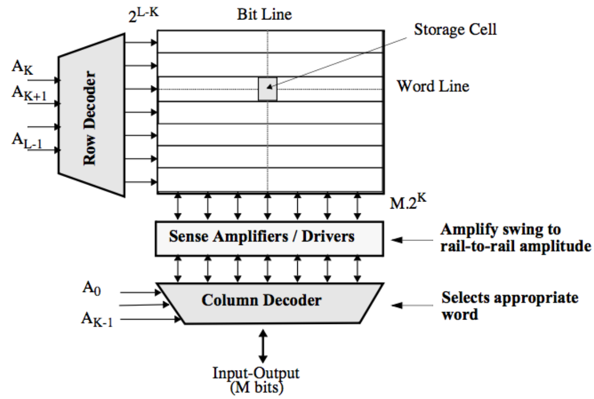
SRAM (kB – MB)

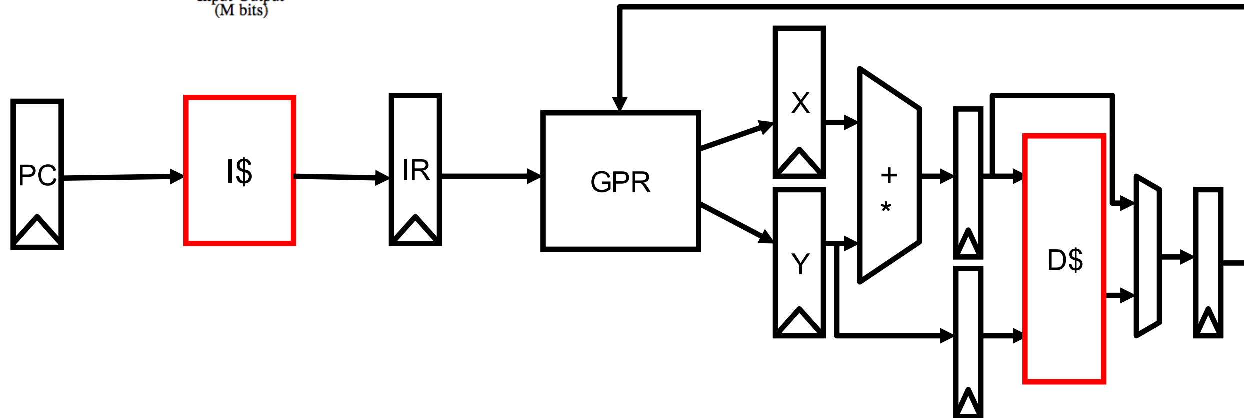


Note:

- Peripheral circuits can account for significant area and energy.
- Reduce peripheral cost overhead with larger memories and packing multiple values in single address.
- Efficient if access (traversal) order matches storage order (data layout)

SRAM





SRAM Power Dominated by Bit Line

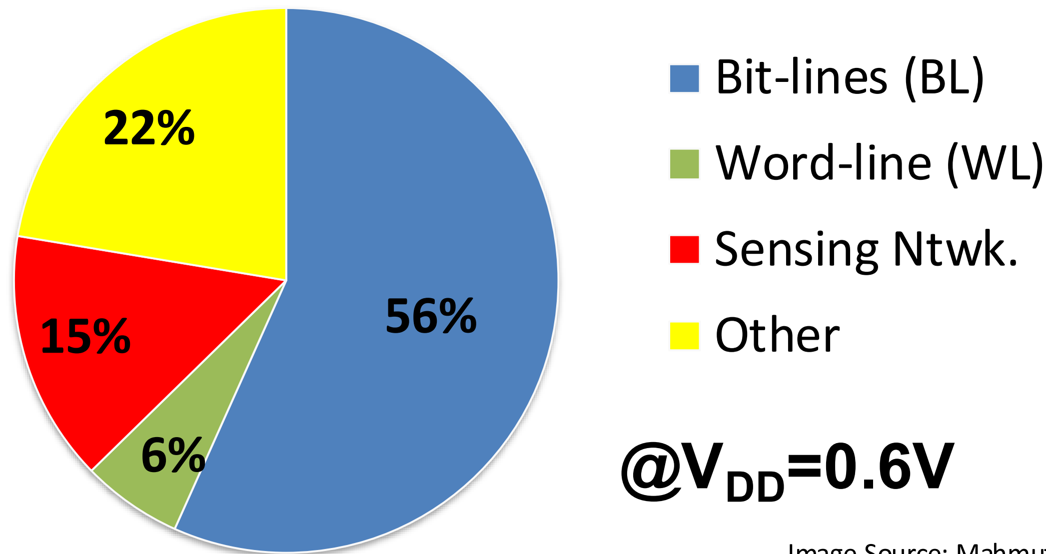
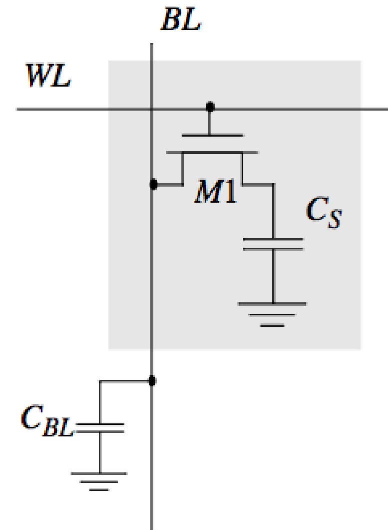


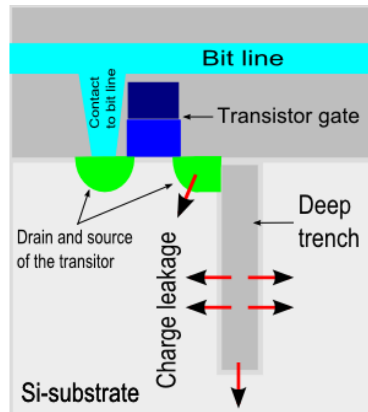
Image Source: Mahmut Sinangil

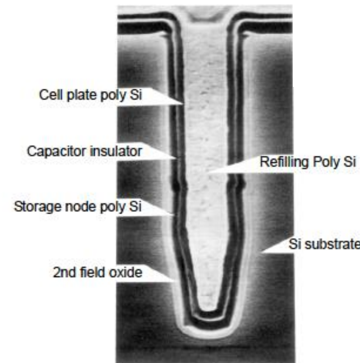
Larger array → Longer bit-lines
→ Higher capacitance → Higher power

DRAM

- Higher density than SRAM
  - 1 transistor per bit-cell
  - Needs periodic refresh
- Special device process

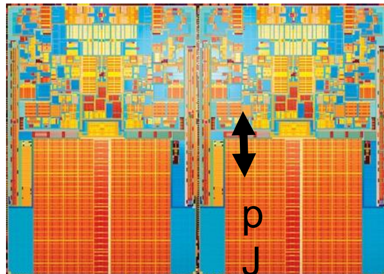


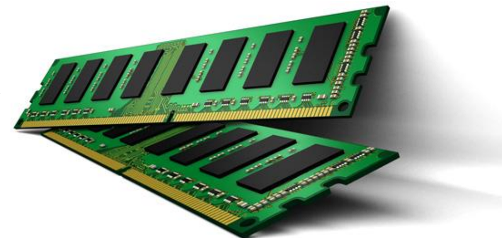




DRAM (GB)

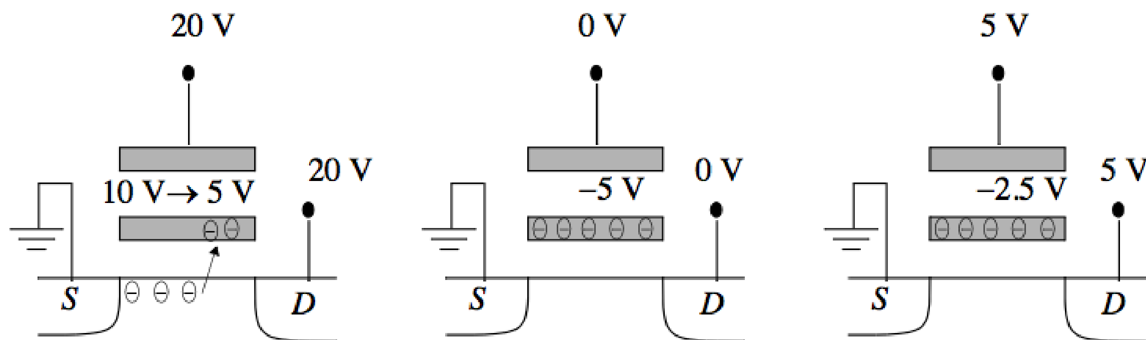
- Higher density than SRAM
  - 1 transistor per bit-cell
  - Needs periodic refresh
- Special device process
  - Usually off-chip (except eDRAM – which is pricey!)
  - Off-chip interconnect has much higher capacitance





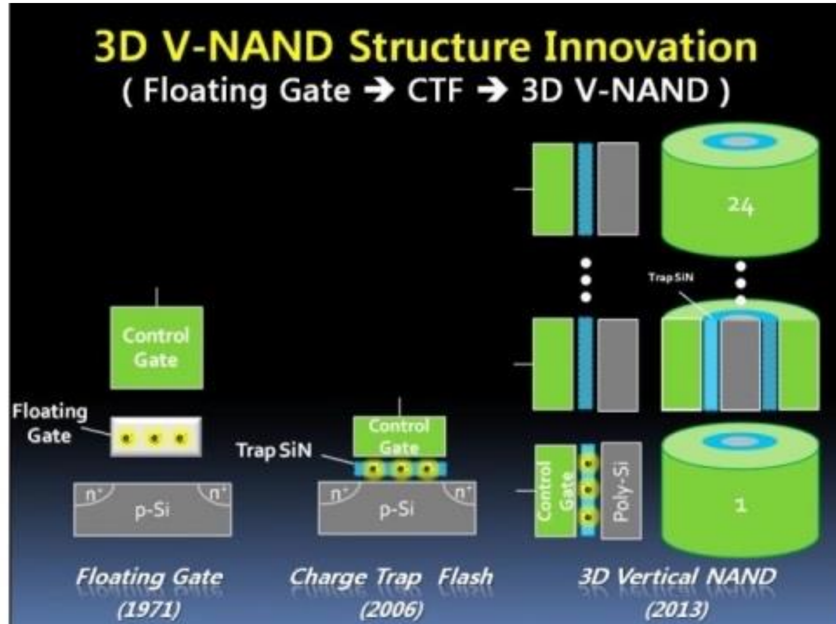
Flash (100GB to TB)

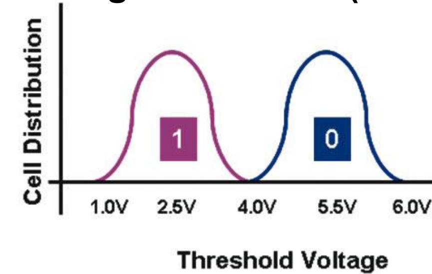
- More dense than DRAM
- Non-volatile
  - Needs high powered write (change V_{TH} of transistor)

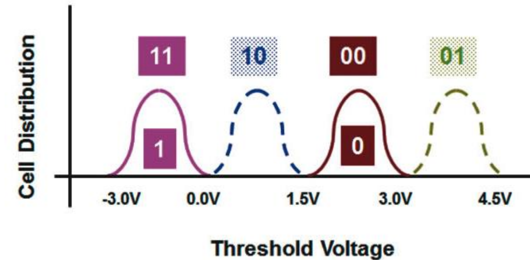


(a) Avalanche injection. (b) Removing programming voltage leaves charge trapped. (c) Programming results in higher V_T.

Flash Memory







Single Level Cell (SLC)

Multi-levels cell (MLC)

48 layer, Ternary level cell (TLC)
Aug 2015
256 Gb per die
(for SSD)

"For our 6th Generation of V-NAND, we have introduced a 3-bit, 100+ layer, single stack, 256-gigabit (and now a 512Gb) chip." Samsung (2022)
https://semiconductor.samsung.com/us/newsroom/tech-blog/the-next-phase-of-v-nand-ingenuity/

Memory Tradeoffs

Cost/bit
Function of circuit type (smaller → cheaper)

Latency
Function of circuit type
(smaller → slower)
and total capacity
(smaller → faster)

Energy/access/bit
Function of total capacity
(smaller → less energy)
and circuit type
(smaller → less energy)

Bandwidth
Increases with
parallelism

Density
Function of circuit type
(smaller → denser)

Most attributes tend to improve with technology scaling, lower voltage and sometimes smaller capacitors

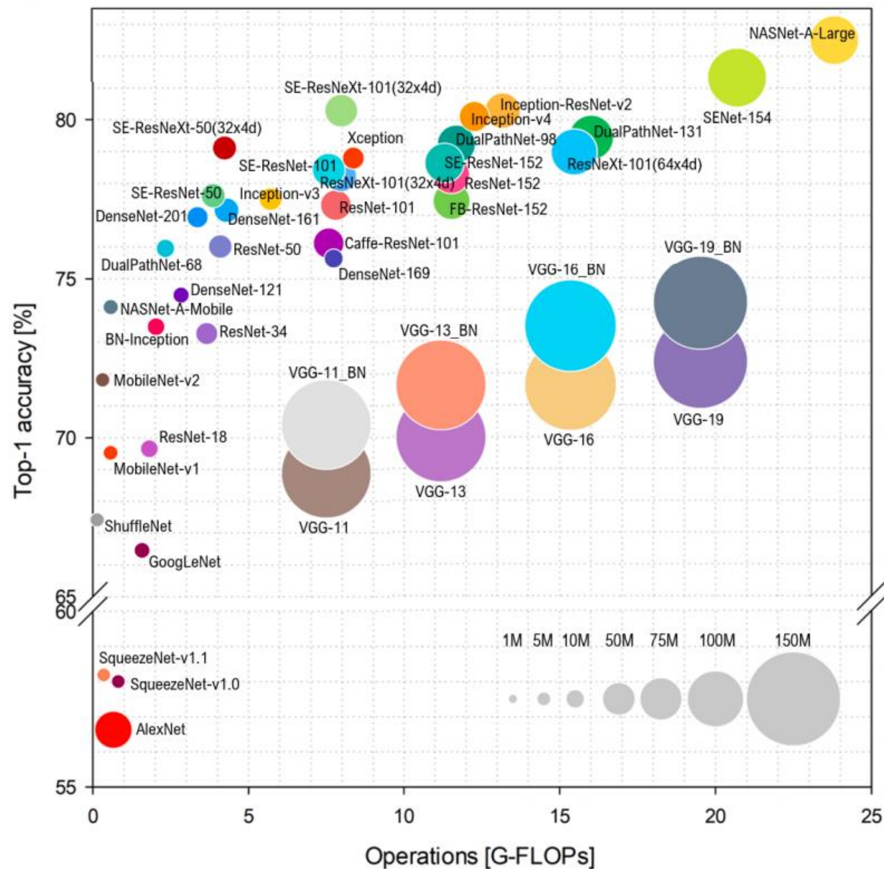
Summary

- Reduce memory access to larger memory with small local buffers
  - Large memory (i.e., DRAM) is slow and has high energy consumption
  - Exploits spatial and temporal locality → increase compute intensity
- Since processing order does not affect result (MACs are commutative), change processing order (loop nest) to improve spatial and temporal locality
- Tradeoffs in storage technology
  - Various tradeoffs in cost, speed, energy, capacity…
  - Different technologies appropriate at different spots in the design

Efficient CNN Models

For more info, refer to chapter 9 in https://doi.org/10.1007/978-3-031-01766-7

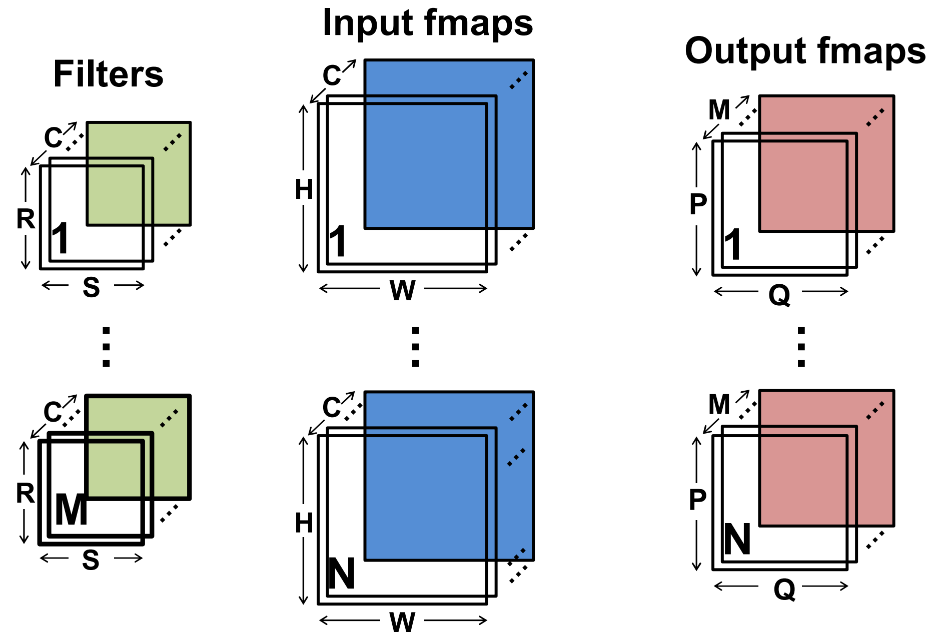
Accuracy vs. Weight & OPs



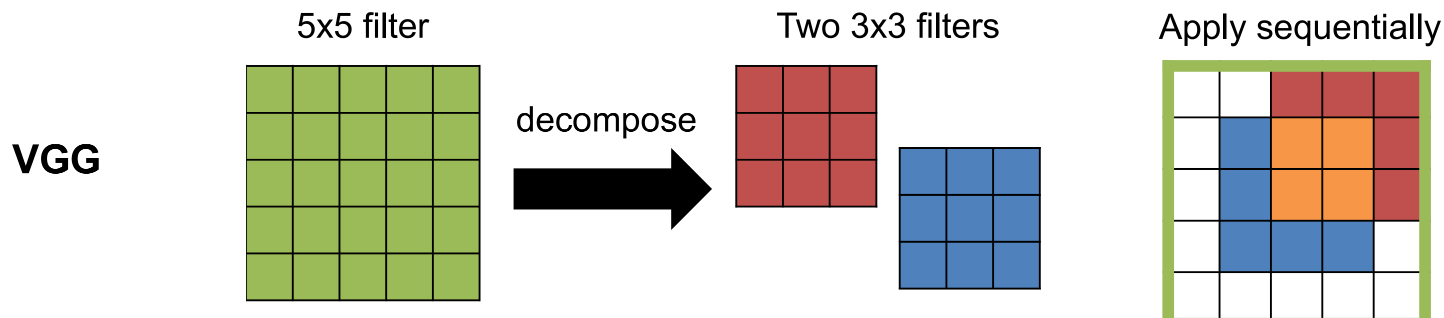
[**Bianco**, *IEEE Access* 2018]

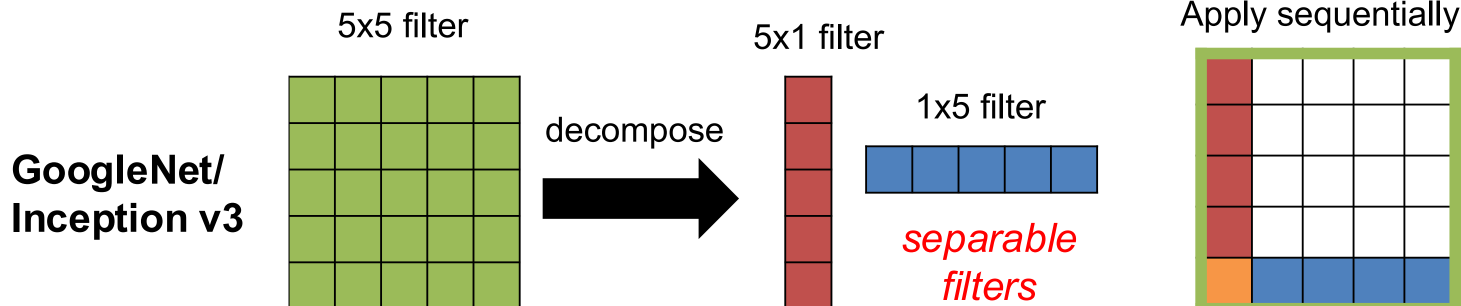
Manual Network Design

- **Reduce Spatial Size (R, S)**
  - stacked filters
- **Reduce Channels (C)**
  - 1x1 convolution, grouped convolution
- **Reduce Filters (M)**
  - feature map reuse across layers



Reduce Spatial Size (R, S): Stacked Small Filters

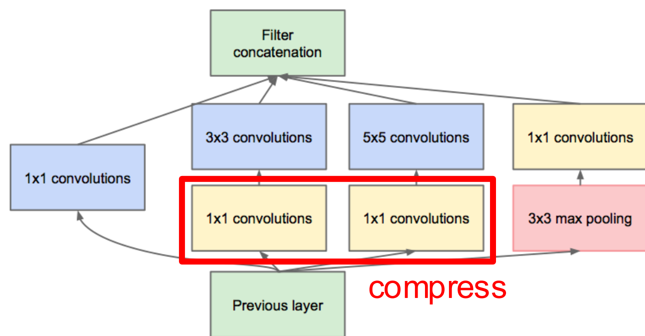




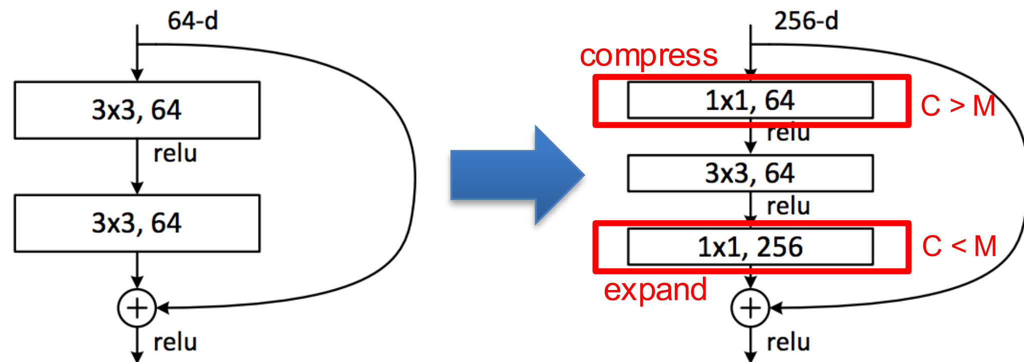
Replace a large filter with **a series of smaller filters** (reduces degrees of freedom)

Reduce Channels (C): 1x1 Convolution

GoogLeNet



ResNet



- Use **1x1 (bottleneck) filter** to capture cross-channel correlation, but not spatial correlation
- Reduce the number of channels in next layer (**compress**), where **C** > **M**

Reduce Channels (C): Grouped Convolutions

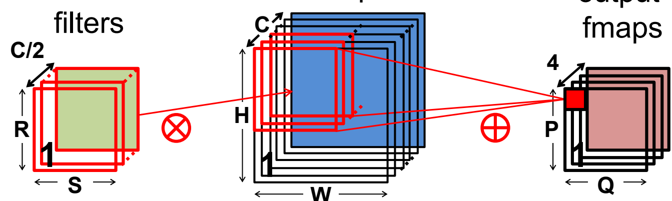
Grouped convolutions reduce the number of **weights and multiplications** at the cost of not sharing information between **groups**

- **Divide** filters into groups (**G**) operating on **subset** of channels.
- Each group has **M/G** filters and processes **C/G** channels.

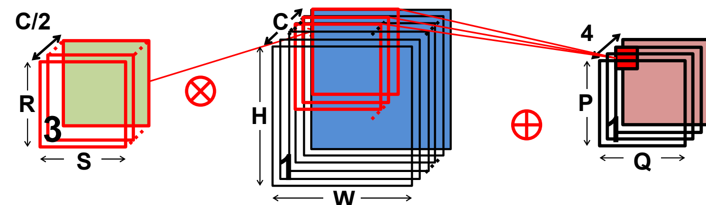
Example for **G=2**: Each filter requires **2x fewer weights and MACs (C → C/2)**

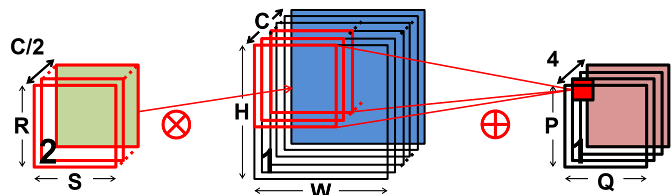
In this example, N=1 & M=4

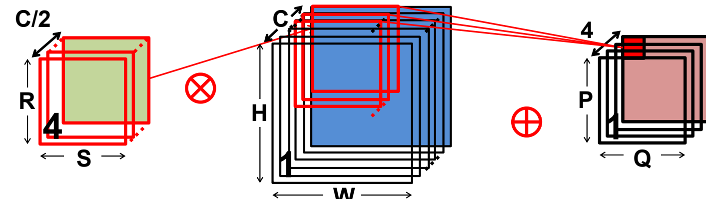
Group 1



Group 2

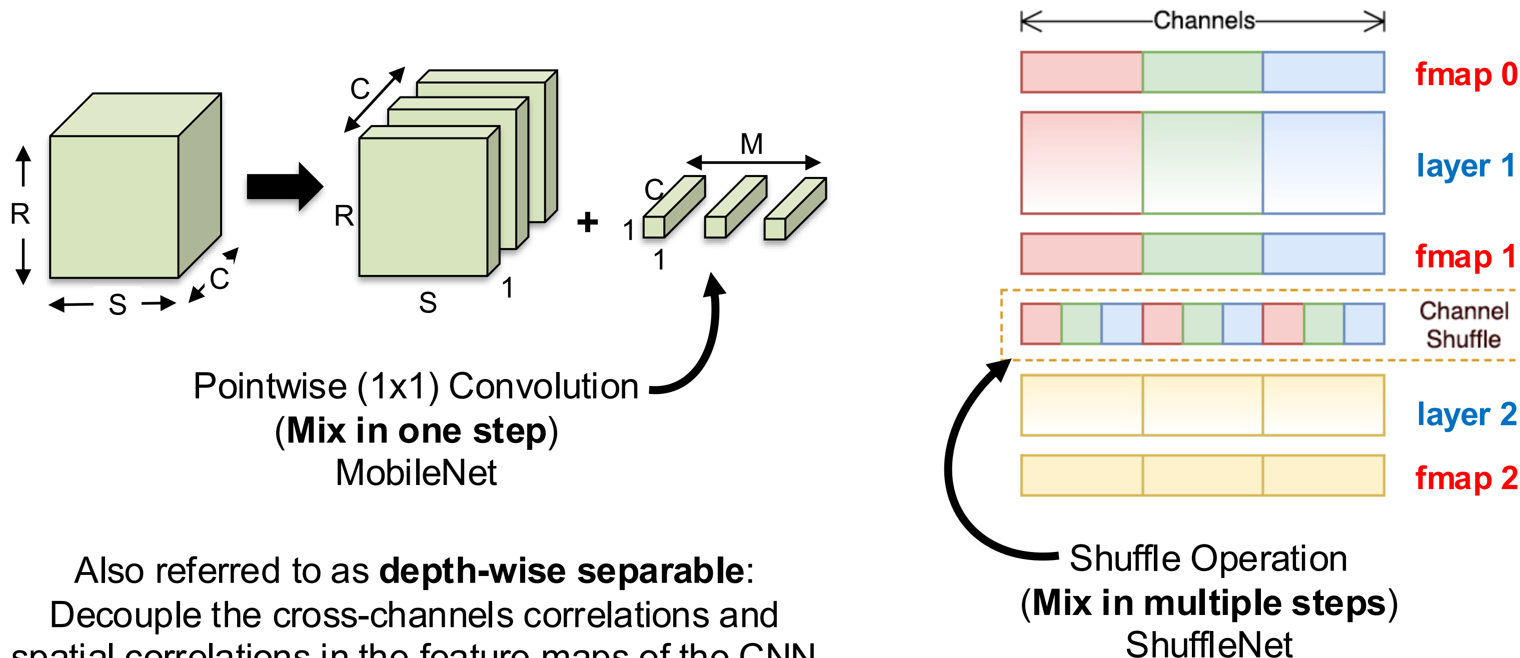






Reduce Channels (C): Grouped Convolutions

Two ways of mixing information from groups



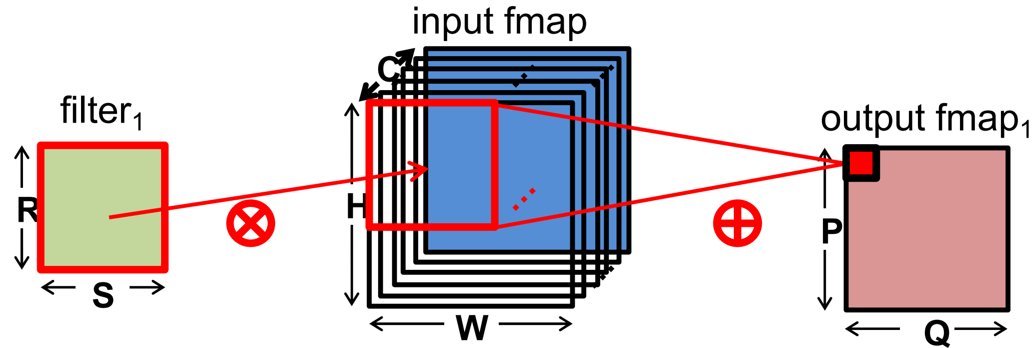
Also referred to as **depth-wise separable**: Decouple the cross-channels correlations and spatial correlations in the feature maps of the CNN

Depth-wise Convolutions

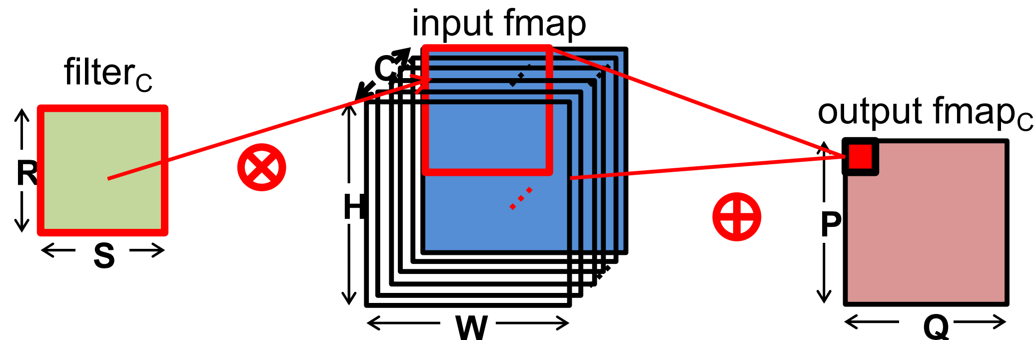
The extreme case of **Grouped Convolutions** is **Depth-wise Convolutions,** where the **number of groups (G) equals number channels (C)** (i.e., one input channel per group)

Typically, M=C
(but does not have to be)

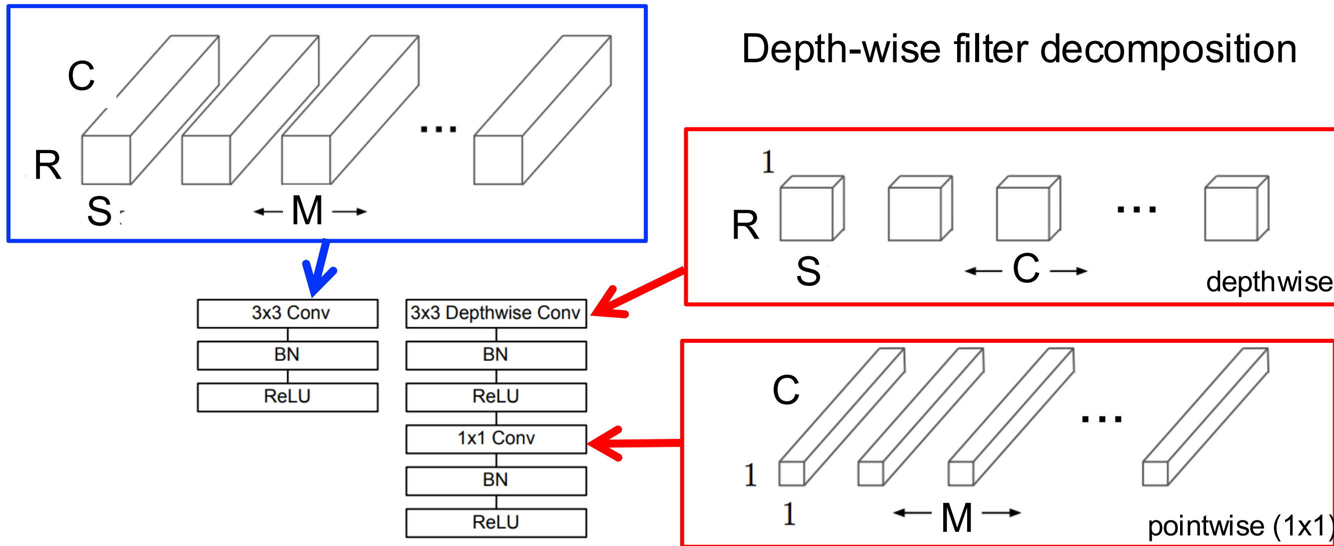
Group 1



Group C



Example: MobileNets



Reduction in MACs

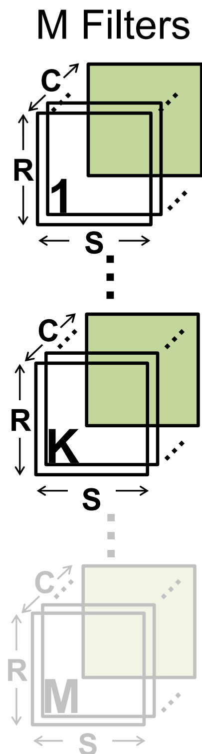
$$\frac{HWCRSM}{HWC(RS+M)} = \frac{RSM}{(RS+M)}$$

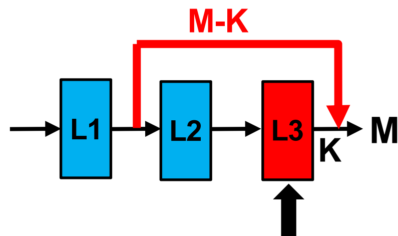
Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

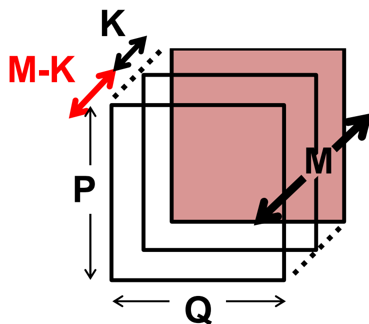
[**Howard**, *arXiv* 2017]

Reduce Filters (M): Feature Map Reuse





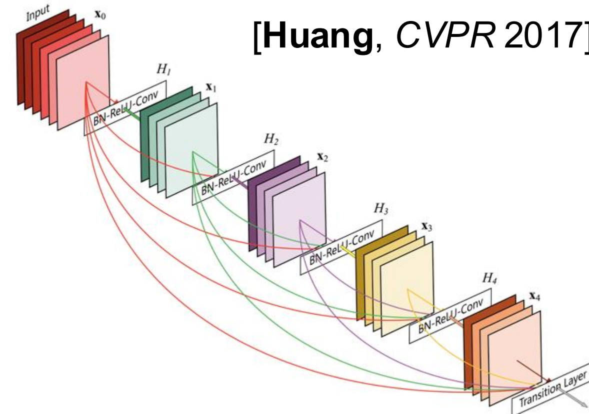
Output fmap with M channels



Reuse **(M-K)** channels in feature maps from previously processed layers

DenseNet reuses feature map from multiple layers

[**Huang**, *CVPR* 2017]

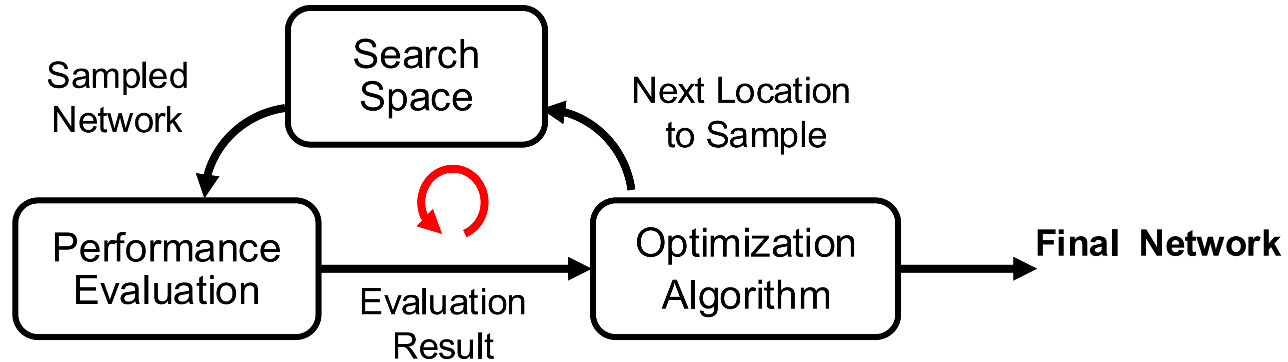


Neural Architecture Search (NAS)

Rather than handcrafting the model, automatically search for it

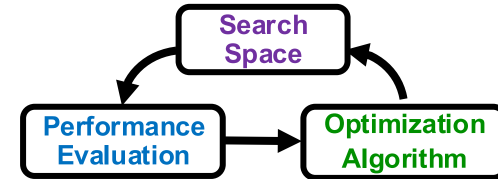
Neural Architecture Search (NAS)

- Three main components:
  - Search Space (what is the set of all samples)
  - Optimization Algorithm (where to sample)
  - Performance Evaluation (how to evaluate samples)

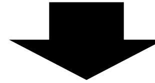


Key Metrics: Achievable DNN **accuracy** and required **search time**

Evaluate NAS Search Time



$$time_{nas} = num_{samples} \times time_{sample}$$



$$time_{nas} \propto \left(size_{search_space} \times \frac{num_{alg_tuning}}{efficiency_{alg}} \right) \times (time_{eval} + time_{train})$$

(1) Shrink the search space

(2) Improve the optimization algorithm

(3) Simplify the performance evaluation

Goal: Improve the efficiency of NAS in the three main components

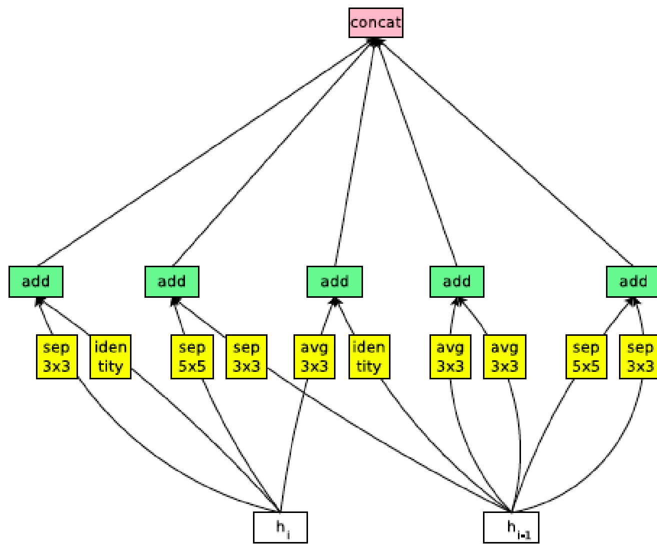
Example: NASNet

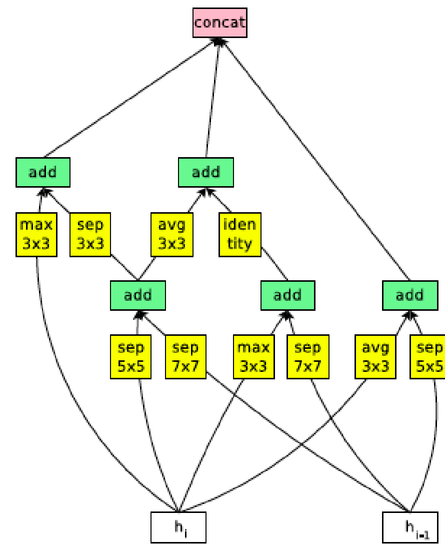
- Search Space: Build model from popular layers

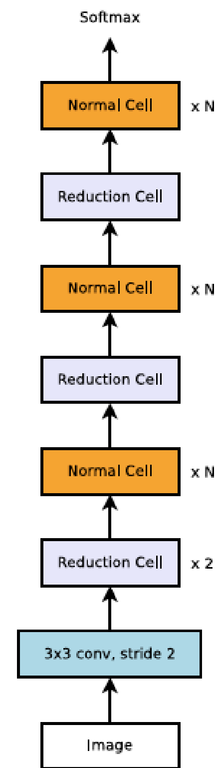
  - Identity
  - 1x3 then 3x1 convolution
  - 1x7 then 7x1 convolution
  - 3x3 dilated convolution
  - 1x1 convolution
  - 3x3 convolution
  - 3x3 separable convolution
  - 5x5 separable convolution
  - 3x3 average pooling
  - 3x3 max pooling
  - 5x5 max pooling
  - 7x7 max pooling

[**Zoph**, *CVPR* 2018]

NASNet: Learned Convolutional Cells



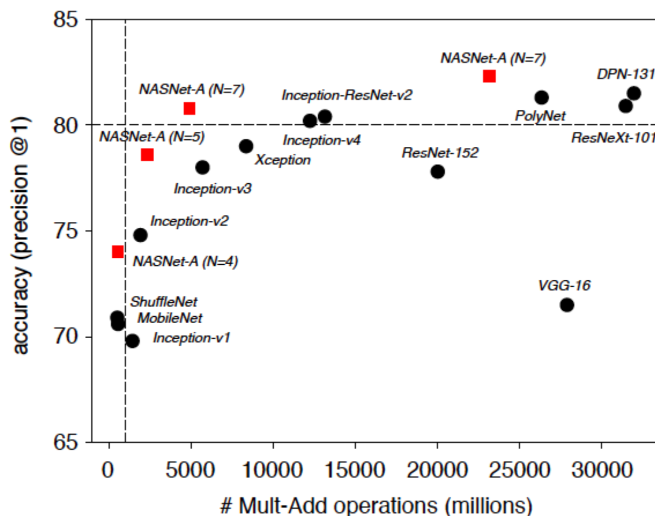


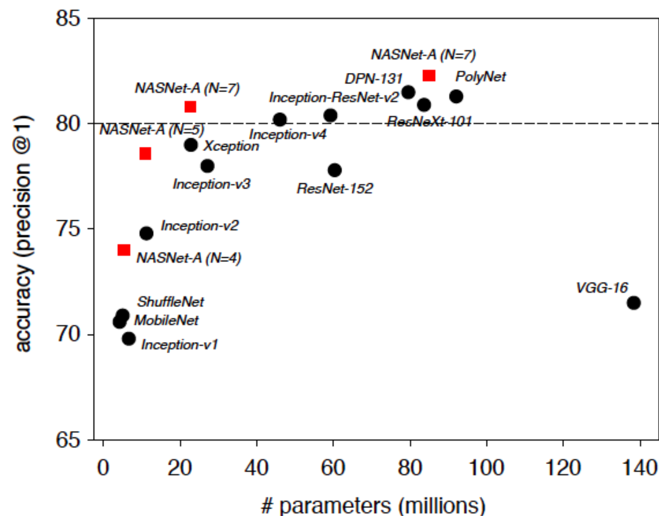


[**Zoph**, *CVPR* 2018]

NASNet: Comparison with Existing Networks

Learned models have improved accuracy vs. 'complexity' tradeoff compared to handcrafted models





[**Zoph**, *CVPR* 2018]

Summary

- Approaches used to improve accuracy by popular CNN models in the ImageNet Challenge
  - Go deeper (i.e., more layers)
  - Stack smaller filters and apply 1x1 bottlenecks to reduce number of weights such that the deeper models can fit into a GPU (faster training)
  - Use multiple connections across layers (e.g., parallel and short cut)
- Efficient models aim to reduce number of weights and number of operations
  - Most use some form of filter decomposition (spatial, depth and channel)
  - Note: Number of weights and operations does not directly map to storage, speed and power/energy. Depends on hardware! *What is impact on compute intensity?*
- Filter shapes vary across layers and models
  - Need flexible hardware!

Warning!

- These works often use **number of weights and operations** to measure “**complexity**”
- Number of weights provides an indication of **storage cost** for inference
- However later in the course, we will see that
  - Number of operations doesn’t directly translate to latency/throughput
  - Number of weights and operations doesn’t directly translate to power/energy consumption
- Understanding the underlying hardware is important for evaluating the impact of these “efficient” CNN models

References

- Book: Chapter 2 & 9
  – https://doi.org/10.1007/978-3-031-01766-7
- Other Works Cited in Lecture (increase accuracy)

  – **LeNet:** LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proc. IEEE 1998.
  – **AlexNet:** Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NeurIPS. 2012.
  – **VGGNet:** Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR 2015.
  – **Network in Network:** Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." ICLR 2014
  – **GoogleNet:** Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR 2015.
  – **ResNet:** He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR 2016.
  – **DenseNet:** Huang, Gao, et al. "Densely connected convolutional networks." CVPR 2017
  – **Wide ResNet:** Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." BMVC 2017.
  – **ResNext:** Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." CVPR 2017
  – **SENets:** Hu, Jie et al., "Squeeze-and-Excitation Networks," CVPR 2018
  – **NFNet:** Brock, Andrew, et al., "High-Performance Large-Scale Image Recognition Without Normalization," arXiv 2021

References

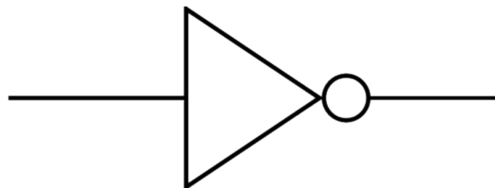
- Other Works Cited in Lecture (increase efficiency)
  - **InceptionV3:** Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." CVPR 2016.
  - **SqueezeNet:** Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size." ICLR 2017.
  - **Xception:** Chollet, François. "Xception: Deep Learning with Depthwise Separable Convolutions." CVPR 2017
  - **MobileNet:** Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
  - **MobileNetv2:** Sandler, Mark et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks," CVPR 2018
  - **MobileNetv3:** Howard, Andrew et al., "Searching for MobileNetV3," ICCV 2019
  - **ShuffleNet:** Zhang, Xiangyu, et al. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices." CVPR 2018
  - **Learning Network Architecture**: Zoph, Barret, et al. "Learning Transferable Architectures for Scalable Image Recognition." CVPR 2018
- Other Works Cited in Lecture (Increase accuracy and efficiency)
  - **EfficientNet:** Tan, Mingxing, et al. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019

Key Metrics and Design Objectives

How can we compare designs?

GOPS/W or TOPS/W?

- GOPS = giga (10^9) operations per second
- TOPS = tera (10^{12}) operations per second
- GOPS/Watt or TOPS/Watt commonly reported in hardware literature to show **efficiency** of design
- However, does not provide sufficient insights on hardware capabilities and limitations (especially if based on peak throughput/performance)



Example: high TOPS per watt can be achieved with inverter (ring oscillator)

Key Metrics: Much more than OPS/W!

- **Accuracy**
  - Quality of result
- **Throughput**
  - Analytics on high volume data
  - Real-time performance (e.g., video at 30 fps)
- **Latency**
  - For interactive applications (e.g., autonomous navigation)
- **Energy and Power**
  - Embedded devices have limited battery capacity
  - Data centers have a power ceiling due to cooling cost
- **Hardware Cost**
  - $$$
- **Flexibility**
  - Range of DNN models and tasks
- **Scalability**
  - Scaling of performance with amount of resources

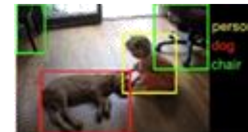
MNIST

ImageNet

Embedded Device

Data Center

Computer Vision



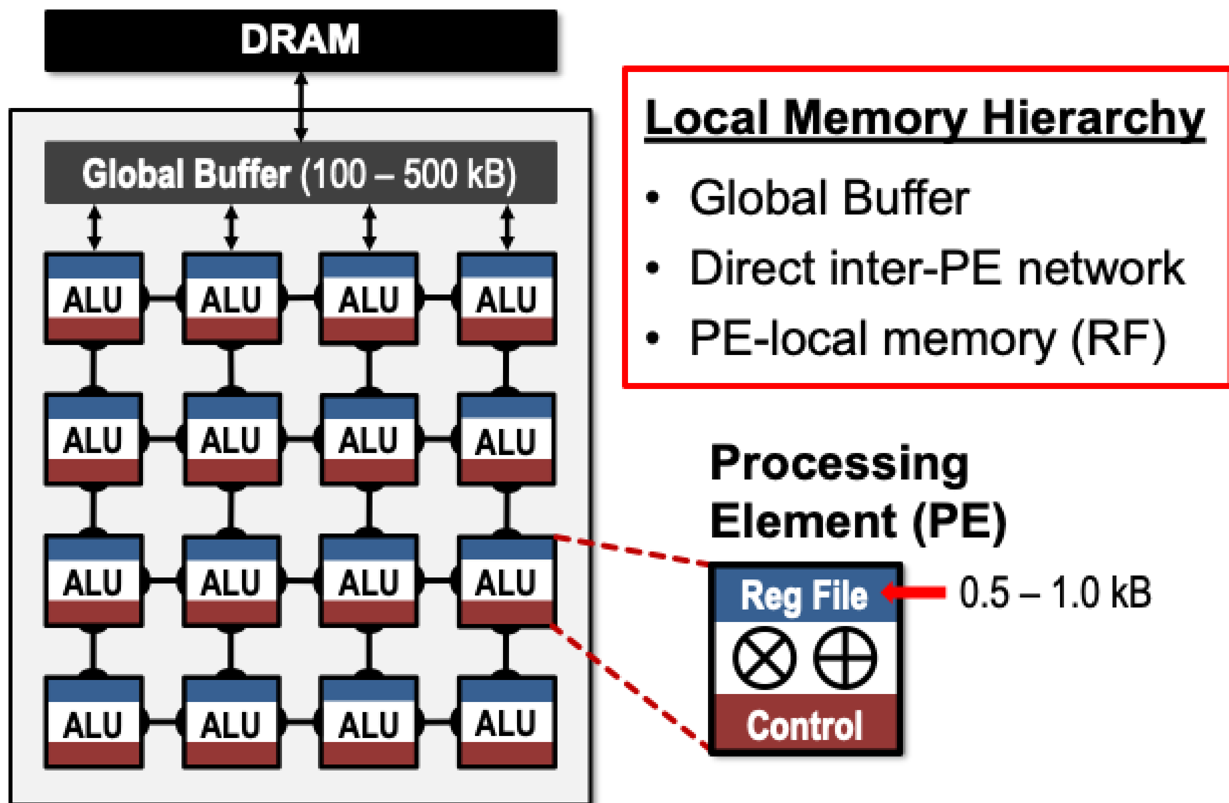
Speech Recognition

[**Sze**, *CICC* 2017]

Evaluating Accuracy

- Important to measure accuracy when considering co-design of algorithm and hardware
- Datasets help provide a way to evaluate and compare different DNN models and training algorithms
- All accuracy is not the same
  - Must consider difficulty of task and dataset to get fair comparison

Typical Architecture for DNN Accelerator



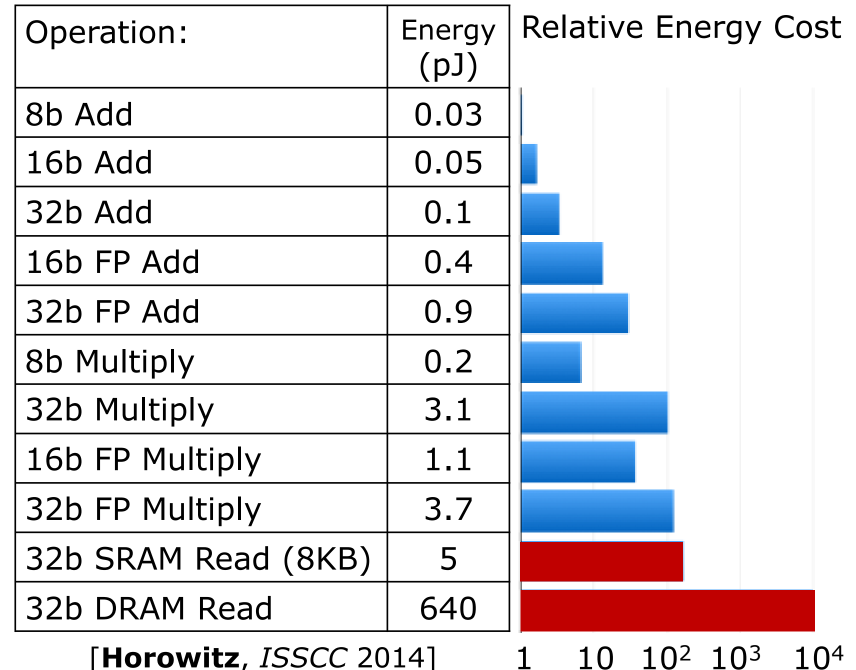
Key Design Objectives of DNN Processor

- **Increase Throughput and Reduce Latency**
  - Reduce time per MAC
    - Reduce critical path → increase clock frequency
    - Reduce instruction overhead
  - Avoid unnecessary MACs (save cycles)
  - Increase number of processing elements (PE) → more MACs in parallel
    - Increase area density of PE or area cost of system
  - Increase PE utilization* → keep PEs busy
    - Distribute workload to as many PEs as possible
    - Balance the workload across PEs
    - Sufficient memory bandwidth to deliver workload to PEs (reduce idle cycles)
- Low latency has an additional constraint of **small batch size**

*(100% = peak performance)

Key Design Objectives of DNN Processor

- **Reduce Energy and Power Consumption**
  - Reduce data movement as it dominates energy consumption
    - Exploit data reuse
  - Reduce energy per MAC
    - Reduce switching activity and/or capacitance
    - Reduce instruction overhead
  - Avoid unnecessary MACs
- Power consumption is limited by heat dissipation, which limits the maximum # of MACs in parallel (i.e., throughput)



Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Multiply	0.2
32b Multiply	3.1
16b FP Multiply	1.1
32b FP Multiply	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

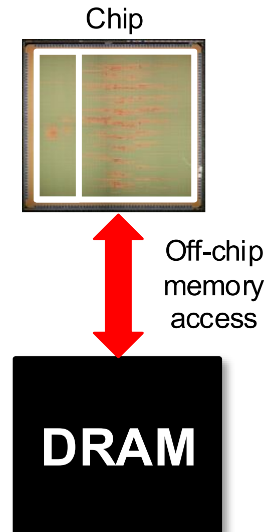
[**Horowitz**, *ISSCC* 2014]

Key Design Objectives of DNN Processor

- **Flexibility**
  - Reduce overhead of supporting flexibility
  - Maintain efficiency across wide range of DNN models
    - Different layer shapes impact the amount of
      - Required storage and compute
      - Available data reuse that can be exploited
    - Different precision across layers & data types (weight, activation, partial sum)
    - Different degrees of sparsity (number of zeros in weights or activations)
    - Types of DNN layers and computation beyond MACs (e.g., activation functions)

Specifications to Evaluate Metrics

- **Accuracy**
  - Difficulty of dataset and/or task should be considered
  - Difficult tasks typically require more complex DNN models
- **Throughput**
  - Number of PEs with utilization (not just peak performance)
  - Runtime for running specific DNN models
- **Latency**
  - Batch size used in evaluation
- **Energy and Power**
  - Power consumption for running specific DNN models
  - Off-chip memory access (e.g., DRAM)
- **Hardware Cost**
  - On-chip storage, # of PEs, chip area + process technology
- **Flexibility**
  - Report performance across a wide range of DNN models
  - Define range of DNN models that are efficiently supported



Evaluation Process

The evaluation process for whether a DNN system is a viable solution for a given application might go as follows:

1. **Accuracy** determines if it can perform the given task
2. **Latency and throughput** determine if it can run fast enough and in real-time
3. **Energy and power consumption** will primarily dictate the form factor of the device where the processing can operate
4. **Cost**, which is primarily dictated by the chip area and external interfaces, determines how much one would pay for this solution
5. **Flexibility** determines the range of tasks it can support

Comprehensive Coverage for Evaluation

- **All metrics** should be reported for fair evaluation of design tradeoffs
- Examples of what can happen if certain metric is omitted:
  - **Without the accuracy given for a specific dataset and task**, one could run a simple DNN and claim low power, high throughput, and low cost – however, the processor might not be usable for a meaningful task
  - **Without reporting the off-chip bandwidth**, one could build a processor with only multipliers and claim low cost, high throughput, high accuracy, and low chip power – however, when evaluating system power, the off-chip memory access would be substantial
- Are results measured or simulated? On what test data?
- Hardware should be evaluated on a wide range of DNNs
  - No guarantee that DNN algorithm designer will use a given DNN model or given reduce complexity approach. **Need flexible hardware!**

MLPerf: Workloads for Benchmarking

23 Companies
7 Institutions

First results in Dec 2018

https://mlperf.org/

A broad ML benchmark suite for measuring performance of ML software frameworks, ML hardware accelerators, and ML cloud platforms.

- A broad suite of DNN models to serve as a common set of benchmarks to measure the performance and enable fair comparison of various software frameworks, hardware accelerators, and cloud platforms for both training and inference of DNNs. **(edge compute in the works!)**
- The suite includes a wide range of DNNs (e.g., CNN, RNN, etc.) for a variety of tasks include image classification, object identification, translation, speech-to-text, recommendation, sentiment analysis and reinforcement learning.
- Categories: cloud/edge; training/inference; closed/open

Weights & MACs → Energy & Latency

- ***Warning:*** *Fewer weights and MACs (indirect metrics) do not necessarily result in lower energy consumption or latency (direct metrics). Other factors also important such as filter shape, batch size and hardware mapping.*

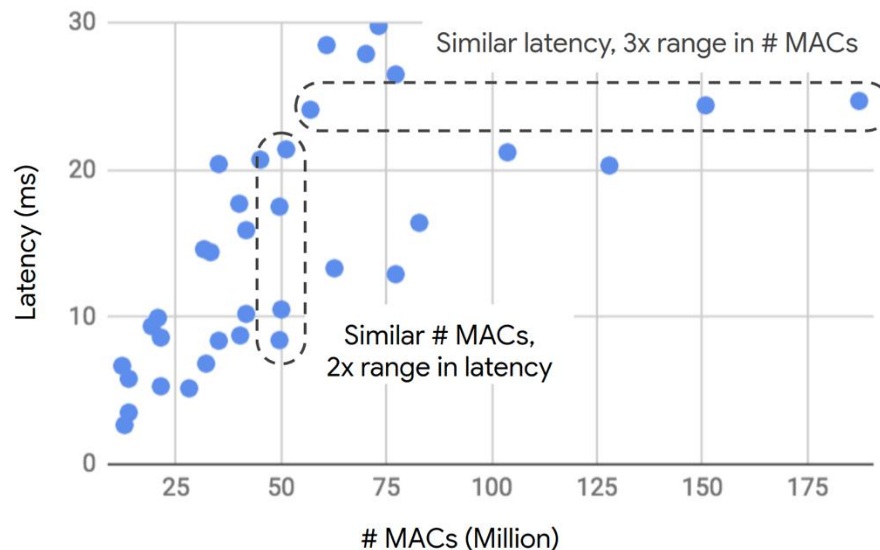
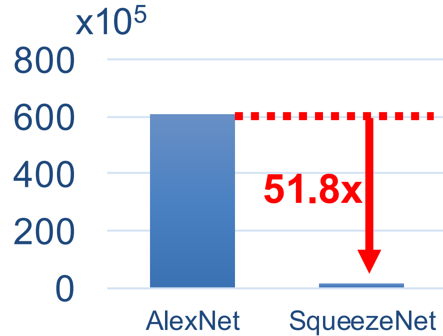


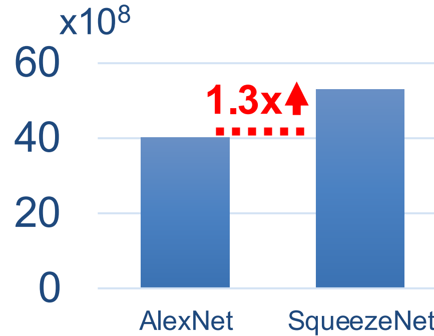
Image Source: Google AI Blog

[**Yang**, *CVPR* 2017], [**Chen**, *SysML*, 2018], [**Yang**, *ECCV* 2018]

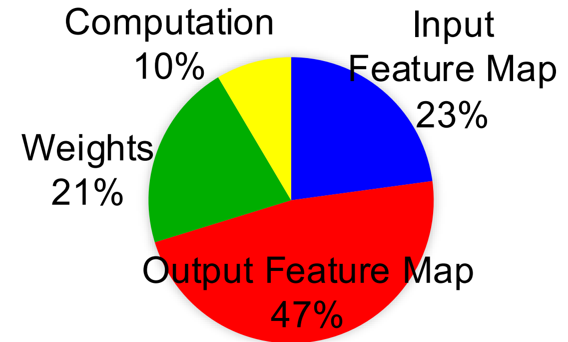
Example: AlexNet vs. SqueezeNet



# of Weights



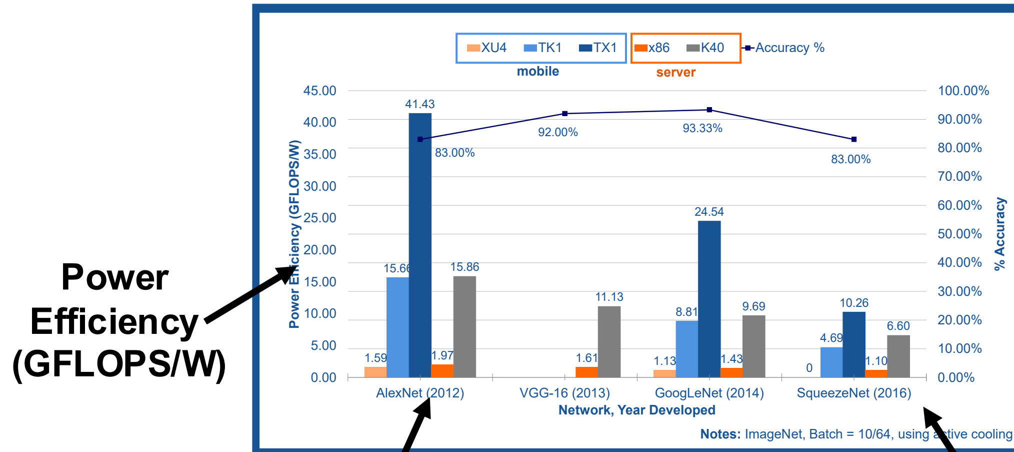
Normalized Energy



Energy Breakdown (SqueezeNet)

Results for SqueezeNetv1.0
Batch size=48

Power Efficiency (GFLOPS/W)



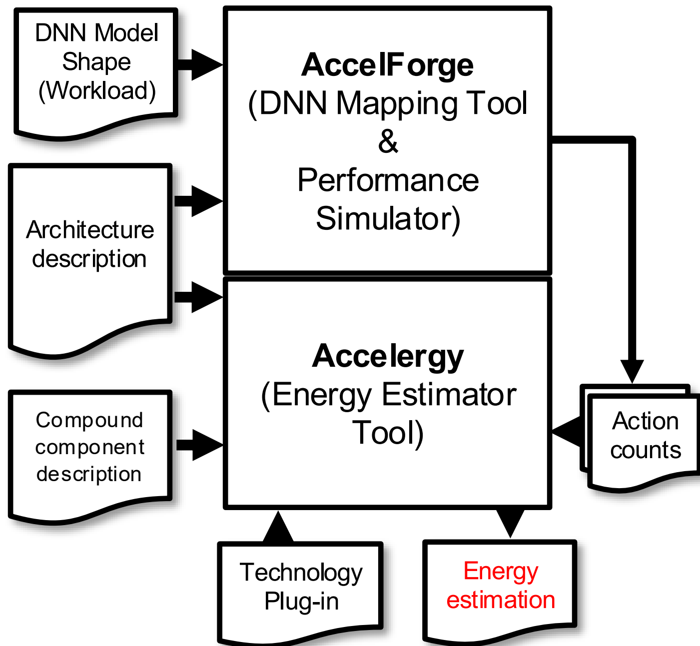
AlexNet

SqueezeNet

[**Movidius**, *Hot Chips* 2016]

DNN Processor Evaluation Tools

- **Require systematic way to**
  - Evaluate and compare wide range of DNN processor designs
  - Rapidly explore design space
- **Accelergy** **[Wu,** *ICCAD* 2019**]**
  - Early-stage energy estimation tool at the architecture level
    - Estimate energy consumption based on architecture level components (e.g., # of PEs, memory size, on-chip network)
  - Evaluate architecture level energy impact of emerging devices
    - Plug-ins for different technologies
- **AccelForge**
  - DNN mapping tool
  - Performance Simulator → Action counts



Labs and final project

Open-source code available at: http://accelergy.mit.edu

Summary

- Evaluate hardware using the appropriate DNN model and dataset
  - Difficult tasks typically require larger models
  - Different datasets for different tasks
  - Number of datasets growing at a rapid pace
- A comprehensive set of metrics should be considered when comparing DNN hardware to fully understand design tradeoffs

References

- Chapter 2 & 3 in Book
  - https://doi.org/10.1007/978-3-031-01766-7
- Other Works Cited in Lecture

  - Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." International Journal of Computer Vision 115.3 (2015): 211-252.
  - Sun, Chen, et al. "Revisiting unreasonable effectiveness of data in deep learning era." arXiv preprint arXiv:1707.02968 (2017).
  - Shrivastava, Ashish, et al. "Learning from simulated and unsupervised images through adversarial training." arXiv preprint arXiv:1612.07828 (2016).
  - T.-J. Yang et al., “NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications,” ECCV 2018.
  - Y.-H. Chen*, T.-J. Yang*, J. Emer, V. Sze, “Understanding the Limitations of Existing Energy-Efficient Design Approaches for Deep Neural Networks,” *SysML Conference* 2018.
  - T.-J. Yang et al., “Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning,” *CVPR* 2017.
  - Chen et al., Eyexam, https://arxiv.org/abs/1807.07928
  - Williams et al., “Roofline: An insightful visual performance model for floating-point programs and multicore architectures,” CACM 2009
  - Wu et al., “Accelergy: An architecture-level energy estimation methodology for accelerator designs,” ICCAD 2019
  - Parashar et al., “Timeloop: A systematic approach to dnn accelerator evaluation,” ISPASS 2019

Extended Einsums

Acknowledgements

Based on collaborations with:

- Michael Gilbert (MIT)
- Nandeeka Nayak (UC Berkeley)
- Toluwa Odemuyiwa (UC Davis)
- Michael Pellauer (NVIDIA)

Example Einsum

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal rank variable (index) values, i.e., the iteration space
- At each point in iteration space:
  - Calculate value on right hand at the specified rank variable values for each operand
  - Assign calculated value to operand at specified rank variable values on left hand side
  - Unless that operand is non-zero, then reduce value into it

Matrix multiply - simple traversal

Tensor: IS[K, N, M]

Rank: K ----->

K = 0 (Rank: M across, Rank: N down)

N \ M	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

K = 1 (Rank: M across, Rank: N down)

N \ M	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

K = 2 (Rank: M across, Rank: N down)

N \ M	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Tensor: A[M, K]

(Rank: K across, Rank: M down)

M \ K	0	1	2
0	1	2	3
1	1	2	2
2	1	5	3
3	5	5	1

Tensor: B[N, K]

(Rank: K across, Rank: N down)

N \ K	0	1	2
0	2	3	3
1	2	1	3
2	3	5	1
3	5	4	3

Tensor: Z[M, N]

(Rank: N across, Rank: M down)

M \ N	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

0:00 / 0:50

Matrix multiply - complex traversal

Tensor: IS[K, N, M]

Rank: K -----> 0

Rank: M (columns), Rank: N (rows)

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Rank: K -----> 1

Rank: M (columns), Rank: N (rows)

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Rank: K -----> 2

Rank: M (columns), Rank: N (rows)

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Tensor: A[M, K]

Rank: K (columns), Rank: M (rows)

	0	1	2
0	1	2	3
1	1	2	2
2	1	5	3
3	5	5	1

Tensor: B[N, K]

Rank: K (columns), Rank: N (rows)

	0	1	2
0	2	3	3
1	2	1	3
2	3	5	1
3	5	4	3

Tensor: Z[M, N]

Rank: N (columns), Rank: M (rows)

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

0:00 / 0:50

Tensor References

Tensor	Description
$A_{k,m}$	• Rank variables: k, m • Rank names: "K", "M" • Rank shapes: K, M
$A_{k,m}^{K,M}$	• Rank variables: k, m • Rank names: "K", "M" • Rank shapes: K, M
$A_{k,m}^{K=X,M=X}$	• Rank variables: k, m • Rank names: "K", "M" • Rank shapes: X, X
$A_{k,m}^{RR=XX,SS=XX}$	• Rank variables: k, m • Rank names: "RR", "SS" • Rank shapes: XX, XX

Einsum Patterns

$$Z_{m,n} = A_{m,k} \times B_{k,n}$$ Matrix-matrix multiply

One rank (k) is contracted, and two ranks (m and n) are uncontracted

$$Z_{m,n} = A_{k,m} \times B_{k,n}$$ Matrix-matrix multiply

Again, one rank (k) is contracted, and two ranks (m and n) are uncontracted

$$Z_{n,m} = A_{k,m} \times B_{n,k}$$ Matrix-matrix multiply

Again, one rank (k) is contracted, and two ranks (m and n) are uncontracted

$$Z_{p,q} = A_{p,r} \times B_{q,r}$$ Matrix-matrix multiply

It doesn’t matter what the rank variables are.

More Einsum Patterns

$Z_m = A_{k,m} \times B_k$	Matrix-vector multiply
$Z_m = B_k \times A_{k,m}$	Matrix-vector multiply (its commutative)
$Z_{m,n} = A_m \times B_n$	Cartesian product
$Z_m = A_m \times B_m$	Element-wise multiply
$Z_m = A_m + B_m$	Element-wise addition (different operator)

Rank Variable Patterns

$$Z_i = A_i \times B_i$$ Element-wise matrix multiplication

$$Z_{i,j} = A_{i,j} \times B_{i,j}$$ Element-wise matrix multiplication

Repeated pairs of rank variables have no effect

$$Z_{(i,j)} = A_{(i,j)} \times B_{(i,j)}$$ Element-wise matrix multiplication

Repeated pairs of rank variables can be treated as one rank variable

$$Z_{ij} = A_{ij} \times B_{ij}$$ Element-wise matrix multiplication

So the tuple coordinates can be treated as a single coordinate, where

$$ij = i * J + j$$

Partitioned Einsum Patterns

$$Z_i = A_i \times B_i$$

Element-wise matrix multiplication

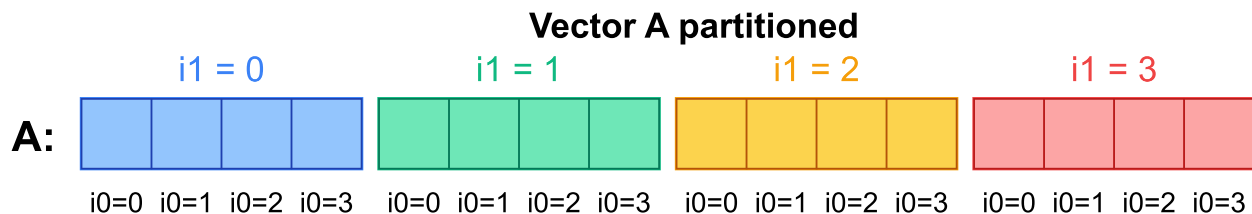
If we assume $i = i1 \times I0 + i0$

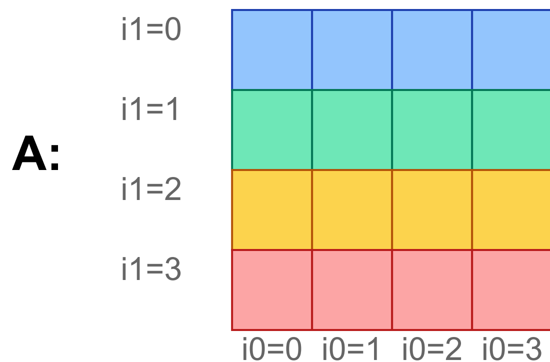
$$Z_{i1,i0} = A_{i1,i0} \times B_{i1,i0}$$

Element-wise matrix multiplication

Now we have a partitioned tensor

Partitioned Tensor Visualization





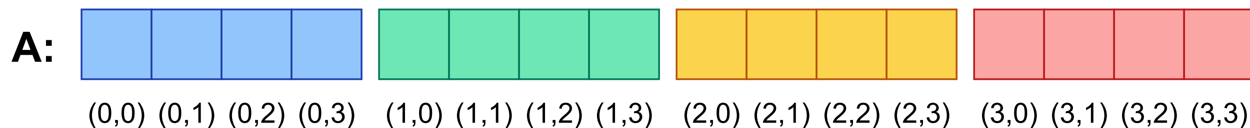
Partitioning always adds a rank

Flattening

$$Z_{i1,i0} = A_{i1,i0} \times B_{i1,i0}$$

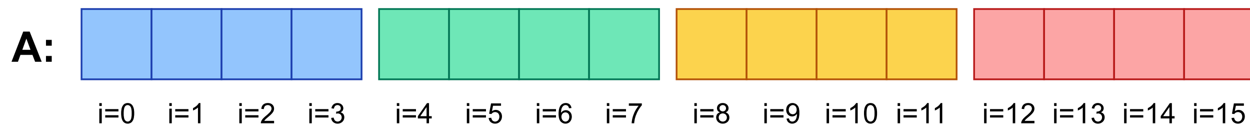
$$Z_{(i1,i0)} = A_{(i1,i0)} \times B_{(i1,i0)}$$

Vector A flattened



Since i = i1 * I0 + i0

Vector A flattened and relabeled



Matrix-Matrix Multiplication Variants?

$$Z_{m,n} = A_{k,m} \times B_{k,n}$$

Matrix-matrix multiply

$$Z_{m,p,n} = A_{k,m,p} \times B_{k,n} \qquad Z_{mp,n} = A_{k,mp} \times B_{k,n}$$

Matrix-matrix multiply

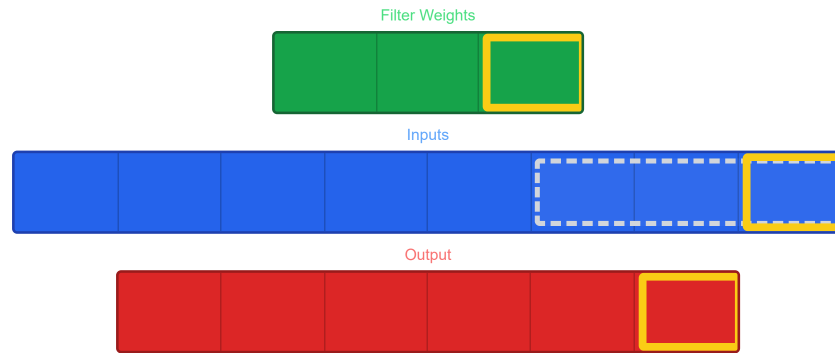
$$Z_{m,n} = A_{k,l,m} \times B_{k,l,n} \qquad Z_{m,n} = A_{kl,m} \times B_{kl,n}$$

Matrix-matrix multiply

$$Z_{m,p,n} = A_{k,l,m,p} \times B_{k,l,n} \qquad Z_{mp,n} = A_{kl,mp} \times B_{kl,n}$$

Matrix-matrix multiply

Convolution



$$O_q = I_{q+s} \times F_s$$

One-dimensional convolution

$$O_q^Q = I_{q+s}^W \times F_s^S$$

One-dimensional convolution

1D convolution - complex traversal

Tensor: IS[Q, S]

Rank: S

Rank: Q	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0

Tensor: F[S]

Rank: S

0	1	2
3	2	3

Tensor: I[W]

Rank: W

0	1	2	3	4	5	6	7
5	3	2	3	4	5	2	2

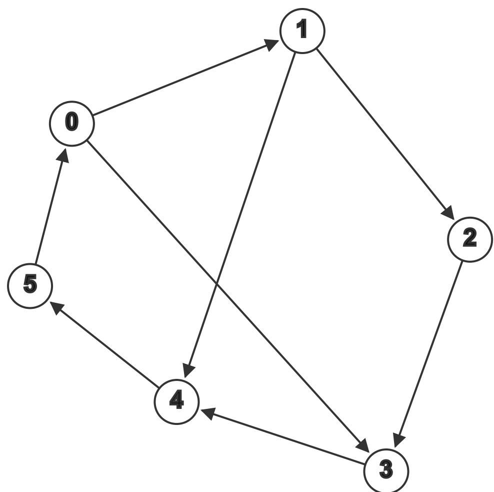
Tensor: O[Q]

Rank: Q

0	1	2	3	4	5
0	0	0	0	0	0

0:00 / 0:20

Rank Shapes



D

S	0	1	2	3	4	5
0		■		■		
1			■		■	
2				■		
3					■	
4						■
5	■					

Adjaceny matrix declaration:

$$G^{S=6,D=6} \quad or \quad G^{S=V,D=V}$$

6.5930/1
Hardware Architectures for Deep Learning

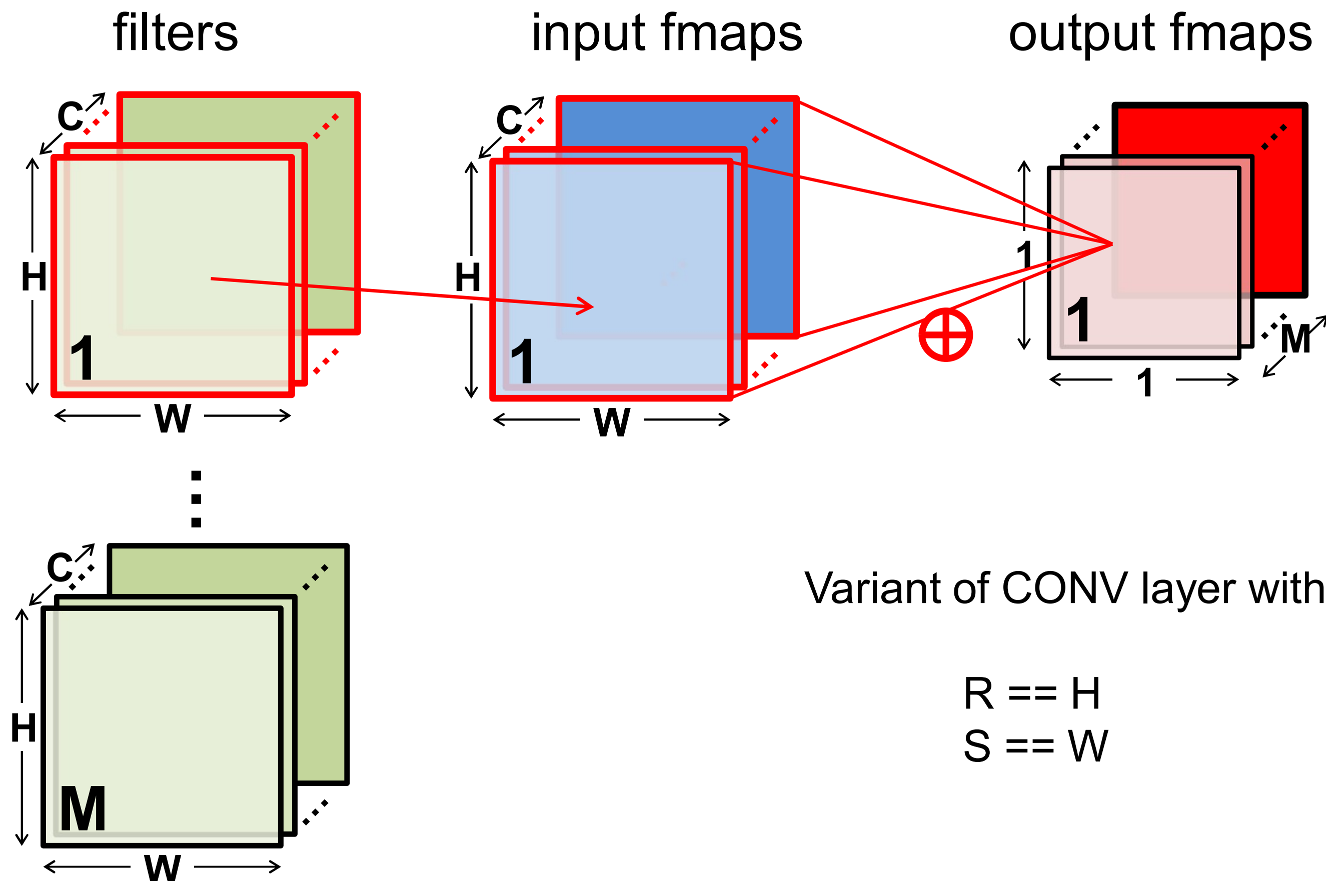
Kernel Computation

February 9, 2026

Joel Emer & Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

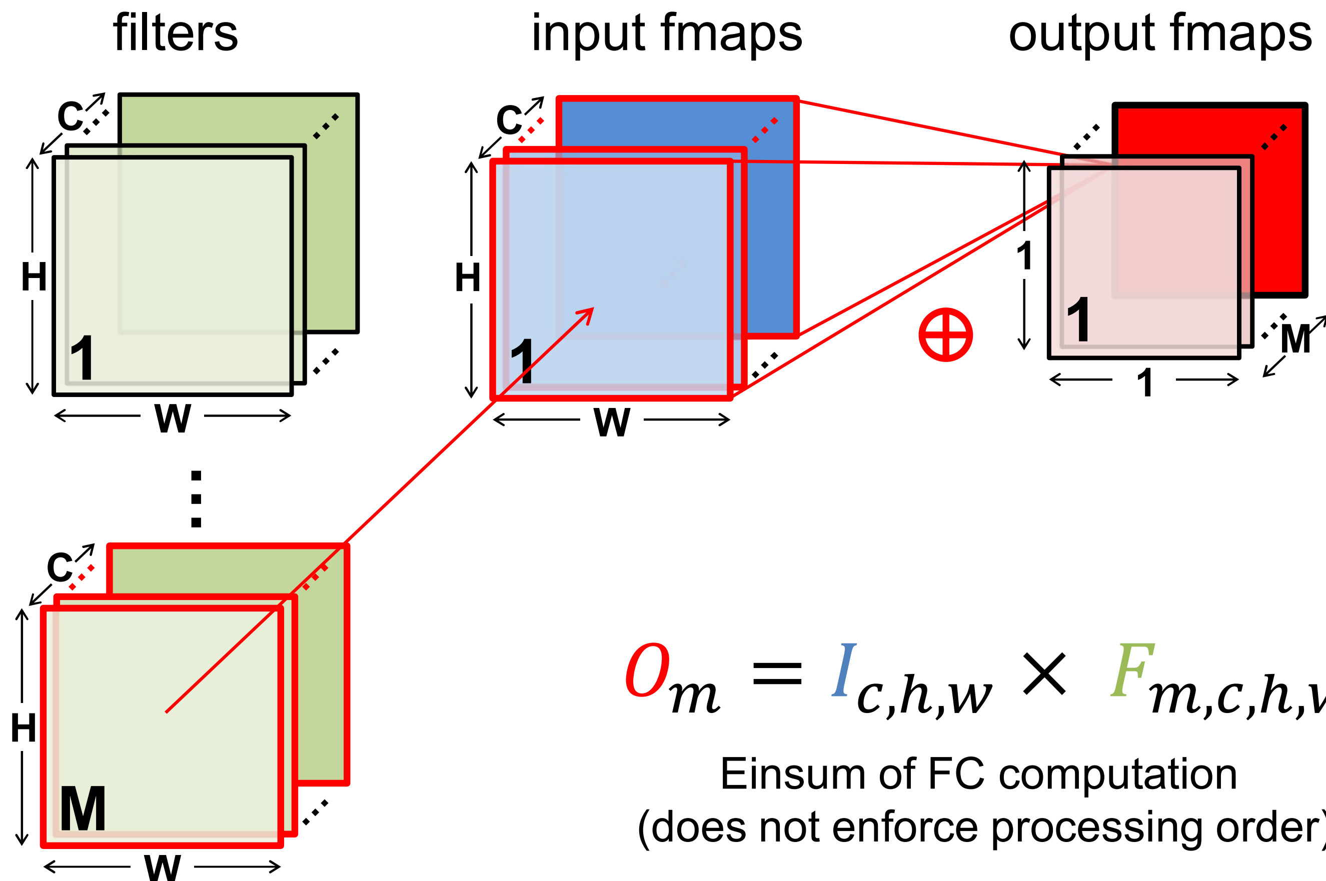
Fully Connected Computation



Variant of CONV layer with:

R == H
S == W

Fully Connected Computation



$$O_m = I_{c,h,w} \times F_{m,c,h,w}$$

Einsum of FC computation
(does not enforce processing order)

Fully Connected Computation

```
int i[C][H][W];      # Input activations
int f[M][C][H][W];   # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
  for c in [0, C):
    for h in [0, H):
       for w in [0, W):
          o[m] += i[c][h][w]*f[m][c][h][w]
```

Should be bias, which we will ignore for simplicity

Loop nest of FC computation
(enforces some processing order)

Convert FC Compute to Matrix-Vector Multiply

```
int i[C][H][W];      # Input activations
int f[M][C][H][W];   # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
  for c in [0, C):
    for h in [0, H):
       for w in [0, W):
          o[m] += i[c][h][w]*f[m][c][h][w]
```

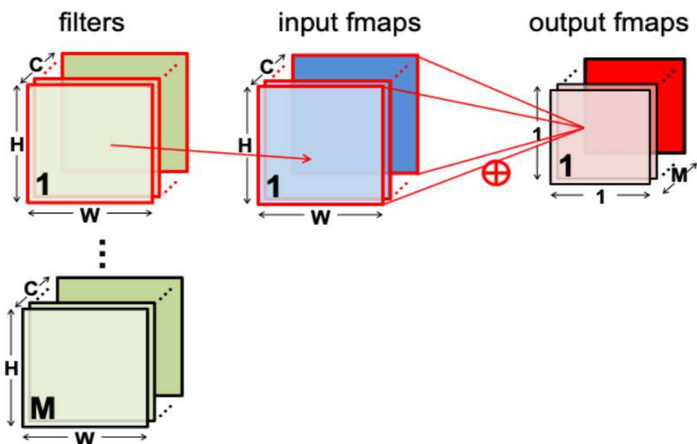
Flatten C, H, W ranks to CHW

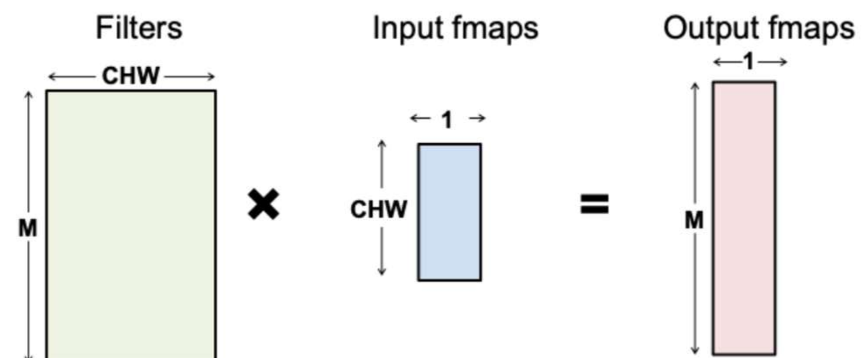
```
int i[CHW];           # Input activations
int f[M][CHW];        # Filter weights
int o[M];             # Output activations


for m in [0, M):
  o[m] = 0;
    for chw in [0, CHW):
       o[m] += i[chw]*f[CHW*m + chw]
```

Convert FC Compute to Matrix-Vector Multiply

```
int i[C][H][W];      # Input activations
int f[M][C][H][W];   # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
  for c in [0, C):
    for h in [0, H):
      for w in [0, W):
        o[m] += i[c][h][w]*f[m][c][h][w]
```

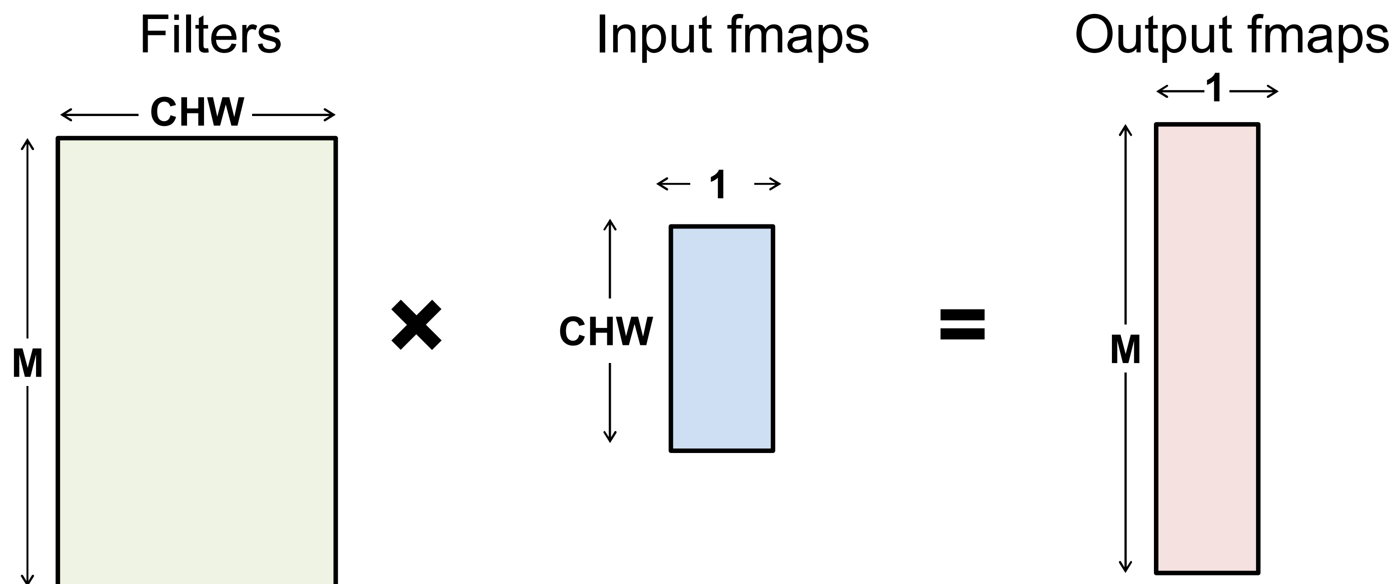
```
int i[CHW];          # Input activations
int f[M][CHW];       # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
    for chw in [0, CHW):
      o[m] += i[chw]*f[m][chw]
```





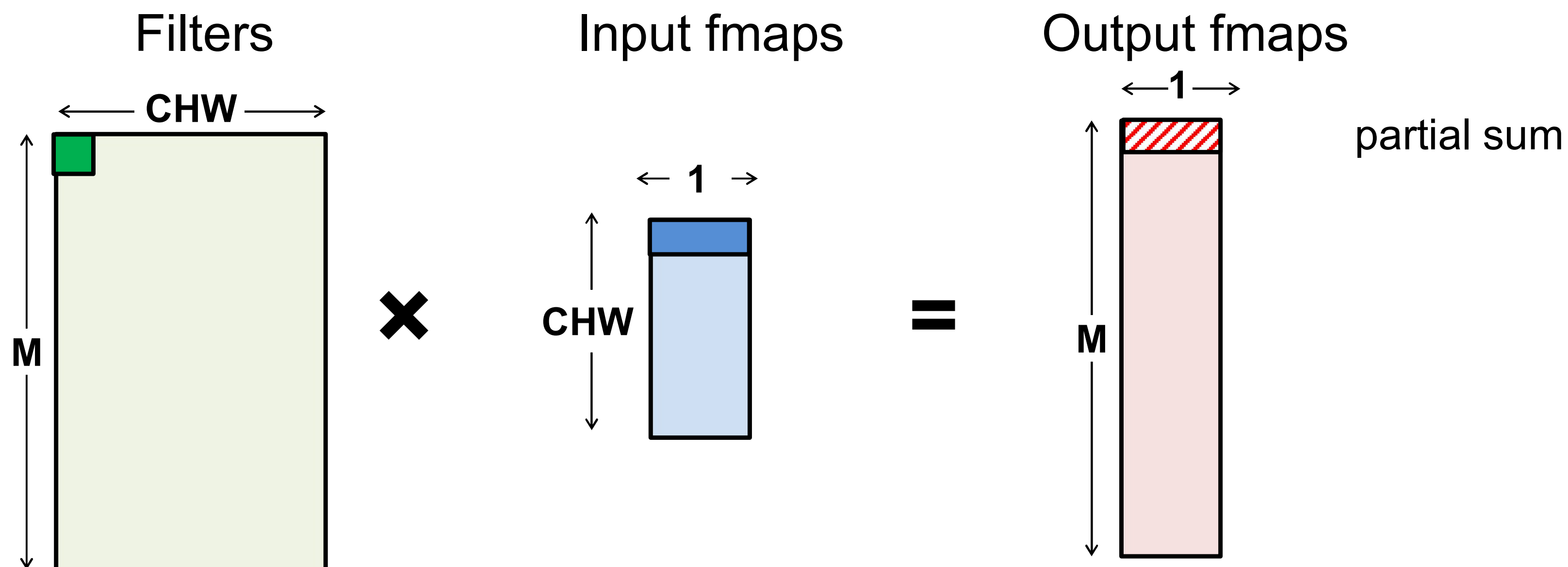
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum



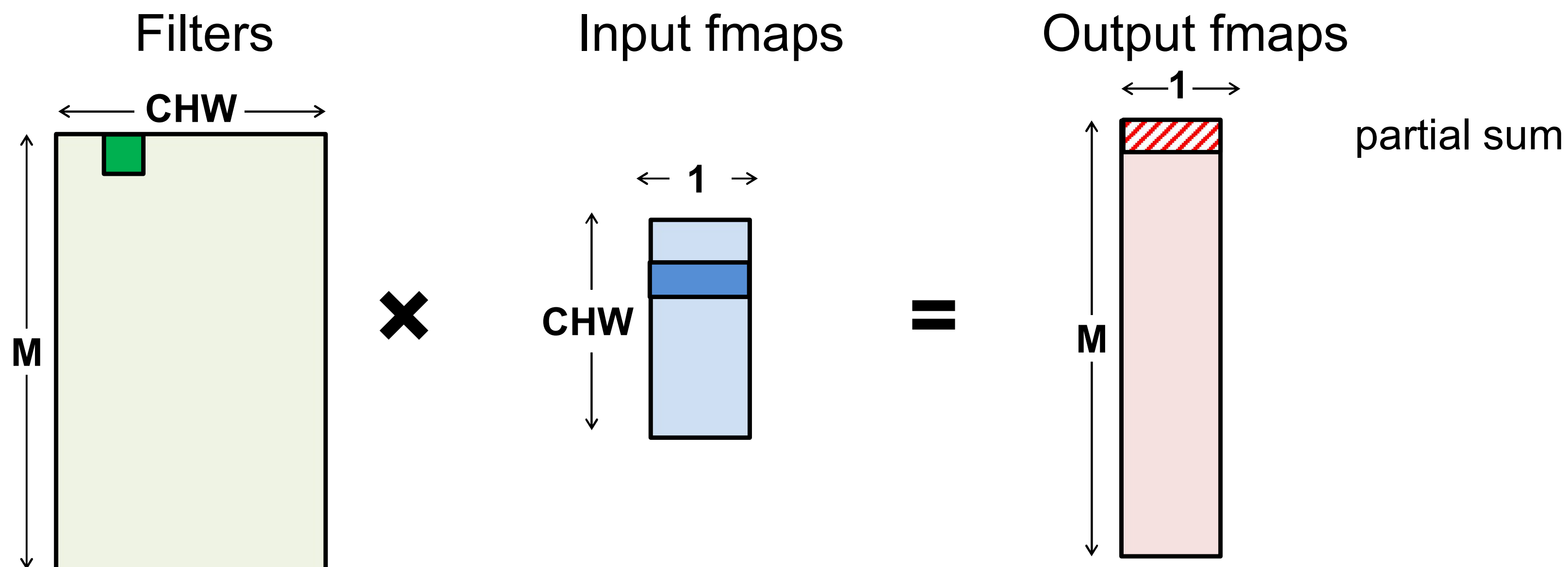
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum
(increment chw)



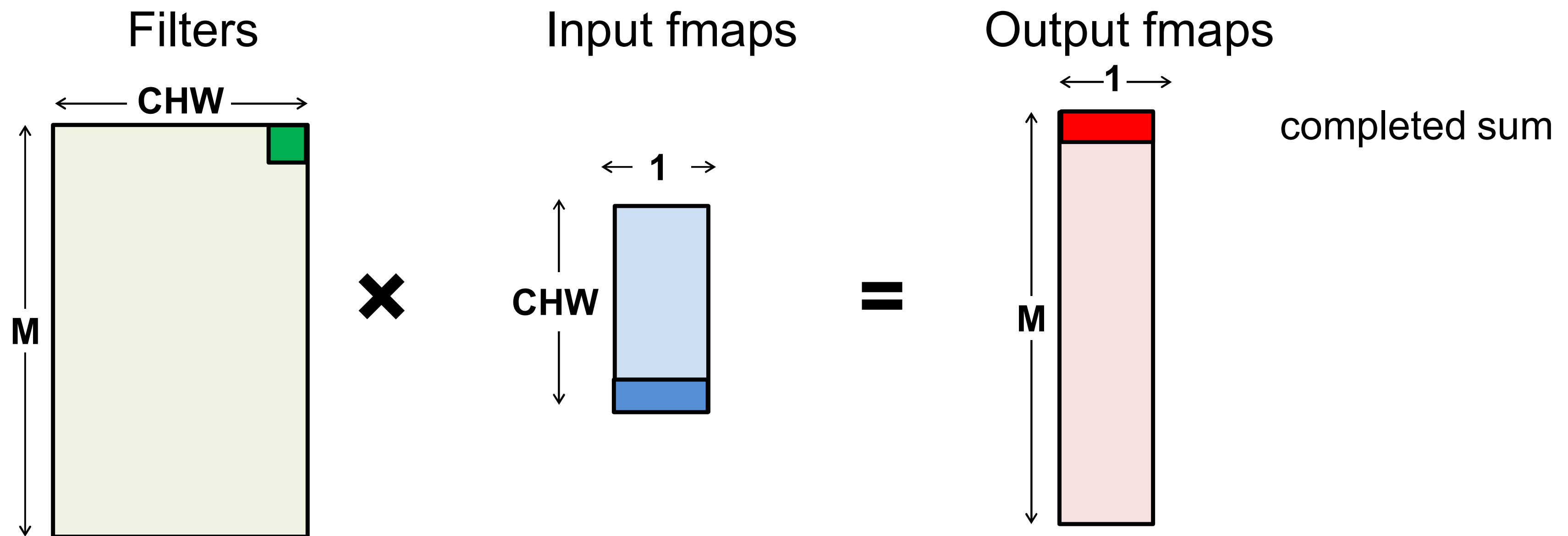
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum
(increment chw)



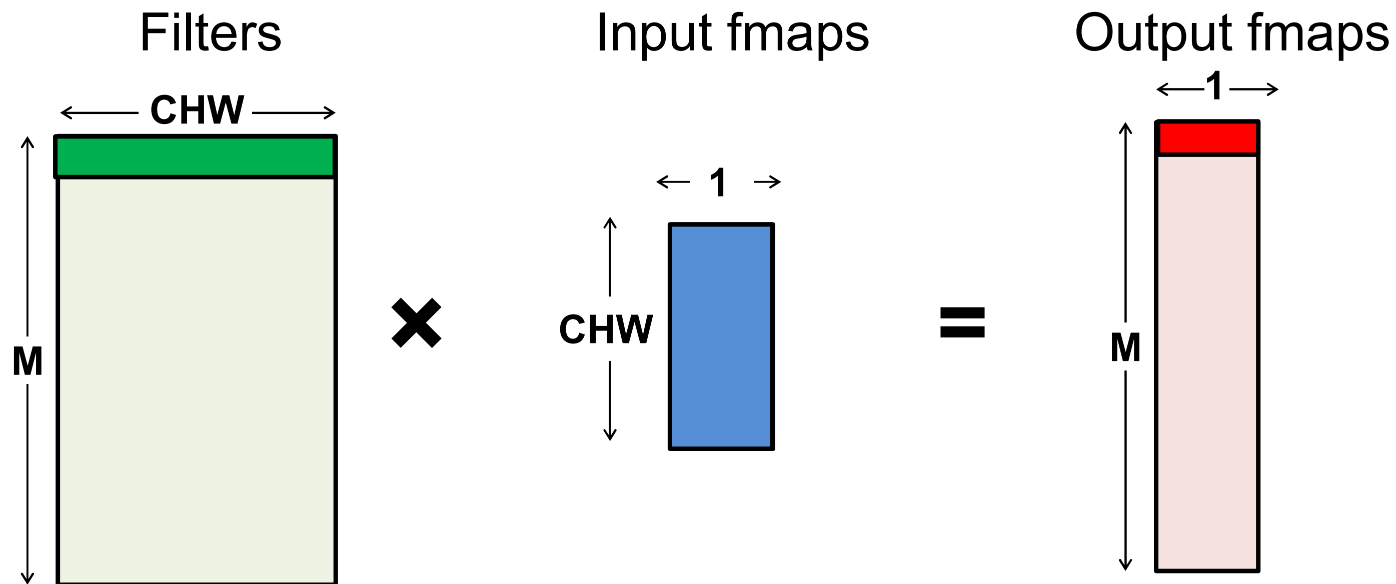
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum
(increment chw)



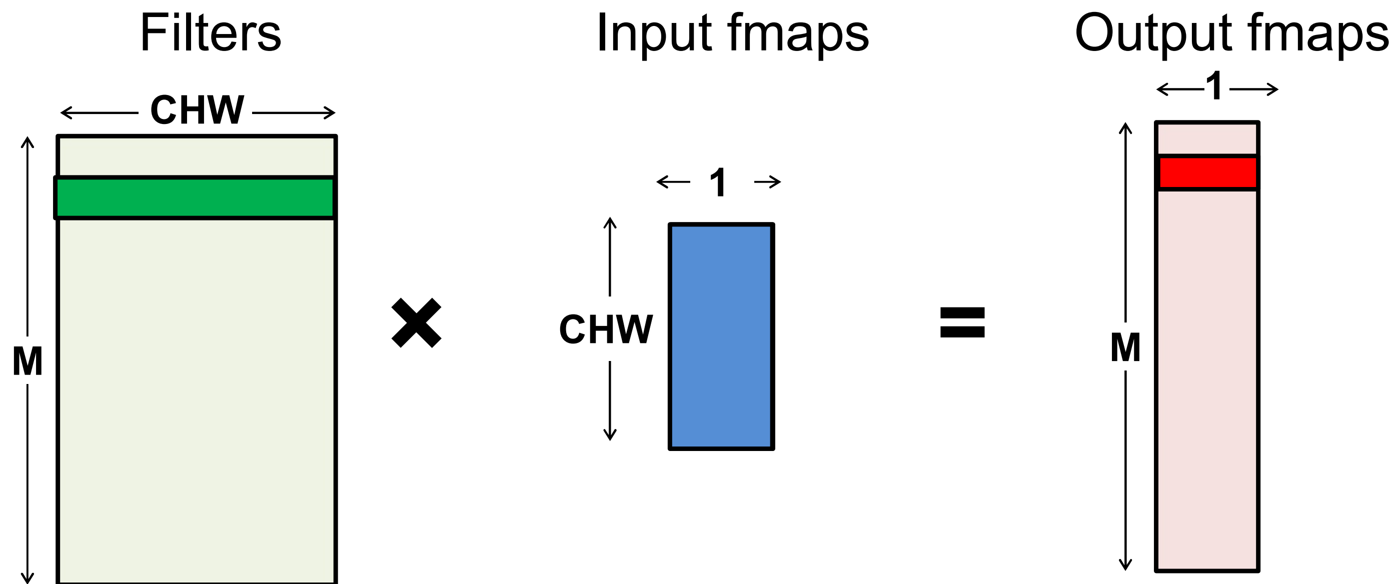
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum



FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum (increment m)



Einsum for Flattened FC

Original Flattened

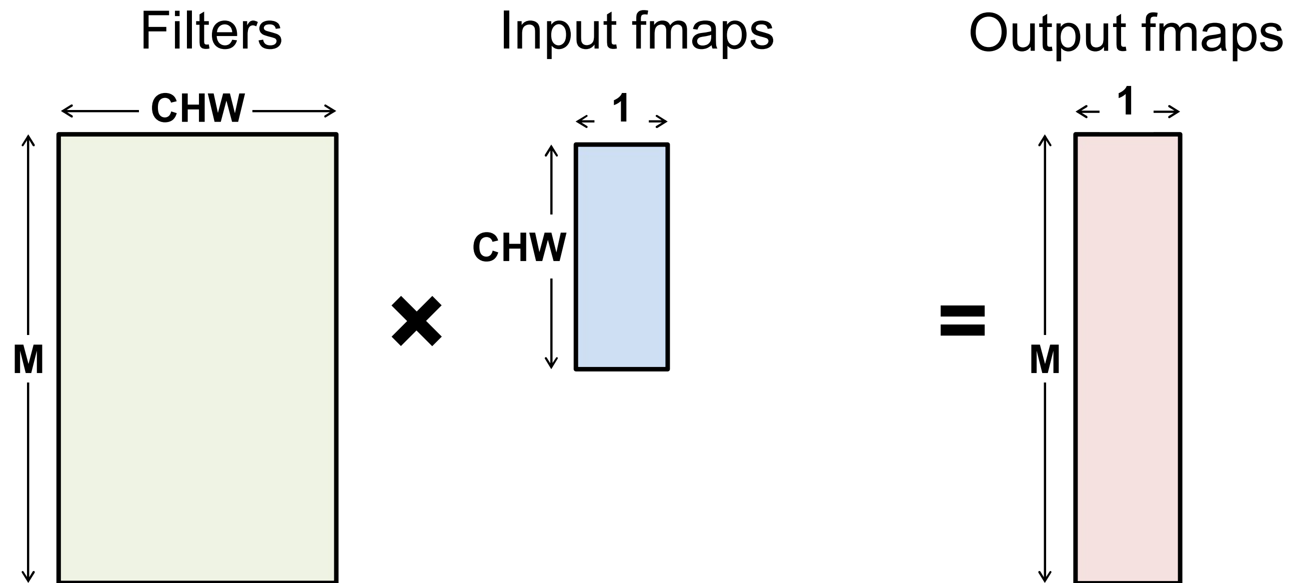
$$I_{c,h,w} \rightarrow I_{H\times W\times c+W\times h+w} \rightarrow I_{chw}$$

$$F_{m,c,h,w} \rightarrow F_{m,H\times W\times c+W\times h+w} \rightarrow F_{m,chw}$$

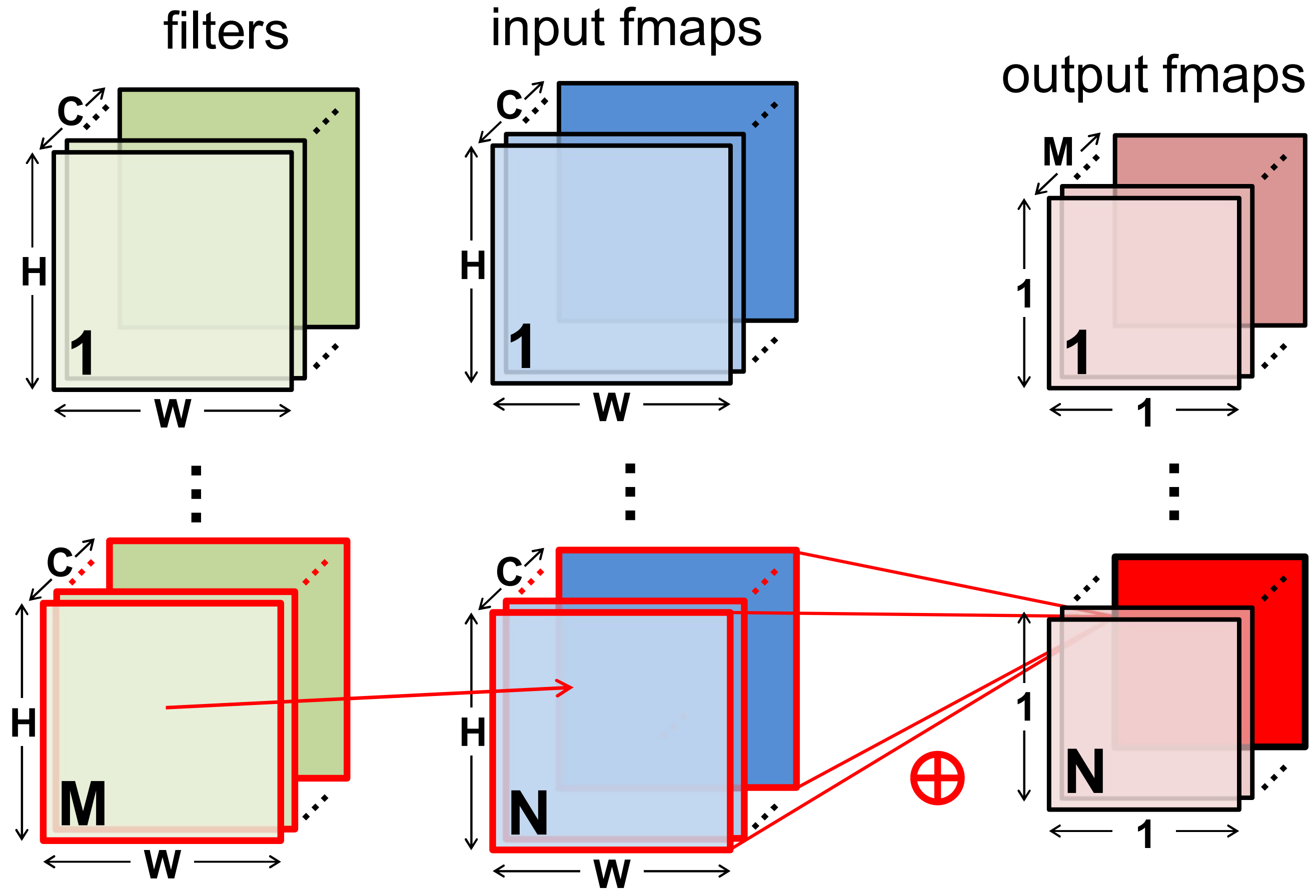
$$O_m = I_{c,h,w} \times F_{m,c,h,w} \rightarrow O_m = I_{chw} \times F_{m,chw}$$

Einsum for FC as Matrix Vector

$$O_m = I_{chw} \times F_{m,chw}$$

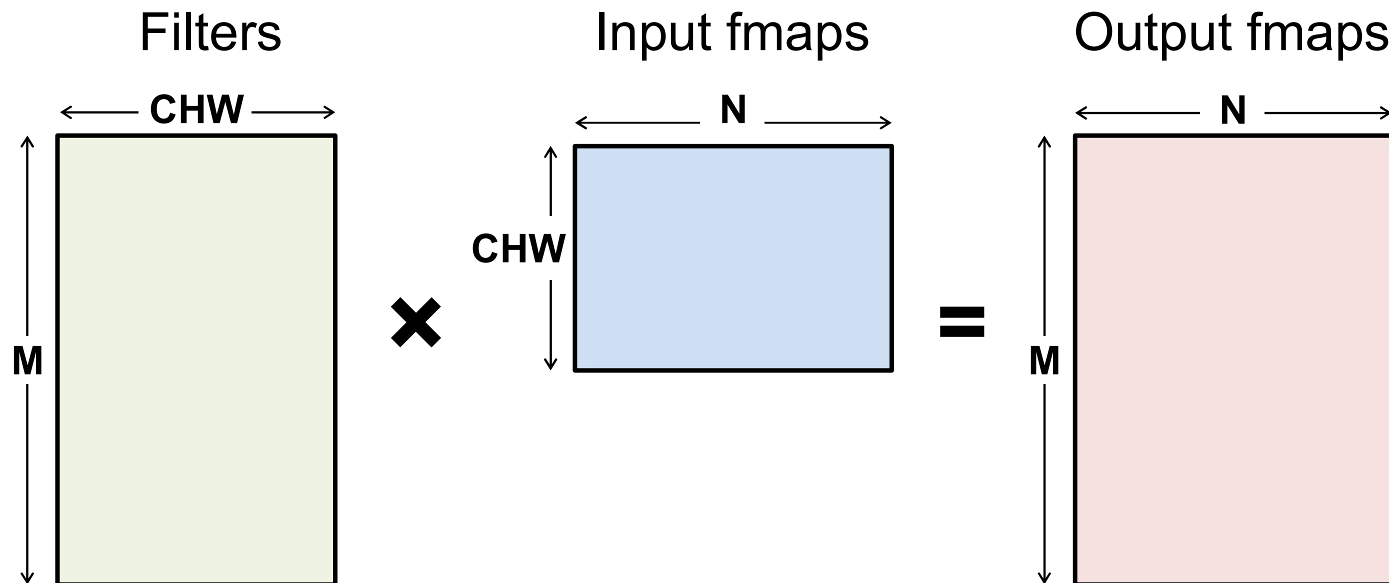


FC Layer – Batch (N)



FC Compute → Matrix-Matrix Multiply

$$O_{\boldsymbol{n},m} = I_{\boldsymbol{n},chw} \times F_{m,chw}$$



After flattening, having a batch size of N turns the **matrix-vector** multiply into a **matrix-matrix** multiply

FC Compute → Matrix-Matrix Multiply

$$O_{n,m} = I_{n,chw} \times F_{m,chw}$$

reduction on rank **chw**

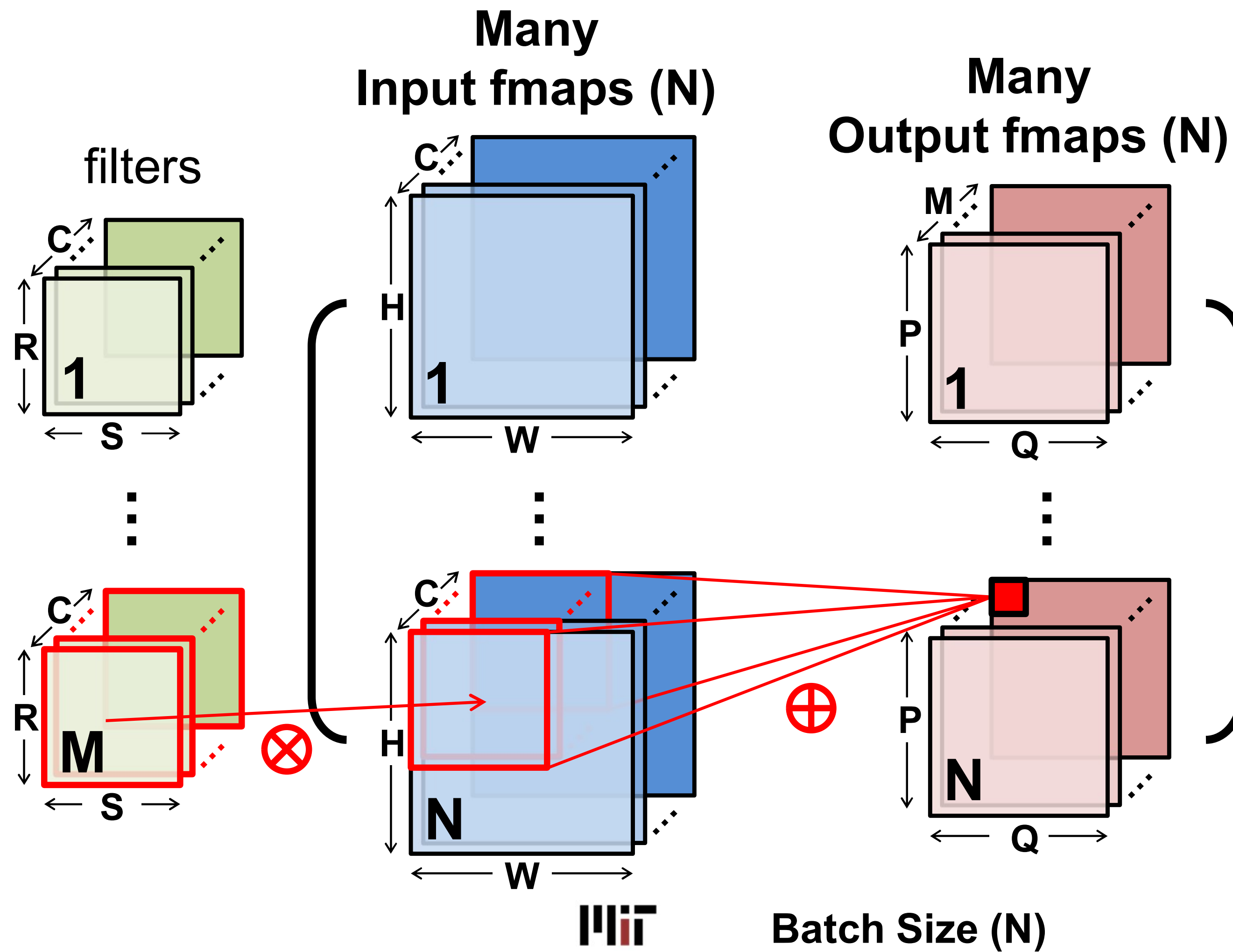
Typical matrix multiplication notation (used in Lab 2)

$$C_{m,n} = A_{m,k} \times B_{k,n}$$

reduction on rank **k**

Note: for Einsum, the order of ranks does not matter

Convolution (CONV) Layer



Convolution Einsum

Algebraic Notation

$$\mathbf{o}[n][m][p][q] = \mathbf{b}[m] + \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \mathbf{i}[n][c][Up+r][Uq+s] \times \mathbf{f}[m][c][r][s].$$

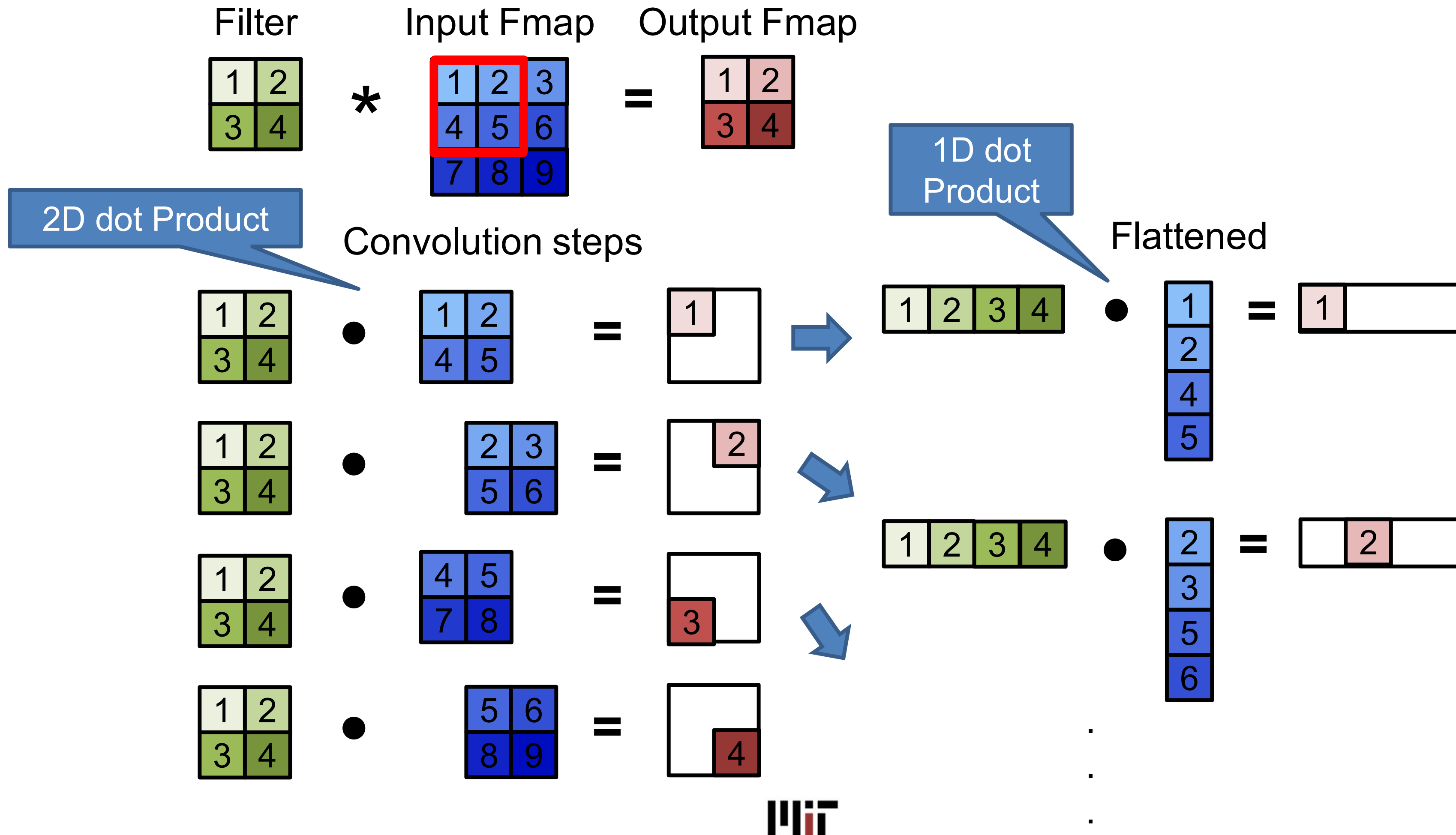
Einsum Notation

$$O_{n,m,p,q} = B_m + I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Note:

- There is a relationship between the ranks for the tensors used for convolution
  - *Not the case for fully connected*
- Specifically, h=U×p+r and w=U×q+s → Use change of variables to capture this relationship

Convolution (CONV) Layer



Convolution (CONV) Layer

Filter

1	2
3	4

*

Input Fmap

1	2	3
4	5	6
7	8	9

=

Output Fmap

1	2
3	4

Convolution

Flattened

[1 2 3 4] • [1 2 4 5] = [1 _ _] [1 2 3 4] • [2 3 5 6] = [_ 2 _] . . .

Convolution (CONV) Layer

Filter * Input Fmap = Output Fmap

Convolution:

Flattened

· · ·

Convolution (CONV) Layer

$[1\ 2\ 3\ 4] \bullet [1\ 2\ 4\ 5]^T = [1\ \ \ ]$ $[1\ 2\ 3\ 4] \bullet [2\ 3\ 5\ 6]^T = [\ \ 2\ \ ]$ . . .

Flattened

Matrix Multiply (by Toeplitz Matrix)

$[1\ 2\ 3\ 4] \times$

1	2	4	5
2	3	5	6
4	5	7	8
5	6	8	9

$= [1\ 2\ 3\ 4]$

Convolution (CONV) Layer

Filter	
1	2
3	4

*

Input Fmap		
1	2	3
4	5	6
7	8	9

=

Output Fmap	
1	2
3	4

Convolution:

Matrix Multiply (by Toeplitz Matrix)

1	2	3	4

×

1	2	4	5
2	3	5	6
4	5	7	8
5	6	8	9

=

1	2	3	4

Convert to matrix multiply using the **Toeplitz Matrix**

(aka ‘im2col’)

Convolution (CONV) Layer

Filter * Input Fmap = Output Fmap

Filter	
1	2
3	4

*

Input Fmap		
1	2	3
4	5	6
7	8	9

=

Output Fmap	
1	2
3	4

Convolution:

Matrix Multiply (by Toeplitz Matrix)

1	2	3	4

×

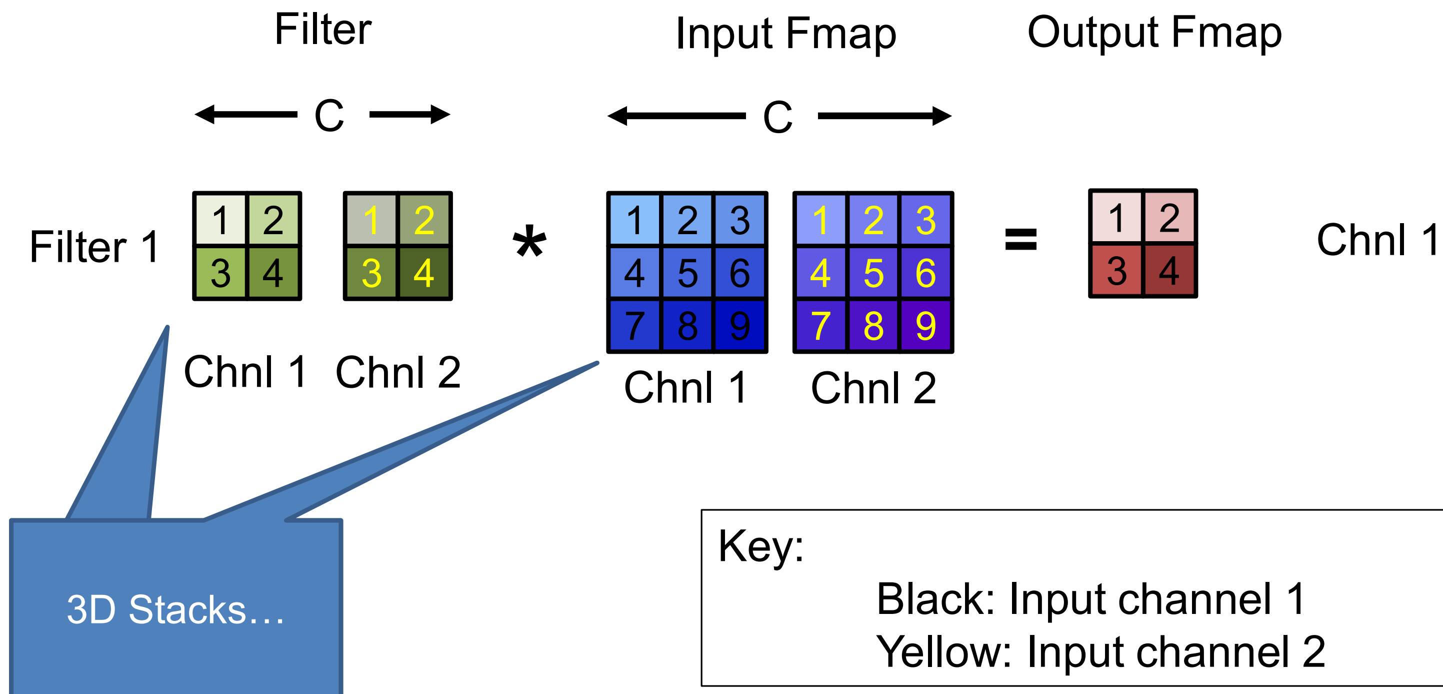
1	2	4	5
2	3	5	6
4	5	7	8
5	6	8	9

=

1	2	3	4

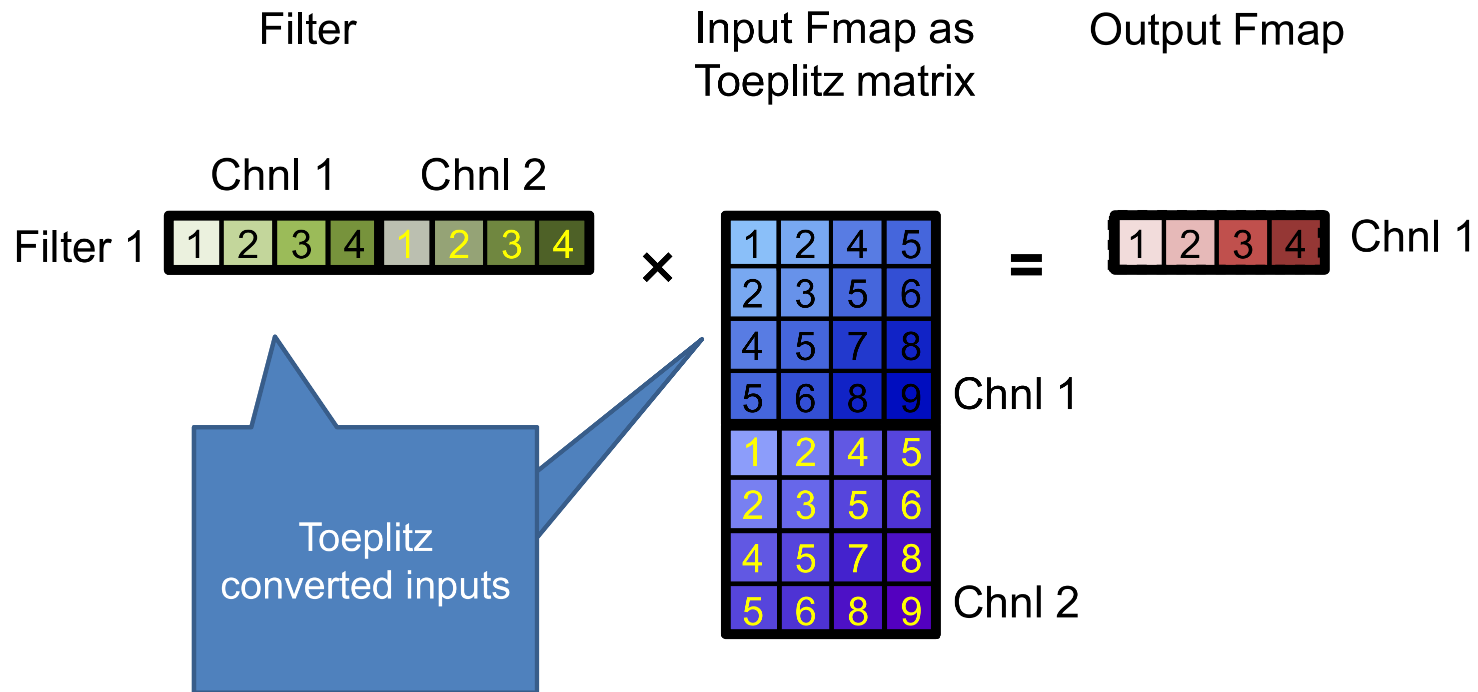
Data is repeated

Convolution (CONV) Layer



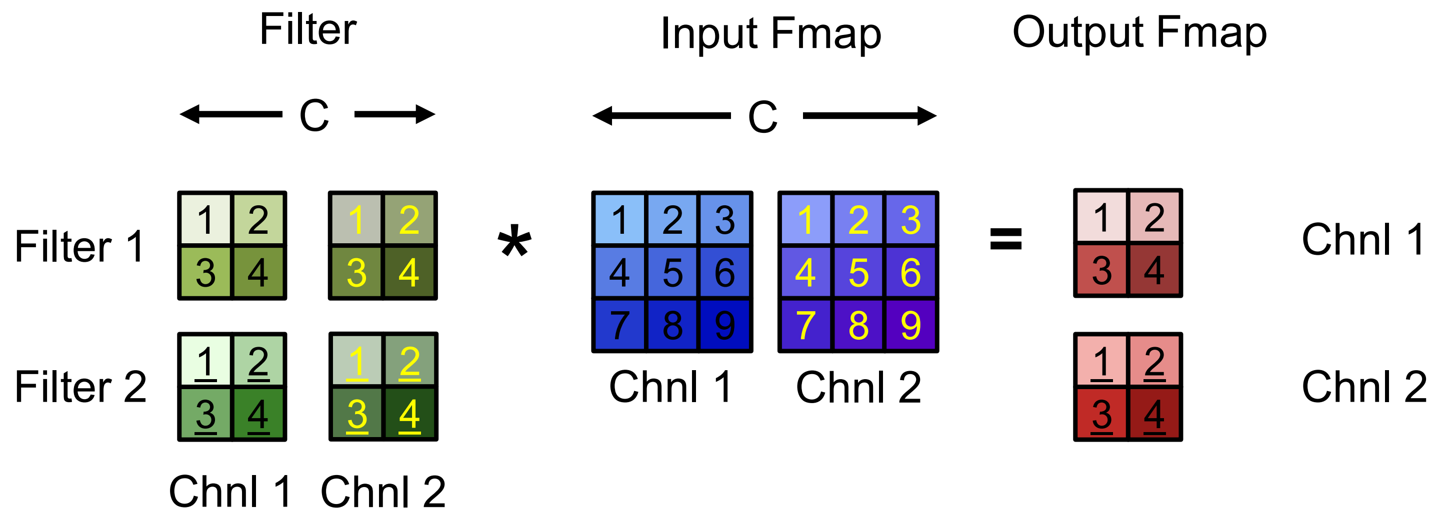
Convolution (CONV) Layer

Multiple Input Channels



Convolution (CONV) Layer

Multiple Input Channels and Output Channels



Key:
Black: Input channel 1
Yellow: Input channel 2
Underlined: Output channel 2

Convolution (CONV) Layer

Multiple Input Channels and Output Channels

Filter

Input Fmap as Toeplitz matrix

Output Fmap

	Chnl 1				Chnl 2			
Filter 1	1	2	3	4	1	2	3	4
Filter 2	1	2	3	4	1	2	3	4

×

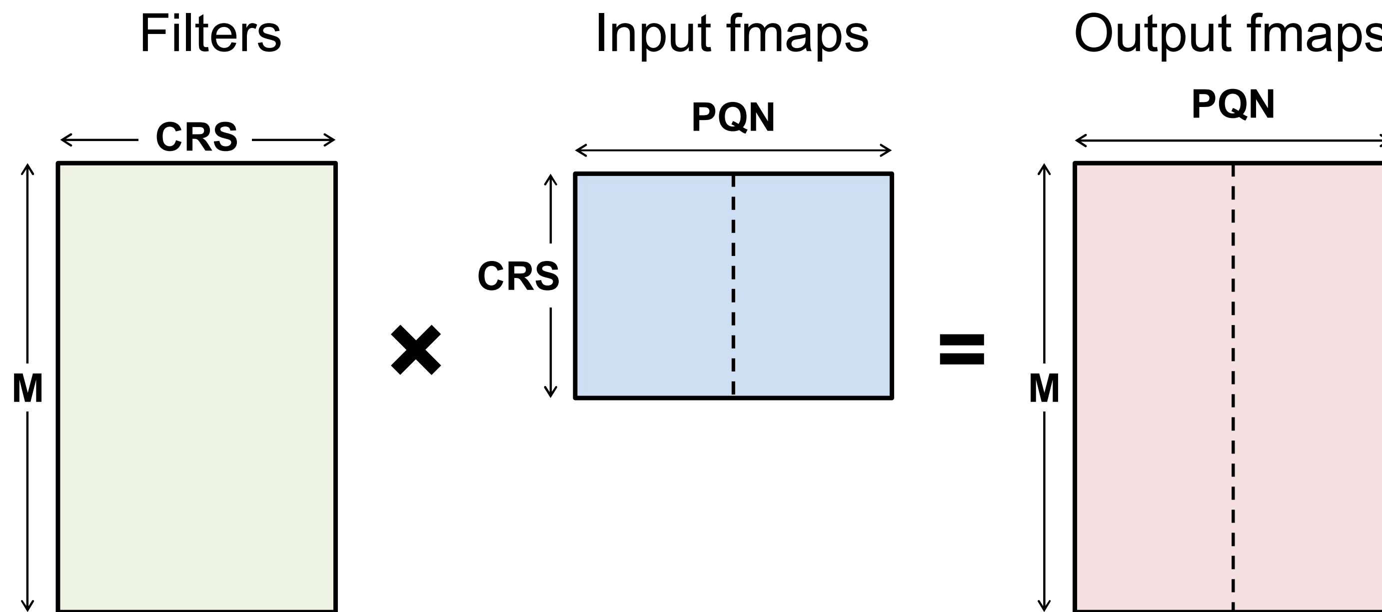
1	2	4	5	
2	3	5	6	
4	5	7	8	
5	6	8	9	Chnl 1
1	2	4	5	
2	3	5	6	
4	5	7	8	
5	6	8	9	Chnl 2

=

1	2	3	4	Chnl 1
1	2	3	4	Chnl 2

Convolution (CONV) Layer → Matrix Multiplication

Dimensions of matrices for matrix multiply in convolution layers with batch size N



where P=H-R+1 and Q=W-S+1

N=2 in example

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$ 1-D convolution

Convolution can be represented in **two** steps

Step (1) $T_{q,s} = I_{q+s}$ Toeplitz conversion

Step (2) $O_q = T_{q,s} \times F_s$ Matrix multiplication

1-D Toeplitz Convolution Einsum

Review: Illustrating **two** steps for 1-D convolution

Filter (F_s) Input Fmap (I_{q+s}) Output Fmap (O_q)

1	2	3

*

1	2	3	4	5	6

=

1	2	3	4

X

=

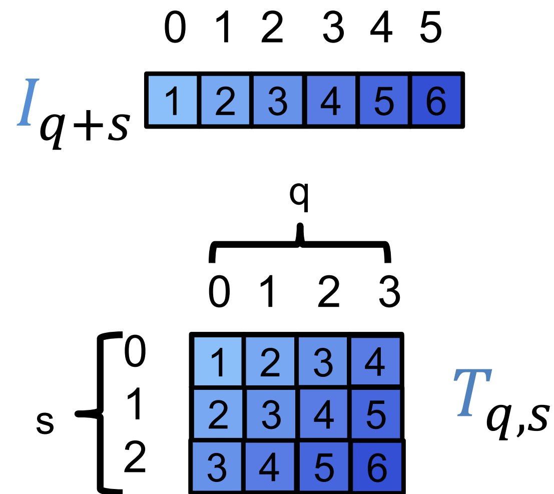
1	2	3	4

Input Fmap ($T_{q,s}$)

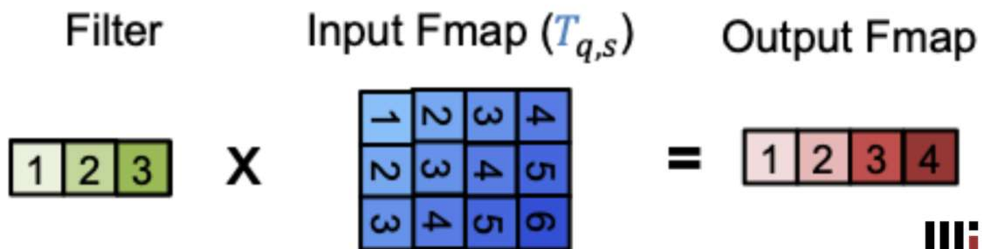
Let's show how the Einsum performs this process…

1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$



q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3
2	0	2
2	1	3
…	…	…



2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Break out Toeplitz conversion

$$T_{c,p,q,r,s} = I_{c,p+r,q+s}$$

Flatten ranks

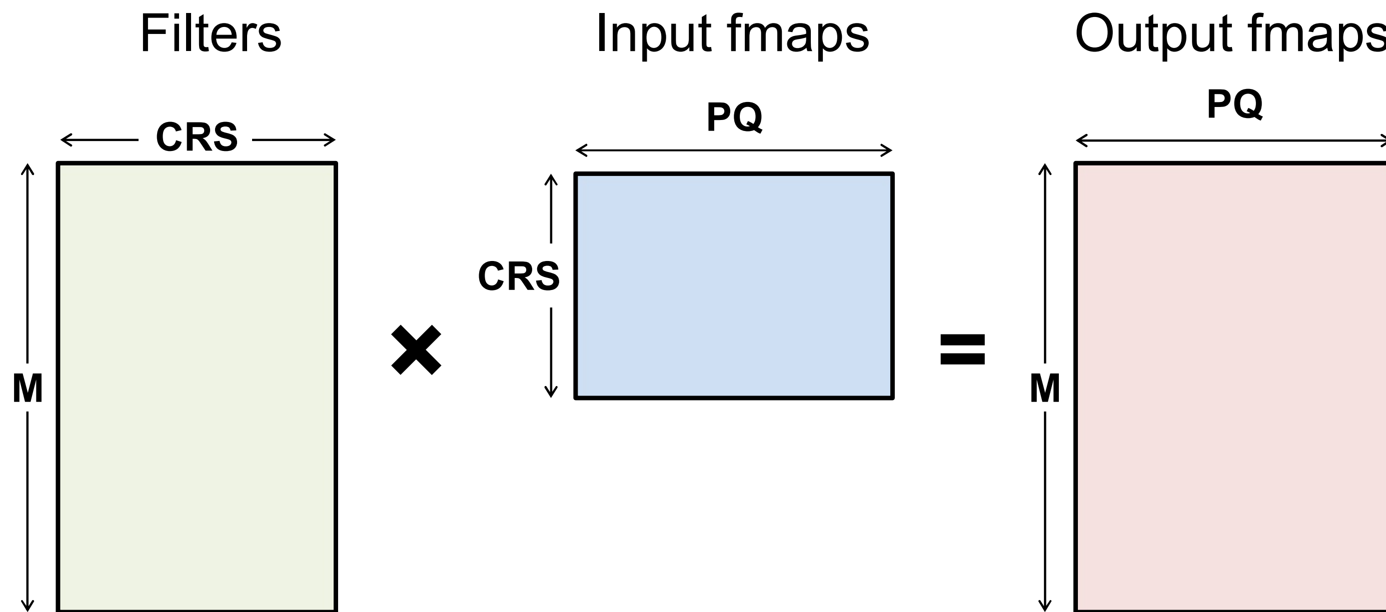
$$T_{pq,crs} \rightarrow T_{c,p,q,r,s}$$

$$F_{m,crs} \rightarrow F_{m,c,r,s}$$

$$O_{m,pq} = T_{pq,crs} \times F_{m,crs}$$

Convolution (CONV) Layer → Matrix Multiplication

$$O_{m,pq} = T_{pq,crs} \times F_{m,crs}$$



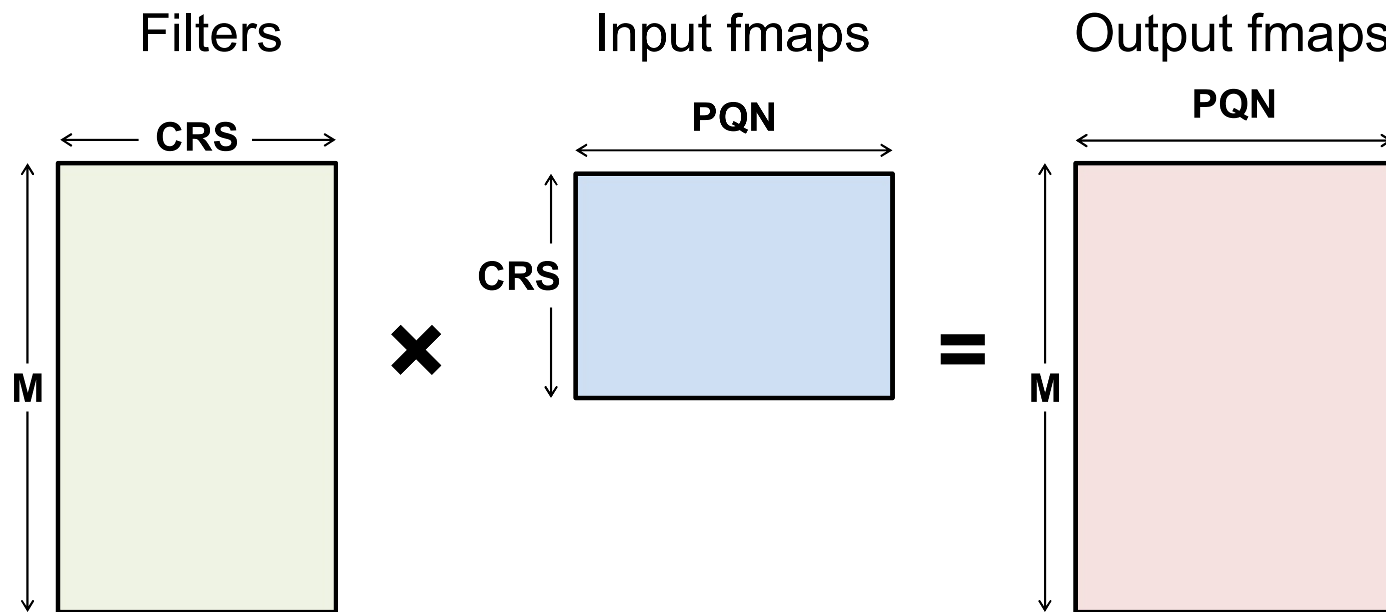
where P=H-R+1 and Q=W-S+1

N=1 in example

Convolution (CONV) Layer → Matrix Multiplication

Include **N** rank

$$O_{m,pqn} = T_{pqn,crs} \times F_{m,crs}$$



where P=H-R+1 and Q=W-S+1

Summary

- We have shown how FC and CONV layers can be converted into matrix multiplication
- There are many ways to compute this matrix multiplication
  - i.e., the MACs can be performed in many different orders and achieve the same output
- However, the processing order impacts hardware metrics such as energy and latency
  - e.g., processing order can affect the amount of data movement

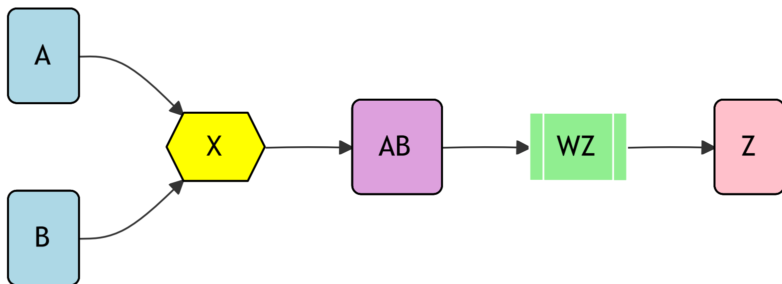
Attention Einsums

Attention Mechanisms as Einsums

Based on collaborations with:

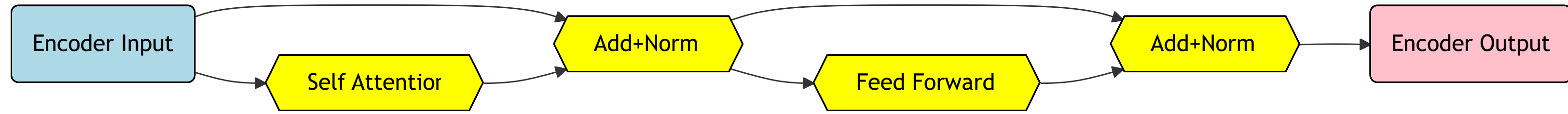
- Alice Wu (UIUC)
- Nandeeka Nayak (UIUC)
- Chris Fletcher (UIUC)
- Michael Pellauer (NVIDIA)

Diagram Conventions

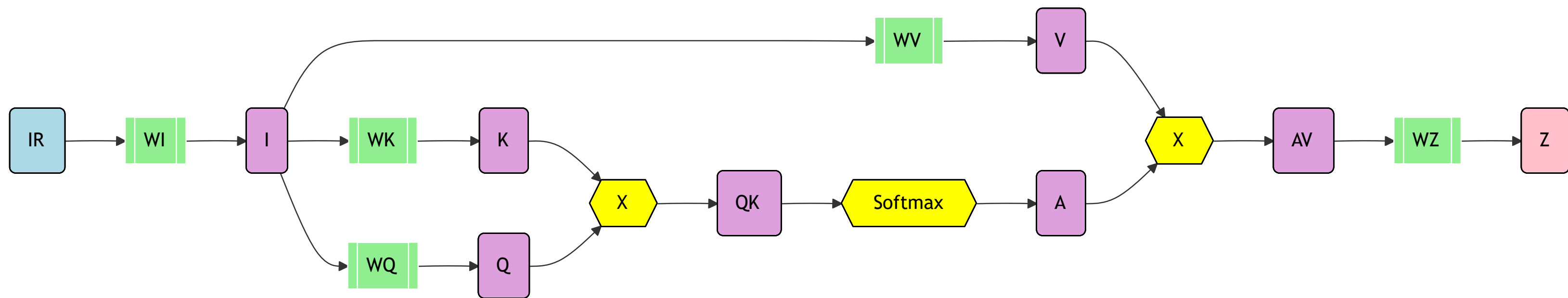


- Rounded box → Tensor
  - Examples: A, B, AB, Z
- Hexagonal box → Operation
  - Examples: $\times$, $softmax$
- Vertical lined box → Projection
  - Examples: $\times WZ$

Overall Encoder Structure



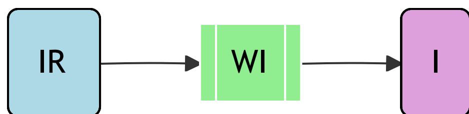
Basic Self-Attention Encoder Computation



Note: Some constant scaling steps are not illustrated.

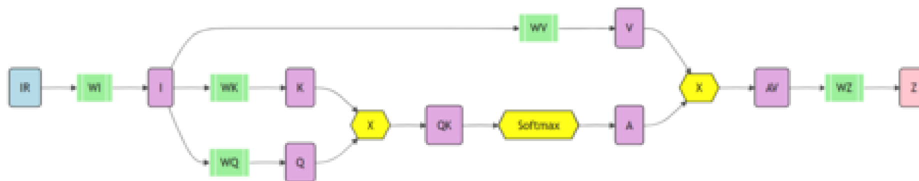
Inputs (Vocabulary Embedding)

Convert word sequence in one-hot representation to tokens in embedding space d.



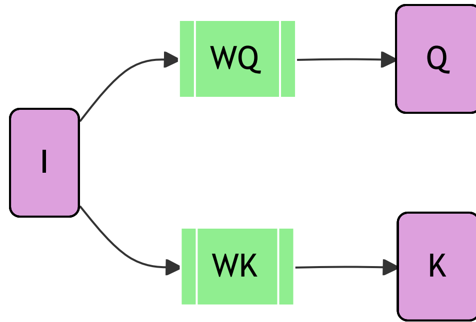
$$I_{m,d} = IR_{m,c} \times WI_{c,d}$$

Note: Only relevant to first attention computation in a chain of transformers



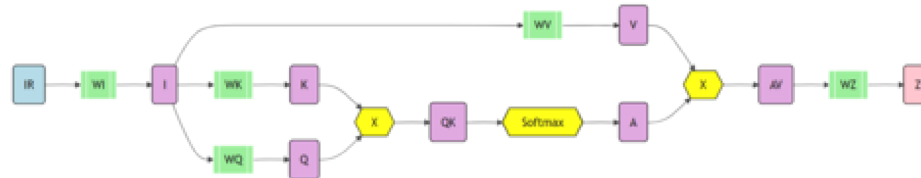
Calculate Key (K) and Query (Q)

Compute projections of I from d-space for "key" (K) and "query" (Q) into e-space



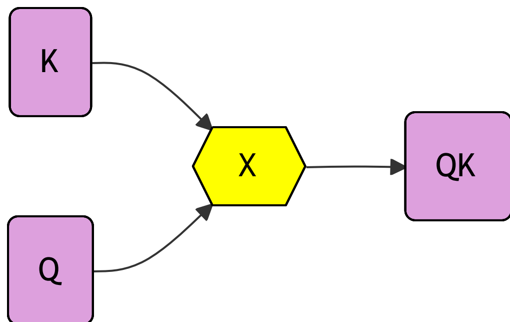
$$Q_{m,e} = I_{m,d} \times WQ_{d,e}$$

$$K_{m,e} = I_{m,d} \times WK_{d,e}$$

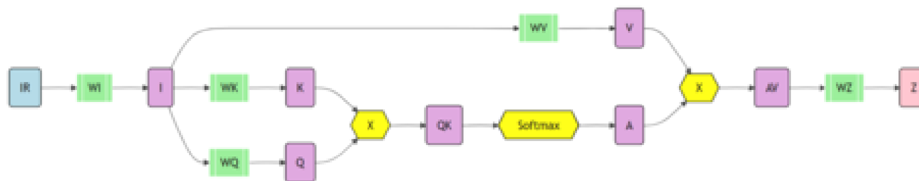


Multiply Key and Query

Create pre-softmax attention matrix



$$QK_{m,p}^{M,P=M} = Q_{p,e}^{M,E} \times K_{m,e}$$



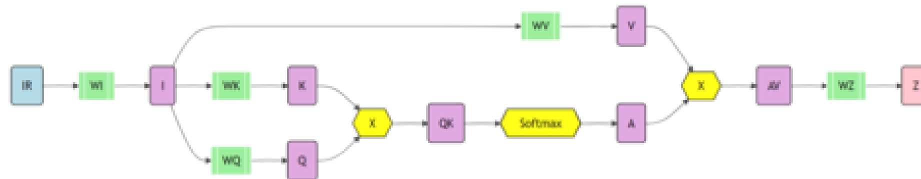
Calculate Softmax

Create numerator (SN) and denominator (SD) for softmax and then do scaling...

$$SN_{m,p} = exp(QK_{m,p})$$

$$SD_p = SN_{m,p}$$

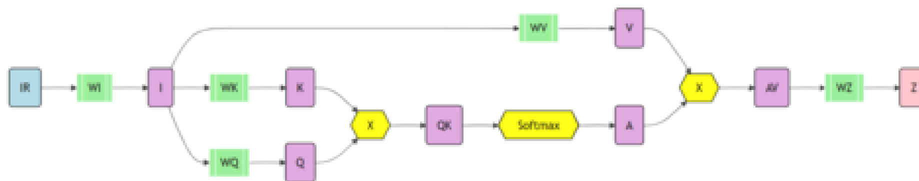
$$A_{m,p} = SN_{m,p}/SD_p$$



Calculate Value (V)

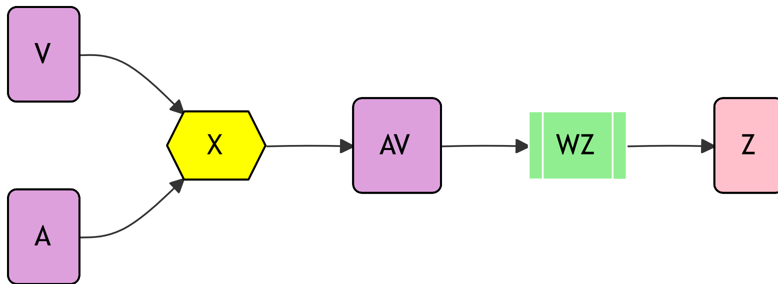
Project I from d-space to f-space to form V.

$$V_{m,f} = I_{m,d} \times WV_{d,f}$$



Create output (Z)

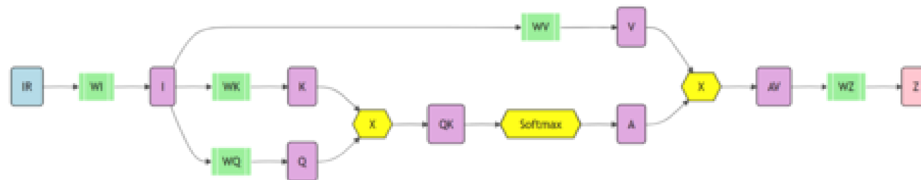
Multiply V and A and project from f-space into g-space



$$AV_{p,f}^{P=M,F} = A_{m,p} \times V_{m,f}$$

$$Z_{p,g} = AV_{p,f} \times WZ_{f,g}$$

Note: Embedding g is often the same as embedding d.



Rank Names

Word/token length ranks

- M - sequence length for the query, key and value in self-attention
- P - alias of sequence length for the rank of QK coming from Q
- R - sequence length for the query in non-self-attention

Vocabulary/embedding ranks

- C - dictionary size (words in vocabulary)
- D - input global space embedding (d_{model})
- E - query and key's local space embedding (d_k)
- F - value's local space embedding (d_v)
- G - output embedding

Other ranks

- B - batch size

Input and Trained Tensors

Input tensors

- $IR^{M,C}$ - raw input (input to first layer only)
- $I^{M,D}$ - input after embedding (input to subsequent layers)

Weight tensors

- $WI^{C,D}$ - Weight tensor to create I
- $WK^{D,E}$ - Weight tensor to create K
- $WQ^{D,E}$ - Weight tensor to create Q
- $WV^{D,F}$ - Weight tensor to create V
- $WZ^{F,G}$ - Weight tensor to create Z

Note: Superscripts are names of the ranks

Principal Intermediate Tensors

Input projections

- $K^{M,E}$ - key
- $Q^{M,E}$ - query
- $V^{M,F}$ - value

Other tensors

- $A^{M,P}$ - Attention tensor
- $Z^{P,G}$ - Output tensor

Other Intermediate Tensors

Product tensors

- $QK^{M,P}$ - $Q \times K$
- $AV^{P,F}$ - $A \times V$

Softmax component tensors

- $SN^{M,P}$ - Softmax numerator
- SD^{P} - Softmax denominator

Full Cascade

$$I_{m,d} = IR_{m,c} \times WI_{c,d}$$

$$K_{m,e} = I_{m,d} \times WK_{d,e}$$

$$Q_{m,e} = I_{m,d} \times WQ_{d,e}$$

$$QK_{m,p}^{M,P=M} = Q_{p,e}^{M,E} \times K_{m,e}$$

$$SN_{m,p} = exp(QK_{m,p})$$

$$SD_{p} = SN_{m,p}$$

$$A_{m,p} = SN_{m,p}/SD_{p}$$

$$V_{m,f} = I_{m,d} \times WV_{d,f}$$

$$AV_{p,f}^{P=M,F} = A_{m,p} \times V_{m,f}$$

$$Z_{p,g} = AV_{p,f} \times WZ_{f,g}$$

Batched Attention

$$I_{b,m,d} = IR_{b,m,c} \times WI_{c,d}$$

$$K_{b,m,e} = I_{b,m,d} \times WK_{d,e}$$

$$Q_{b,m,e} = I_{b,m,d} \times WQ_{d,e}$$

$$QK_{b,m,p}^{B,M,P=M} = Q_{b,p,e}^{B,M,E} \times K_{b,m,e}$$

$$SN_{b,m,p} = exp(QK_{b,m,p})$$

$$SD_{b,p} = SN_{b,m,p}$$

$$A_{b,m,p} = SN_{b,m,p}/SD_{b,p}$$

$$V_{b,m,f} = I_{b,m,d} \times WV_{d,f}$$

$$AV_{b,p,f}^{B,P=M,F} = A_{b,m,p} \times V_{b,m,f}$$

$$Z_{b,p,g} = AV_{b,p,f} \times WZ_{f,g}$$

Multi-headed Attention

(skipping initial embedding step)

$$K_{b,h,m,e} = I_{b,m,d} \times WK_{d,h,e}$$

$$Q_{b,h,m,e} = I_{b,m,d} \times WQ_{d,h,e}$$

$$QK_{b,h,m,p}^{B,H,M,P=M} = Q_{b,h,p,e}^{B,H,M,E} \times K_{b,h,m,e}$$

$$SN_{b,h,m,p} = exp(QK_{b,h,m,p})$$

$$SD_{b,h,p} = SN_{b,h,m,p}$$

$$A_{b,h,m,p} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$V_{b,h,m,f} = I_{b,m,d} \times WV_{d,h,f}$$

$$AV_{b,h,p,f}^{B,H,P=M,F} = A_{b,h,m,p} \times V_{b,h,m,f}$$

$$C_{b,p,h\times F+f}^{B,P=M,G=H\times F} = AV_{b,h,p,f}$$

$$Z_{b,p,d} = C_{b,p,f} \times WZ_{g,d}$$