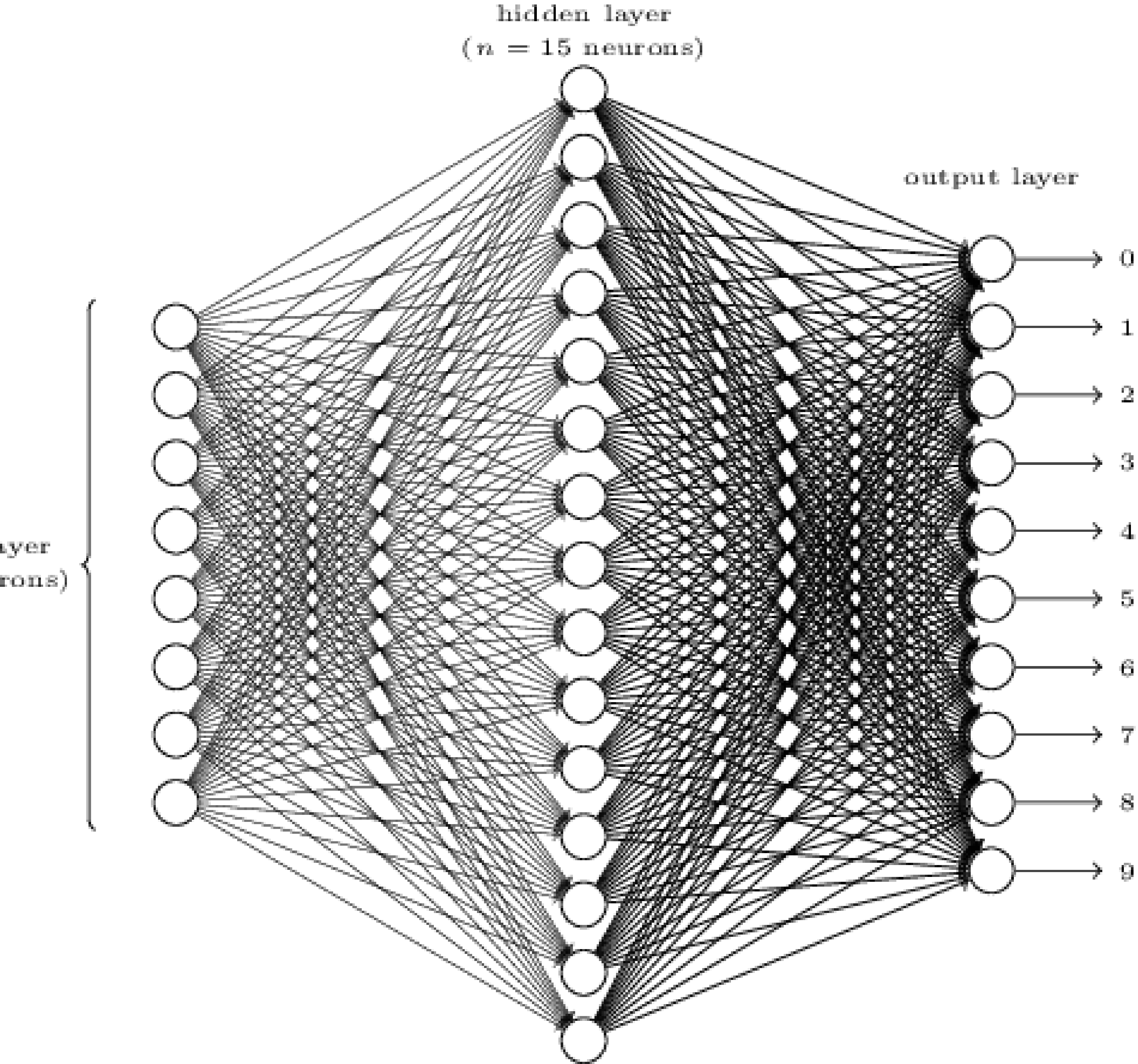
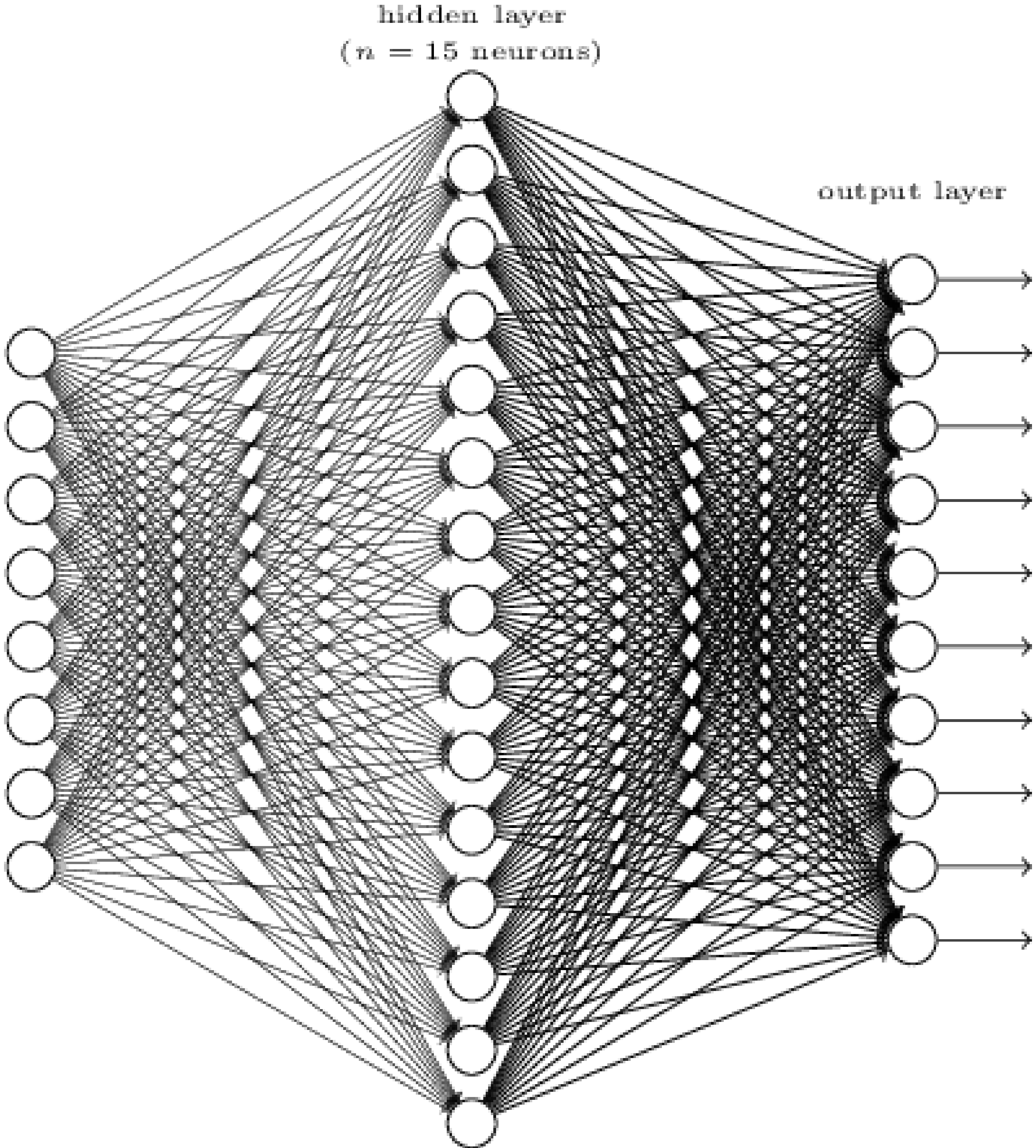
Convolutional neural networks

Nail Ibrahimli



Limitations of MLP network architecture



Limitations of MLP network architecture

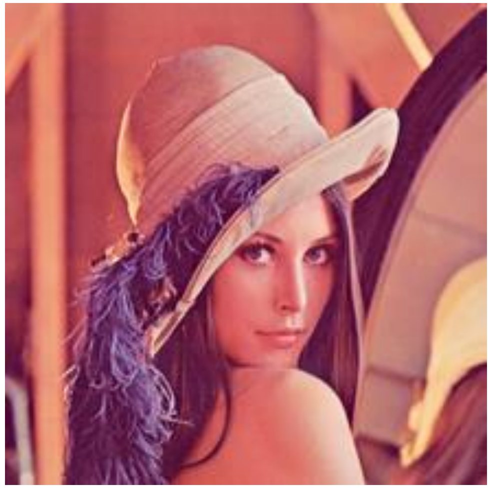
- **High Dimensionality & Loss of Spatial Information**
- When using MNIST, each 28×28 image is flattened into a 784-element vector.
- This flattening ignores the 2D structure of images, making it harder for the network to capture spatial relationships.
- **Large Number of Parameters**
- Fully connected layers in an MLP lead to an explosion in parameters as input size increases.
- More parameters increase computational cost and risk of overfitting.
- **Inefficient for Local Feature Extraction**
- MLPs do not inherently learn localized features (e.g., edges, textures).
- They struggle to capture patterns that are position invariant, unlike convolutional layers.
- **Scalability Issues**
- As the complexity or resolution of images grows, MLPs become less practical compared to convolutional architectures.

Properties of Images: Image Locality

Properties of Images: Image Locality

- **Ordered Pixels:**

Pixels are arranged in a specific order, forming a grid.



Properties of Images: Image Locality

- **Ordered Pixels:**

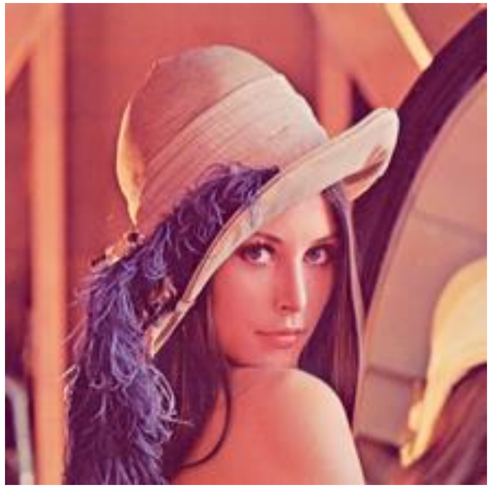
Pixels are arranged in a specific order, forming a grid.

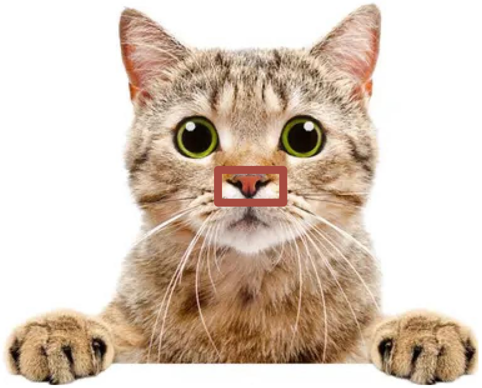
- **Spatial Correlation:**

Neighboring pixels tend to be related, capturing local features.

- **Exploitable Structure:**

This order allows models like CNNs to leverage local patterns effectively.

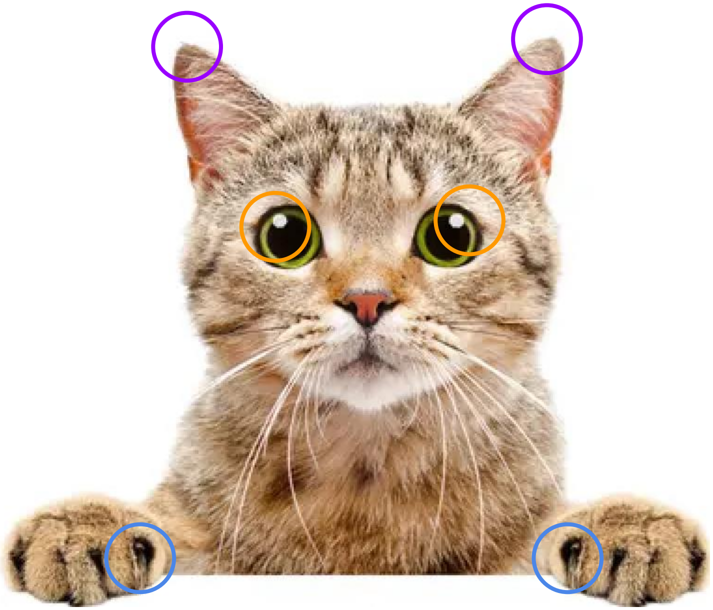




Properties of Images: Image Stationarity

Properties of Images: Image Stationarity

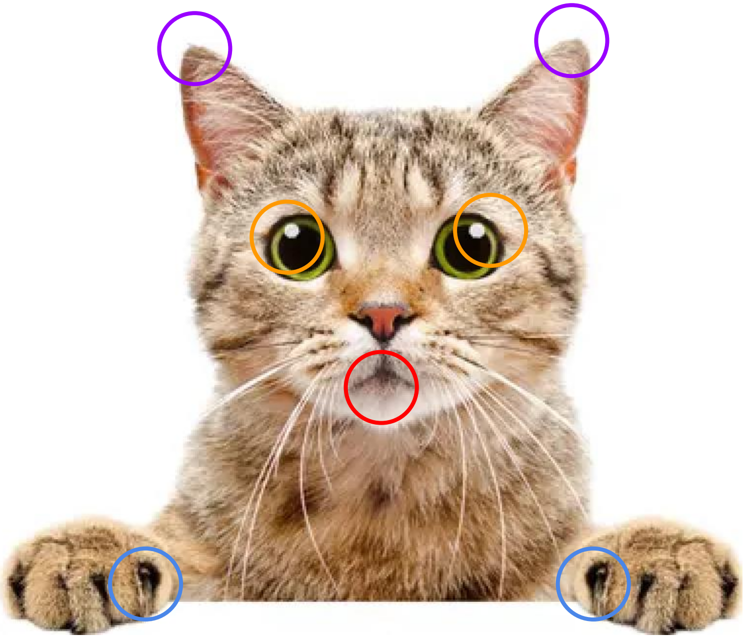
- **Consistent Statistical Properties:**
  The distribution of pixel values remains relatively consistent across the image.
- **Repeated Patterns:**
  Similar features (e.g., edges, textures) can occur anywhere in the image.
- **Enables Weight Sharing:**
  Supports convolution operations where the same filters can detect patterns regardless of their location.

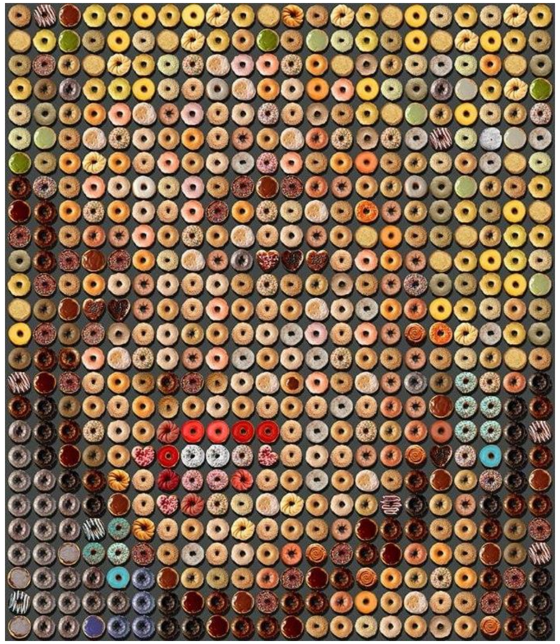


Properties of Images: Image Compositionality

Properties of Images: Image Compositionality

- **Hierarchical Structure:**
Images are built from simple elements (e.g., edges, corners) that combine to form more complex structures.
- **Layered Feature Composition:**
Basic patterns merge into higher-level features, enabling robust recognition of complex objects.
- **Efficient Representation:**
Leveraging compositionality helps models learn and generalize from simpler, reusable components.





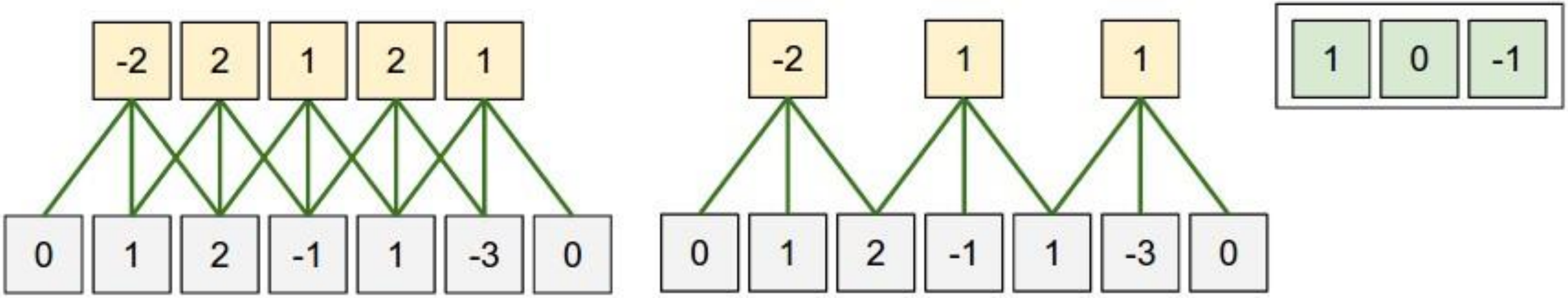
Properties of Images

Locality:
Pixels are arranged in a structured grid; local groups contain correlated information.

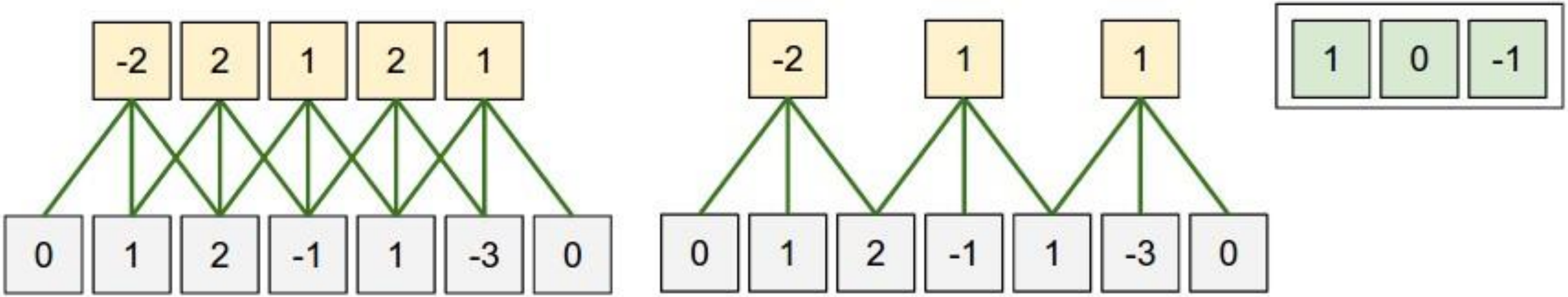
Stationarity:
Statistical properties are consistent across the image; similar patterns (e.g., edges) appear everywhere, allowing effective weight sharing.

Compositionality:
Simple elements combine hierarchically to form complex features, enabling efficient and robust representations.

Introduction to 1-D Convolution



Introduction to 1-D Convolution



Sliding Window Operation:
A filter (kernel) slides along the input sequence, computing a weighted sum at each position.

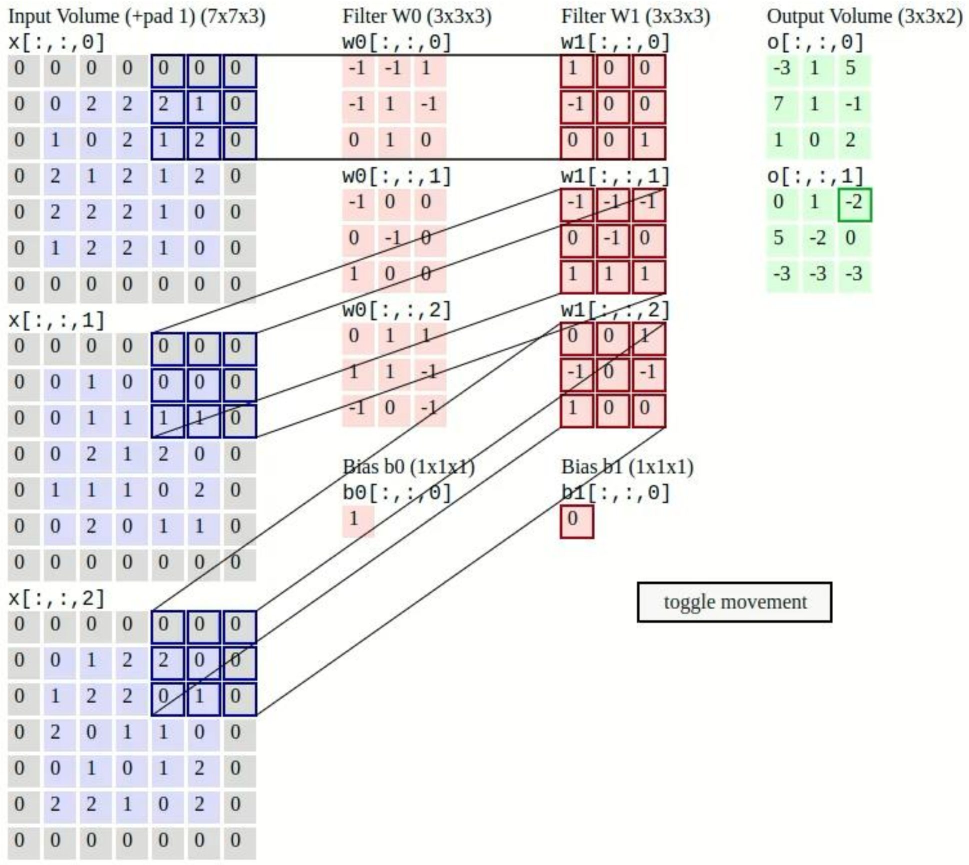
Local Feature Extraction:
Captures local patterns from adjacent elements in the sequence.

Translation Equivariance:
The same filter is applied across the entire input, ensuring features are detected regardless of their position.

Efficiency:
Reduces parameters by sharing weights, making it computationally efficient.

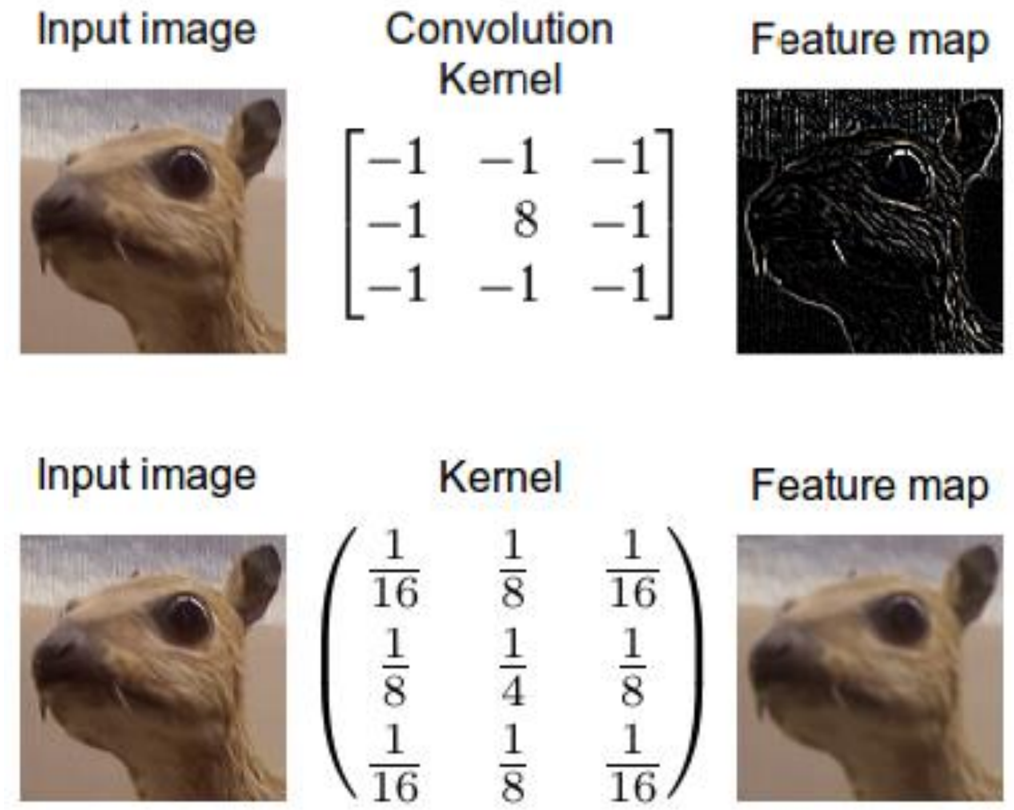
Image Convolution (2D Convolution)

- **Sliding Window Operation:**
A small filter (kernel) moves across the image, computing weighted sums of pixel values.
- **Local Feature Detection:**
Captures edges, textures, and patterns by emphasizing spatial relationships.
- **Weight Sharing & Efficiency:**
The same filter is applied across the image, reducing parameters and improving generalization.

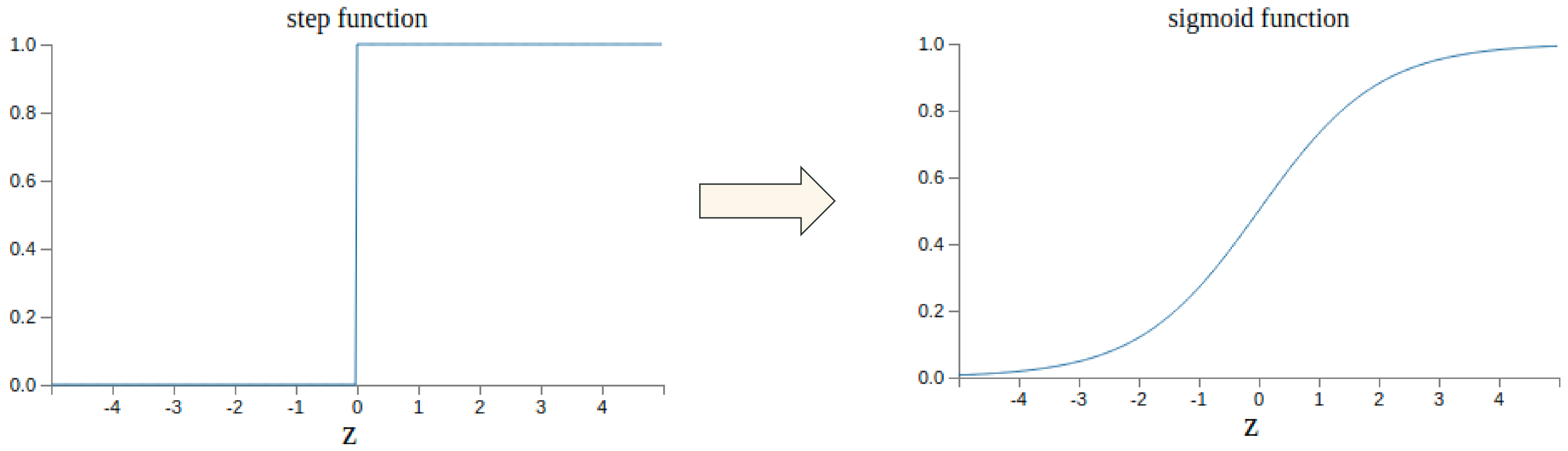


2D Convolution: Edge Detection & Smoothing

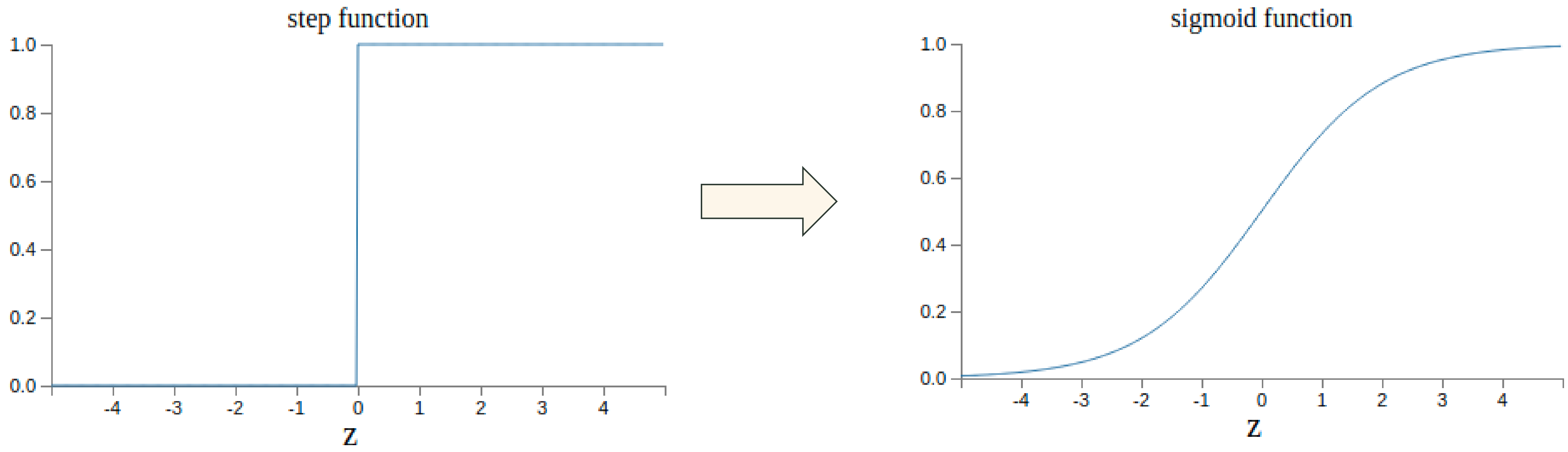
- **Edge Detection (Laplacian Kernel):**
  - Enhances edges by highlighting regions with rapid intensity changes.
  - Captures important structural details in the image.
- **Smoothing (Gaussian Kernel):**
  - Blurs the image by averaging neighboring pixels.
  - Reduces noise while preserving general structure.



Sigmoid activation



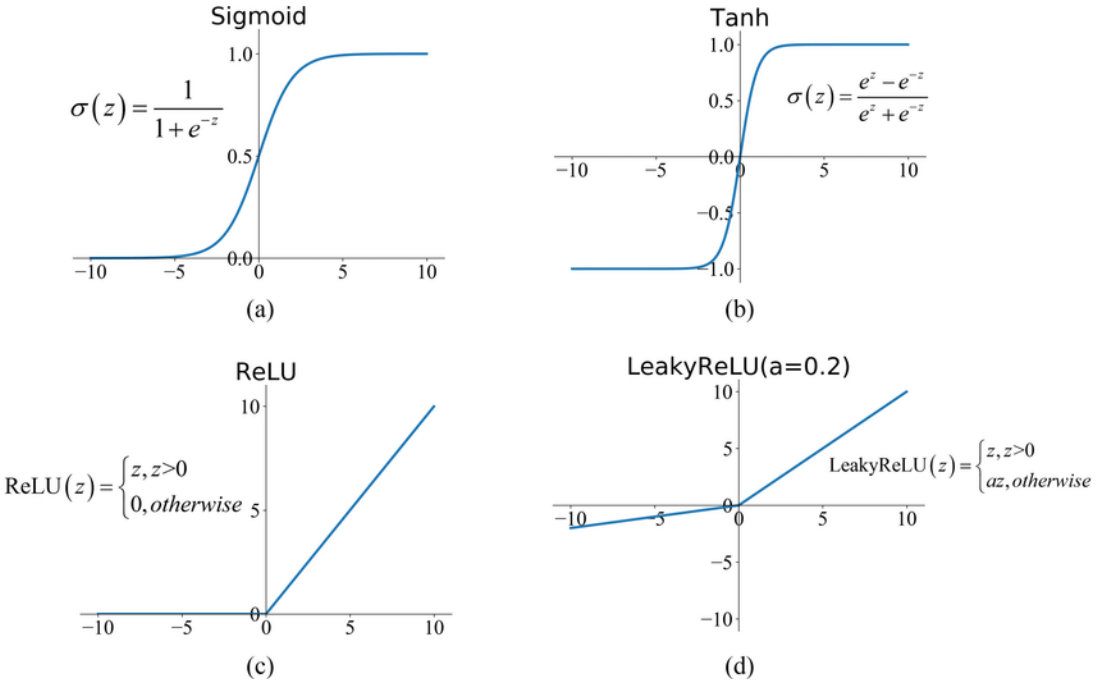
Sigmoid activation



Vanishing Gradient Problem:
•Gradients become very small for extreme values, slowing down learning in deep networks.
Non-Zero Mean Output:
•Outputs range from (0,1), causing imbalanced weight updates and inefficient learning.

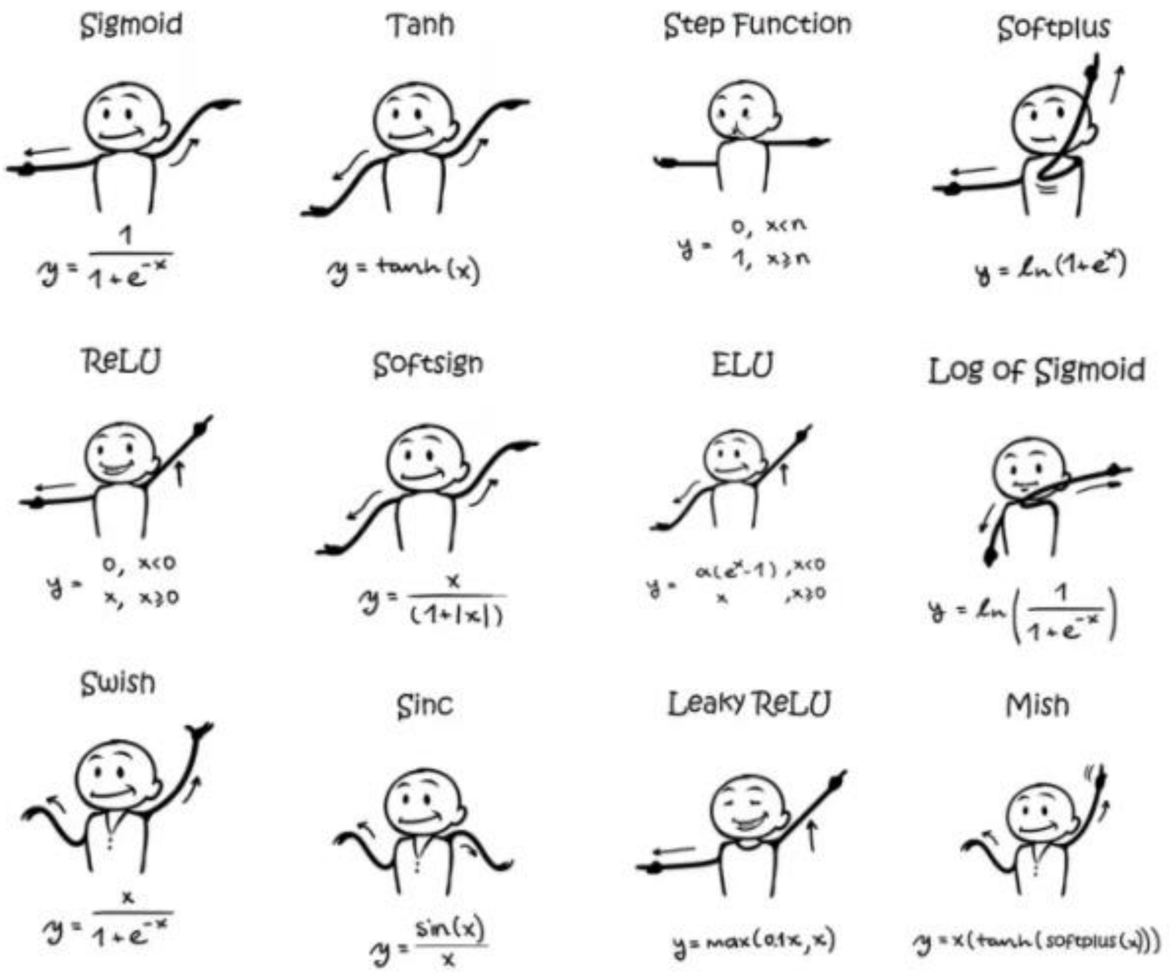
Activation Functions in Neural Networks

- **Sigmoid:**
  - Outputs in (0,1), prone to vanishing gradients and slow learning.
- **Tanh:**
  - Outputs in (-1,1), zero-centered but still suffers from vanishing gradients.
- **ReLU (Rectified Linear Unit):**
  - Outputs max(0, x), mitigates vanishing gradients but can have dead neurons (dying ReLU problem).
- **Leaky ReLU & Variants:**
  - Allows small negative values to prevent dead neurons.
- **Softmax (for Classification):**
  - Converts logits into probabilities, used in the final layer for multi-class classification.

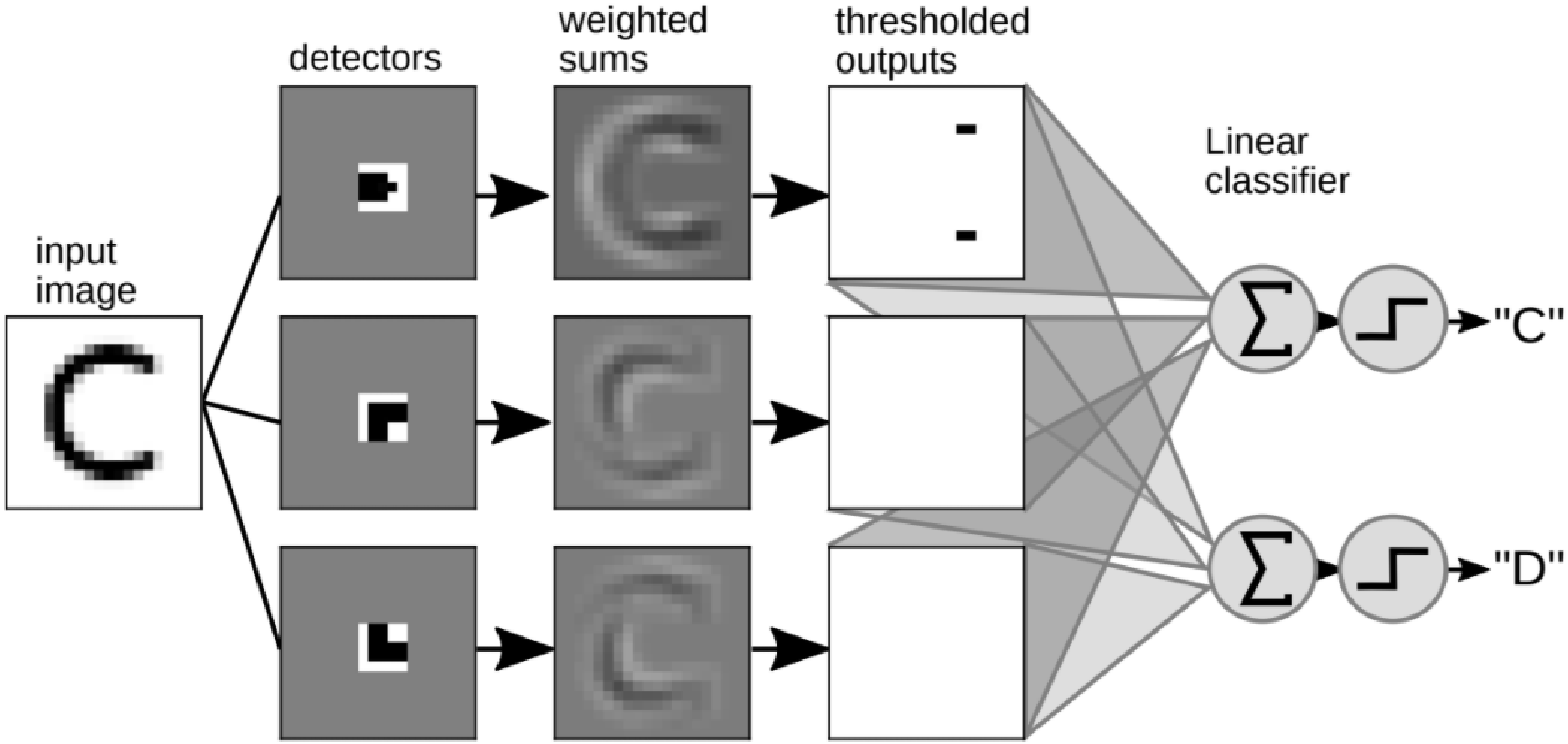


Activations

[PyTorch activation functions]

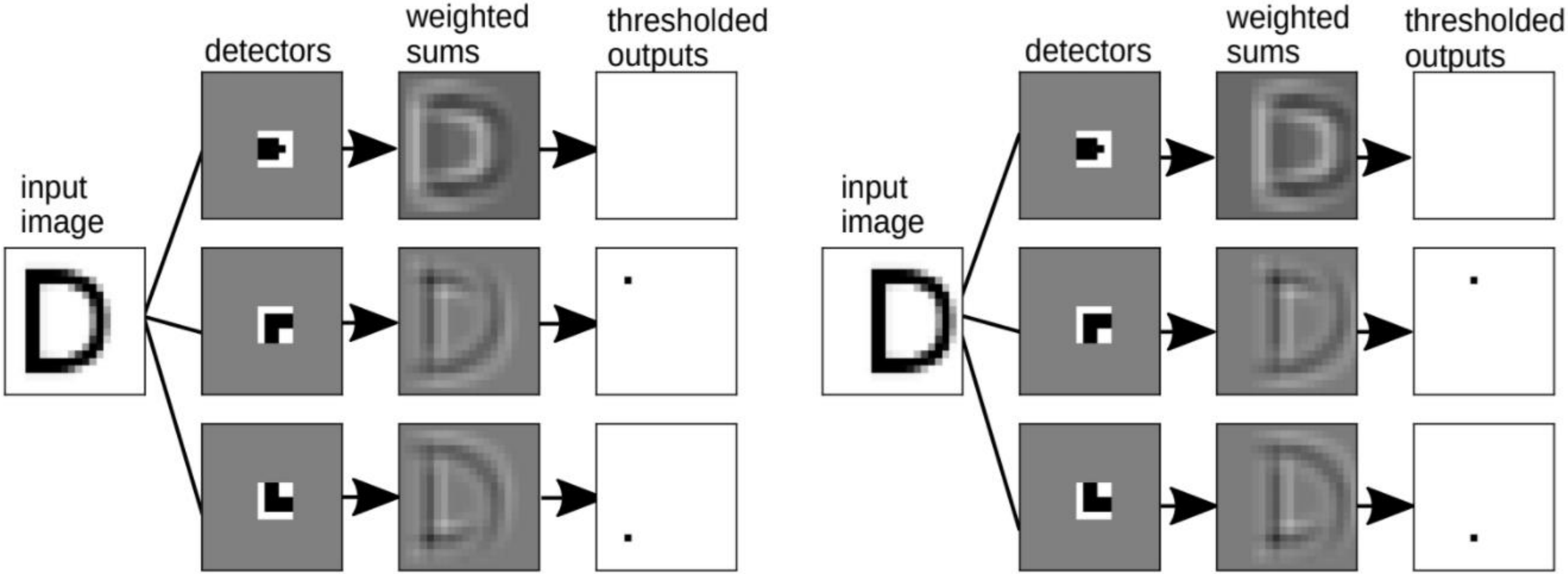


Convolution motivation



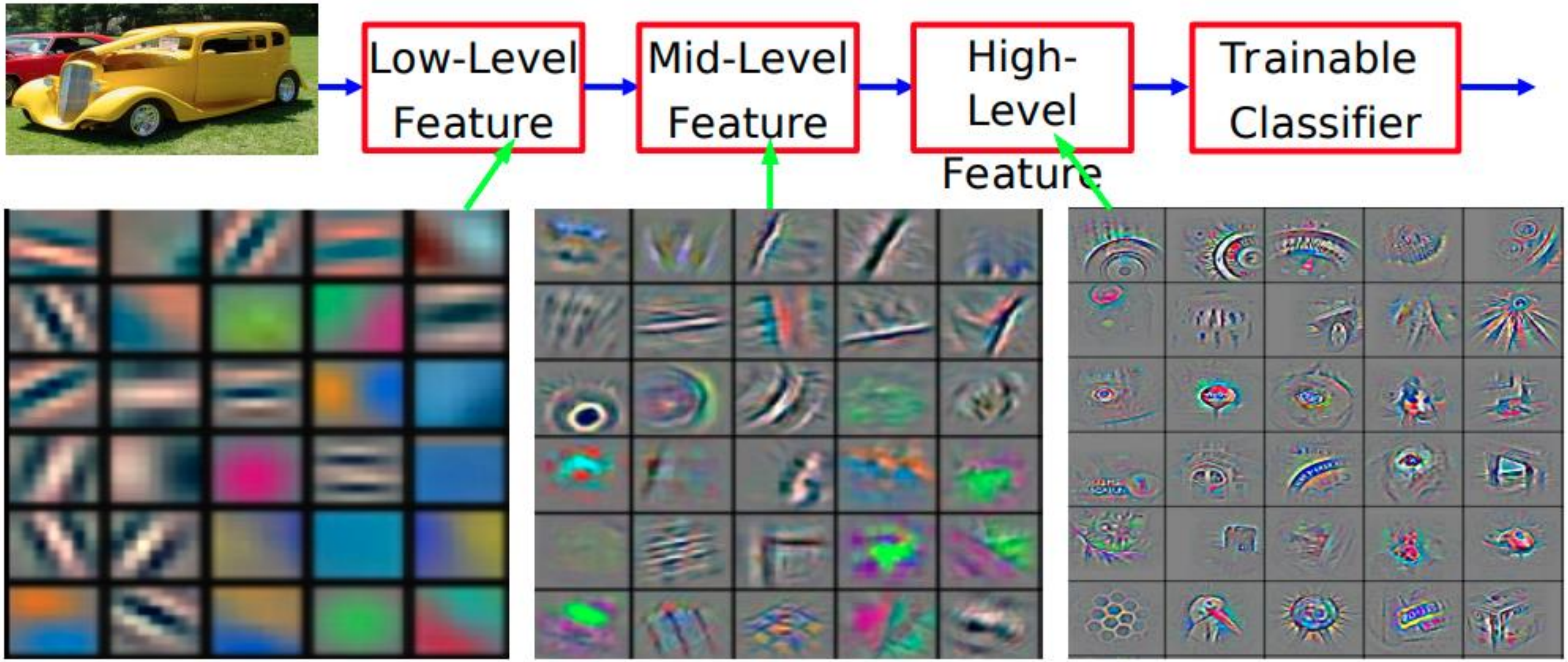
Slide credit: Yann Lecun

Convolution motivation



Slide credit: Yann Lecun

Convolutional features



Top image credit: Yann Lecun
Bottom image credit: Visualizing and Understanding Convolutional Networks (Zeiler & Fergus, 2013)

Common CNN Architecture

Convolutional Layers (Conv + ReLU):

- Extracts local patterns like edges and textures.
- Uses ReLU activation to introduce non-linearity.

Pooling Layers (Max/Average Pooling):

- Reduces spatial dimensions while retaining important features.
- Increases translation invariance and reduces computation.

Stacking Conv & Pooling Layers:

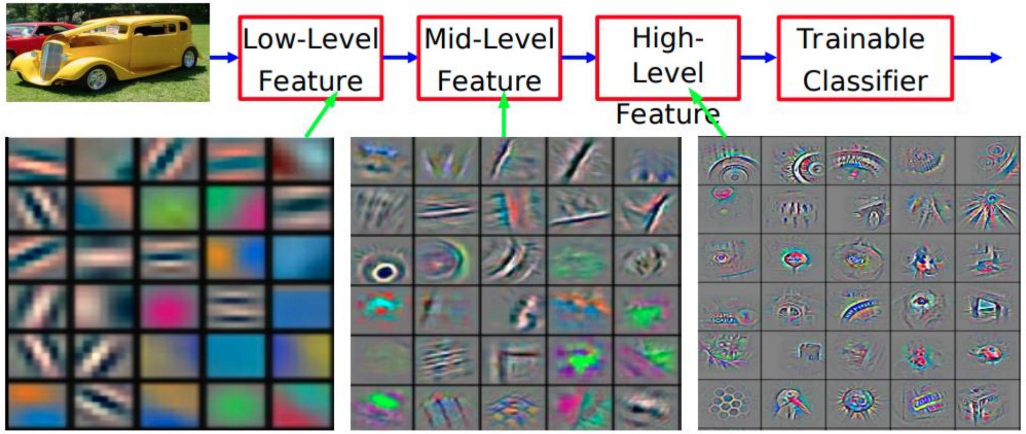
- Multiple layers capture hierarchical features (simple to complex).

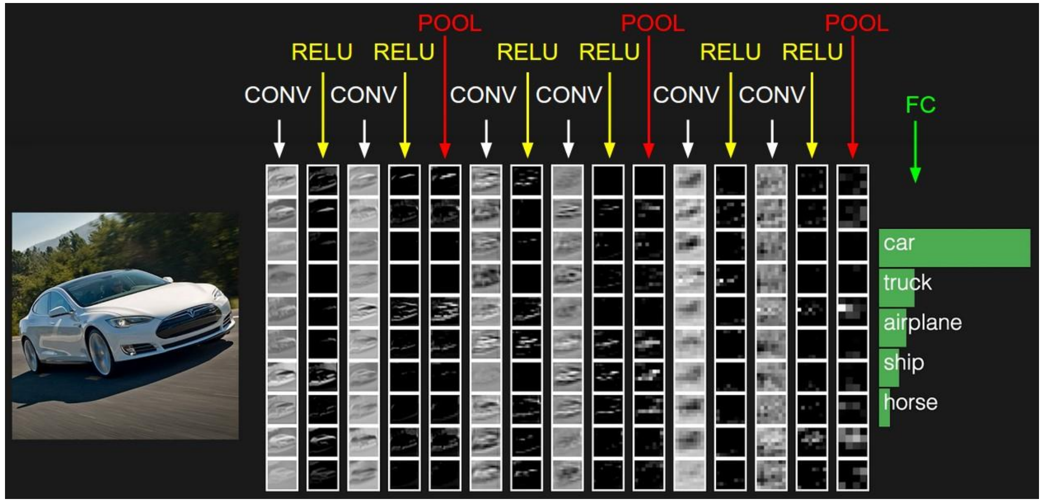
Fully Connected (FC) Layers:

- Flattened feature maps are passed through dense layers for classification.

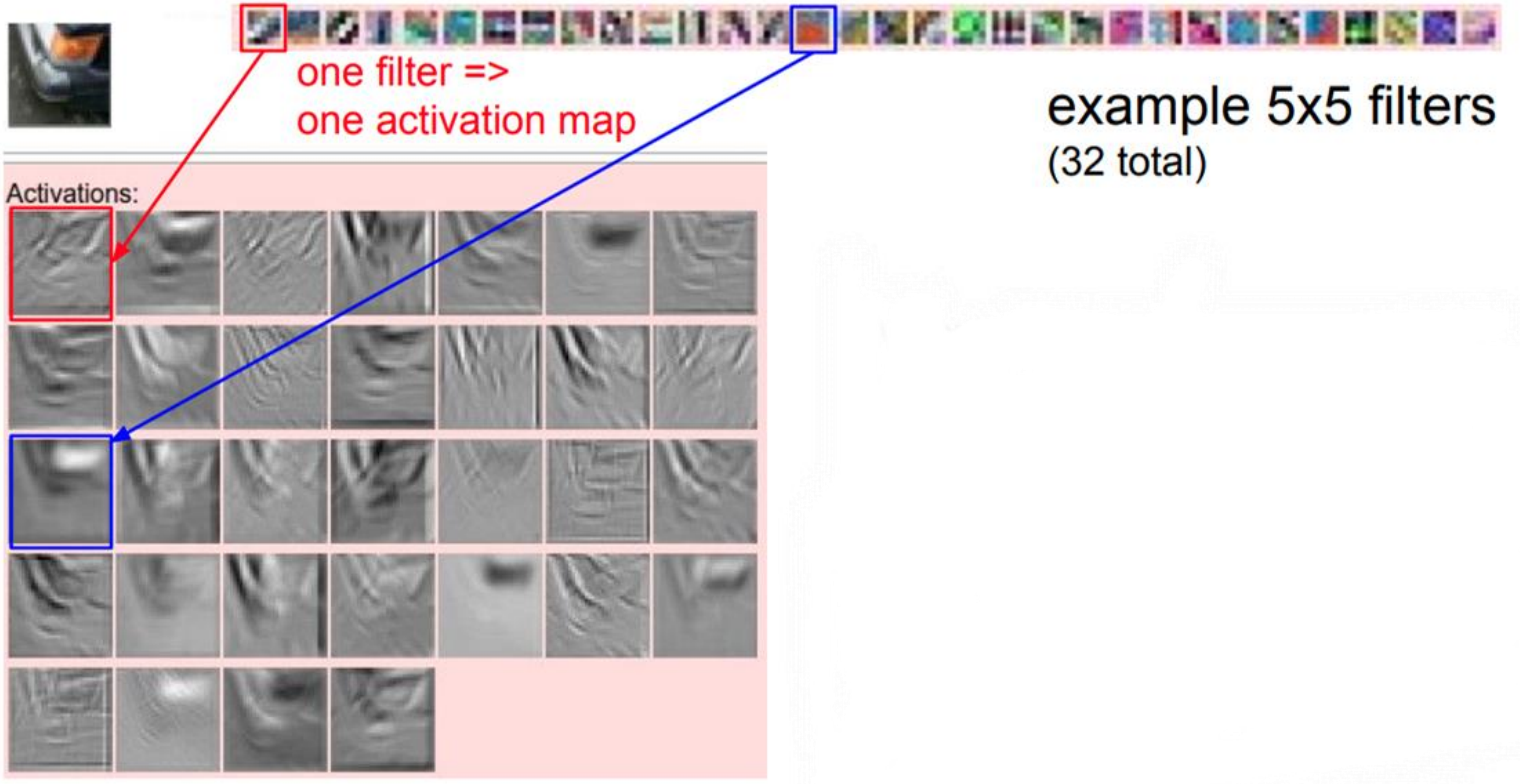
Output Layer:

- Softmax (multi-class) or Sigmoid (binary) activation for final predictions.





Convolutional kernels



Convolutional low-level features

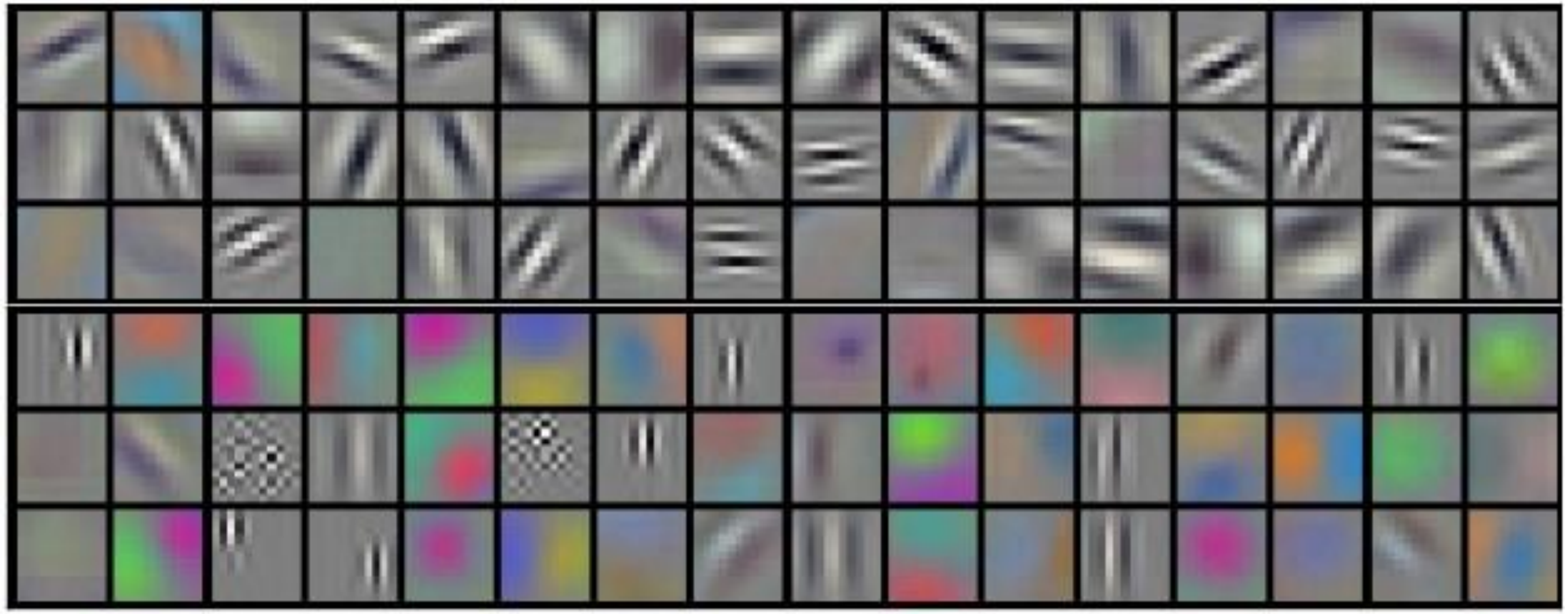
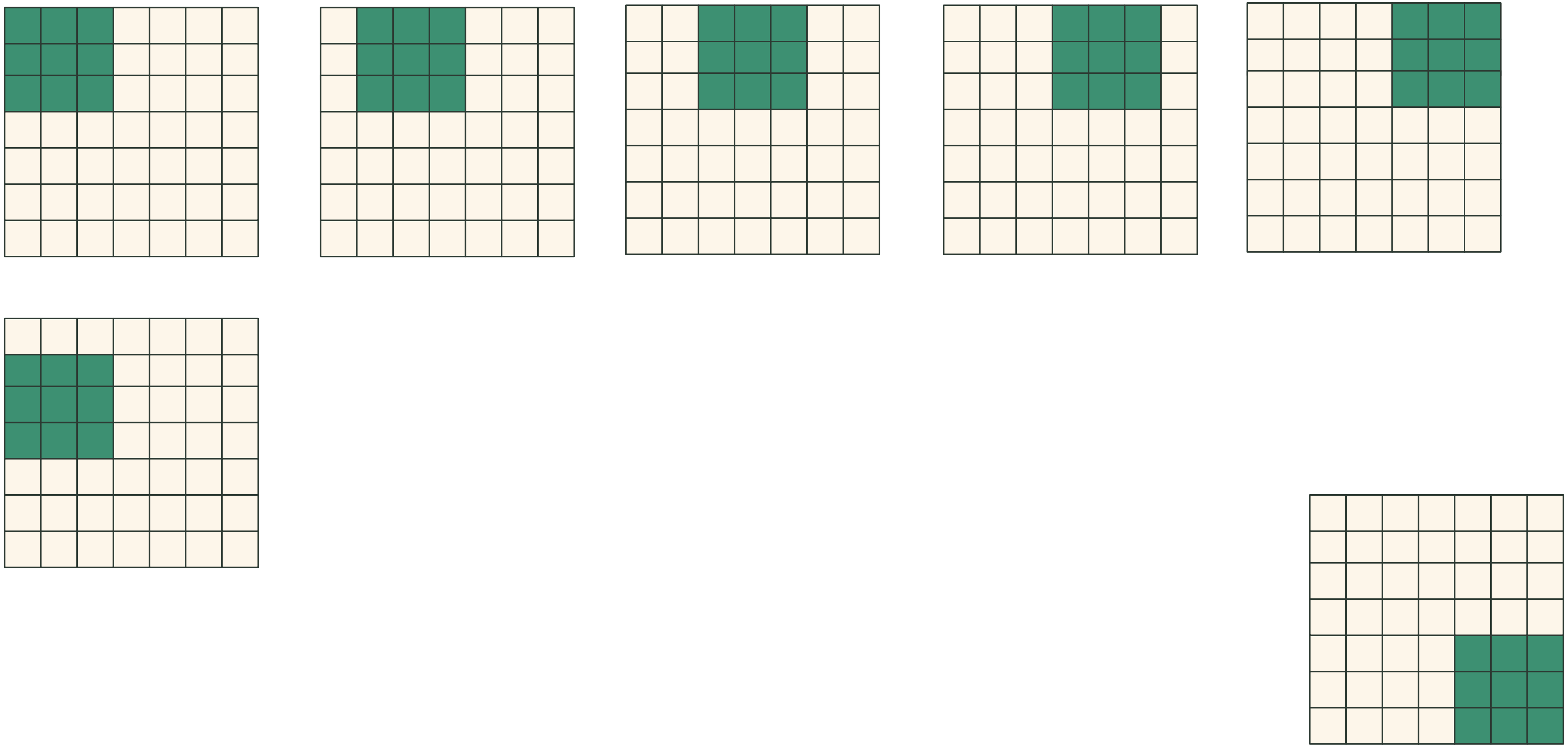


Image credit: Stanford CS231n

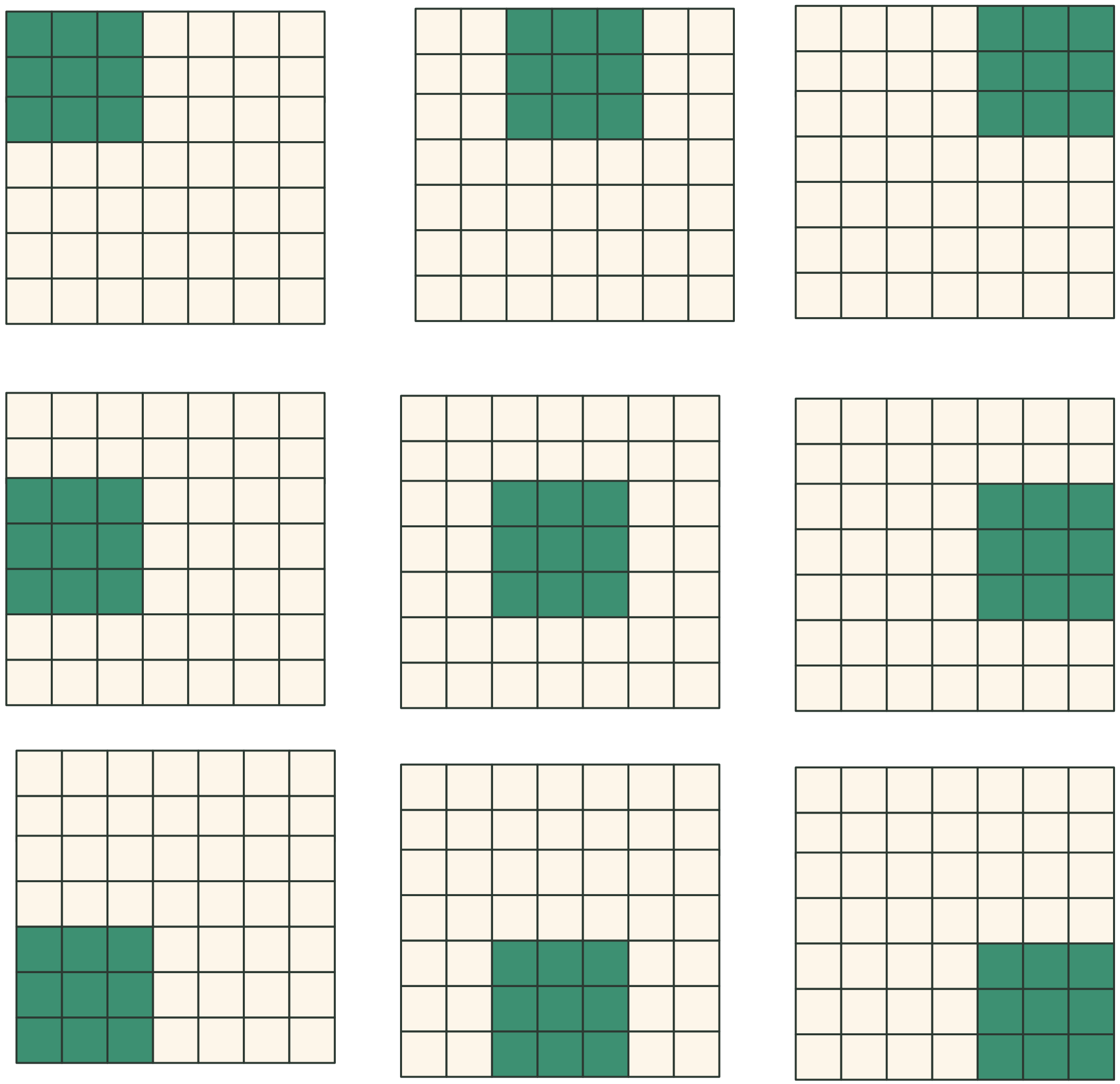
Convolution operation

N=7,F=3,S=1



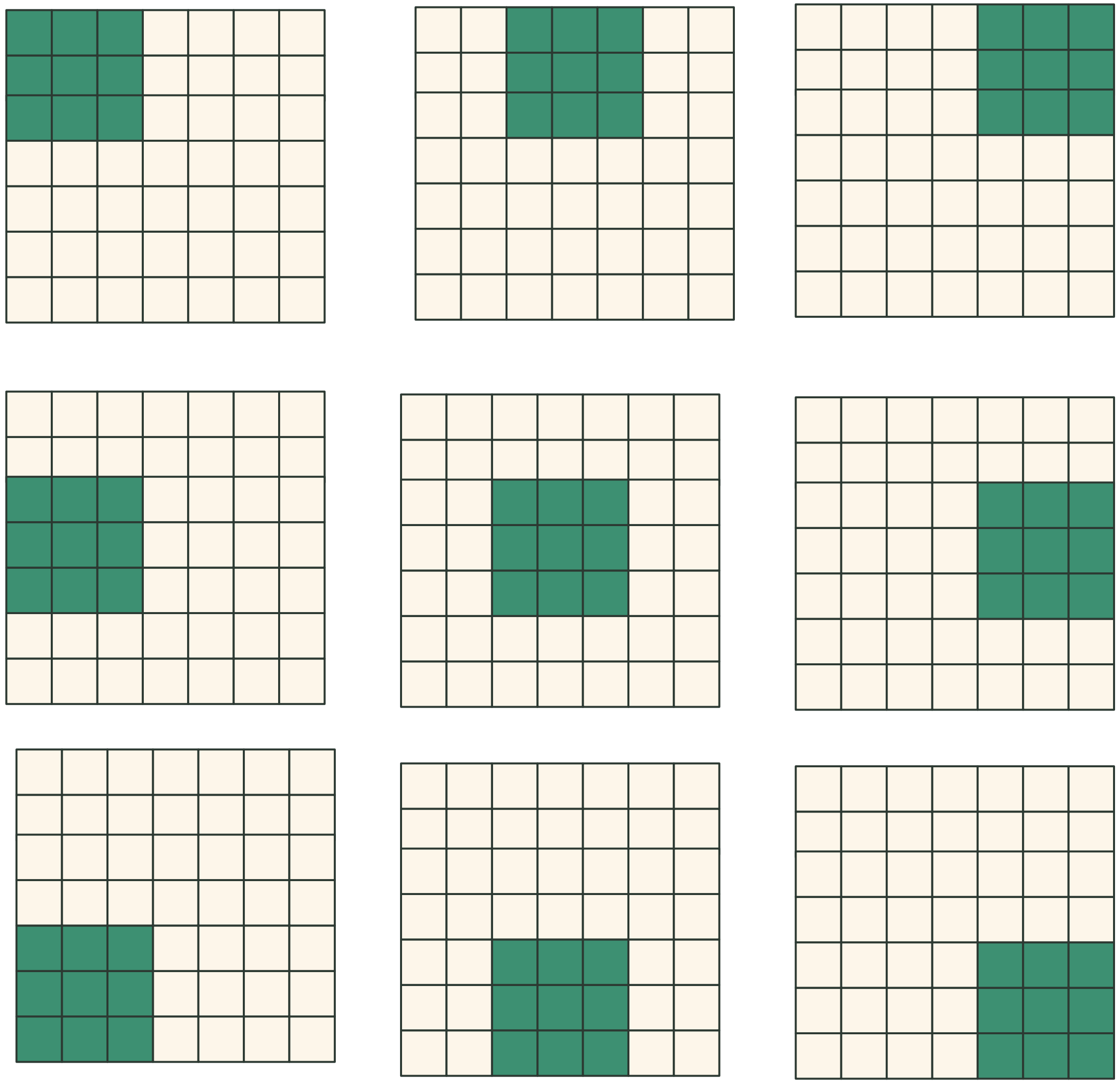
Convolution operation

N=7,F=3,S=2



Convolution operation

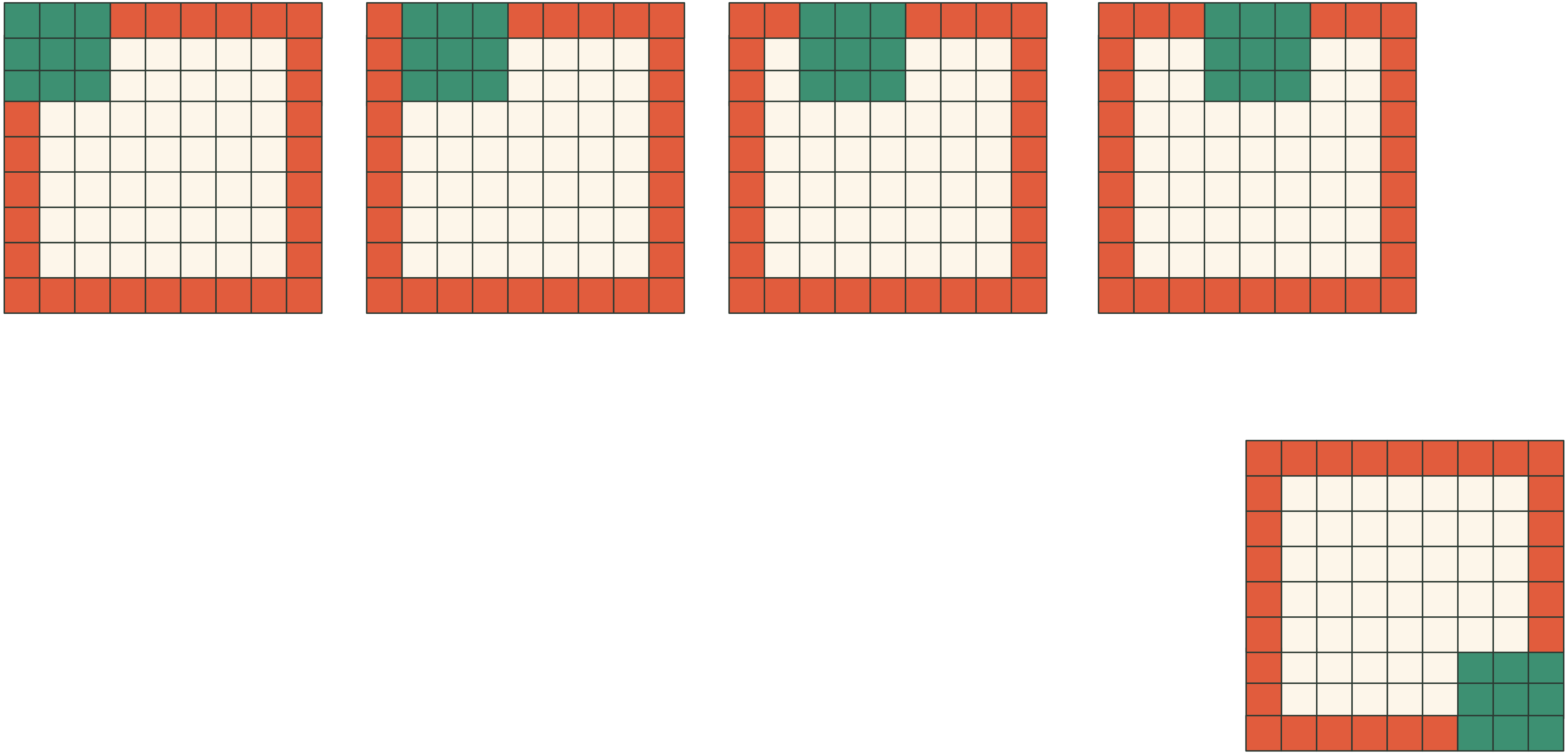
N=7,F=3,S=2



Output = (N-F)/S+1

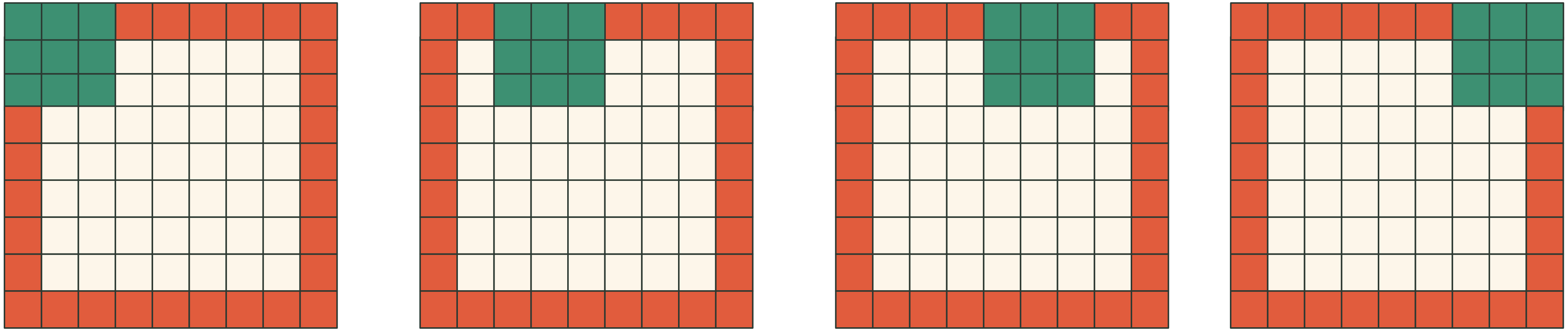
Convolution operation

`N=7, F=3, S=1, P=1`

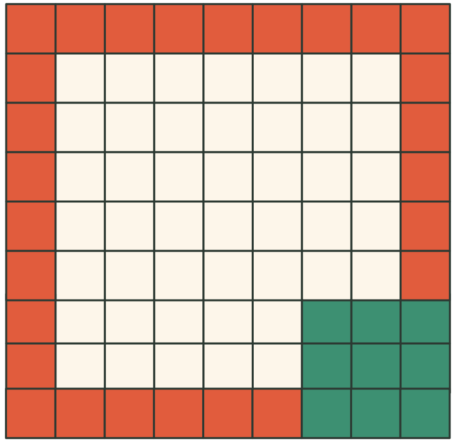


Convolution operation

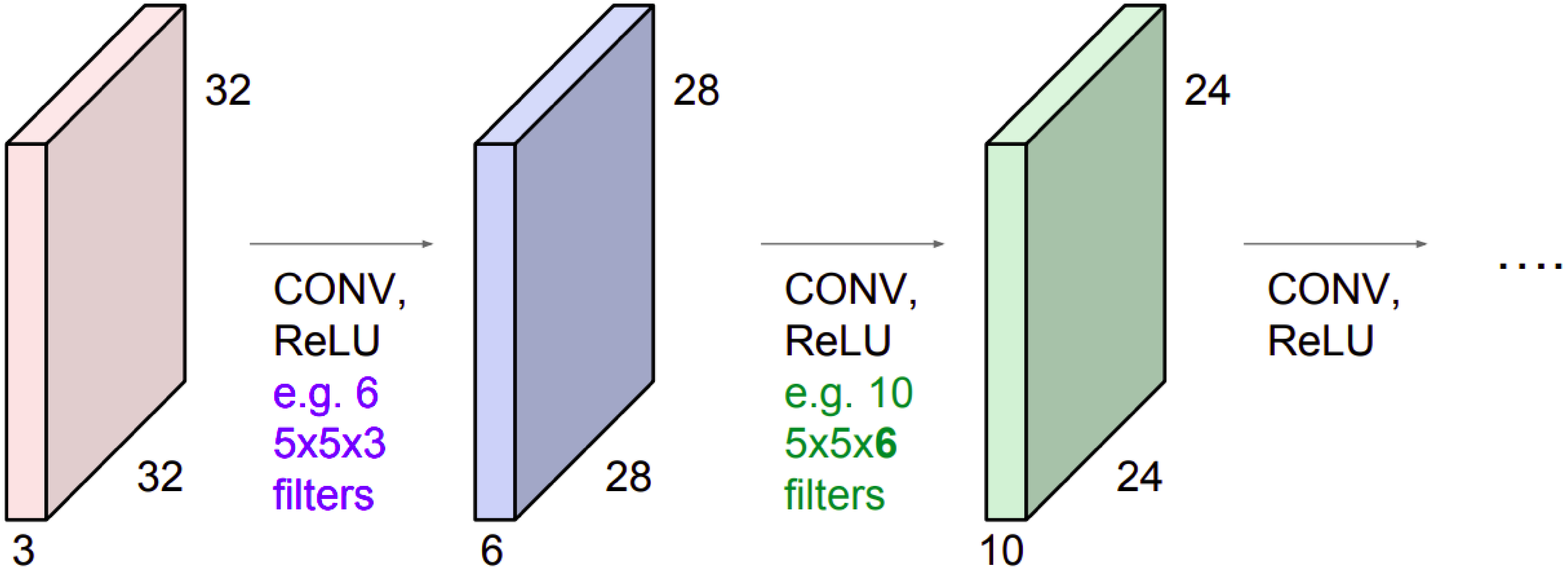
N=7 F=3, S=2, P=1



Output = (N-F+2P)/S+1



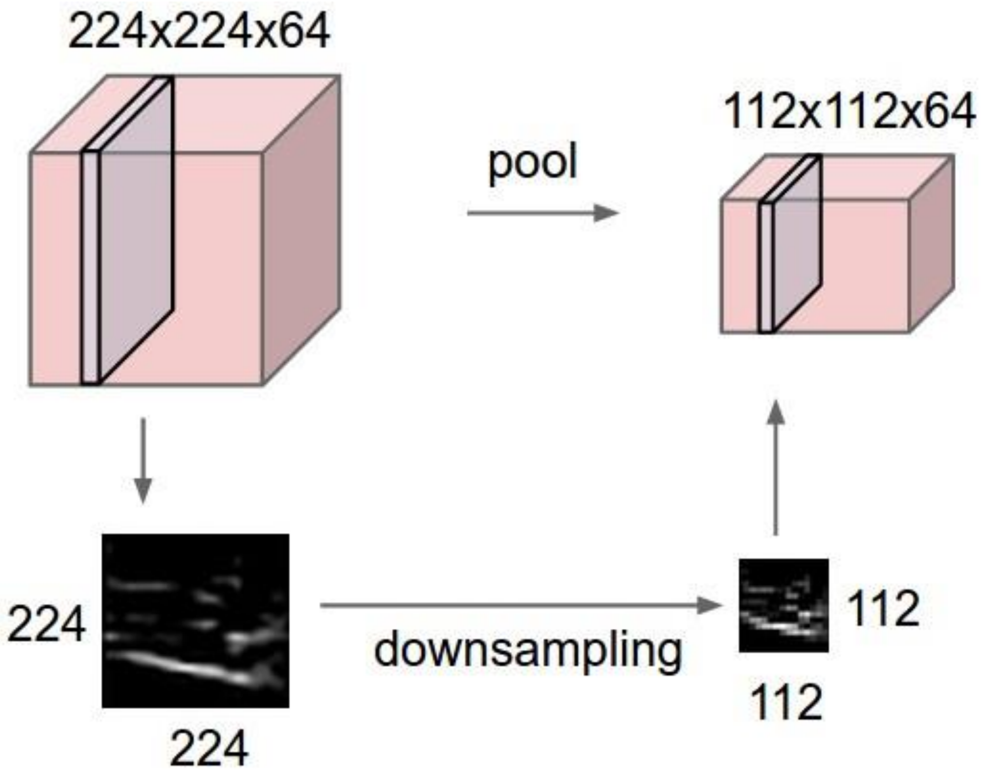
Number of parameters



Pooling layer in CNN

Types of Pooling:

- **Max Pooling:**
    - Selects the maximum value from a window (e.g., 2x2), preserving the most important features.
- **Average Pooling:**
    - Computes the average value in the window, emphasizing smoother features.



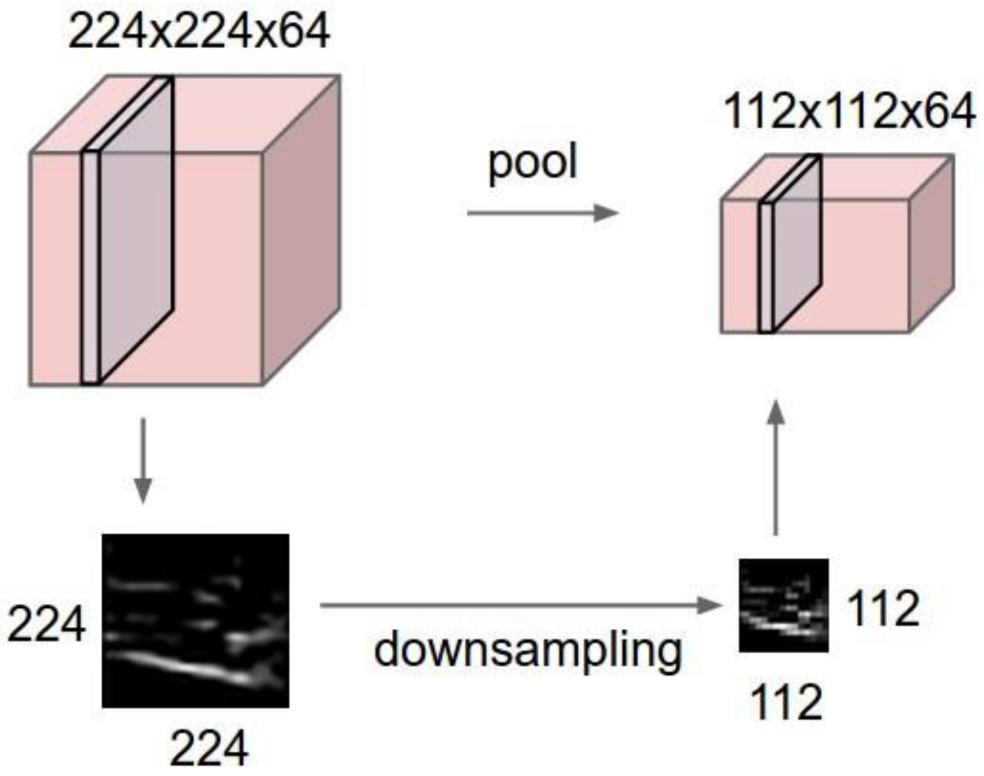
Pooling layer in CNN

Types of Pooling:

- **Max Pooling:**
  - Selects the maximum value from a window (e.g., 2x2), preserving the most important features.
- **Average Pooling:**
  - Computes the average value in the window, emphasizing smoother features.

Benefits:

- **Dimensionality Reduction:**
  - Reduces the number of parameters and computation.
- **Translation Invariance:**
  - Helps the model become less sensitive to slight translations of features.
- **Control overfitting**



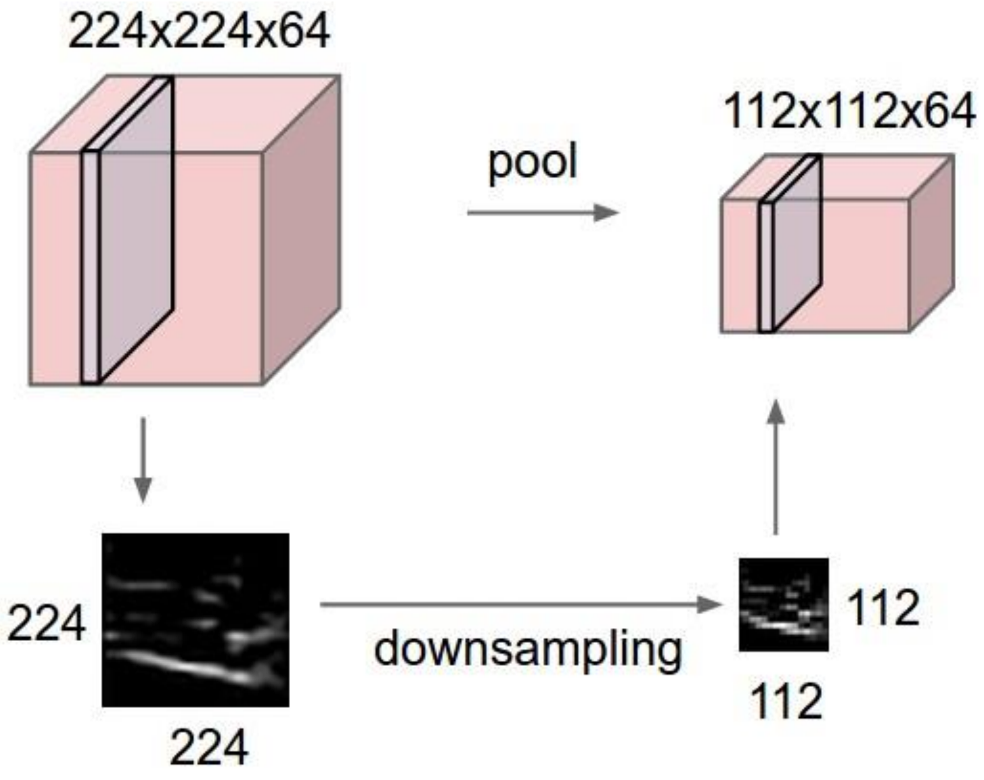
Pooling layer in CNN

Types of Pooling:

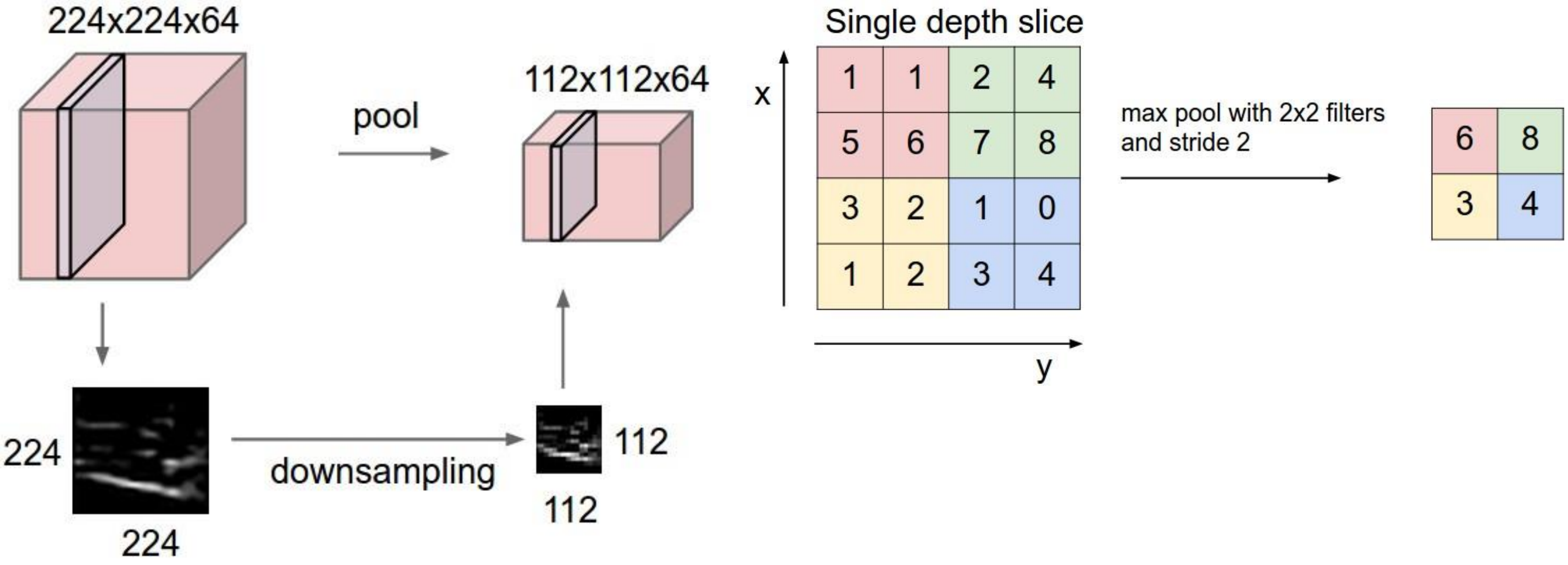
- **Max Pooling:**
  - Selects the maximum value from a window (e.g., 2x2), preserving the most important features.
- **Average Pooling:**
  - Computes the average value in the window, emphasizing smoother features.

Benefits:

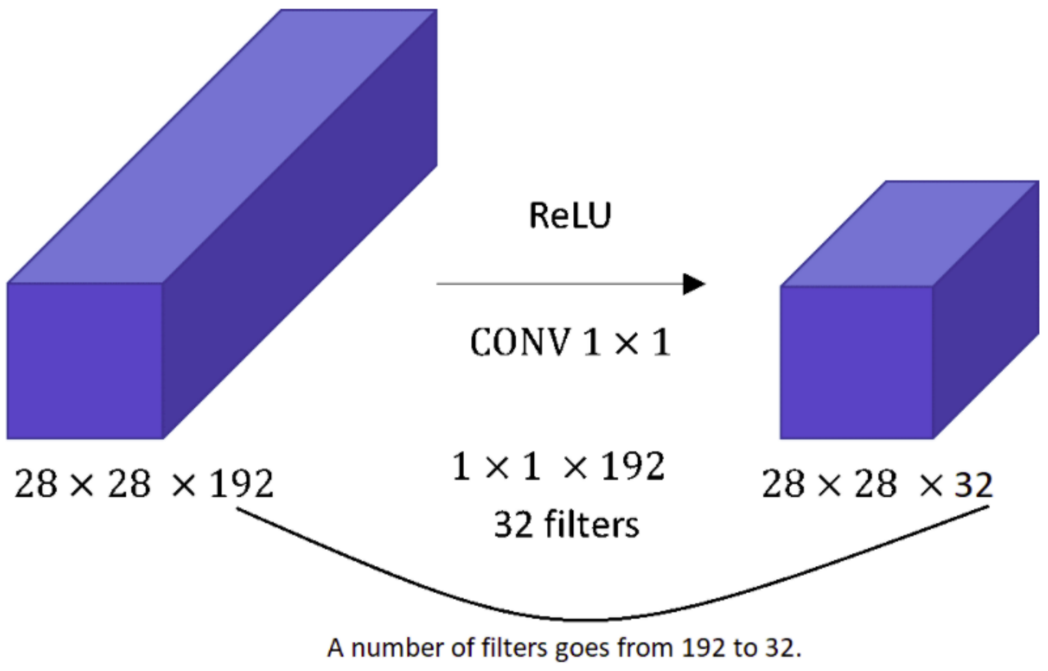
- **Dimensionality Reduction:**
  - Reduces the number of parameters and computation.
- **Translation Invariance:**
  - Helps the model become less sensitive to slight translations of features.
- **Control overfitting**



Pooling layer (Maxpool)



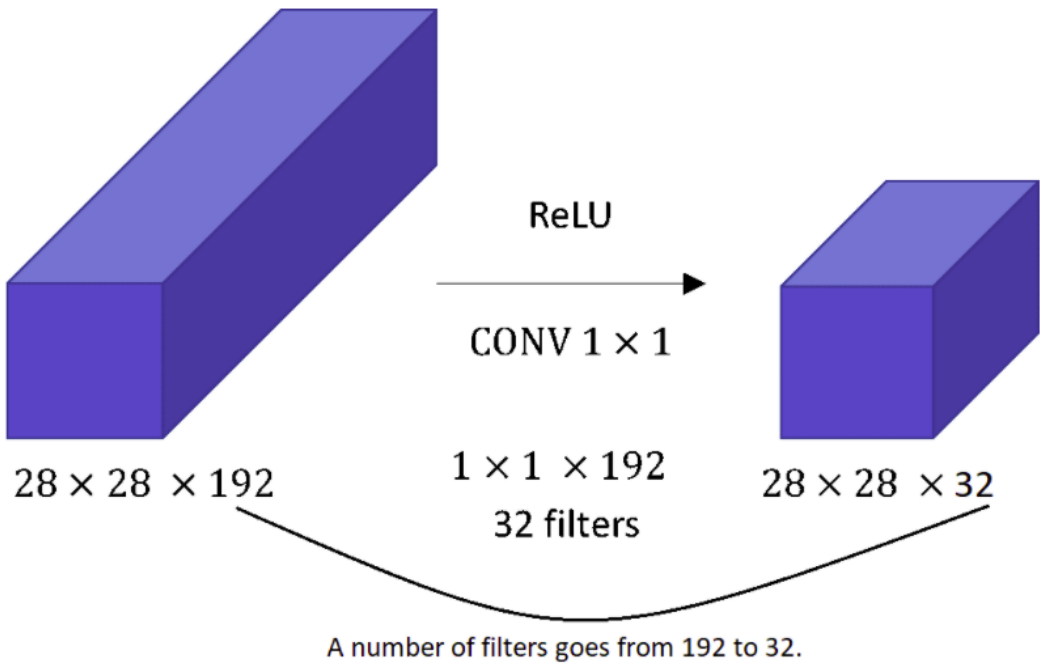
1x1 Convolutions in CNN



Purpose:

- Applies a convolution with a filter size of 1x1, processing individual pixels while leveraging depth channel information.

1x1 Convolutions in CNN



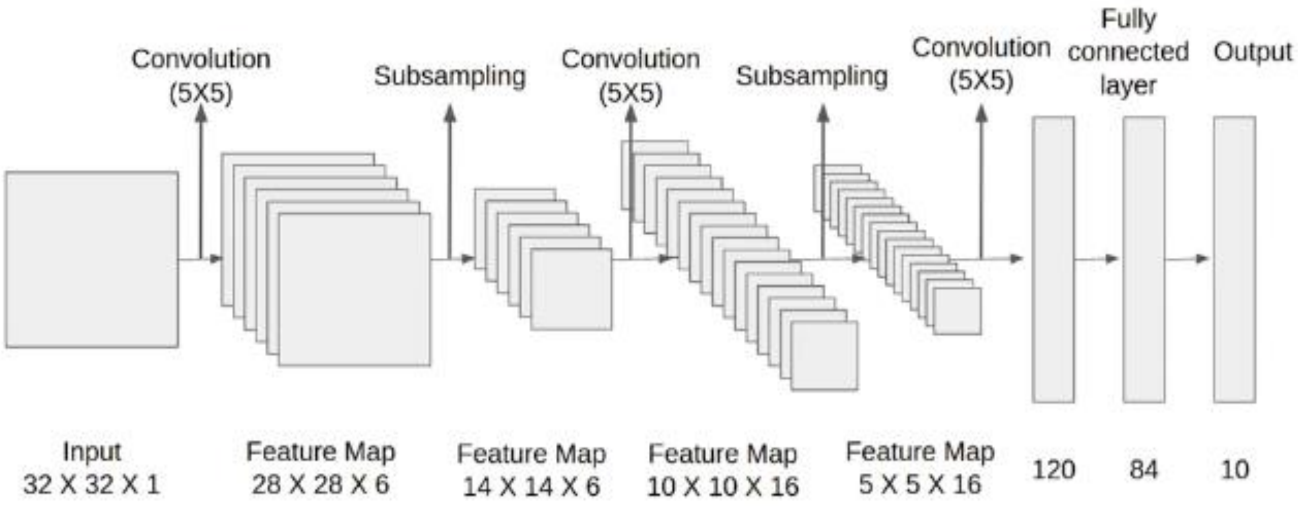
Purpose:

- Applies a convolution with a filter size of 1x1, processing individual pixels while leveraging depth channel information.

Key Benefits:

- **Dimensionality Reduction:**
    - Reduces the number of channels (depth) without affecting spatial dimensions.
- **Channel-wise Interactions:**
    - Allows the model to learn complex relationships between channels, improving feature representation.
- **Computational Efficiency:**
    - Lightweight operation, reducing the number of computations in deeper networks.

LeNet5 Architecture



Overview:

- Early CNN for digit classification (MNIST), proposed by Yann LeCun in the 1990s.

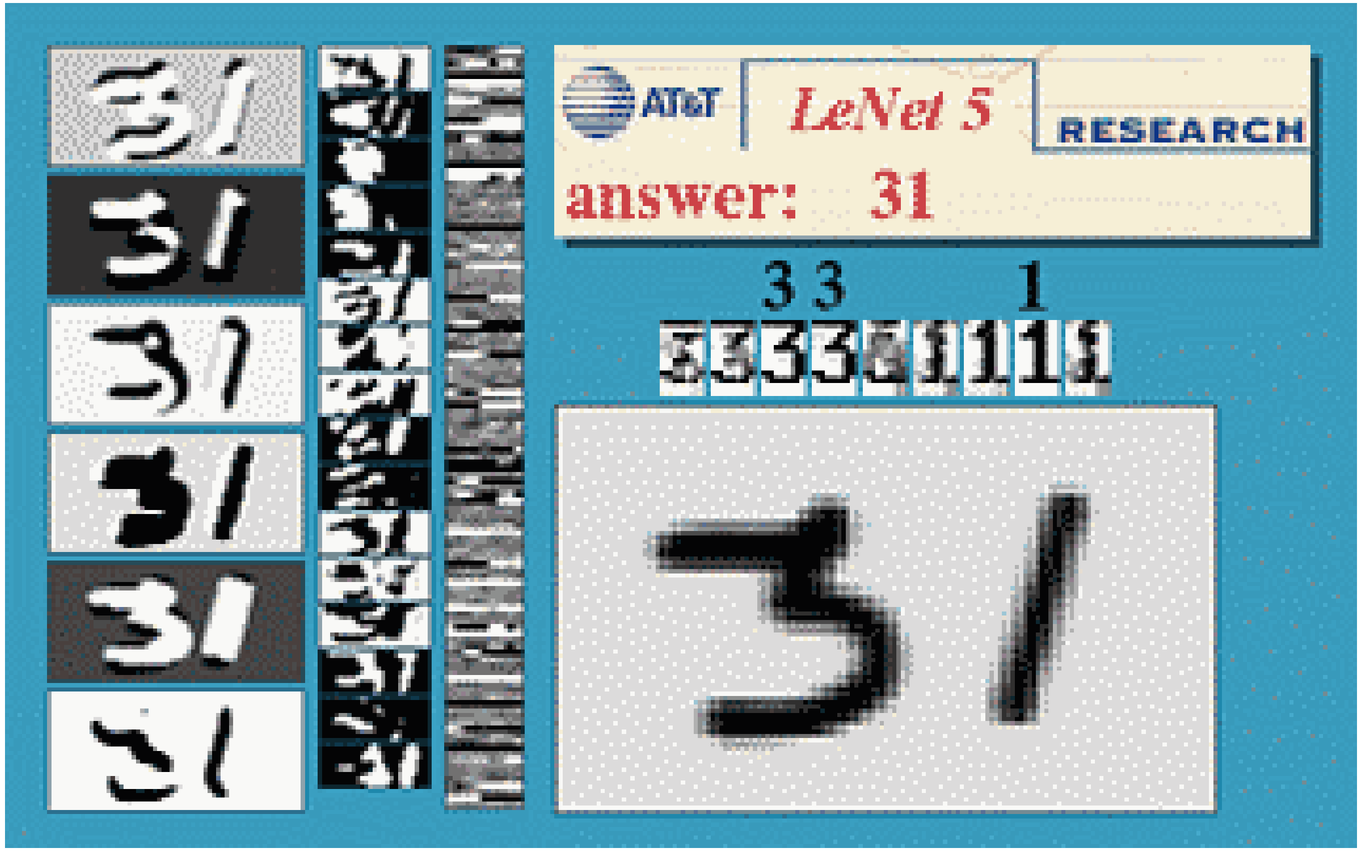
Architecture:

- **Input:** 32x32 grayscale image.
- **Conv Layer 1:** 6 filters (5x5), output 28x28x6.
- **Pool Layer 1:** 2x2 max pooling, output 14x14x6.
- **Conv Layer 2:** 16 filters (5x5), output 10x10x16.
- **Pool Layer 2:** 2x2 max pooling, output 5x5x16.
- **FC Layers:** 120, 84 units.
- **Output Layer:** 10 units for classification.

Key Features:

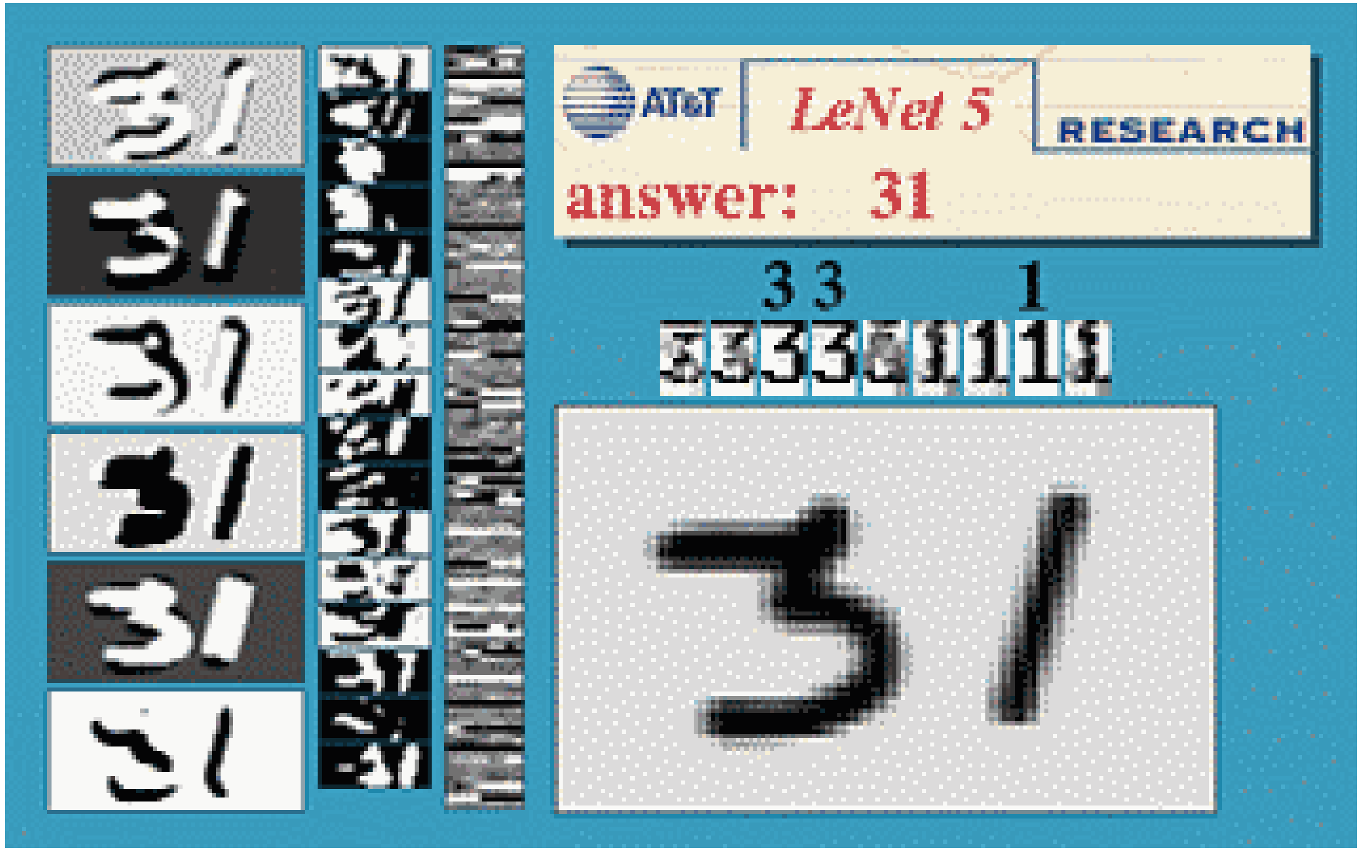
- Introduced CNNs with convolution and pooling layers for feature extraction.

LeNet5



Credit: Yann Lecun

LeNet5




```python
class LeNet5(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 20, 5, 1)
        self.fc1 = nn.Linear(4*4*20, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*20)
        x = F.relu(self.fc1)
        x = self.fc2(x)
        return F.logsoftmax(x, dim=1)
```

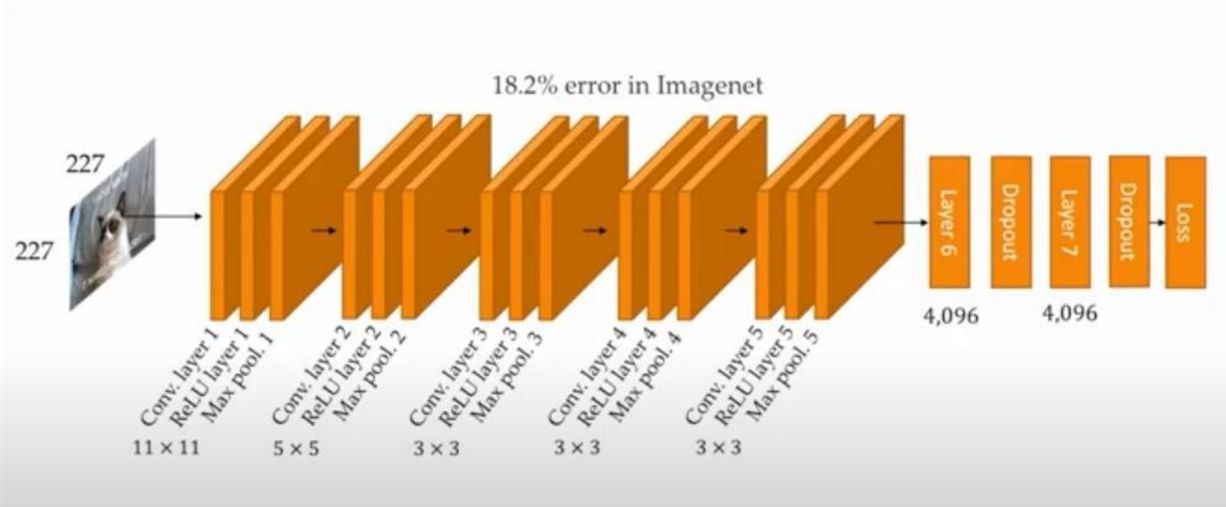
AlexNet architecture

Overview:

- Deep CNN designed by Alex Krizhevsky, won the 2012 ImageNet competition.

Key Features:

- **ReLU Activation** for faster training.
- **5 Convolutional Layers** and **3 Max Pooling Layers** for feature extraction.
- **3 Fully Connected Layers** for classification.
- **Dropout** for regularization and **GPU acceleration** for efficient training.



ImageNet 2012