Improving Power Efficiency for Edge Inferencing Through Memory Management Optimizations

Nathan Levy
Samsung
September 2020

Outline

- Introduction
- Tiling and forwarding
- Filters management
- Case study of different modern convolutional neural networks
- Conclusion

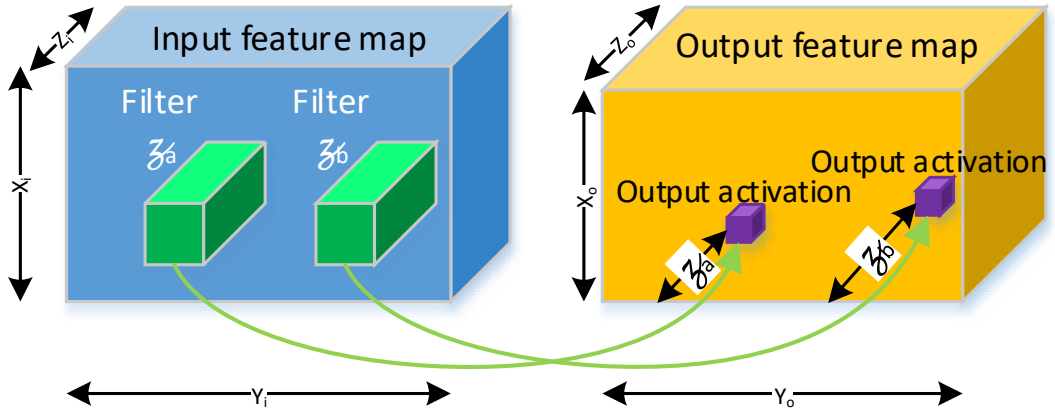
Counting the I/O in the TOps per Watt KPI

- Power consumption is composed of:
  - Compute power: perform the arithmetic computations
  - I/O power: bring the data to the compute engine and back
- For convolutional neural networks, I/O power can be higher than compute
- **The compiler must optimize the memory management**
- Scope: convolutional neural network (CNN) acceleration
  - Focus on convolutions
  - Assume the network is fixed (and quantized, pruned…)
  - Inference only

Description of the working model

What is a convolution?

- Input: feature map and filters
- Apply: multiply and accumulate
- Output: feature map



CNN can be processed by CPU, DSP, GPU, NPU/TPU

- These often have multiple cores
- These have multiple levels of memories (cache and scratch-pad)

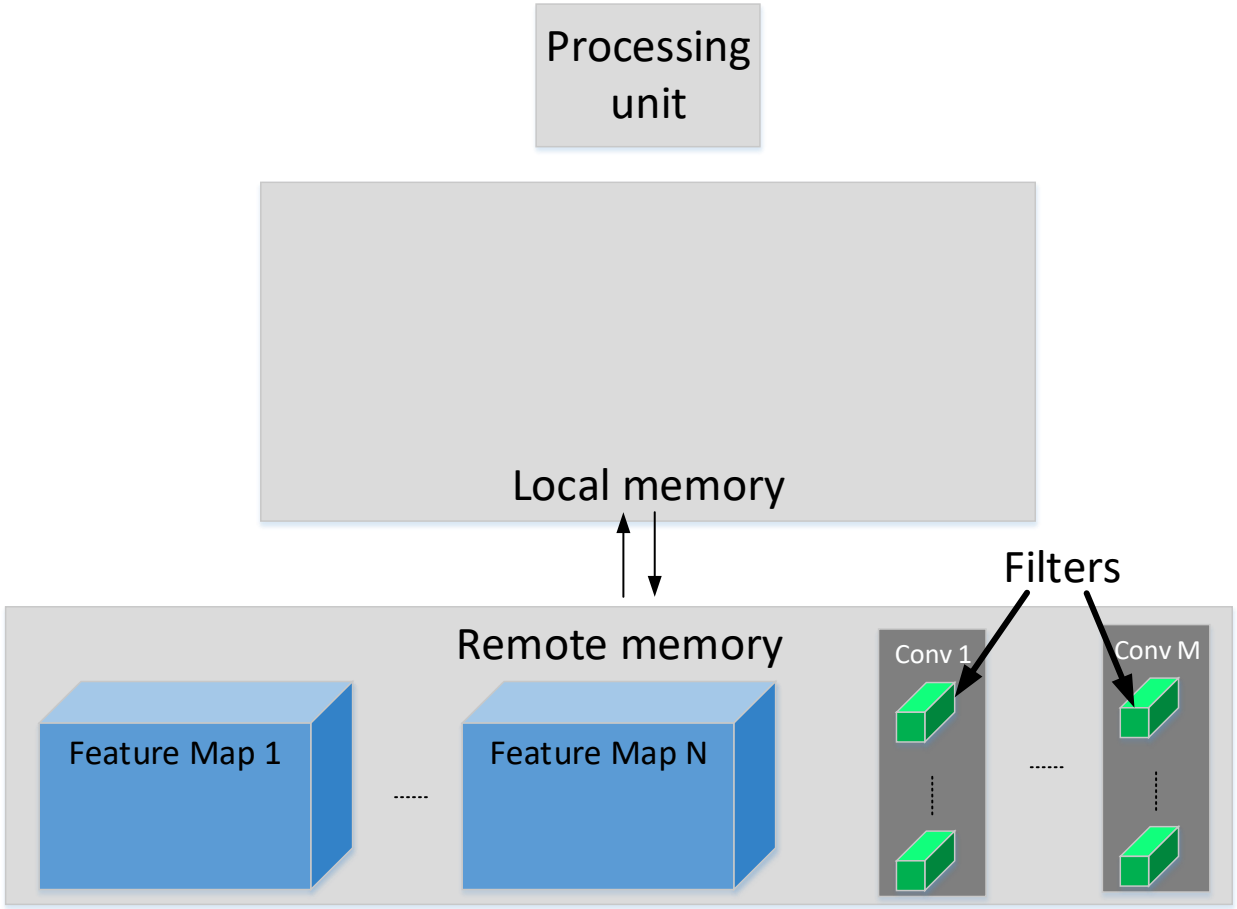
Description of the working model

Simplified toy example:

- Single processing core
- Small local memory with “free” access and big remote memory with expensive access

Remote memory contains:

- Input and output feature maps
- Convolution filters
- Intermediate feature maps (if needed)

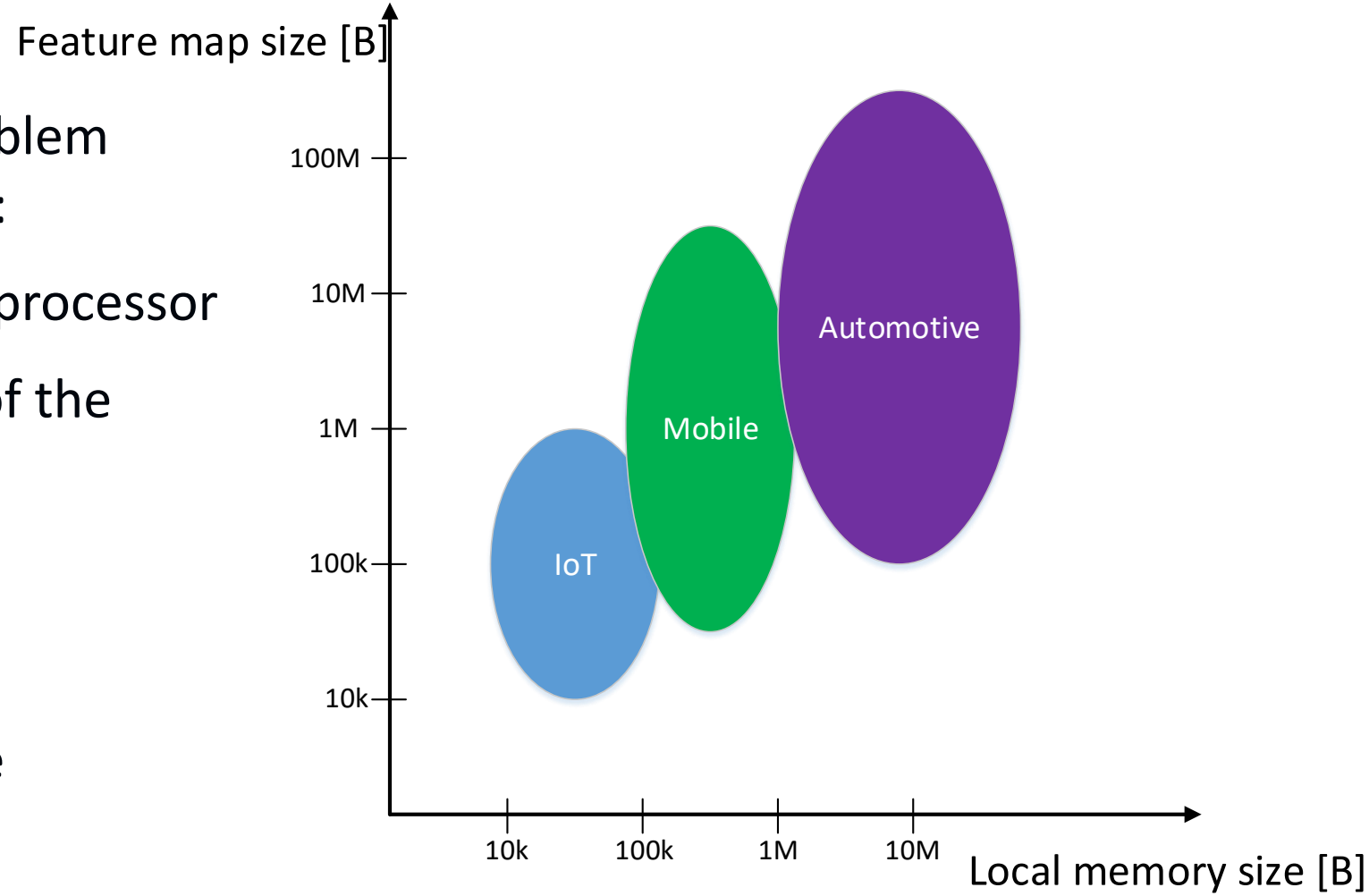


Typical orders of magnitude for NPU

- The memory management problem depends on the relative size of:
  - The local memory of the processor
  - The typical feature map of the network
- Our working point:

  Feature map size ~

  10 * Local memory size



Tiling and forwarding

SAMSUNG

Tiling and Forwarding

How to overcome the small memory size:

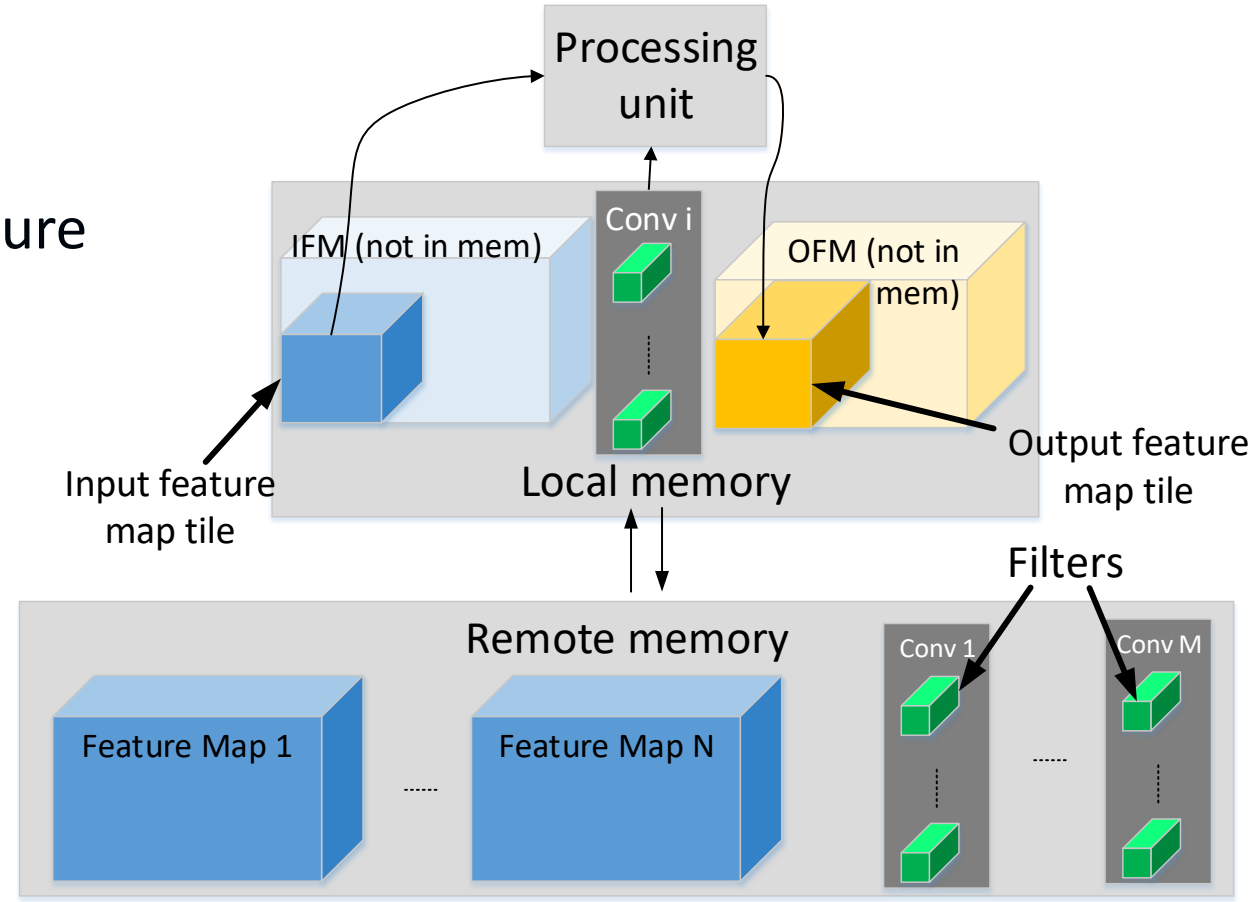
- Split the input feature map into smaller parts → **tiles**
- Apply the convolution on the input feature map tile

  → Creates an output feature map tile

The output feature tile can be either:

- Written in remote memory
- Used for the next convolution

→ This is called **forwarding**



How to tile and forward

How does the compiler tile and forward to minimize power consumption?

Key optimization parameters:

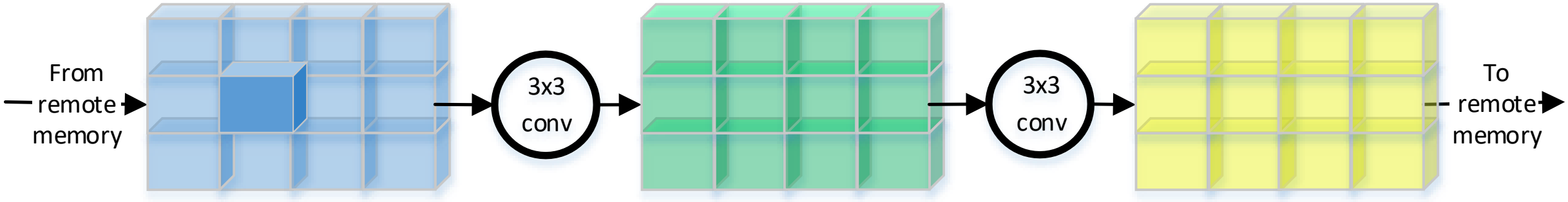
- Power consumption: bandwidth (I/O) + compute

  → **Focus of this talk**

- Processing throughput (operations/second)
- Chip area → cost
- System complexity, flexibility and maintenance

Tiling and increasing receptive fields

2 convolutions with 3x4 tiling. Without access to remote memory in the middle (forwarding)

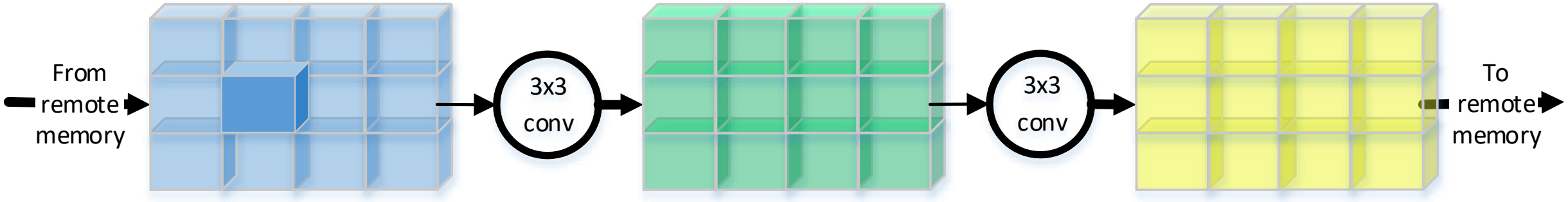


For i in [1-12]:

- Load tile i of blue feature map from remote memory
- Run 1st convolution → tile i of green feature map
- Run 2nd convolution → tile i of yellow feature map
- Store tile i of yellow feature map to remote memory

Tiling and increasing receptive fields

2 convolutions with 3x4 tiling. Without access to remote memory in the middle (forwarding)

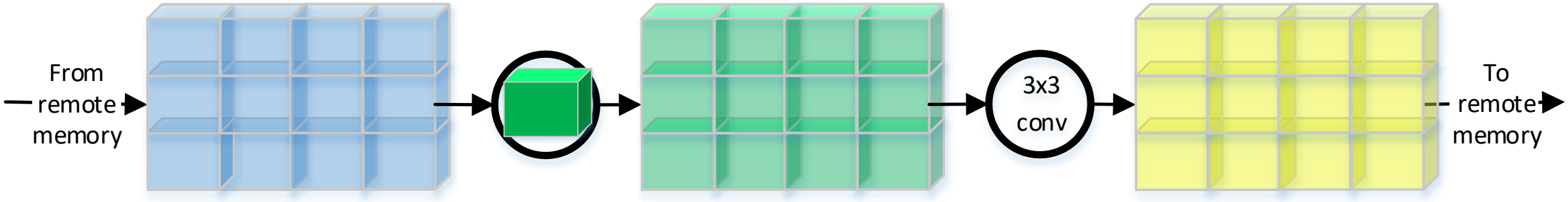


For i in [1-12]:

- Load tile i of blue feature map from remote memory
- Run 1st convolution → tile i of green feature map
- Run 2nd convolution → tile i of yellow feature map
- Store tile i of yellow feature map to remote memory

Tiling and increasing receptive fields

2 convolutions with 3x4 tiling. Without access to remote memory in the middle (forwarding)

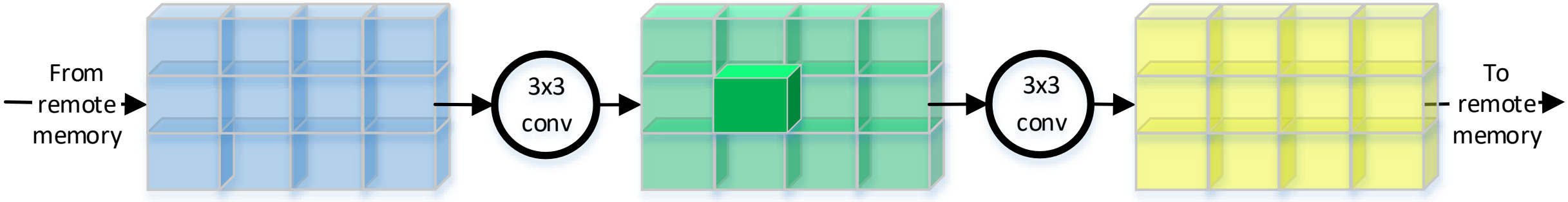


For i in [1-12]:

- Load tile i of blue feature map from remote memory
- Run 1st convolution → tile i of green feature map
- Run 2nd convolution → tile i of yellow feature map
- Store tile i of yellow feature map to remote memory

Tiling and increasing receptive fields

2 convolutions with 3x4 tiling. Without access to remote memory in the middle (forwarding)

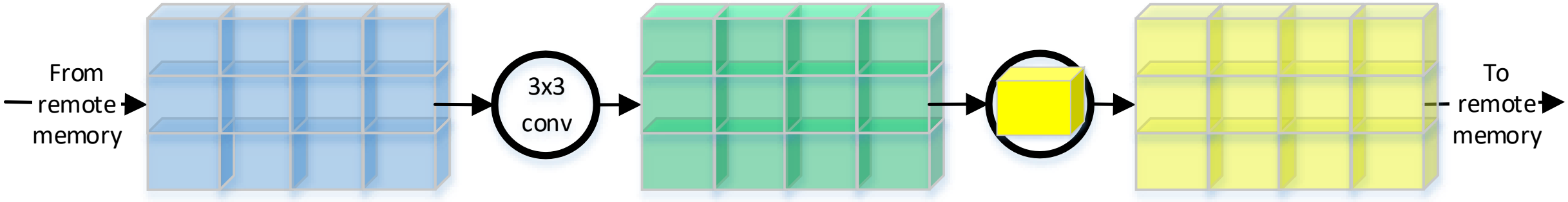


For i in [1-12]:

- Load tile i of blue feature map from remote memory
- Run 1st convolution → tile i of green feature map
- Run 2nd convolution → tile i of yellow feature map
- Store tile i of yellow feature map to remote memory

Tiling and increasing receptive fields

2 convolutions with 3x4 tiling. Without access to remote memory in the middle (forwarding)

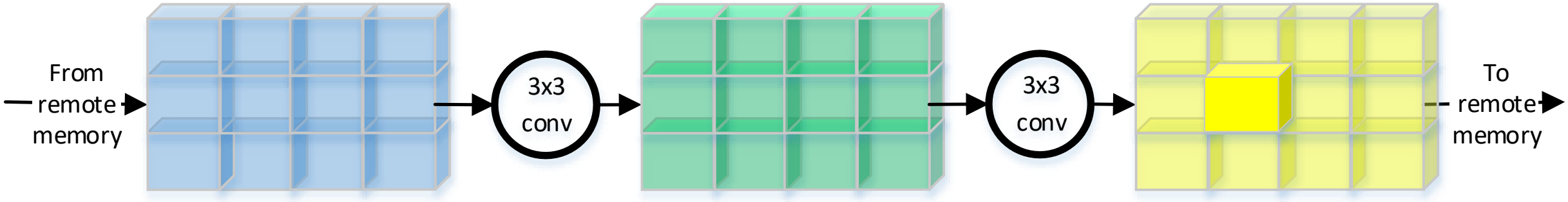


For i in [1-12]:

- Load tile i of blue feature map from remote memory
- Run 1st convolution → tile i of green feature map
- Run 2nd convolution → tile i of yellow feature map
- Store tile i of yellow feature map to remote memory

Tiling and increasing receptive fields

2 convolutions with 3x4 tiling. Without access to remote memory in the middle (forwarding)

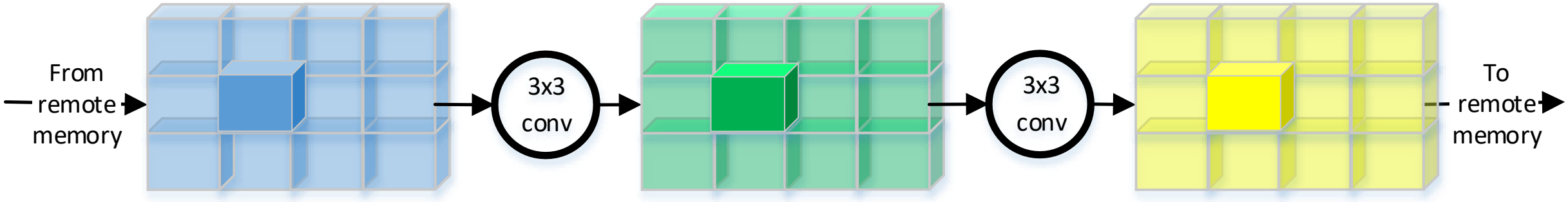


For i in [1-12]:

- Load tile i of blue feature map from remote memory
- Run 1st convolution → tile i of green feature map
- Run 2nd convolution → tile i of yellow feature map
- Store tile i of yellow feature map to remote memory

Tiling and increasing receptive fields

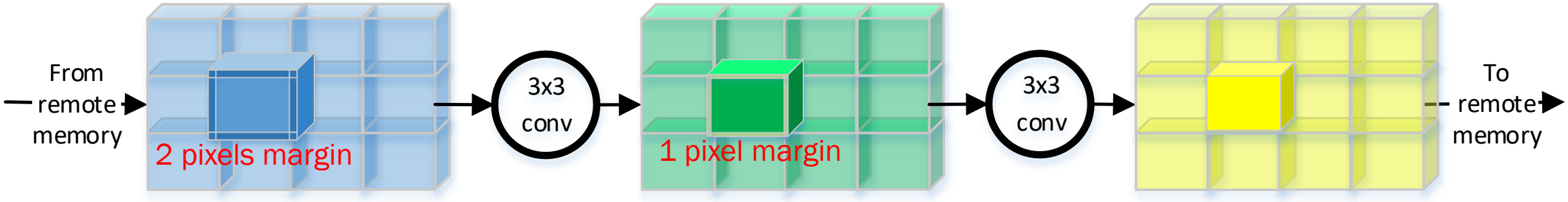
2 convolutions with 3x4 tiling. Without access to remote memory in the middle (forwarding)



Problem: 3x3 convolution requires margins

Tiling and increasing receptive fields

2 convolutions with 3x4 tiling. Without access to remote memory in the middle (forwarding)

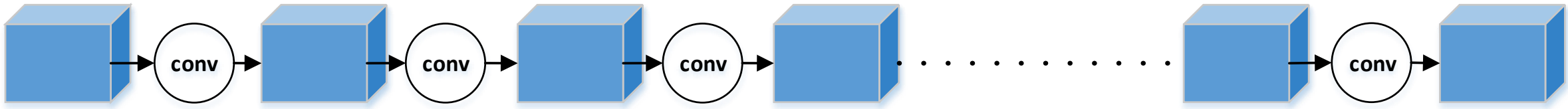


Solution: Bigger blue and green tiles

Problems:

- Load more data for the first (blue) layer
- Compute more data for the intermediate (green) layer

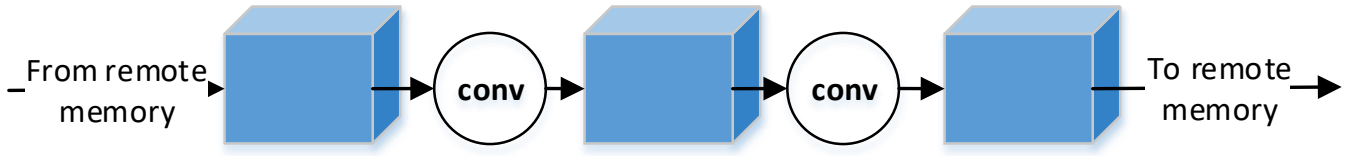
Tiling and increasing receptive fields



Consider a long chain of convolutions. How many stages of forwarding do you want?

- Forwarding tiles through as many layers as possible avoids dumping to remote memory
- This causes the overlaps to grow
  - More data to load for the first layer
  - Redundant processing

2 convolutions



# of Convolutions	Feature Map Tiling	Bandwidth	Bandwidth per convolution	Operations per convolution	Energy per convolution
2	3x4	+6%	-47%	+5%	-30%
3	3x	+12	-63%	10%	-39
4		+	1%		
5	4x4	+29%	-74%	+26%	-42%

Break the feature map into 12 to fit in memory

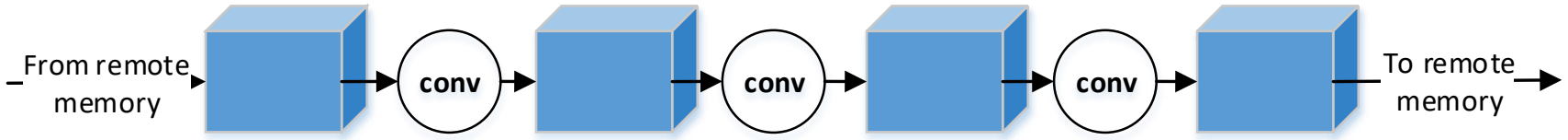
Load the overlaps

Compute the overlaps

Less I/O
More compute

All numbers are compared to “no forwarding”

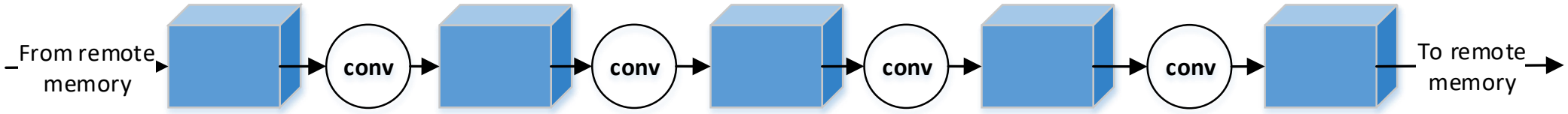
3 convolutions



# of Convolutions	Feature Map Tiling	Bandwidth	Bandwidth per convolution	Operations per convolution	Energy per convolution
2	3x4	+6%	-47%	+5%	-30%
3	3x4	+12%	-63%	+10%	-39%
4	3x4	+17%	-71%	+16%	-43%
5	4x4	+29%	-74%	+26%	-42%

All numbers are compared to “no forwarding”

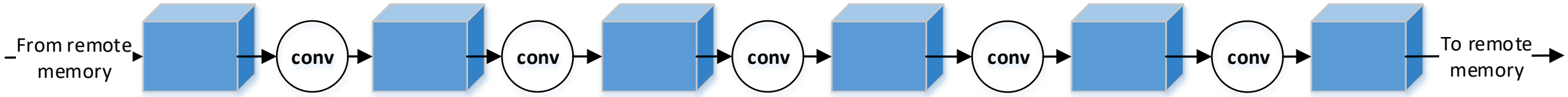
4 convolutions



# of Convolutions	Feature Map Tiling	Bandwidth	Bandwidth per convolution	Operations per convolution	Energy per convolution
2	3x4	+6%	-47%	+5%	-30%
3	3x4	+12%	-63%	+10%	-39%
4	3x4	+17%	-71%	+16%	-43%
5	4x4	+29%	-74%	+26%	-42%

All numbers are compared to “no forwarding”

5 convolutions



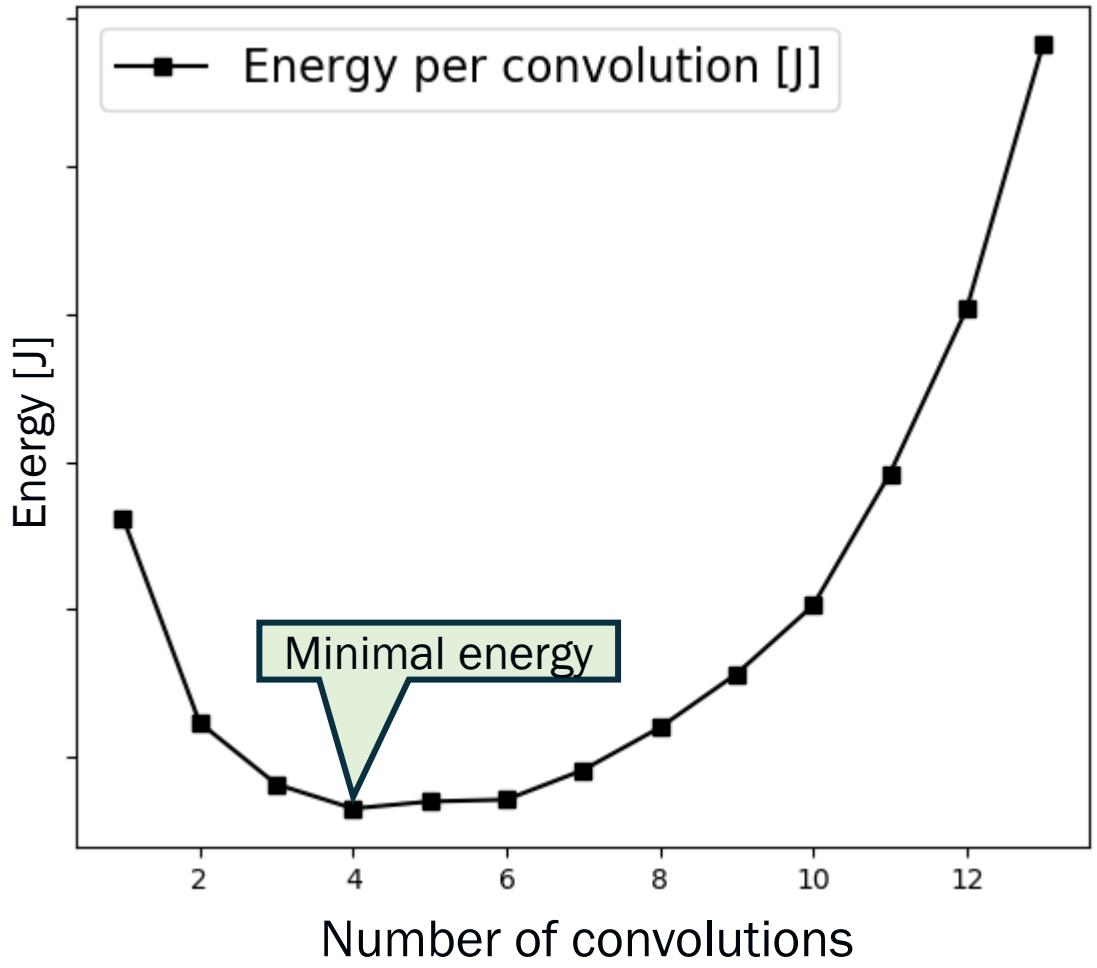
# of Convolutions	Feature Map Tiling	Bandwidth	Bandwidth per convolution	Operations per convolution	Energy per convolution
2		+6%	-47%	+5%	-30%
3		+12%	-63%	+10%	-39%
4	3x4	+17%	-71%	+16%	-43%
5	4x4	+29%	-74%	+26%	-42%

Overlaps are so big we need smaller tiles to fit in local memory

All numbers are compared to “no forwarding”

Tiling and increasing receptive fields

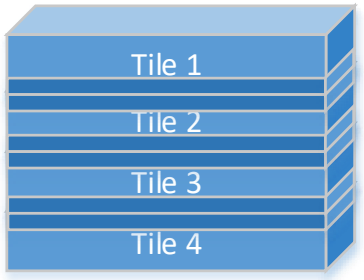
- The trade-off between I/O and compute power leads to a U-shaped behavior
- The trade-off requires an optimization mechanism in the compiler
- In our toy example, all the convolutions are identical
- In a real CNN, the convolutions are different, so the optimization problem is not one-dimensional

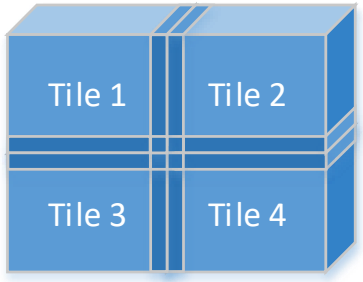


Tiling and increasing receptive fields

More constraints to take into account:

- Some hardware will prefer vertical tiling for memory access
  - Increases overlap sizes





- Some hardware will impose constraints on tile size (multiple of a given number)

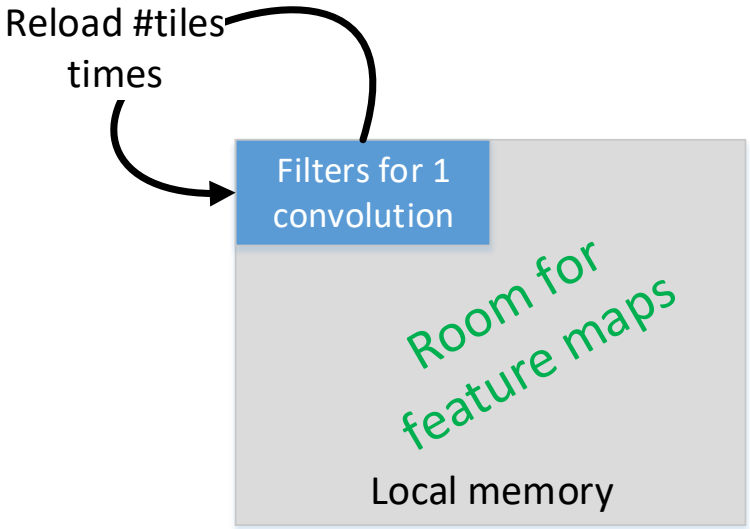
Out-of-scope ways to deal with overlaps:

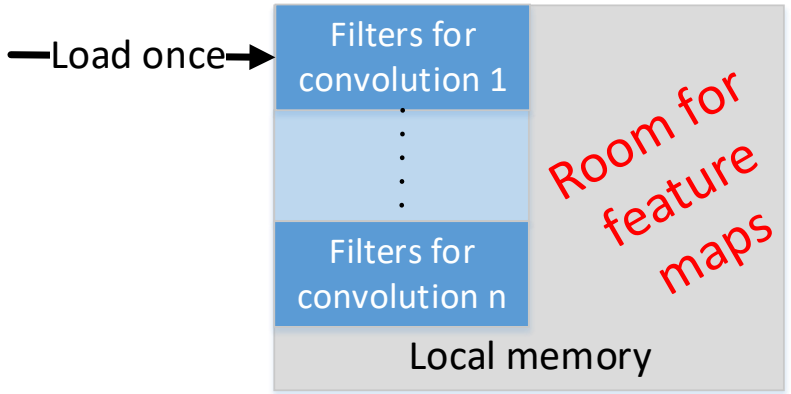
- Multi-core processing → hardware considerations and system complexity
- Keep overlap data instead of loading/computing again: takes up memory and requires memory fragmentation management → system complexity

Filters management

SAMSUNG

Filters management





Option 1: reload the filters from remote to local memory for each tile

→ wasted bandwidth

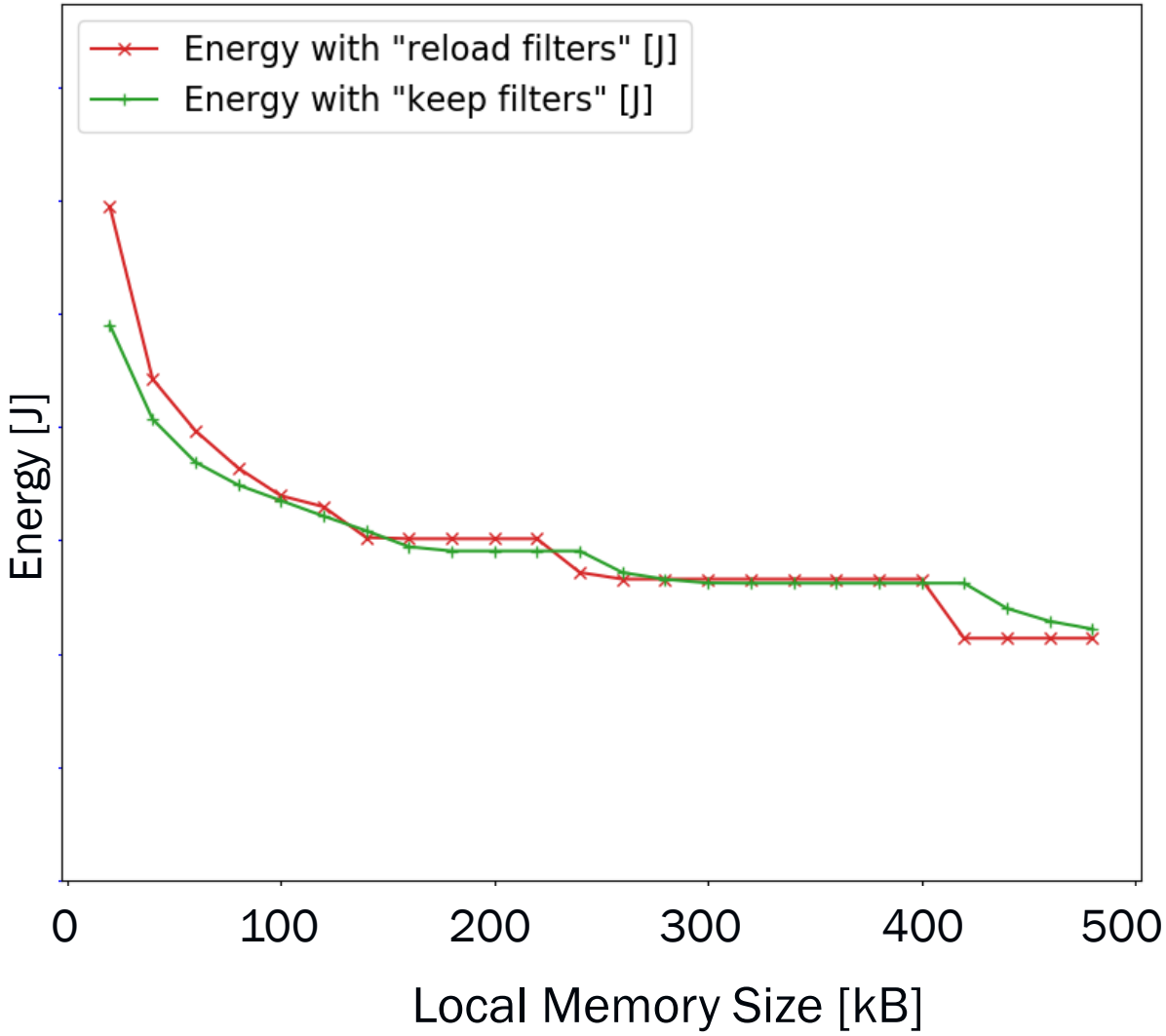
Option 2: keep the filters of all convolutions in local memory while processing all the tiles

→ wasted space in the local memory

→ may increase the tiles number

Filters management

- No single strategy is the best all the time
- Choosing the right strategy can save up to 20% energy
- In practice, the problem is even more complicated due to different convolutions through the network
- The compiler should take the decision to keep/reload the filters for each convolution independently



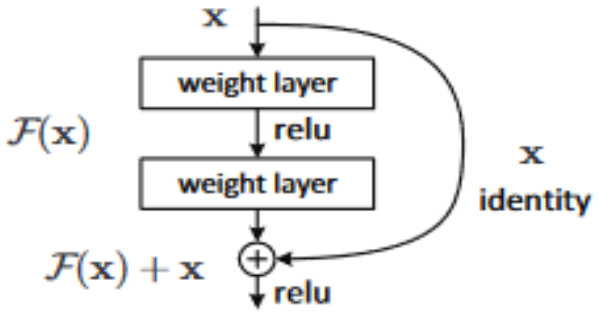
Case study of different modern convolutional neural networks

SAMSUNG

Skip connections – residual block

Skip connections were introduced in the ResNet paper as a way to facilitate back propagation

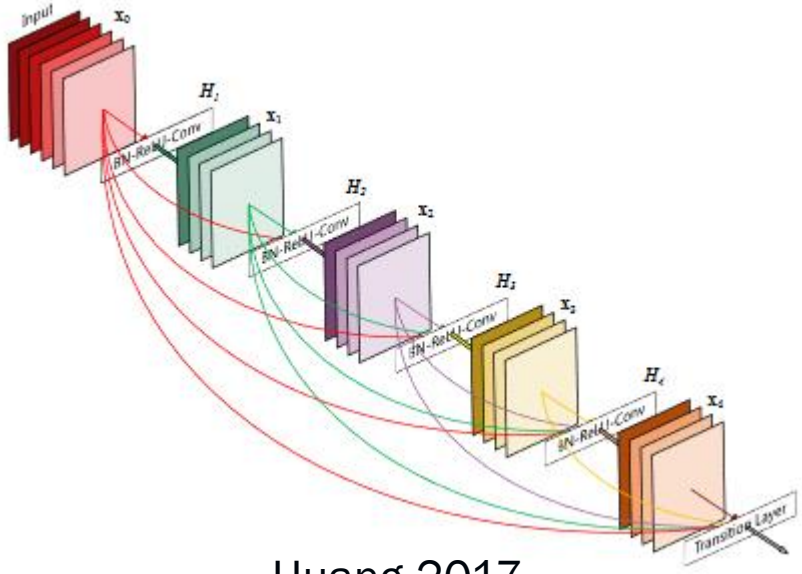
- **Forwarding** becomes more challenging: one strives to keep both main path and skip connection in local memory



He 2016

Intensive usage in DenseNet

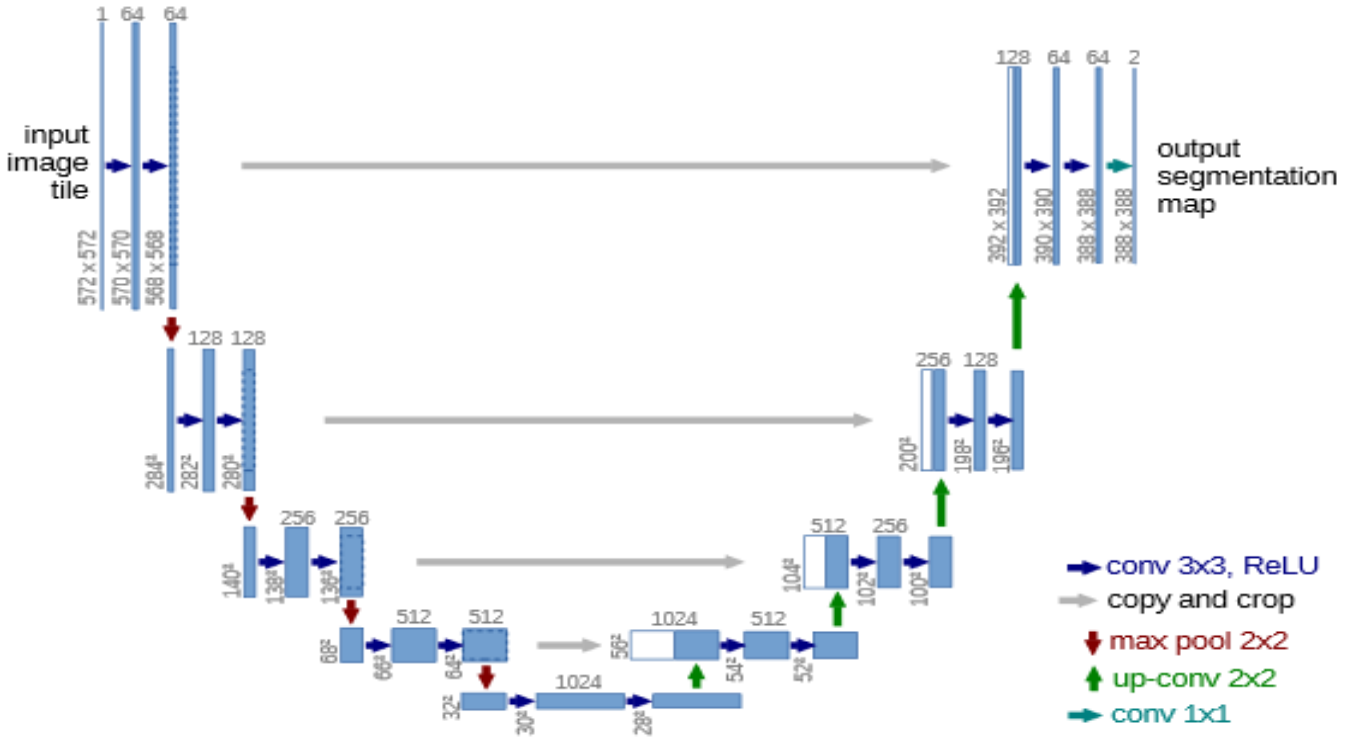
- Up to 6 skip connections alive simultaneously



Huang 2017

Skip connections – segmentation networks

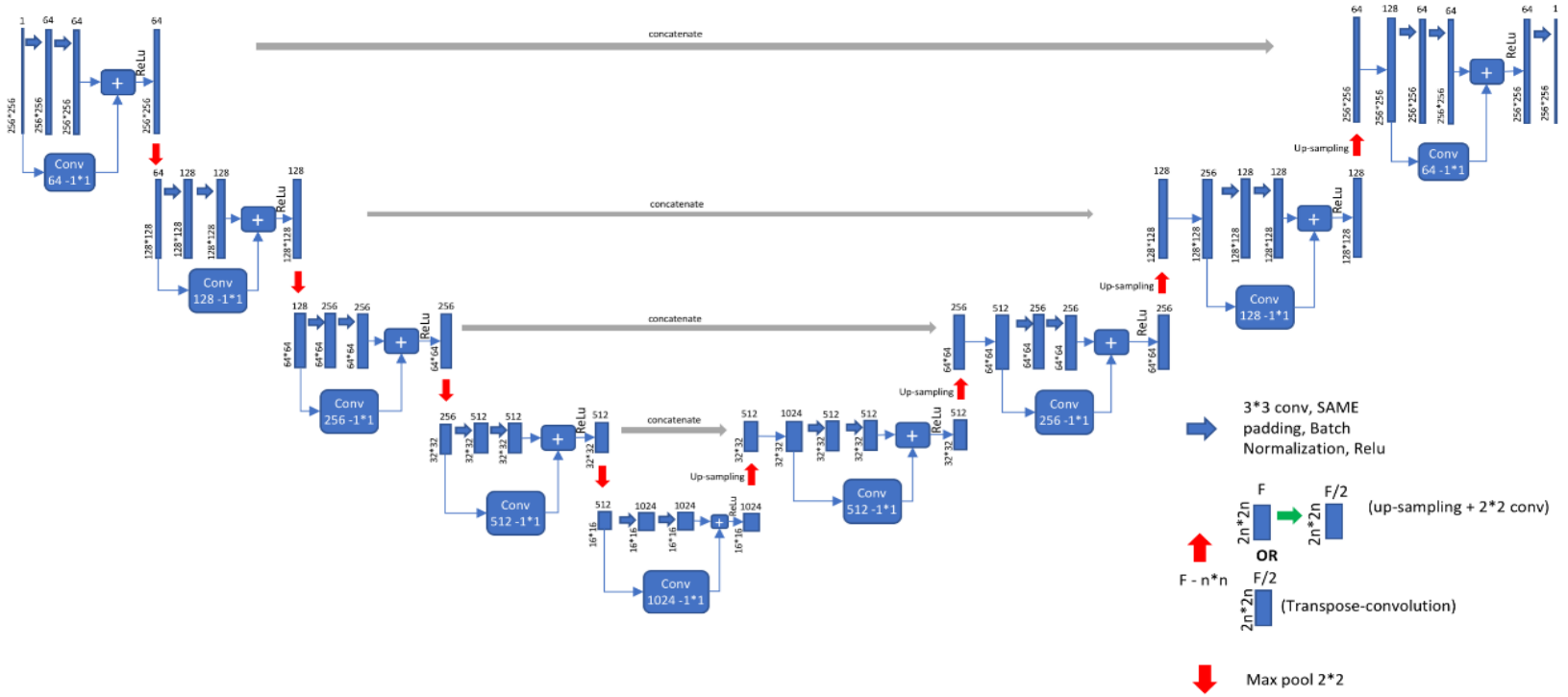
U-net adds skip connections to encoder-decoder structure to improve quality of segmentation



Ronneberger 2015.

Skip connections – segmentation networks

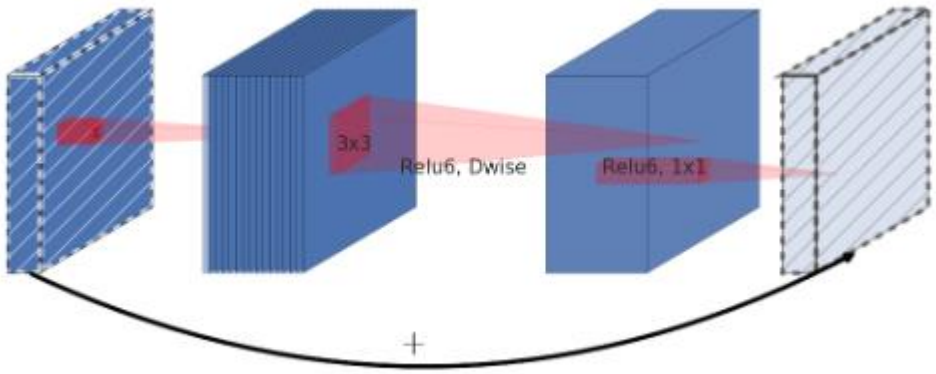
It can be mixed with residual blocks



Singla 2019

Inverted bottleneck

- Inverted bottleneck introduced in MobileNet family
- Splitting the 3x3 convolution into 3x3 depthwise convolution + 1x1 convolution reduces the amount of parameters
  - Can more easily be kept in memory and/or reloaded
  - 1x1 convolutions don't create overlaps
- Skip connection on thinner feature map to minimize memory impact



Sandler 2018

Conclusion

- Memory management is a complicated optimization problem
- The compiler affects many performance indices, in particular power and runtime
- The tradeoff is not intuitive and requires some evaluation mechanism

How do we do it?

- Map all the performance indices (power, runtime…) to a single cost
    - The mapping may depend on the use case
- The compiler is able to analyze its decisions and evaluate the cost
- The compiler optimizes memory management to minimize the cost

Conclusion

What we didn't cover:

- Pre-fetching for runtime reduction
- Multicore edge devices
- We described fixed hardware and fixed network. We actually co-design:
  - Memory size, processing throughput, hardware limitations
  - Network design to avoid access to remote memory
    - Use of network architecture search
  - Quantization and pruning to avoid access to remote memory
    - Choice of bit-width and pruning rate per layer

Resources

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

Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *International Conference on Medical image computing and computer-assisted intervention*. Springer, Cham, 2015.

https://github.com/Nishanksingla/UNet-with-ResBlock

Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.