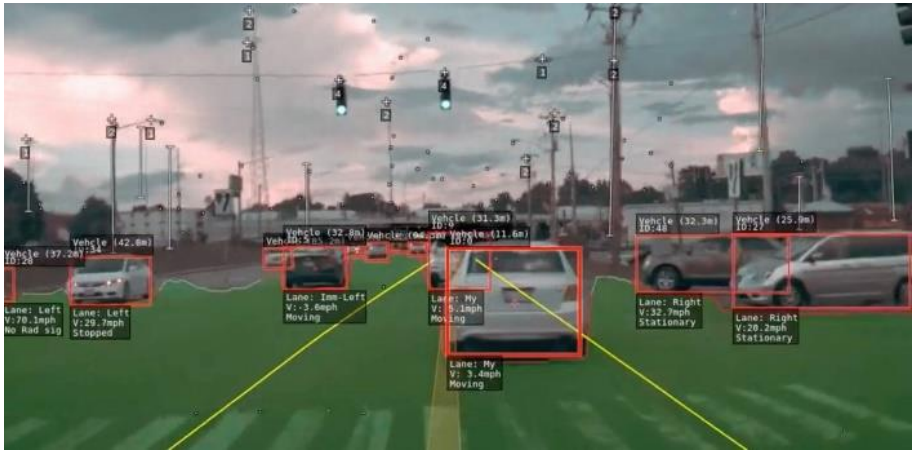
CNN-Inference on CGRAs under Throughput Constraints

Christian Heidorn, Michael Witterauf, Frank Hannig, and Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Motivation: CNNs in Autonomous Driving

- Real-time constraints
- Common Accelerators
  - Bigger batch sizes
- Batchsize 1
  - Low latency for one image
  - High throughput (frames per second)
  - GOPS/W



Source: https://electrek.co/2018/10/15/tesla-new-autopilot-neural-net-v9/

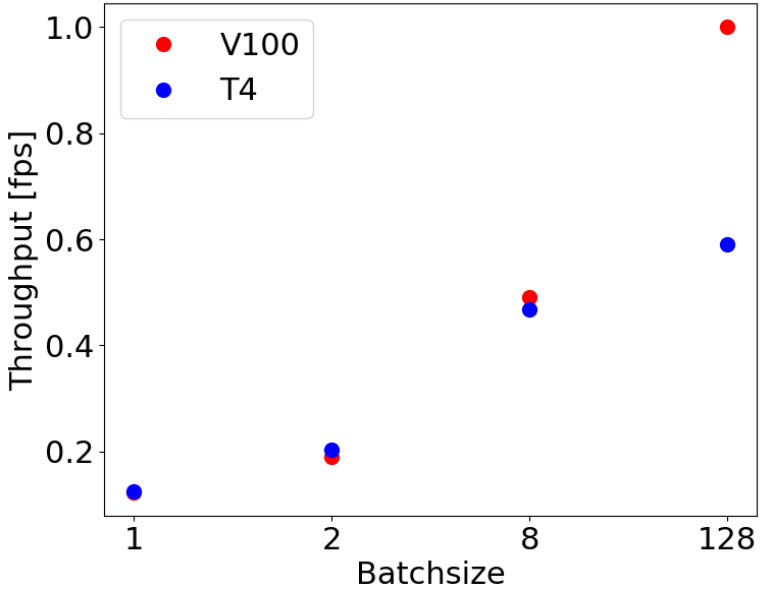
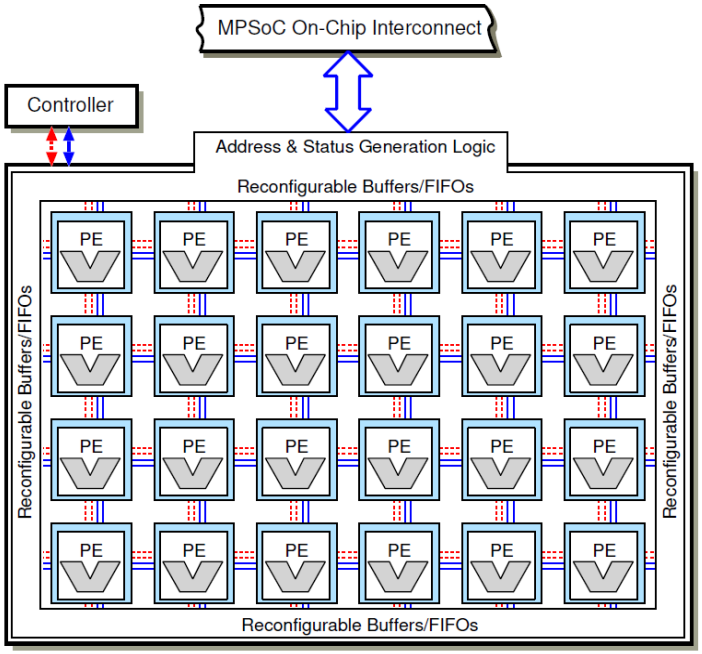


Image per Second (Normalized) for mobileNetv1 on Teslas T4 vs.Nvidias Volta
Values taken from: https://developer.nvidia.com/deep-learning-performance-training-inference

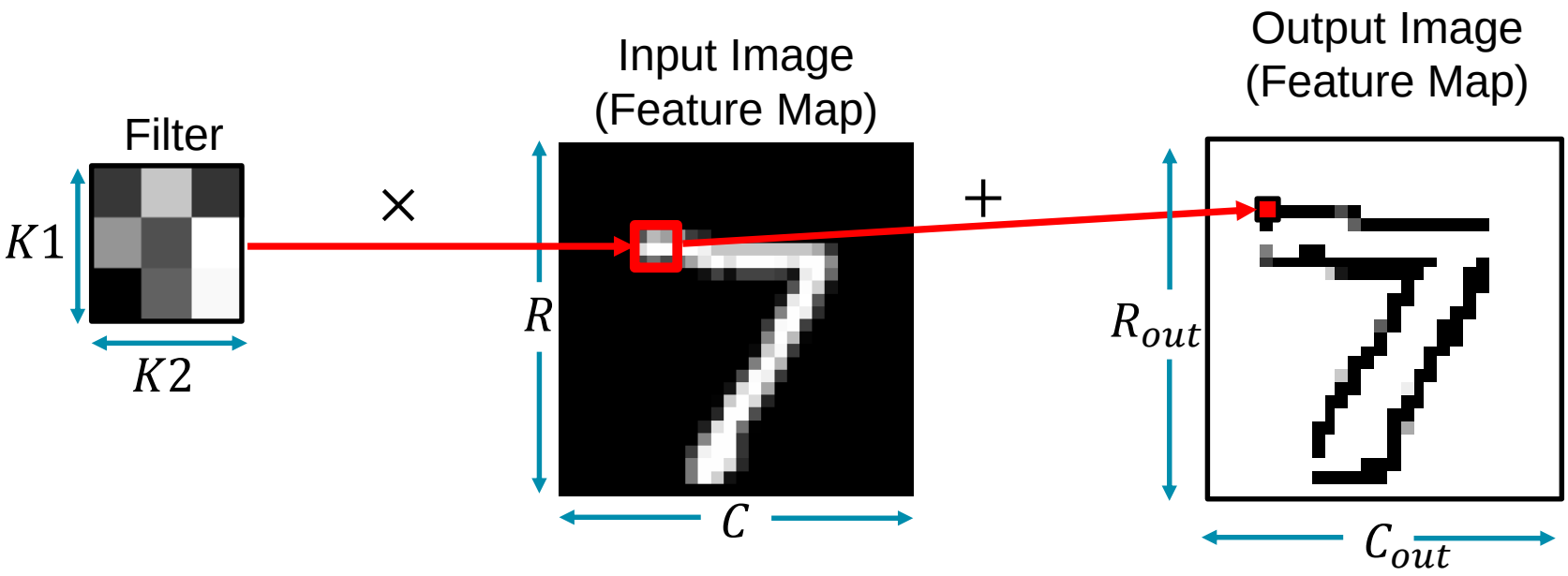
Coarse-Grained Reconfigurable Arrays (CGRAs)

- Class of massively parallel processor architectures
  - Array of processing elements (PEs)
  - Stand-alone or accelerator
- Flexibility at runtime
  - Reconfigurable at functional unit level
  - Programmable, reconfigurable interconnect
- Applications
  - Image processing, object recognition
  - Linear algebra (matrix / vector computations)
  - . . . and other streaming applications



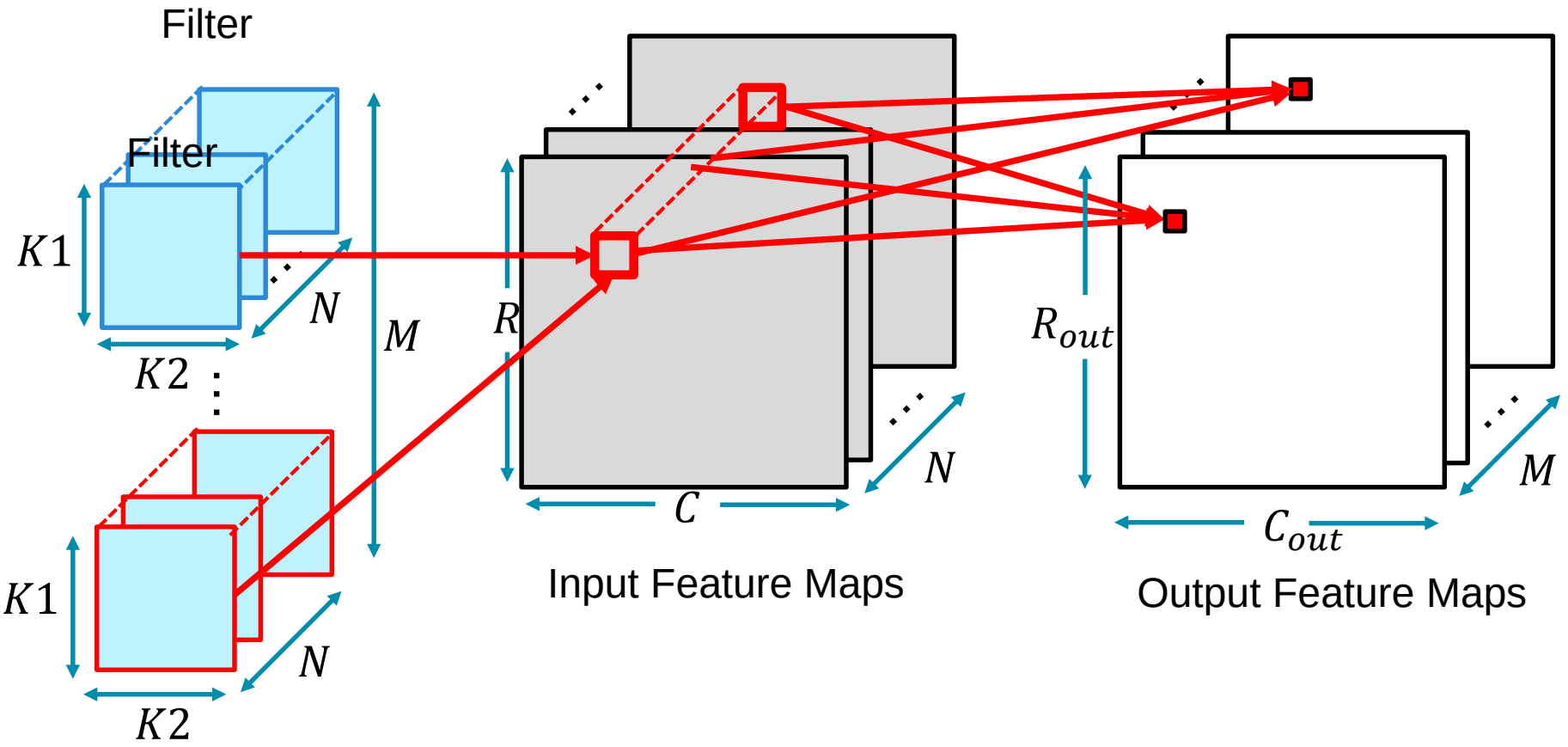
Contents

Convolutions in CNNs



- Elementwise multiplication of learned filter parameters (weights) with image pixels
- Accumulation of the partial sums → Output pixel

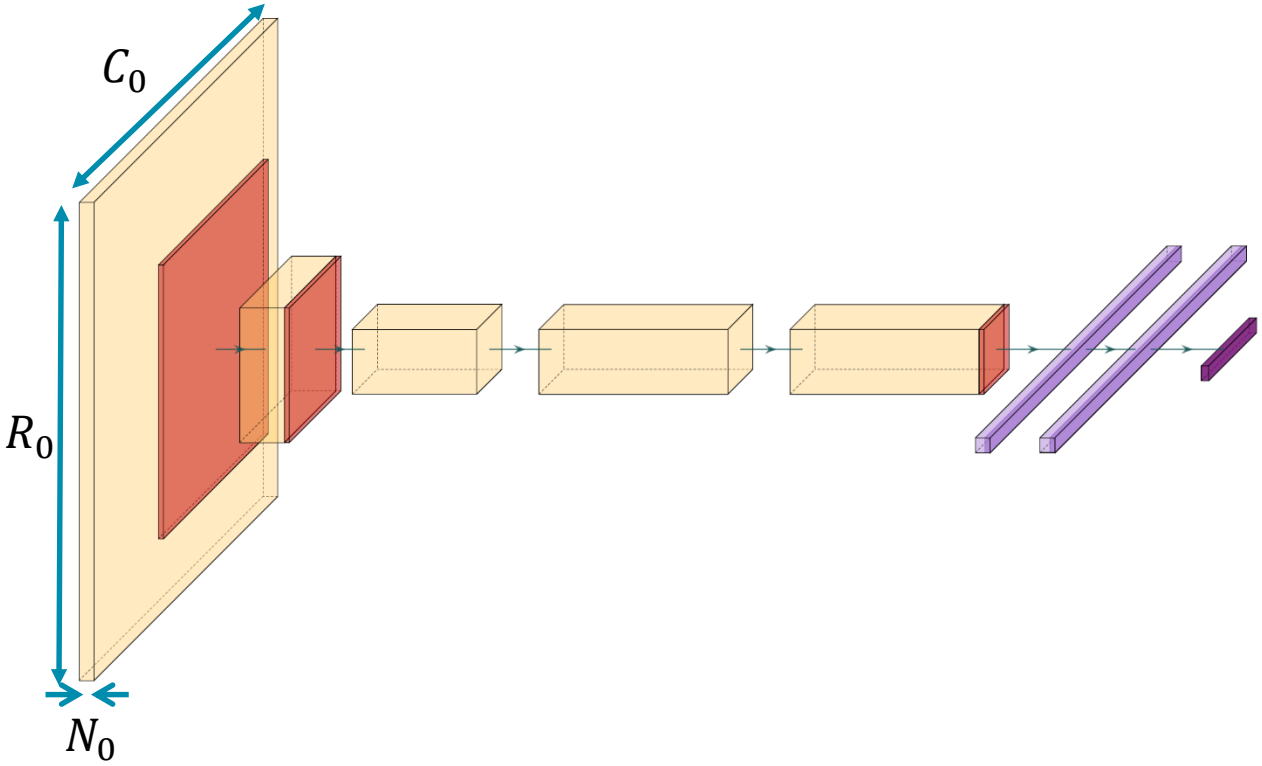
Convolutions in CNNs



- Many input Feature Maps (N)
- Many Filters → Many output Feature Maps (M)

Convolutional Neural Network (CNN)

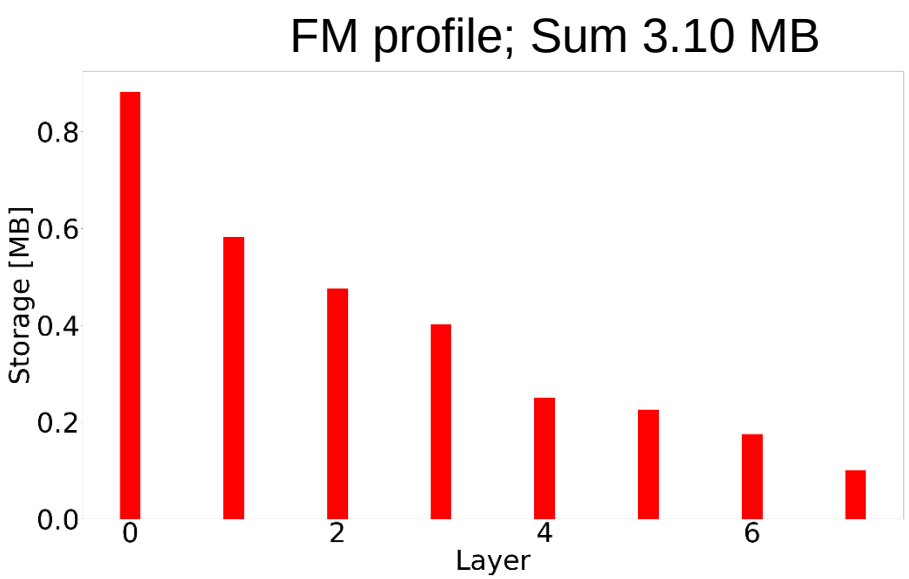
Conv0: $M_i \times N_i \times K1_i \times K2_i$
Pool1: $Kp1_i \times Kp2_i, S1, S2$
Conv2: $M_i \times N_i \times K1_i \times K2_i$
Pool3: $Kp1_i \times Kp2_i, S1, S2$
Conv4: $M_i \times N_i \times K1_i \times K2_i$
Conv5: $M_i \times N_i \times K1_i \times K2_i$
Conv6: $M_i \times N_i \times K1_i \times K2_i$
Pool7: $Kp1_i \times Kp2_i, S1, S2$
FC8: $M_i \times N_i$
FC9: $M_i \times N_i$
FC10: $M_i \times N_i$



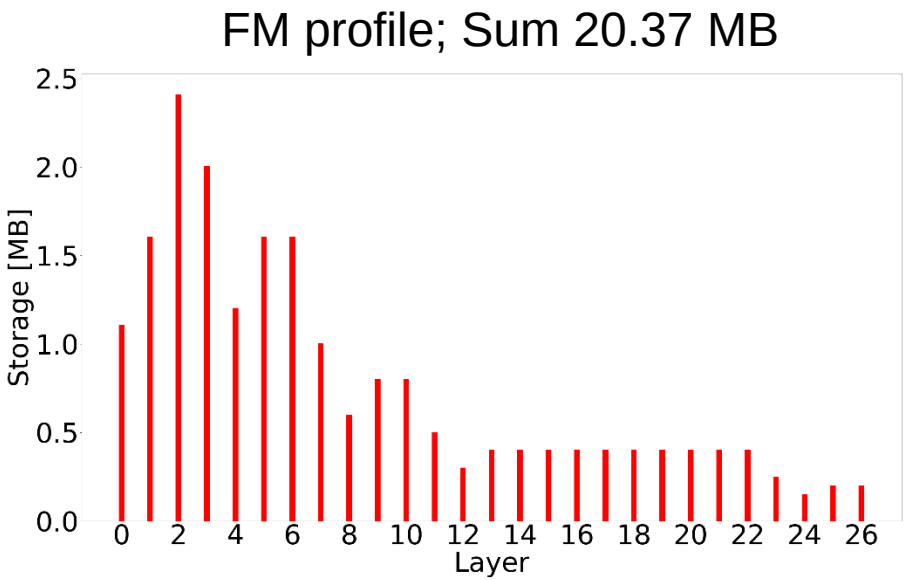
- Layers are stacked on each other
- Different orders, types, and parameter sizes

Contents

CNNs burdens – AlexNet vs. MobileNet



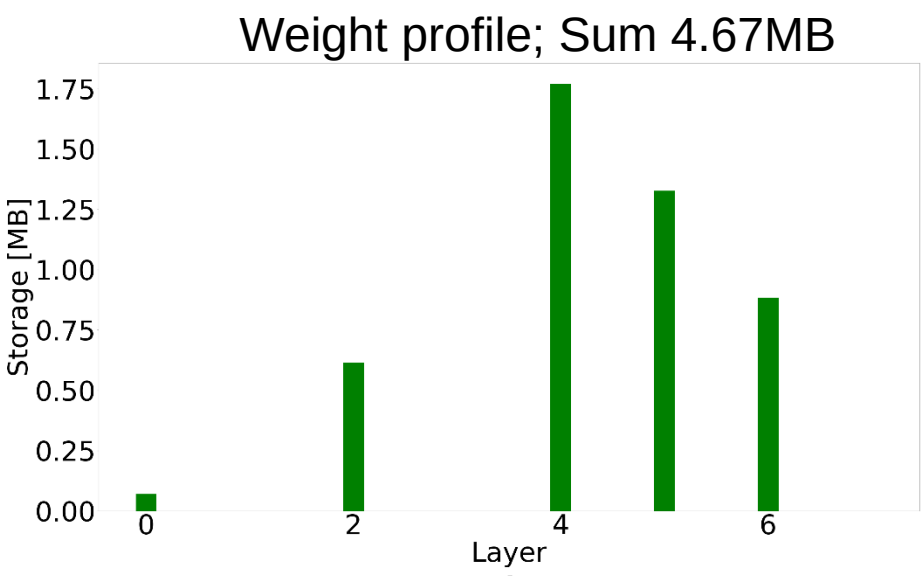
a) AlexNet



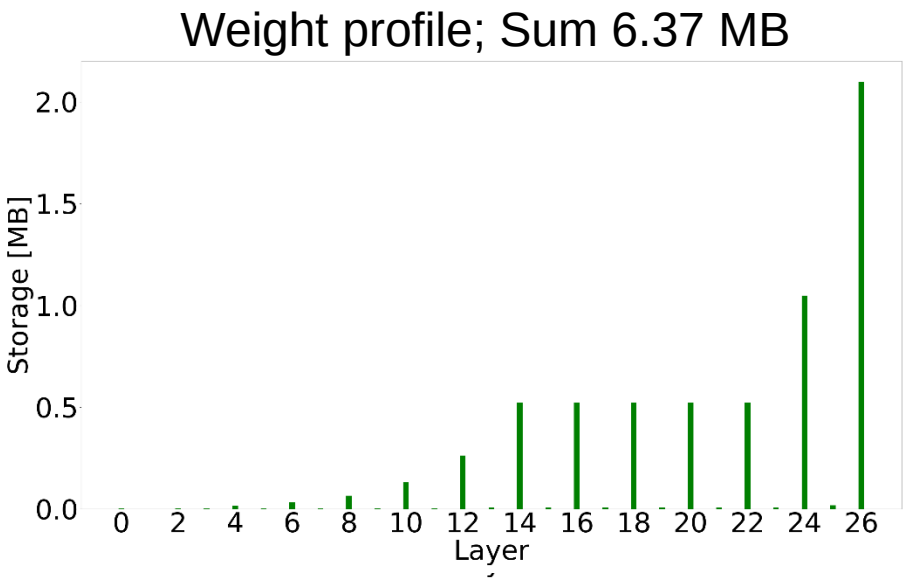
b) MobileNetV1

- Increasing number of layers
- Different Workload (Feature map size and weight size) for each layer (operations, layer shapes)

CNNs burdens – AlexNet vs. MobileNet



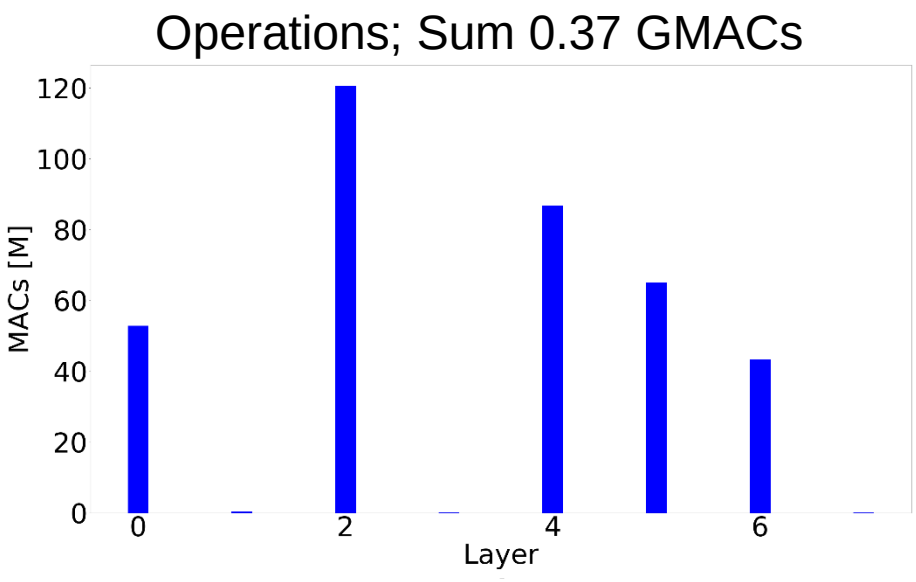
a) AlexNet



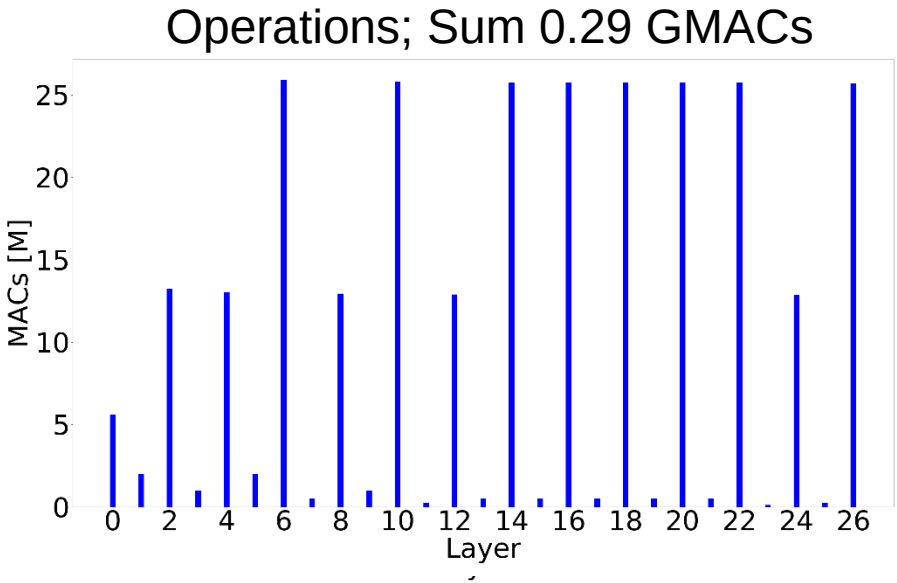
b) MobileNetV1

- Increasing number of layers
- Different Workload (Feature map size and weight size) for each layer (operations, layer shapes)

CNNs burdens – AlexNet vs. MobileNet



a) AlexNet



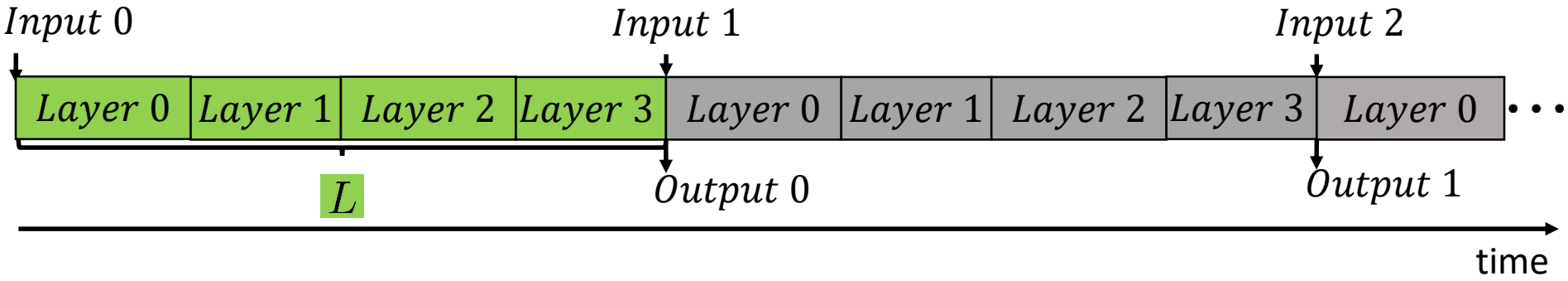
b) MobileNetV1

- Observation #1: Increasing Feature Map size with more layers appearing in CNNs
- Observation #2: MACs for each layer vary significantly

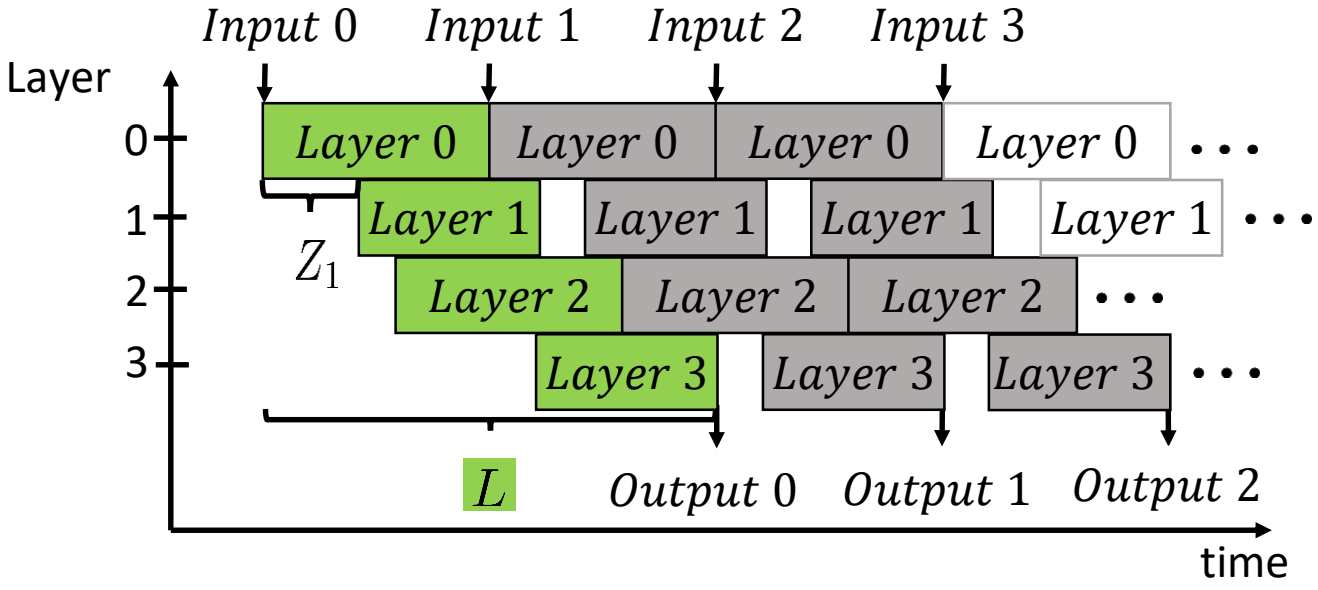
  → full resource utilization hard to achieve

Layer-by-layer vs. Layer-parallel processing

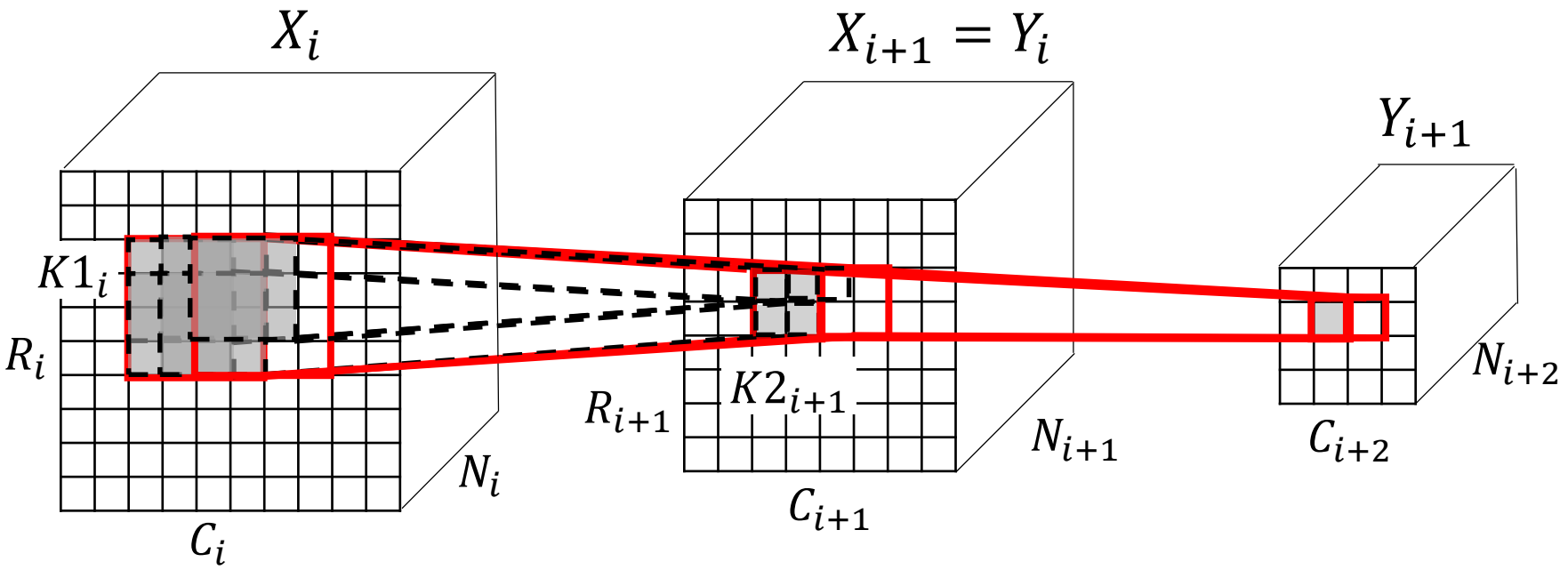
- Layer-by-layer execution



- Layer-parallel execution

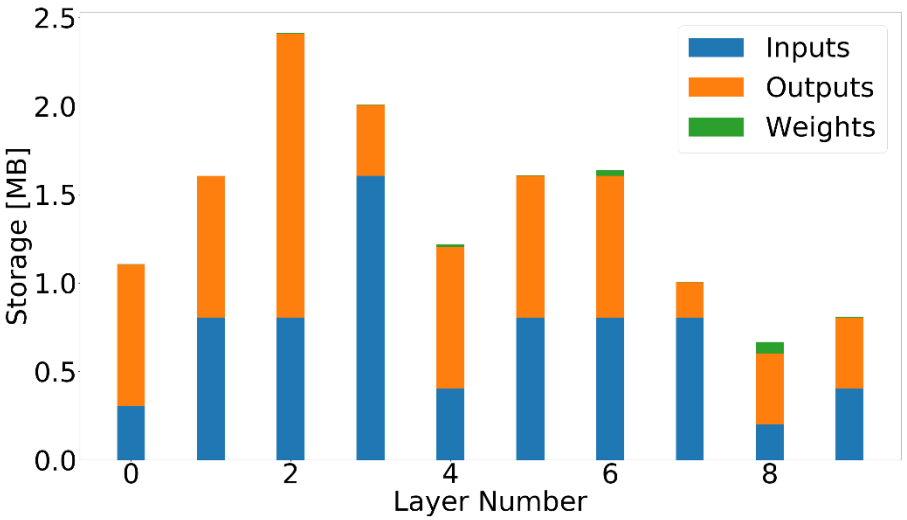


Principle of Layer Fusion

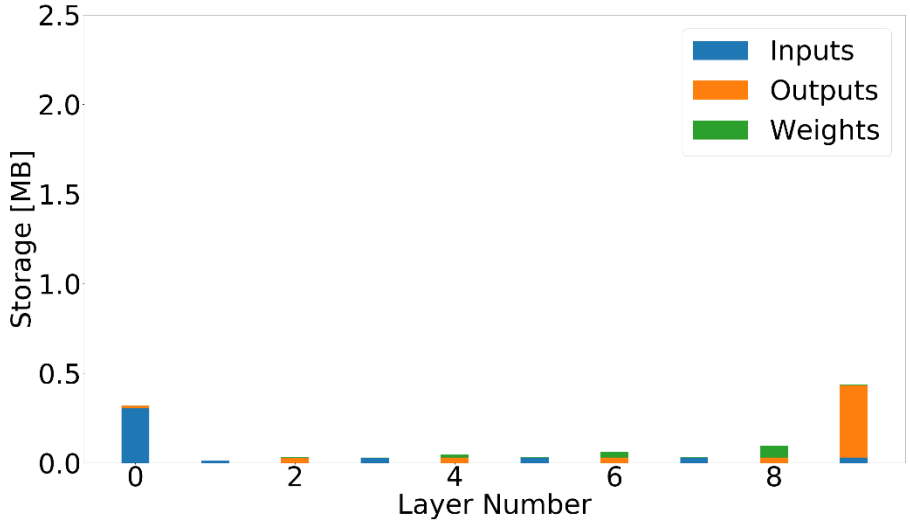


- Idea: Reduce storage and data transfers for feature maps
- Observation: Each successive layer depends on a small area (denoted as *receptive field*) of the previous one
- I.e., 1 value in y_{i+1} depends on the values in a 4x4 receptive field of the input x_i

Storage reduction via layer-parallel execution



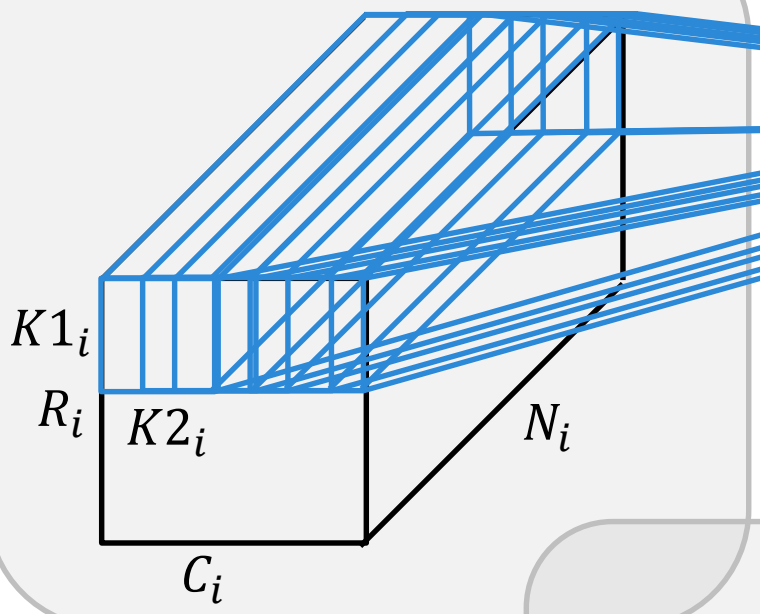
a) MobileNetv1, Layer-by-layer

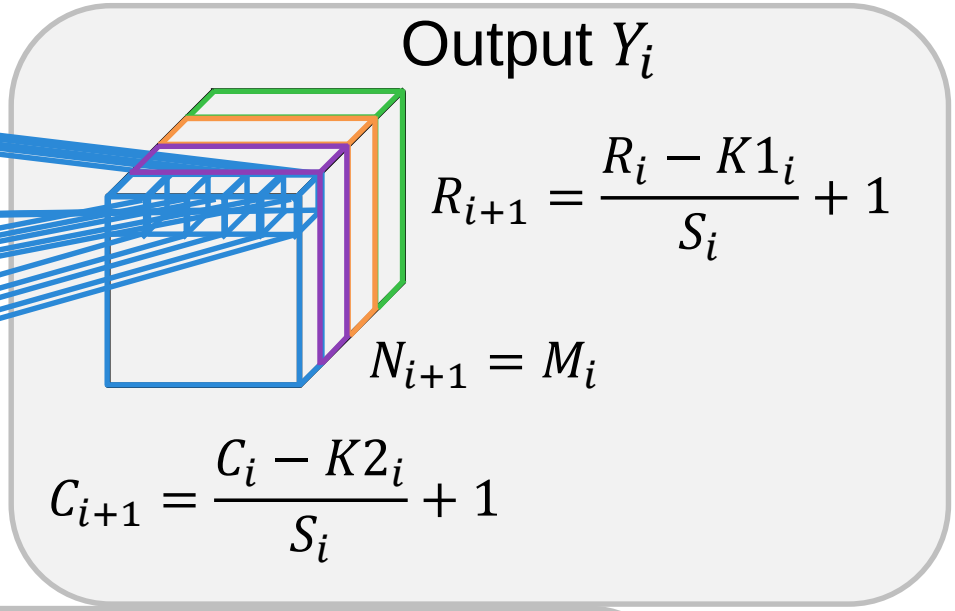


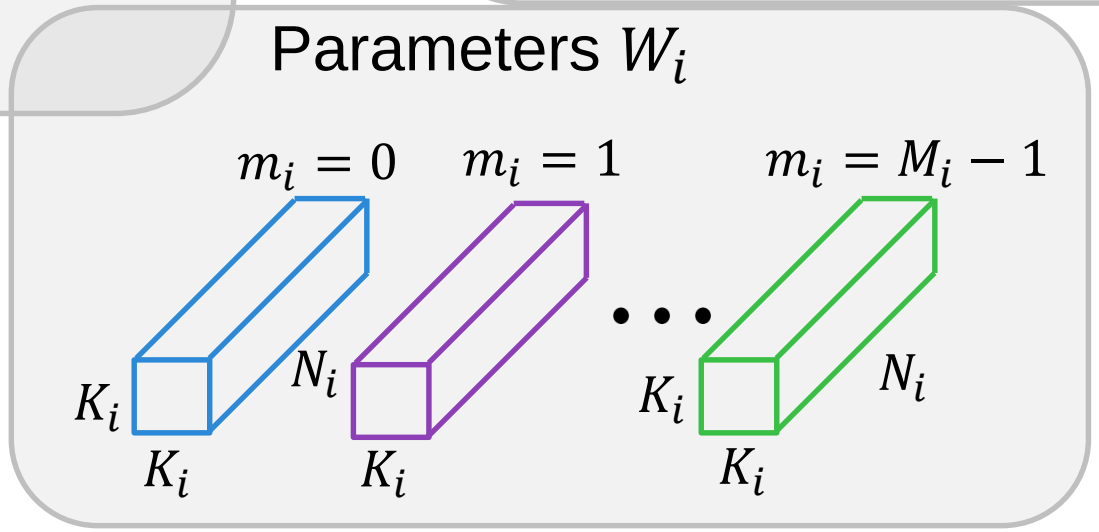
b) MobileNetv1, Layer-parallel

- Layer-by-layer Execution (a)
  - All input feature maps fully needed to start with computation of next layer
- Layer-parallel Execution (b)
  - Only subregion of feature map needed to start computation of next layer
  - Significant reduction of storage (for some nets up to **95%**)
  - Reordering of computation needed to achieve a maximal overlap

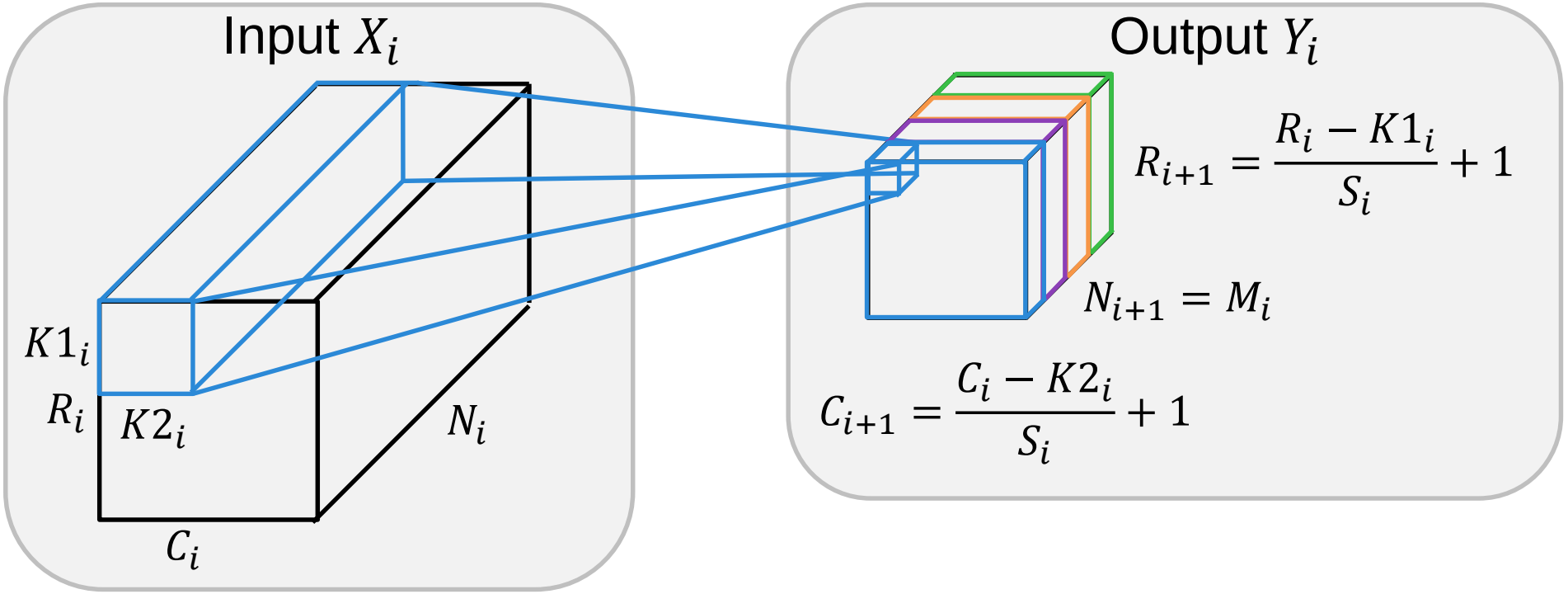
Convolution Layer







Convolution Layer




```
for (m = 0; m < M; m++)      //number of output feature maps (#filters)
  for (r = 0; r < Rout; r++)    //number of output feature map rows
    for (c = 0; c < Cout; c++)  //number of output feature map columns
      for (n = 0; n < N; n++)      //number of input feature maps
        for (k1 = 0; k1 < K1; k1++)     //filter kernel dimension 1
          for (k2 = 0; k2 < K2; k2++)    //filter kernel dimension 2
            Y[m][r][c] += W[m][n][k1][k2] * X[n][S*r+k1][S*c+k2];
      Y[m][r][c] += bias[m];
```

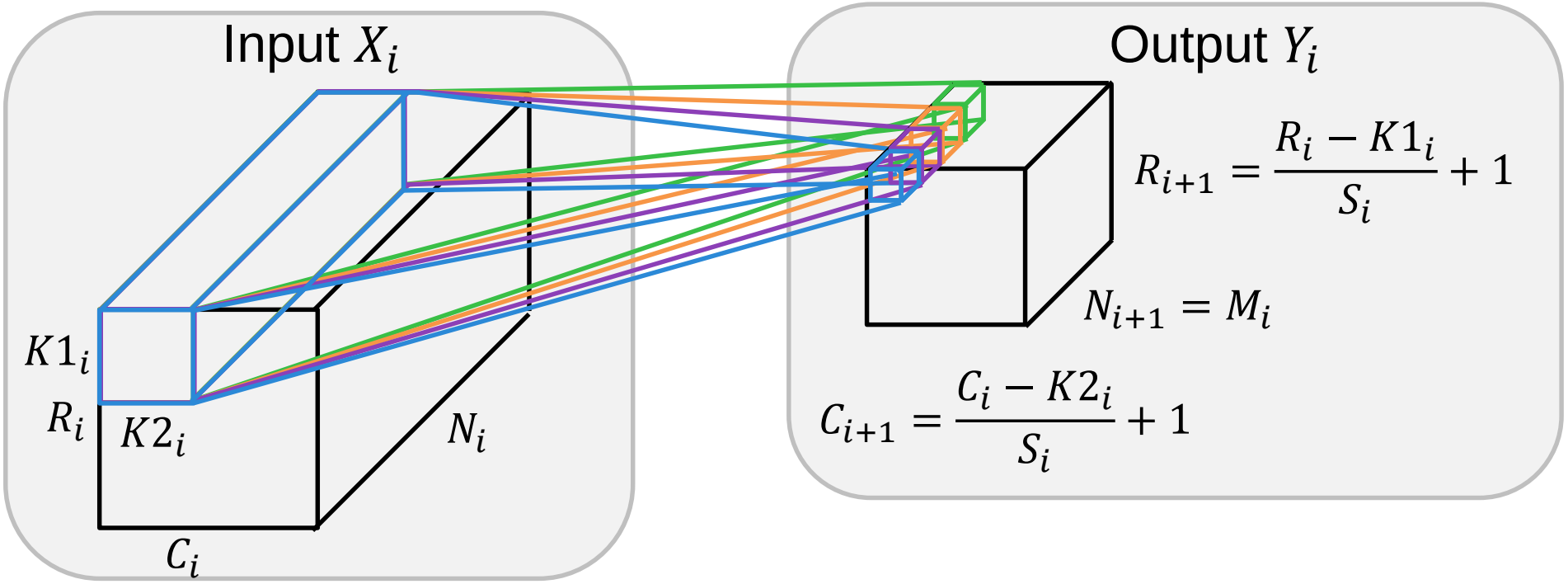
Loop Transformations

```
for (m = 0; m < M; m++)       //number of output feature maps (#filters)
  for (r = 0; r < Rout; r++)    //number of output feature map rows
    for (c = 0; c < Cout; c++)  //number of output feature map columns
      for (n = 0; n < N; n++)       //number of input feature maps
        for (k1 = 0; k1 < K1; k1++)     //filter kernel dimension 1
          for (k2 = 0; k2 < K2; k2++)    //filter kernel dimension 2
            Y[m][r][c] += W[m][n][k1][k2] * X[n][S*r+k1][S*c+k2];
      Y[m][r][c] += X[m];
```

```
for (r = 0; r < Rout; r++)      //number of output feature map rows
  for (c = 0; c < Cout; c++)    //number of output feature map columns
    for (k1 = 0; k1 < K1; k1++){    //filter kernel dimension 1
      for (k2 = 0; k2 < K2; k2++)    //filter kernel dimension 2
        for (n = 0; n < N; n+=delta)    //number of input feature maps
          for (m = p; m < M; m+=P)      //number of output feature maps
            forall (d = 0; d < delta; d++)
              Y[m][r][c] += W[m][n+d][k1][k2] * X[n][S*r+k1][S*c+k2];
            if (k1 == K1 && k2 == K2 && n  == N)
              Y[m][r][c] += bias[m];
```

- Loop permutation and unrolling
  - Move loops over filter and no. of feature maps (n, m) to innermost position
  - Allocate multiple filters to a PE, parallel filter execution between PEs
  - Innermost loop (d) corresponds to parallel computation within PEs (ILP)

Convolutional Layer

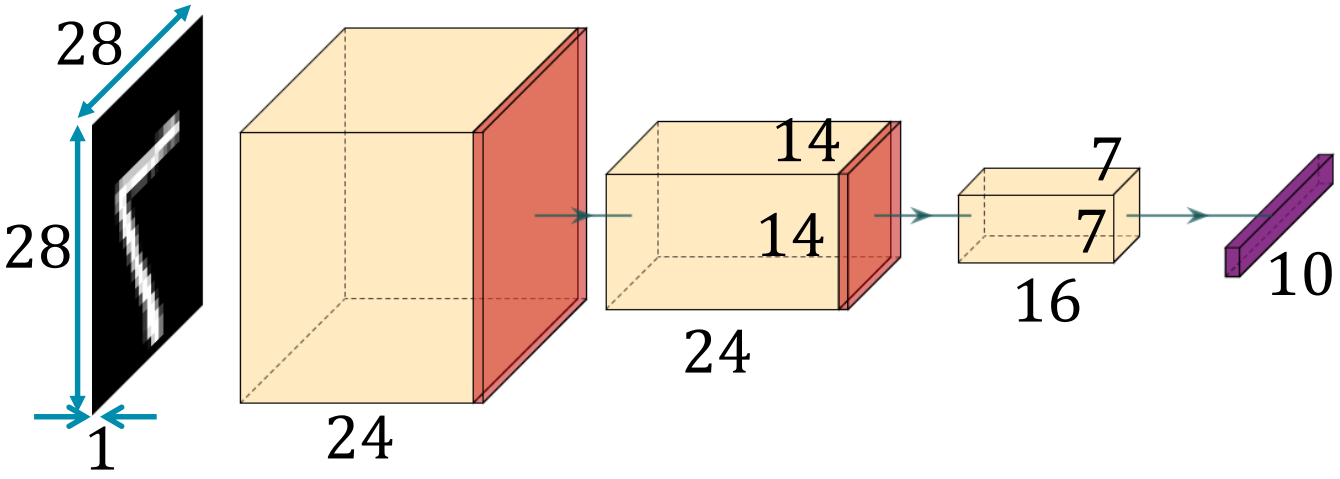



```
for (r = 0; r < Rout; r++)      //number of output feature map rows
  for (c = 0; c < Cout; c++)    //number of output feature map columns
    for (k1 = 0; k1 < K1; k1++){    //filter kernel dimension 1
      for (k2 = 0; k2 < K2; k2++)   //filter kernel dimension 2
        for (n = 0; n < N; n++)          //number of input feature maps
          for (m = p; m < M; m+=P)      //number of output feature maps
            forall (d = 0; d < delta; d++)
              Y[m][r][c] += W[m][n+d][k1][k2] * X[n][S*r+k1][S*c+k2];
            if (k1 == K1 && k2 == K2 && n  == N)
              Y[m][r][c] += bias[m];
```

Contents

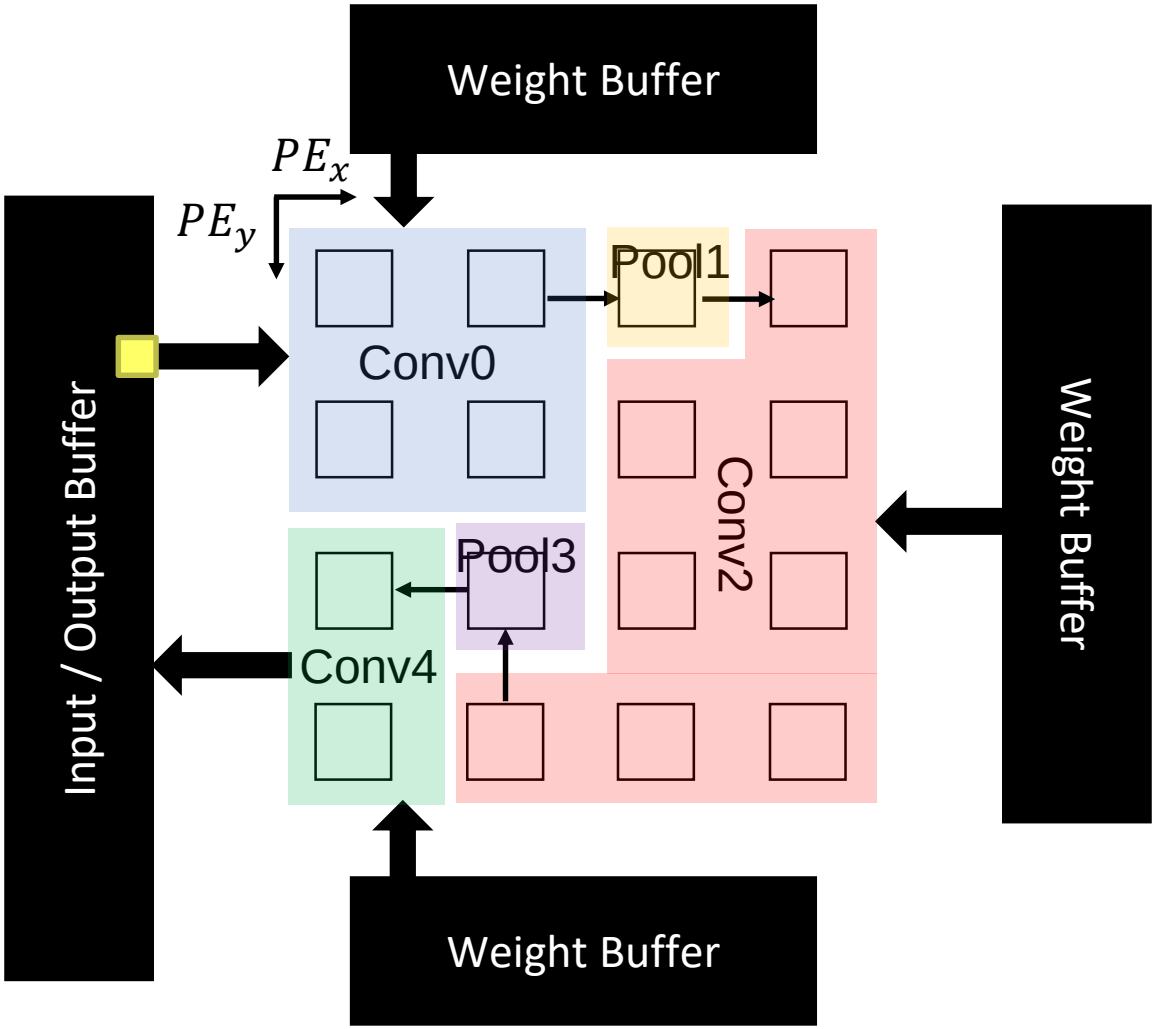
Convolutional Neural Network (CNN)

Conv0: $24 \times 1 \times 3 \times 3$
Pool1: $2 \times 2, S = 2$
Conv2: $24 \times 24 \times 3 \times 3$
Pool3: $2 \times 2, S = 2$
Conv4: $16 \times 24 \times 3 \times 3$
FC5: 10×784

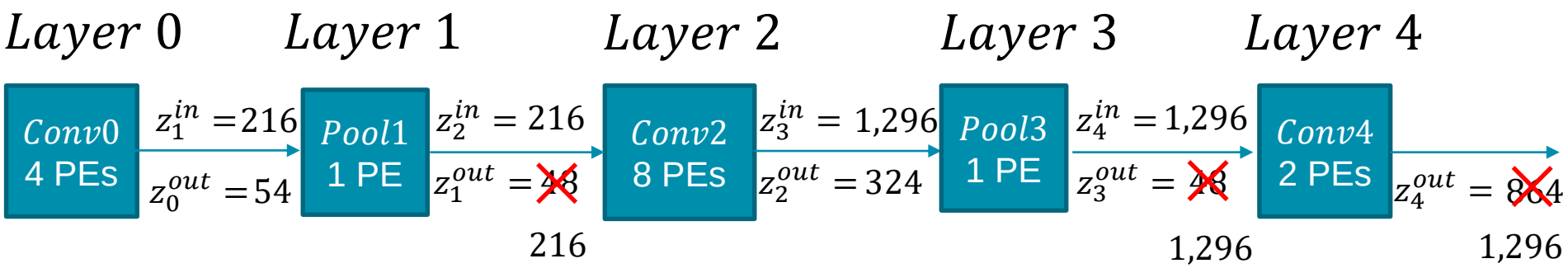


	Input	Conv0 (+Pool1)	Conv2 (+Pool3)	Conv4	Fc
Storage (KB) (inp+par+outp)	0.8	19.8 (23.5)	14.5 (5.9)	5.4	8.6
MACs	-	169,368	1,016,088	169,360	7,850

Mapping of CNN Layers onto 4x4 CGRA



Calculus – Layer-Parallel Processing



Required cycles to compute M_i output pixels:

$$z_i^{out} = \left\lceil \frac{M_i}{P_i} \right\rceil \cdot \left\lceil \frac{N_i}{\delta_i} \right\rceil \cdot K1_i \cdot K2_i$$

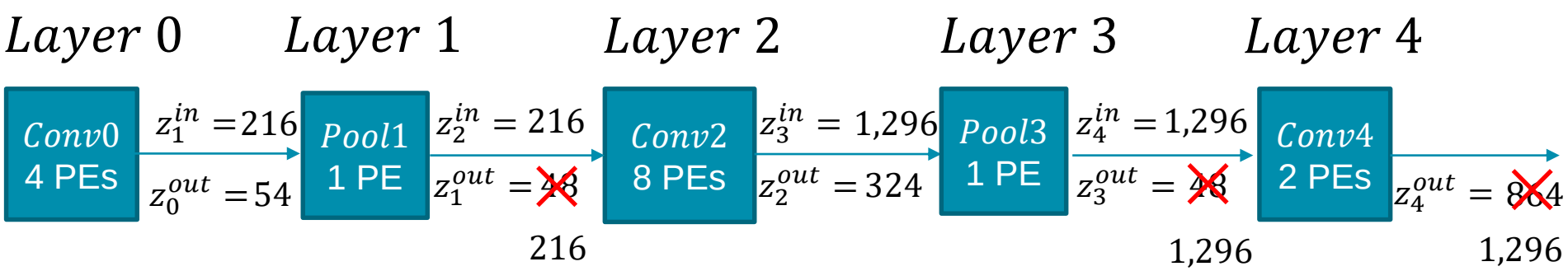
Number of pixels in R_i, C_i to start computing:

$$F_i = \min(K1_i \cdot K2_i, S1_i \cdot S2_i)$$

$$z_i^{in} = z_{i-1}^{out} \cdot F_i$$

$$z_i^{in} \leq z_i^{out}$$

Calculus – Layer-Parallel Processing



Start time layer i:

$$t_i = \sum_{j=0}^{i} Z_j = \sum_{j=0}^{i} z_j^{in}$$

Latency layer i:

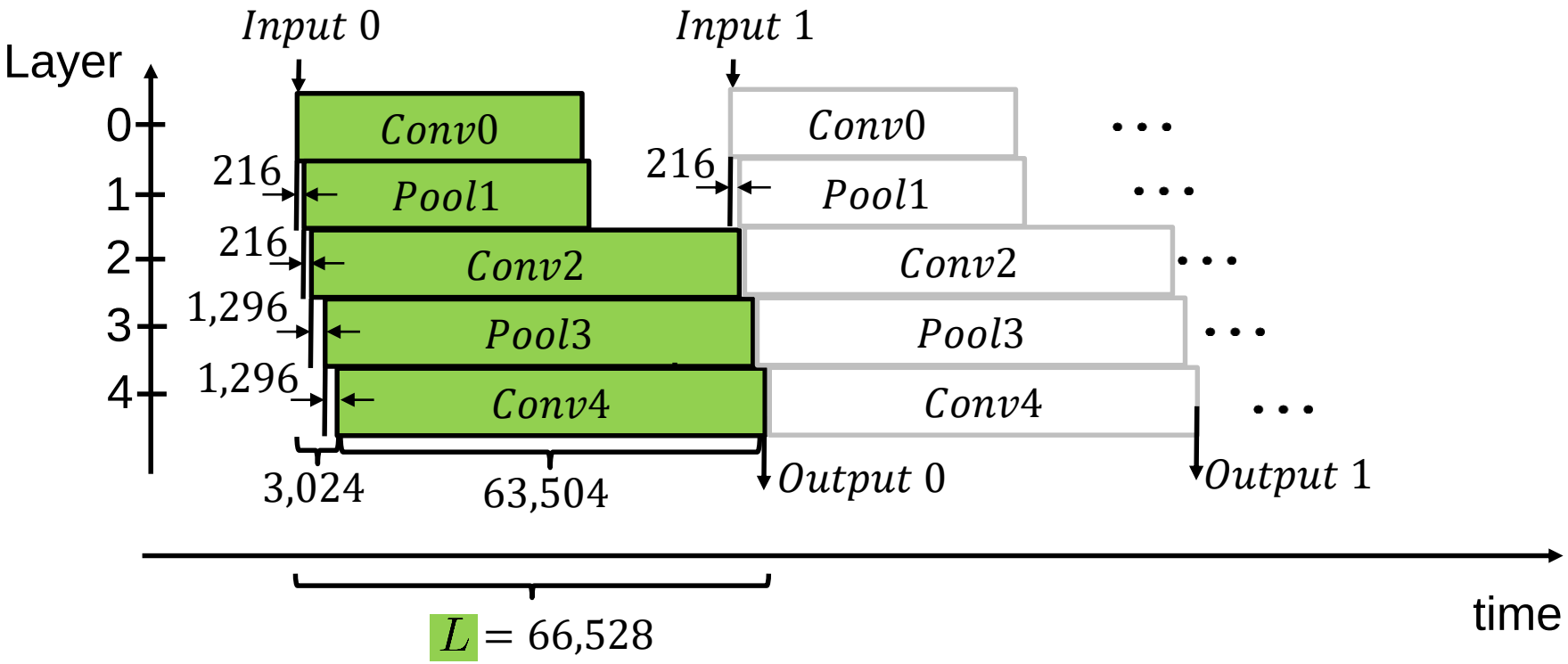
$$L_i = z_i^{out} \cdot R_{i+1} \cdot C_{i+1},$$

Overall latency:

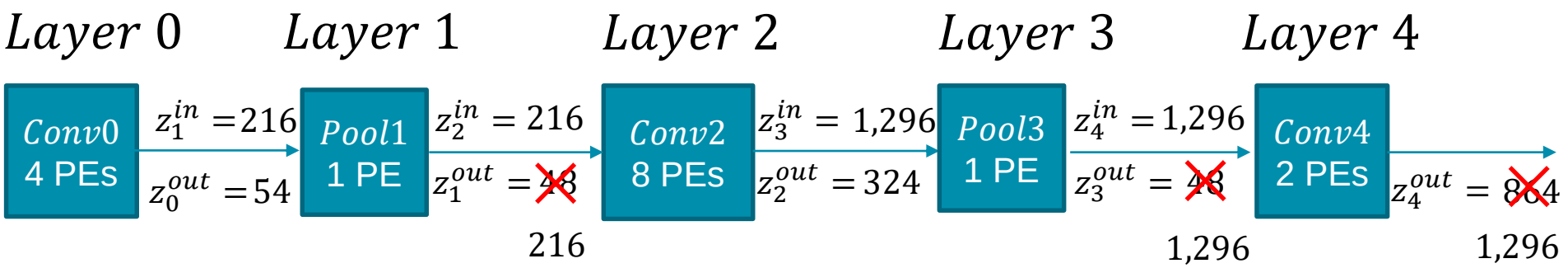
$$L = t_{V-1} + L_{V-1}$$

Z_i	0	216	216	1,296	1,296
t_i	0	216	432	1,728	3,024
L_i	42,336	42,336	63,504	63,504	63,504
L					66,528

Layer-Parallel Execution Schedule



Comparison of 16 PE implementations



	Layer-by-Layer	Layer-parallel
Latency L [cycles]	159,936	66,528
Througput T [fps]	312.6	787.4

Thank you!
Questions?

CNN Inference on CGRAs
under Throughput Constraints

Christian Heidorn, Michael Witterauf, Frank Hannig, and Jürgen Teich

References

[1] F. Hannig, V. Lari, S. Boppu, A. Tanase, and O. Reiche, *Invasive Tightly-Coupled Processor Arrays: A Domain-Specific Architecture/Compiler Co-Design Approach*, ACM Transactions on Embedded Computing Systems (TECS), 13(4s), pp. 133:1-133:29, 2014. https://doi.org/10.1145/2584660

[2] É. Sousa, A. Tanase, F. Hannig, and J. Teich, *A Reconfigurable Memory Architecture for System Integration of Coarse-Grained Reconfigurable Arrays*, In Proc. of the Int'l Conference on Reconfigurable Computing and FPGAs (ReConFig), 8 pp., IEEE, December 4-6, 2017. https://doi.org/10.1109/RECONFIG.2017.8279768

[3] C. Heidorn, M. Witterauf, F. Hannig, and J. Teich, *Efficient Mapping of CNNs onto Tightly Coupled Processor Arrays*, 2019.

[4] Y. LeCun and C. Cortes, *MNIST handwritten digit database*, 2010.

[5] https://electrek.co/2018/10/15/tesla-new-autopilot-neural-net-v9/, visited 09-2019.

[6] https://developer.nvidia.com/deep-learning-performance-training-inference , visited 09-2019