Like an Open Book? Read Neural Network Architecture with Simple Power Analysis on 32-bit Microcontrollers

Raphaël Joud[1,2], Pierre-Alain Moëllic[1,2], Simon Pontié[1,2], and Jean-Baptiste Rigaud[3]

1 CEA Tech, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne, F-13541 Gardanne, France, {name.surname}@cea.fr
2 Univ. Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France
3 Mines Saint-Etienne, CEA, Leti, Centre CMP, F-13541 Gardanne France, surname@emse.fr

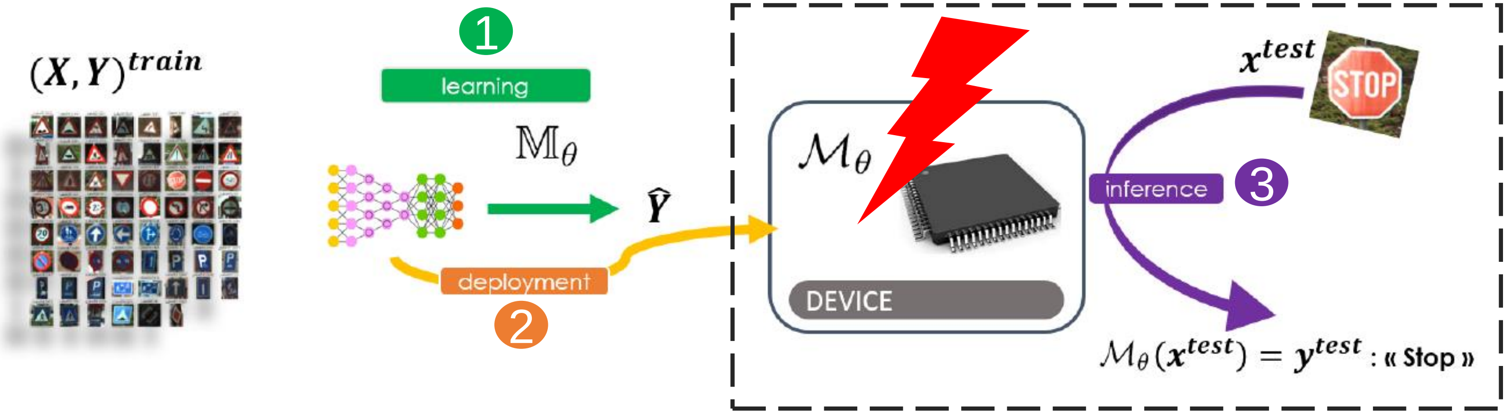
Outline

- Introduction
- Challenges
- Scope and threat model
- Model analysis
- Layer analysis
- Discussion & Perspectives

Threats related to Machine Learning

- Large-scale Machine Learning (ML) model deployment
- Wide variety of applications and HW platforms involved
- Questions about ML security and related attack surface
- Attacking a ML model:
  - Integrity
  - Availability
  - **Confidentiality**

Theoretical / API-based attack
Side channel analysis

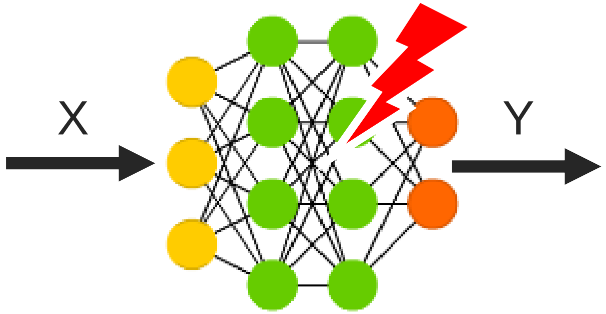


Overall attack surface

Attack Surface

Algorithmic / API-based attacks

Target theoretical flaws – Include adversarial examples, data poisoning, membership inference, model extraction…



Overall attack surface

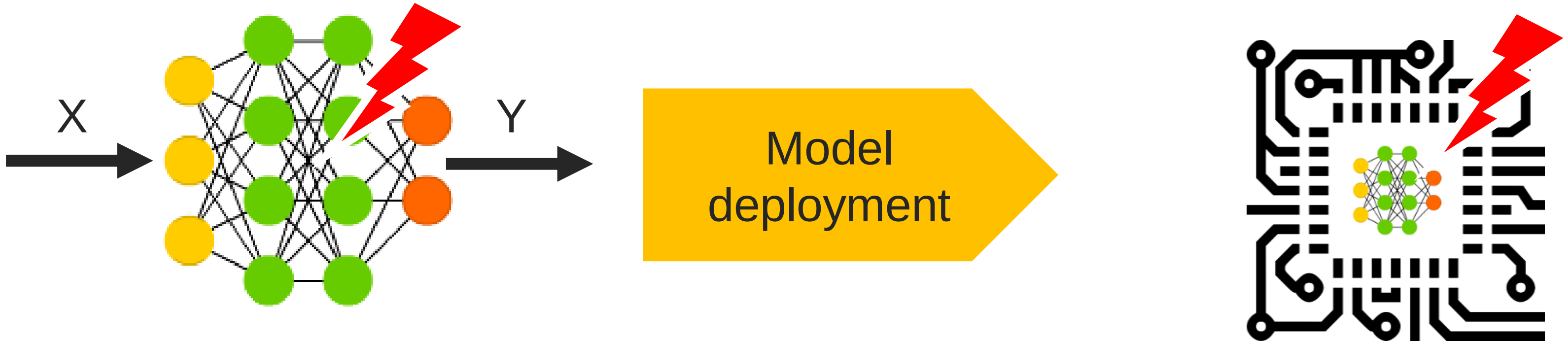
Attack Surface

Algorithmic / API-based attacks

Implementation-based attacks

Target theoretical flaws – Include adversarial examples, data poisoning, membership inference, model extraction…

Target implementation flaws – Include physical attacks like Fault Injection Analysis (FIA) and Side-Channel Analysis (SCA)



Outline

- Introduction
- Challenges
- Scope and threat model
- Model analysis
- Layer analysis
- Discussion & Perspectives

Stakes related to model architecture

- Third-party vs Ad-hoc architectures

- Architecture knowledge is required to carry out various attacks

- Often required for parameters extraction whatever the attack surface considered [1 to 7]

Attack surface / method	Example of attacks requiring architecture knowledge
API-based	Cryptanalytic extraction [1,2] High Accuracy and High Fidelity [3]
FIA-based	Deepsteal [4] Fault Injection & Safe-error [5]
SCA-based	Practical Introduction [6] Reverse-engineering DNN [7]

State of the art of architecture extraction

Attack	Physical target	Targeted models	Used techniques
Duddu *et al.* (2018) [8]	MLaaS	CNN	TA & Regression
Yu *et al.* (2020) [9] Yli *et al.* (2021) [10]	FPGA	BNN	SEMA
Luo *et al.* (2022) [11]	FPGA	CNN & ResNet	SEMA
Chmielewski *et al.* (2021) [12]	GPU	CNN	SEMA & TA
Batina *et al.* (2019) [13]	µC	MLP & CNN	CEMA
Xiang *et al.* (2020) [14]	µC	CNN	SPA & ML
Ours	**µC**	**MLP & CNN**	**SEMA only**

MLaaS: ML as a Service,
TA: Timing Analysis,

SEMA: Simple EM Analysis,
SPA: Simple Power Analysis

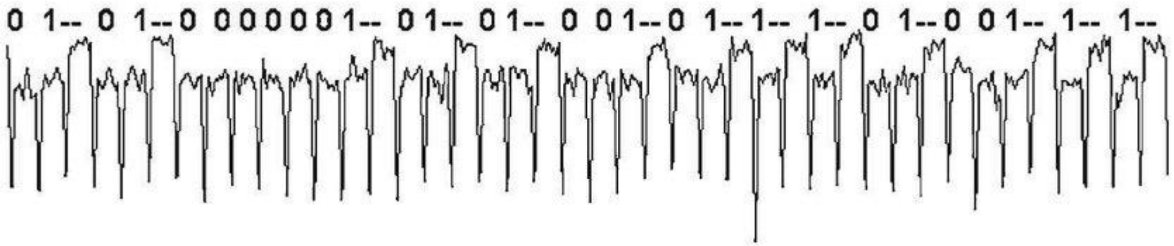
What is a Simple Power / EM Analysis ?

SPA / SEMA:

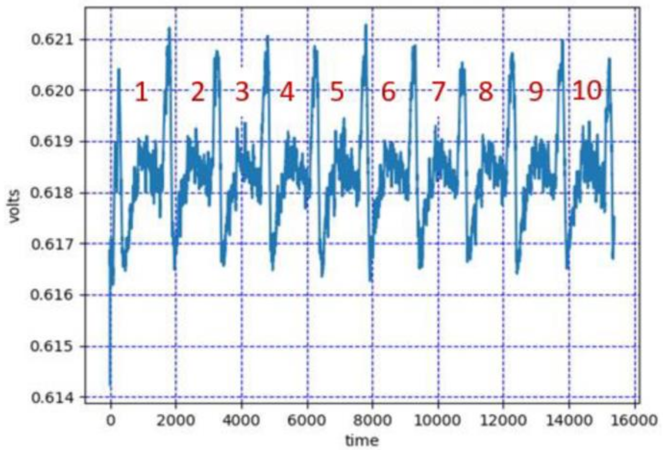
- “Direct” interpretation of power consumption / EM measurements collected

- Can allow to identify parts of codes or even secret information

- Often requires knowledge of used implementation library



SPA on RSA emanations



SPA on AES emanations

Outline

- Introduction
- Challenges
- Scope and threat model
- Model analysis
- Layer analysis
- Discussion & Perspectives

Scope and Threat model

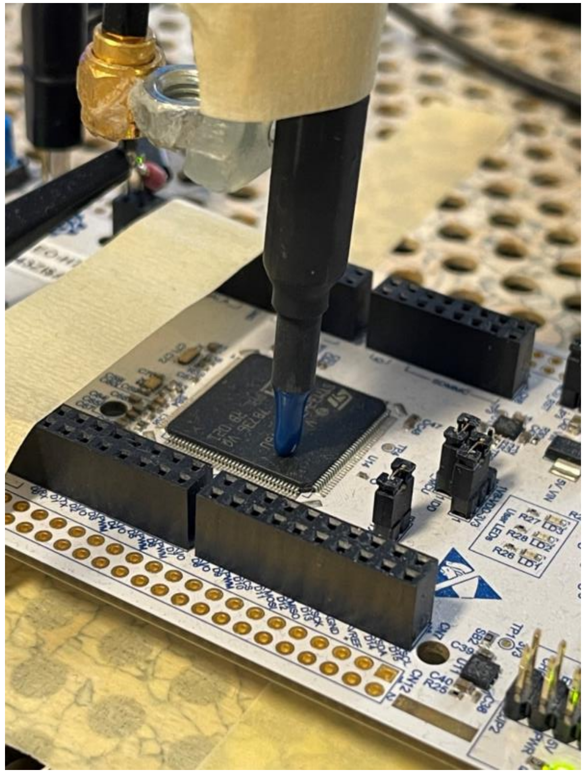
- Goal: extract models architecture as precisely as possible

- Targeted models: quantified (8-bits) and implemented with open-source libraries: **NNoM** [15] & **CMSIS-NN** by ARM [16]

  ➔ Analyse layer C code to anticipate corresponding EM leakages (cache optimisations are disabled)

- Black-box context: no knowledge of model architecture nor parameters

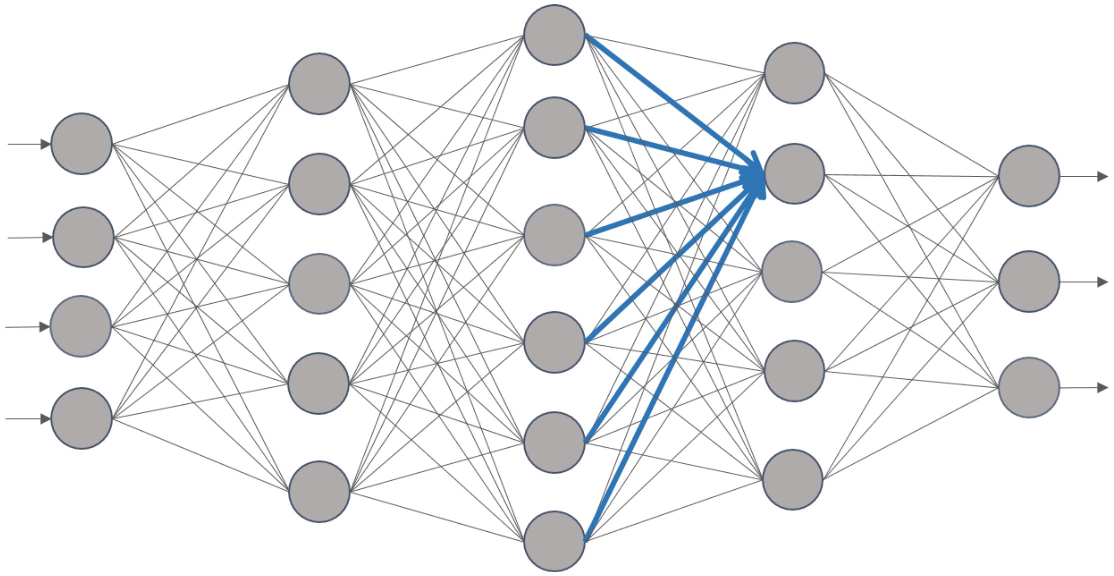
- EM traces acquired with only 1 input: avoids desynchronisation problems & allows to average traces



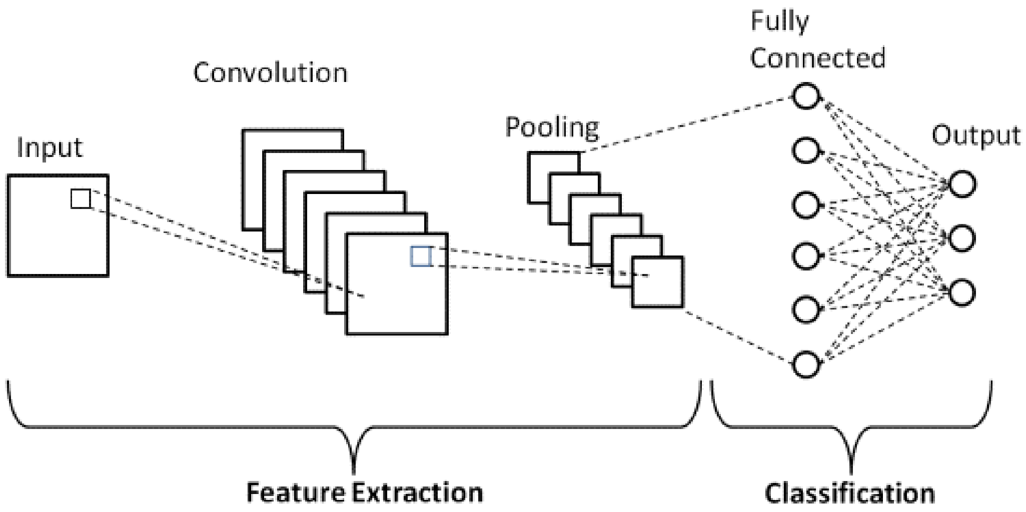
Scope and Threat model

- Studied architectures:

MultiLayer Perceptron (MLP)

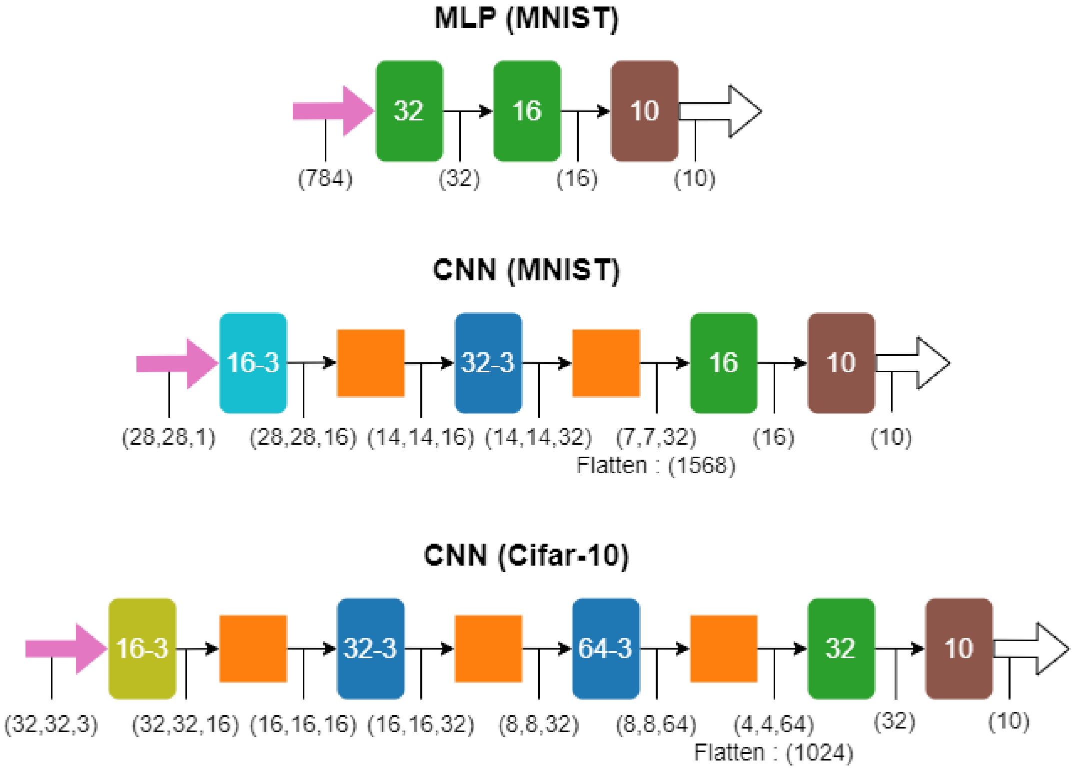


Convolutional Neural Networks (CNN)



Scope and Threat model

- 3 different targeted models:



Target	Hyper-parameter	Notation
Conv. layer	Output shape	H_{out}
	# Kernels	K
	Kernel size	Z
	Stride, Padding	S, P
MaxPool layer	Output shape	H_{out}
	Filter size	Z_{pool}
Dense layer	# Neurons	N_e
Activation layer	ReLU or not	Ø
Model	# Layers	L
	Layers nature	Ø

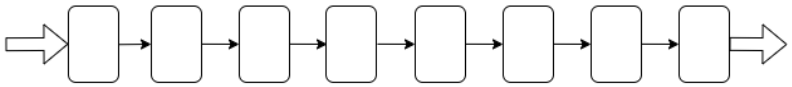
Outline

- Introduction
- Challenges
- Scope and threat model
- Model analysis
- Layer analysis
- Discussion & Perspectives

Model Analysis

Analyse the overall shape of inference trace:

1. Find the number of layers:
    - Straightforward or can rely on frequency spectrum

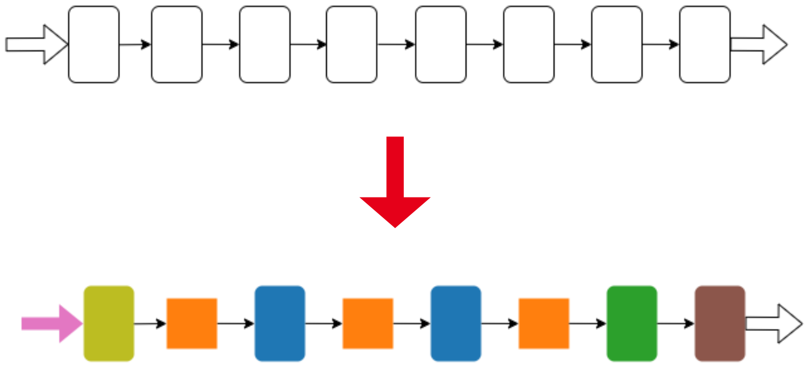


2. Identify their nature:
    - Use layer execution time (complexity)
    - Use patterns of specific layer
3. Extract hyper-parameters of each of them

Model Analysis

Analyse the overall shape of inference trace:

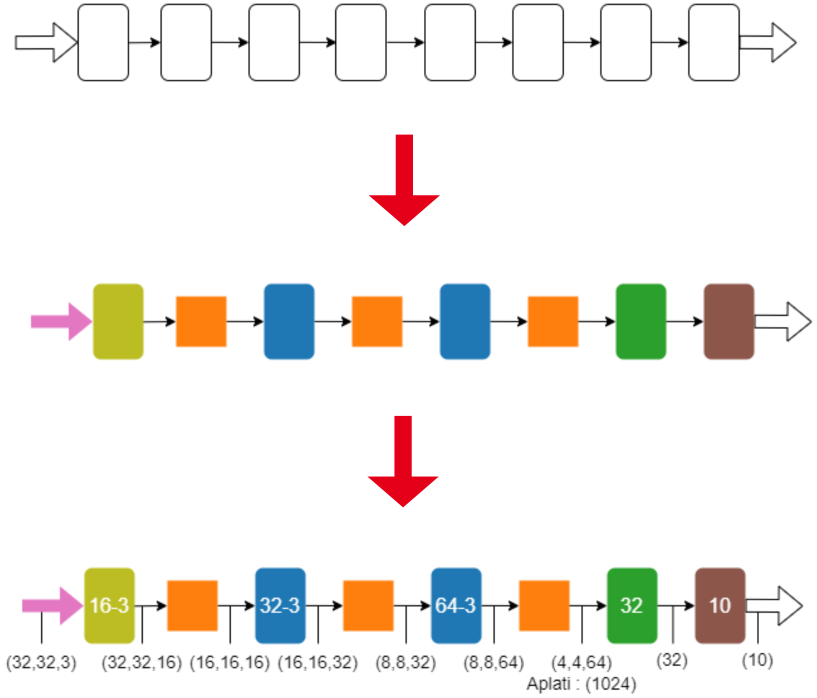
1. Find the number of layers:
   - Straightforward or can rely on frequency spectrum
2. Identify their nature:
   - Use layer execution time (complexity)
   - Use patterns of specific layer
3. Extract hyper-parameters of each of them



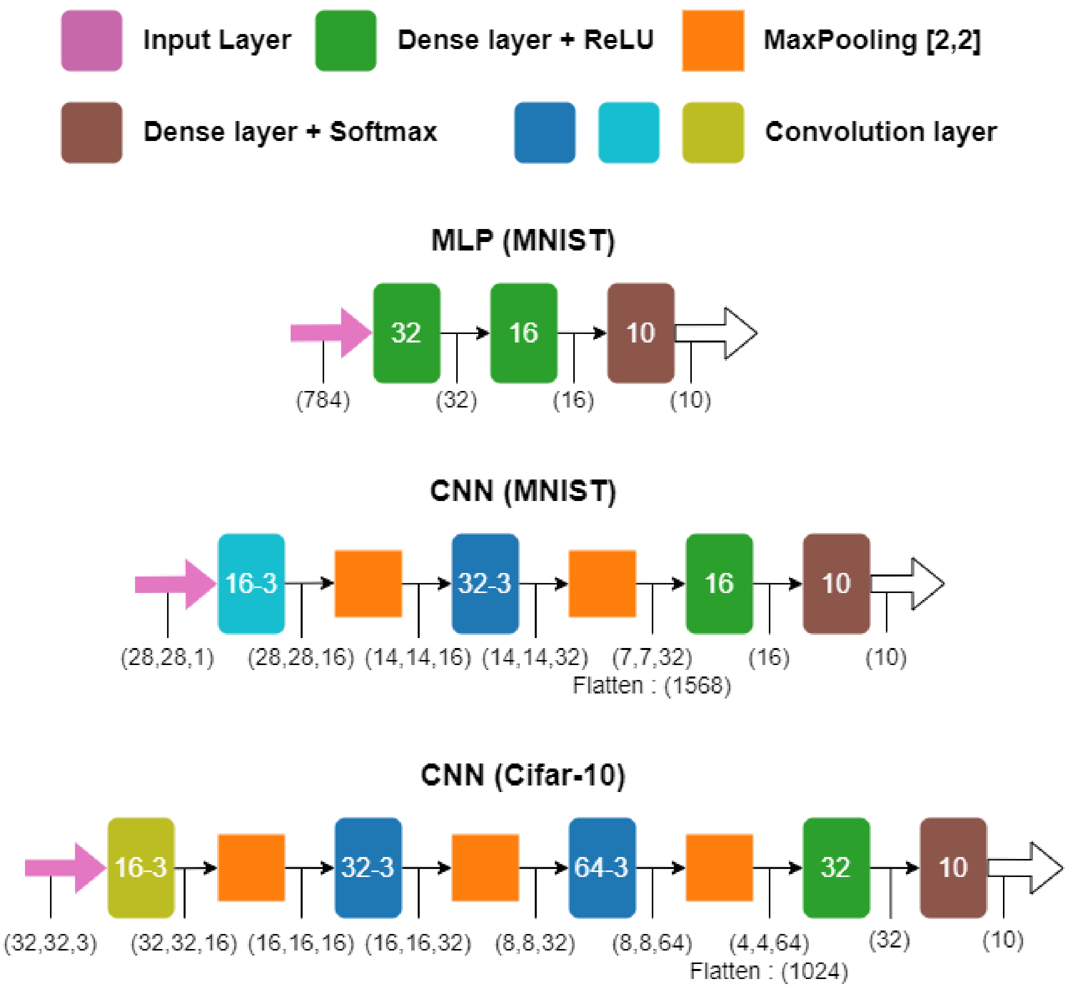
Model Analysis

Analyse the overall shape of inference trace:

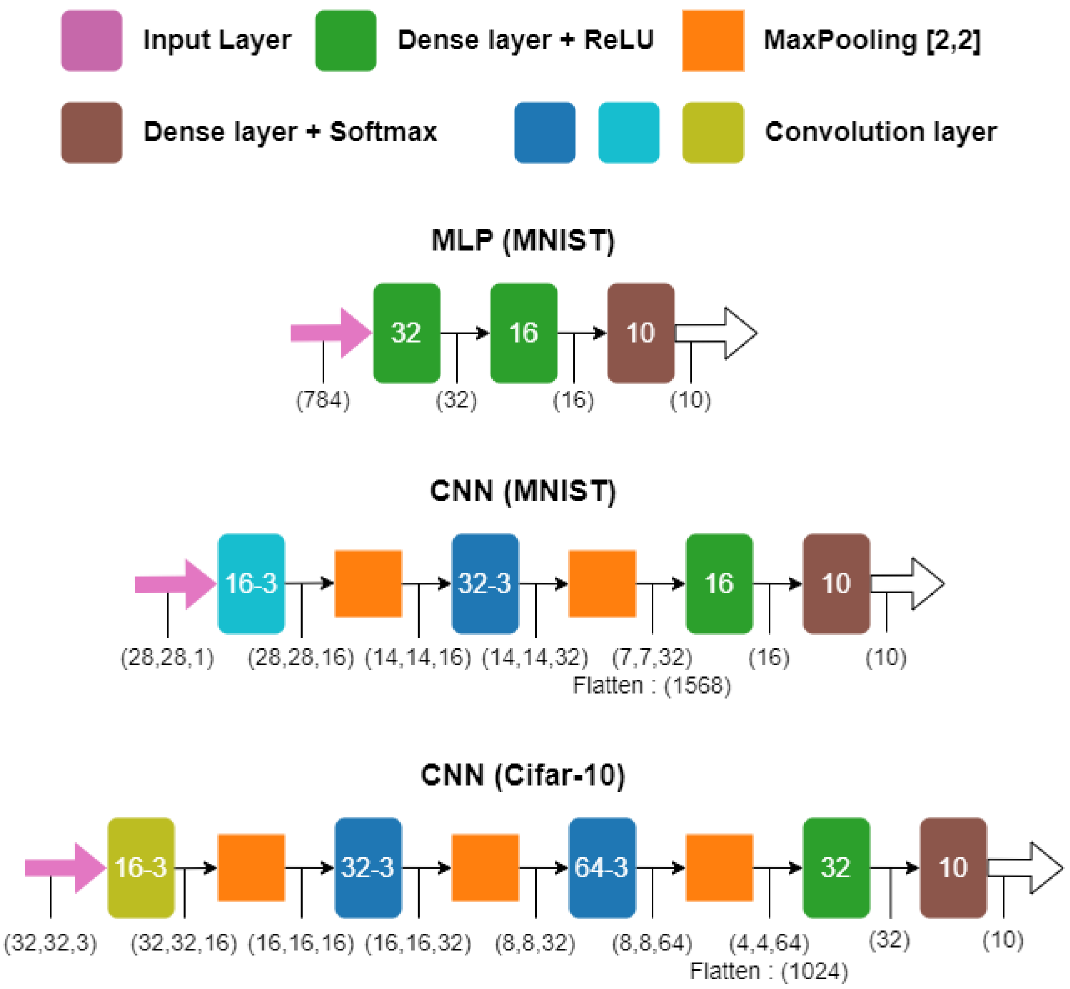
1. Find the number of layers:
   - Straightforward or can rely on frequency spectrum
2. Identify their nature:
   - Use layer execution time (complexity)
   - Use patterns of specific layer
3. Extract hyper-parameters of each of them

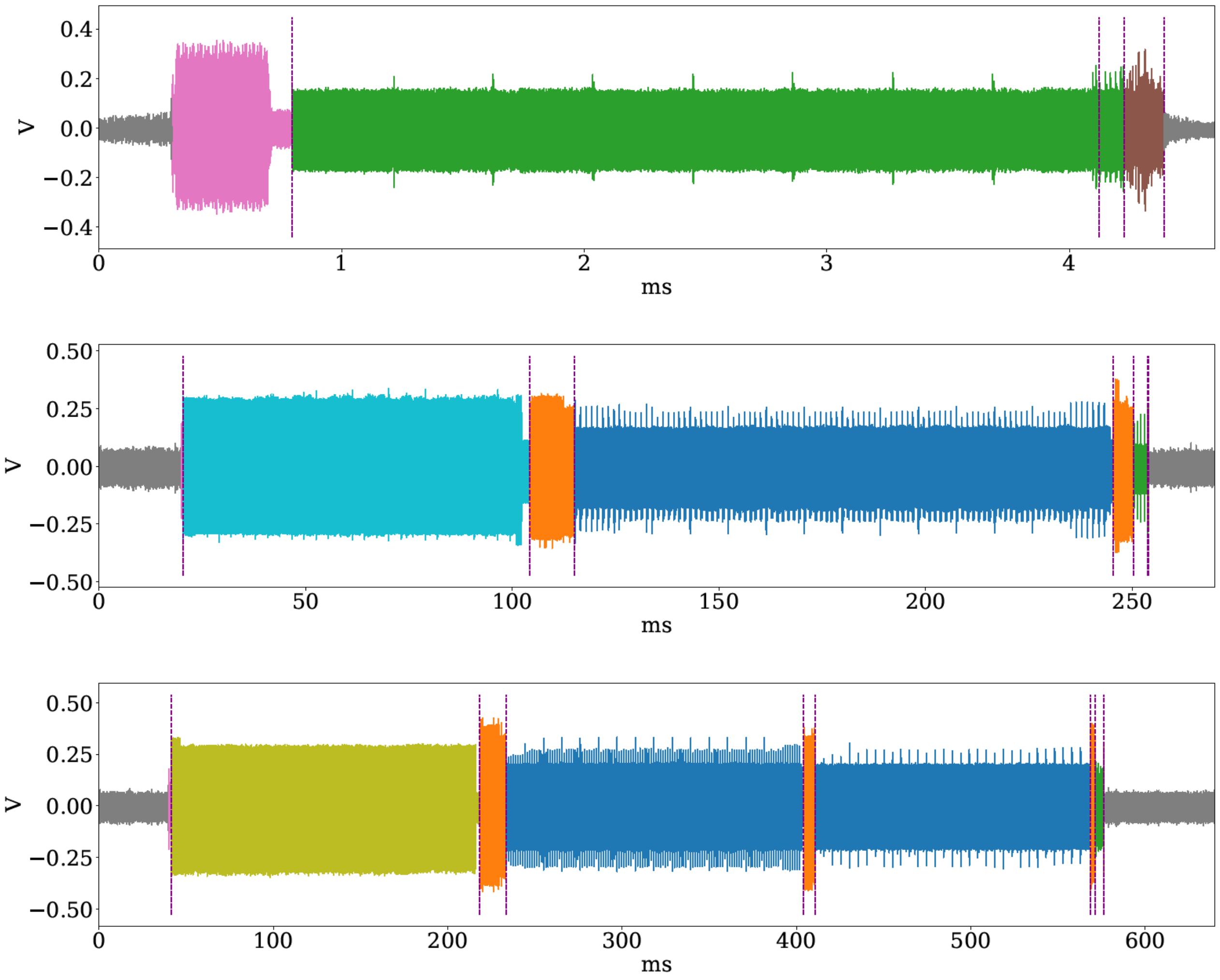


Model Analysis

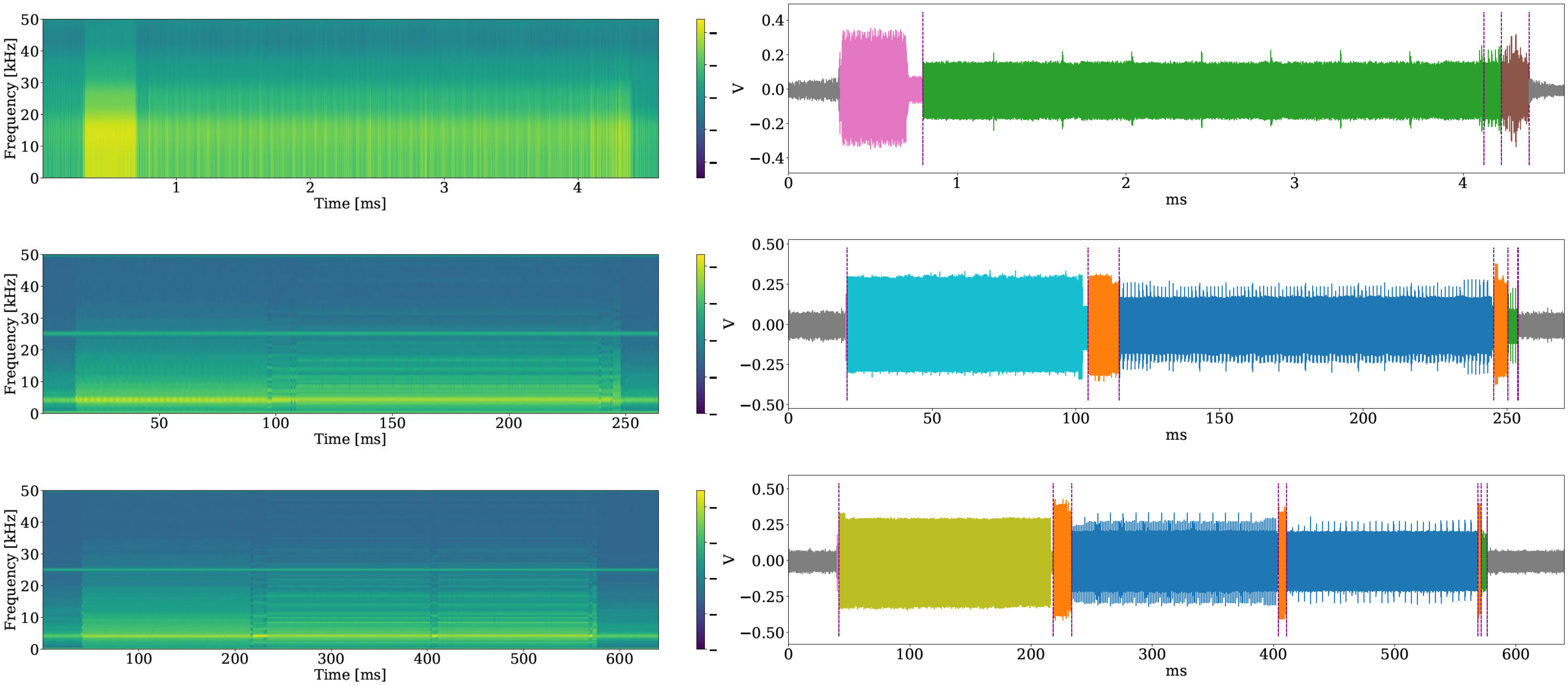


Model Analysis



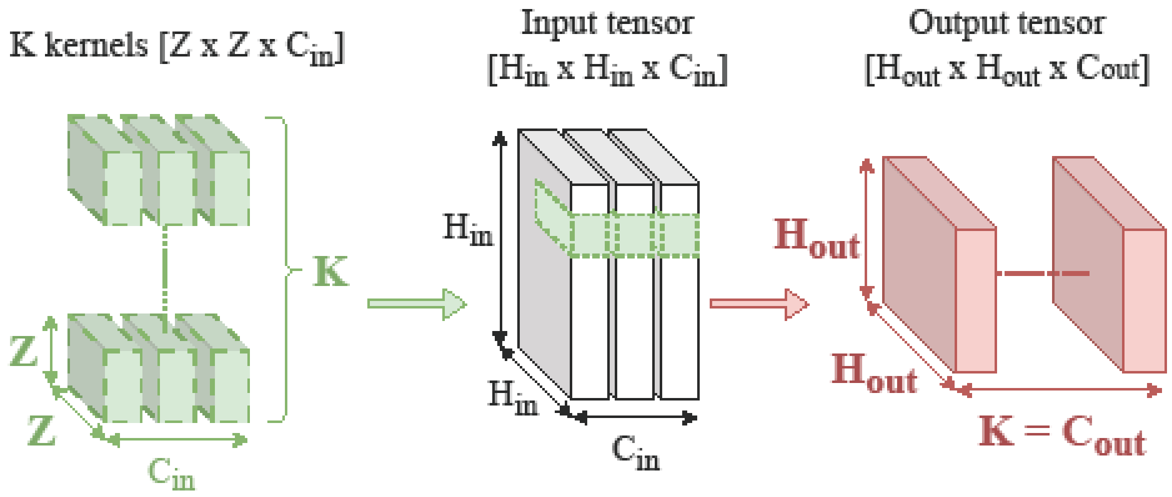


Model Analysis



Outline

Convolution layers



- Allows to extract: H_{out}, K and Z
- Allows to deduce: S and P

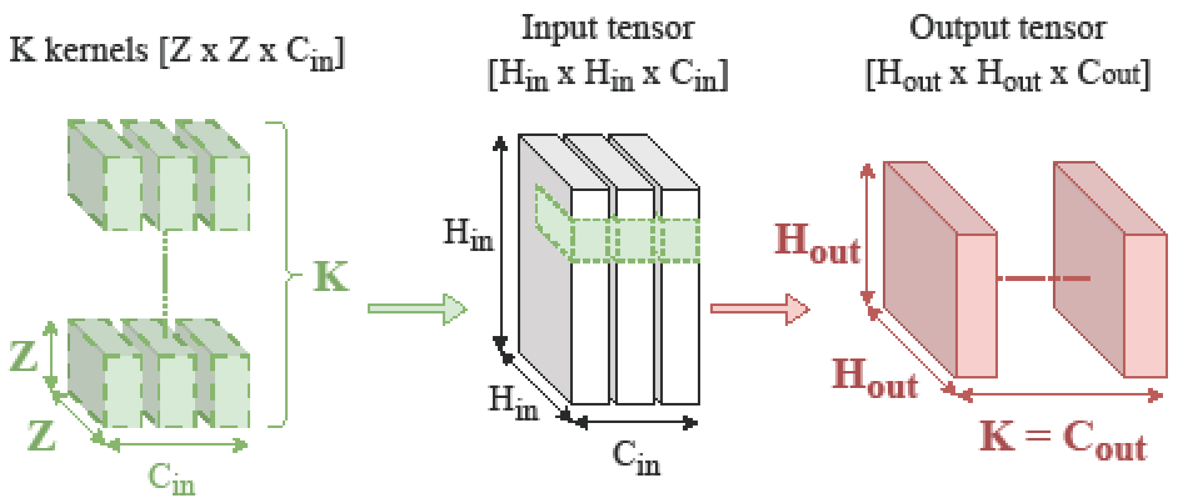
Algorithm 1 General convolution implementation

Input: Input tensor I_{in} of size $H_{in}^2 \cdot C_{in}$, ker (Kernel tensor), S, P, Output size I_{out} of size $H_{out}^2 \cdot C_{out}$

Output: Filled I_{out}

for $i_y \leftarrow 0,\ i_y < H_{out},\ i_y++1$ **do** ▷ Iterate over H_{out} (y-axis)
  for $i_x \leftarrow 0,\ i_x < H_{out},\ i_x++1$ **do** ▷ idem (x-axis)
    $\text{buff}_{in} \leftarrow im2col(I_{in}, i_y, i_x, S, P, H_{in}, C_{in})$ ▷ Apply Im2col conversion
    if $len(\text{buff}_{in}) == 2 \times C_{in} \times Z^2$ **then** ▷ Checks if 2 input columns are set
      $GeMM(\text{buff}_{in}, ker, C_{out}, C_{in} \times Z^2, I_{out})$ ▷ Perform matrix-product
      $\text{buff}_{in} \leftarrow 0$ ▷ Buffer reset
    end if
  end for
end for

Convolution layers



- Allows to extract: H_{out}, K and Z
- Allows to deduce: S and P

Algorithm 1 General convolution implementation

Input: Input tensor I_{in} of size $H_{in}^2 \cdot C_{in}$, ker (Kernel tensor), S, P, Output size I_{out} of size $H_{out}^2 \cdot C_{out}$

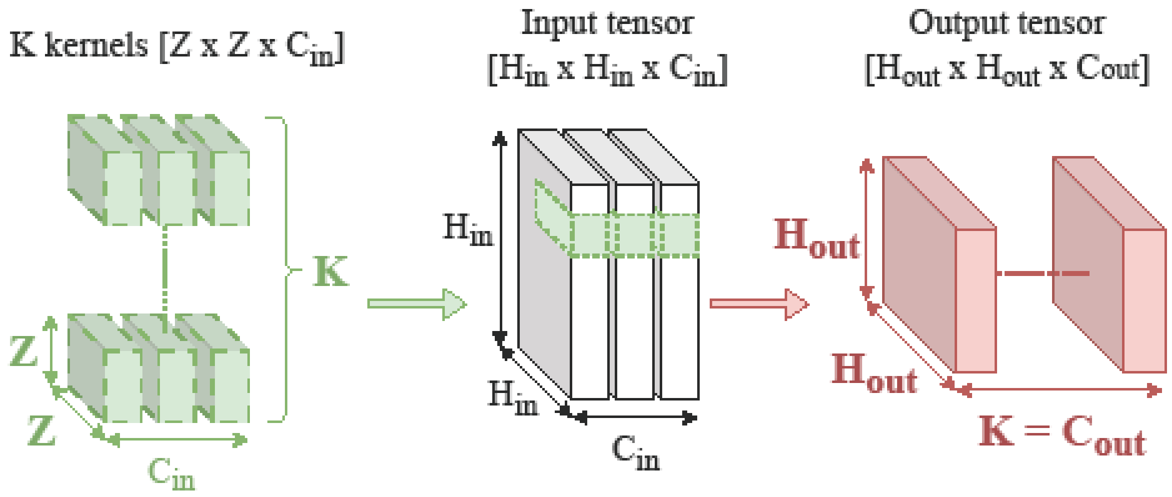
Output: Filled I_{out}

for $i_y \leftarrow 0,\ i_y < H_{out},\ i_y{+}{+}1$ **do** ▷ Iterate over H_{out} (y-axis)
  for $i_x \leftarrow 0,\ i_x < H_{out},\ i_x{+}{+}1$ **do** ▷ idem (x-axis)
    $\mathrm{buff}_{in} \leftarrow im2col(I_{in}, i_y, i_x, S, P, H_{in}, C_{in})$ ▷ Apply Im2col conversion
    if $len(\mathrm{buff}_{in}) == 2 \times C_{in} \times Z^2$ **then** ▷ Checks if 2 input columns are set
      $GeMM(\mathrm{buff}_{in}, ker, C_{out}, C_{in} \times Z^2, I_{out})$ ▷ Perform matrix-product
      $\mathrm{buff}_{in} \leftarrow 0$ ▷ Buffer reset
    end if
  end for
end for

$H_{out} \times H_{out}$ iterations

GeMM function called $H_{out} \times (H_{out}/2)$ times

Convolution layers



- Allows to extract: H_{out}, K and Z
- Allows to deduce: S and P

Algorithm 1 General convolution implementation

Input: Input tensor I_{in} of size $H_{in}^2 \cdot C_{in}$, ker (Kernel tensor), S, P, Output size I_{out} of size $H_{out}^2 \cdot C_{out}$

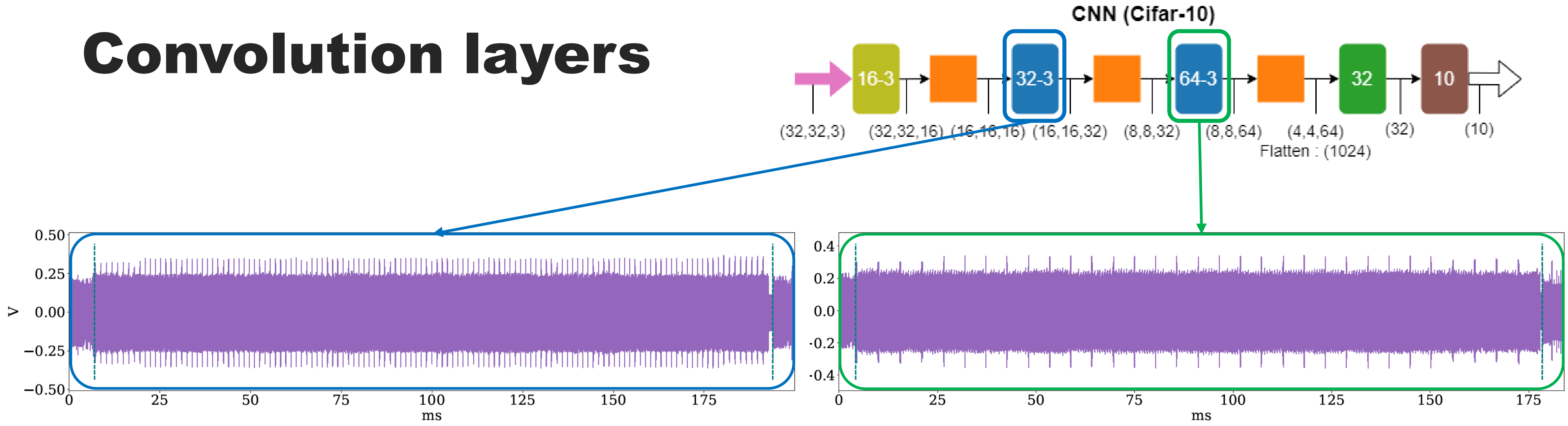
Output: Filled I_{out}

```
for i_y ← 0, i_y < H_out, i_y + +1 do                          ▷ Iterate over H_out (y-axis)
    for i_x ← 0, i_x < H_out, i_x + +1 do                      ▷ idem (x-axis)
        buff_in ← im2col(I_in, i_y, i_x, S, P, H_in, C_in)     ▷ Apply Im2col conversion
        if len(buff_in) == 2 × C_in × Z^2 then                 ▷ Checks if 2 input columns are set
            GeMM(buff_in, ker, C_out, C_in × Z^2, I_out)       ▷ Perform matrix-product
            buff_in ← 0                                        ▷ Buffer reset
        end if
    end for
end for
```

With N_p the number of pattern corresponding to GeMM function call:

$$H_{out} = \sqrt{2 \times N_p}$$

Convolution layers



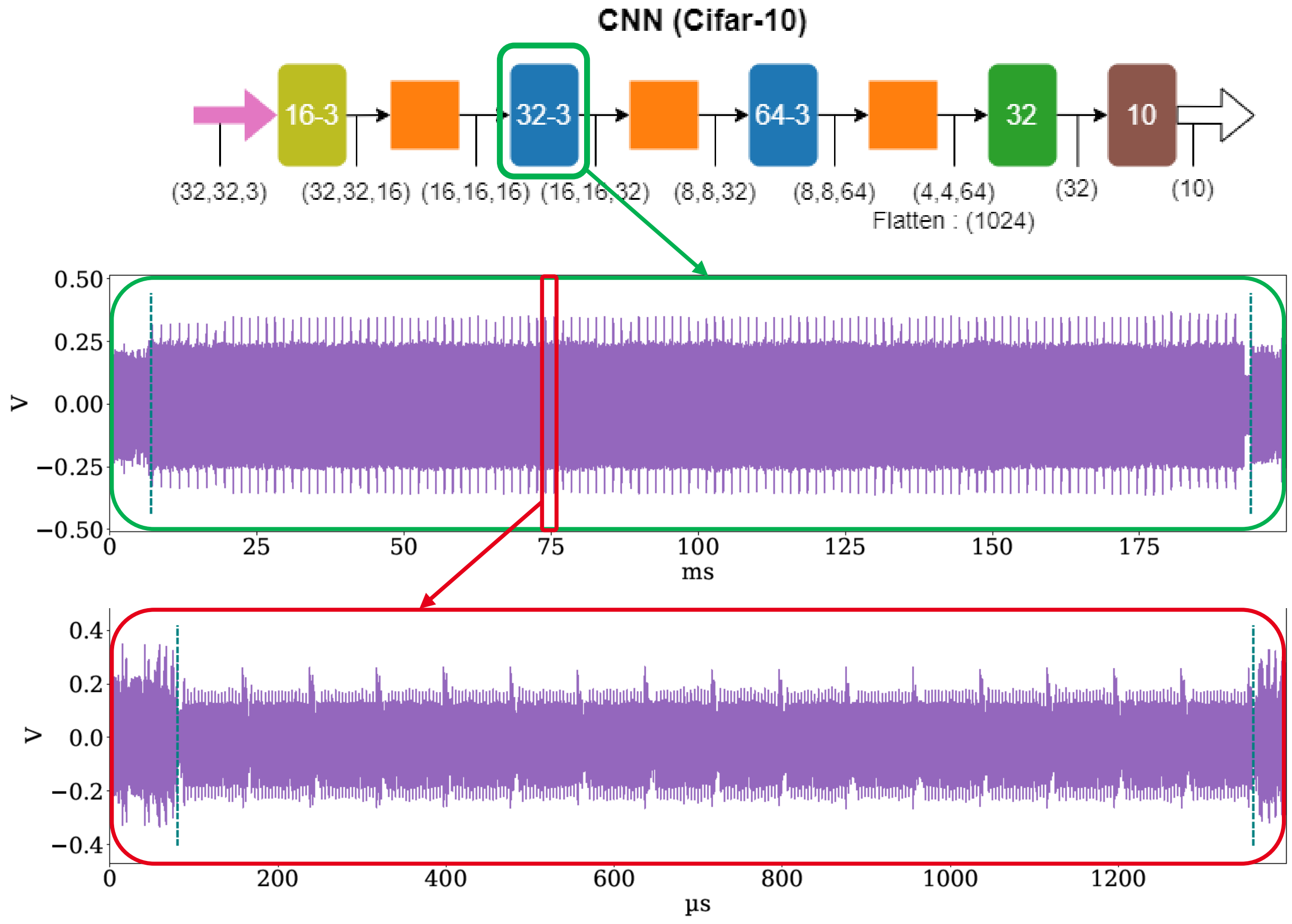
$N_p = 128$ iterations $\Leftrightarrow H_{out} = 16$

$N_p = 32$ iterations $\Leftrightarrow H_{out} = 8$

- Allows to extract: H_{out}, K and Z
- Allows to deduce: S and P
- → $\boldsymbol{H_{out} = \sqrt{2 \times N_p}}$

Convolution layers

- Allows to extract: H_{out}, K and Z
- Allows to deduce: S and P
- Russian dolls effect to get K



Convolution layers

- Allows to extract: H_{out}, K and Z
- Allows to deduce: S and P
- Russian dolls effect to get K

Algorithm 1 Matrix-product for convolution (**GeMM**)

Input: buff_{in}, ker, C_{out}, $C_{in} \times Z^2$, I_{out}, $bias$ (bias tensor)
Output: Partly filled I_{out}

```
rowCnt ← C_out >> 1                                          ▷ Set rowCnt = K/2
for rowCnt > 0, rowCnt − −1 do                                ▷ Iterate over K/2
    sum, sum1, sum2, sum3 = init_sum(bias, C_in × Z^2)
    colCnt ← (C_in × Z^2) >> 2                                ▷ Set colCnt
    for colCnt > 0, colCnt − −1 do
        simd_mac(sum, sum1, sum2, sum3, buff_in, ker)
    end for
    if (C_in × Z^2) & 0x3 then
        Manage_colCnt_remainder(sum, sum1, sum2, sum3, buff_in, bias, C_in × Z^2)
    end if
    apply_mac(sum, sum1, sum2, sum3, C_out, I_out)
end for
if C_out & 0x3 then                                           ▷ Check presence of remainders
    sum, sum1 = init_sum(bias, C_in × Z^2)
    colCnt ← (C_in × Z^2) >> 2                                ▷ Set colCnt
    for colCnt > 0, colCnt − −1 do
        simd_mac(sum, sum1, buff_in, ker)
    end for
    if (C_in × Z^2) & 0x3 then
        Manage_colCnt_remainder(sum, sum1, buff_in, bias, C_in × Z^2)
    end if
    apply_mac(sum, sum1, C_out, I_out)
end if
```

Convolution layers

- Allows to extract: H_{out}, K and Z
- Allows to deduce: S and P
- Russian dolls effect to get K
- With $\mathrm{N_p}$ the number of visible patterns corresponding to iterations of `for` loop (3-11):

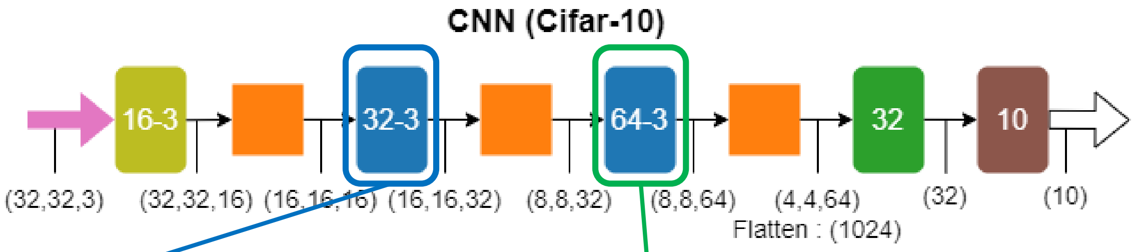
$$rowCnt\colon K = C_{out} = 2 \times N_p$$

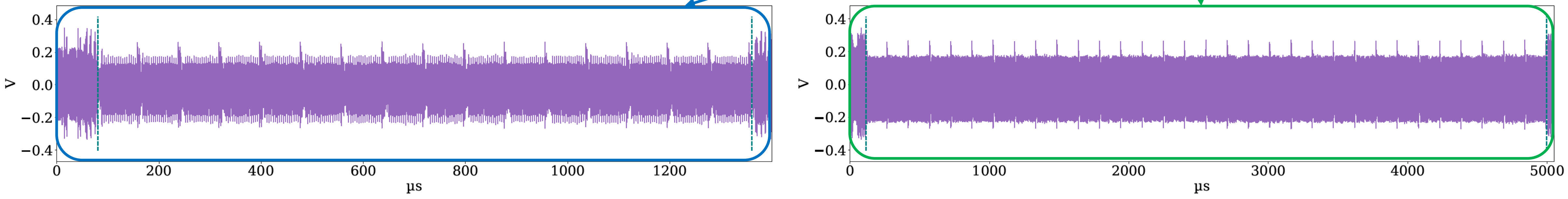
Algorithm 1 Matrix-product for convolution (**GeMM**)

Input: buff$_{in}$, ker, C_{out}, $C_{in} \times Z^2$, I_{out}, $bias$ (bias tensor)
Output: Partly filled I_{out}

1: $rowCnt \leftarrow C_{out} >> 1$ ▷ Set $rowCnt = K/2$
2: **for** $rowCnt > 0,\ rowCnt - -1$ **do** ▷ Iterate over $K/2$
3: $sum, sum1, sum2, sum3 = init\_sum(bias, C_{in} \times Z^2)$
4: $colCnt \leftarrow (C_{in} \times Z^2) >> 2$ ▷ Set $colCnt$
5: **for** $colCnt > 0,\ colCnt - -1$ **do**
6: $simd\_mac(sum, sum1, sum2, sum3, \mathrm{buff}_{in}, ker)$
7: **end for**
8: **if** $(C_{in} \times Z^2)$ & 0x3 **then**
9: $Manage\_colCnt\_remainder(sum, sum1, sum2, sum3, \mathrm{buff}_{in}, bias, C_{in} \times Z^2)$
10: **end if**
11: $apply\_mac(sum, sum1, sum2, sum3, C_{out}, I_{out})$
12: **end for**
13: **if** C_{out} & 0x3 **then** ▷ Check presence of remainders
14: $sum, sum1 = init\_sum(bias, C_{in} \times Z^2)$
15: $colCnt \leftarrow (C_{in} \times Z^2) >> 2$ ▷ Set $colCnt$
16: **for** $colCnt > 0,\ colCnt - -1$ **do**
17: $simd\_mac(sum, sum1, \mathrm{buff}_{in}, ker)$
18: **end for**
19: **if** $(C_{in} \times Z^2)$ & 0x3 **then**
20: $Manage\_colCnt\_remainder(sum, sum1, \mathrm{buff}_{in}, bias, C_{in} \times Z^2)$
21: **end if**
22: $apply\_mac(sum, sum1, C_{out}, I_{out})$
23: **end if**

Convolution layers





$N_p = 16$ iterations $\Leftrightarrow K = 32$

$N_p = 32$ iterations $\Leftrightarrow K = 64$

- With $\mathrm{N_p}$ the number of visible patterns corresponding to iterations of first `for` loop:

$$rowCnt\colon\ K = C_{out} = 2 \times N_p$$

Convolution layers

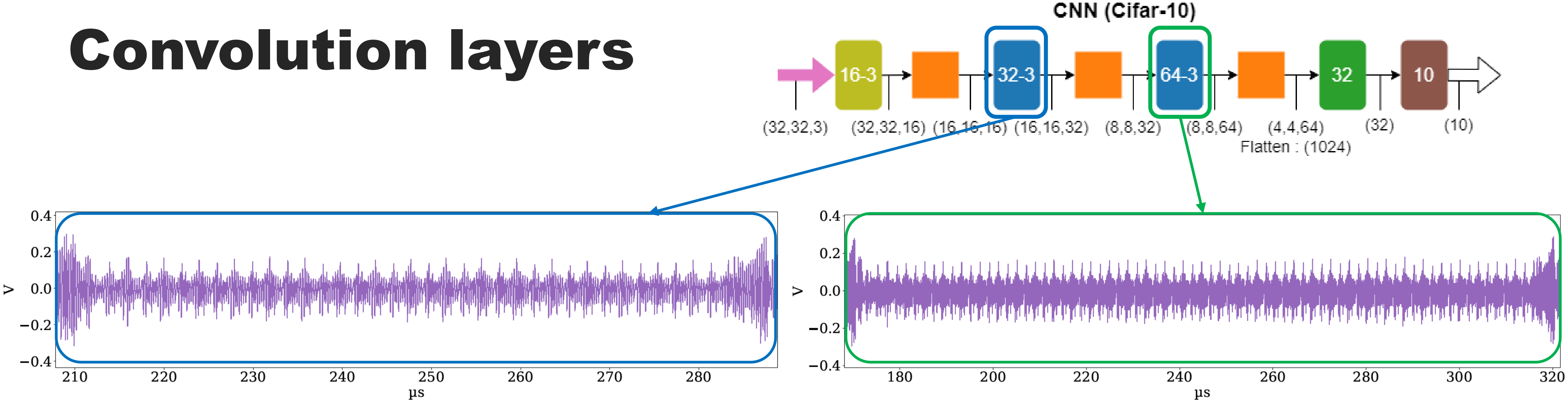
- Allows to extract: H_{out}, K and Z
- Allows to deduce: S and P
- Russian dolls effect to get K and Z
- $rowCnt$: $K = C_{out} = 2 \times N_p$
- $colCnt$: $Z = \sqrt{\frac{(4 \times N_p)}{C_{in}}}$

Algorithm 1 Matrix-product for convolution (**GeMM**)

Input: buff$_{in}$, ker, C_{out}, $C_{in} \times Z^2$, I_{out}, $bias$ (bias tensor)
Output: Partly filled I_{out}

```
rowCnt ← C_out >> 1                                          ▷ Set rowCnt = K/2
for rowCnt > 0, rowCnt − −1 do                               ▷ Iterate over K/2
    sum, sum1, sum2, sum3 = init_sum(bias, C_in × Z^2)
    colCnt ← (C_in × Z^2) >> 2                               ▷ Set colCnt
    for colCnt > 0, colCnt − −1 do
        simd_mac(sum, sum1, sum2, sum3, buff_in, ker)
    end for
    if (C_in × Z^2) & 0x3 then
        Manage_colCnt_remainder(sum, sum1, sum2, sum3, buff_in, bias, C_in × Z^2)
    end if
    apply_mac(sum, sum1, sum2, sum3, C_out, I_out)
end for
if C_out & 0x3 then                                          ▷ Check presence of remainders
    sum, sum1 = init_sum(bias, C_in × Z^2)
    colCnt ← (C_in × Z^2) >> 2                               ▷ Set colCnt
    for colCnt > 0, colCnt − −1 do
        simd_mac(sum, sum1, buff_in, ker)
    end for
    if (C_in × Z^2) & 0x3 then
        Manage_colCnt_remainder(sum, sum1, buff_in, bias, C_in × Z^2)
    end if
    apply_mac(sum, sum1, C_out, I_out)
end if
```

Convolution layers



$N_p = 36$ iterations $\Leftrightarrow Z = 3$ with $C_{in} = 16$

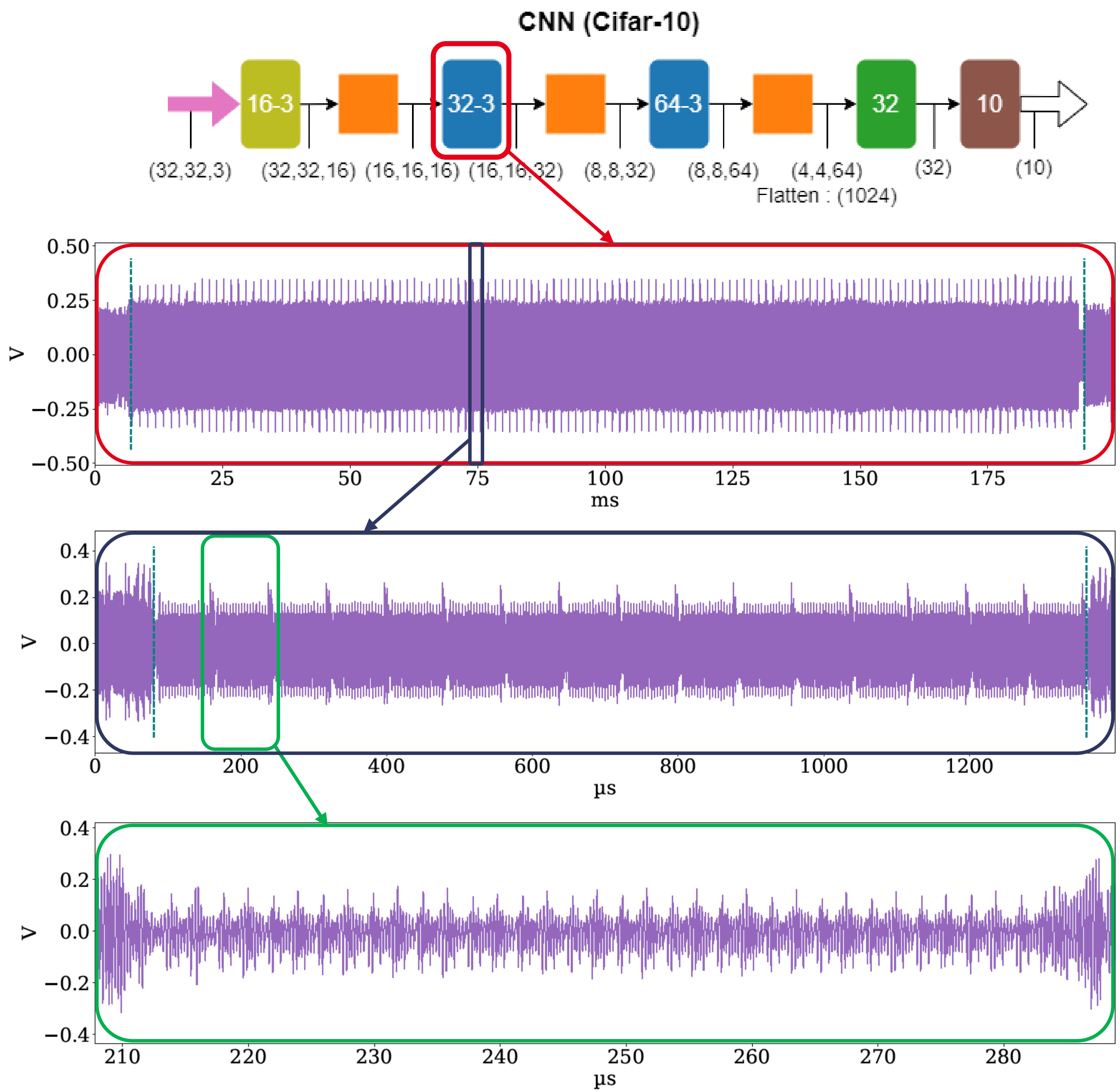
$N_p = 72$ iterations $\Leftrightarrow Z = 3$ with $C_{in} = 32$

- With $\mathrm{N_p}$ the number of visible patterns corresponding to iterations of second `for` loop:

$$colCnt\colon Z = \sqrt{\frac{(4 \times N_p)}{C_{in}}}$$

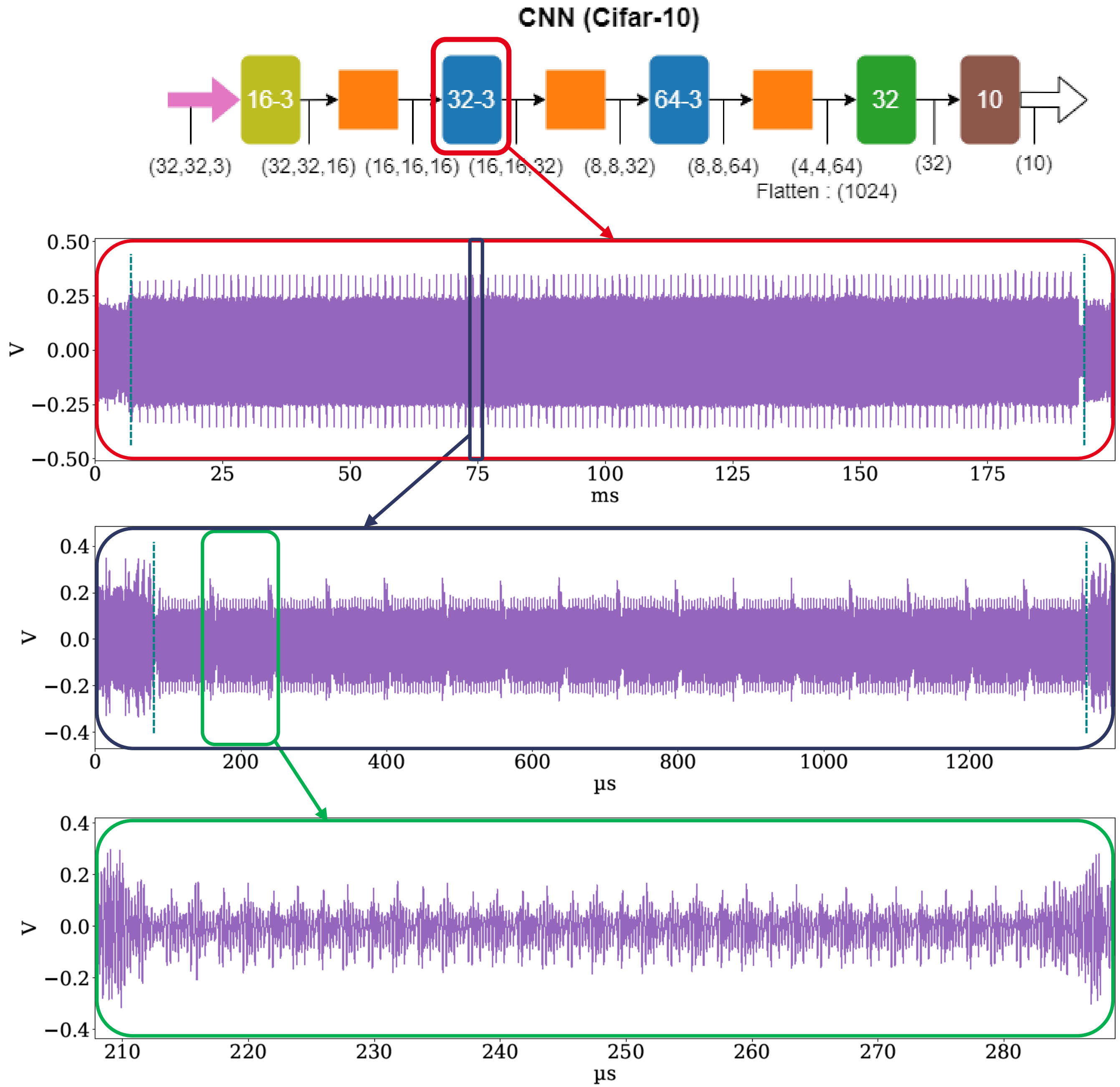
Convolution layers

- Look at `for` loops iterations
- Russian Dolls effect
- Allows to extract:
  - H_{out}
  - K
  - Z

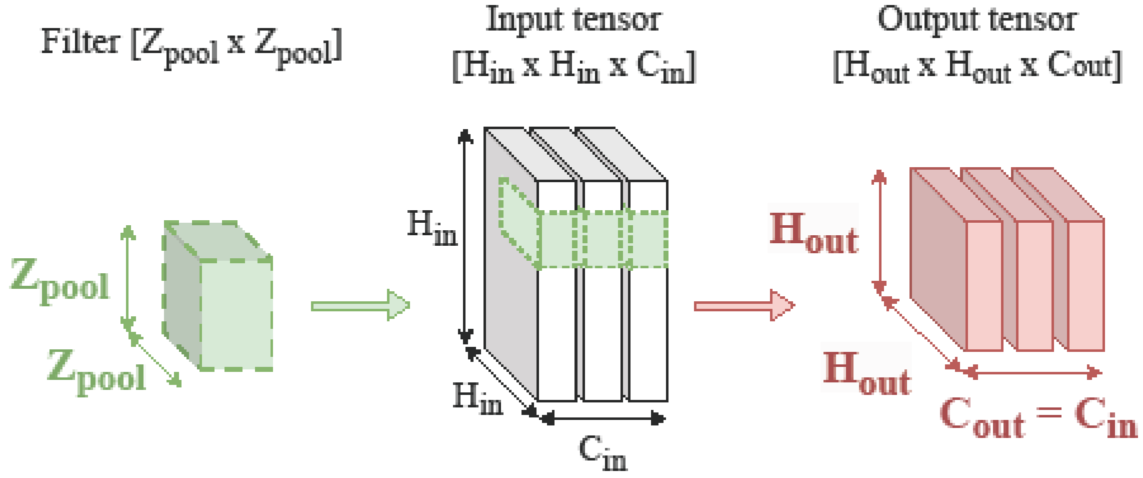


Convolution layers

- Look at `for` loops iterations
- Russian Dolls effect
- Allows to extract:
  - H_{out}
  - K
  - Z
- Then deducing:
  - S and P as $P \leq Z$



MaxPool layers



- Allows to extract H_{out} and deduce Z_{pool}
- 2 distinct parts

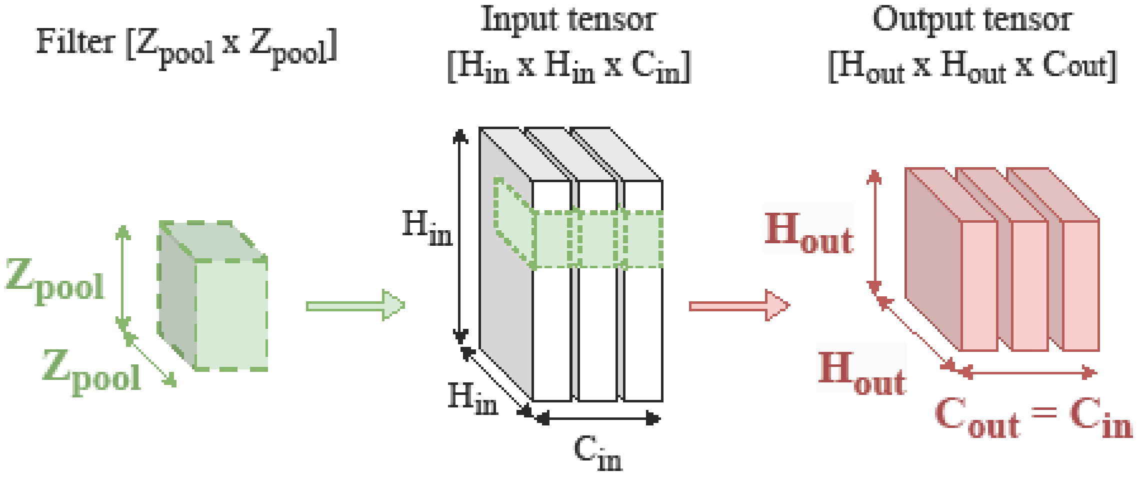
$$H_{out} = \frac{H_{in}}{Z_{pool}}$$

Algorithm 1 MaxPool - `arm_maxpool_q7_HWC`

Input: Input tensor I_{in} of size $H_{in}^2 \cdot C_{in}$, Output tensor I_{out} of size $H_{out}^2 \cdot C_{out}$, P, S, H_{ker}
Output: Filled I_{out}

for $i_y \leftarrow 0,\ i_y < H_{in},\ i_y++1$ **do** ▷ Pooling along x-axis
    for $i_x \leftarrow 0,\ i_x < H_{out},\ i_x++1$ **do**
        $win_{start}, win_{stop} \leftarrow set\_window(i_y, i_x, I_{in}, H_{ker}, P, S)$
        $compare\_and\_remplace\_if\_larger(win_{start}, win_{stop}, i_y, i_x, I_{in})$
    end for
end for
for $i_y \leftarrow 0,\ i_y < H_{out},\ i_y++1$ **do** ▷ Pooling along y-axis, iterates directly over H_{out}
    $row_{start}, row_{stop} \leftarrow set\_rows(i_y, I_{in}, I_{out}, H_{ker}, P, S)$
    $compare\_replace\_then\_apply(row_{start}, row_{stop}, I_{in}, I_{out})$
end for

MaxPool layers



- Allows to extract H_{out} and deduce Z_{pool}
- 2 distinct parts

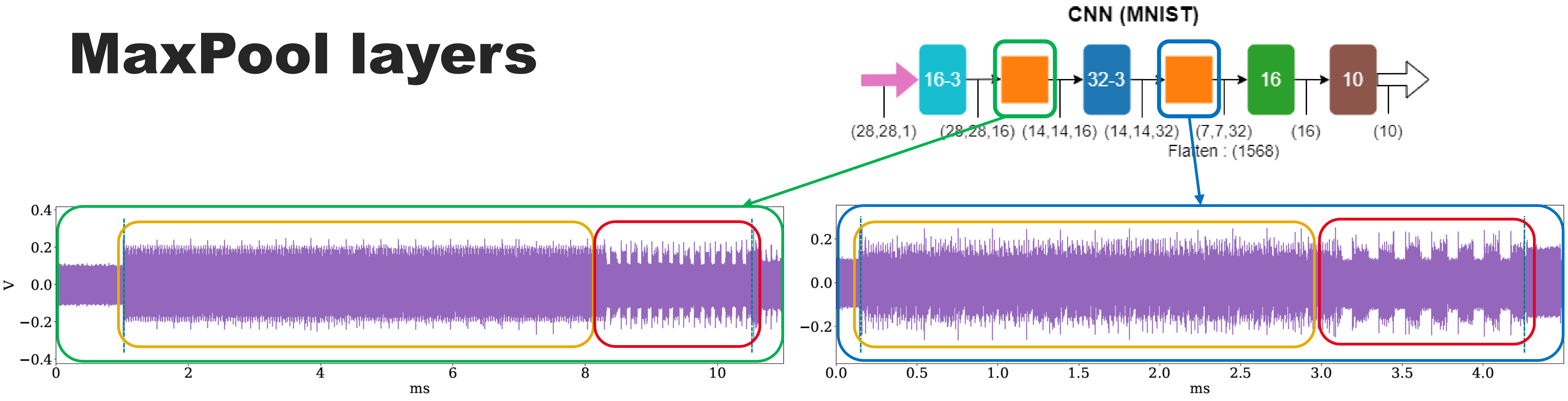
$$H_{out} = \frac{H_{in}}{Z_{pool}}$$

Algorithm 1 MaxPool - `arm_maxpool_q7_HWC`

Input: Input tensor I_{in} of size $H_{in}^2 \cdot C_{in}$, Output tensor I_{out} of size $H_{out}^2 \cdot C_{out}$, P, S, H_{ker}
Output: Filled I_{out}

```
for i_y ← 0, i_y < H_in, i_y++1 do                 ▷ Pooling along x-axis
    for i_x ← 0, i_x < H_out, i_x++1 do
        win_start, win_stop ← set_window(i_y, i_x, I_in, H_ker, P, S)
        compare_and_remplace_if_larger(win_start, win_stop, i_y, i_x, I_in)
    end for
end for
for i_y ← 0, i_y < H_out, i_y++1 do    ▷ Pooling along y-axis, iterates directly over H_out
    row_start, row_stop ← set_rows(i_y, I_in, I_out, H_ker, P, S)
    compare_replace_then_apply(row_start, row_stop, I_in, I_out)
end for
```

MaxPool layers



$N_p = 14$ iterations $\Leftrightarrow H_{out} = 14$

$N_p = 7$ iterations $\Leftrightarrow H_{out} = 7$

- Look at `for` loops iterations
- Two distinct parts: one easier to analyse

- Allows to extract: H_{out}
- Allows to deduce: Z_{pool}

Dense layers

- Neurons managed by groups of 4
- Patterns length depends on input shape

Algorithm 1 Dense layer - `arm_fully_connected_q7_opt`

Input: Input vector I_{in} of size H_{in}, weight vector ker of size N_e, bias matrix $bias$ of size N_e, output vector I_{out} of size H_{out}, P, S, H_{ker}

Output: Filled I_{out}

$rowCnt \leftarrow N_e >> 2$ ▷ Nb. neurons divided by 4

for $rowCnt > 0,\ rowCnt - -1$ **do** ▷ Iterate directly over $N_e/4$

$sum, sum1, sum2, sum3 = init\_sum\_with\_bias(bias, rowCnt)$

$colCnt \leftarrow H_{in} >> 2$

for $colCnt > 0,\ colCnt - -1$ **do**

$simd\_mac(sum, sum1, sum2, sum3, ker, colCnt)$

end for

$apply\_mac(sum, sum1, sum2, sum3, rowCnt, I_{out})$

end for

$rowCnt \leftarrow N_e\ \&\ 0x3$ ▷ Manage remainders if any

for $rowCnt > 0,\ rowCnt - -1$ **do**

$sum = init\_sum\_with\_bias(bias, rowCnt)$

$colCnt \leftarrow H_{in} >> 2$

for $colCnt > 0,\ colCnt - -1$ **do**

$mac(sum, ker, colCnt)$

end for

$apply\_mac(sum, rowCnt, I_{out})$

end for

Dense layers

- Neurons managed by groups of 4
- Patterns length depends on input shape
- Allows to extract:
  - $N_e = 4 \times N_p$

Algorithm 1 Dense layer - `arm_fully_connected_q7_opt`

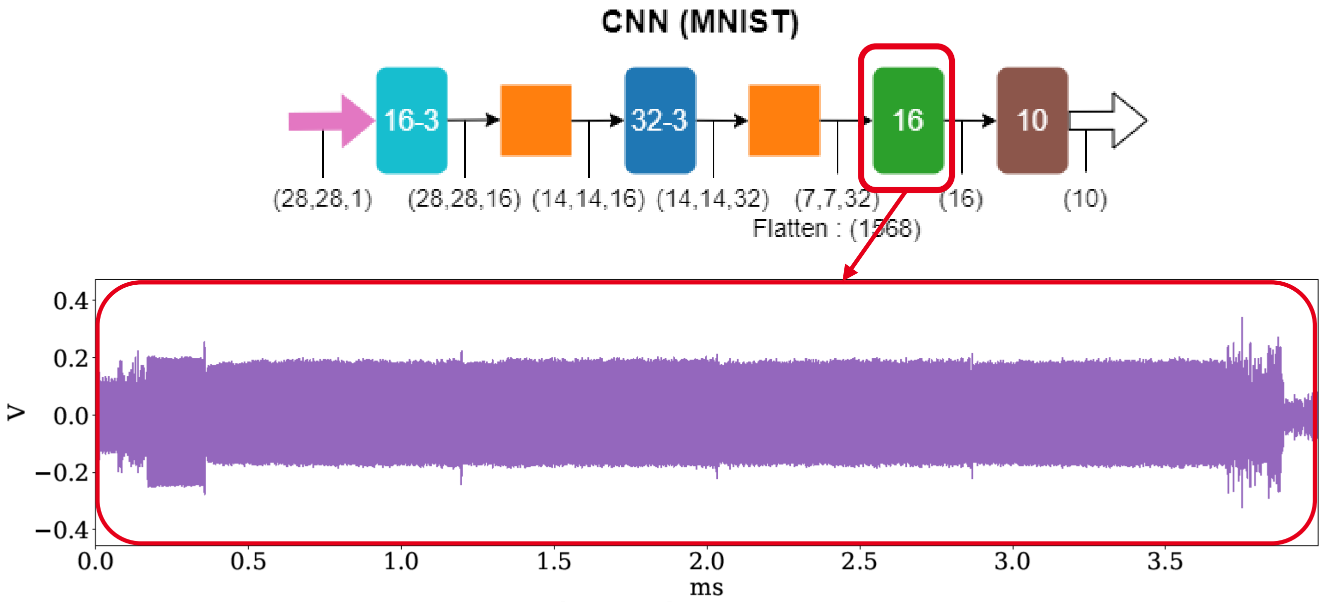
Input: Input vector I_{in} of size H_{in}, weight vector ker of size N_e, bias matrix $bias$ of size N_e, output vector I_{out} of size H_{out}, P, S, H_{ker}

Output: Filled I_{out}

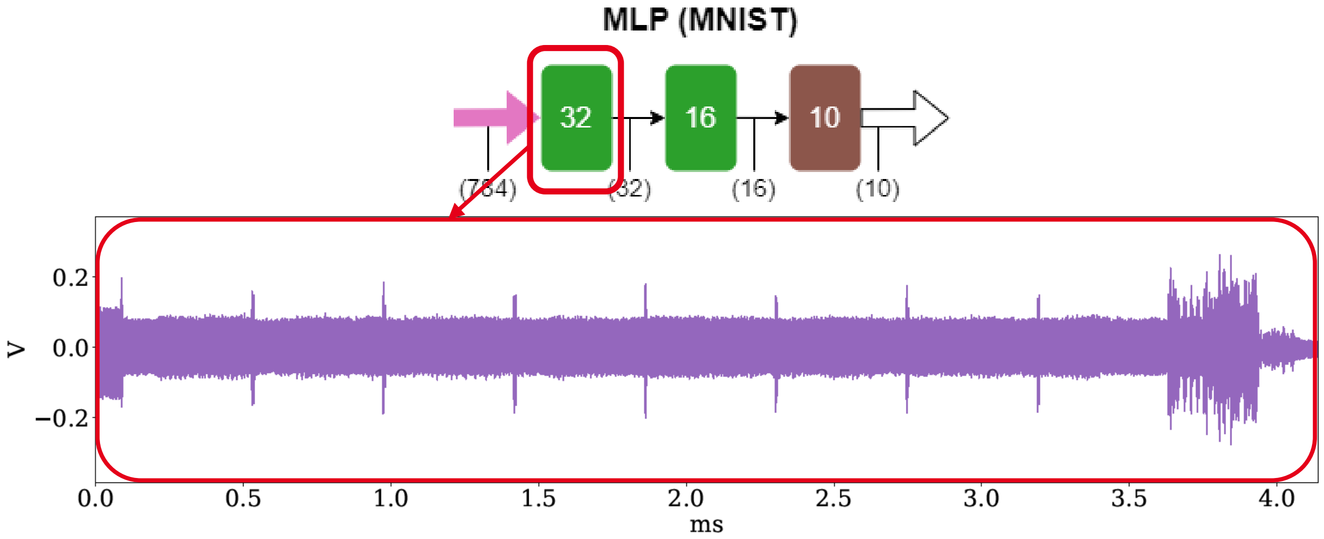
```
rowCnt ← N_e >> 2                                        ▷ Nb. neurons divided by 4
for rowCnt > 0, rowCnt − −1 do                           ▷ Iterate directly over N_e/4
    sum, sum1, sum2, sum3 = init_sum_with_bias(bias, rowCnt)
    colCnt ← H_in >> 2
    for colCnt > 0, colCnt − −1 do
        simd_mac(sum, sum1, sum2, sum3, ker, colCnt)
    end for
    apply_mac(sum, sum1, sum2, sum3, rowCnt, I_out)
end for
rowCnt ← N_e & 0x3                                       ▷ Manage remainders if any
for rowCnt > 0, rowCnt − −1 do
    sum = init_sum_with_bias(bias, rowCnt)
    colCnt ← H_in >> 2
    for colCnt > 0, colCnt − −1 do
        mac(sum, ker, colCnt)
    end for
    apply_mac(sum, rowCnt, I_out)
end for
```

Dense layers

- Neuron managed by groups of 4
- Patterns length depends on input shape
- Allows to extract:
  - $N_e = 4 \times N_p$
- Usual cases when $N_e \% \, 4 = 0$



$N_p = 4$ iterations $\Leftrightarrow N_e = 16$



$N_p = 8$ iterations $\Leftrightarrow N_e = 32$

Dense layers

- Neuron managed by groups of 4
- Patterns length depends on input shape
- Allows to extract:
  - $N_e = 4 \times N_p$
- Usual cases when $N_e \% \, 4 = 0$
- Special cases when $N_e \% \, 4 \neq 0$

Algorithm 1 Dense layer - `arm_fully_connected_q7_opt`

Input: Input vector I_{in} of size H_{in}, weight vector ker of size N_e, bias matrix $bias$ of size N_e, output vector I_{out} of size H_{out}, P, S, H_{ker}

Output: Filled I_{out}

$rowCnt \leftarrow N_e >> 2$ ▷ Nb. neurons divided by 4

for $rowCnt > 0, \; rowCnt - -1$ **do** ▷ Iterate directly over $N_e/4$

    $sum, sum1, sum2, sum3 = init\_sum\_with\_bias(bias, rowCnt)$

    $colCnt \leftarrow H_{in} >> 2$

    for $colCnt > 0, \; colCnt - -1$ **do**

        $simd\_mac(sum, sum1, sum2, sum3, ker, colCnt)$

    end for

    $apply\_mac(sum, sum1, sum2, sum3, rowCnt, I_{out})$

end for

$rowCnt \leftarrow N_e \; \& \; 0x3$ ▷ Manage remainders if any

for $rowCnt > 0, \; rowCnt - -1$ **do**

    $sum = init\_sum\_with\_bias(bias, rowCnt)$

    $colCnt \leftarrow H_{in} >> 2$

    for $colCnt > 0, \; colCnt - -1$ **do**

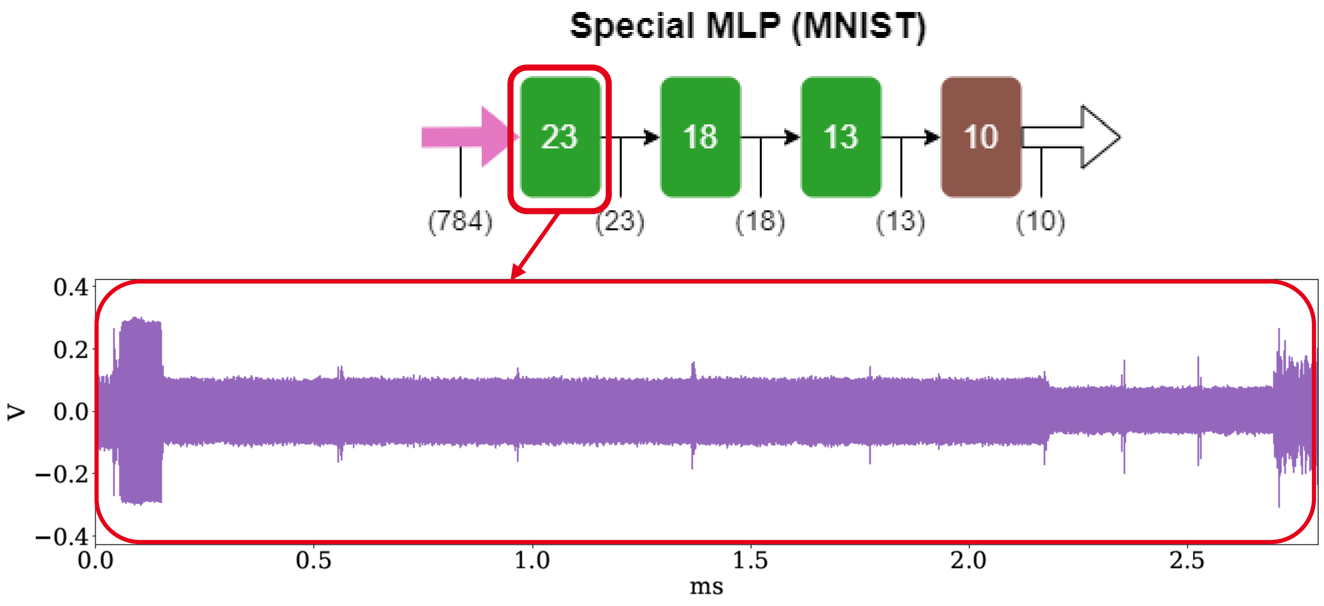
        $mac(sum, ker, colCnt)$

    end for

    $apply\_mac(sum, rowCnt, I_{out})$

end for

Dense layers

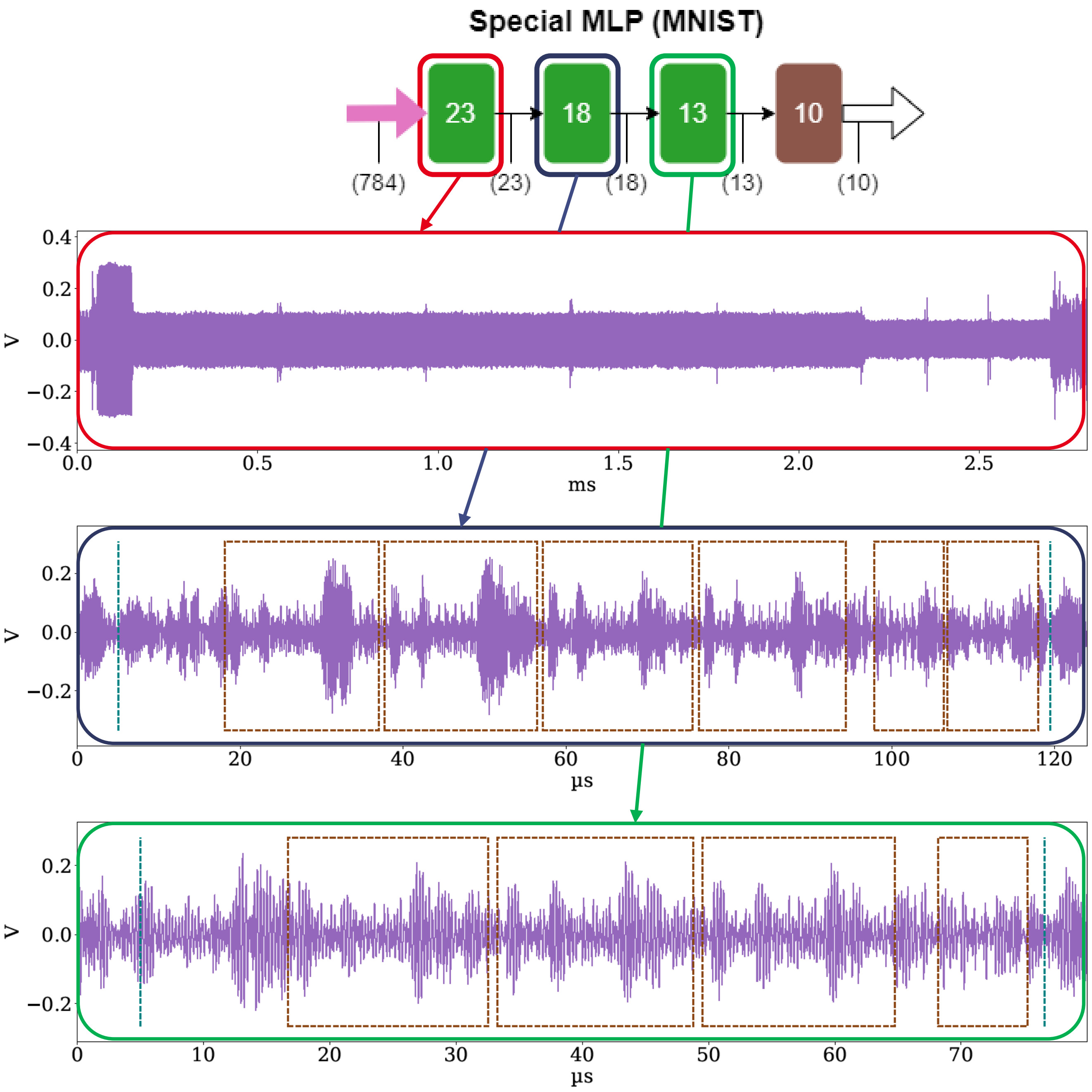


$N_p = 5 + 3$ iterations $\Leftrightarrow N_e = 4 \times 5 + 3 = 23$

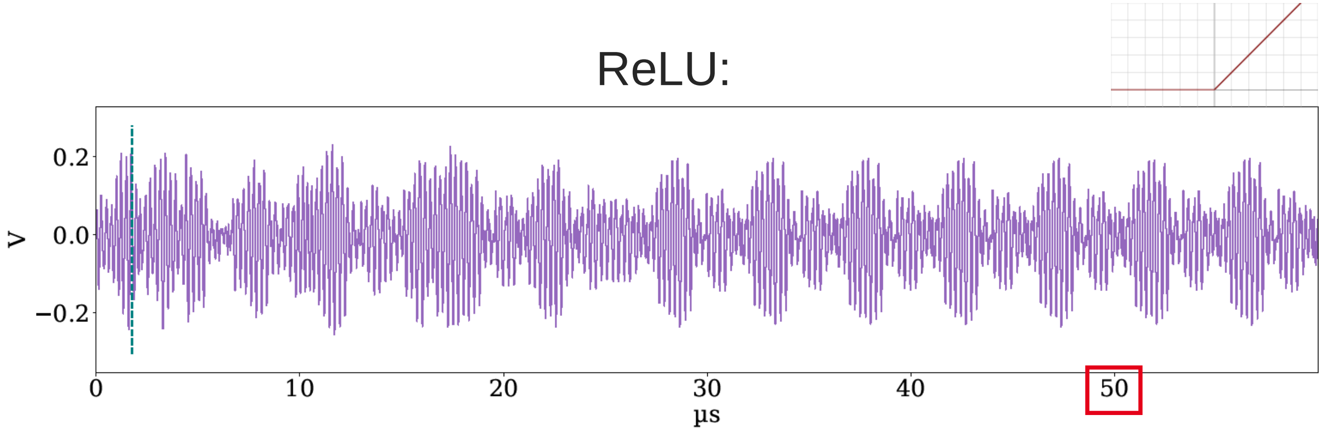
- Neuron managed by groups of 4
- Patterns length depends on input shape
- Allows to extract:
  - $N_e = 4 \times N_p$
- Usual cases when $N_e \% \, 4 = 0$
- Special cases when $N_e \% \, 4 \neq 0$

Dense layers

- Neuron managed by groups of 4
- Patterns length depends on input shape
- Allows to extract:
  - $N_e = 4 \times N_p$
- Usual cases when $N_e \% \, 4 = 0$
- Special cases when $N_e \% \, 4 \neq 0$
- Illustration with additional triggers



Activation layers



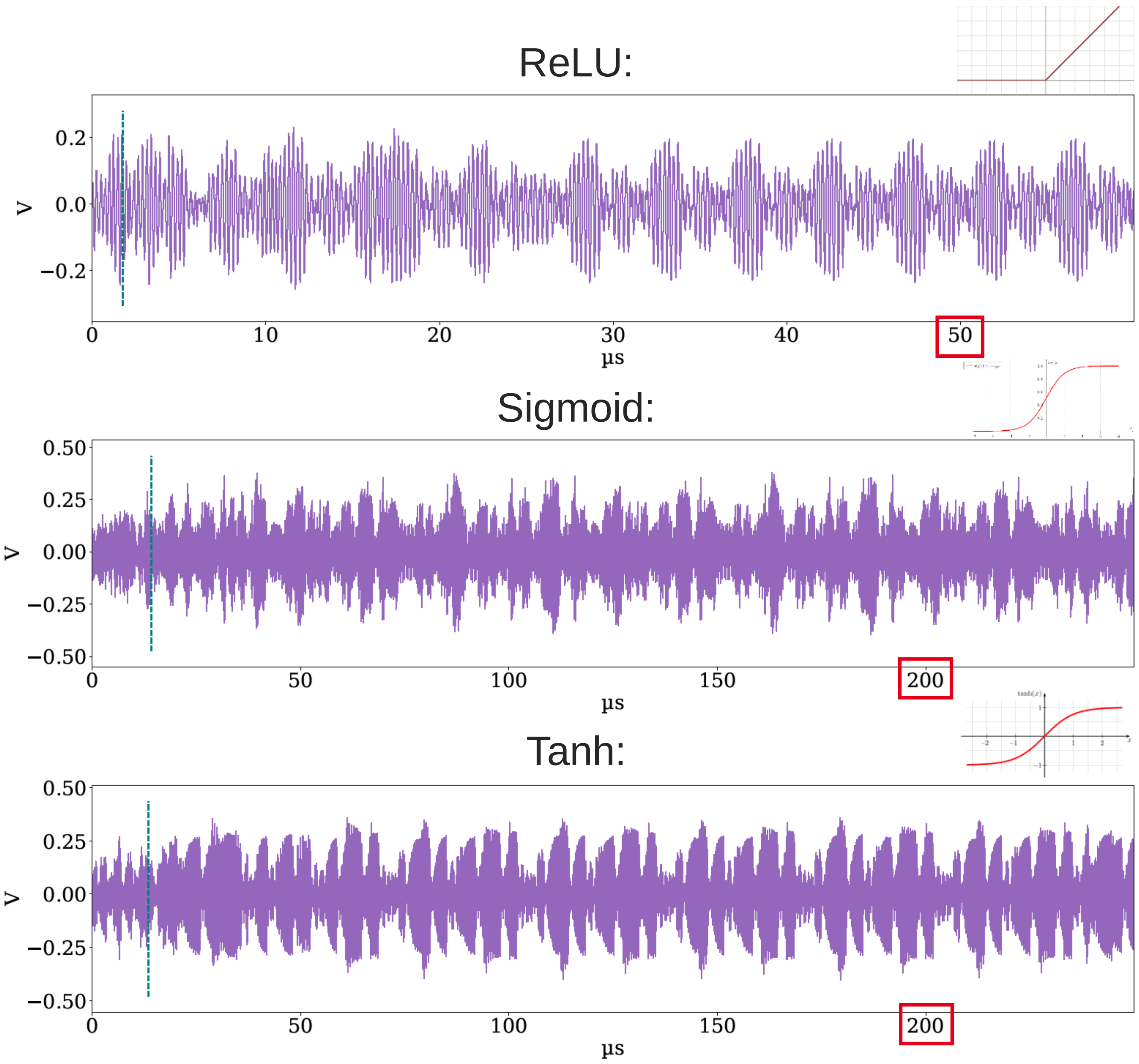
- Distinguish ReLU from Tanh and Sigmoid
- Can use duration of activation layers
- EM patterns give additional hints

Layer	ReLU	Sigmoid	Tanh
0	1,84 ms	8,24 ms	8,24 ms
1	0,92 ms	3,21 ms	3,58 ms
2	7,60 µs	15,6 µs	20,64 µs

Activation layers

- Distinguish ReLU from Tanh and Sigmoid
- Can use duration of activation layers
- EM patterns give additional hints

Layer	ReLU	Sigmoid	Tanh
0	1,84 ms	8,24 ms	8,24 ms
1	0,92 ms	3,21 ms	3,58 ms
2	7,60 µs	15,6 µs	20,64 µs



Outline

- Introduction
- Challenges
- Scope and threat model
- Model analysis
- Layer analysis
- Discussion & Perspectives

Discussions and Perspectives

- Several works studying architecture extraction
- Attack surface for such a threat is significantly extended by SCA
- No need of complex exploitation methods (e.g., heavy supervised profiling step)
- Important step for parameters extraction [1 to 7]
- Some more complex cases allowing to approximate hyper-parameter (could be handled with pattern detection tools)
- Critical need to protect architecture from extraction [11, 17, 18]
- Hard challenges to develop efficient protections 32-bit microcontrollers

References

1. Carlini, N. et al. "Cryptanalytic extraction of neural network models." Annual International Cryptology Conference. 2020.
2. Canales-Martínez, I., et al. "Polynomial Time Cryptanalytic Extraction of Neural Network Models." arXiv 2023.
3. Jagielski, M., et al. "High accuracy and high fidelity extraction of neural networks." 29th USENIX security symposium (USENIX Security 20). 2020.
4. Rakin, A. et al. "Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories." IEEE SP, 2022.
5. Hector, K,. et al. "Fault Injection and Safe-Error Attack for Extraction of Embedded Neural Network Models." SECAI (ESORICS Workshop) 2023.
6. Joud, R, et al. "A Practical Introduction to Side-Channel Extraction of Deep Neural Network Parameters." CARDIS 2022.
7. Gongye, C. et al. "Reverse-engineering deep neural networks using floating-point timing side-channels." ACM/IEEE DAC, 2020.
8. Duddu, V., et al. "Stealing neural networks via timing side channels." arXiv 2018.
9. Yu, H. et al. "Deepem: Deep neural networks model recovery through em side-channel information leakage." IEEE HOST 2020.
10. Yli-Mäyry, V., et al. "Extraction of binarized neural network architecture and secret parameters using side-channel information." IEEE ISCAS. 2021.
11. Luo, Y., et al. "NNReArch: A Tensor Program Scheduling Framework Against Neural Network Architecture Reverse Engineering." IEEE FCCM. 2022.
12. Chmielewski, L. et al. "On reverse engineering neural network implementation on gpu." ACNS 2021, Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, 2021.
13. Batina, L., et al. "CSI-NN: Reverse engineering of neural network architectures through electromagnetic side channel." USENIX Security. 2019.
14. Xiang, Y., et al. "Open dnn box by power side-channel attack." IEEE Transactions on Circuits and Systems II: Express Briefs 2020
15. Ma, J.: A higher-level Neural Network library on Microcontrollers (NNoM) (2020).
16. Lai, L. et al. "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus." arXiv. 2018.
17. Chabanne, H., et al. "Telepathic headache: Mitigating cache side-channel attacks on convolutional neural networks." ACNS, 2021.
18. Li, J., et al. "Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing." IEEE HOST. 2021.

Acknowledgements

- This work is supported by (CEA-Leti) the European project InSecTT (ECSEL JU 876038) and by the French ANR in the framework of the Investissements d'avenir program (ANR-10-AIRT-05, irtnanoelec); and (Mines Saint-Etienne) by the French program ANR PICTURE (AAPG2020).

- This work benefited from the French Jean Zay supercomputer with the AI dynamic access program.

Thank you for your attention

Contacts:

Pierre-Alain Moëllic
pierre-alain.moellic@cea.fr

Simon Pontié
Simon.pontie@cea.fr

Raphaël Joud
raphael.joud@cea.fr

Jean-Baptiste Rigaud
rigaud@emse.fr