SAL
SILICON AUSTRIA LABS

# VLSI FOR MACHINE LEARNING

PEDRO JULIAN, DIEGO GIGENA, NICOLÁS RODRÍGUEZ

CAE 2023, Córdoba

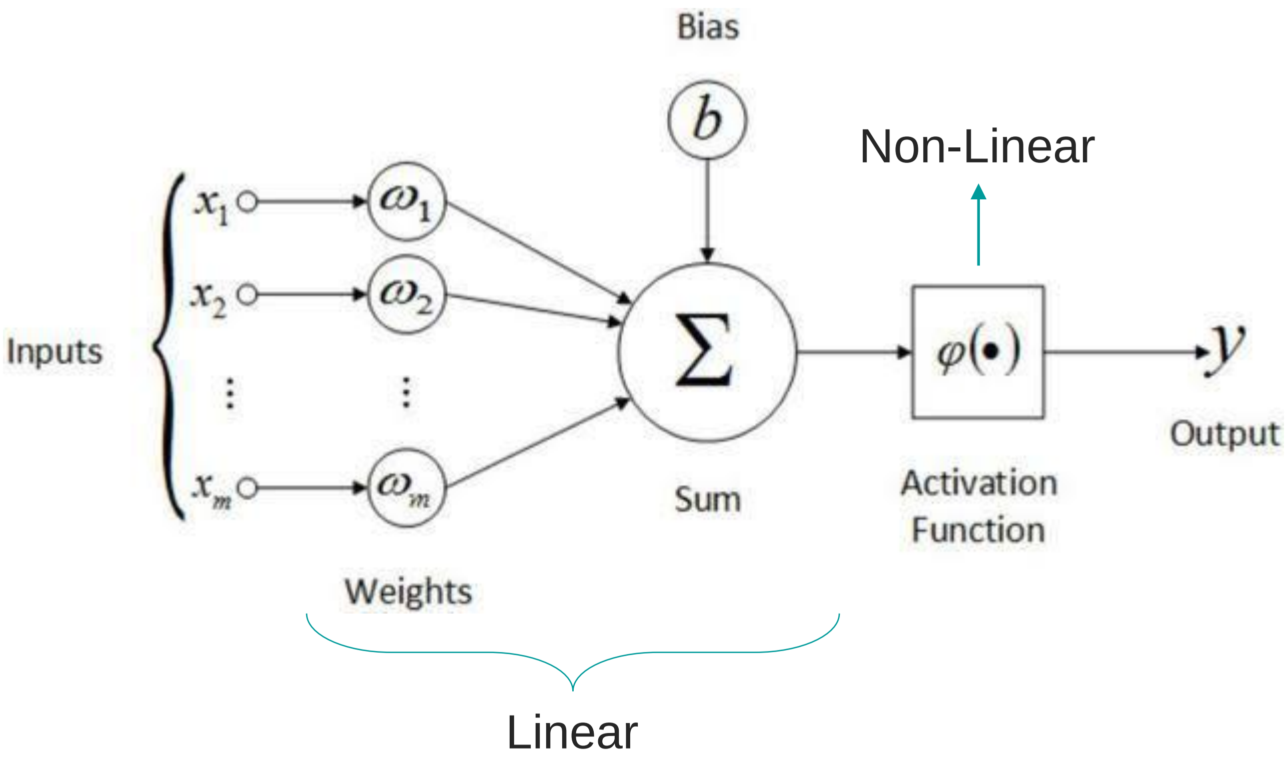
SAL
SILICON AUSTRIA LABS

# 2. BASIC BLOCKS OF DNN

# WHAT IS A NEURON

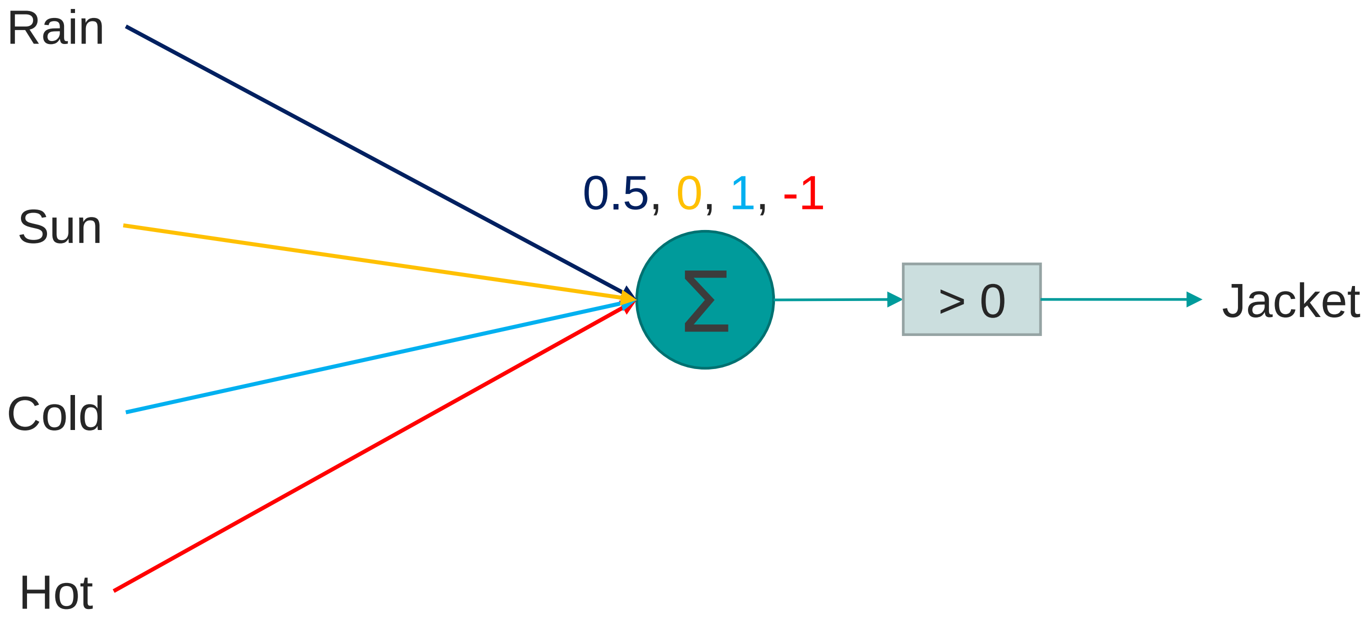
# THE PERCEPTRON

- Simple model of a Neuron
- Foundational Element of NN
- Components:
  - Input Values
  - Weighted connections
  - Adder with Bias
  - Activation Function (AF)
- How does it work?
  - Adition of weighted values
  - Decision using AF



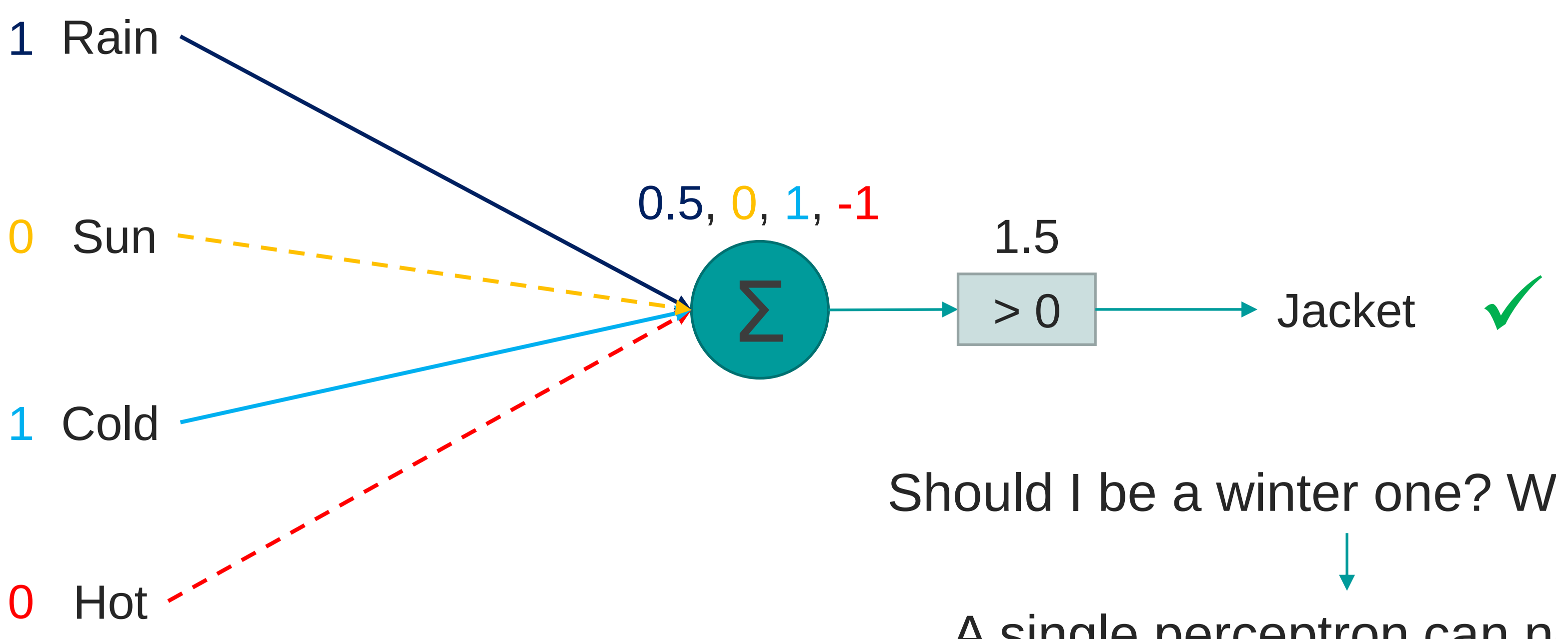
# EXAMPLE

How‘s the weather like? → Should I wear a jacket?



# EXAMPLE

How‘s the weather like? → Should I wear a jacket?




Should I be a winter one? Waterproof?

↓

A single perceptron can not tell…

# WHY ARTIFICIAL NEURONS

SAL SILICON AUSTRIA LABS

- Relatively simple models of a real neuron
- Group of neurons as universal approximator
- Can solve complex tasks
- And most importantly…

They can learn!!!

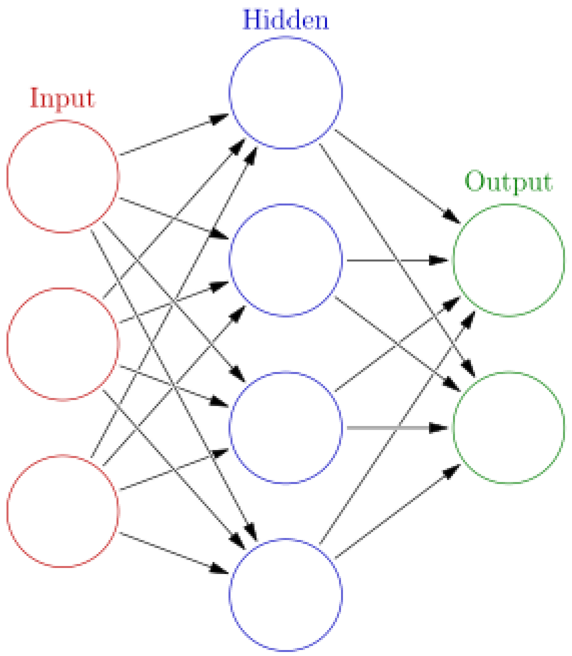
SAL SILICON AUSTRIA LABS

# DEEP NEURAL NETWORKS

# ARTIFICIAL NEURAL NETWORK

SAL
SILICON AUSTRIA LABS

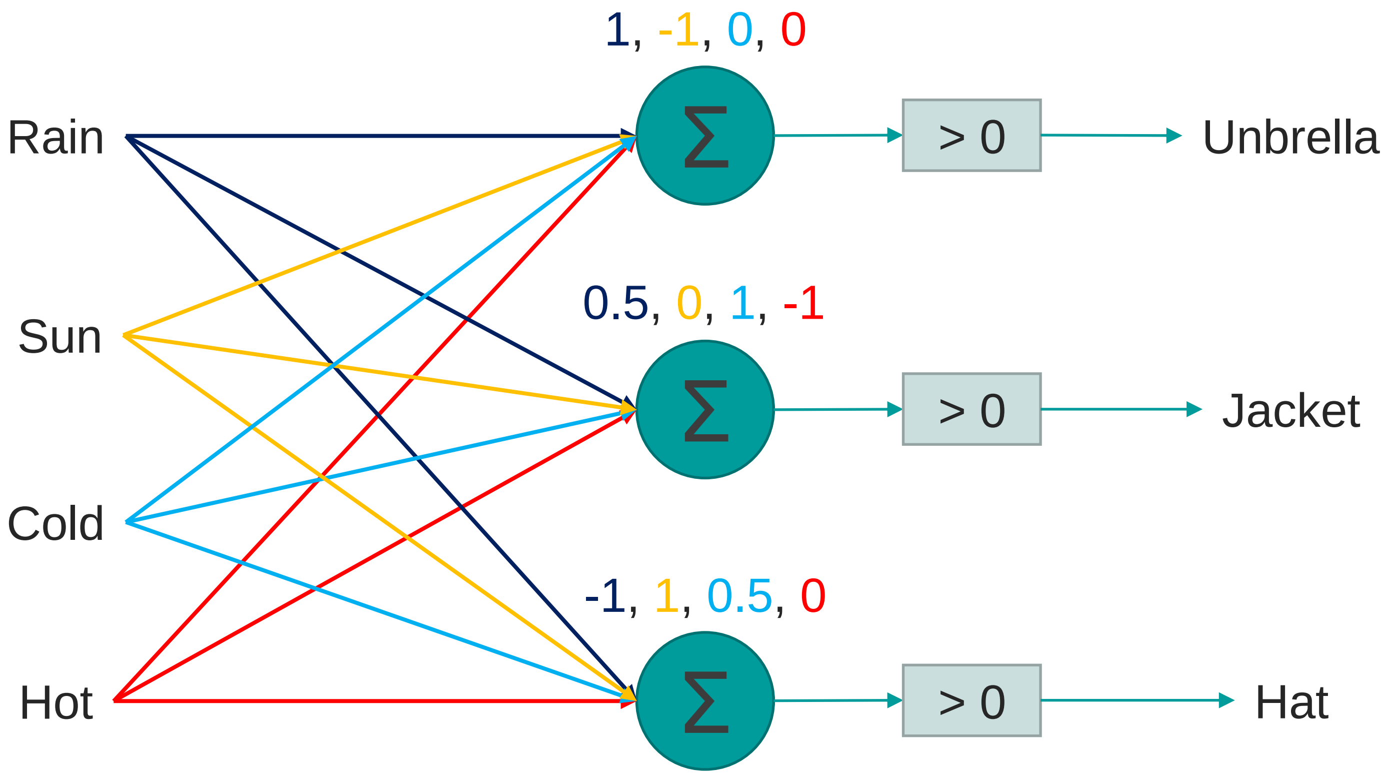
- Group of neurons to solve more complex tasks
- Groups with shared inputs, between input and output, compose a “Hidden Layer”
- When all inputs are shared:
  - Fully-Connected Layer
  - Dense Layer
- A single Hidden Layer is enough to be called ANN



# EXPANDED EXAMPLE

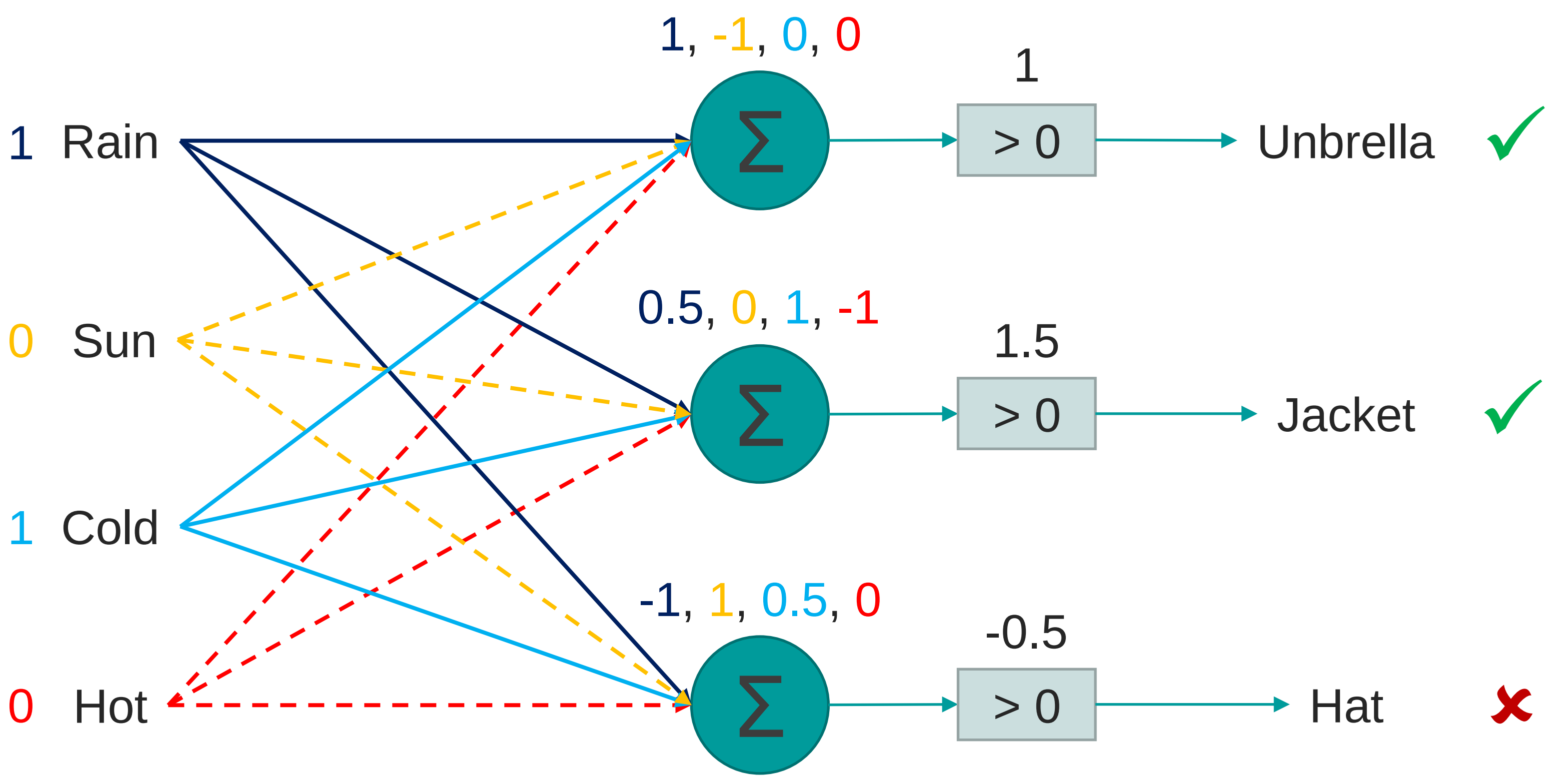
SAL
SILICON AUSTRIA LABS

How‘s the weather like? → What should I wear?



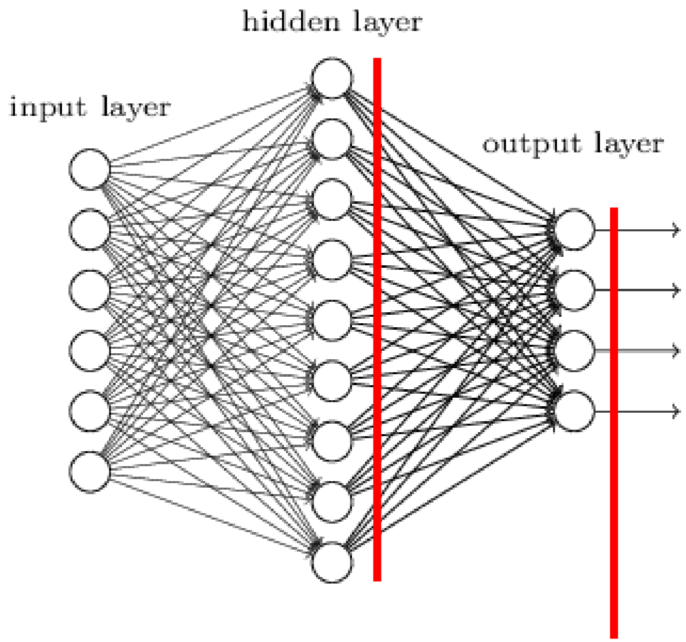
# EXPANDED EXAMPLE

How‘s the weather like? → What should I wear?



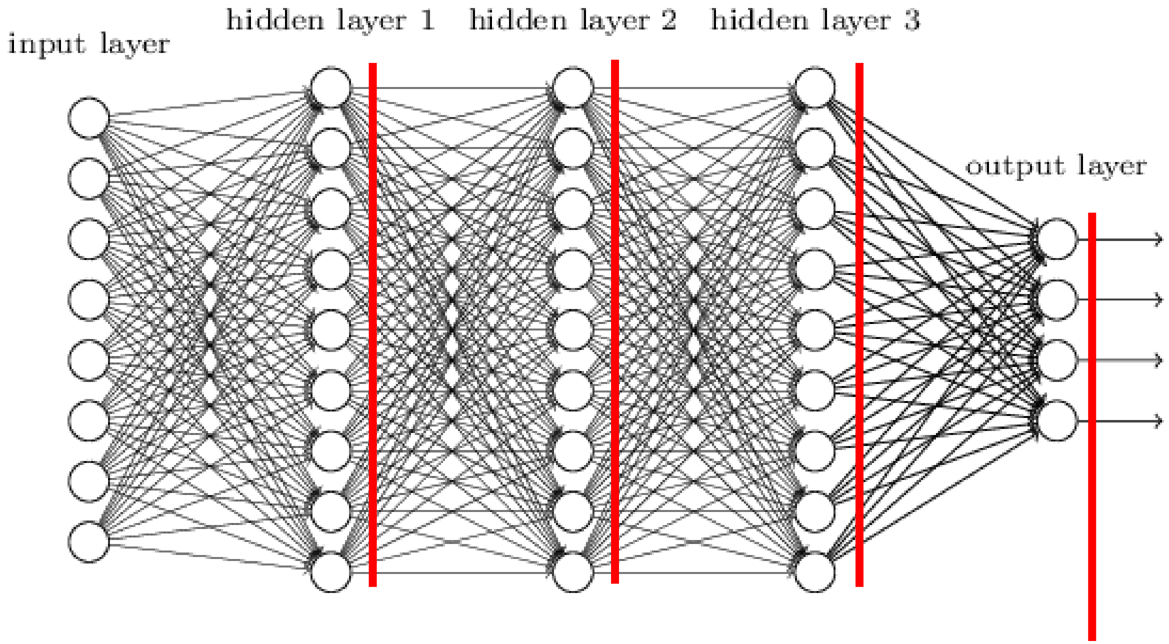
# WHAT IS DEEP

"Non-deep" feedforward neural network




Umbrella, Jacket and/or Hat

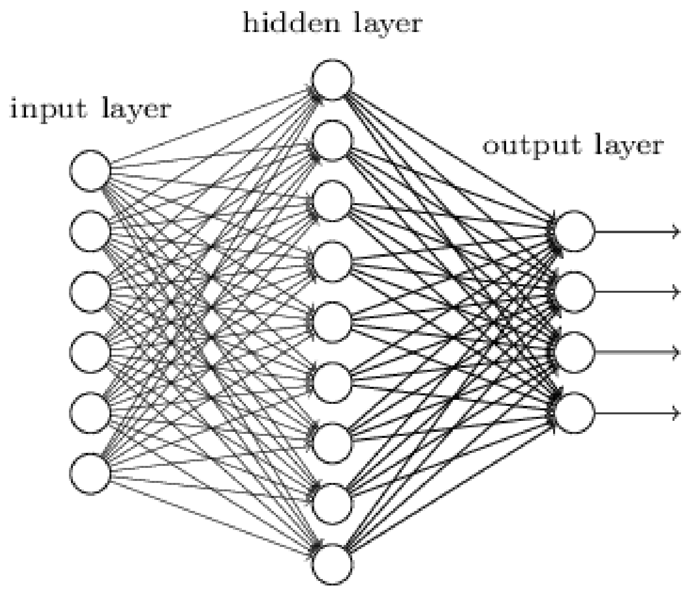
Deep neural network



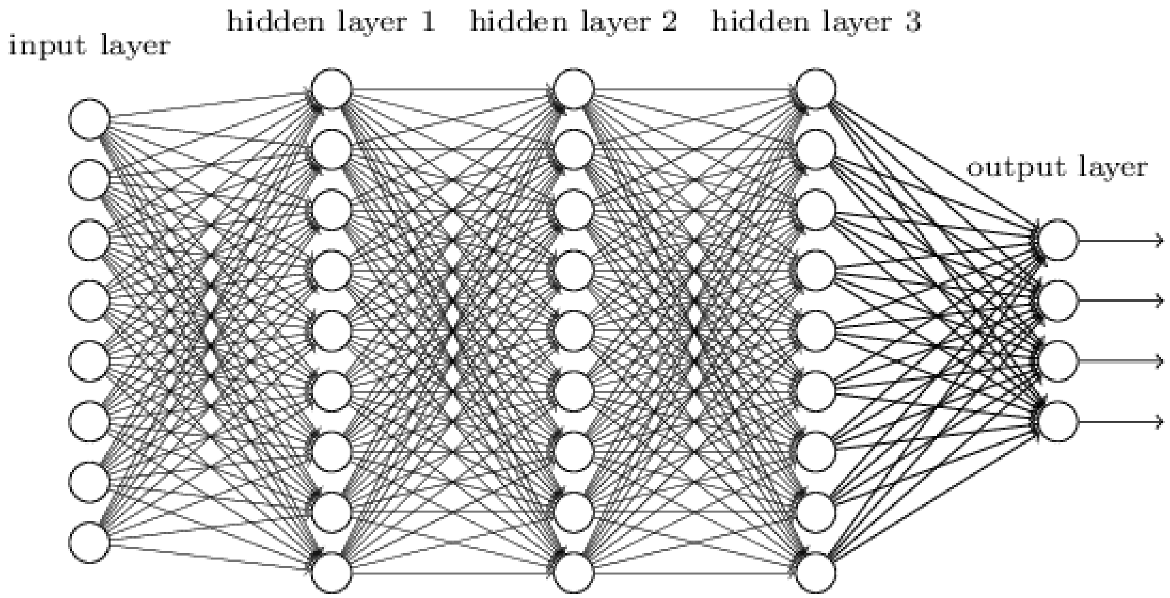

Winter waterproof Jacket

# WHAT IS DEEP

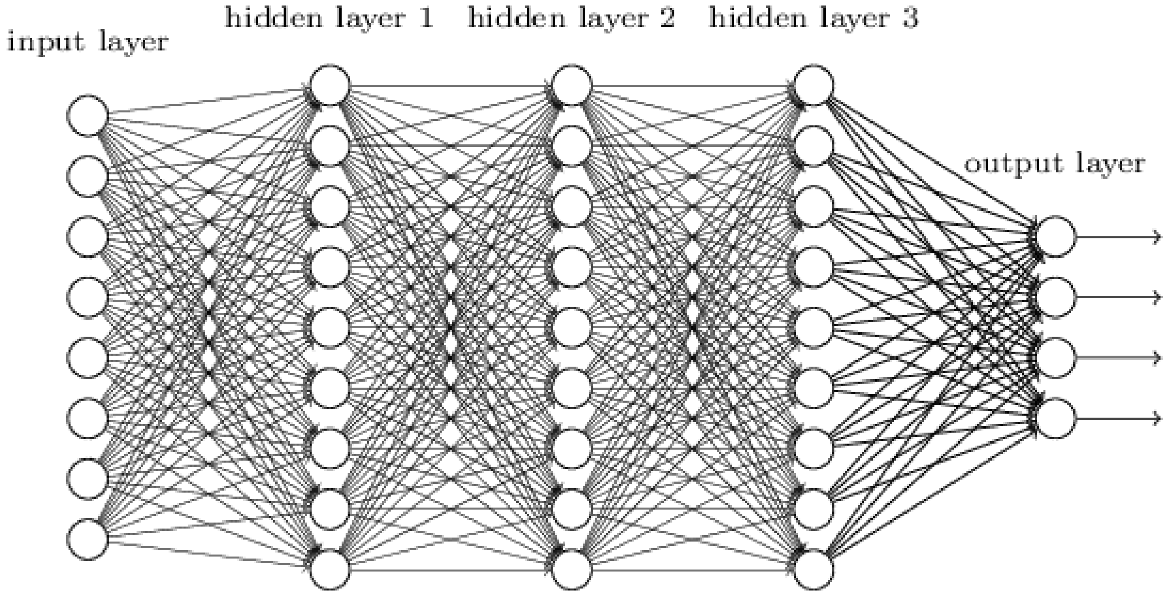
- No Activation Functions?

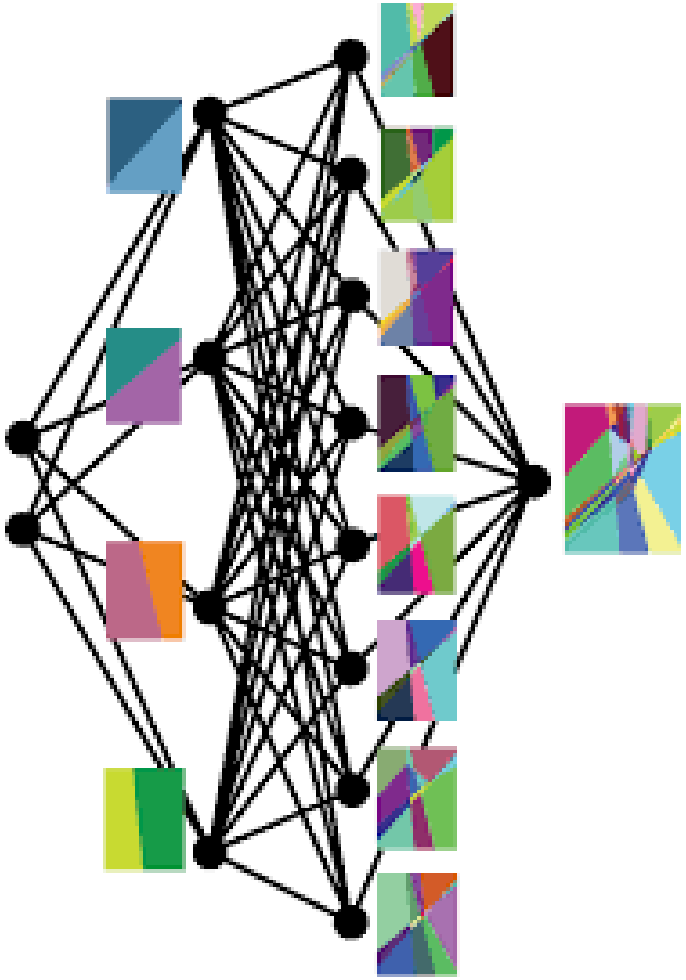



≡



# LINEAR REGIONS

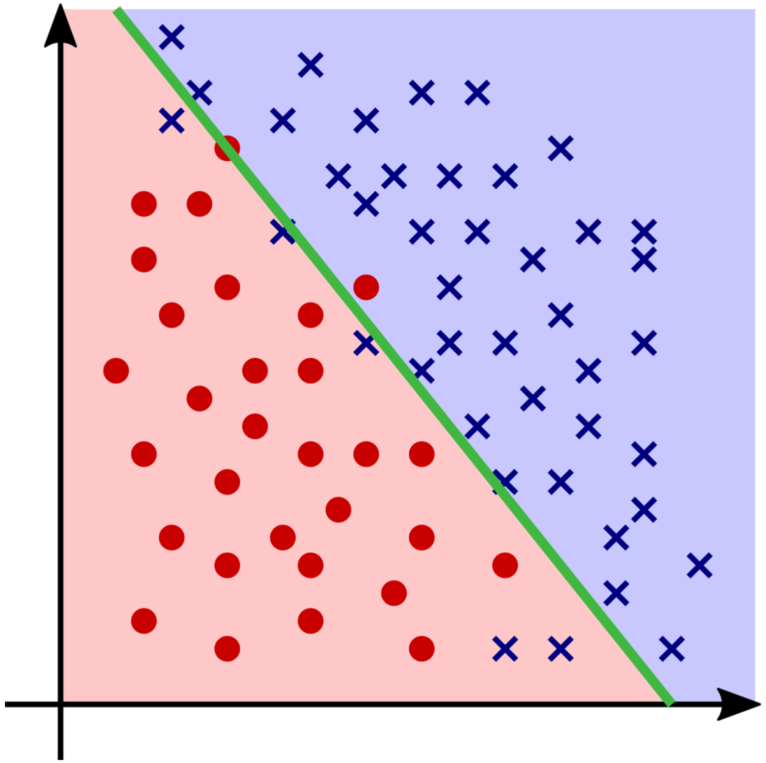


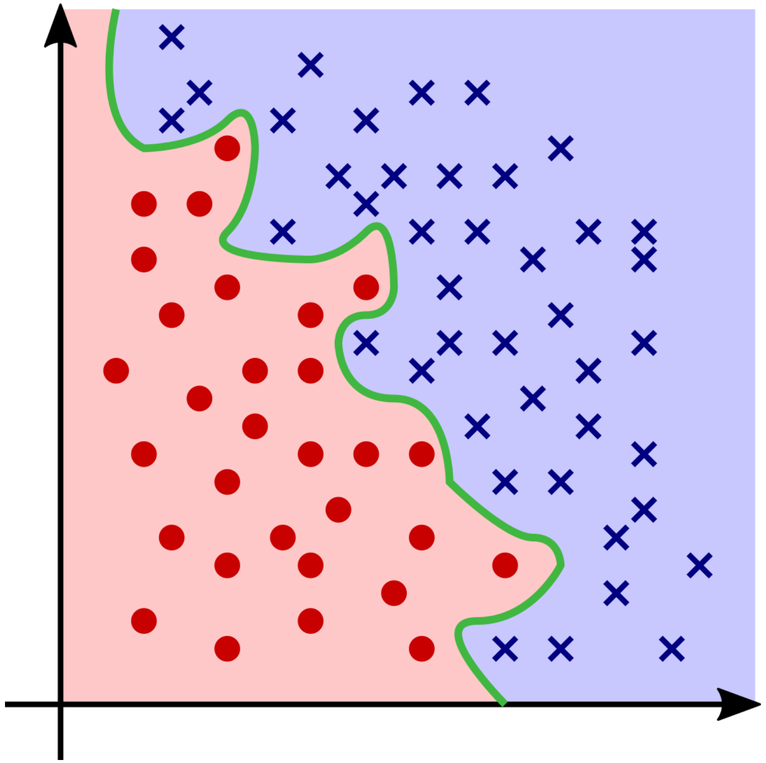
# ADDING LAYERS

SAL SILICON AUSTRIA LABS

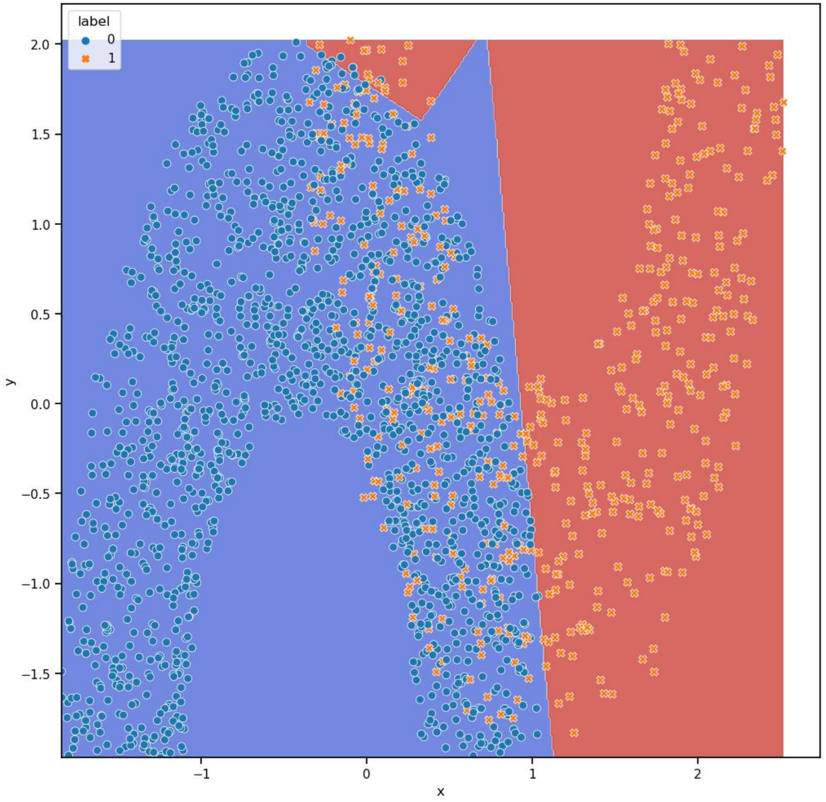
Single Neuron



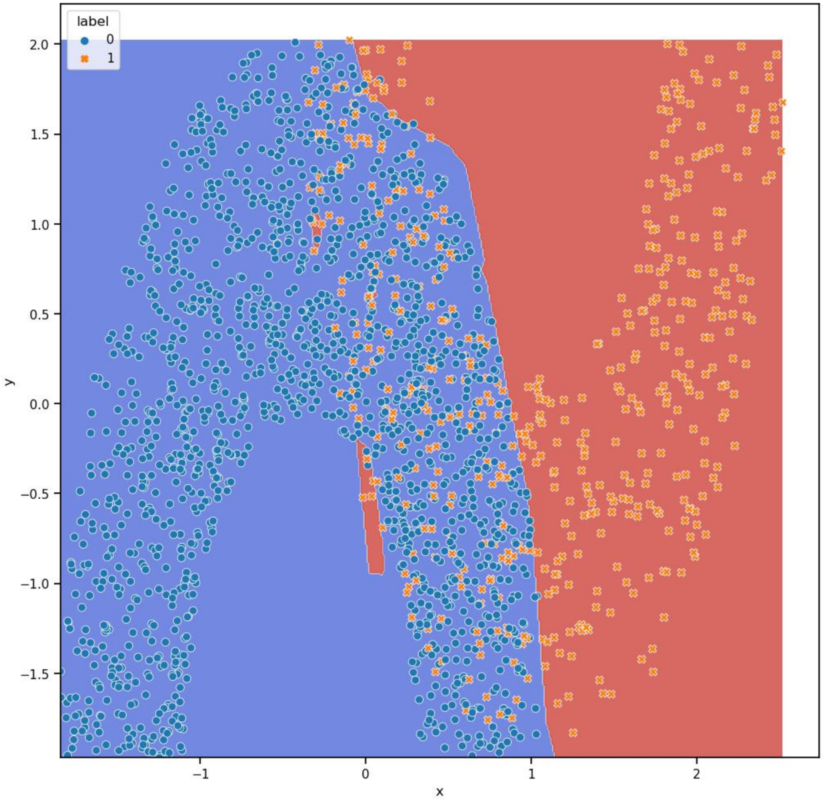
Multiple Neurons/Layers



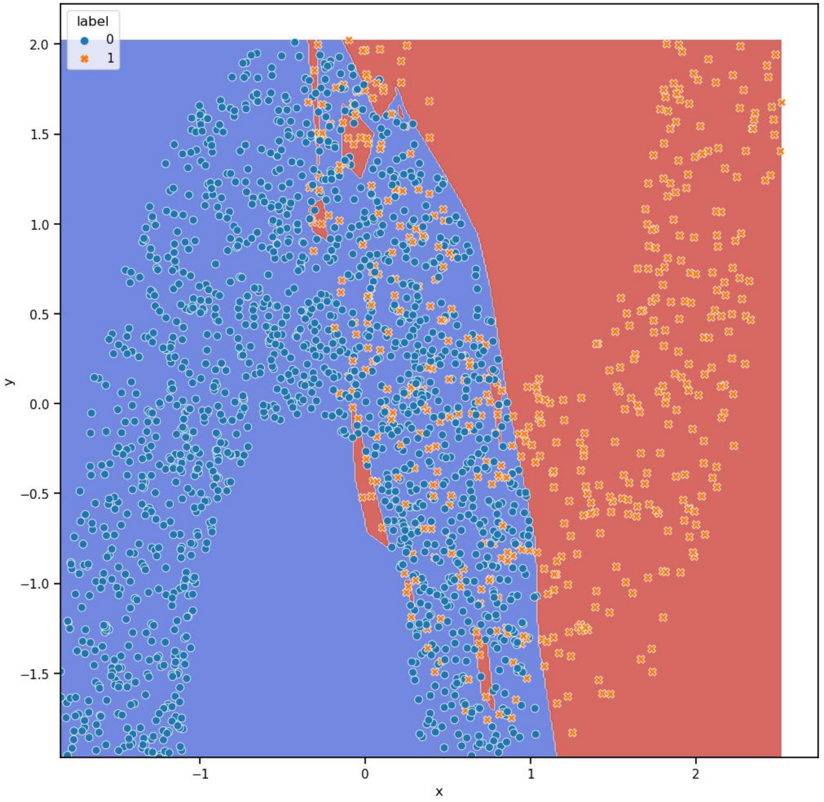
# ADDING LAYERS




Single Hidden Layer, 10 neurons



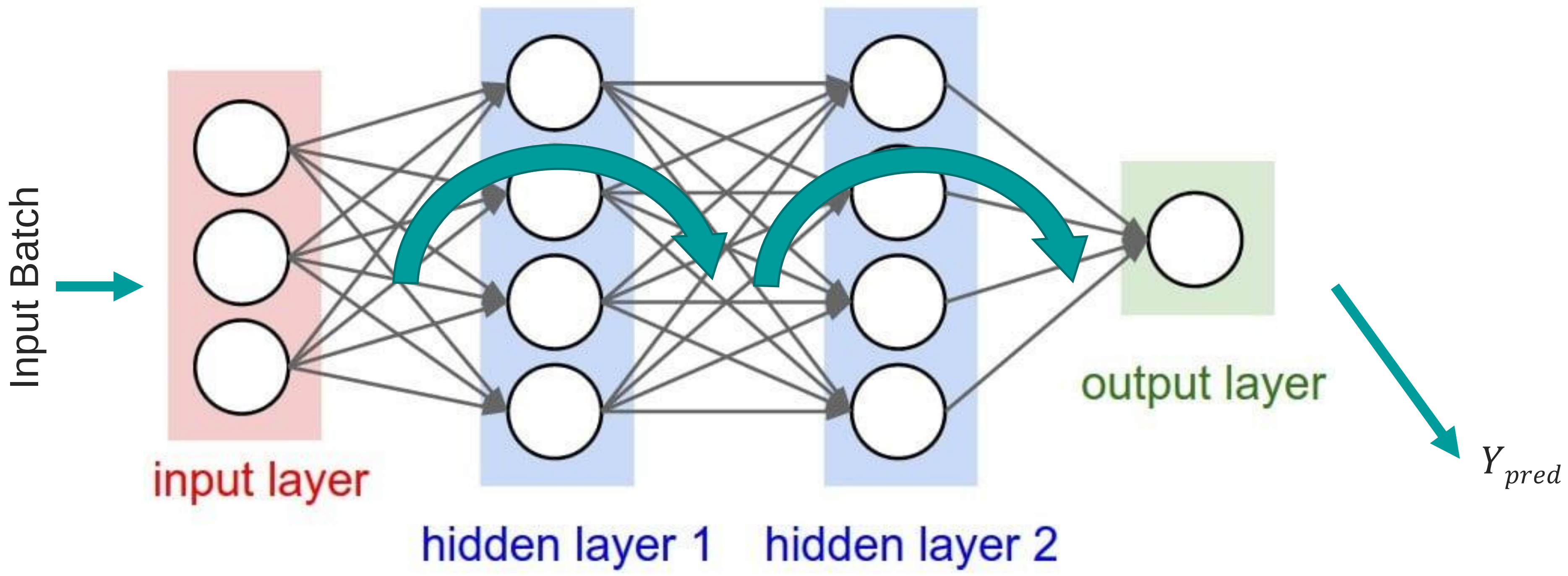

4 Hidden Layers, 10 neurons each




8 Hidden Layers, 10 neurons each

# BACKPROPAGATION

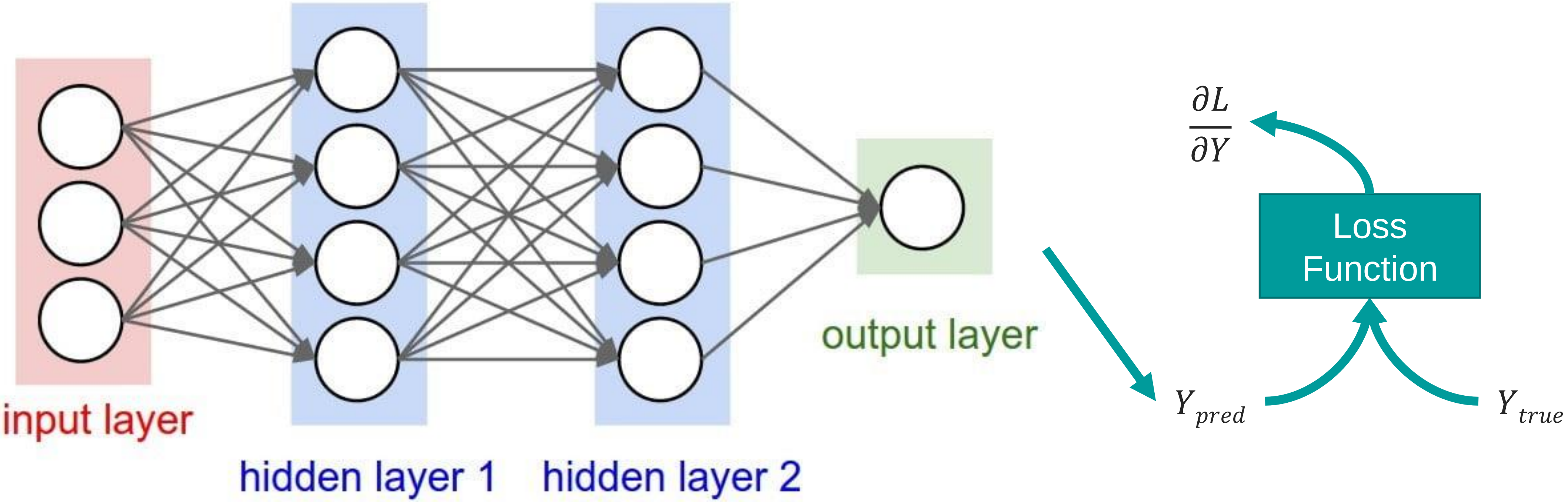
## Forward Pass



# BACKPROPAGATION

## Loss Computation

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_{true,\,i} - Y_{pred,\,i}\right)^2$$

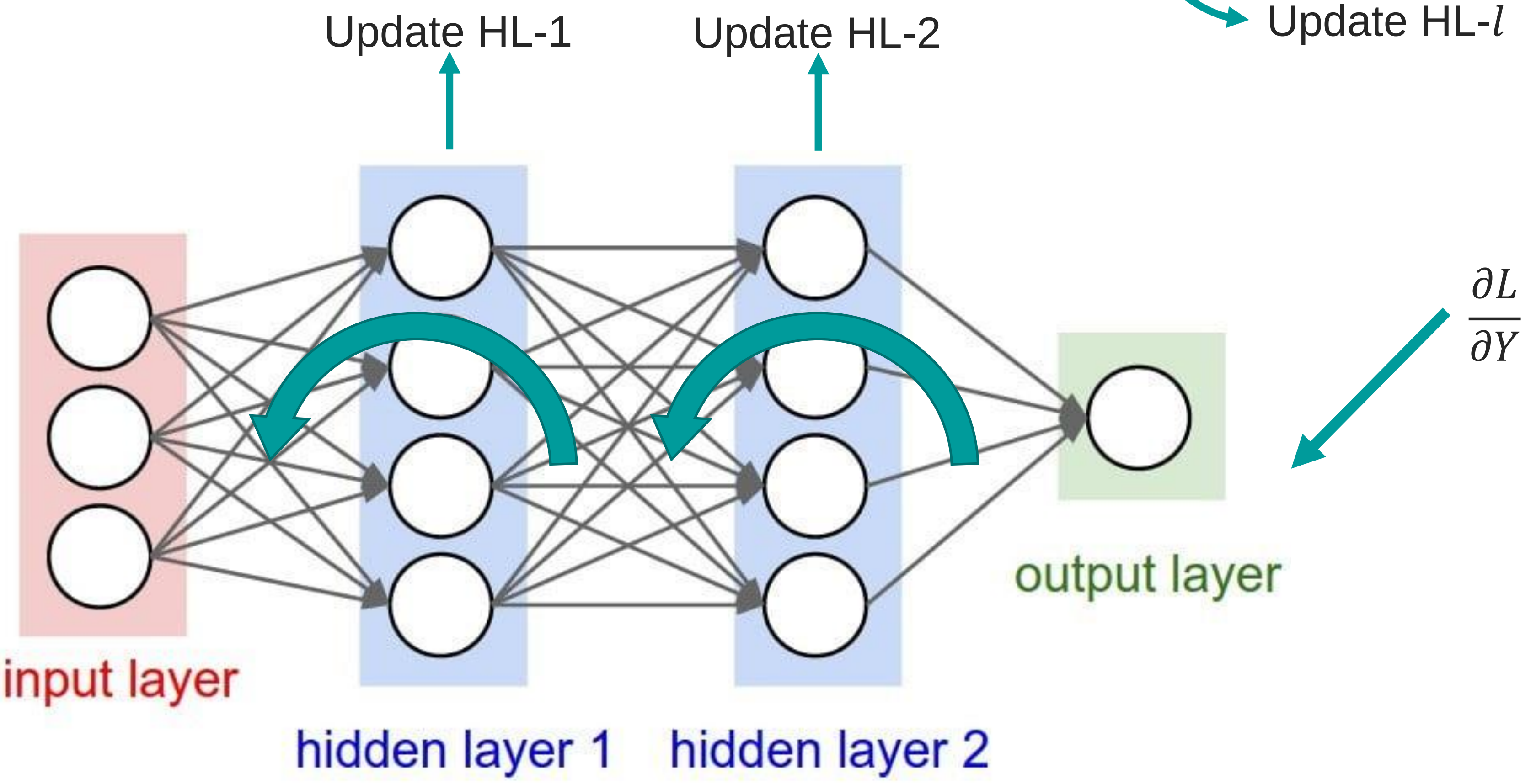


# BACKPROPAGATION

## Backward Pass

Basic SGD Optimizer

Update HL-$l$
$$\begin{cases} w_k^l = w_{k-1}^l - \eta \dfrac{\partial L}{\partial w^l} \\ b_k^l = b_{k-1}^l - \eta \dfrac{\partial L}{\partial y^l} \end{cases}$$



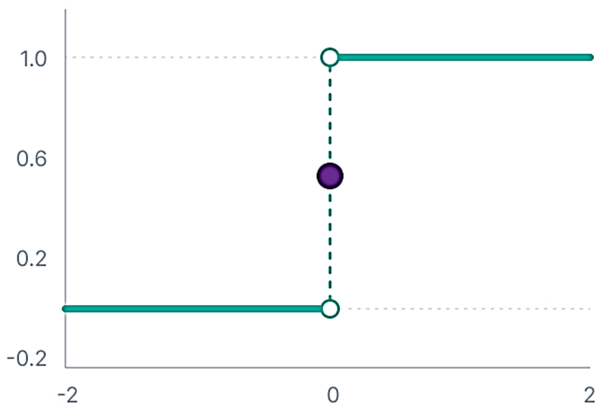
SILICON AUSTRIA LABS

# ACTIVATION FUNCTIONS

# TYPE OF ACTIVATIONS

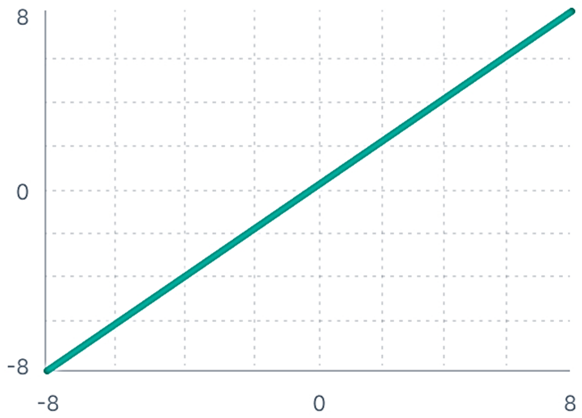
- AF makes the neuron’s decision
- Types of AF:
  - Binary Step
  - Linear
  - Sigmoid

**Binary Step Function**



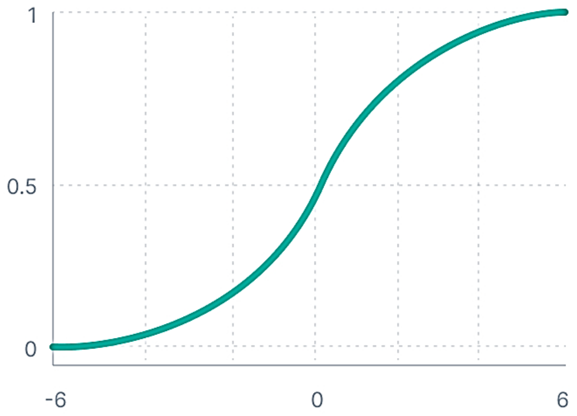

$$f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$$

**Linear Activation Function**




$$f(x) = x$$

**Sigmoid / Logistic**

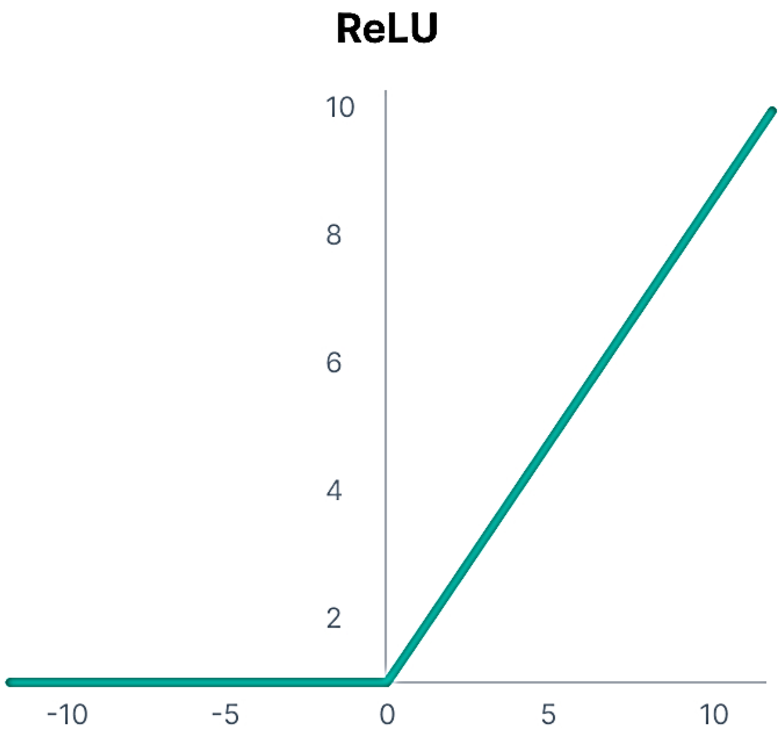



$$f(x) = \frac{1}{1 + e^{-x}}$$

# RELU

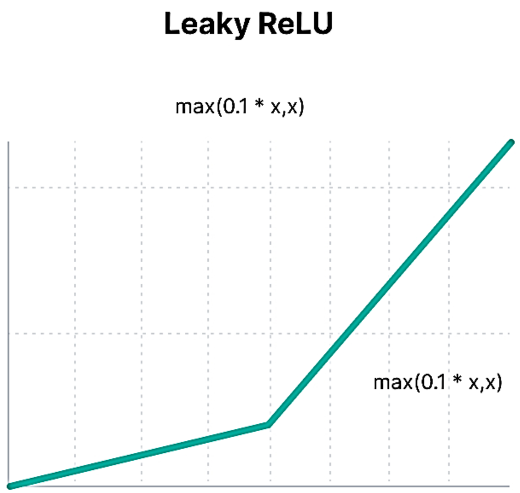
- Most popular Activation Function
- Rectified Linear Unit
- Advantages:
  - Make NN efficient as only activates some neurons
  - Accelerates Training due to its linear, non-saturating property
- Disadvantage:
  - Dying ReLU problem

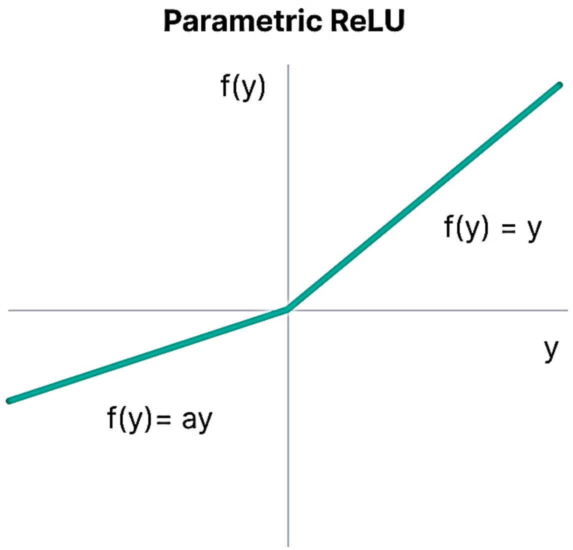
$$f(x) = \max(0, x)$$

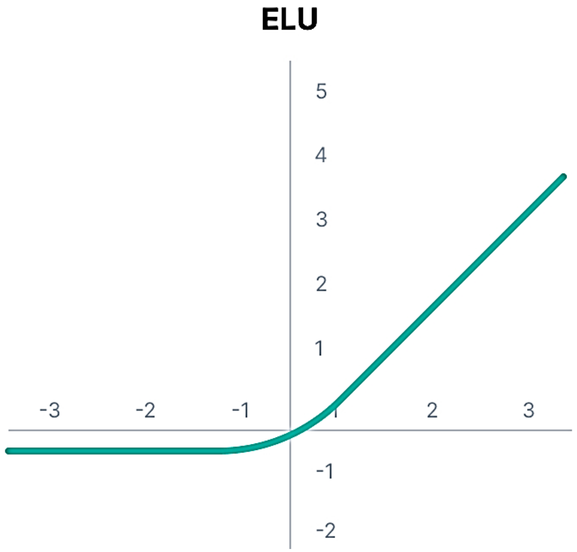


# RELU

- Variants to solve Dying ReLU while keep linear and non-saturation property



SAL
SILICON AUSTRIA LABS

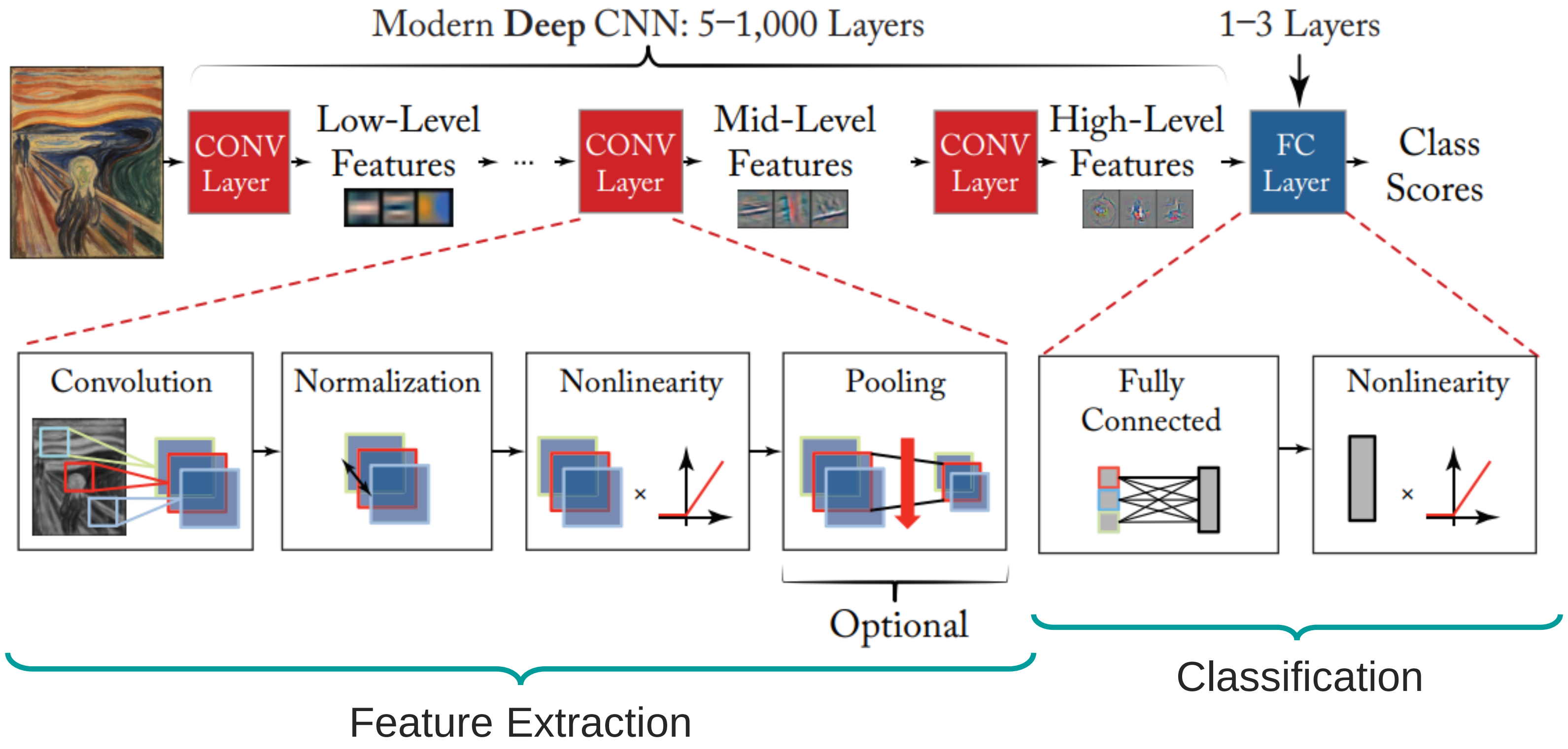
# CONVOLUTIONAL NEURAL NETWORKS

# WHY USE A CNN

- Problem too complex
- DNN requires many neurons

Many parameters and operations

- CNN Layers:
  - Conv. Filters
  - Normalization
  - Down-sample
  - Fully-Connected

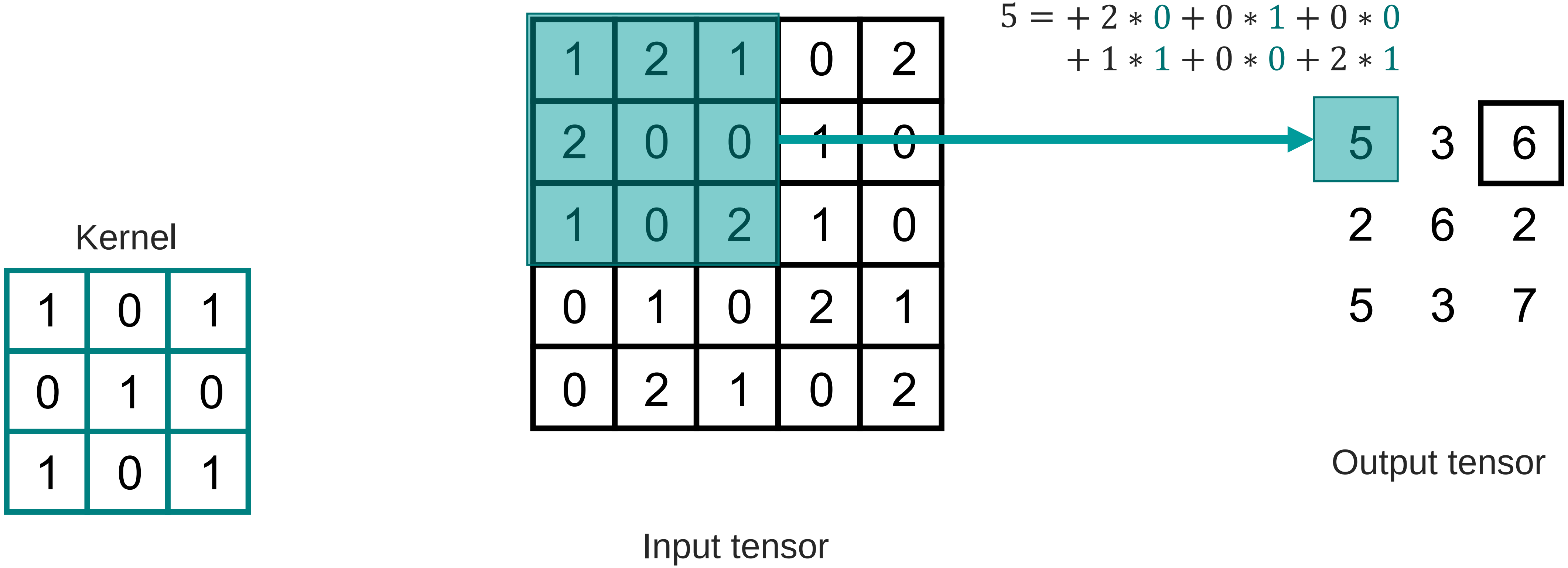


# CONV LAYER



- Input tensor $(I_h, I_w, I_{ch})$
- $N$ x Kernels $(K_h, K_w, K_{ch})$

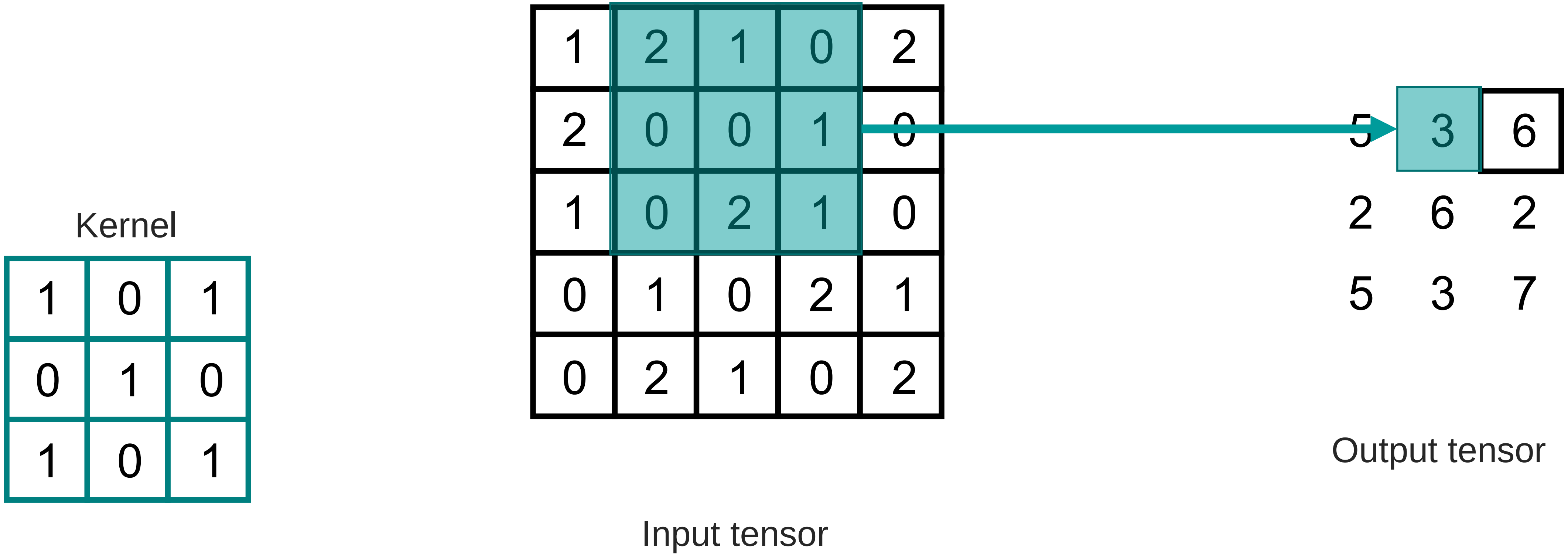
$$5 = \begin{aligned} &1*1+2*0+1*1 \\ &+2*0+0*1+0*0 \\ &+1*1+0*0+2*1 \end{aligned}$$



# CONV LAYER

SAL SILICON AUSTRIA LABS

- Input tensor $(I_h, I_w, I_{ch})$
- $N$ x Kernels $(K_h, K_w, K_{ch})$



# CONV LAYER

- Input tensor $(I_h, I_w, I_{ch})$
- $N$ x Kernels $(K_h, K_w, K_{ch})$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 1 | 0 | 2 |
| 2 | 0 | 0 | 1 | 0 |
| 1 | 0 | 2 | 1 | 0 |
| 0 | 1 | 0 | 2 | 1 |
| 0 | 2 | 1 | 0 | 2 |

Input tensor

| | | |
|---|---|---|
| 5 | 3 | 6 |
| 2 | 6 | 2 |
| 5 | 3 | 7 |

Output tensor

# CONV LAYER



- Input tensor $(I_h, I_w, I_{ch})$
- $N$ x Kernels $(K_h, K_w, K_{ch})$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 1 | 0 | 2 |
| 2 | 0 | 0 | 1 | 0 |
| 1 | 0 | 2 | 1 | 0 |
| 0 | 1 | 0 | 2 | 1 |
| 0 | 2 | 1 | 0 | 2 |

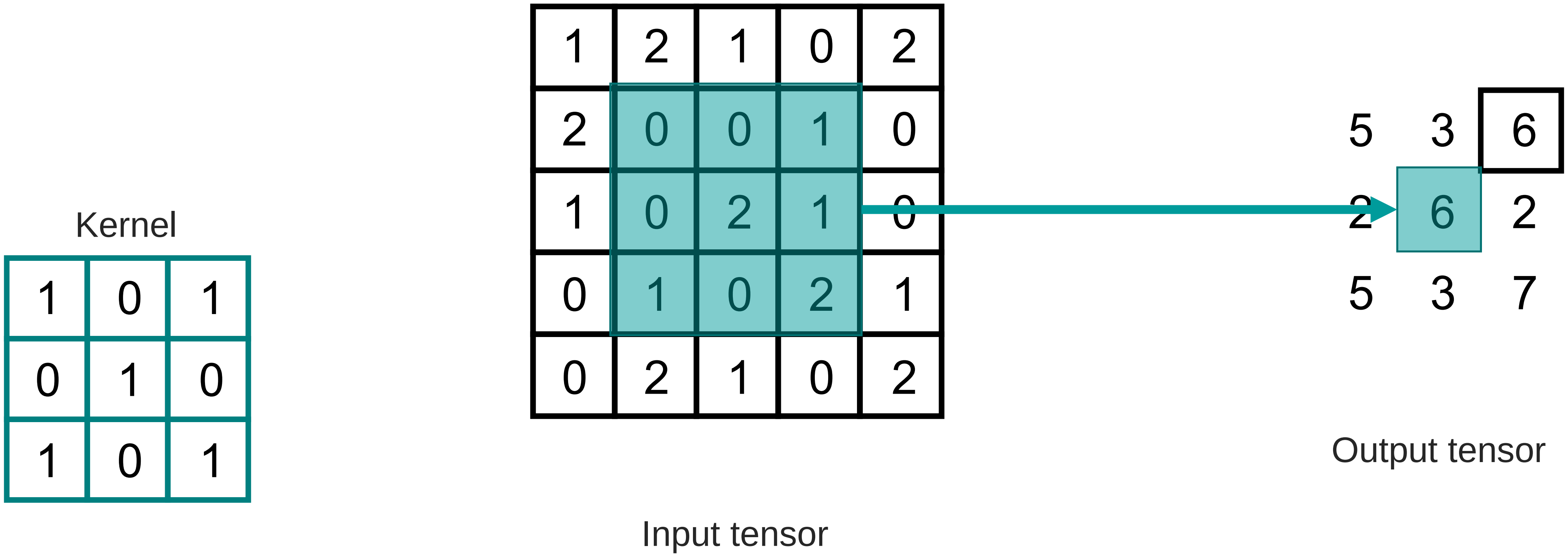
Input tensor

| | | |
|---|---|---|
| 5 | 3 | 6 |
| 2 | 6 | 2 |
| 5 | 3 | 7 |

Output tensor

# CONV LAYER

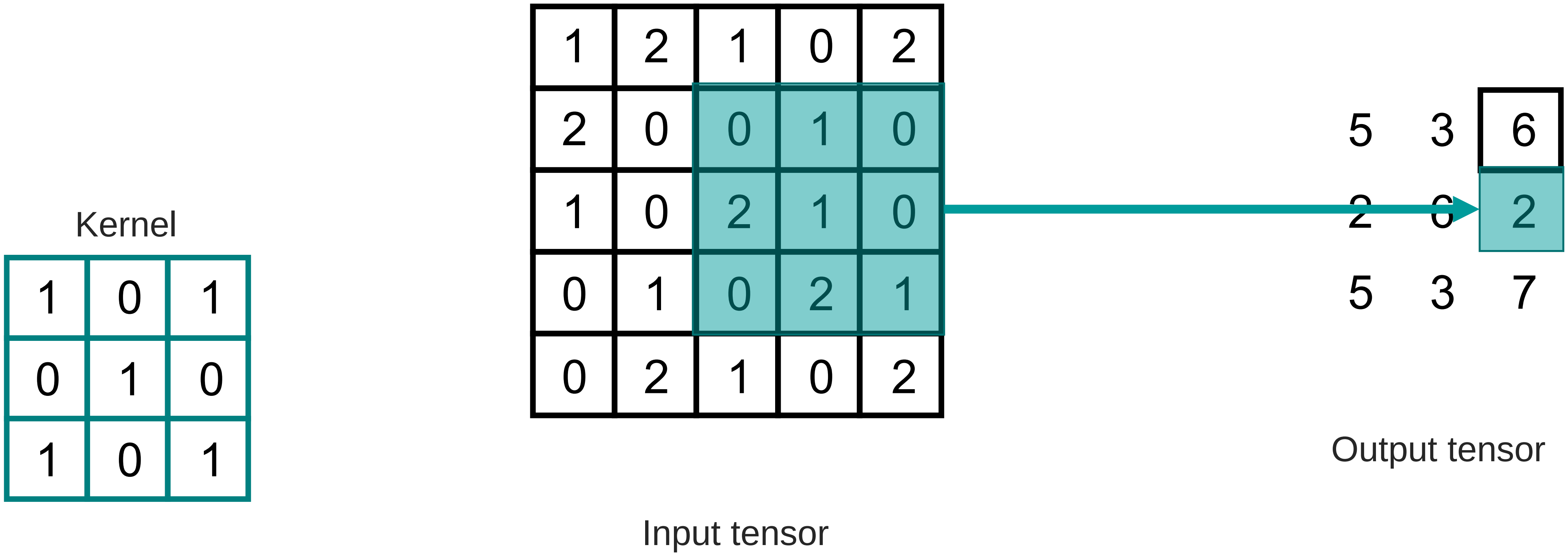
- Input tensor $(I_h, I_w, I_{ch})$
- $N$ x Kernels $(K_h, K_w, K_{ch})$



# CONV LAYER



- Input tensor $(I_h, I_w, I_{ch})$
- $N$ x Kernels $(K_h, K_w, K_{ch})$



# CONV LAYER

$$O_{h/w} = I_{h/w} - (K_{h/w} - 1)$$

- Input tensor $(I_h, I_w, I_{ch})$
- $N$ x Kernels $(K_h, K_w, K_{ch})$
- 2D Conv:
  - $K_{ch} = I_{ch}$
  - $N = O_{ch}$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 1 | 0 | 2 |
| 2 | 0 | 0 | 1 | 0 |
| 1 | 0 | 2 | 1 | 0 |
| 0 | 1 | 0 | 2 | 1 |
| 0 | 2 | 1 | 0 | 2 |

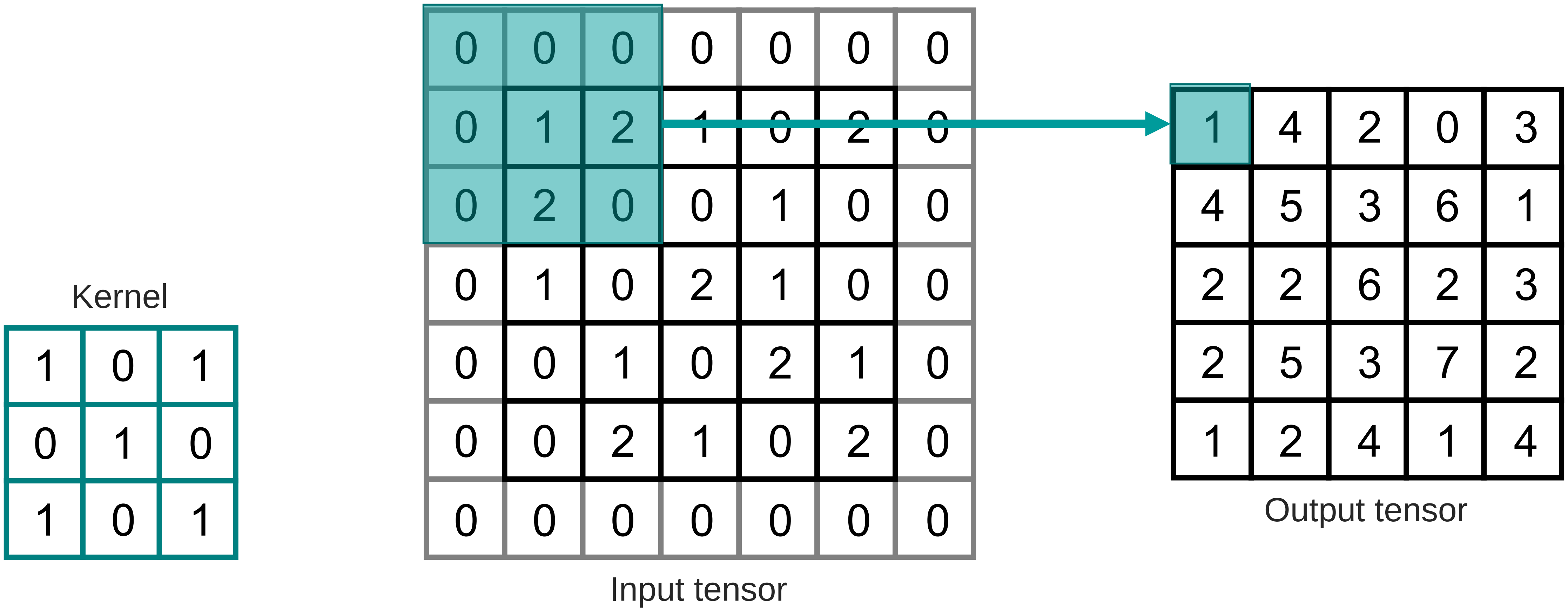
Input tensor

| | | |
|---|---|---|
| 5 | 3 | $\boxed{6}$ |
| 2 | 6 | 2 |
| 5 | 3 | 7 |

Output tensor

# CONV LAYER

- Padding $(P_h, P_w)$



# CONV LAYER

SAL SILICON AUSTRIA LABS

$$O_{h/w} = I_{h/w} - \left(K_{h/w} - 1\right) + 2 * P_{h/w}$$

- Padding $(P_h, P_w)$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 0 | 2 | 0 |
| 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Input tensor

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

Output tensor

# CONV LAYER

- Padding $(P_h, P_w)$
- Stride $(S_h, S_w)$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 0 | 2 | 0 |
| 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Input tensor

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 6 | 3 |
| 1 | 4 | 4 |

Output tensor

# CONV LAYER

- Padding $(P_h, P_w)$
- Stride $(S_h, S_w)$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 0 | 2 | 0 |
| 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Input tensor

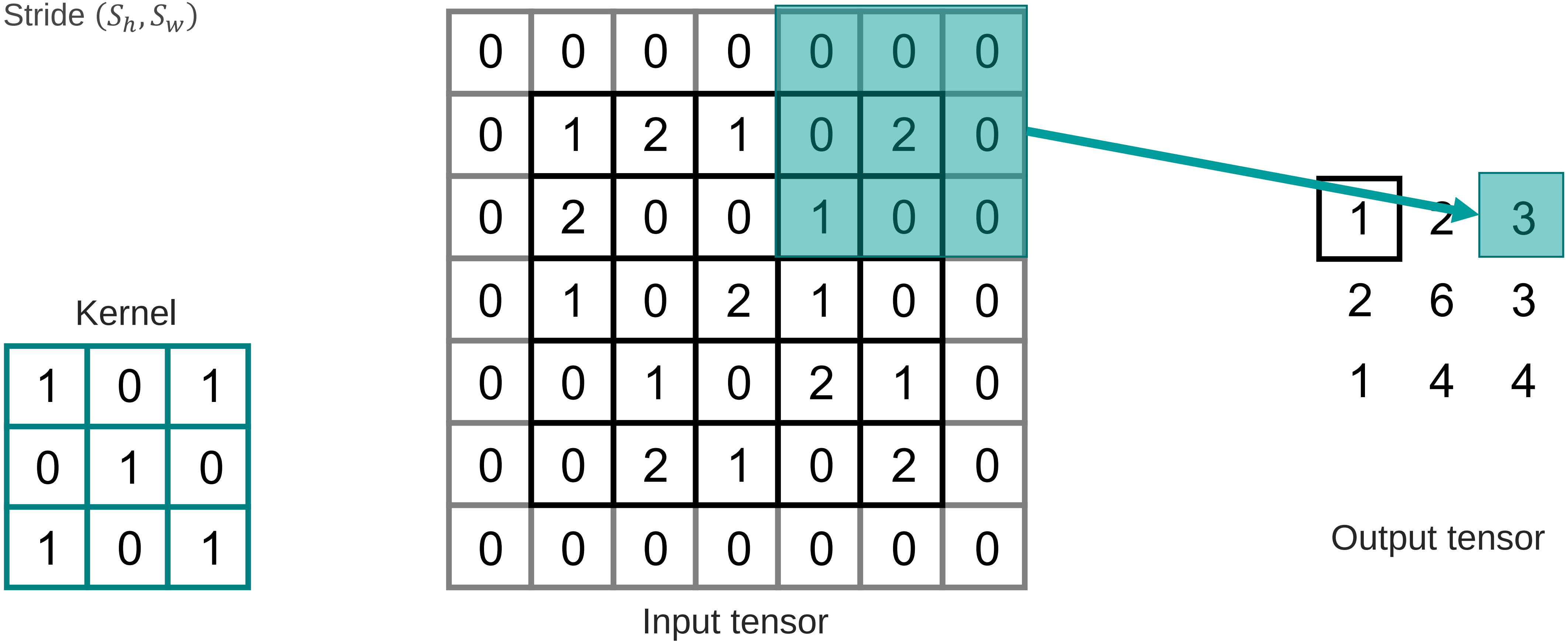
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 6 | 3 |
| 1 | 4 | 4 |

Output tensor

# CONV LAYER



- Padding $(P_h, P_w)$
- Stride $(S_h, S_w)$



# CONV LAYER

- Padding $(P_h, P_w)$
- Stride $(S_h, S_w)$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 0 | 2 | 0 |
| 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Input tensor

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 6 | 3 |
| 1 | 4 | 4 |

Output tensor

# CONV LAYER

$$O_{h/w} = \left\lfloor \frac{I_{h/w} - \left(K_{h/w} - 1\right) + 2 * P_{h/w} - 1}{S_{h/w}} + 1 \right\rfloor$$

- Padding $(P_h, P_w)$
- Stride $(S_h, S_w)$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

Input tensor

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 0 | 2 | 0 |
| 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Output tensor

| | | |
|---|---|---|
| $\boxed{1}$ | 2 | 3 |
| 2 | 6 | 3 |
| 1 | 4 | 4 |

# CONV LAYER

SAL SILICON AUSTRIA LABS

- Padding $(P_h, P_w)$
- Stride $(S_h, S_w)$
- Dilation $(D_h, D_w)$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 0 | 2 | 0 |
| 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

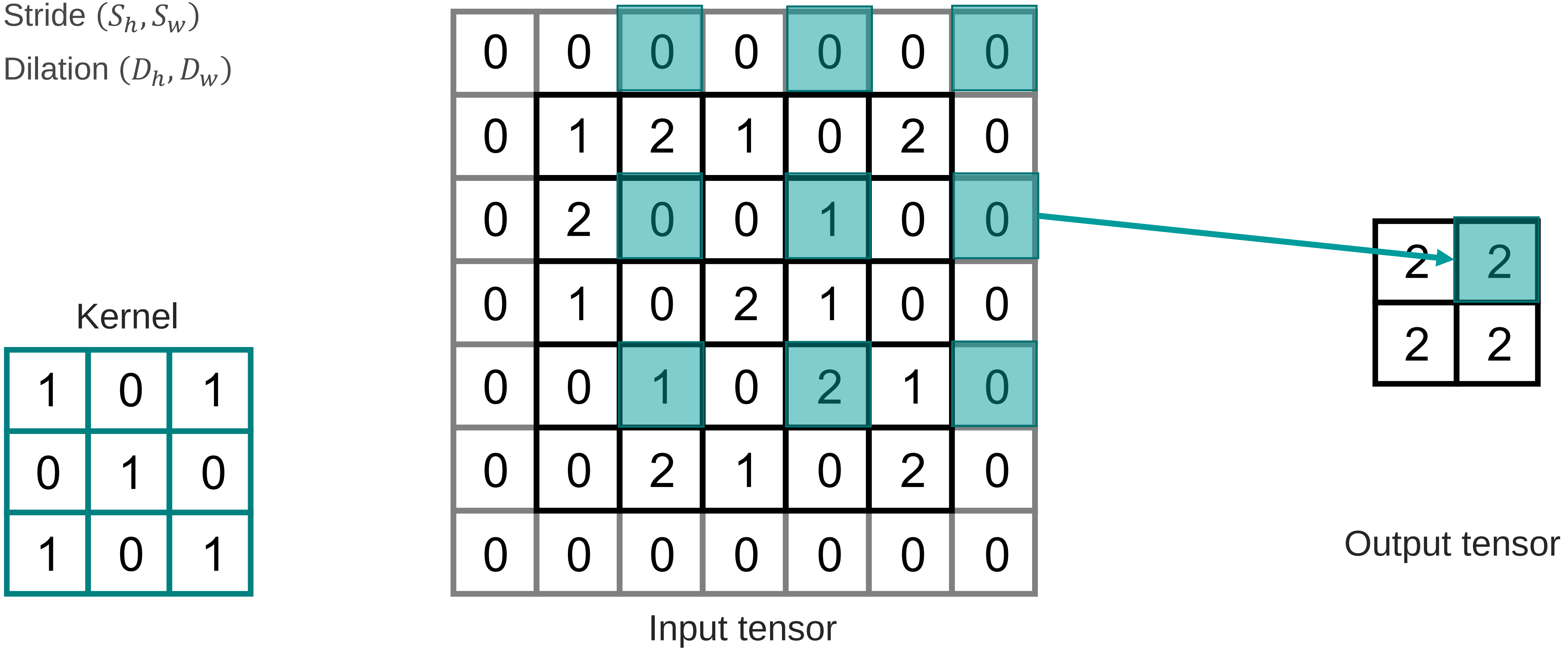
Input tensor

| | |
|---|---|
| 2 | 2 |
| 2 | 2 |

Output tensor

# CONV LAYER

- Padding $(P_h, P_w)$
- Stride $(S_h, S_w)$
- Dilation $(D_h, D_w)$



# CONV LAYER

$$O_{h/w} = \left\lfloor \frac{I_{h/w} - D_{h/w} * \left(K_{h/w} - 1\right) + 2 * P_{h/w} - 1}{S_{h/w}} + 1 \right\rfloor$$

- Padding $(P_h, P_w)$
- Stride $(S_h, S_w)$
- Dilation $(D_h, D_w)$

Kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 0 | 2 | 0 |
| 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

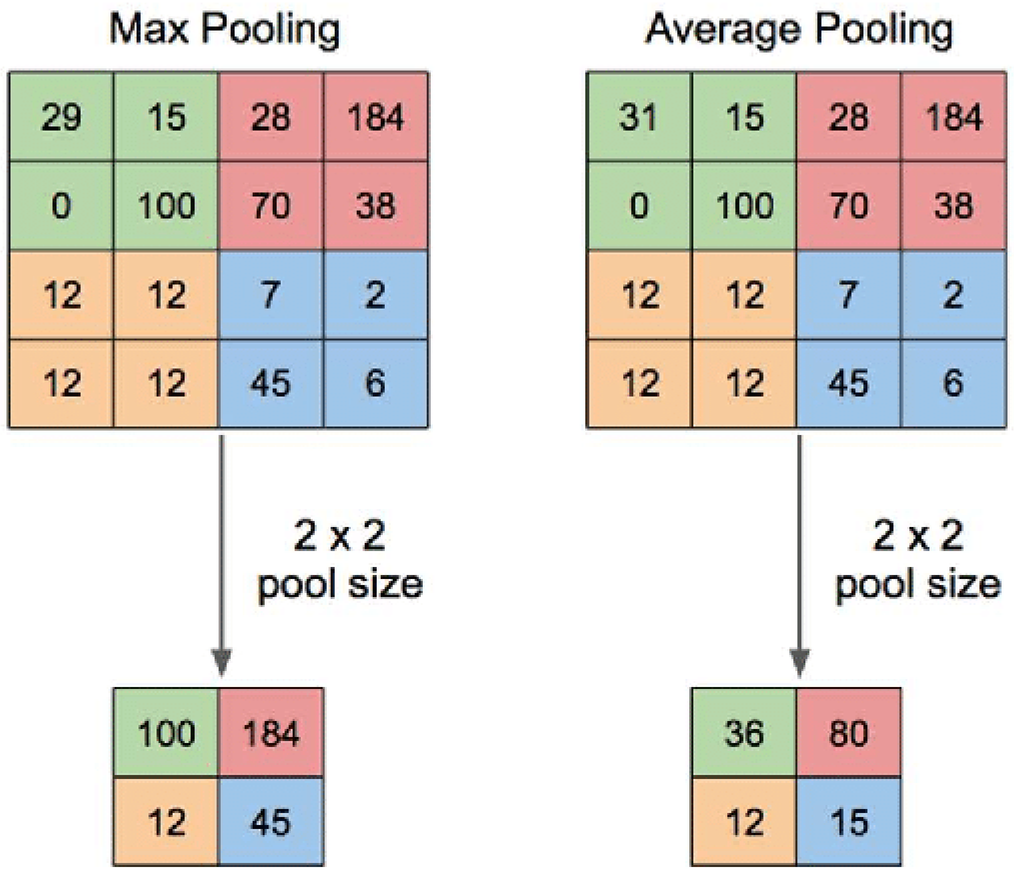
Input tensor

| | |
|---|---|
| 2 | 2 |
| 2 | 2 |

Output tensor

# POOL LAYERS

- Used for down-sampling
- Generally used with stride as kernel size: $S_{h/w} = K_{h/w}$
- Common Types:
  - MaxPool
  - AvgPool



# BATCH NORM

- Range of layer's output differs from sample to sample
  - Original input data has samples with different distribution/range
  - Weights update after every batch
- Scales each layer's output features
  - Running feature mean: $E(x)$
  - Running feature variance: $Var(x)$
  - Trainable feature weight: $\gamma$
  - Trainable feature bias: $\beta$
- Stabilizes and speeds-up training

$$y = \frac{x - E(x)}{\sqrt{Var(x) + \varepsilon}} * \gamma + \beta$$

# BATCH NORM FUSION

SAL SILICON AUSTRIA LABS

Conv/FC $y = b + \sum w \times x$

BatchNorm $z = \frac{y - \mu}{\sqrt{\sigma^2 + \varepsilon}} * \gamma + \beta$

$$z = \frac{(b + \sum w \times x) - \mu}{\sqrt{\sigma^2 + \varepsilon}} * \gamma + \beta$$

$$z = \underbrace{\left[\gamma \times \frac{b - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta\right]}_{\hat{b}} + \sum \underbrace{\left[\frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}} \times w\right]}_{\hat{w}} \times x$$

Constant after training

$E(y) \longrightarrow \mu$

$Var(y) \longrightarrow \sigma^2$

$$\boxed{z = \hat{b} + \sum \hat{w} \times x}$$

# QUESTIONS ?

SAL
SILICON AUSTRIA LABS

# UNFOLD THE FUTURE