EE-508: Hardware Foundations for Machine Learning

Deeper DIVE into CDNNs

University of Southern California

Ming Hsieh Department of Electrical and Computer Engineering

Instructor:
Arash Saifhashemi

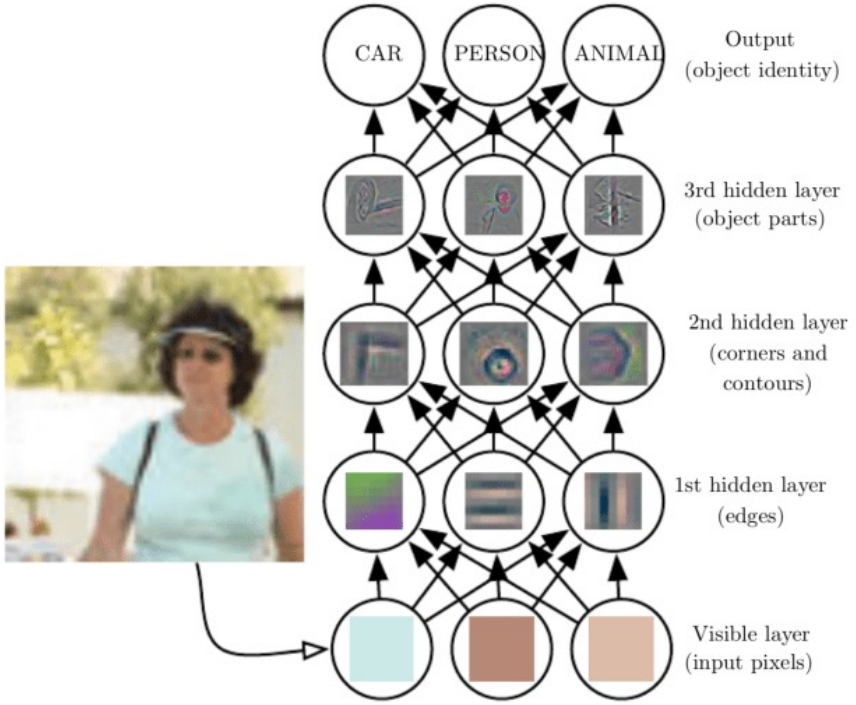
Deeper DIVE into CDNNs

A type of deep learning architecture commonly used in computer vision tasks such as image classification, object detection, and image segmentation.

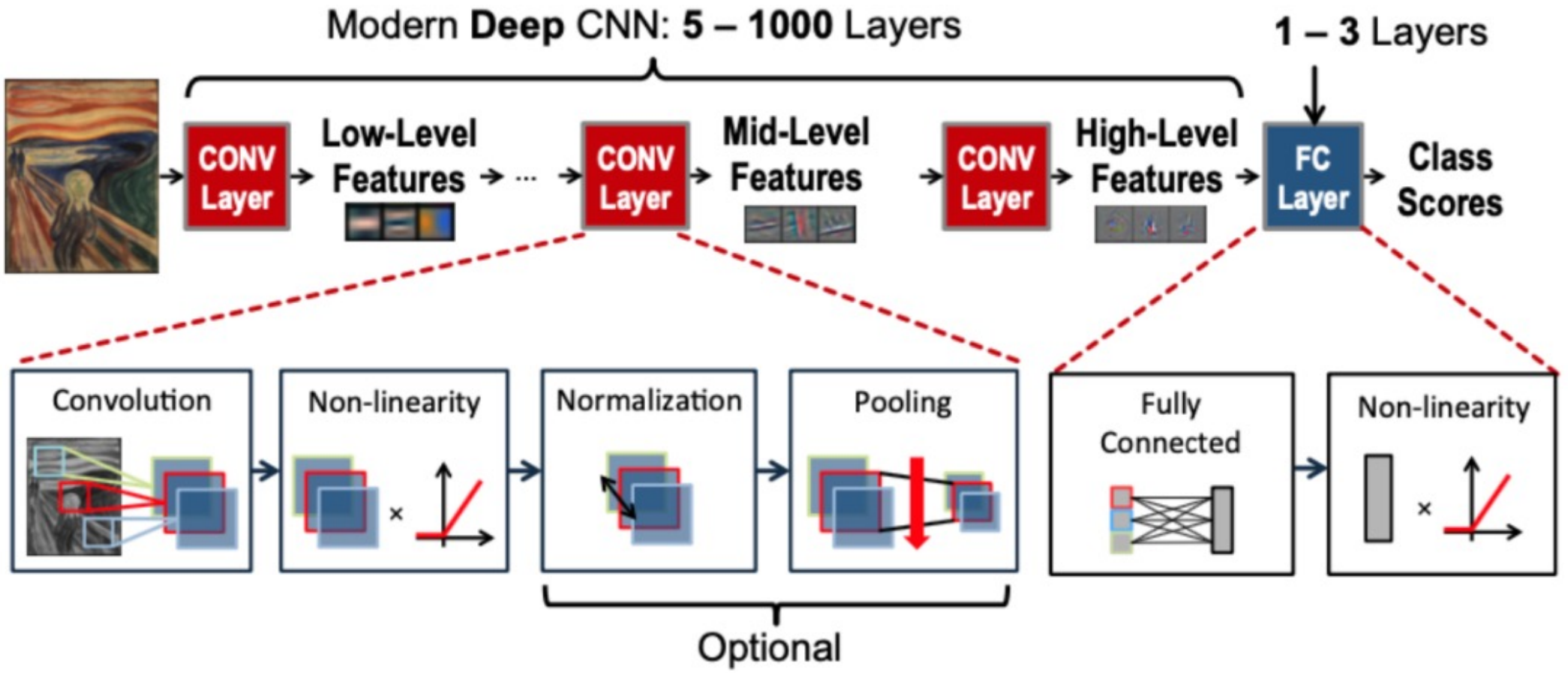
CNNs excel at extracting spatial features from images through the use of convolutional filters.

Convolutional Neural Network

- First Layer:
  - Identify **edges**, by comparing the brightness of neighboring pixels
- Second layer:
  - Given the edges, find **corners and contours**
- Third layer:
  - Given a collection of corner and contours, find **object parts**
- Last layer:
  - Given the object parts, **label** the object



source: http://www.deeplearningbook.org



Deep Convolutional Neural Networks

Source: eyeriss.mit.edu

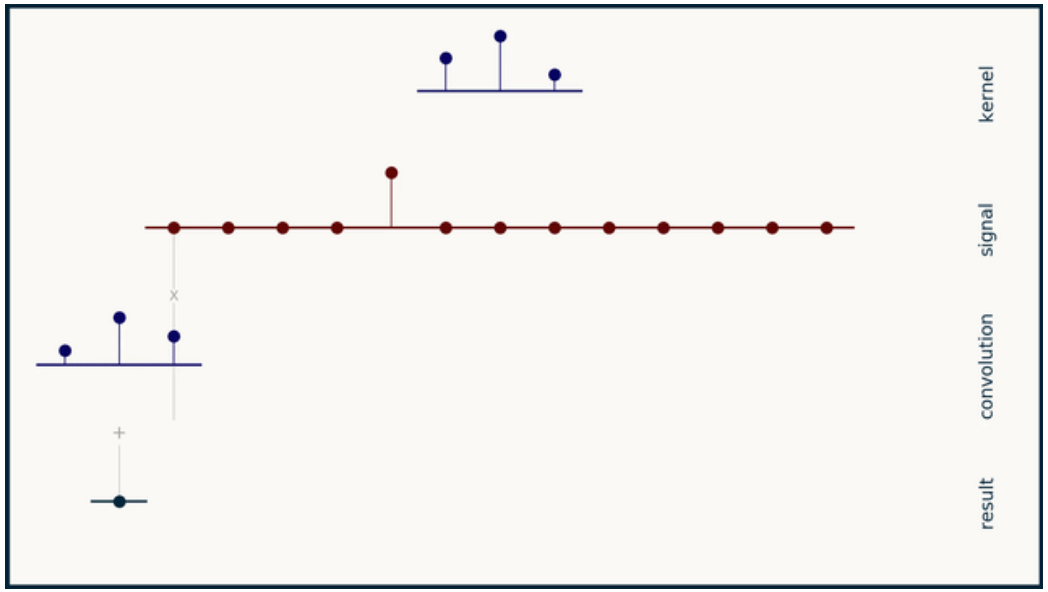
Convolution

- Definition:
  - A mathematical operation
    - flipping one function (or sequence) and sliding it over another, calculating the area under their product for each position
- Purpose:
  - Filtering: Smooth, remove noise, adjust frequencies.
  - System Response: Determine response of linear, time-invariant systems.

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau.$$

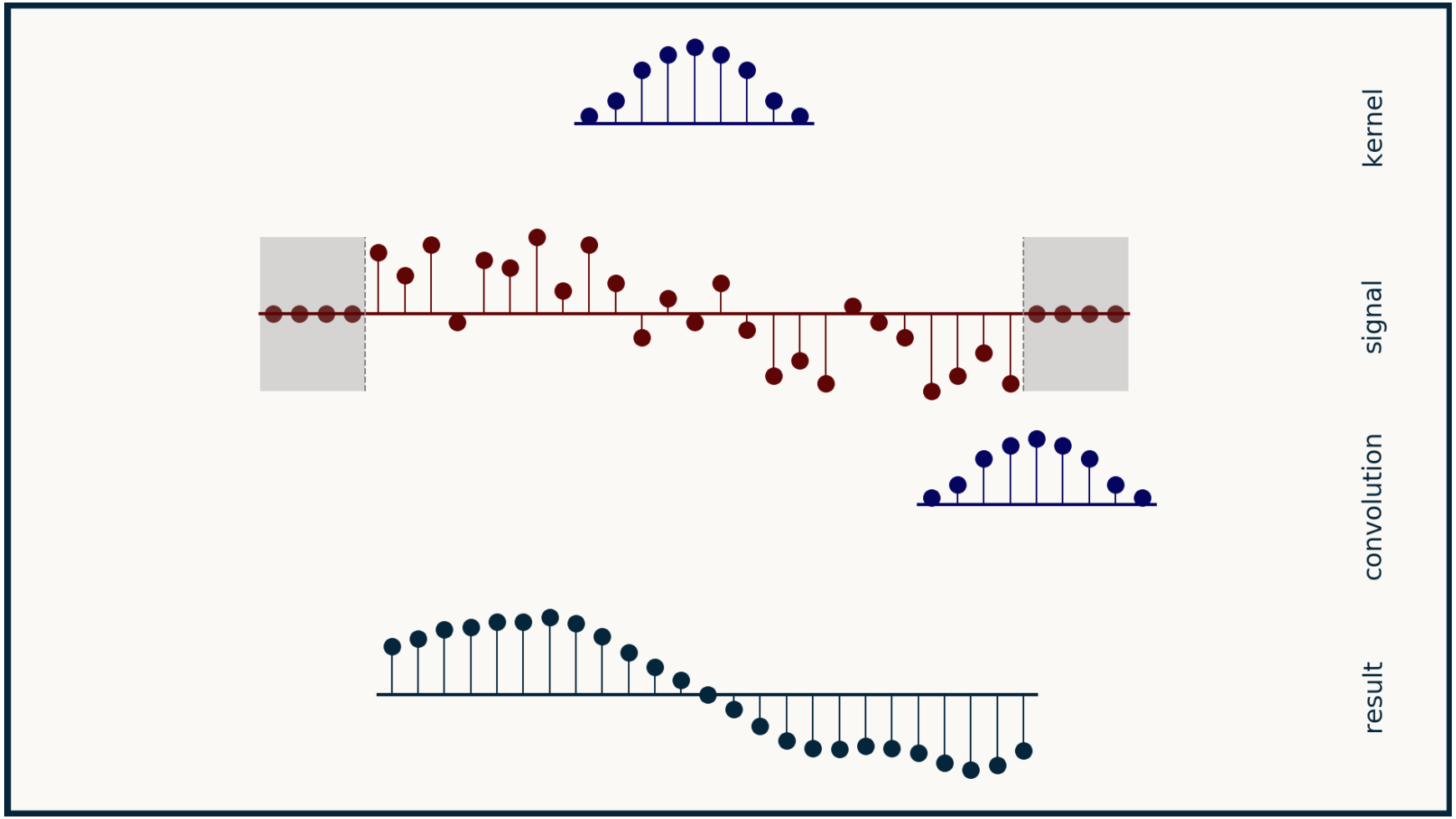
An equivalent definition is (see commutativity):

$$(f * g)(t) := \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau.$$



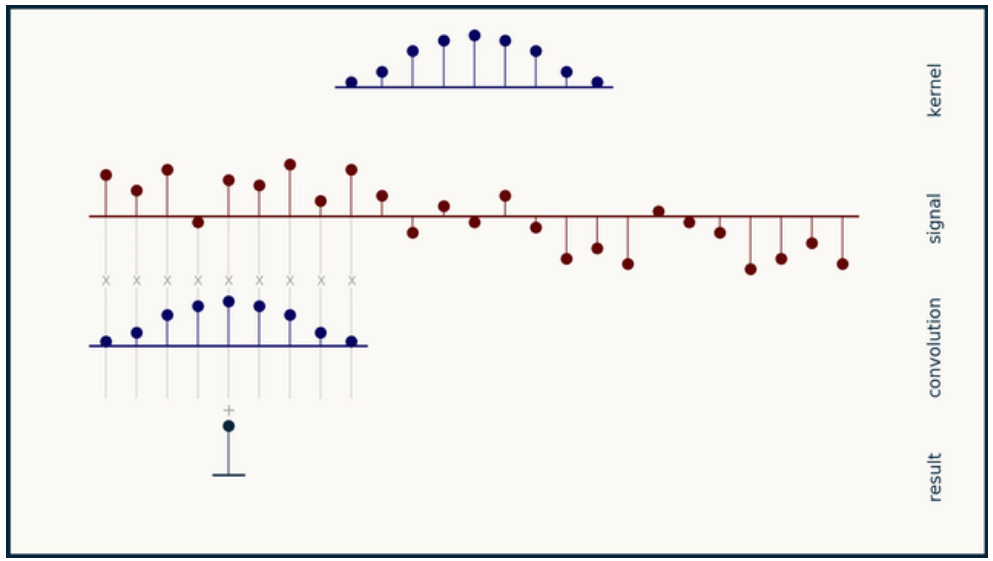
1D Convolution

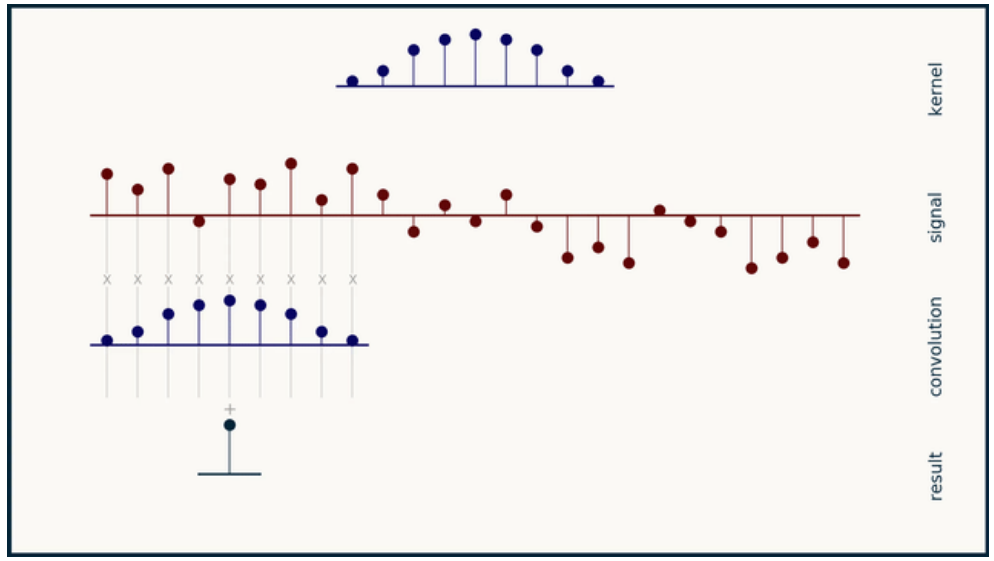
Source: e2eml.school

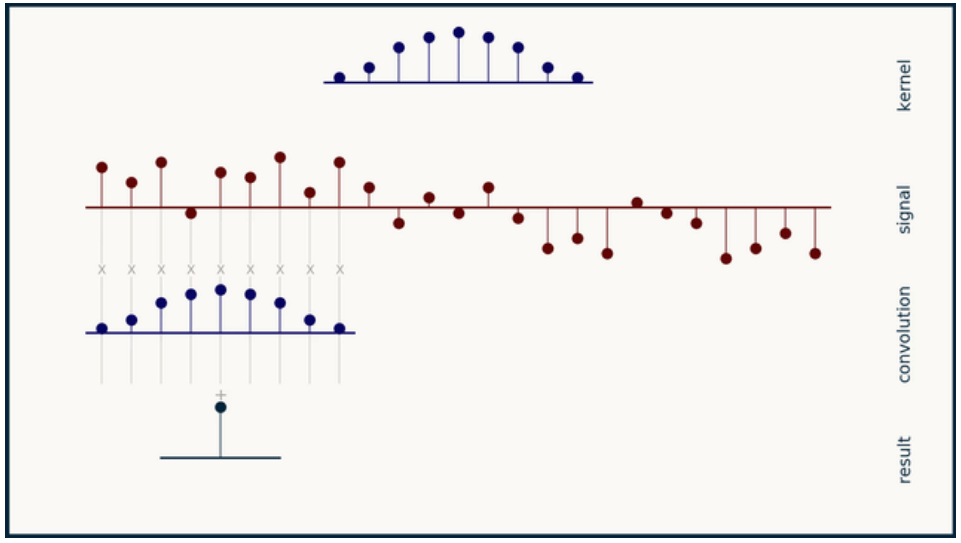


Padding in 1D Convolution

Source: e2eml.school

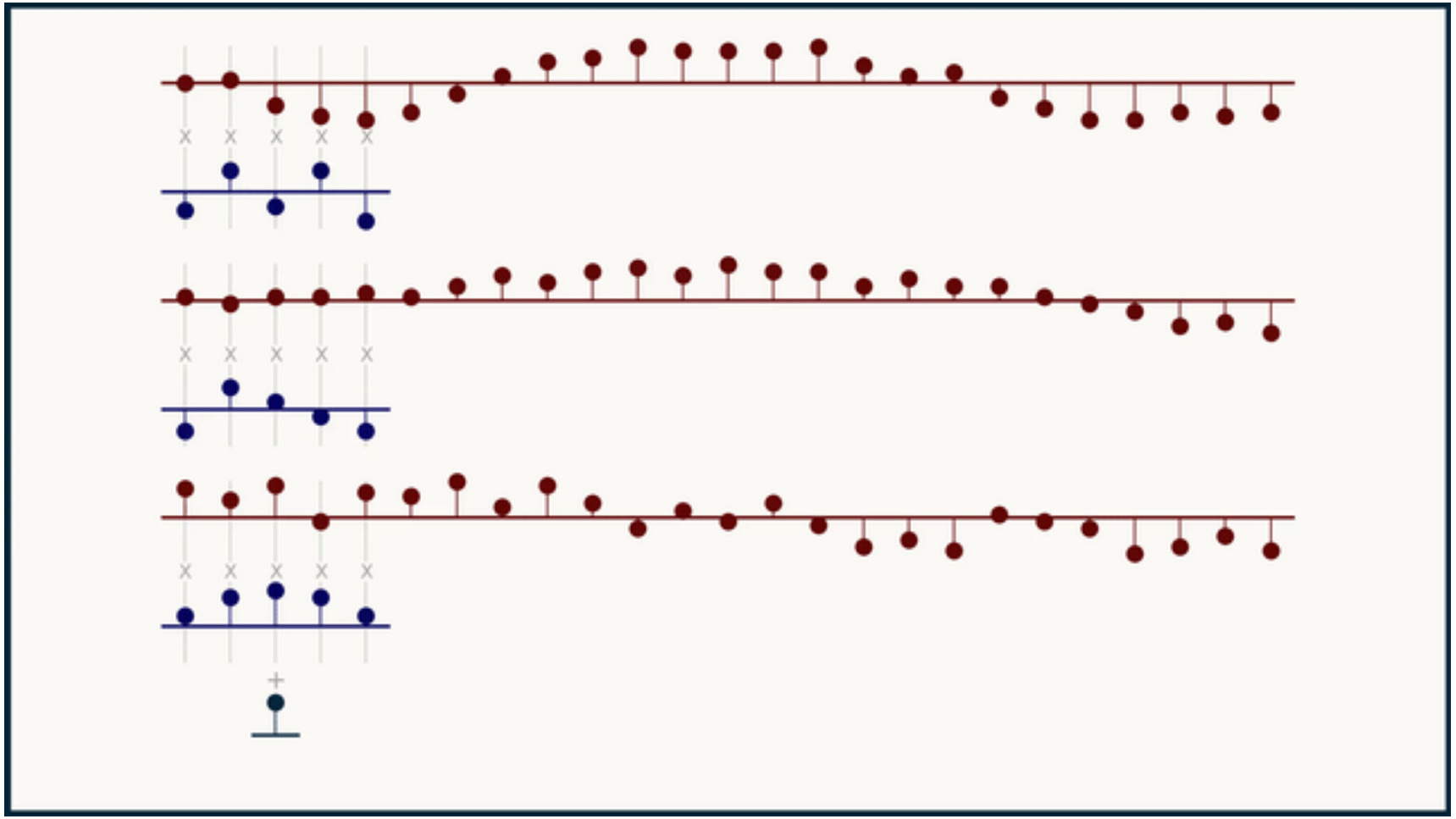






Stride in 1D Convolution

Source: e2eml.school



Multiple Channels in 1D Convolution
Each kernels is convolved with a channel separately. The output at each point is the summation of all results

Source: e2eml.school

Tensors

$$1$$

Scalar (0D)

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

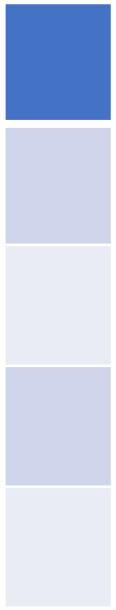
Vector (1D)

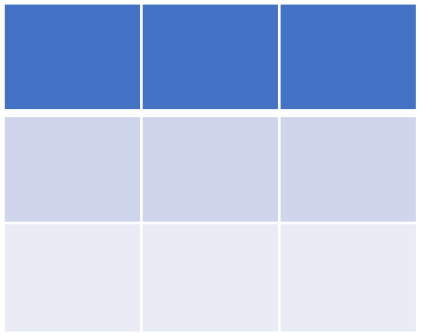
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

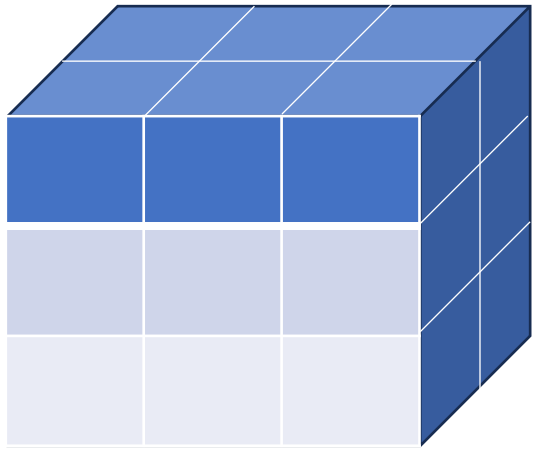
Matrix (2D)

$$\begin{bmatrix} [1 & 2] & [3 & 4] \\ [5 & 6] & [7 & 8] \end{bmatrix}$$

Tensor (Higher than 2 dimension)

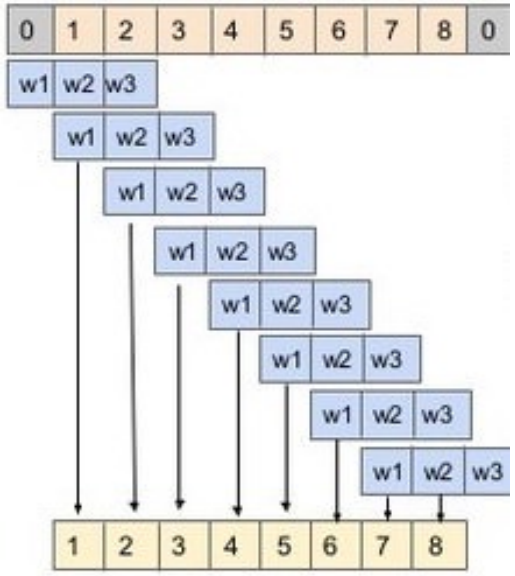


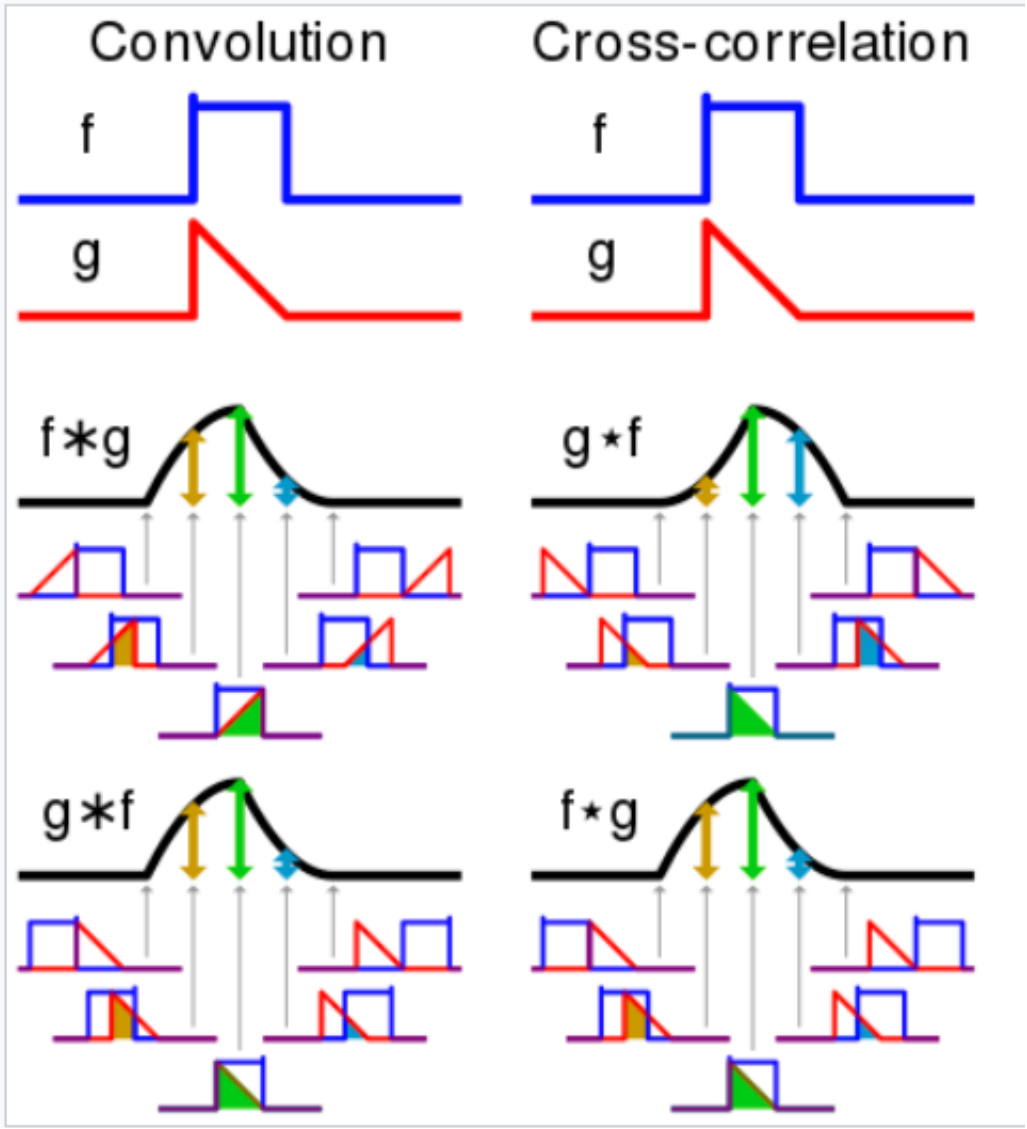




Convolution in Deep Learning

- Note:
  - In deep learning, the operation is often a simplified version of the traditional mathematical convolution, more similar to cross-correlation.

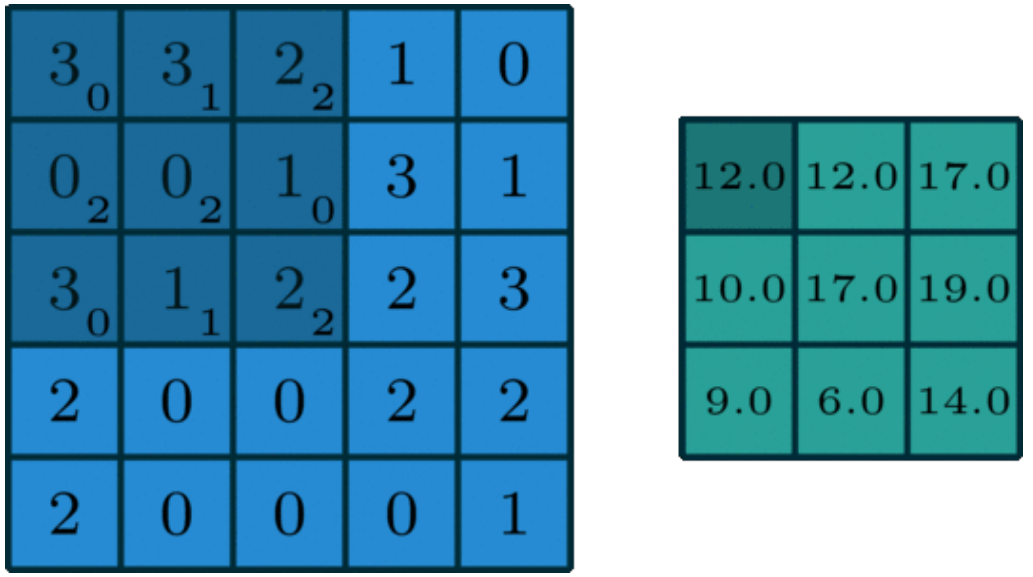




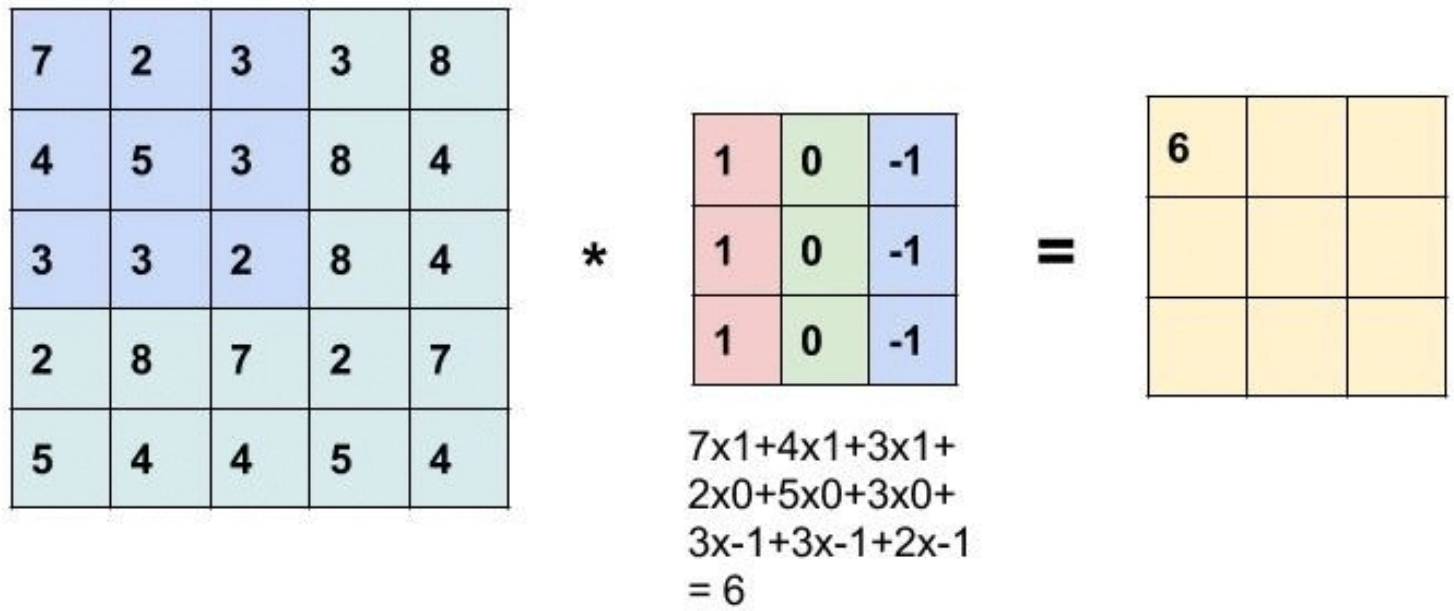
Source: Wikipedia

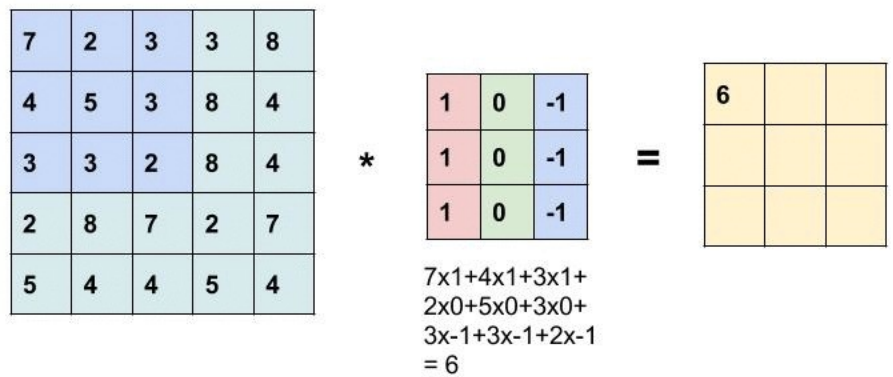
Convolution in Deep Learning

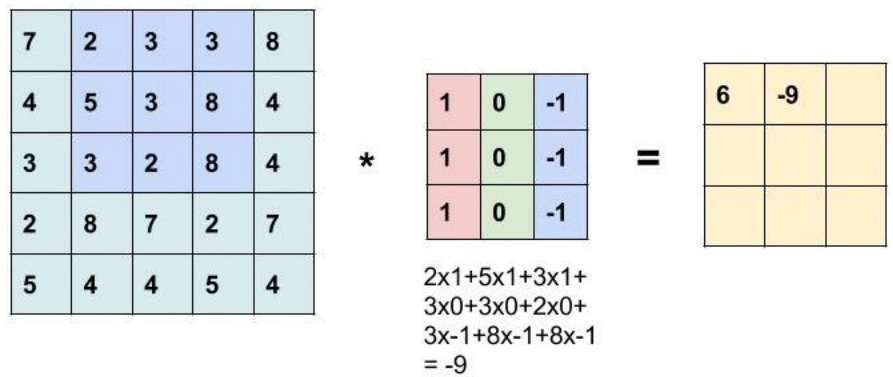
- Definition:
  - A mathematical operation where a **filter** (or **kernel**) slides over the input data (like an image) to produce a feature map.
- Purpose:
  - Extracts local features from input.
  - Captures spatial hierarchies in data.
  - Reduces computational complexity compared to fully connected layers.

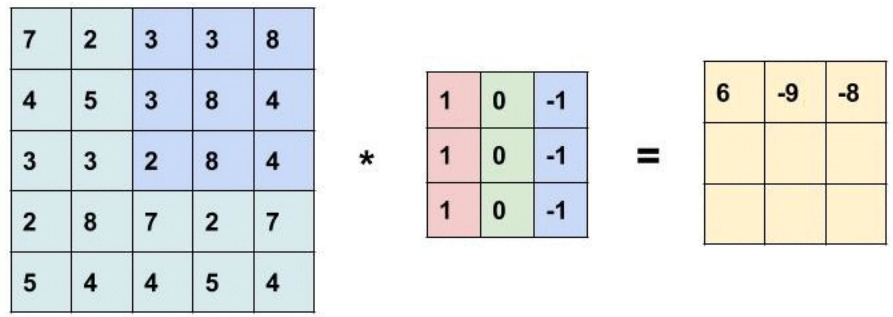


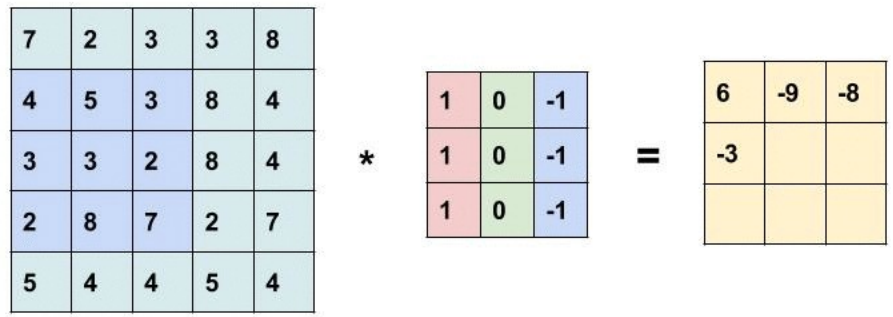
flipping one function (or sequence) and sliding it over another, calculating the area under their product for each position

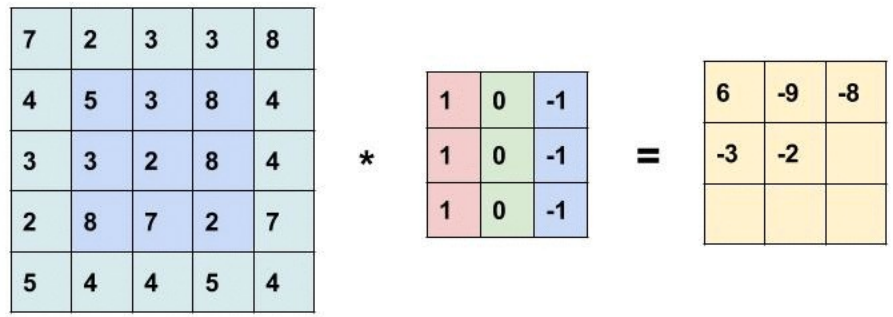

7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6
7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

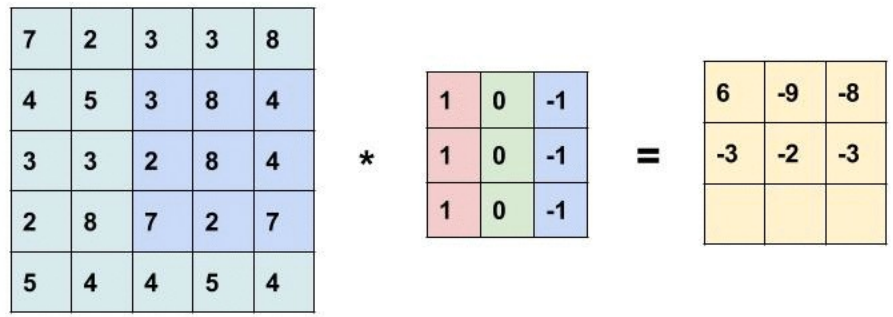

7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6
7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

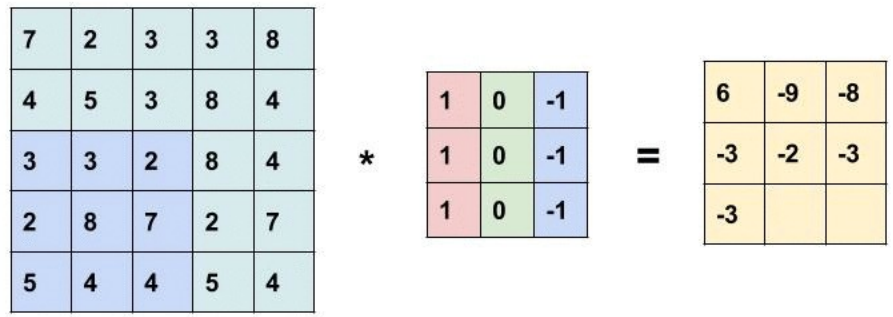

7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
2x1+5x1+3x1+
3x0+3x0+2x0+
3x-1+8x-1+8x-1
= -9
=
6 -9

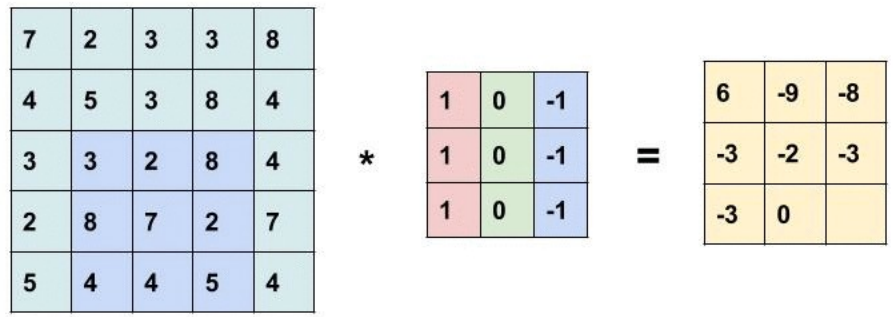

7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6 -9 -8

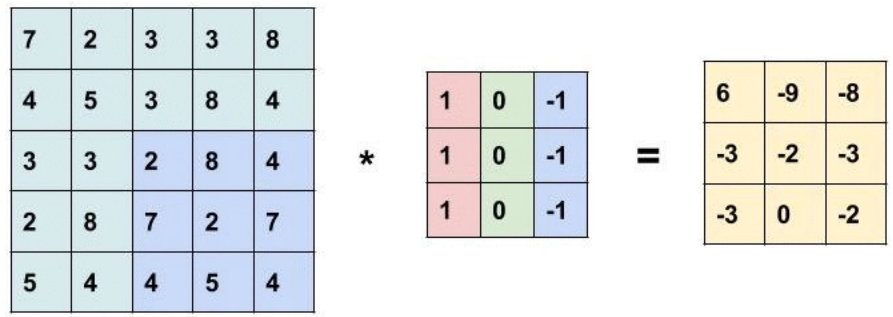

7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6 -9 -8
-3

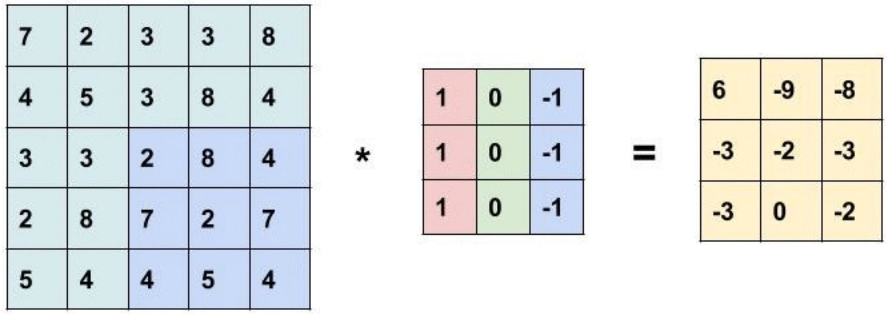

7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6 -9 -8
-3 -2


7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6 -9 -8
-3 -2 -3


7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6 -9 -8
-3 -2 -3
-3

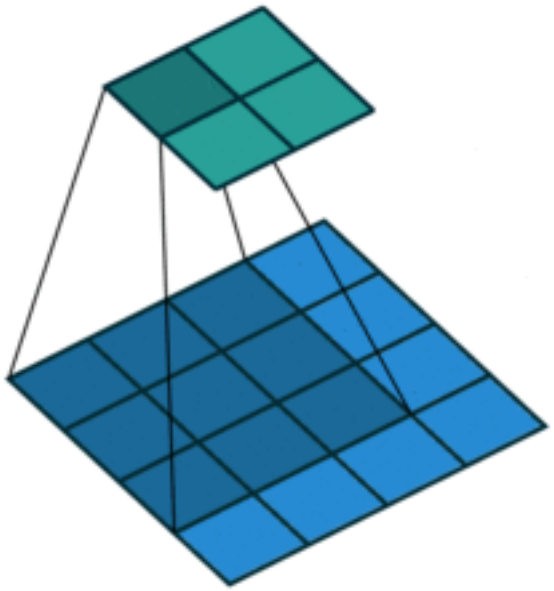

7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6 -9 -8
-3 -2 -3
-3 0


7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6 -9 -8
-3 -2 -3
-3 0 -2

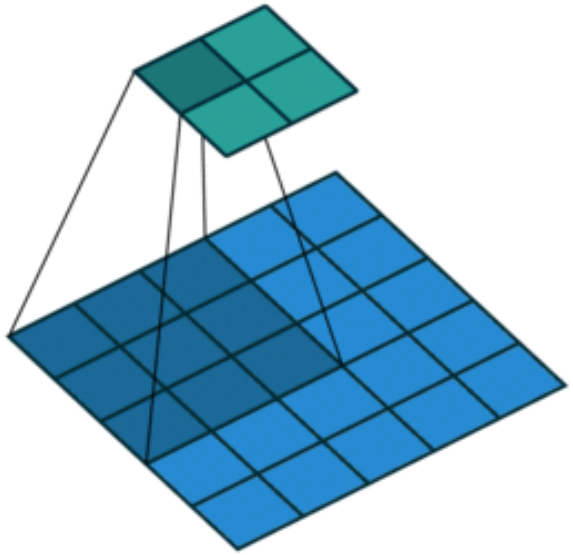


$$Output\ Feature\ Map\ = \frac{(Input\ Feature\ Map - \ Filter\ + \ Stride)}{Stride}$$

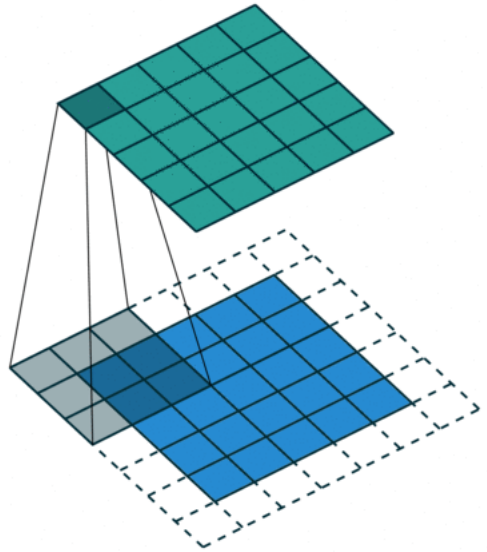
$$\frac{(5 - 3\ + 1)}{1} = 3$$



No padding, no strides

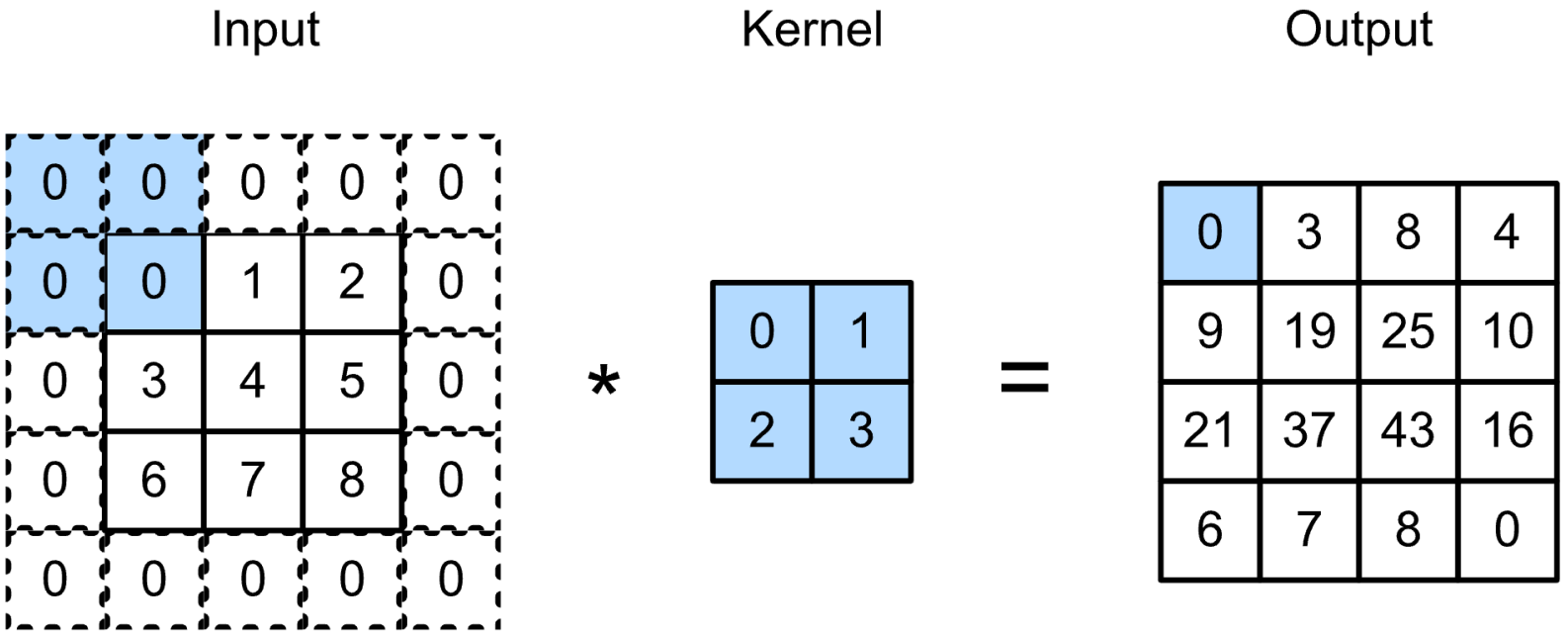


No padding, strides



Half padding, no strides

Source: github.com/vdumoulin

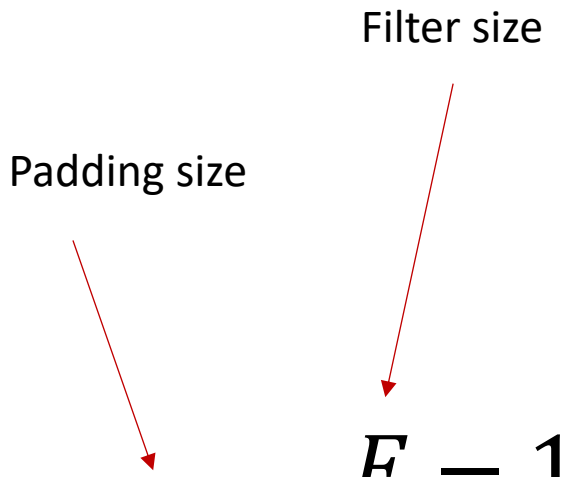


$$Output\ Feature\ Map\ = \frac{(Input\ Feature\ Map - \ Filter\ + \ Stride + 2 \times padding)}{Stride}$$

$$\frac{(3 - 2\ + 1 + 2)}{1} = 4$$

Padding

- Why?
  - Preserve spatial dimensions.
    - What should be the padding size?
  - Control reduction rate of dimensions.
  - Allow convolutions on small-sized inputs.
- Reduce boundary effects.
- Types of Padding:
  - Valid: No padding (reduces dimensions).
  - Same: Preserves input dimensions.



$$P = \frac{F - 1}{2}$$

To preserve dimensions (stride = 1)

$$Output\ Feature\ Map = \frac{(Input\ Feature\ Map - Filter + Stride + 2 \times padding)}{Stride}$$

Convolution (Stride 1)

Filter (3x3)

0	1	0
1	1	1
0	1	0

Input Feature Map (5x5)

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

Output Feature Map (3x3)

7	8	8
5	6	7
6	5	7

Convolution (Stride 2)

Filter (3x3)

0	1	0
1	1	1
0	1	0

Input Feature Map (5x5)

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

Output Feature Map (2x2)

7	8
6	7

Convolution (Stride 3)

Filter (3x3)

0	1	0
1	1	1
0	1	0

Input Feature Map (5x5)

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

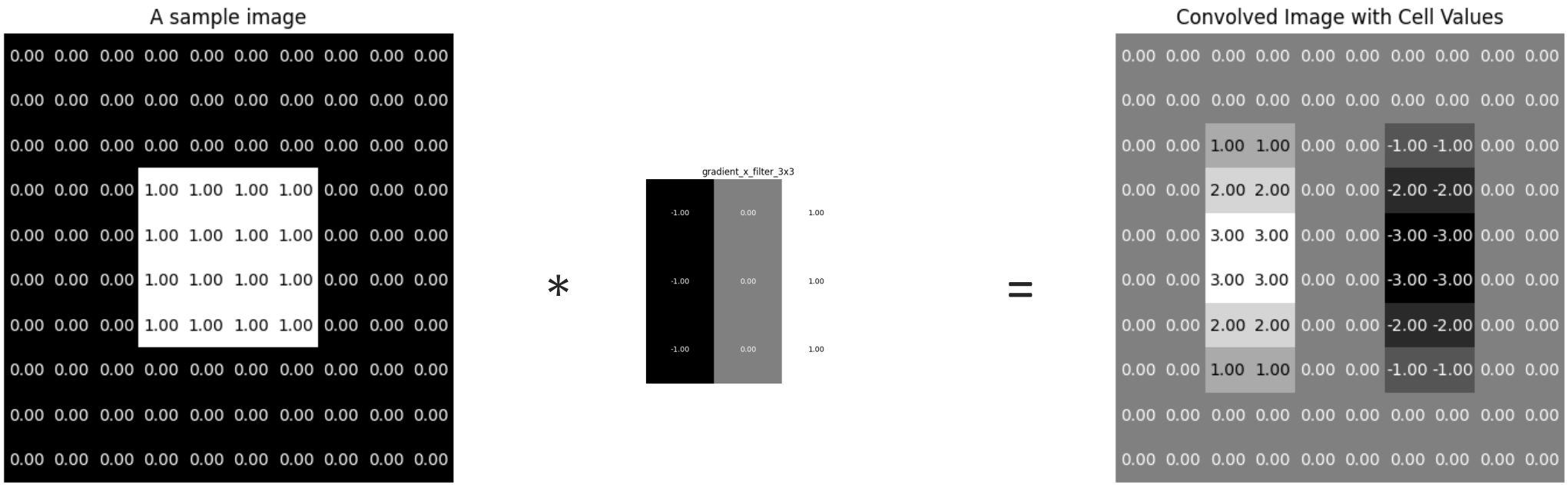
Output Feature Map (1x1)

7

Source: eyeriss.mit.edu

$$Output\ Feature\ Map = \frac{(Input\ Feature\ Map - Filter + Stride)}{Stride}$$

CNN Homework



Detecting Edges Using Convolution

Another Example

Checkerboard Input Image Segment

255	0	255	0	255
0	255	0	255	0
255	0	255	0	255
0	255	0	255	0
255	0	255	0	255

Checkerboard Filter

1	-1	1
-1	1	-1
1	-1	1

Checkerboard Output Feature Map

510	-765	765	-765	510
-765	1275	-1020	1275	-765
765	-1020	1275	-1020	765
-765	1275	-1020	1275	-765
510	-765	765	-765	510

Non-Checkerboard Input Image Segment

255	255	255	255	255
255	255	255	255	255
255	255	255	255	255
255	255	255	255	255
255	255	255	255	255

Non-Checkerboard Filter

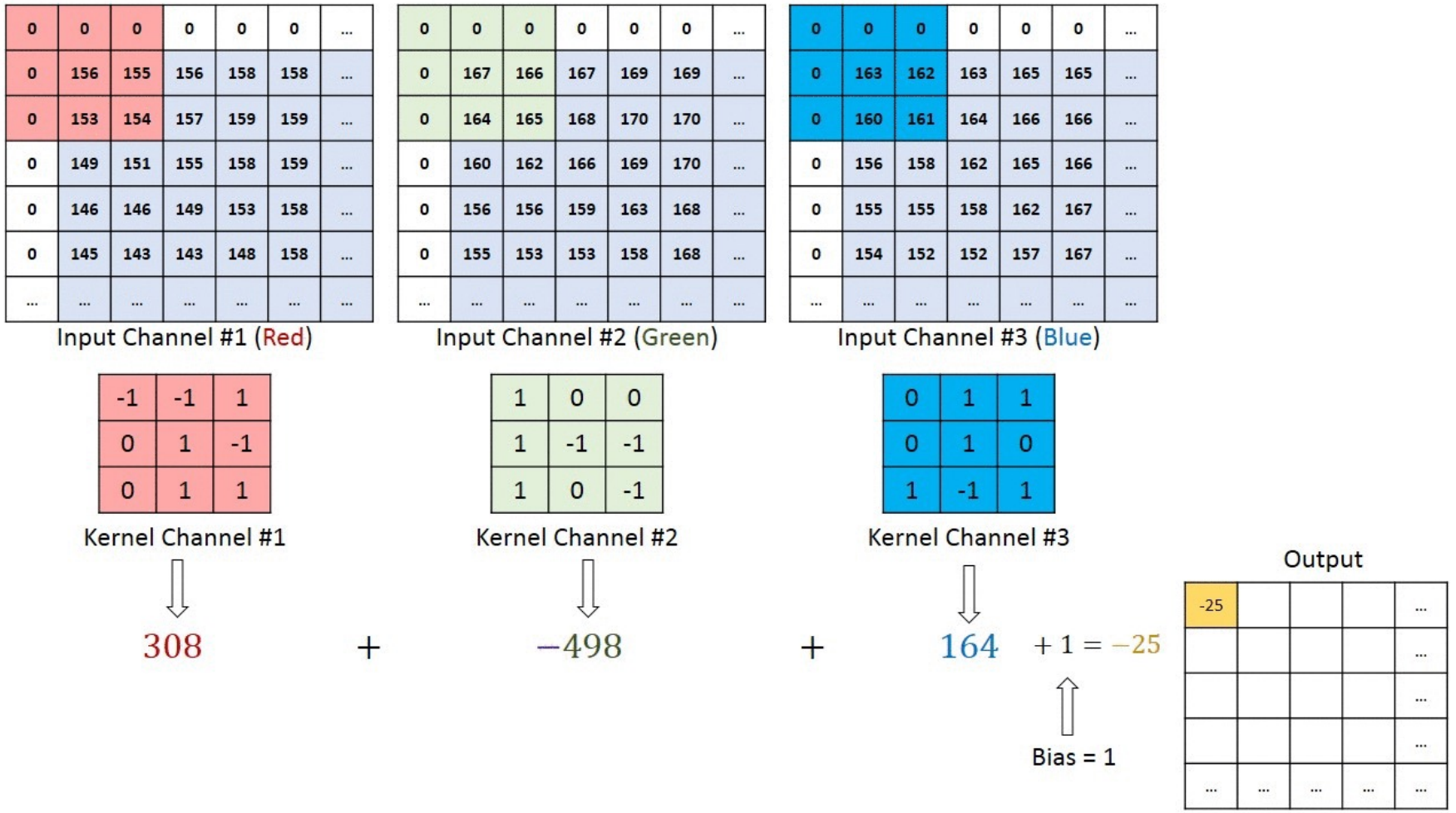
1	-1	1
-1	1	-1
1	-1	1

Non-Checkerboard Output Feature Map

0	0	0	0	0
0	255	255	255	0
0	255	255	255	0
0	255	255	255	0
0	0	0	0	0

Detecting Textures Using Convolution

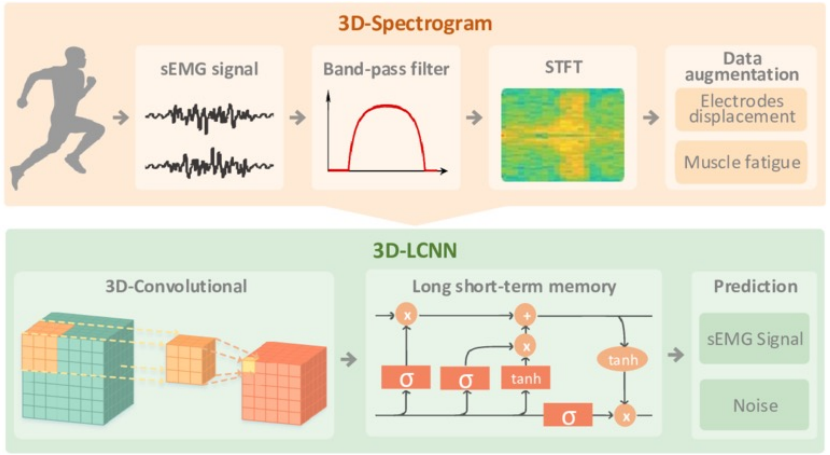
Convolution on a 3-Channel Image (Typically RGB Image in 2D Convolution)



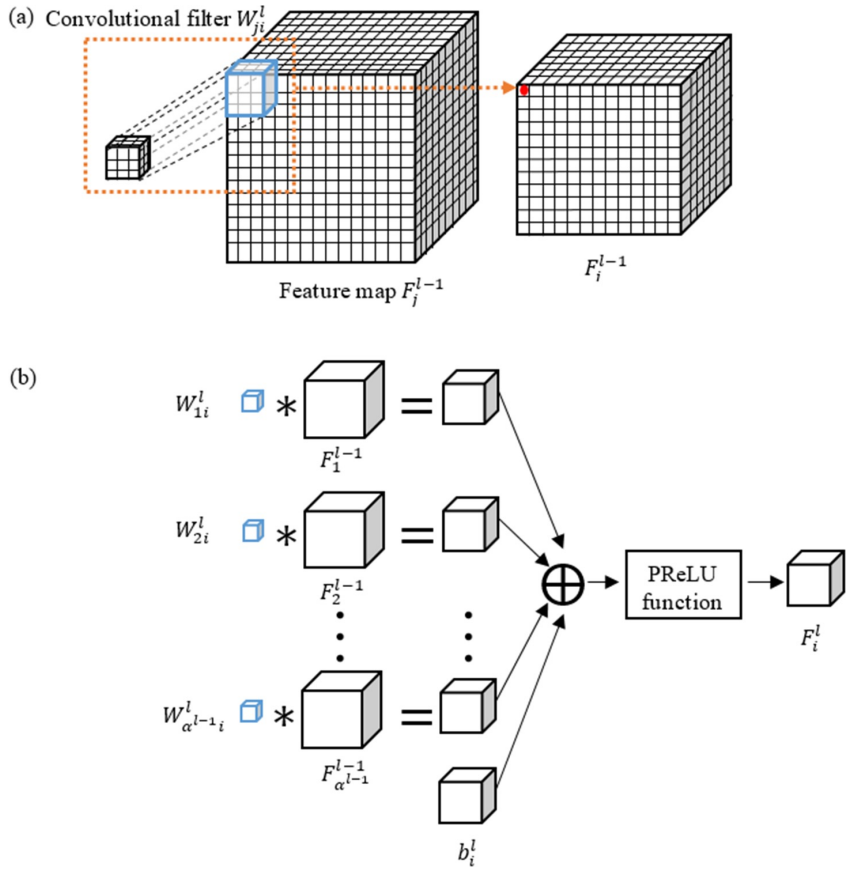
Convolution of a 3 channel image with a 3x3x3 kernel

3D Convolution

- Extension of 2D convolution to volumetric data.
- From (k,k,C) in 2D to (k,k,k,C) in 3D.
  - C: Number of channels
- Applications:
  - Medical Imaging: Capture features across MRI/CT slices.
  - Video Processing: Analyze spatial and temporal features in sequences.
- Computational Complexity:
  - More parameters and calculations than 2D.



Source: Min-Wen Lin1 et al.



Generation of the ith feature map in the lth layer

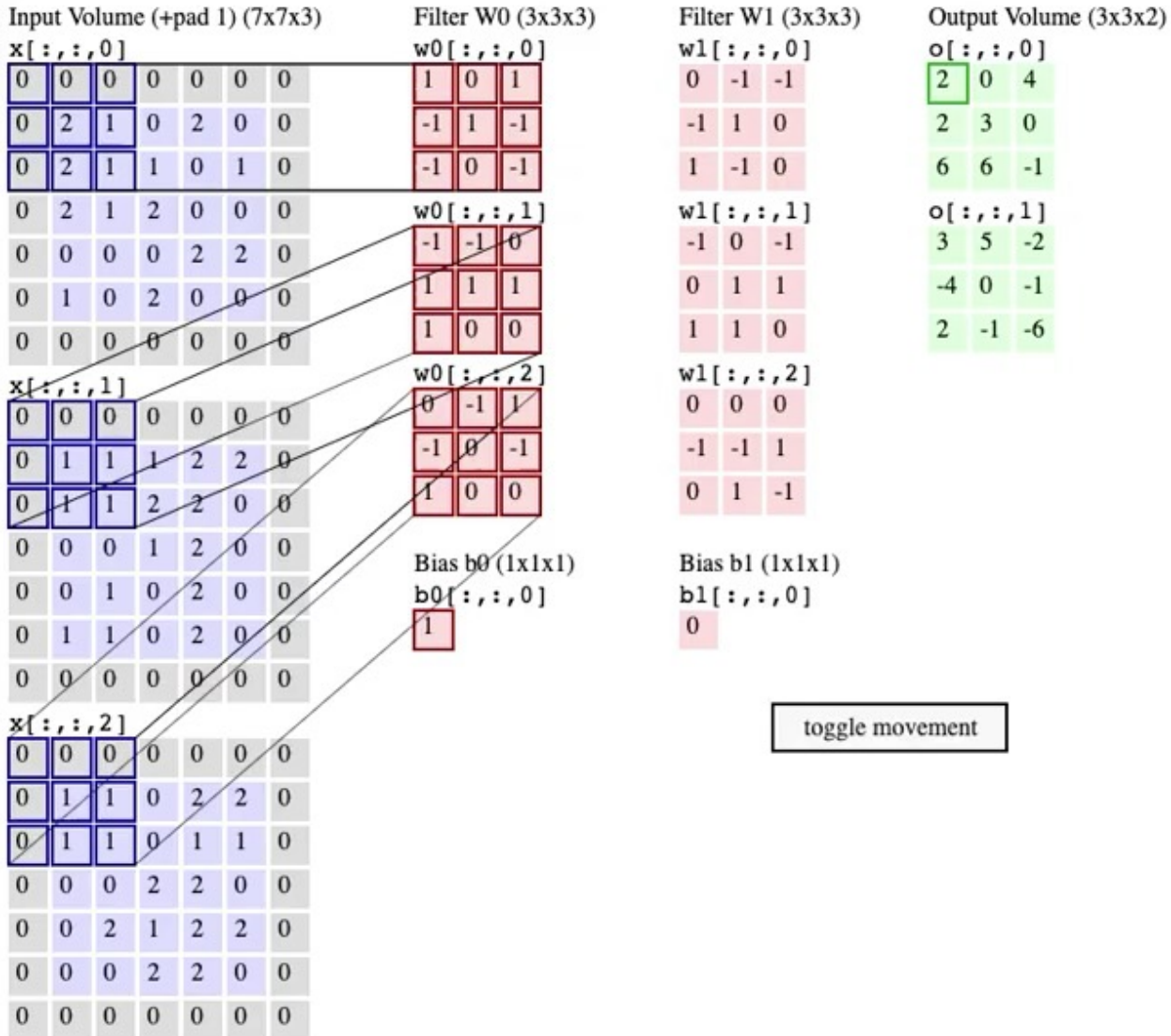
Source: Yan Liu et all.

Several frames as potential input for a 3D Convolution

Source: mediacollege.com

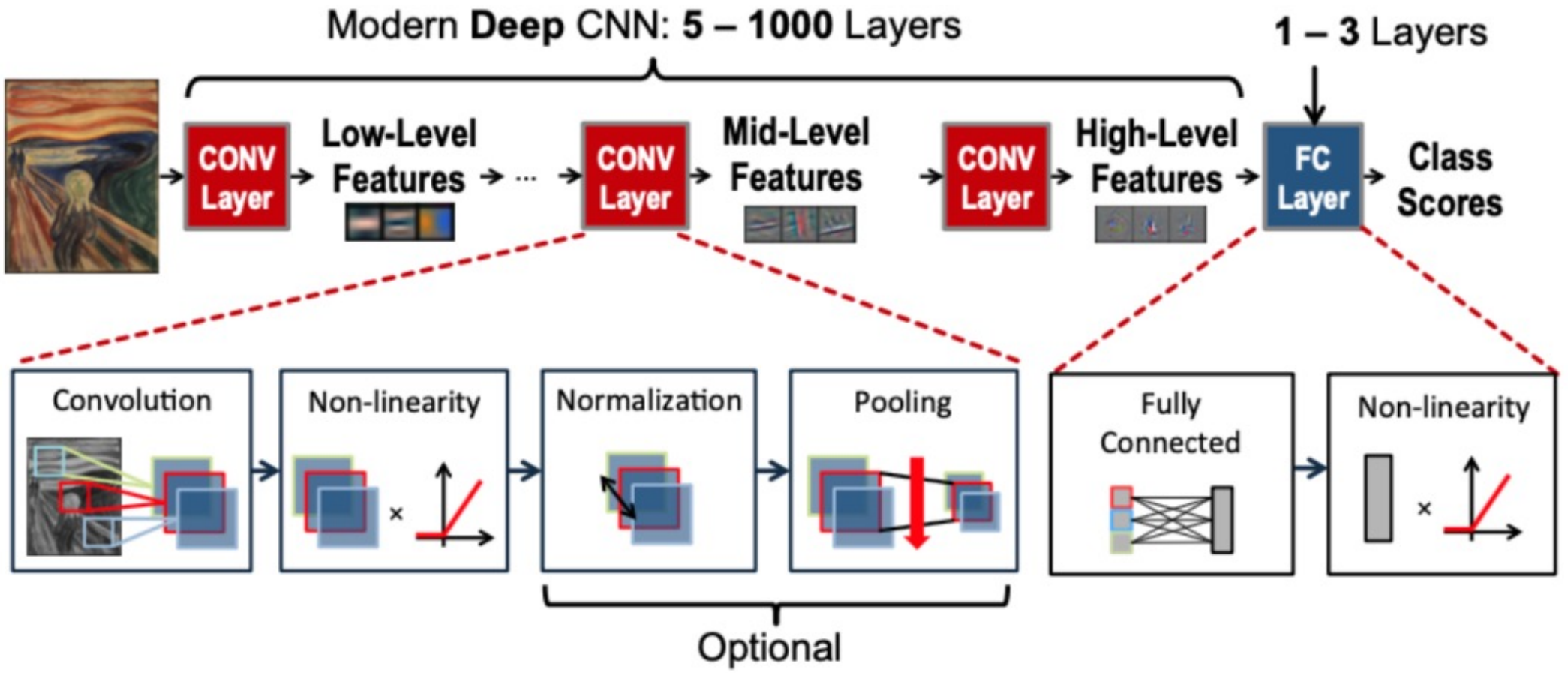
3D Conv is Different than Conv on 3-Channel Image!

- **Convolution on a 3-Channel Image (Typically RGB Image in 2D Convolution):**
  - The filter (or kernel) has the same depth as the number of channels in the image.
    - For instance, for an RGB image, the filter would have a depth of 3.
  - The convolution operation is applied to all channels at once, and the results are summed up to produce a single output per filter application. Each filter gives a 2D feature map as output.
  - The depth of the filter (3 in the case of RGB images) does not change as it moves across the image.
  - Detecting features like edges, textures, or color patterns within the 2D spatial plane of the image.
- **3D Convolution:**
  - Both the input and the filters are 3-dimensional.
  - Common in like medical imaging, video processing or sequences of 2D frames.
  - The filter is a 3D cube that moves along three axes: height, width, and depth.
  - Each application of the filter produces a single output value, and moving the filter throughout the volume generates a 3D feature map.
  - Detect features across a volume, considering spatial relationships in three dimensions (such as changes over time in video frames or depth in volumetric data).



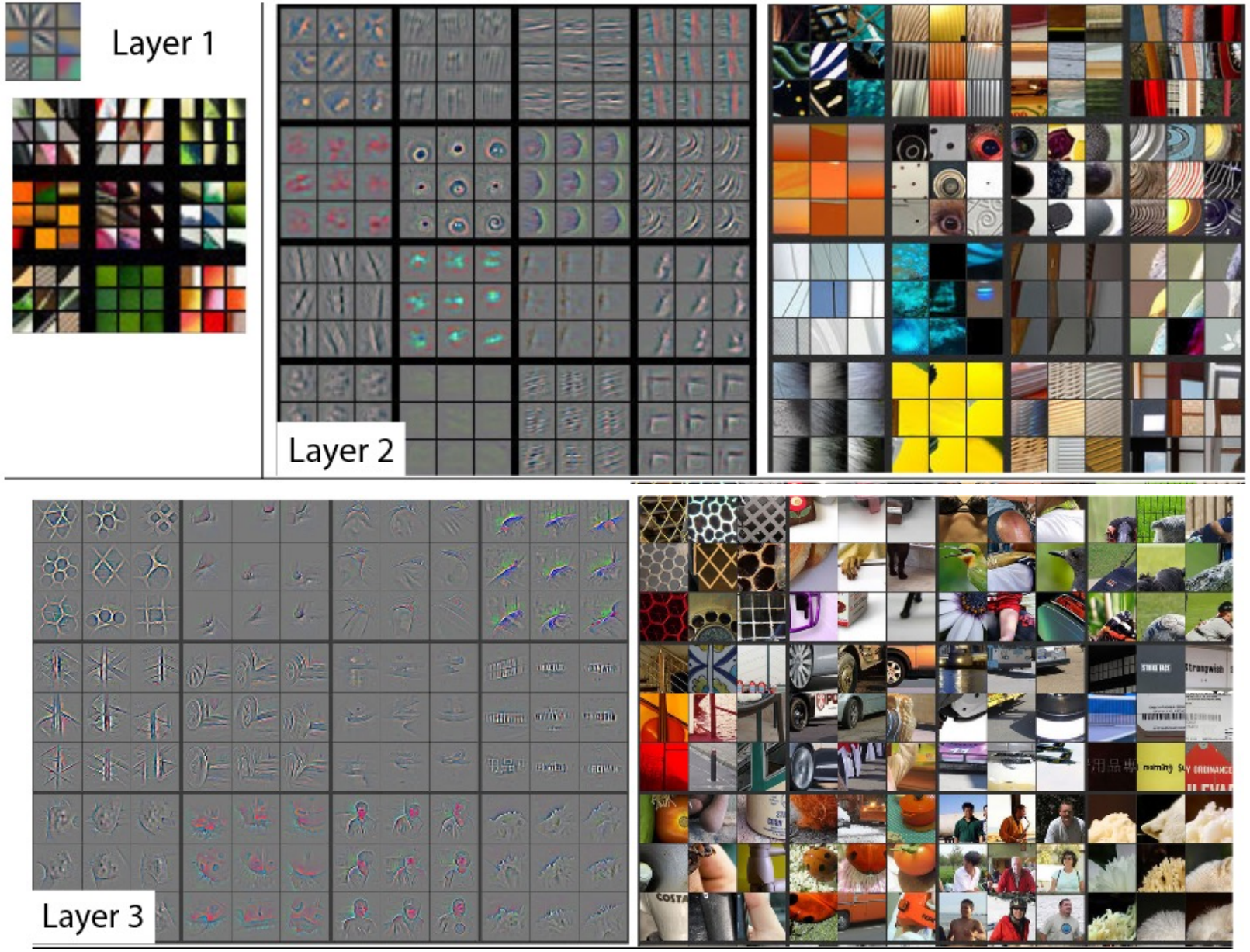
Source: https://cs231n.github.io/convolutional-networks/

Back to CDNNs



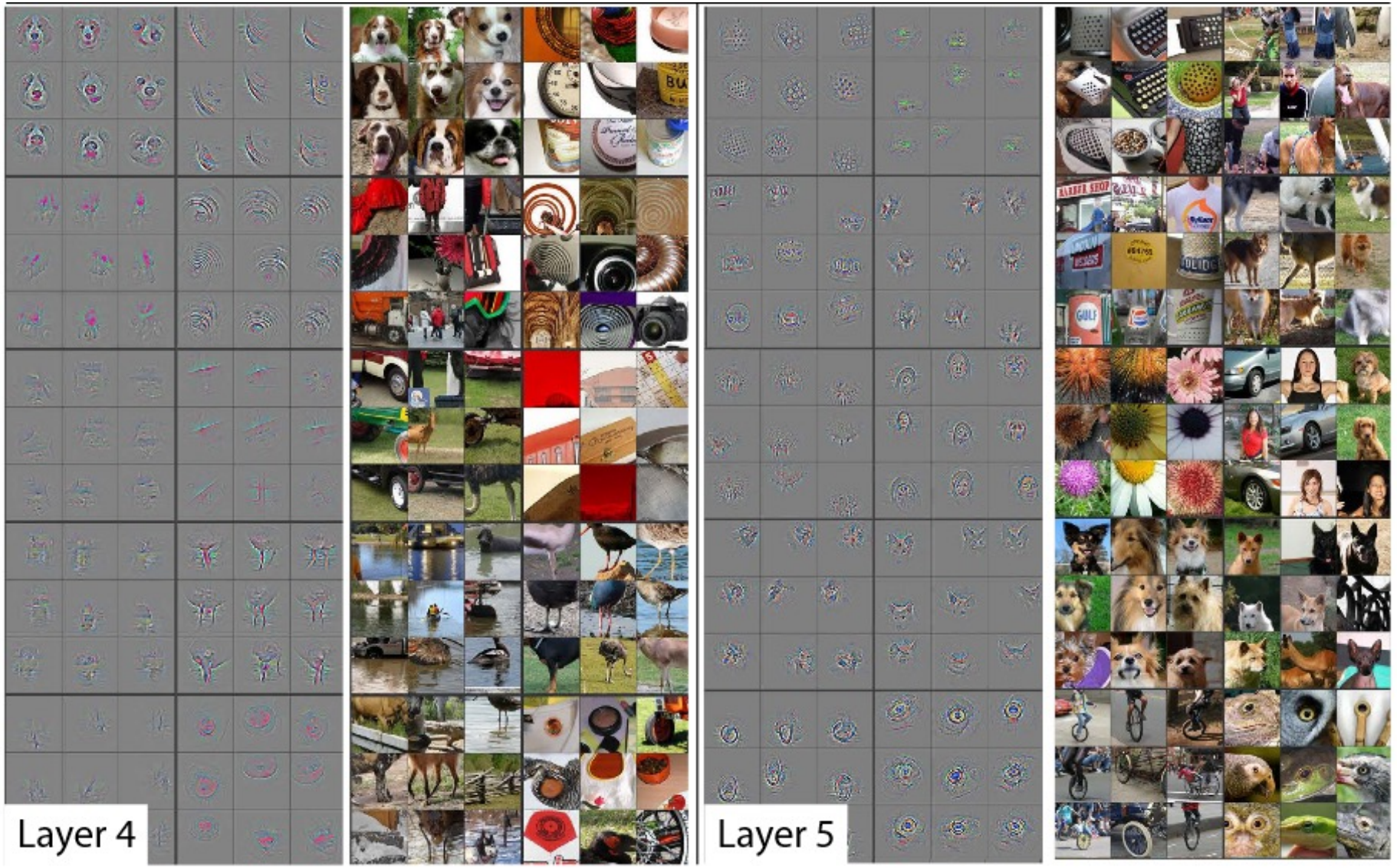
Deep Convolutional Neural Networks

Source: eyeriss.mit.edu



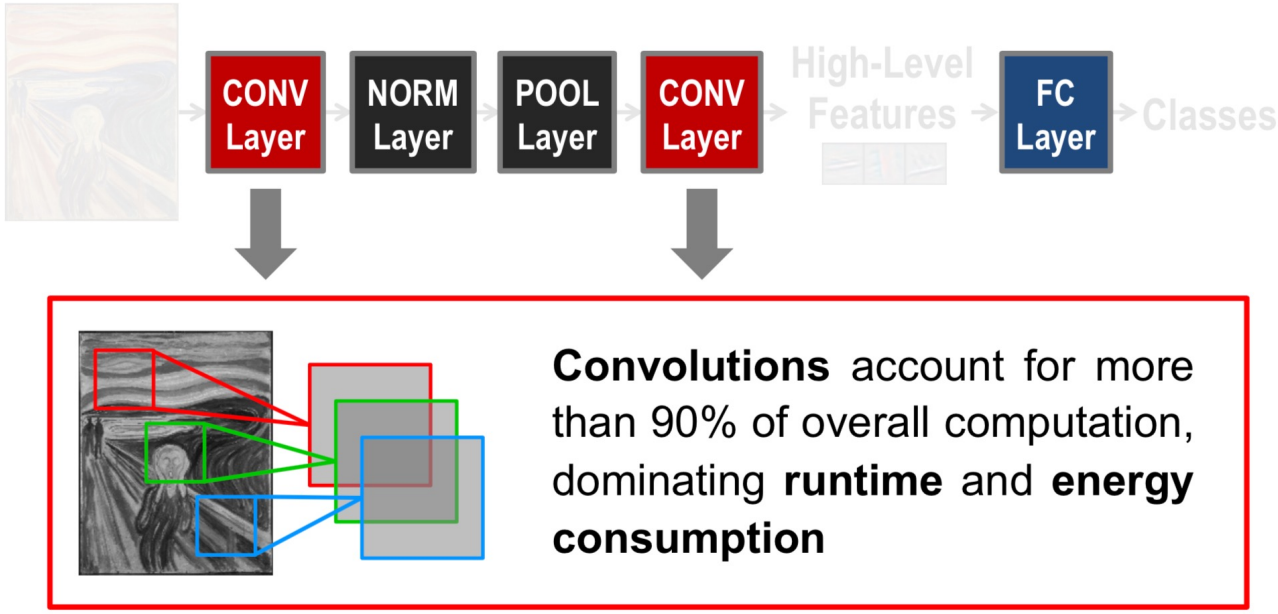
Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data.

Source: Visualizing and Understanding Convolutional Networks. Matthew D. Zeiler et all.



Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data.

Source: Visualizing and Understanding Convolutional Networks. Matthew D. Zeiler et all.

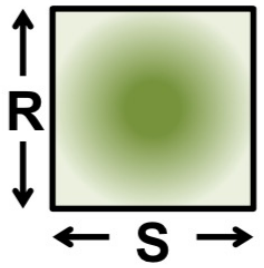


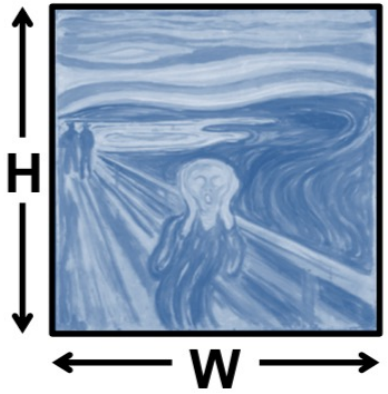
Deep Convolutional Neural Networks

Convolution process

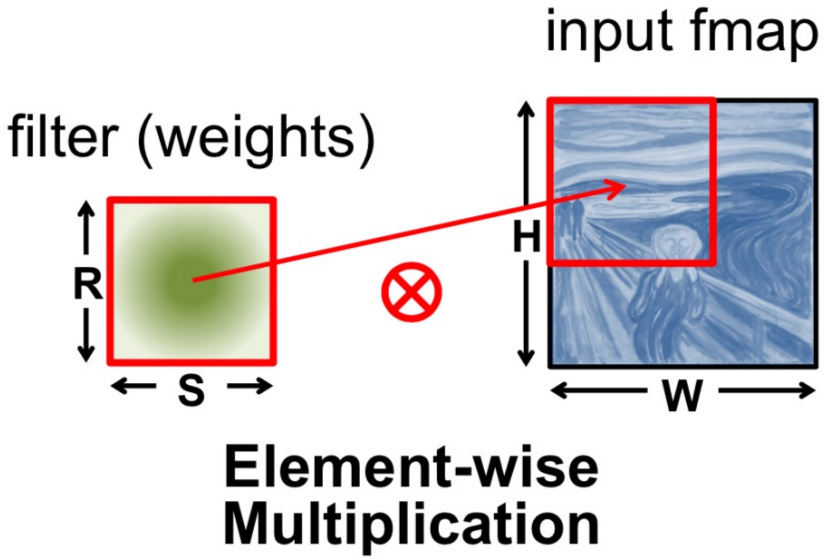
a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)

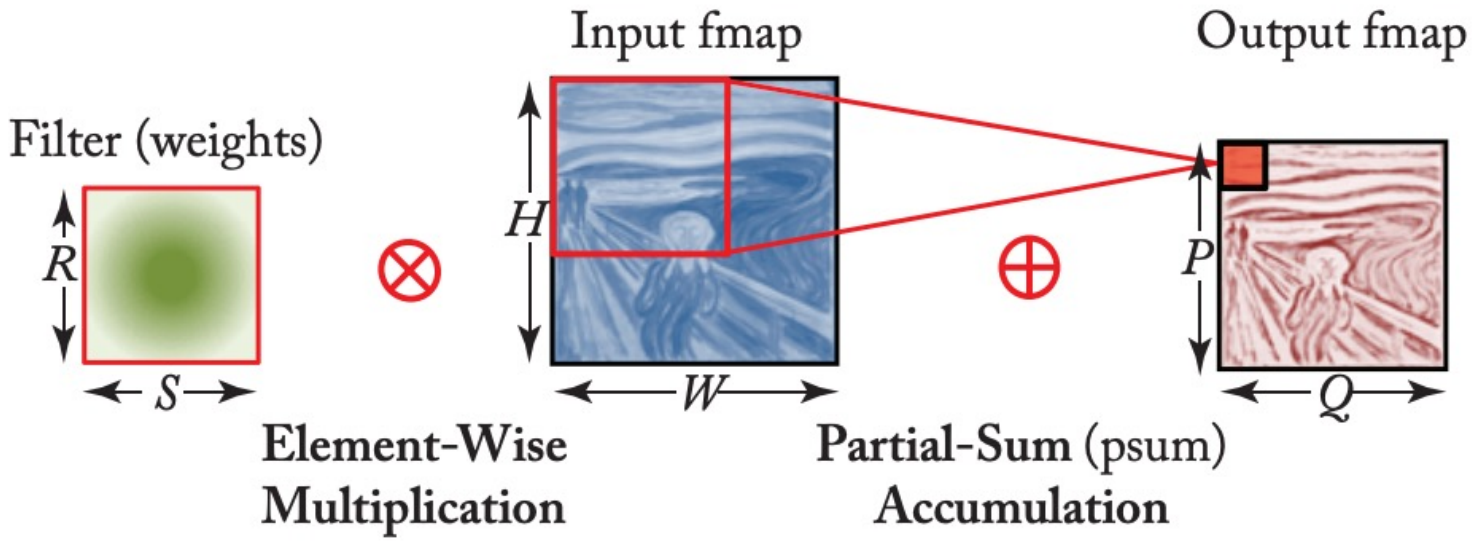




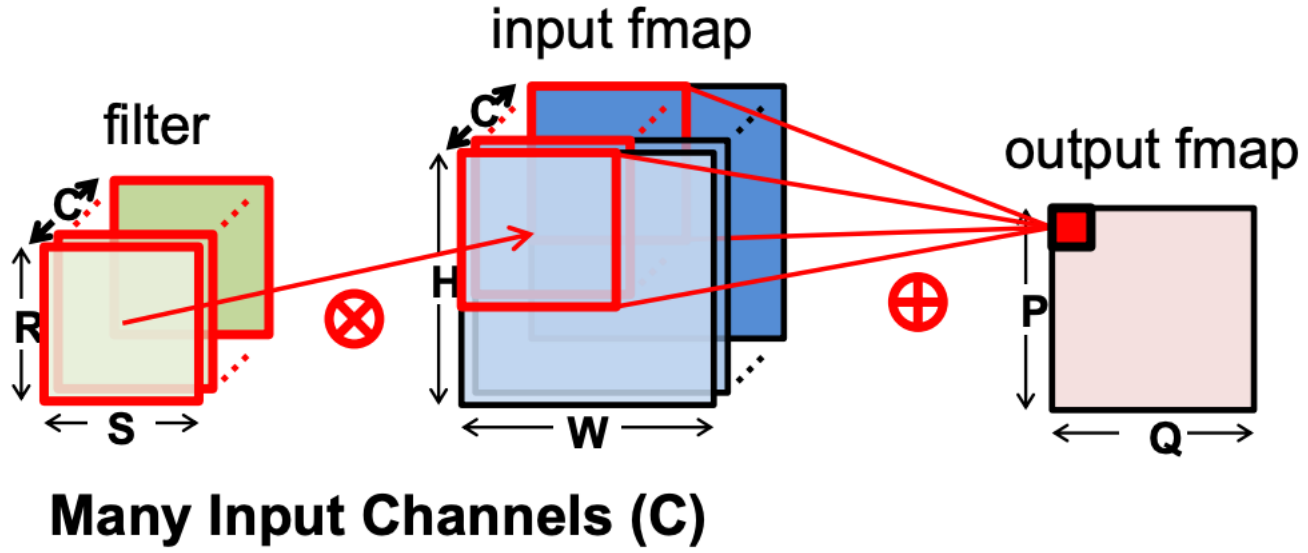
Convolution (CONV) Layer



Convolution (CONV) Layer



Convolution (CONV) Layer

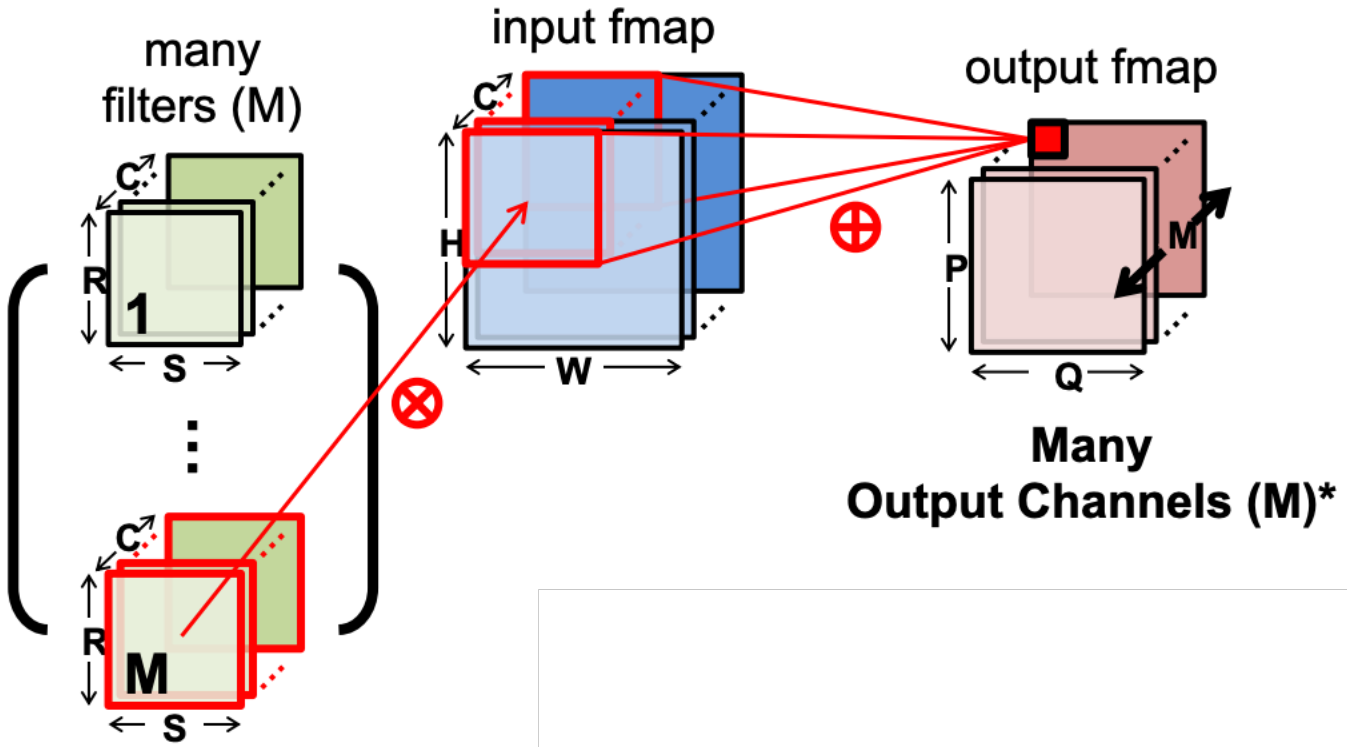


There are **C input channels** and **1 filter** of **depth C.**

- Each **C filters** produce **1 output fmap** of **depth 1.**

Learning across the channels

E.g. R, G, B Channels

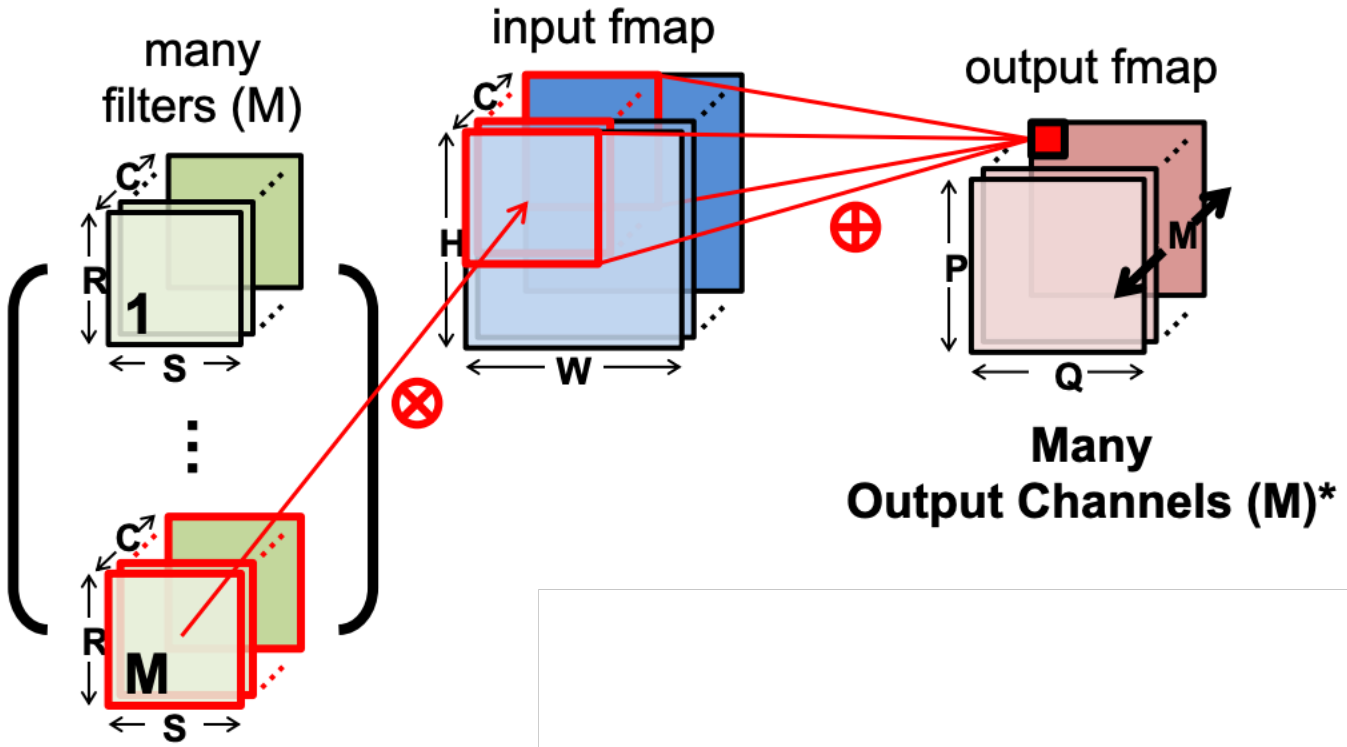


There are **C input channels** and **M filters** of **depth C.**

- Each **C filters** produce **1 output fmap** of **depth 1**

Multiple output channels

M = Number of output channels = Number of filters = Width of the network



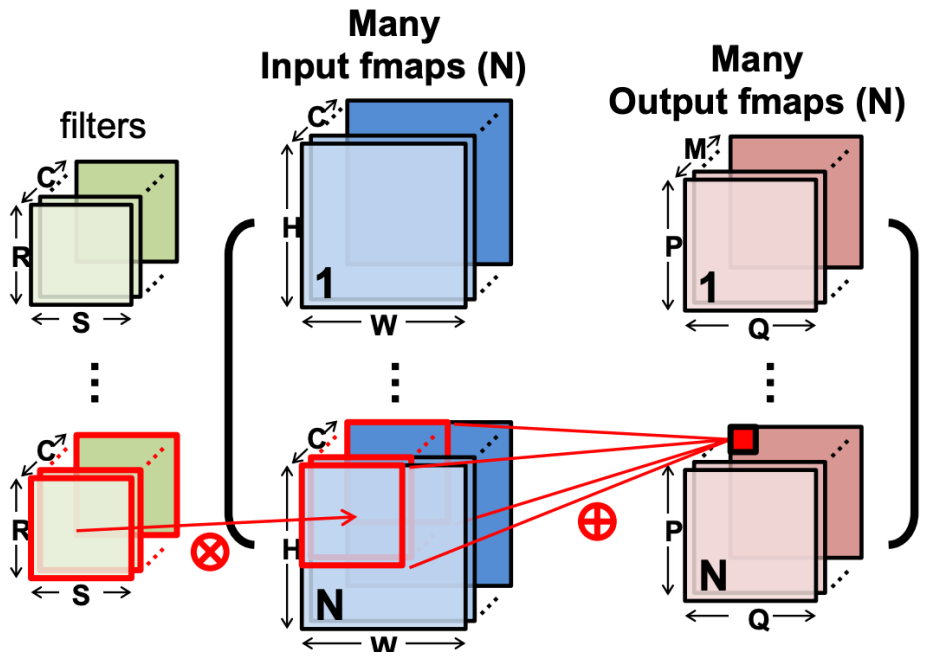
There are **C input channels** and **M filters** of **depth C.**

- Each **C filters** produce **1 output fmap** of **depth 1**
- There are **MxC** filters, so the final depth of output fmap is **M**.

Multiple output channels

M = Number of output channels = Number of filters = Width of the network

Batch multiple inputs



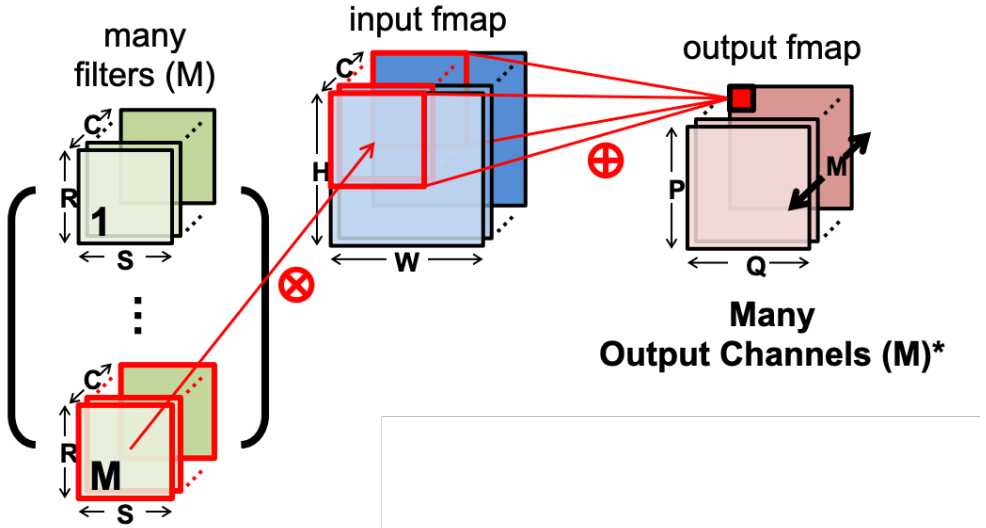
There are **NxC input channels** and **M filters** of **depth C.**

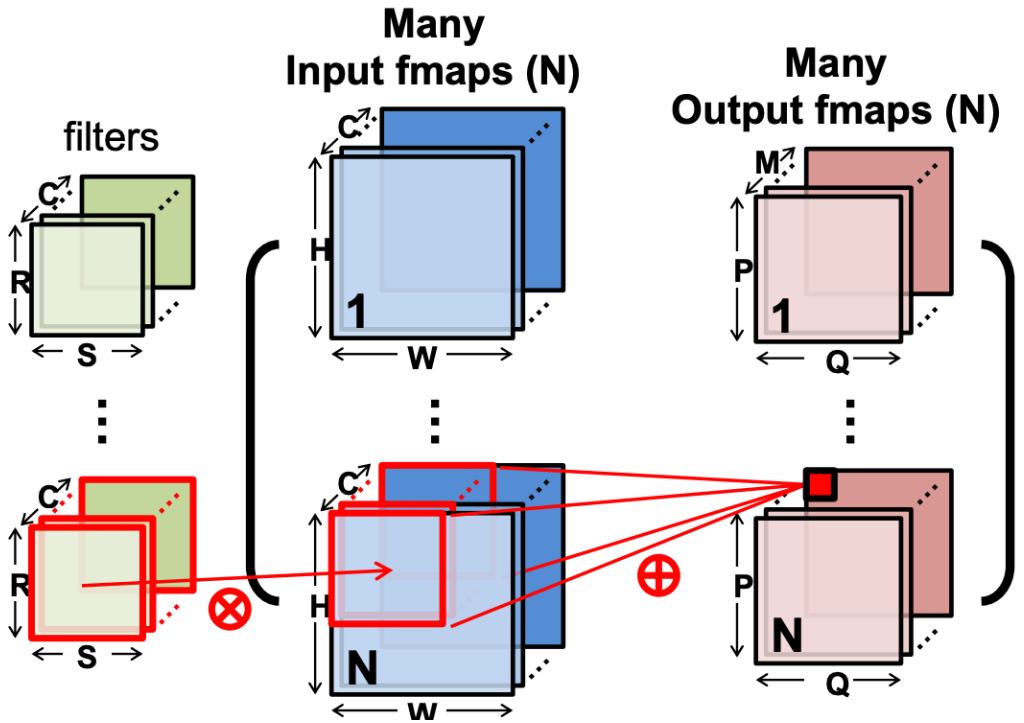
- Each **C filters** produce **1 output fmap** of **depth 1**
- There are **MxC** filters, so the final depth of output fmap is **M**.
- There will be **N** such output fmaps

Processing N Inputs in one batch.

CNN Decoder

- N – Number of input fmaps/output fmaps (batch size)
- C – Number of channels in input fmaps (activations) & filters (weights)
- H – Height of input fmap (activations)
- W – Width of input fmap (activations)
- R – Height of filter (weights)
- S – Width of filter (weights)
- M – Number of channels in output fmaps (activations)
- P – Height of output fmap (activations)
- Q – Width of output fmap (activations)
- U – Stride of convolution





CONV Layer Tensor Computation

Output fmap (O) — Biases (B) — Input fmap (I) — Filter weights (W)

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

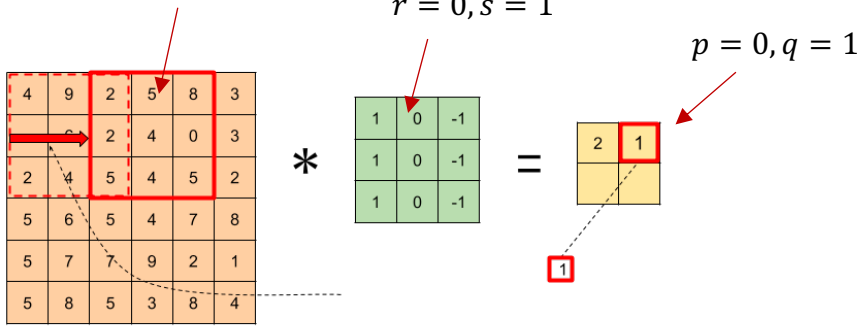
$$0 \le n < N, 0 \le m < M, 0 \le p < P, 0 \le q < Q,$$
$$P = (H - R + U)/U, Q = (W - S + U)/U.$$

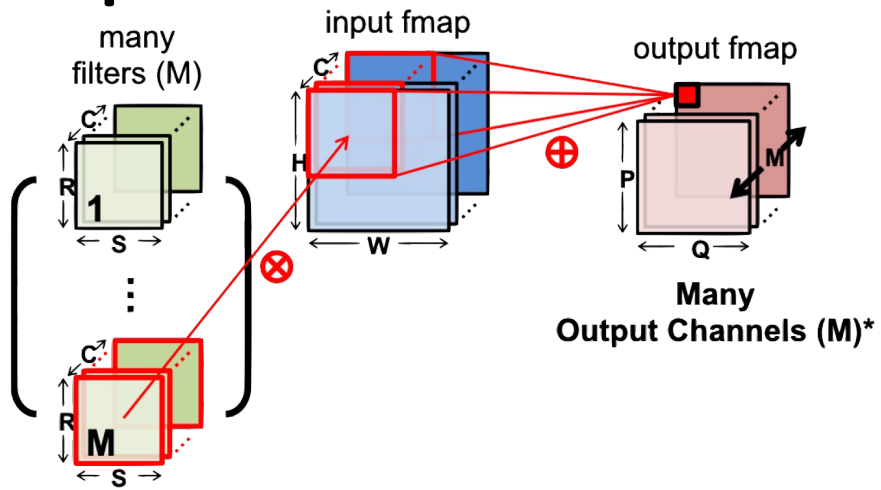
Shape Parameter	Description
N	batch size of 3-D fmaps
M	# of 3-D filters / # of ofmap channels
C	# of ifmap/filter channels
H/W	ifmap plane height/width
R/S	filter plane height/width (= H or W in FC)
P/Q	ofmap plane height/width (= 1 in FC)
U	Stride size

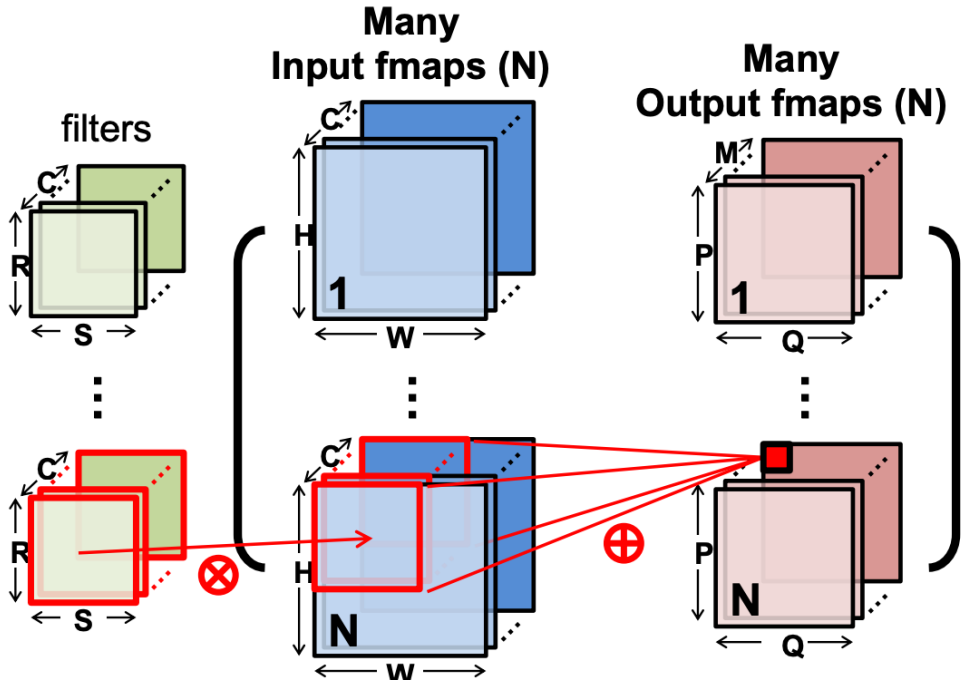
$I[U\times p + r][U\times q + s] = I[0][3]$

$r = 0, s = 1$

$p = 0, q = 1$





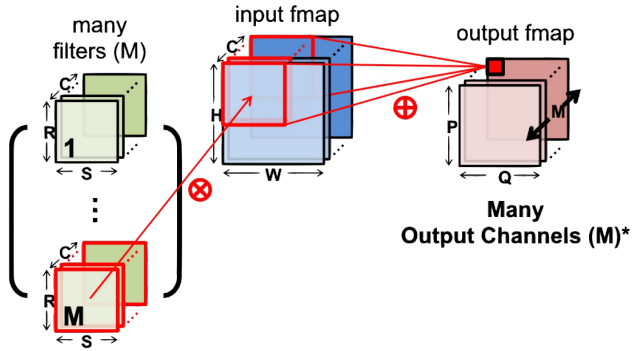


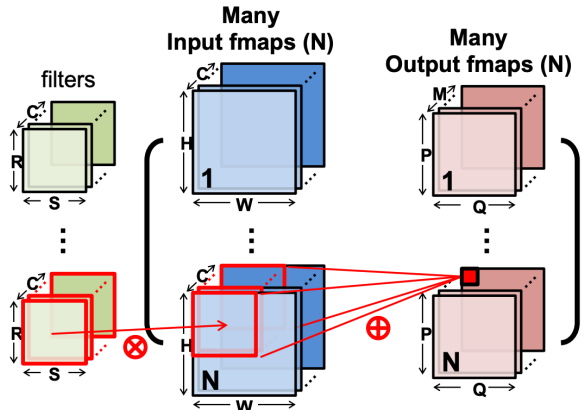
Tensor Notation

Output fmap (O) Biases (B) Input fmap (I) Filter weights (W)

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

$$0 \le n < N, 0 \le m < M, 0 \le p < P, 0 \le q < Q,$$
$$P = (H - R + U)/U, Q = (W - S + U)/U.$$



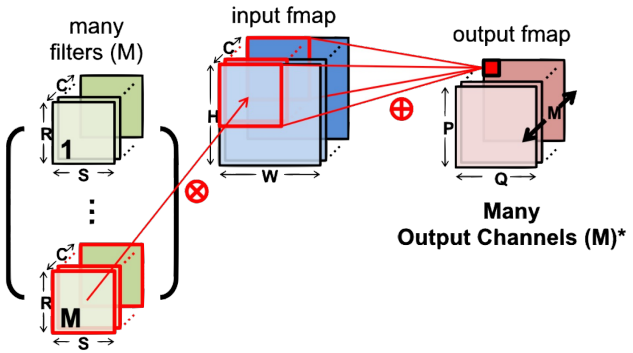


$$O_{n,m,p,q} = B_m + \sum_{c,r,s} I_{n,c,Up+r,Uq+s} \times F_{m,c,r,s}$$

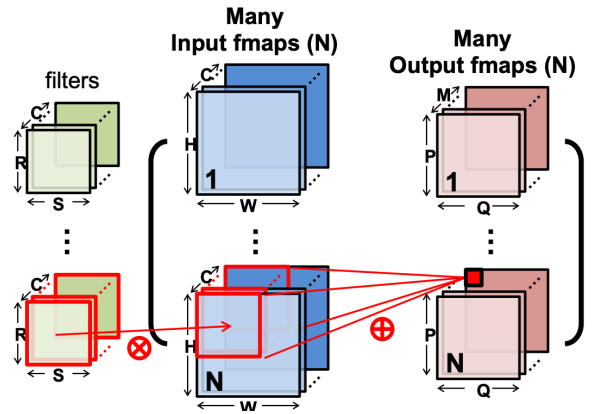
Naïve Implementation

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

Shape Parameter	Description
N	batch size of 3-D fmaps
M	# of 3-D filters / # of ofmap channels
C	# of ifmap/filter channels
H/W	ifmap plane height/width
R/S	filter plane height/width (= H or W in FC)
P/Q	ofmap plane height/width (= 1 in FC)
U	Stride size



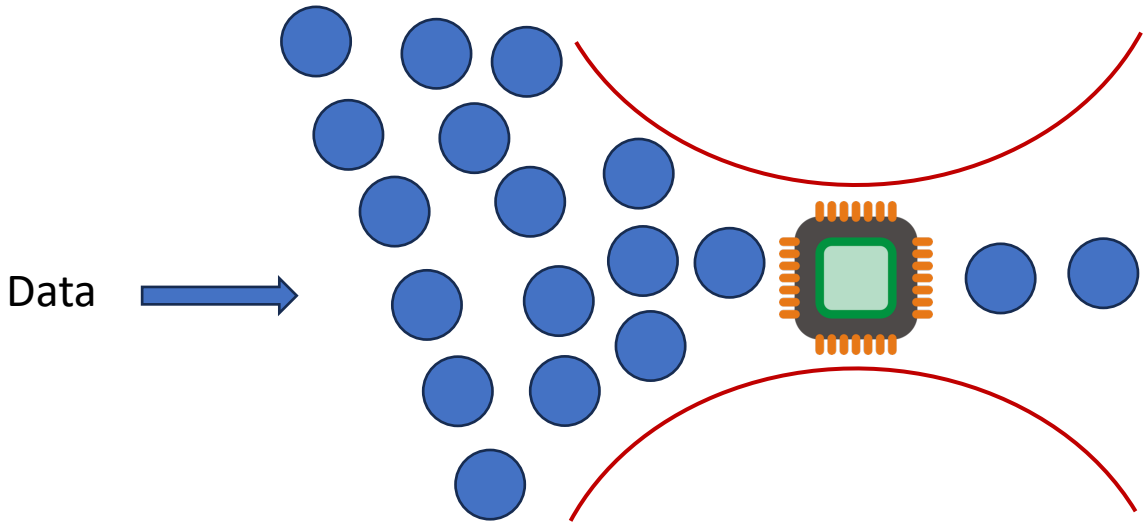

```
# input shape: [batch_size, input_channels, input_height, input_width]
# kernel shape: [output_channels, input_channels, kernel_height, kernel_width]
# bias is a tensor of shape [output_channels]

output = np.zeros((batch_size, output_channels, output_height, output_width))

for n in range(batch_size):
  for m in range(output_channels):
    for q in range(output_width):
      for p in range(output_height):

        output[n, m, p, q] = bias[m]
        for c in range(input_channels):
          for s in range(kernel_width):
            for r in range(kernel_height):
              output[n, m, p, q] += input[n, c, U*p + r, U*q + s] * kernel[m, c, r, s]

        output[n, m, p, q] = Activation(output[n, m, p, q])
```

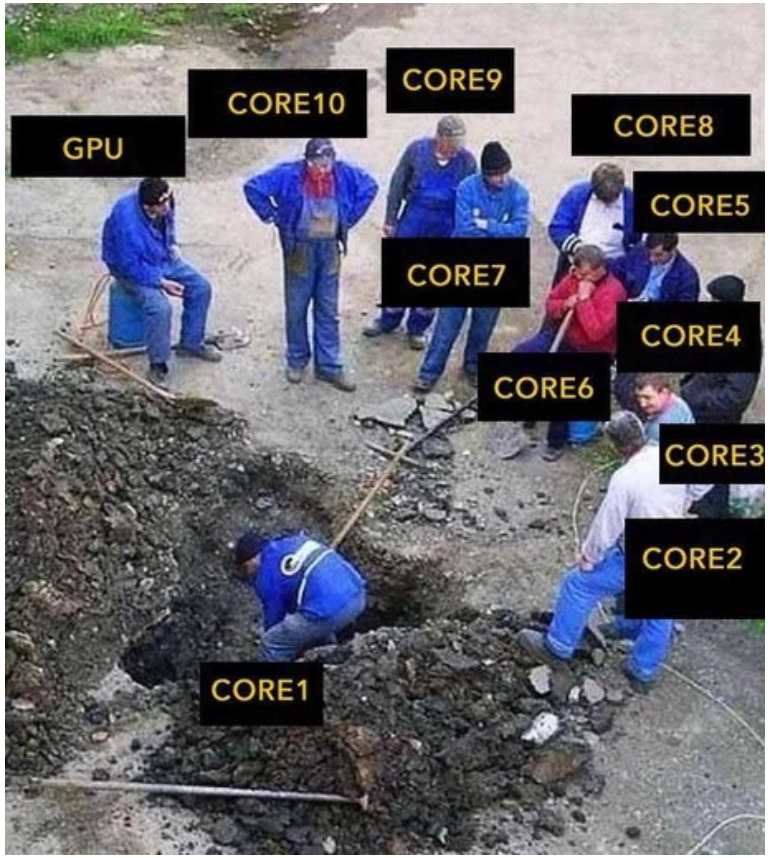


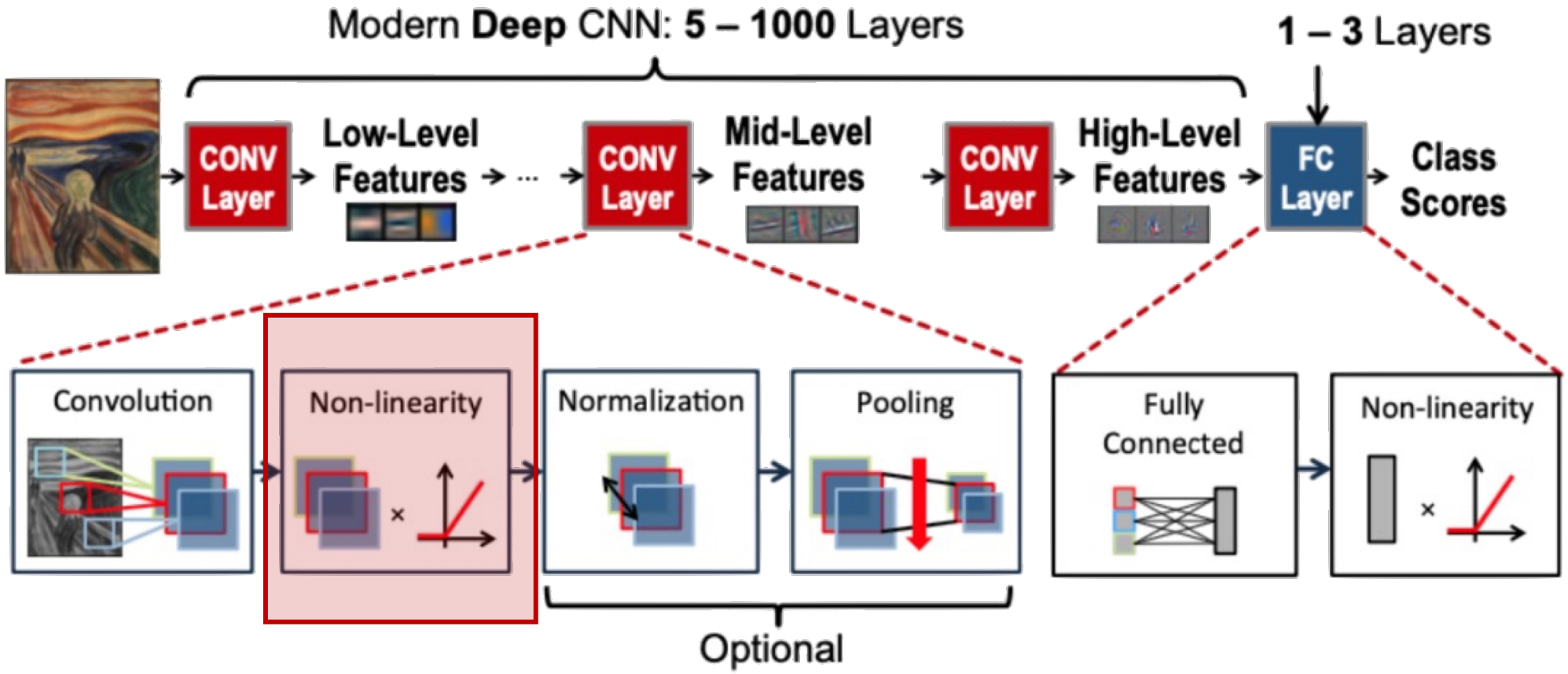
Embarrassingly Parallel Problem

(noun)

A problem where **little or no effort** is needed to separate the problem into a number of **parallel tasks.**





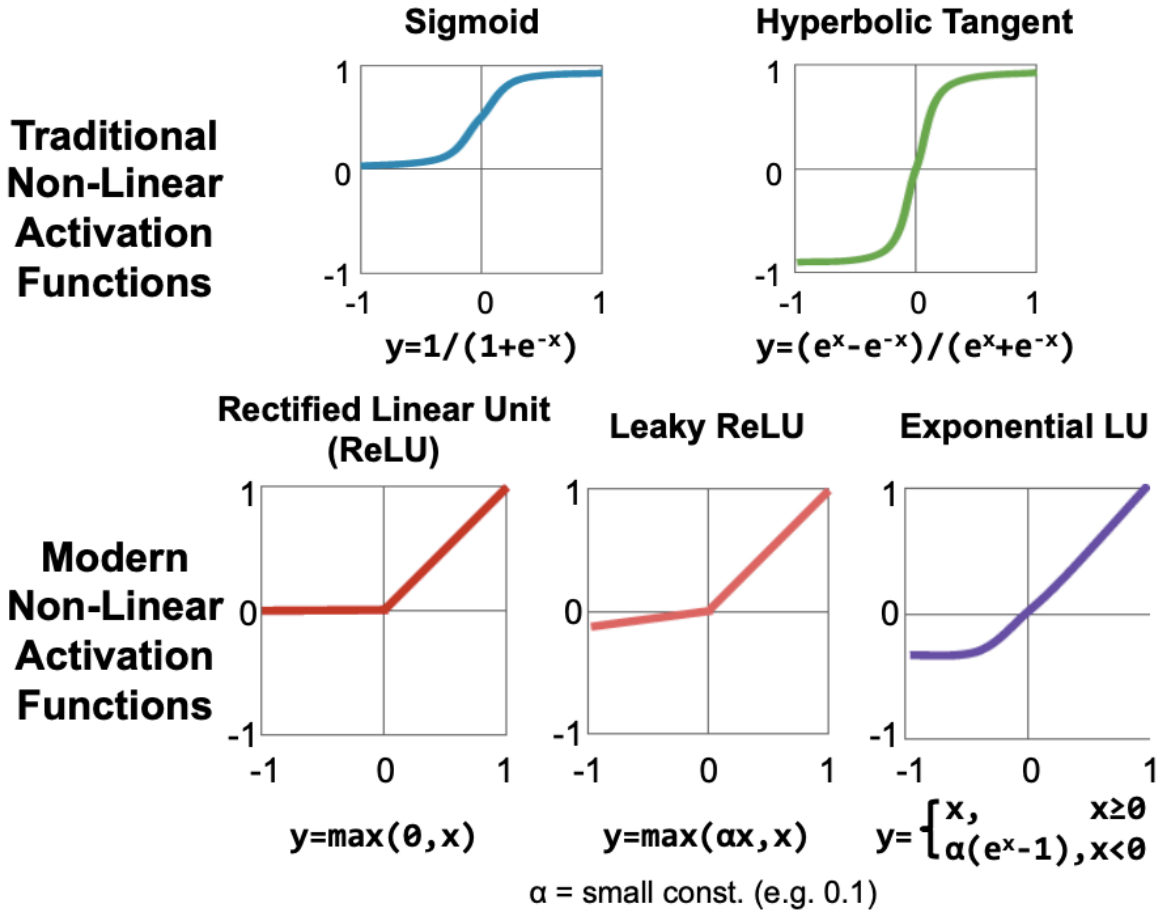


Deep Convolutional Neural Networks

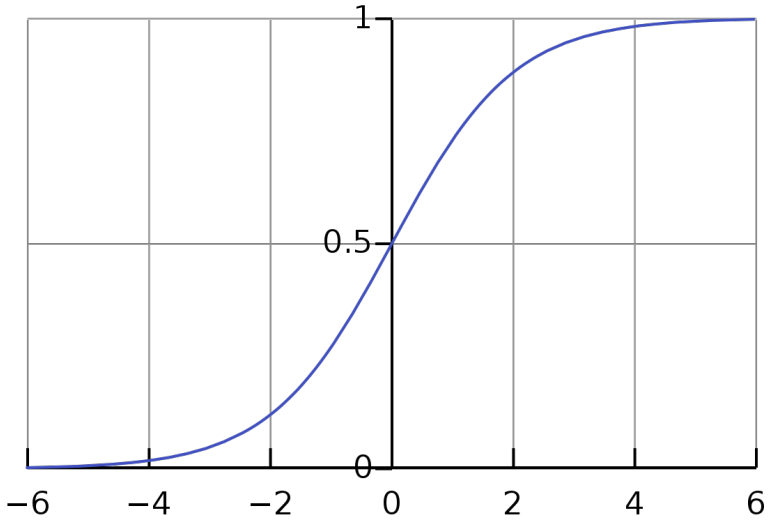
Source: eyeriss.mit.edu

Non-Linearity

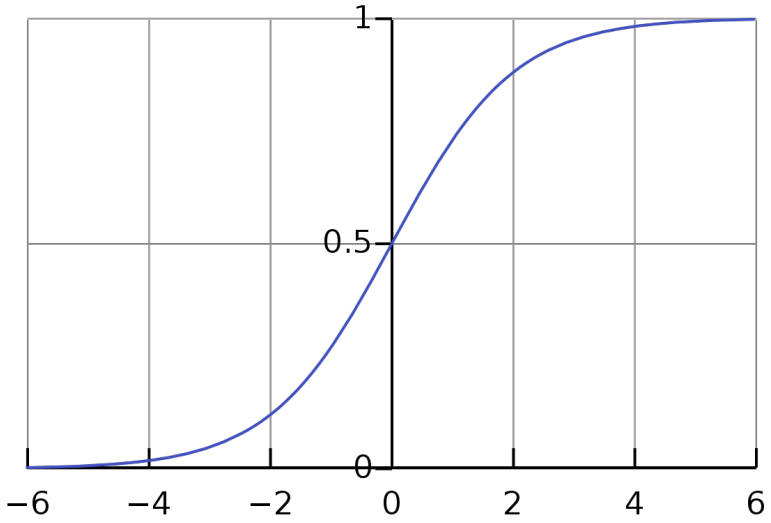
- Activation Function
  - Non-linear function applied to the output of a neuron or layer.
- Why Are Activation Functions Needed?
  - **Model complex**, non-linear relationships in the data.
  - **Depth**: Without non-linearities, no matter how many layers the network has, it would behave like a single-layer model.



Reminder: Sigmoid Function (Logistic Function)

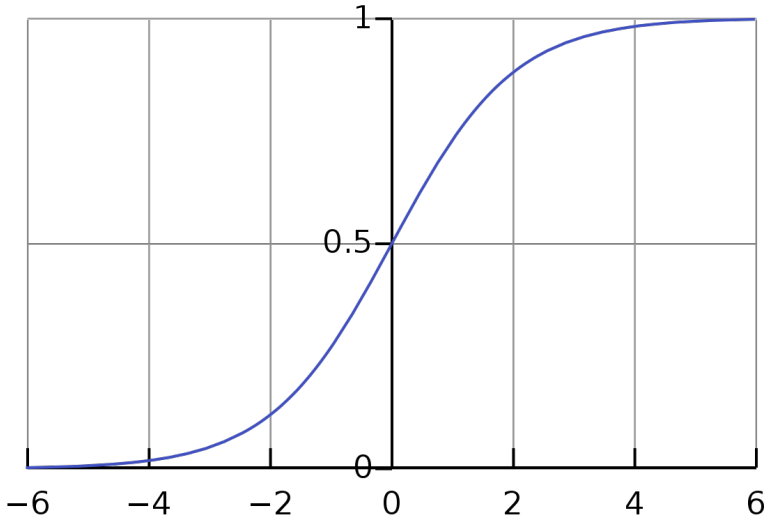


Reminder: Sigmoid Function (Logistic Function)



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

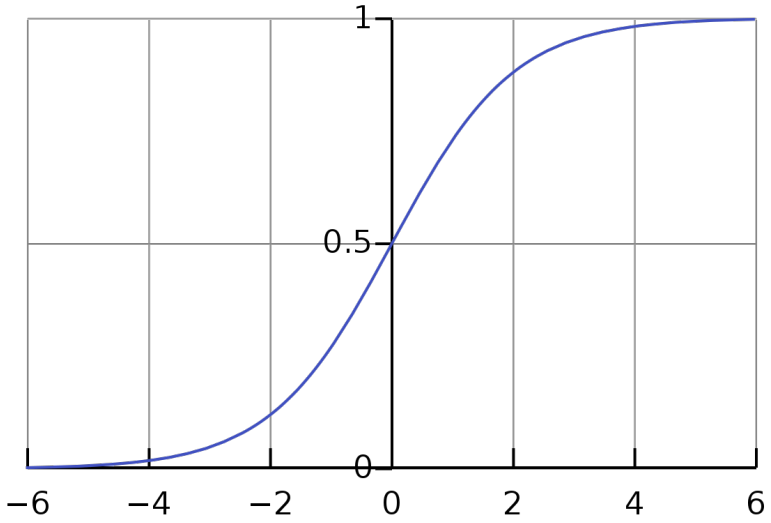
Reminder: Sigmoid Function (Logistic Function)



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

Range is between 0 and 1

Reminder: Sigmoid Function (Logistic Function)

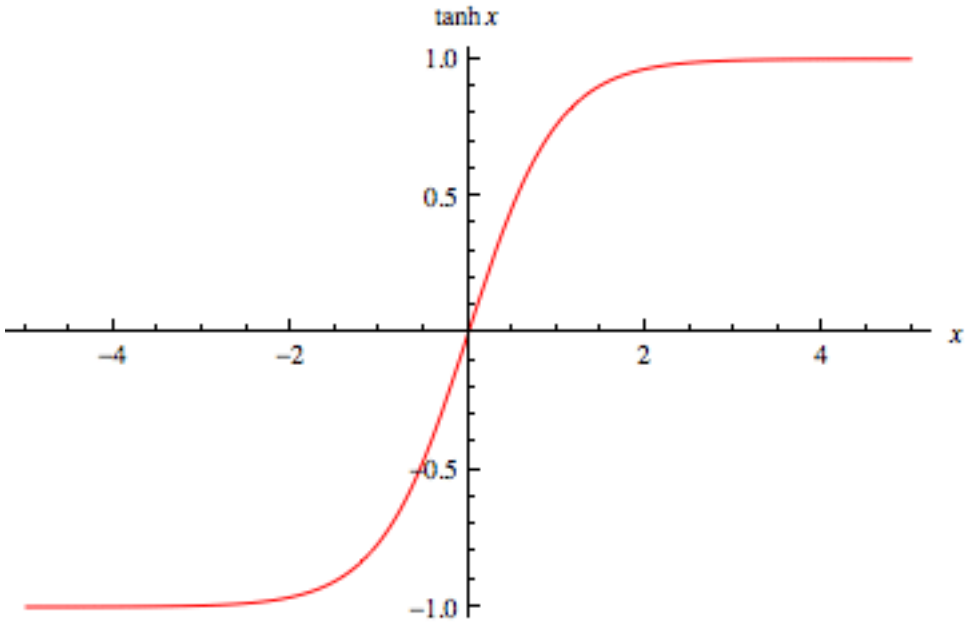


$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{1+e^{x}}$$

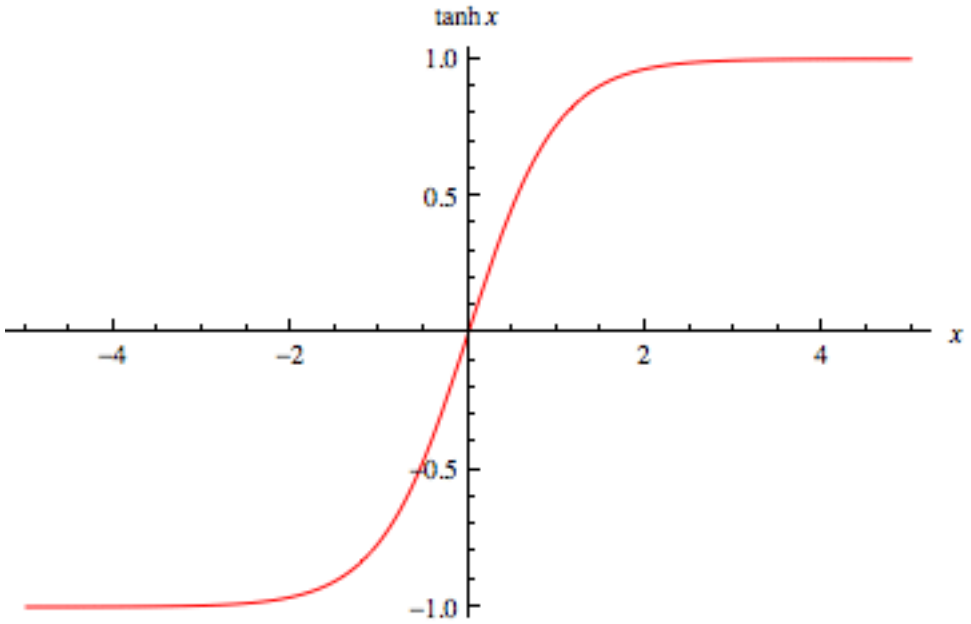
Range is between 0 and 1

$\sigma(f(x))$ maps $f(x)$ to a percentage.

Reminder: tanh Function

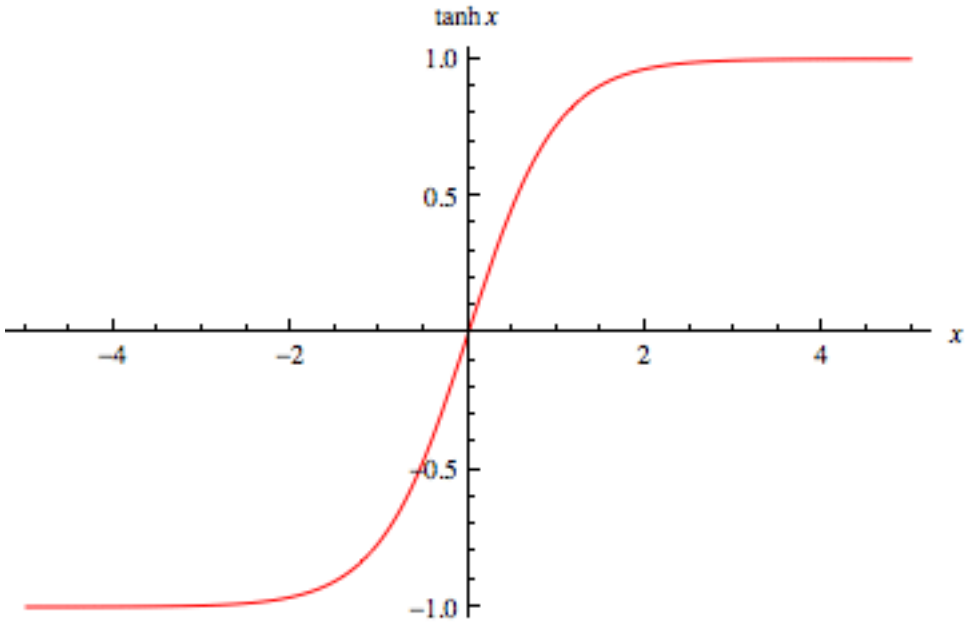


Reminder: tanh Function



$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

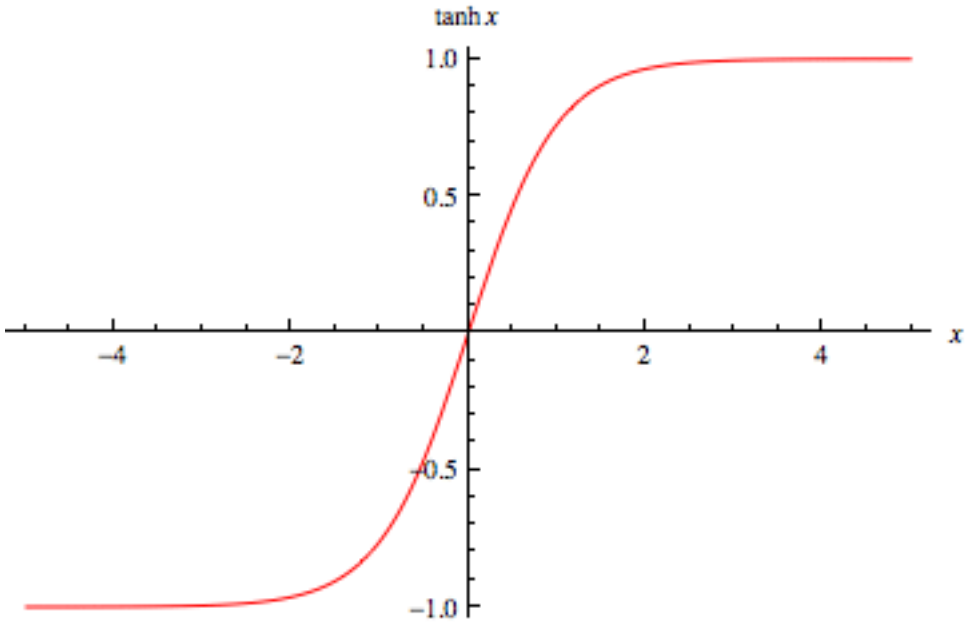
Reminder: tanh Function



$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Range is between -1 and 1

Reminder: tanh Function



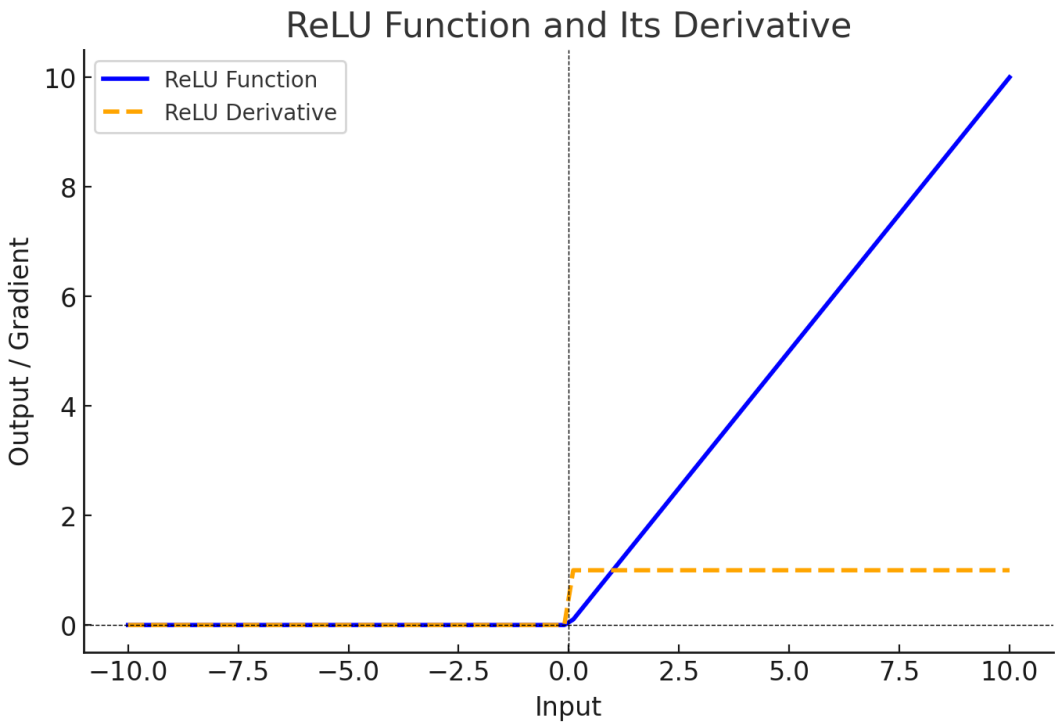
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Range is between -1 and 1

$\sigma(f(x))$ maps $f(x)$ to -1 and 1.

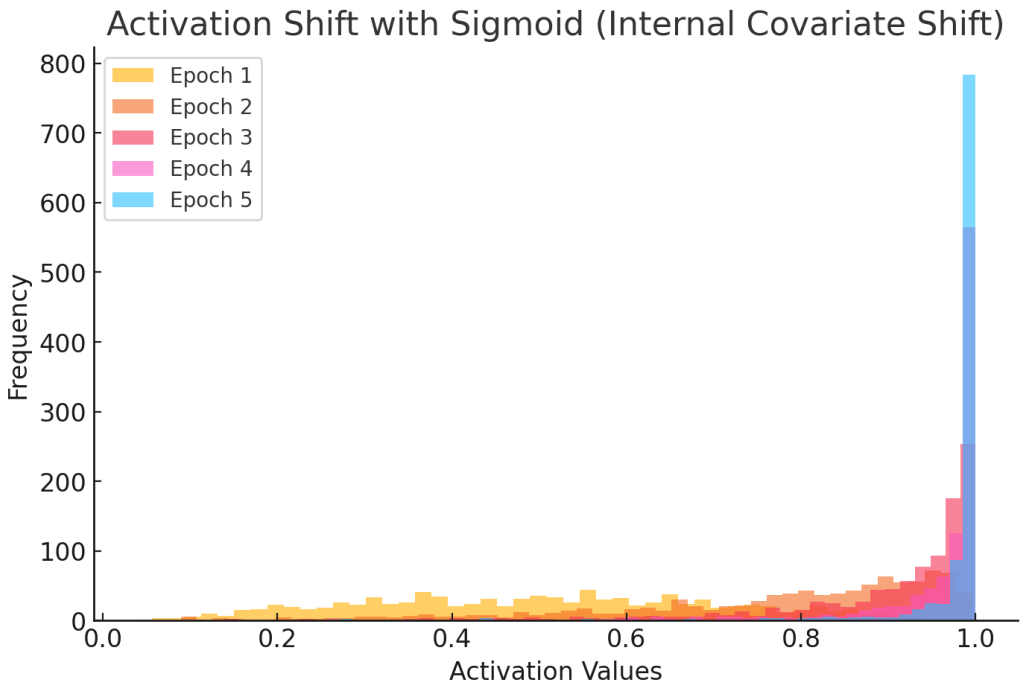
Why ReLU?

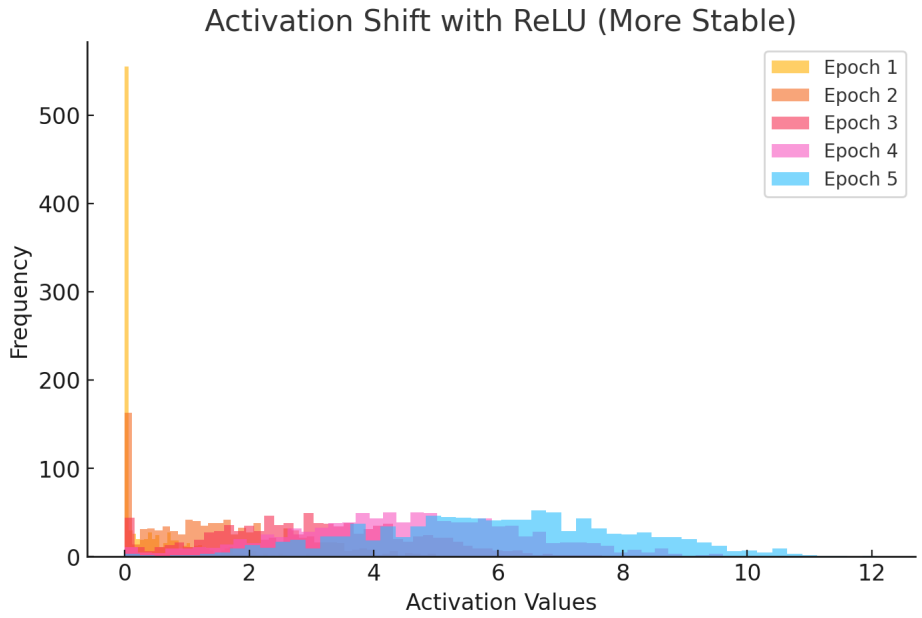
- **Simplicity**: Computationally efficient, allowing models to train faster and requires less computational resources.
- **No Saturation for Positive Inputs**: Unlike sigmoid/tanh, ReLU does not squash large positive values, so gradients remain significant.
- **Simpler Derivative**: The gradient is **1 for positive inputs** and **0 for negative inputs**, preventing small gradient values from accumulating.
- **Prevents Exponential Decay of Gradients**: Unlike sigmoid/tanh, which produce tiny gradients for large inputs, ReLU allows backpropagation to flow efficiently.



No Gradient Saturation

Why ReLU?

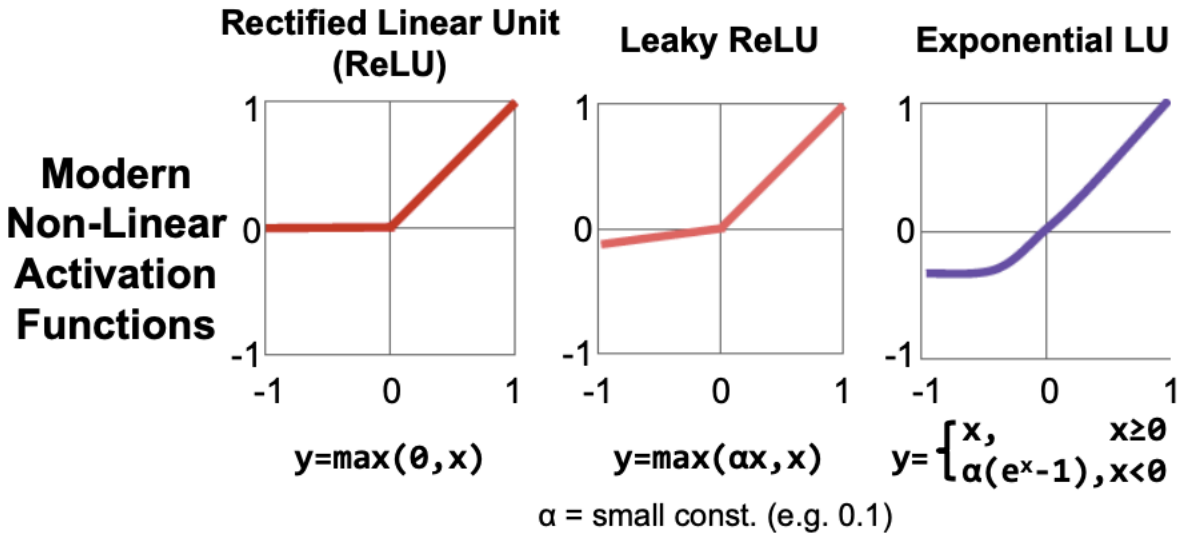


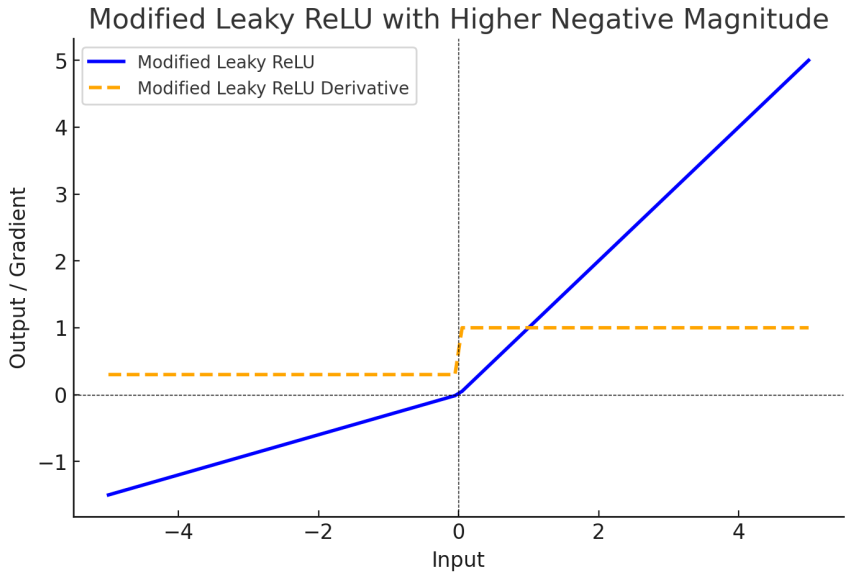


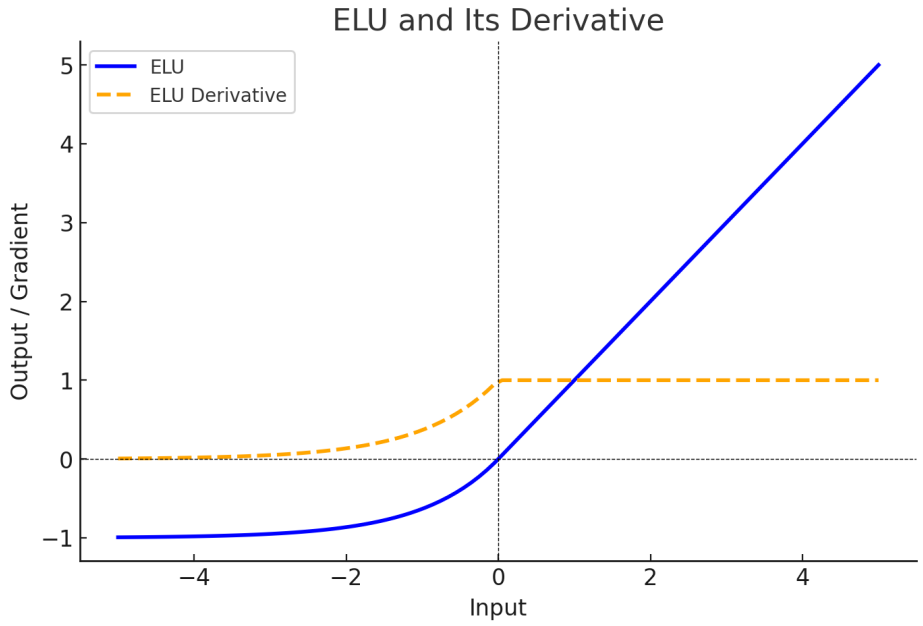
The **activation distributions shift significantly** over epochs.

The **activation distributions remain more stable** across epochs.

Non-Linearity



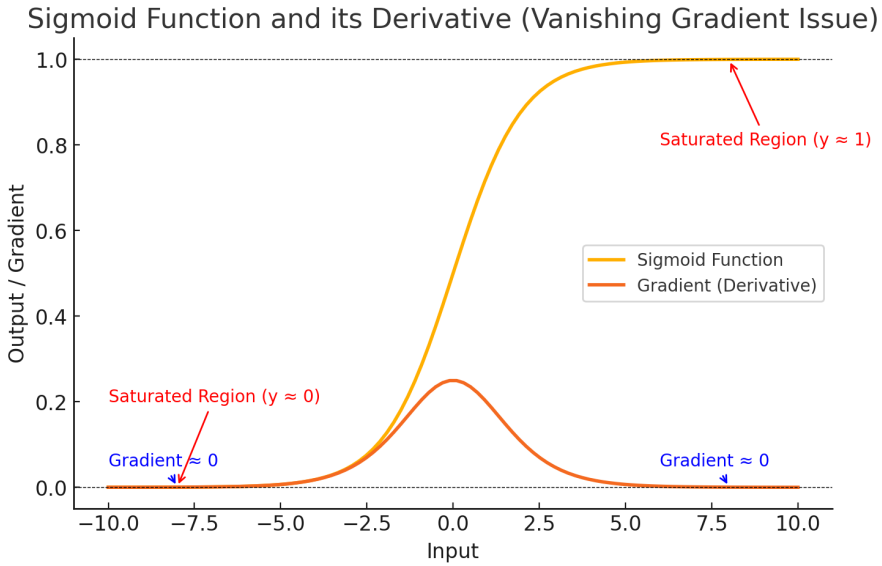




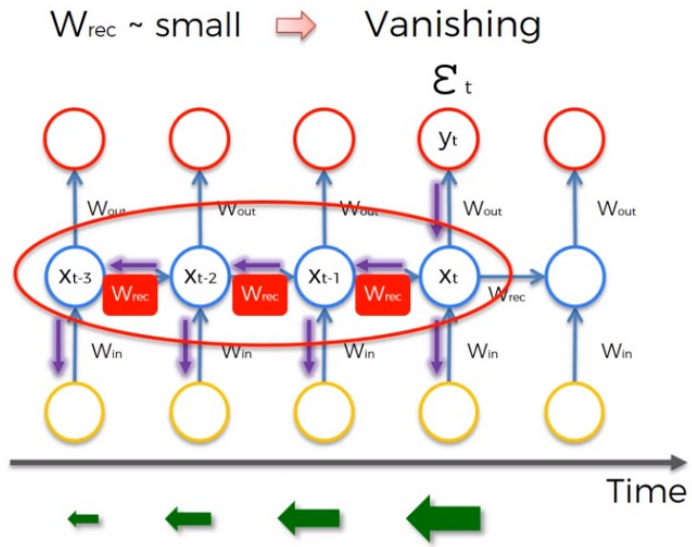
- Leaky ReLU and Exponential LU:
  - **Smooth transition** for x<0 (unlike Leaky ReLU).
  - Addresses the dying ReLU problem
  - Computationally More Expensive

Vanishing Gradient Problem

- **During Backpropagation:**
  - The gradient for each layer is the product of many terms (derivatives of activation functions) from subsequent layers.
- **Exponential Decay:**
  - When these derivatives are less than 1 (as with the sigmoid or tanh functions in their saturated regions), their product can shrink exponentially.
  - This leads to much smaller gradients for earlier layers.
- **Training Impact:**
  - Early layers receive very little gradient information, which means their weights are barely updated.

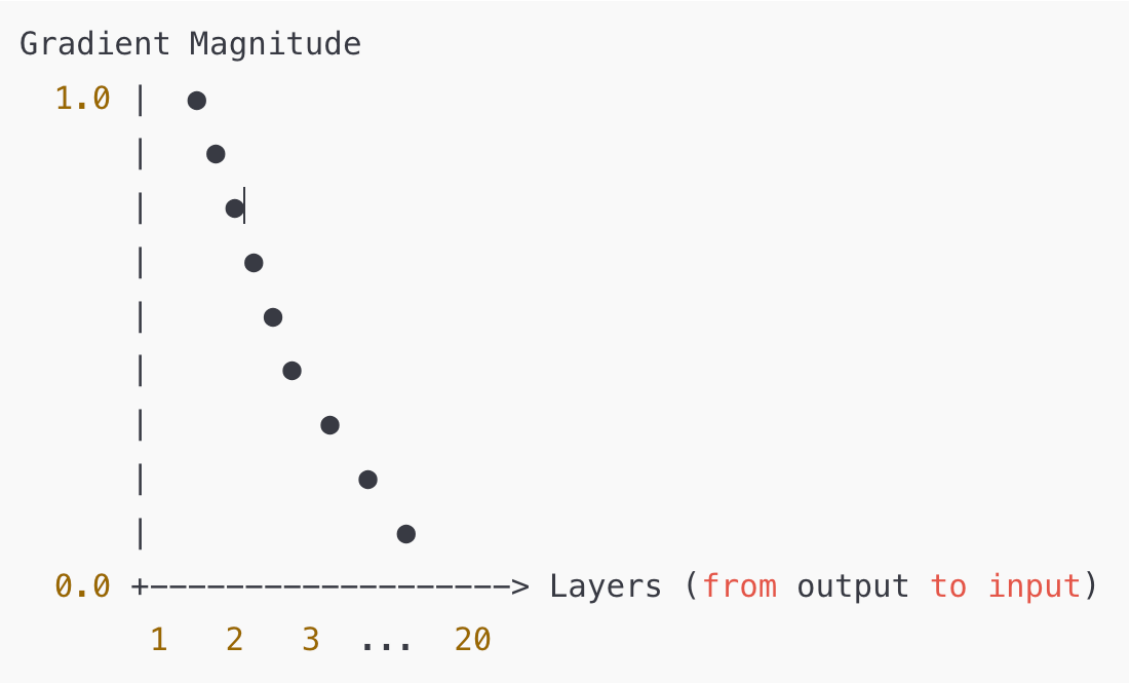


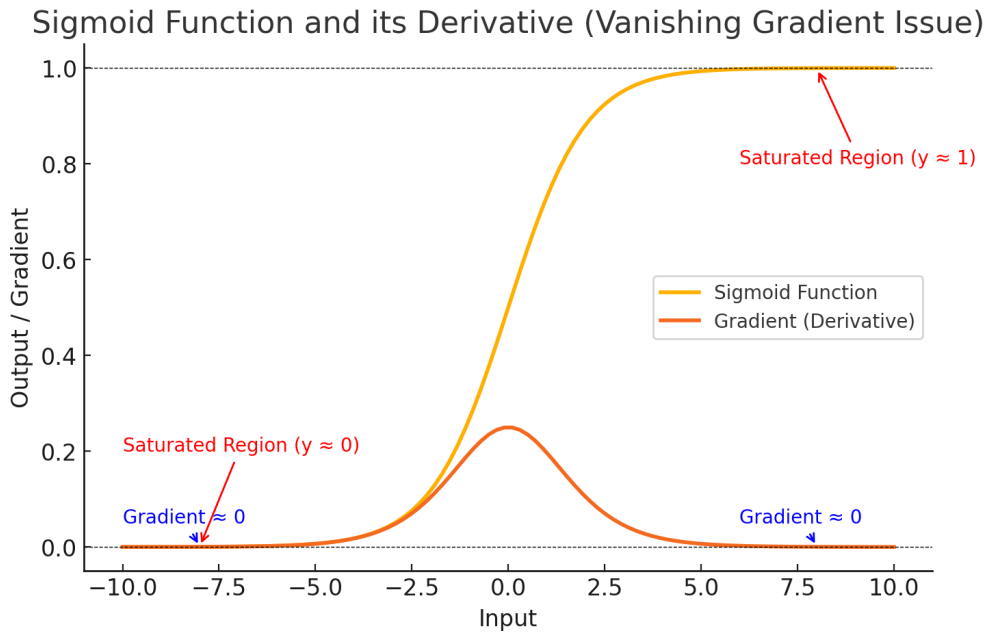
$$W_{new} = W_{old} - \alpha \left(\frac{\partial Loss}{W_{old}} \right)$$



Source: superdatascience.com

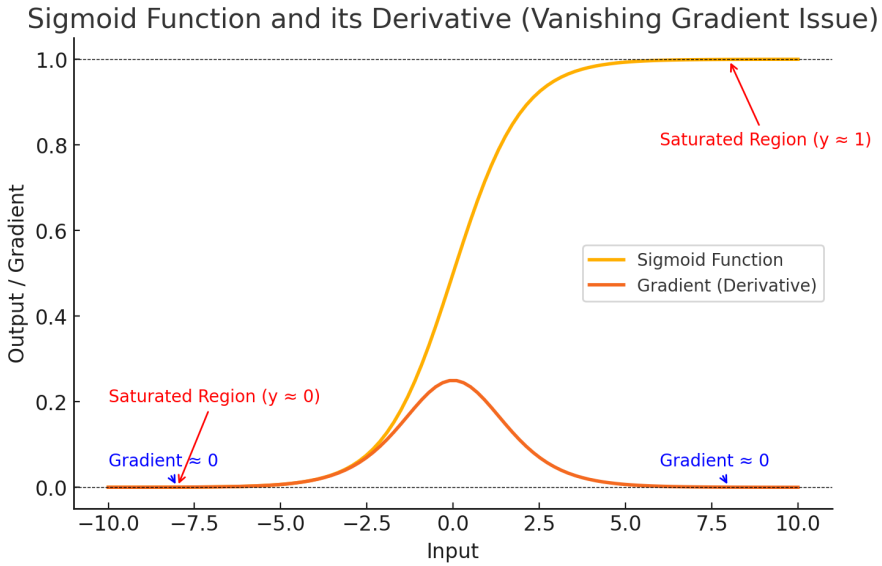
Vanishing Gradient Problem





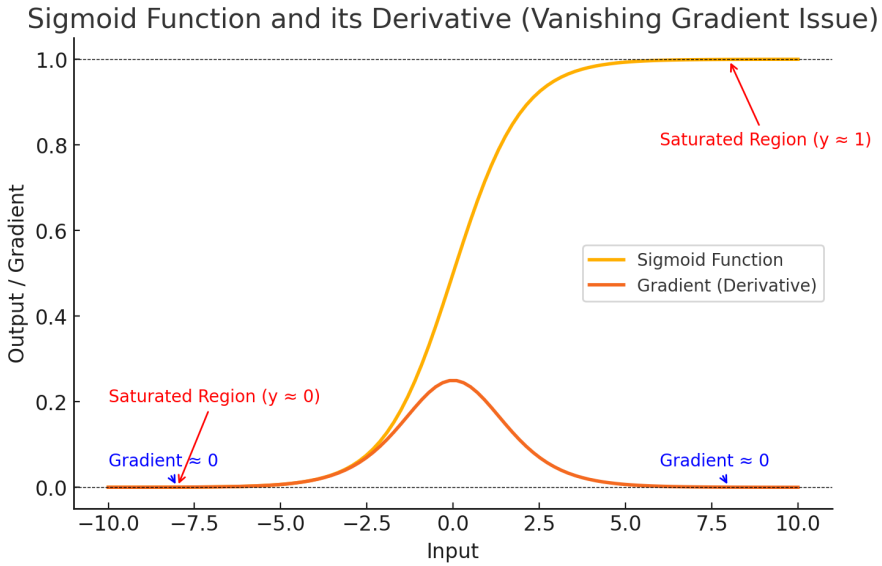
Vanishing Gradient Problem

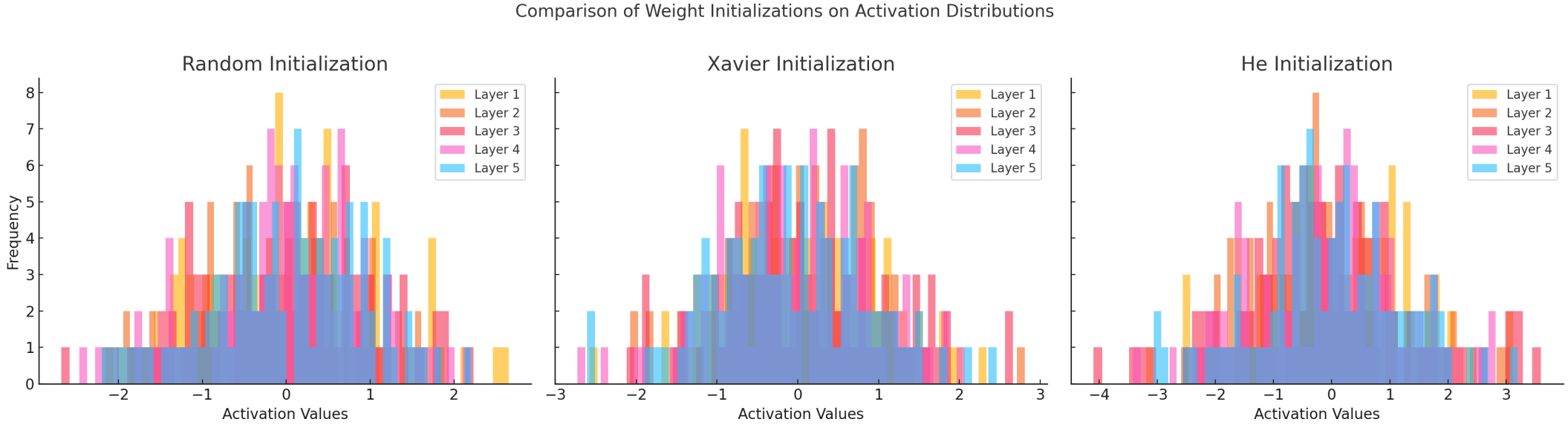
- Solutions?
  - ReLU, Leaky ReLU and variants
  - Others:
    - Weight initialization: adjust initial weights based on layers' sizes



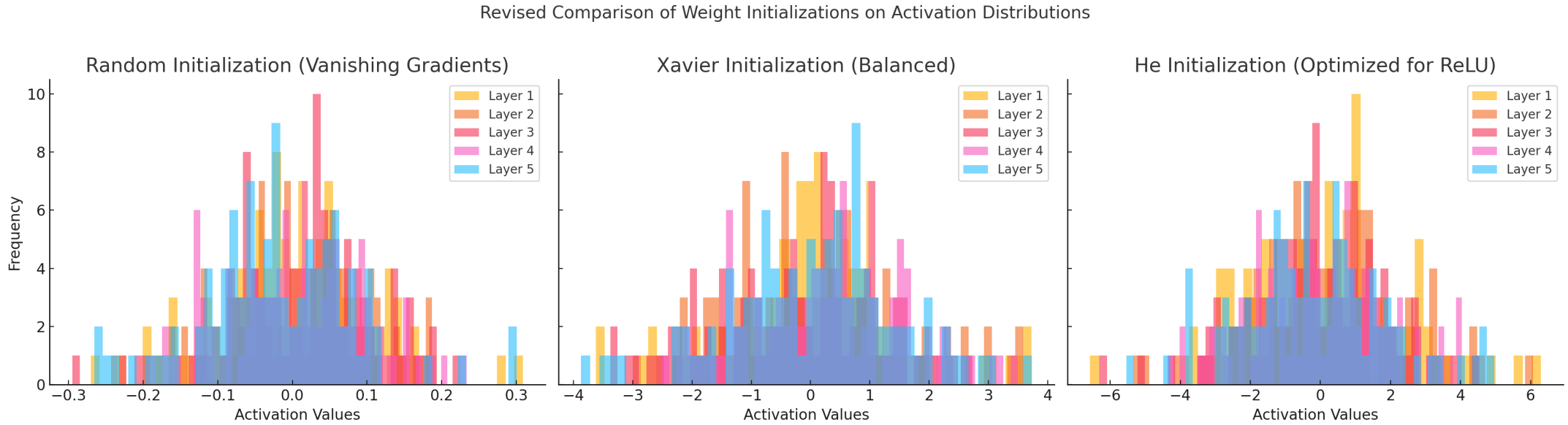
Vanishing Gradient Problem

- Solutions?
  - ReLU, Leaky ReLU and variants
  - Others:
    - Weight initialization: adjust initial weights based on layers' sizes
      - **Too small weights** → Activations shrink → Gradients vanish.
      - **Too large weights** → Activations explode → Unstable training.





Vanishing Gradient Problem



1.Random Initialization (Left) – Vanishing Gradients

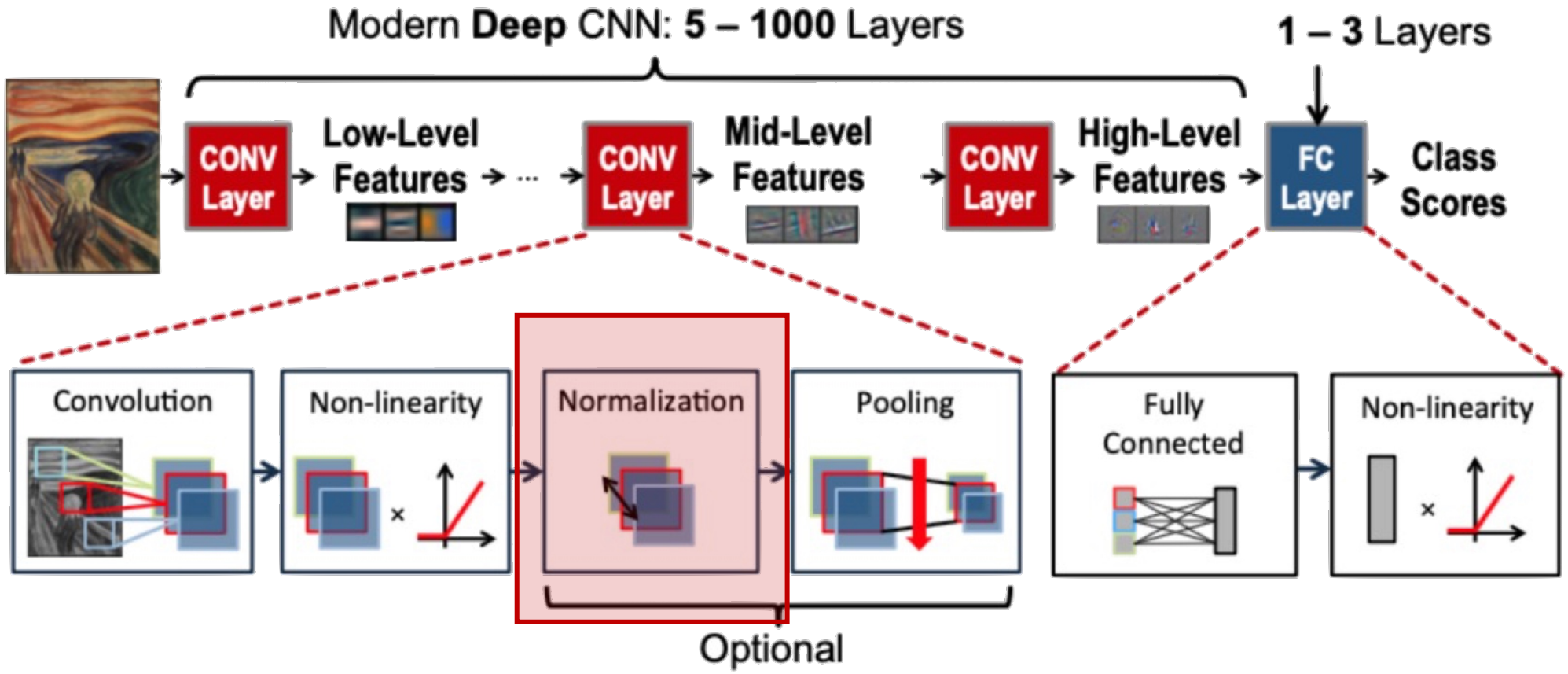
1. Activations **shrink drastically** across layers.
2. Leads to **very small gradients**, slowing down training.
3. Causes **vanishing gradient problem** in deep networks.

2.Xavier Initialization (Middle) – Balanced Activations

1. Keeps activations **within a stable range** across layers.
2. **Prevents vanishing/exploding gradients**, making training smooth.
3. Works well for **Sigmoid/Tanh networks**.

3.He Initialization (Right) – Optimized for ReLU

1. Maintains **larger activations**, ensuring stable variance.
2. Designed specifically for **ReLU networks**.
3. Prevents the **vanishing gradient problem** in deep layers.

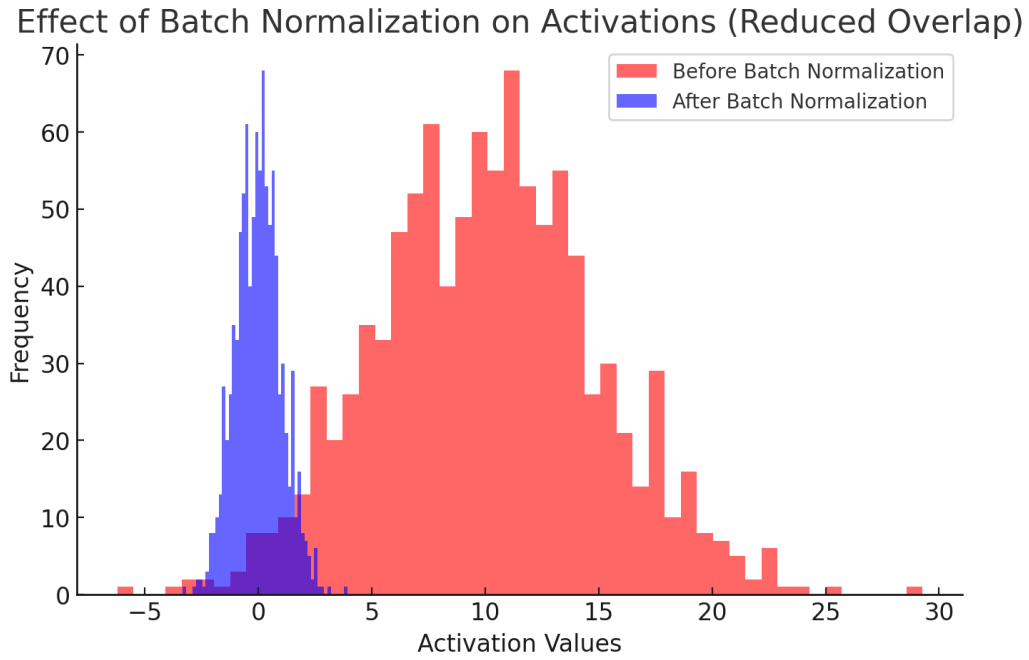


Deep Convolutional Neural Networks

Source: eyeriss.mit.edu

How Batch Normalization Helps with Vanishing Gradient Problem

- **Normalizing Activations**: It ensures that activations remain in a stable range, preventing saturation in nonlinear functions like Sigmoid or Tanh.



- **Before Batch Normalization (Red)**: Activations have a **high mean (~10)** and **wide spread**, leading to possible saturation.
- **After Batch Normalization (Blue)**: Activations are **centered around 0** with a **normalized variance**, keeping them in an optimal range for learning.
- This stabilization **prevents vanishing gradients** and speeds up training.

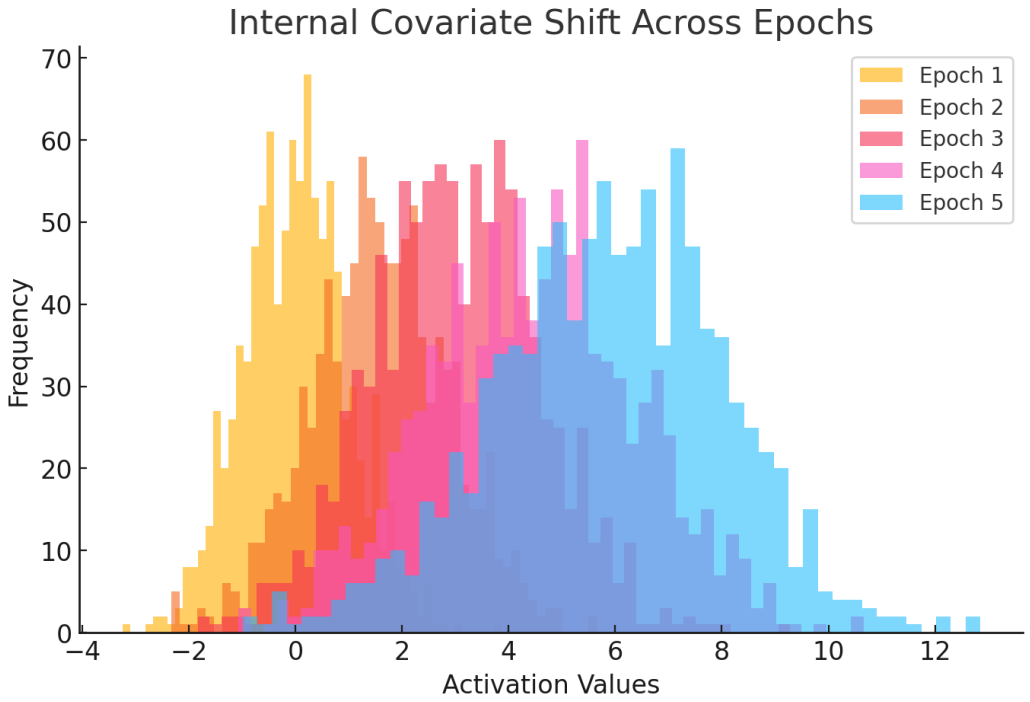
Internal covariate shift

- Changes in model parameters during learning change the distributions of the outputs of each hidden layer.
- This means that later layers need to adapt to these (often noisy) changes during training.

Analogy: You're hitting a **moving** target.
Each time you take a shot, you receive feedback and adjust
but at the same time, the target moves

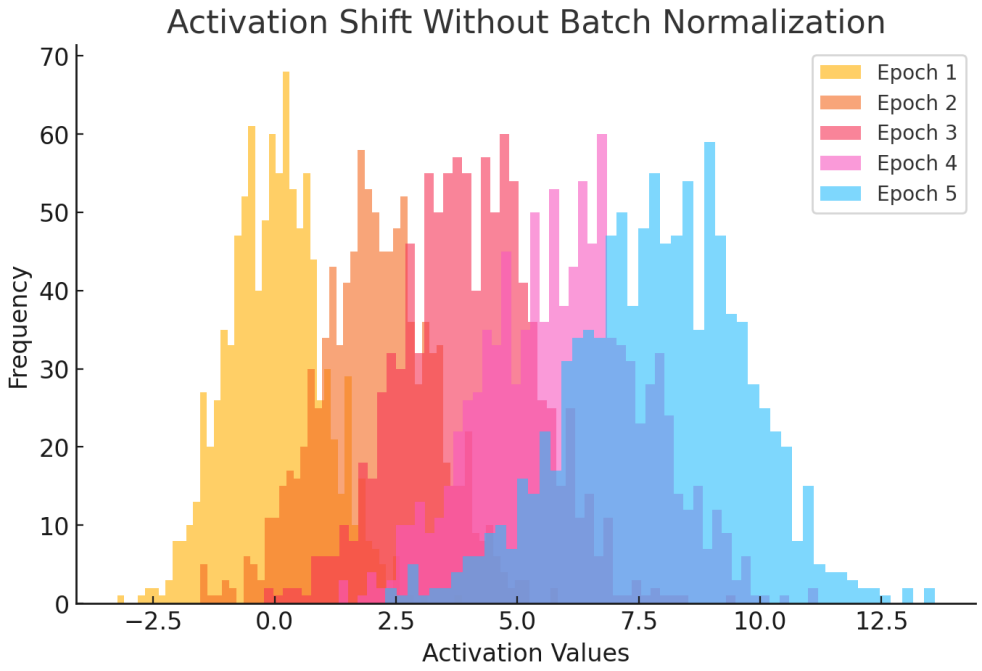
Internal Covariate Shift

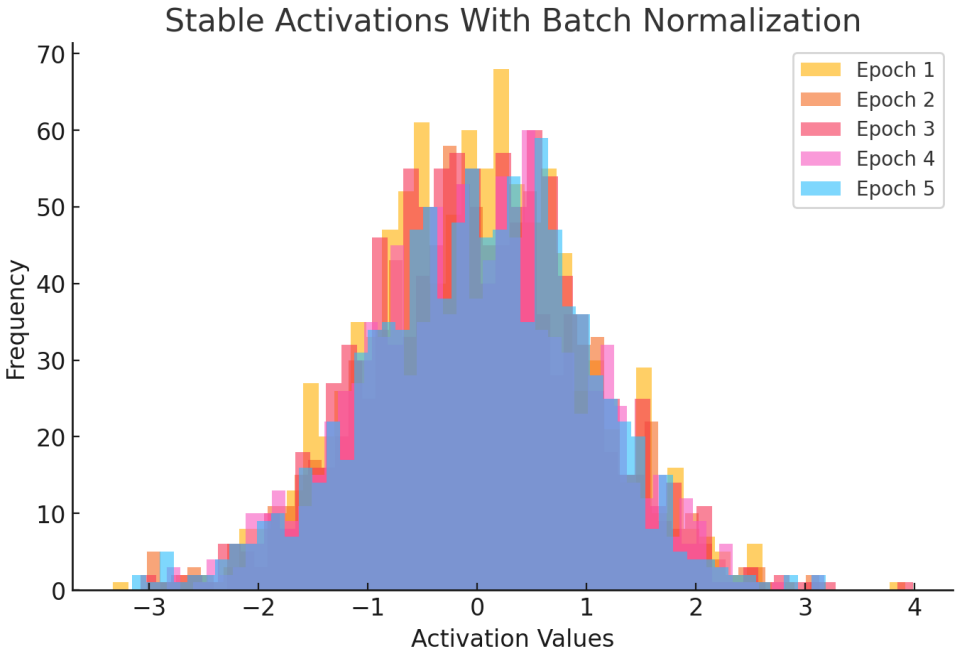
- **Internal Covariate Shift** occurs when the distribution of activations changes across layers and training epochs.
  - This forces deeper layers to constantly adapt to new distributions, slowing down training.
  - **Before Training:** Initial activations are relatively stable.
  - **During Training:** Weights update, causing shifts in activation distributions.
  - **Problem:** Large shifts disrupt learning, requiring lower learning rates and careful weight initialization.



How Batch Normalization Helps with Vanishing Gradient Problem

- **Reducing Internal Covariate Shift**: Keeps layer inputs with consistent distribution across training steps.



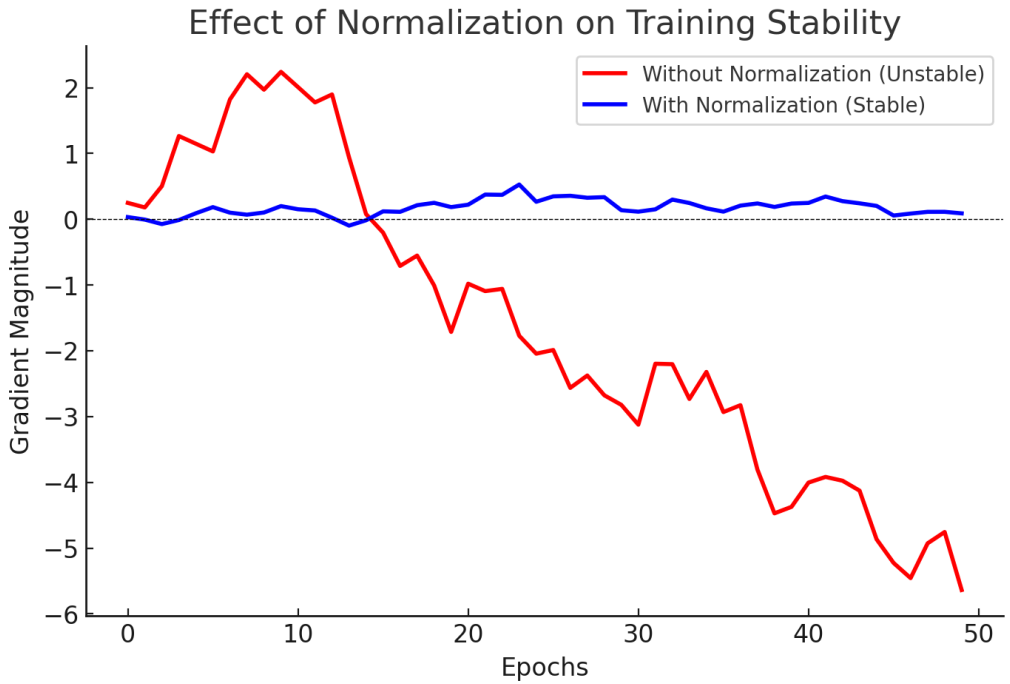


With normalization, the activations remain **centered and normalized**, maintaining a stable distribution across epochs.

This stability helps **reduce internal covariate shift**, leading to **faster and more efficient training**.

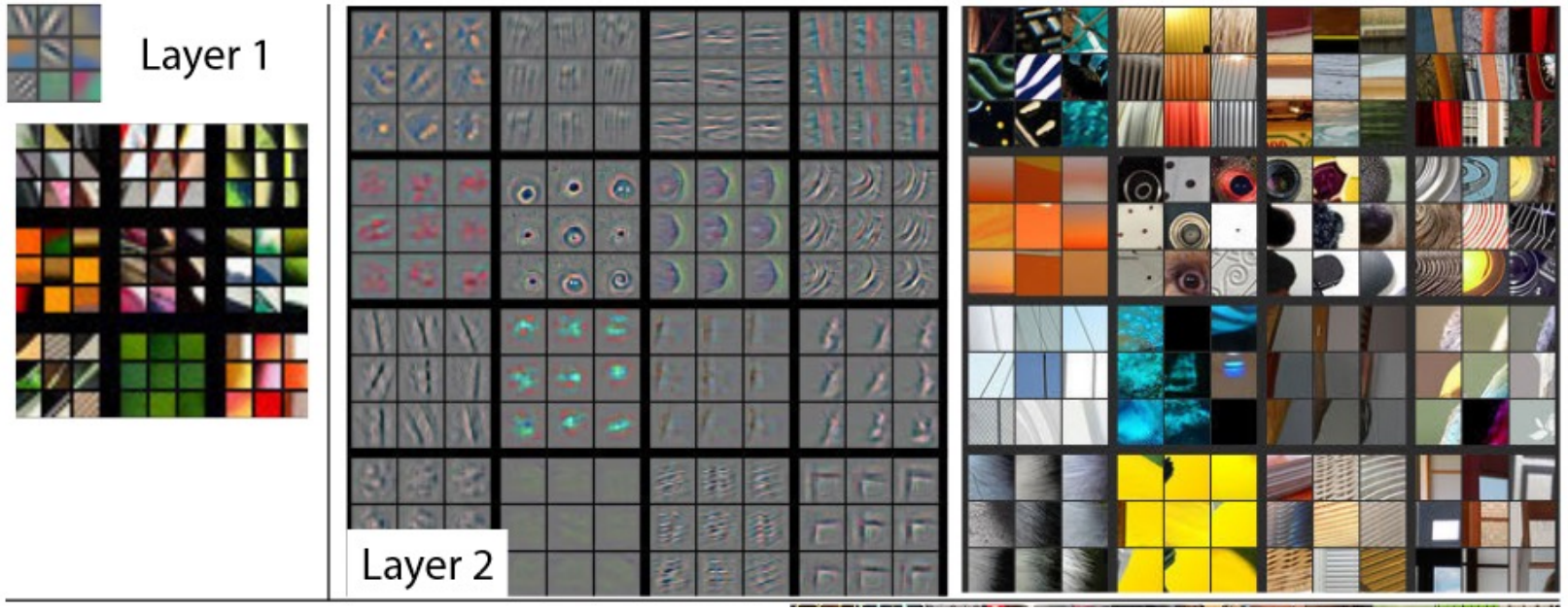
How Batch Normalization Helps with Vanishing Gradient Problem

- **Faster Convergence**: Helps gradients stay within a useful range, improving training efficiency.
  - **Faster Convergence:** Larger updates lead to quicker optimization.
  - **Better Gradient Flow:** Normalization keeps gradients in a useful range.
  - **Less Sensitivity to Weight Initialization:** Avoids issues caused by poor weight scaling.



- **Red Line (Without Normalization)**:
  - Large fluctuations indicate **unstable gradients**.
  - **Exploding gradients** cause sharp increases.
  - **Vanishing gradients** cause near-zero updates, slowing learning.
- **Blue Line (With Normalization)**:
  - **Controlled, stable gradient updates**.
  - Prevents drastic changes, allowing **higher learning rates** without instability.
  - Leads to **smoother convergence**.

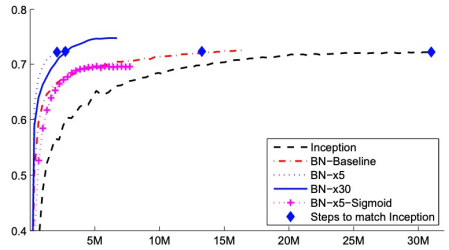
Internal covariate shift



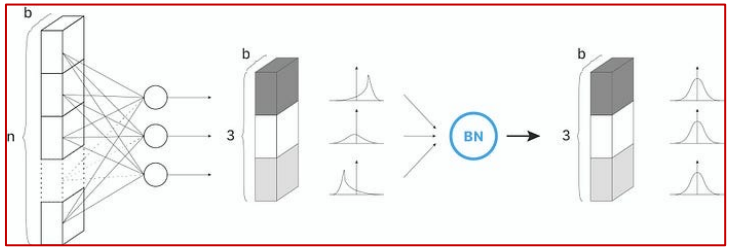
Example of the problem in CNNs:
suppose early in training, the first layer learns to recognize edges in images, and the second layer learns to combine those edges into simple shapes.

If the first layer's weights change significantly (due to large gradients from the loss function), the second layer may no longer receive the edge information in the same way it did before, and the shapes it has learned to recognize may become distorted or unrecognizable.

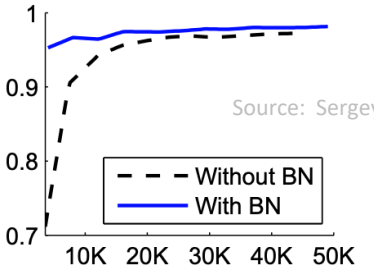
Normalization



Accuracy on the ImageNet (2012) validation set, vs the number of training steps. Note that with batch normalization (BN), we converge faster.



Source: Johann Huber



Source: Sergey Ioffe et al.

Accuracy of the MNIST network with and without Batch Normalization, vs. the number of training steps.

Mean of all x_i

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Std. dev.

Small constant added for stability

Normalize the inputs so that mean = 0 and std. dev = 1

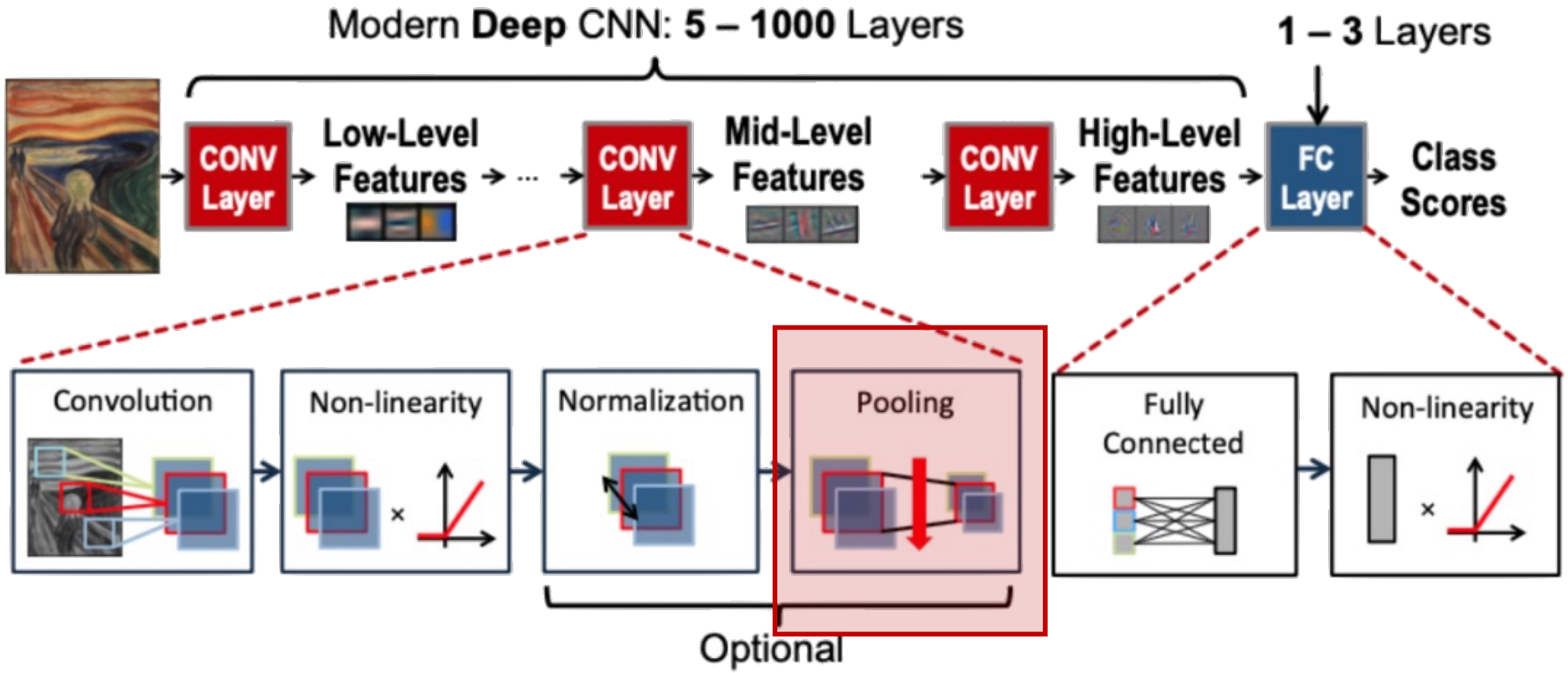
$$y_i = \gamma \hat{x}_i + \beta$$

Scale and shift the normalized values with learnable parameters

Learnable

- **Addresses Internal covariate shift**
- **Higher** accuracy and **faster** training

Scale and shift introduce some level of covariate shift, but it is a controlled and learnable shift that allows the model to optimize its internal representations



Deep Convolutional Neural Networks

Source: eyeriss.mit.edu

Pooling

- Common Types:
  - **Max** Pooling **(Feature Strengthening)**
  - **Average** Pooling
- Purpose:
  - Reduce Overfitting:
    - By reducing spatial dimensions, pooling helps in achieving a form of invariance to minor changes, translations, and rotations in the image.
  - Computation Efficiency
  - Hierarchy of Features: Allows CNNs to recognize more complex patterns in subsequent layers.

$$O_{n,m,p,q} = \text{Max}(I_{n,c,Up+r,Uq+s})$$

10	20	50	60
30	40	70	80
5	15	55	65
35	45	75	85

40	80
45	85

Max Pooling

$$O_{n,m,p,q} = \frac{\sum_{r,s} I_{n,c,Up+r,Uq+s}}{U^2}$$

25	65
20	60

Average Pooling

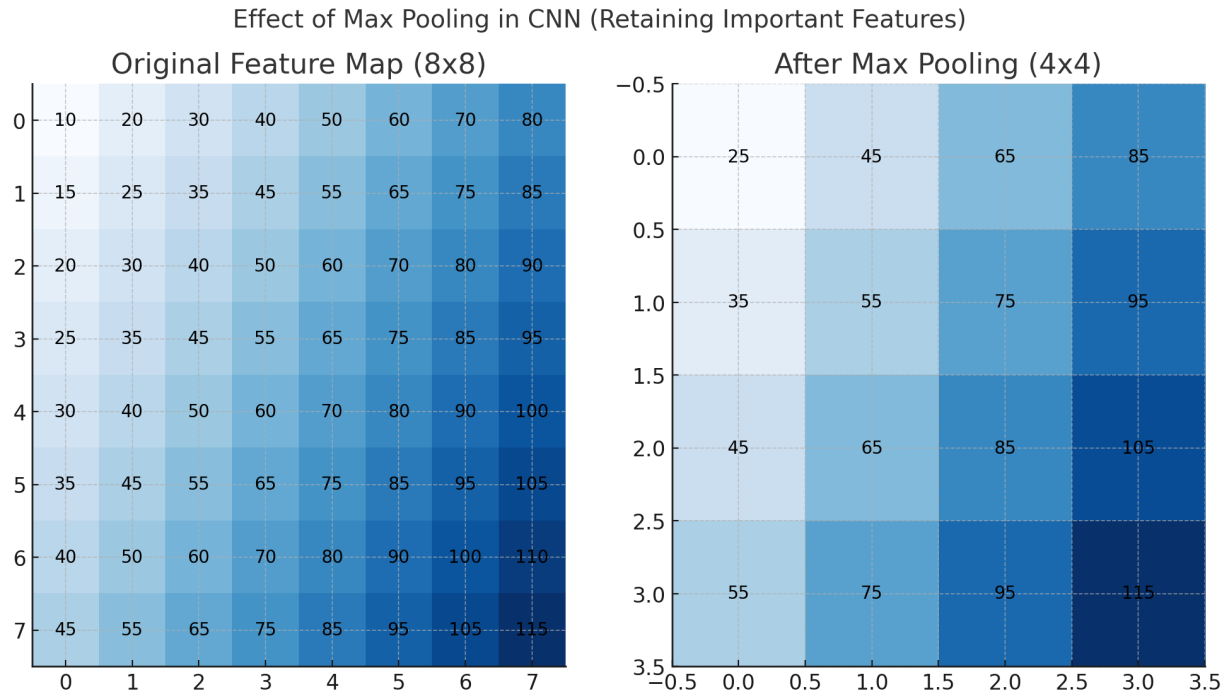
Naïve 6-layer for-loop max-pooling implementation:

```
for n in [0..N):
    for m in [0..M):
        for q in [0..Q):
            for p in [0..P):

                max = -Inf
                for r in [0..R):
                    for s in [0..S):
                        if I[n][m][Up+r][Uq+s] > max:
                            max = I[n][m][Up+r][Uq+s];

                O[n][m][p][q] = max
```

for each pooled value

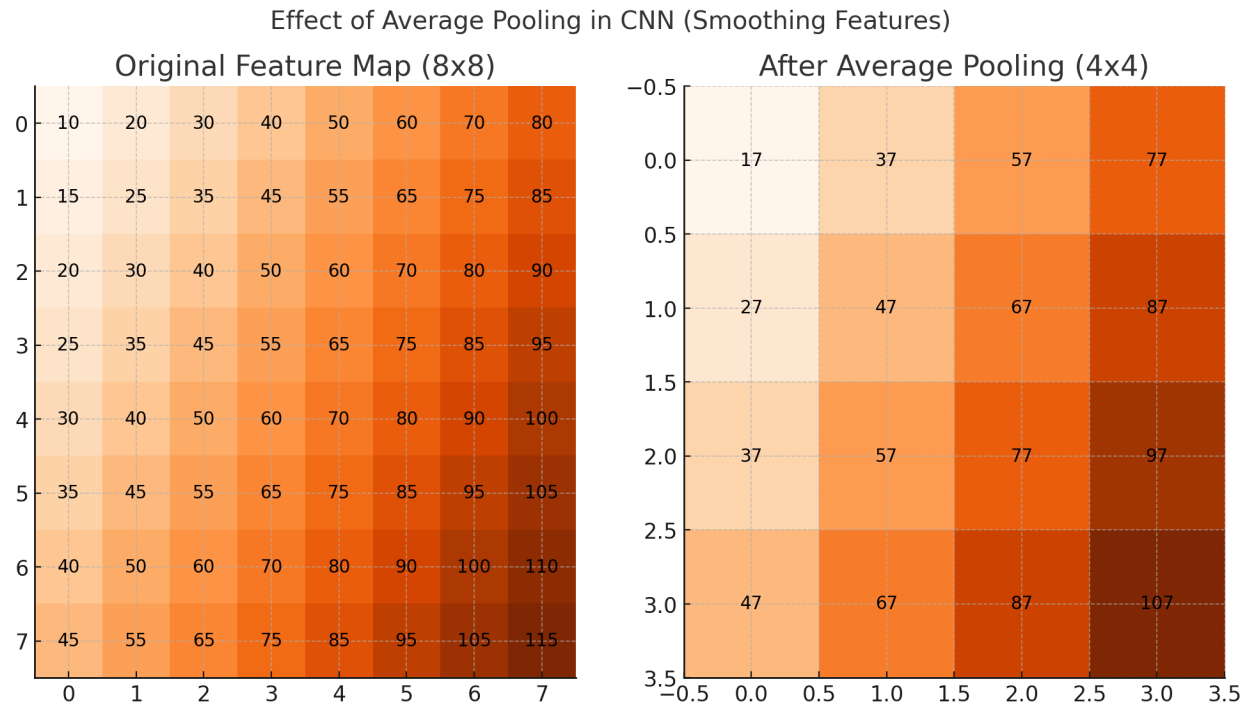
find the max in a window

Max Pooling

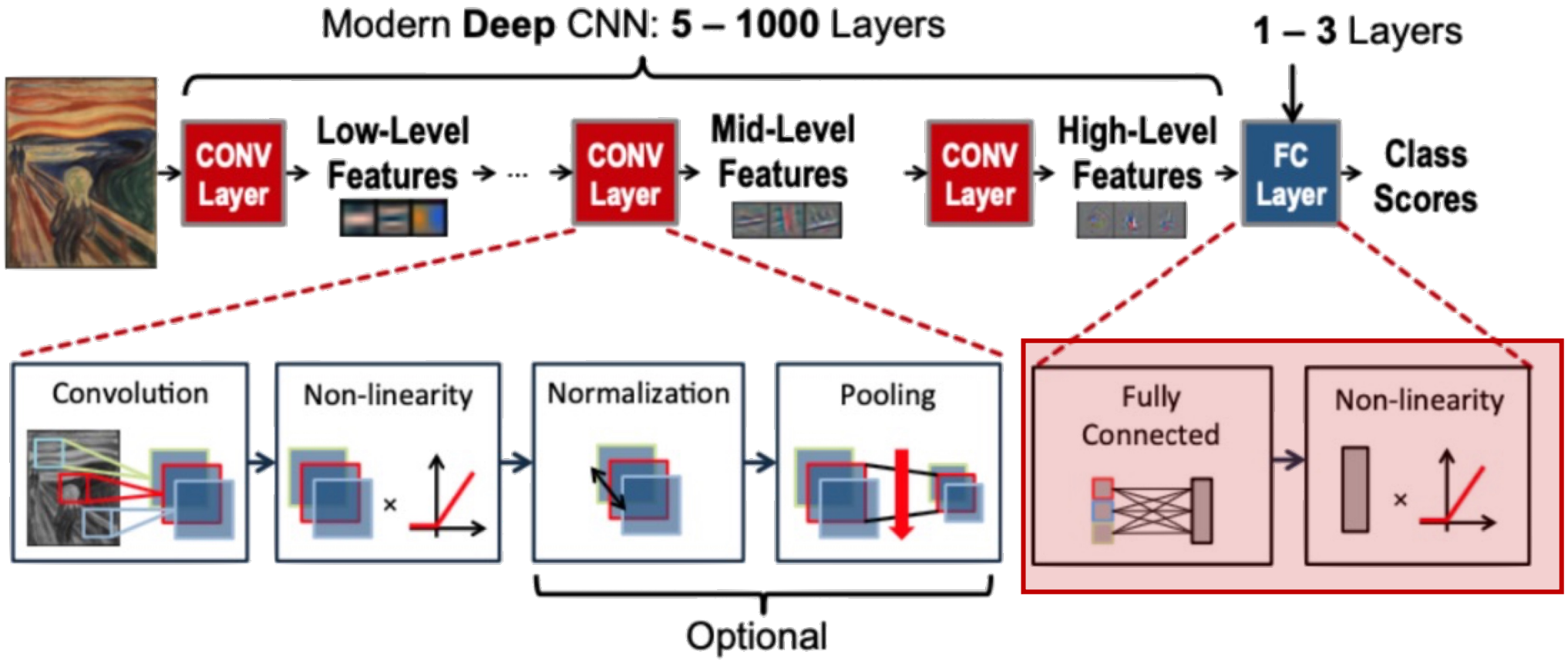


- Each value represents the **maximum** from a **2×2 region** in the original feature map.
- Retains the **most important/highest** activations while reducing size.

Average Pooling



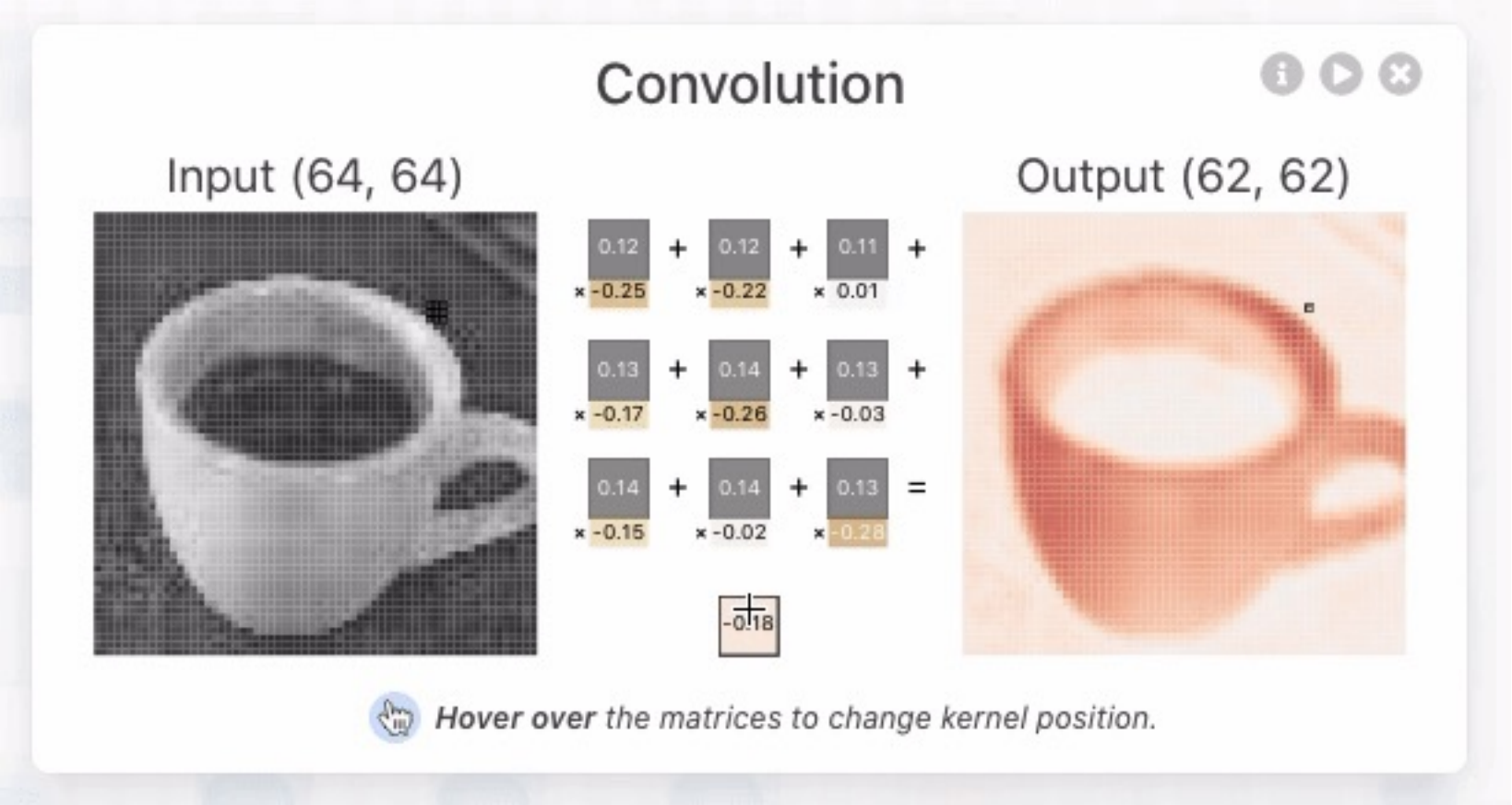
- Each value represents the **average** from a **2×2 region** in the original feature map.
- Preserves overall **patterns** rather than just the strongest activations.



Deep Convolutional Neural Networks

Source: eyeriss.mit.edu

CNN Explainer Demo



https://poloclub.github.io/cnn-explainer/

CNN Explainer uses *TensorFlow.js*, an in-browser GPU-accelerated deep learning library to load the pretrained model for visualization.