Deep Learning Bootcamp

A hands-on intensive workshop covering neural fundamentals,
computer vision, and natural language processing using modern deep learning tools.

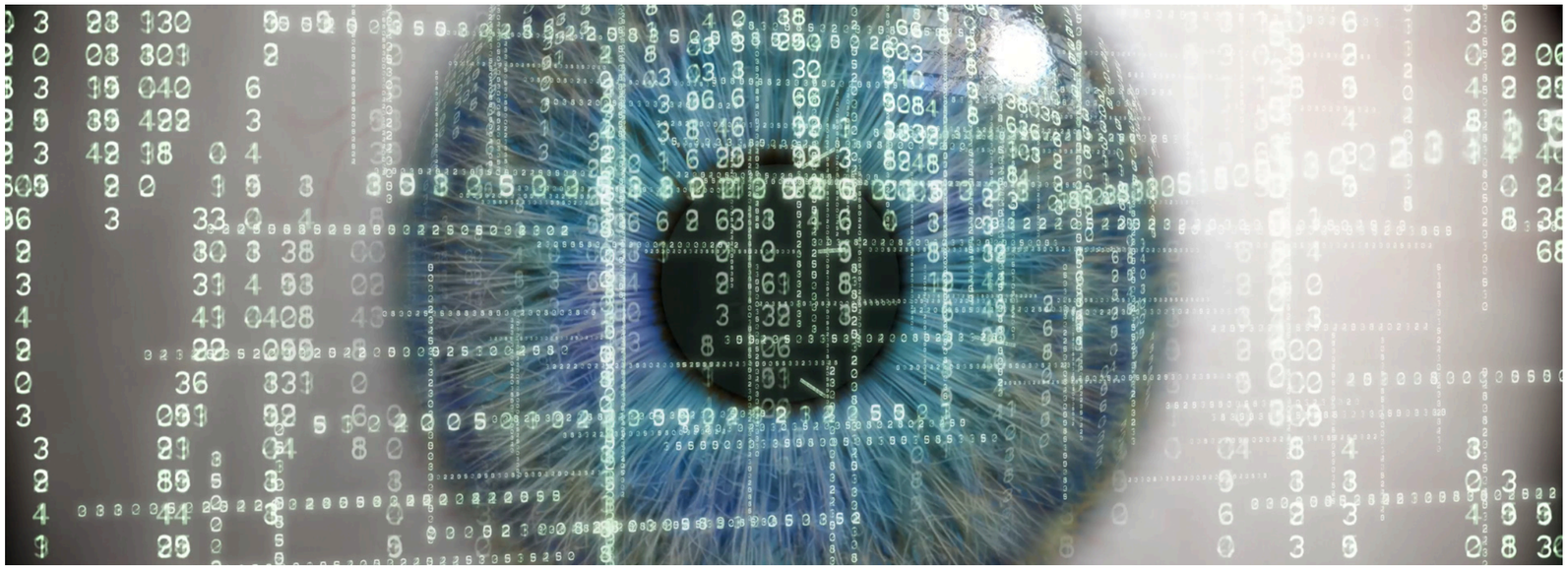
by: Lama Ayash

Day 2

Course Outlines

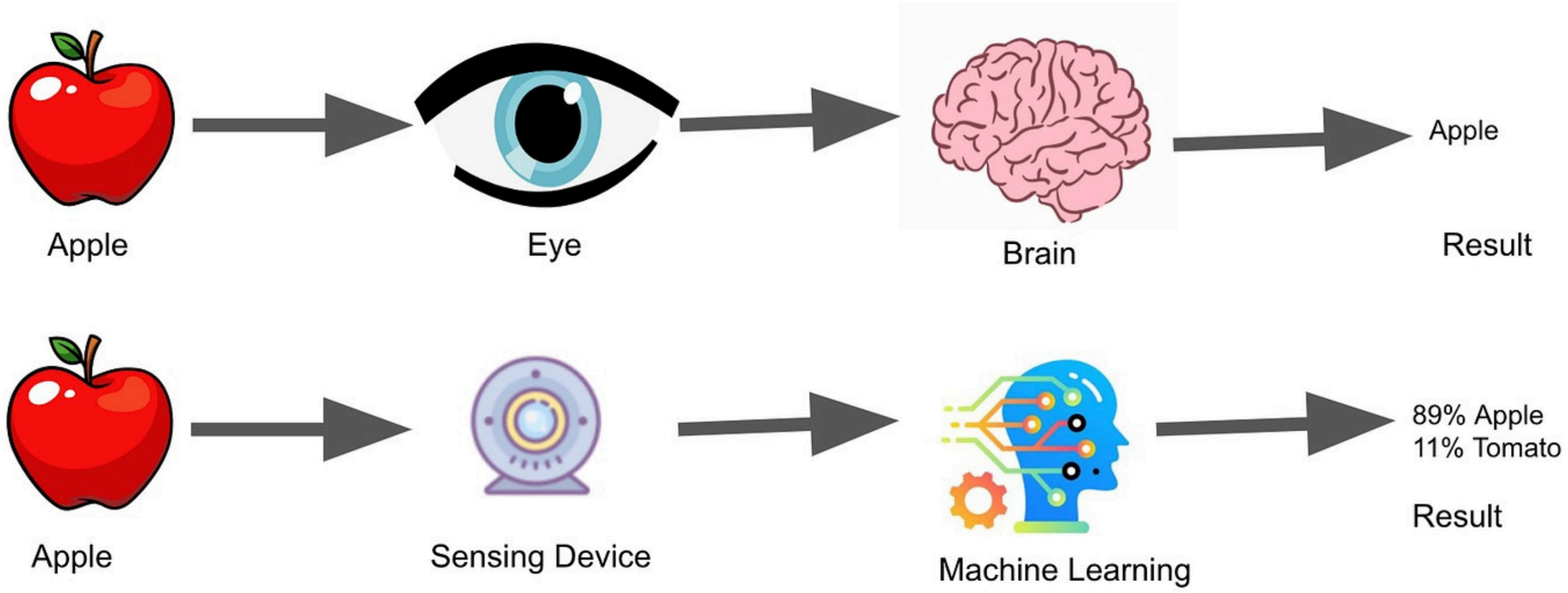
Computer Vision

Computer Vision (CV) enables machines to see, understand, and interpret the visual world. Inspired by how humans perceive and process visual information.



Computer Vision

Computer Vision (CV) enables machines to see, understand, and interpret the visual world. Inspired by how humans perceive and process visual information.



Computer Vision Tasks

- Image Classification

Classification is the simplest and most common task in computer vision.

The class in this case is a cat

The class in this case is a dog

Computer Vision Tasks

- Object Detection and Localization

  The goal is to detect and locate specific objects within a larger image.

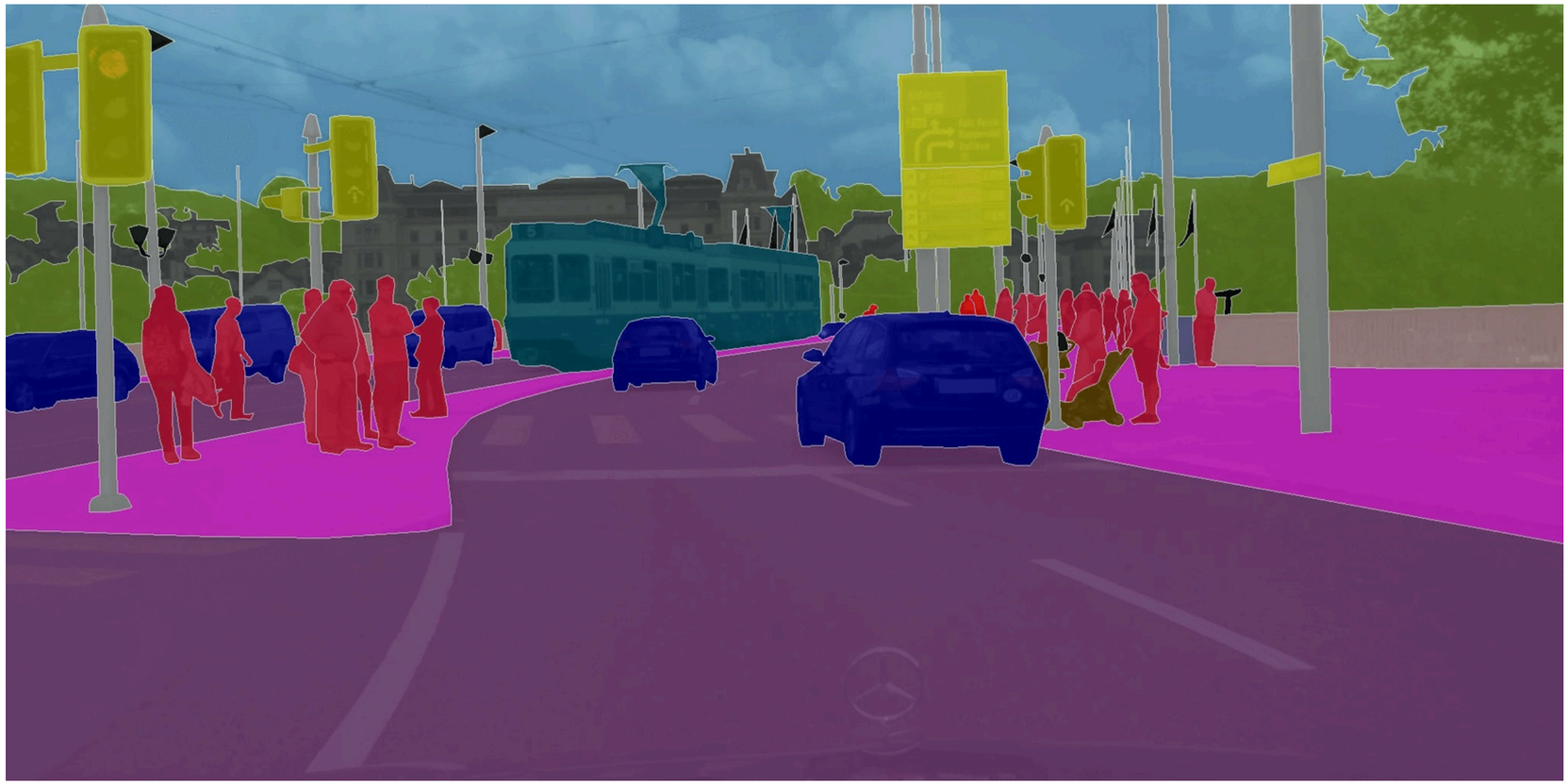
Class: No mask detected

Class: A mask is detected in the image

Computer Vision Tasks

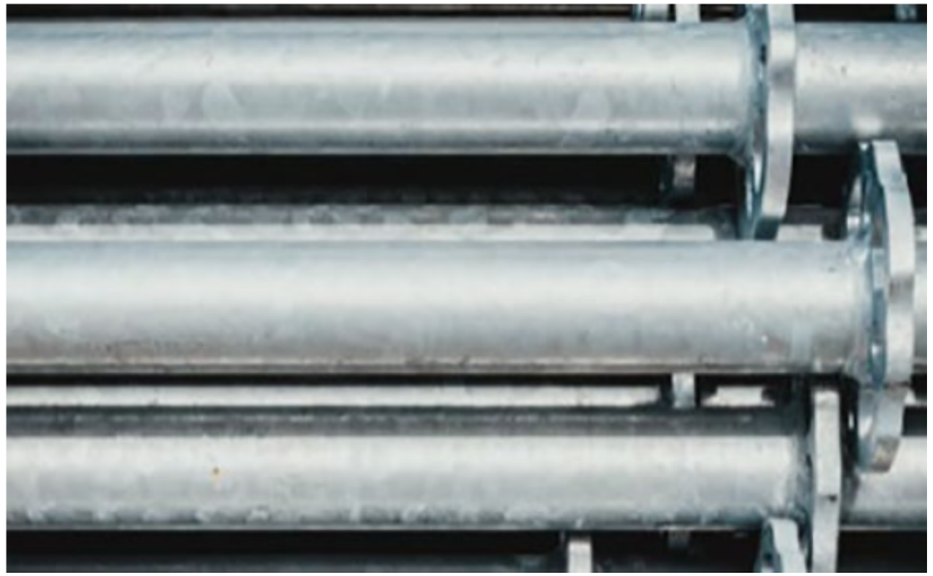
- Image Segmentation

  Segmentation divides an image into distinct regions or objects based on color, texture, and boundaries.



Computer Vision Tasks

- Anomaly Detection
  - Detects unusual patterns or defects in images by comparing them to normal reference data.



Perfect examples of steel pipes

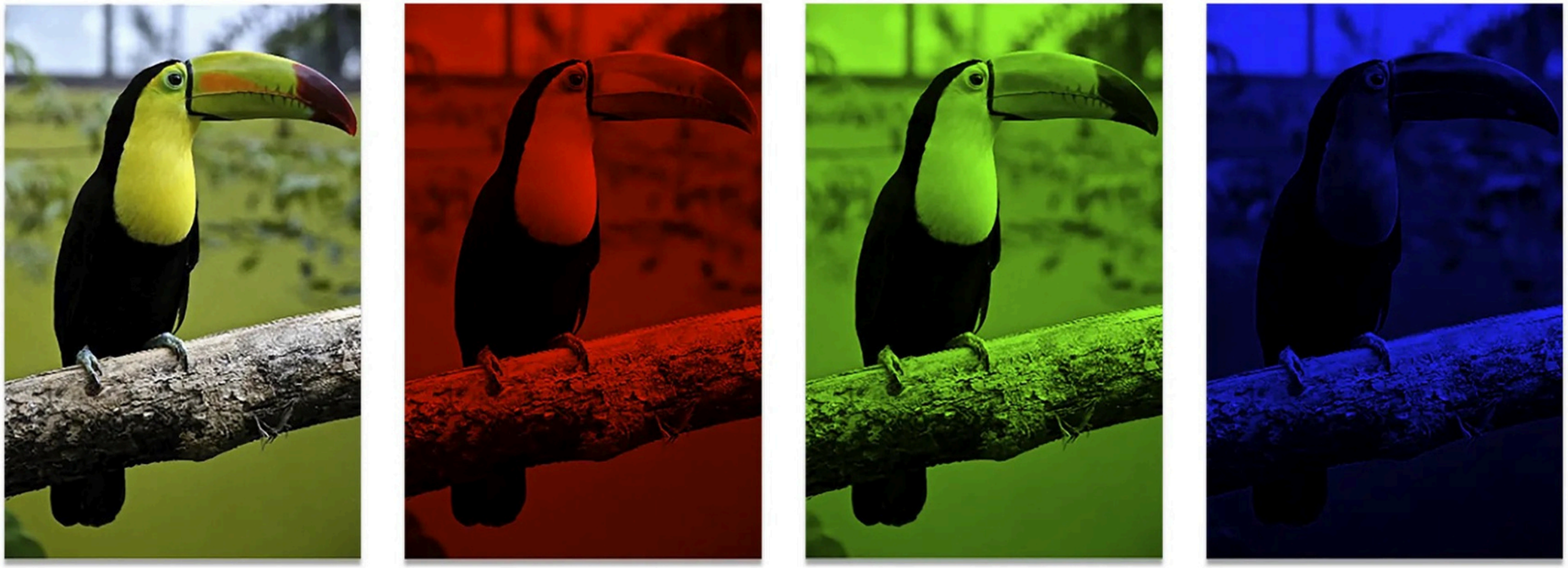
Channels

- A **channel** represents one dimension of color or intensity information in an image.

2	36	40	200
195	190	20	180
40	54	200	200
30	40	200	180

Channles

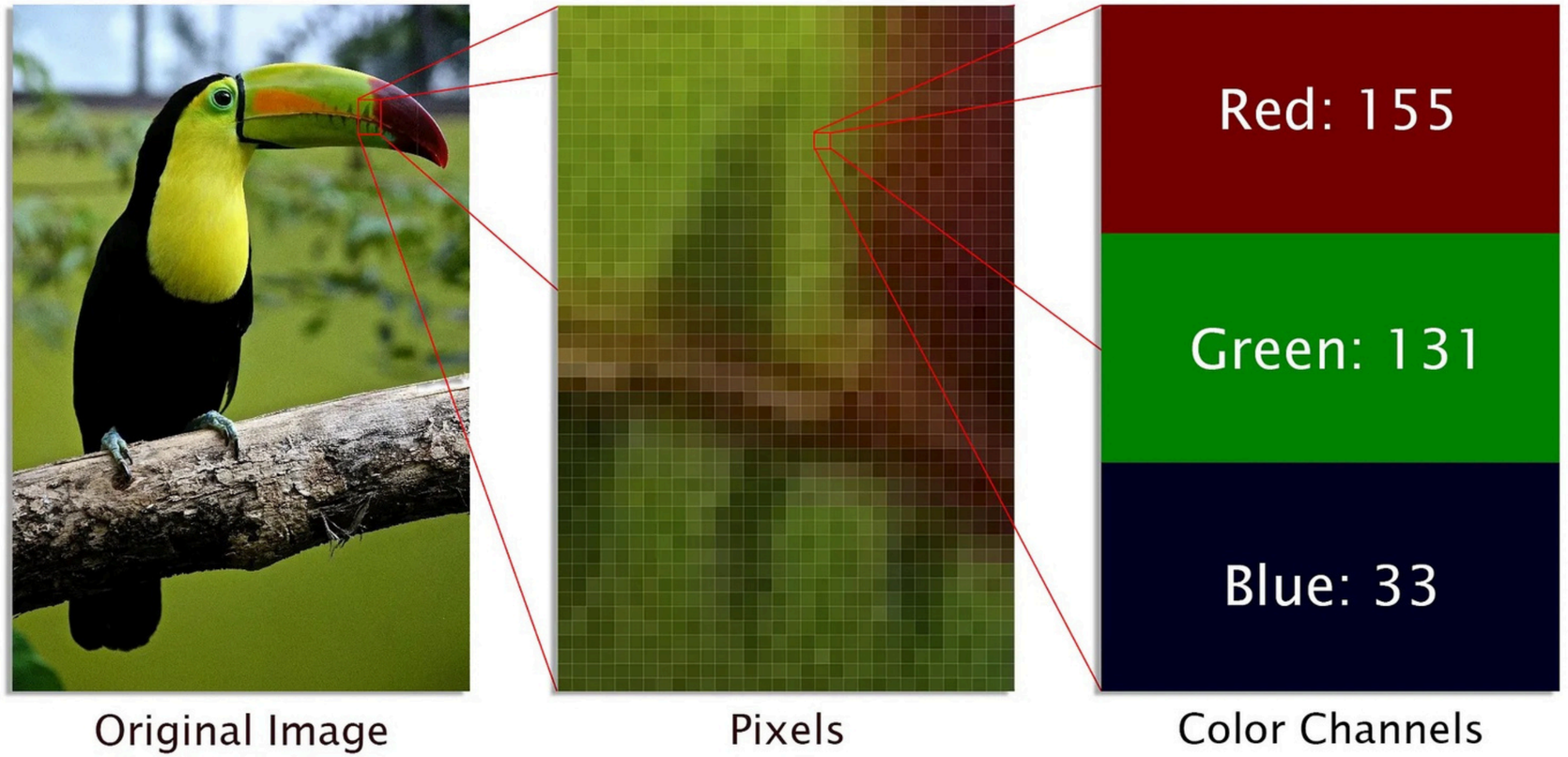
- A color image has three channels: Red, Green, and Blue (RGB).



Original Image | Red Channel | Green Channel | Blue Channel

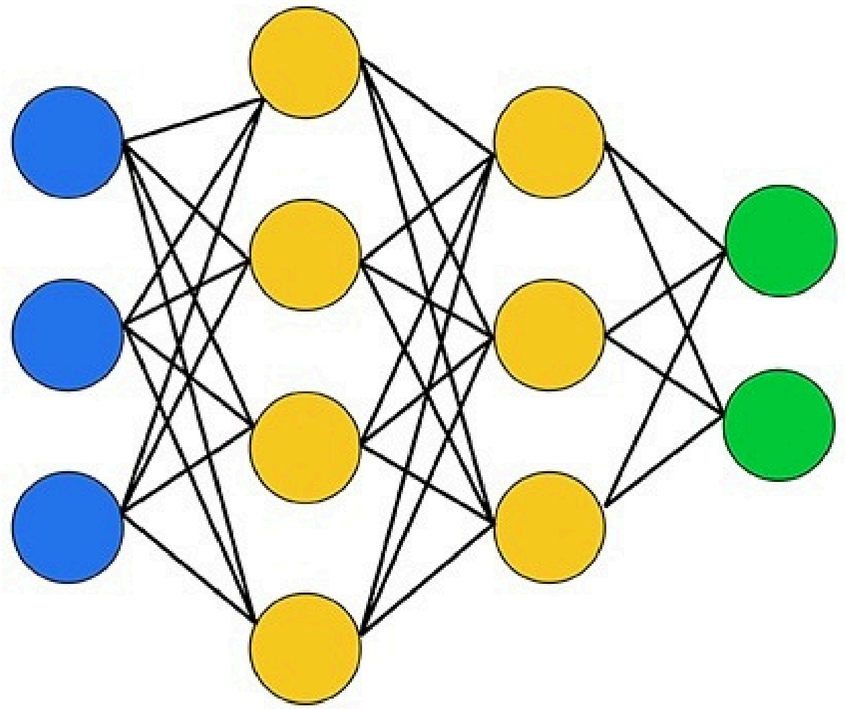
Channles

- A color image has three channels: Red, Green, and Blue (RGB).



Vision Models

- A color image has three channels: Red, Green, and Blue (RGB).



But can we apply the same approach to images?

Vision Models

Little or no invariance to shifting, scaling, and other forms of distortion

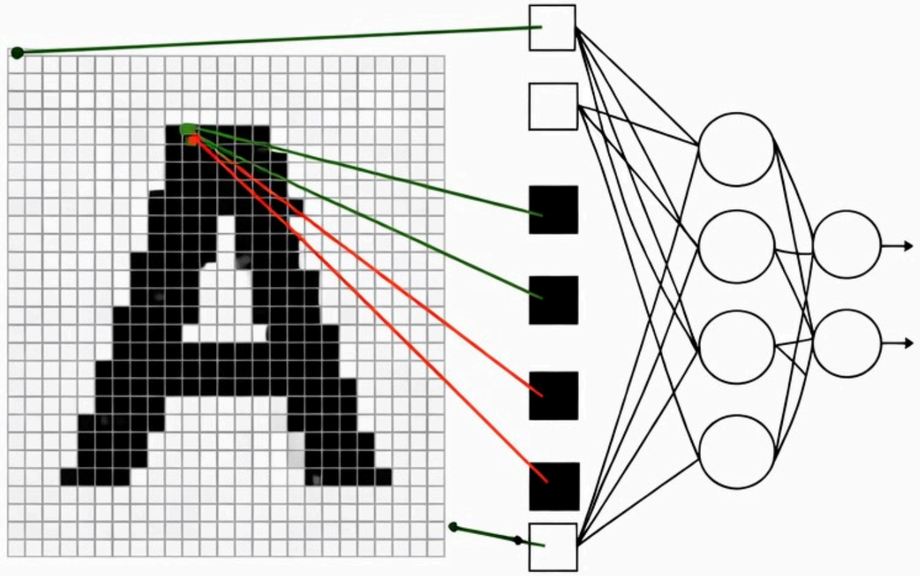


Figure 2: Original "A" character.

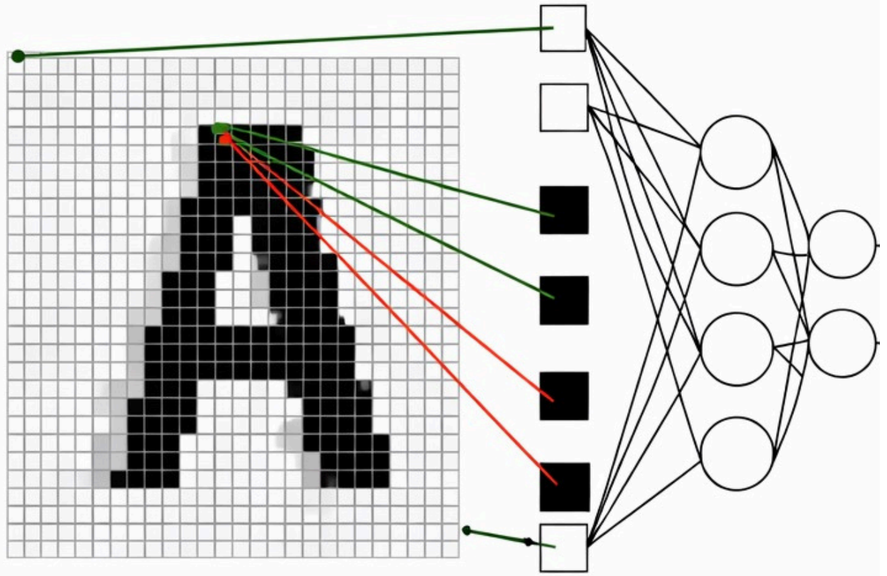
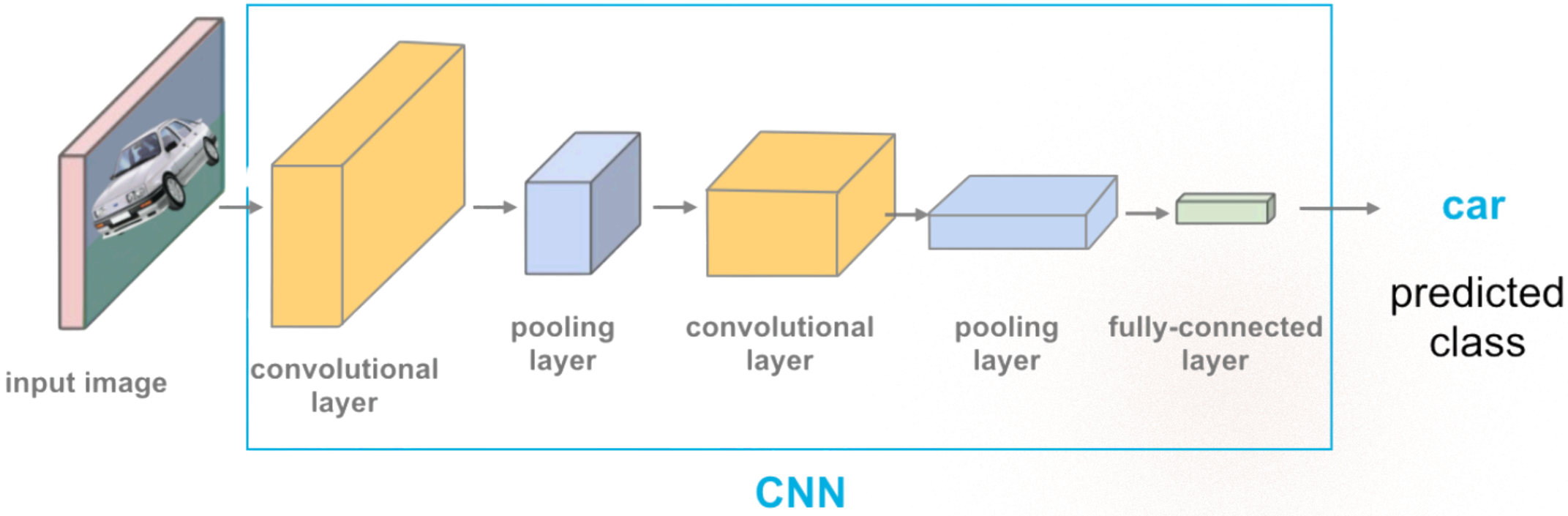


Figure 3: Shifted "A" character.

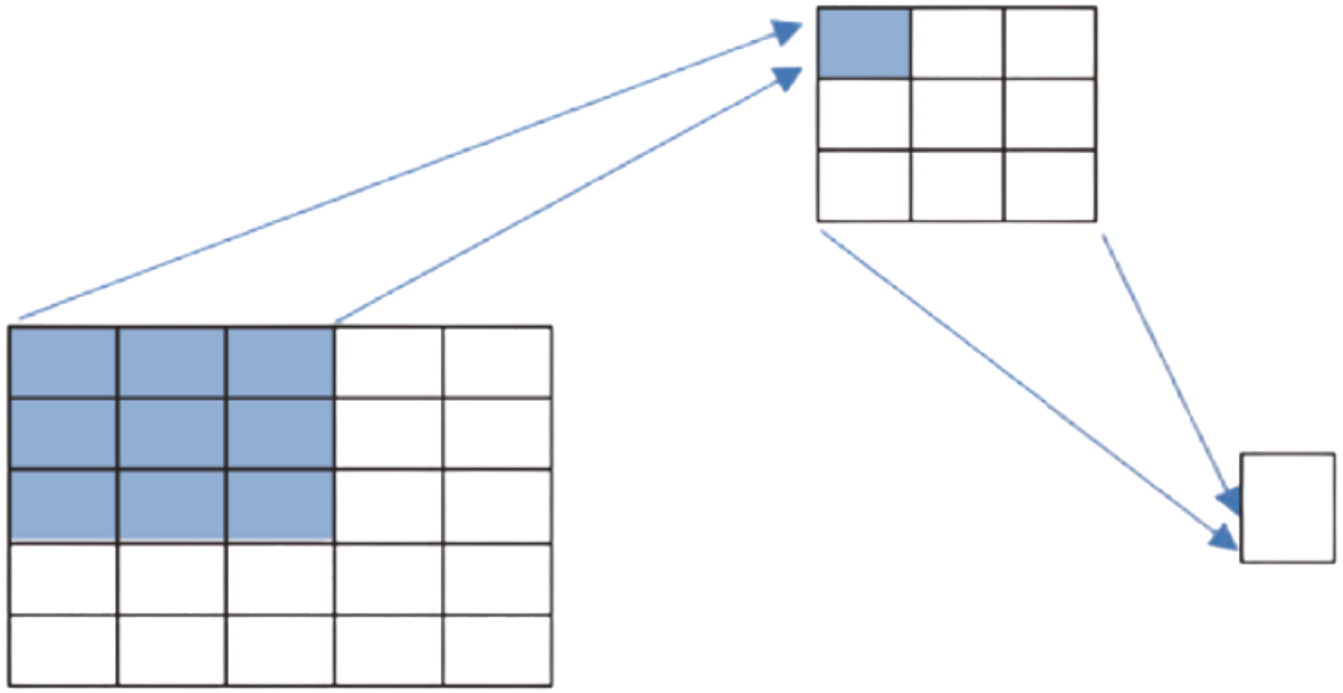
Convolutional Neural Networks

- A CNN is a deep network of neurons with learnable filters that perform convolution operations on inputs, usually images, to extract hierarchical features.



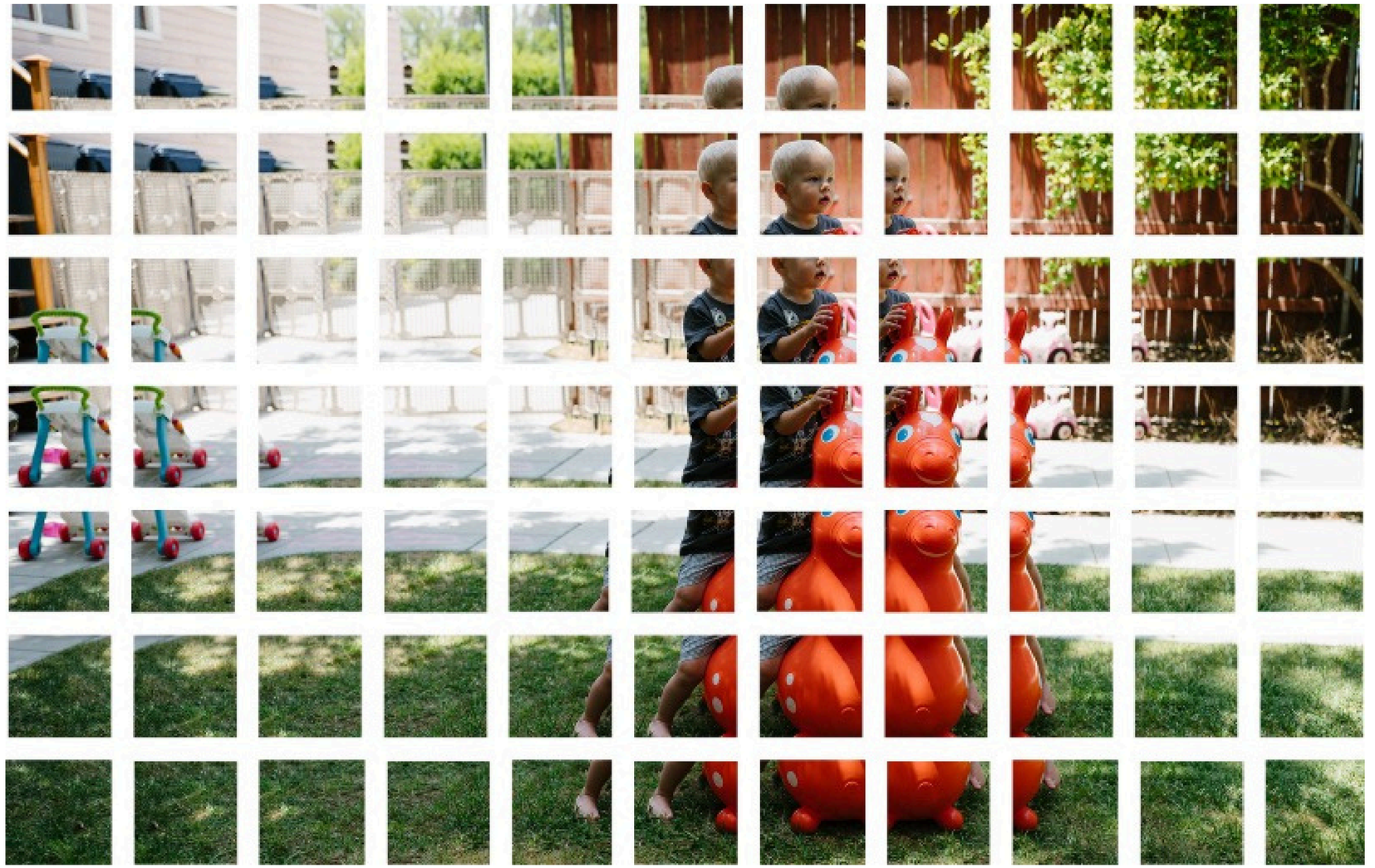
Convolutional Neural Networks

- It applies a set of learnable filters **(kernels)** that slide across the input, performing convolution operations to extract local patterns like edges, textures, and shapes.



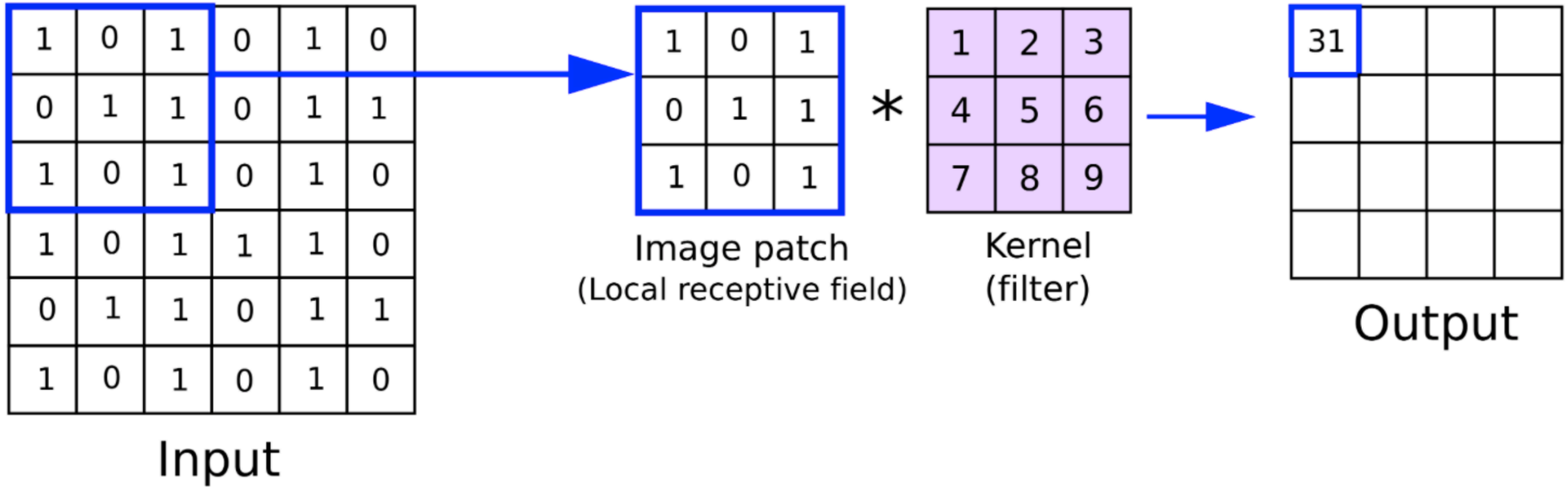
Convolutional Neural Networks

Convolutional Neural Networks



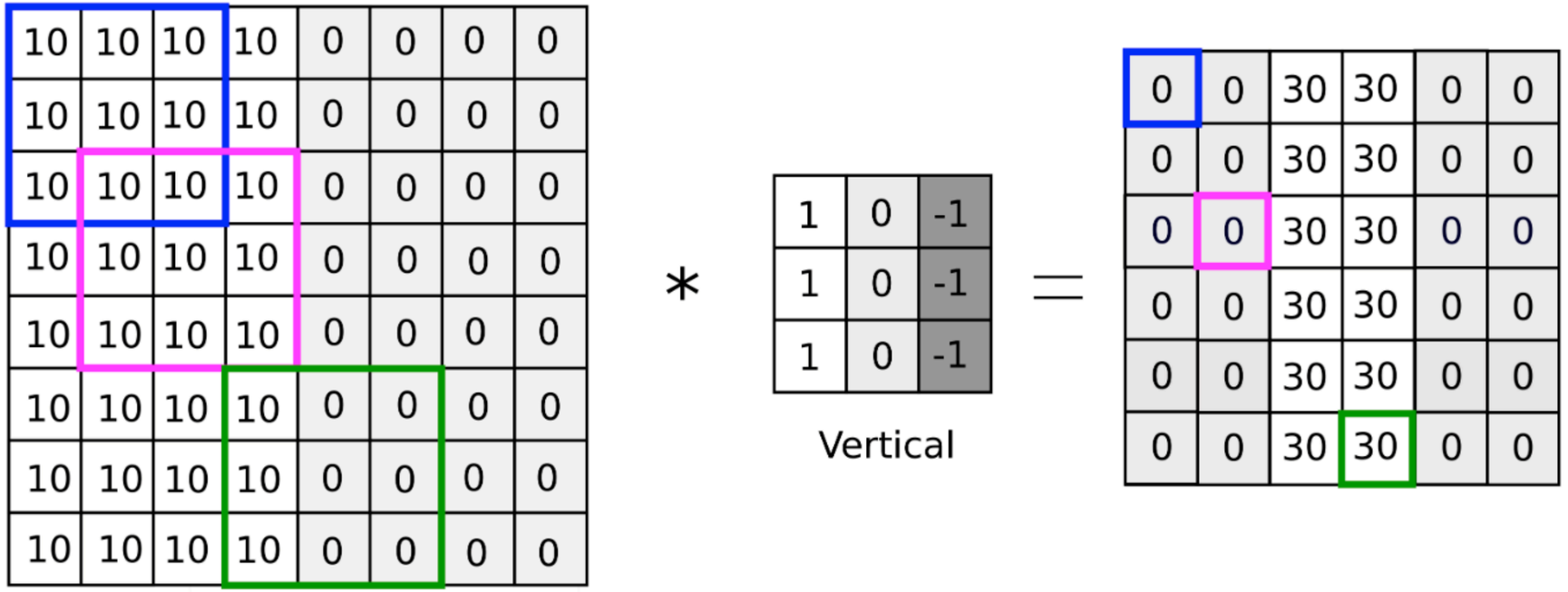
Convolution Layer

- Convolution involves applying a kernel (filter) over an image to extract features.
- A kernel is a small matrix (like: 3×3, 5×5).



Convolution Layer

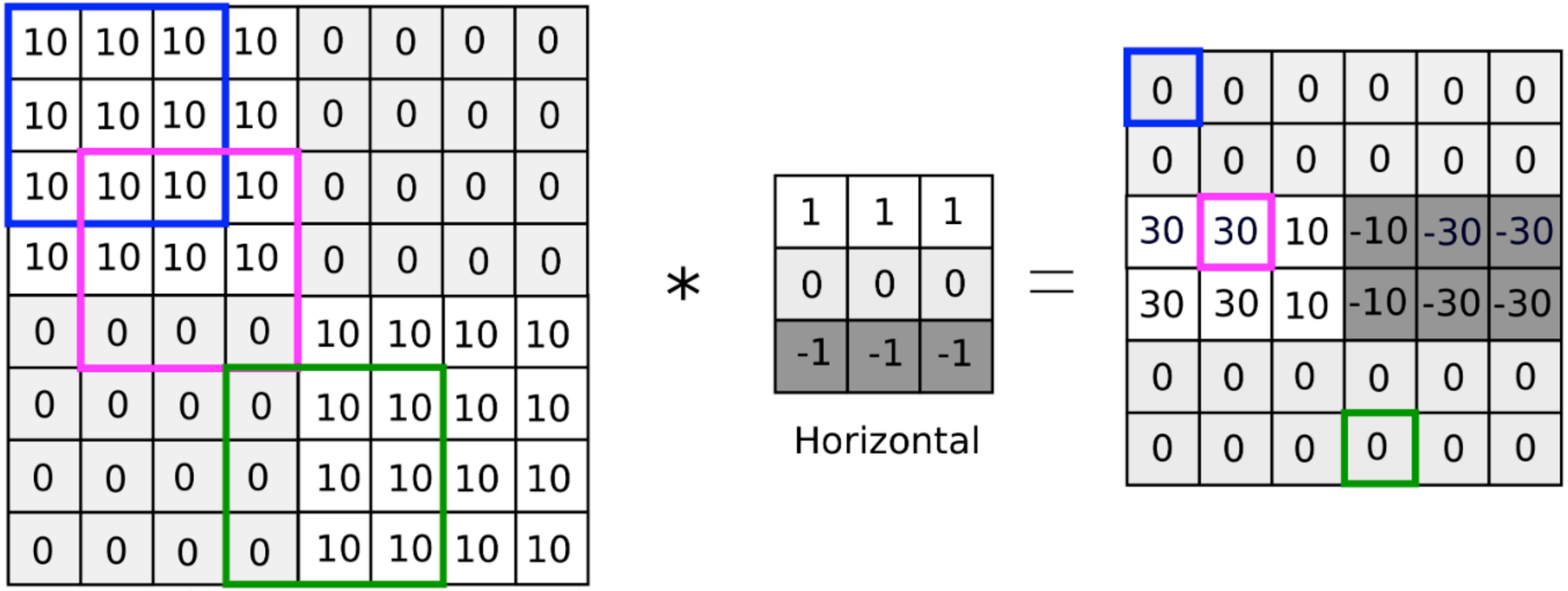
- Edge detection is an example of the usefulness of convolution in identifying the regions in an image where there is a sharp change in colors or intesntiy.



An example of vertical edge detection

Convolution Layer

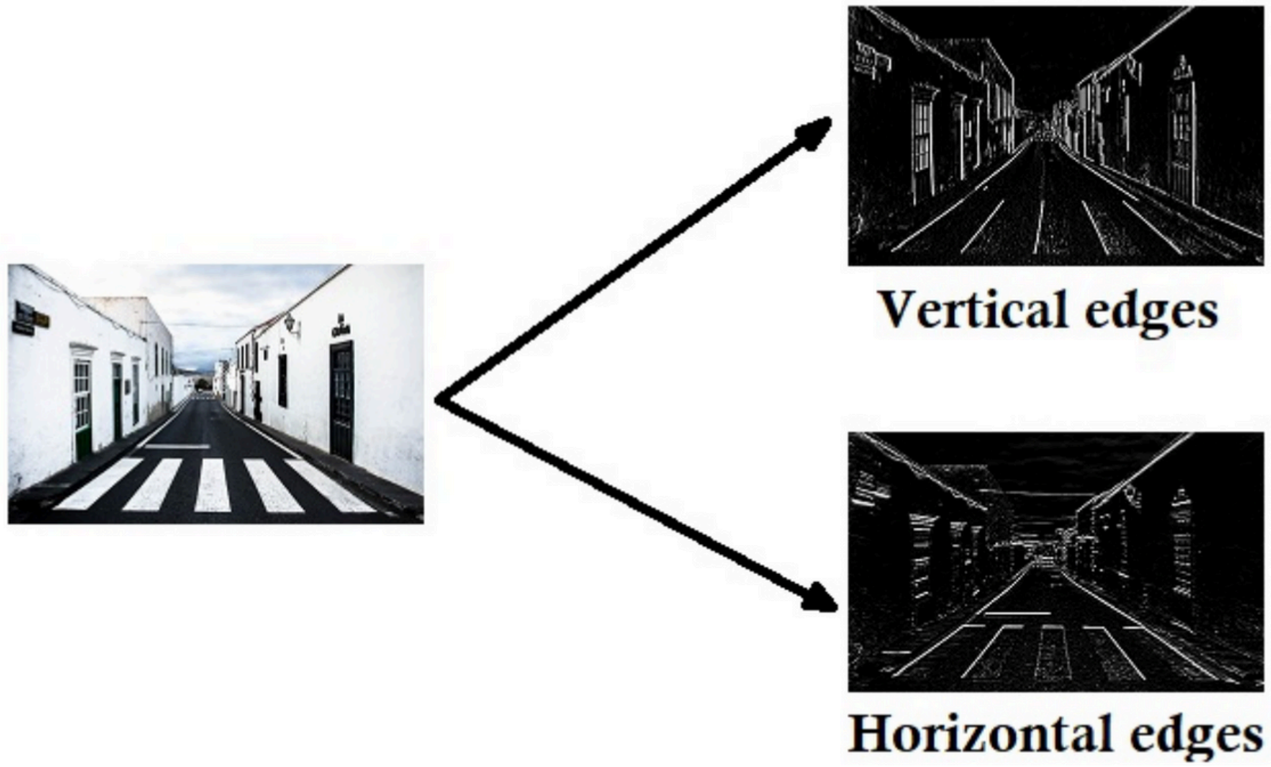
- Edge detection is an example of the usefulness of convolution in identifying the regions in an image where there is a sharp change in colors or intesntiy.



An example of horizontal edge detection

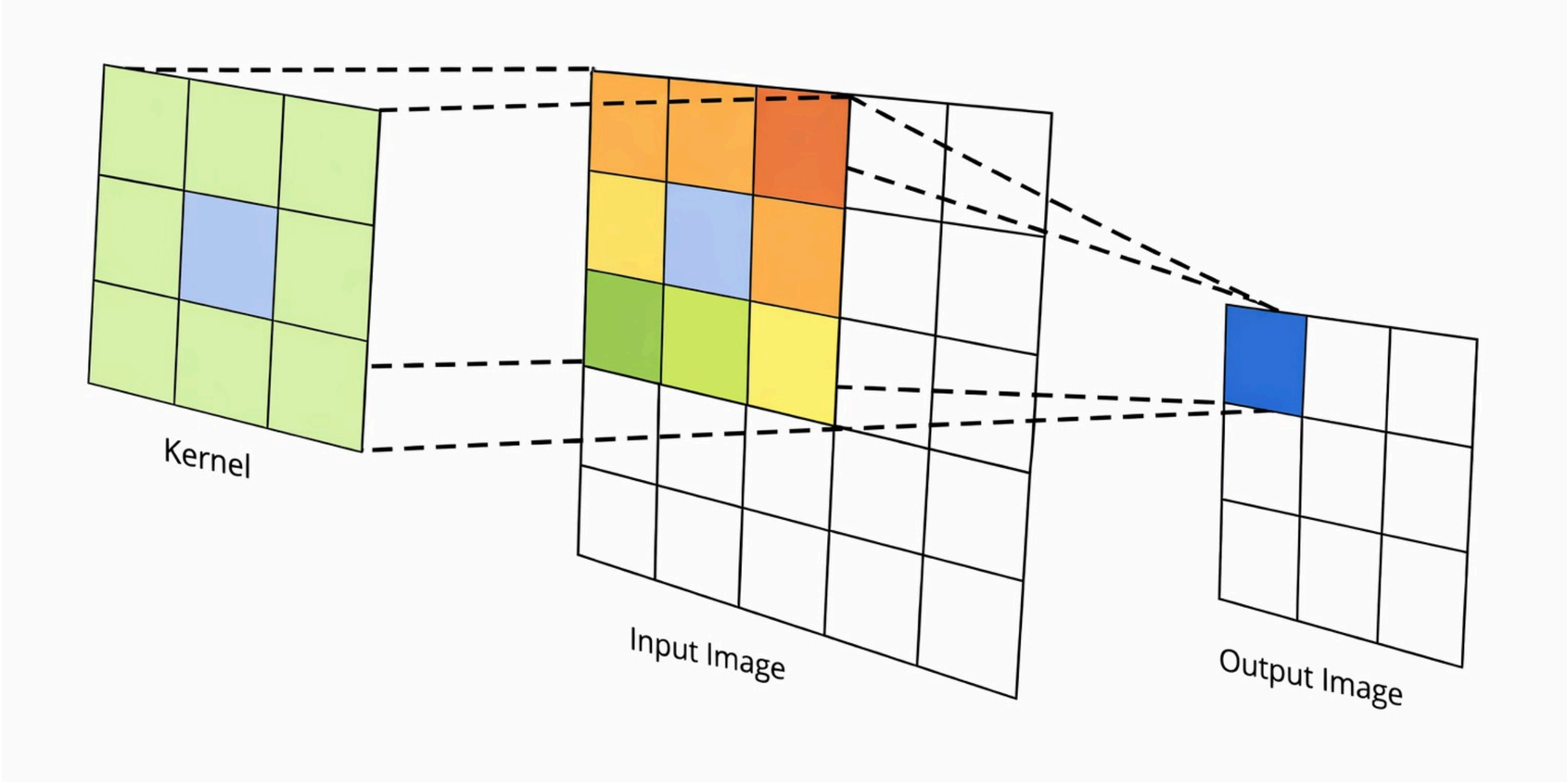
Convolution Layer

- Edge detection is an example of the usefulness of convolution in identifying the regions in an image where there is a sharp change in colors or intesntiy.



Convolution Layer

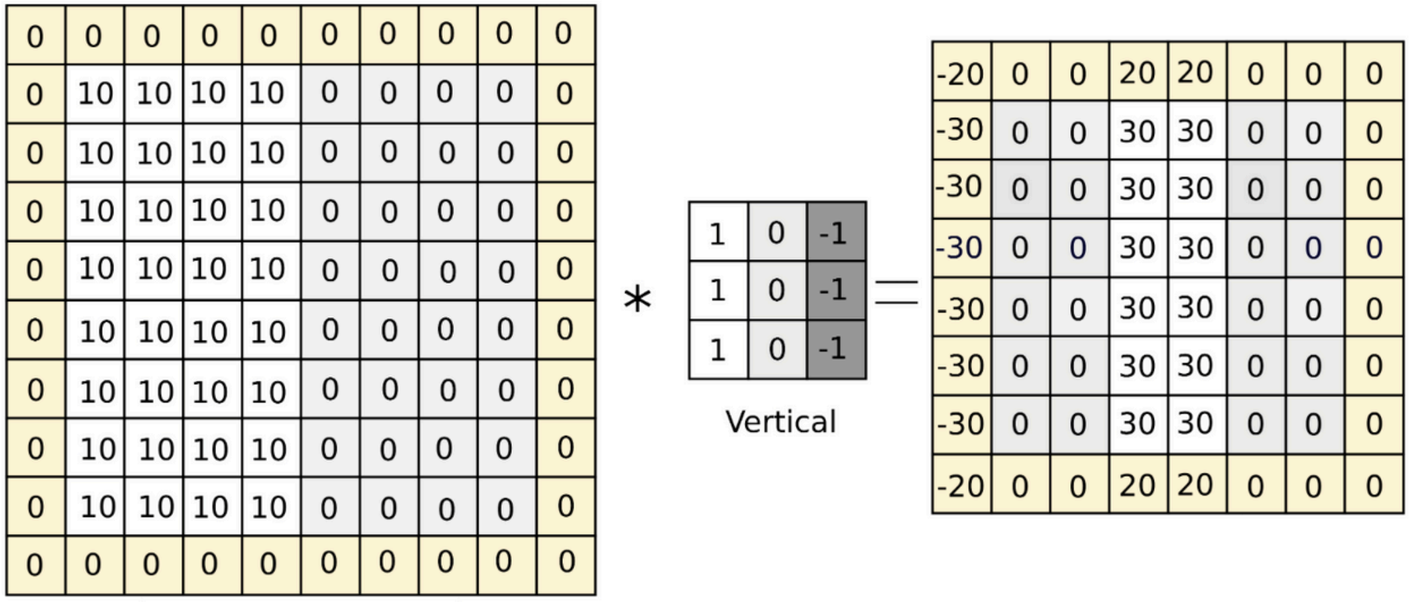
- The kernel is slid across the image, computing a dot product with local regions at each position.



Padding

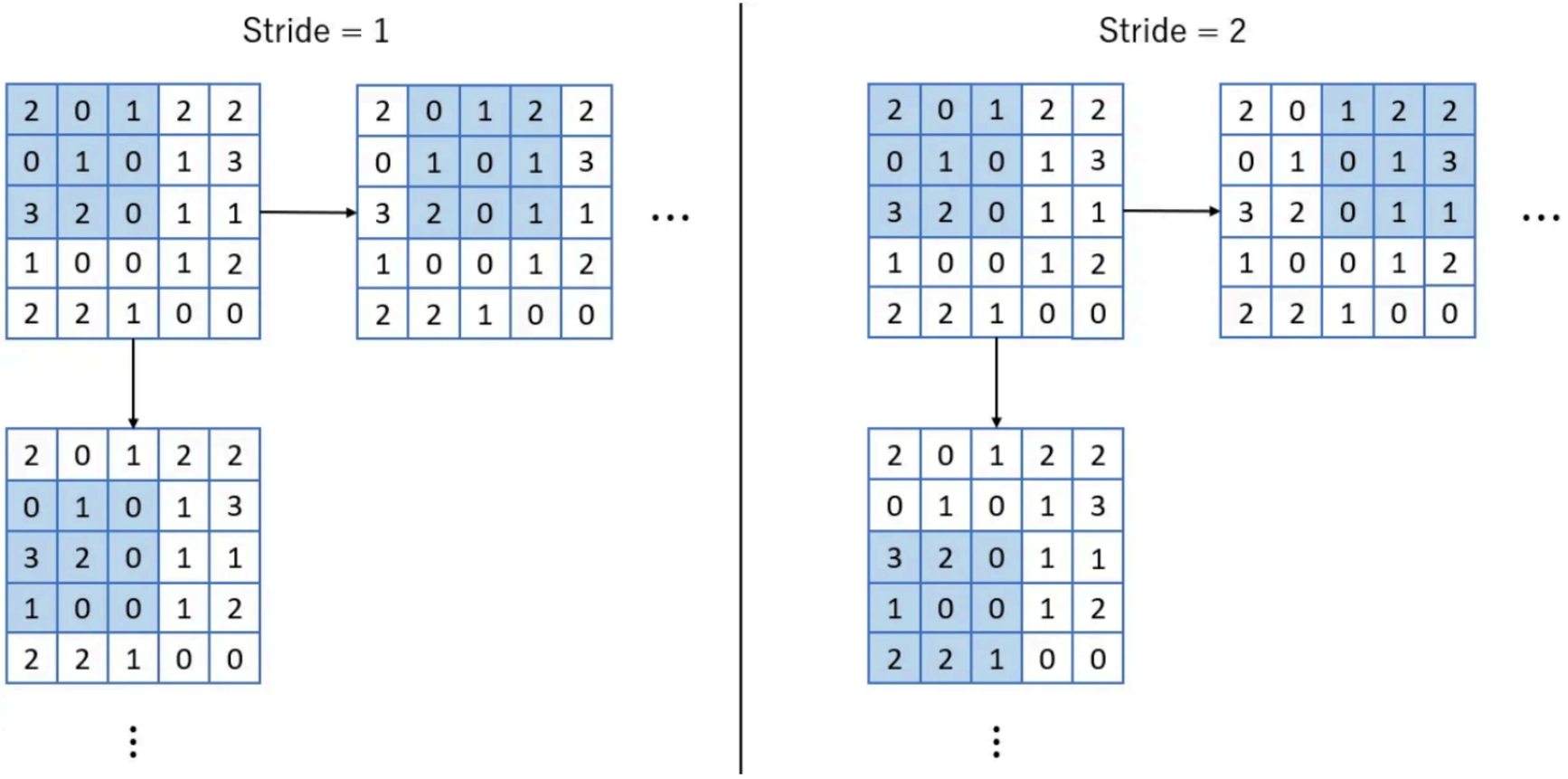
- Padding means adding extra layers of zeros around the image borders.
- If padding size = p, new input dimension becomes:

$$(n + 2p) \times (n + 2p)$$



Stride

- Stride refers to the speed at which a filter translates across an image.



Output Dimensions in Convolution

$$n_{\text{out}} = \left\lfloor \frac{n_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

n_{in} : number of input features

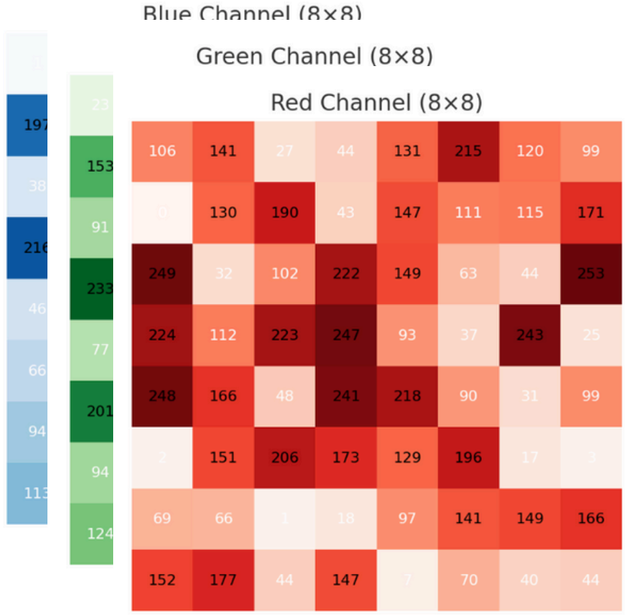
n_{out} : number of output features

k : convolution kernel size

p : convolution padding size

s : convolution stride

Output Dimensions in Convolution



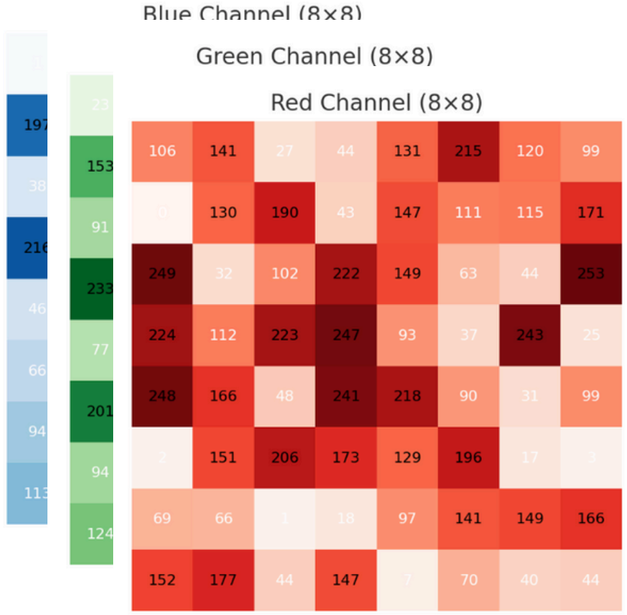
Hight=8

Width=8

Depth=3

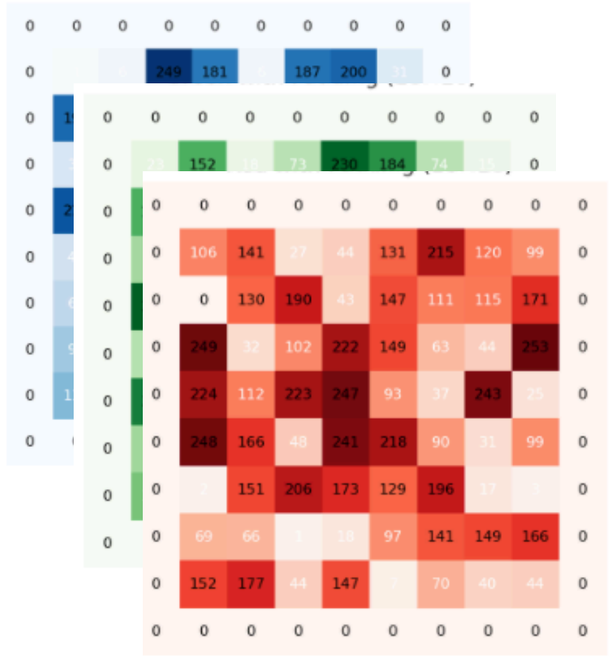
Output Dimensions in Convolution

Input size:



(8*8*3)

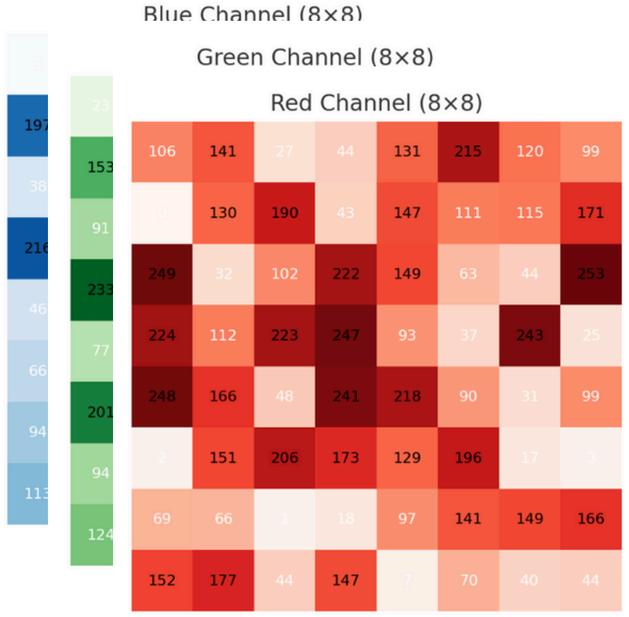
Padding:



(1)

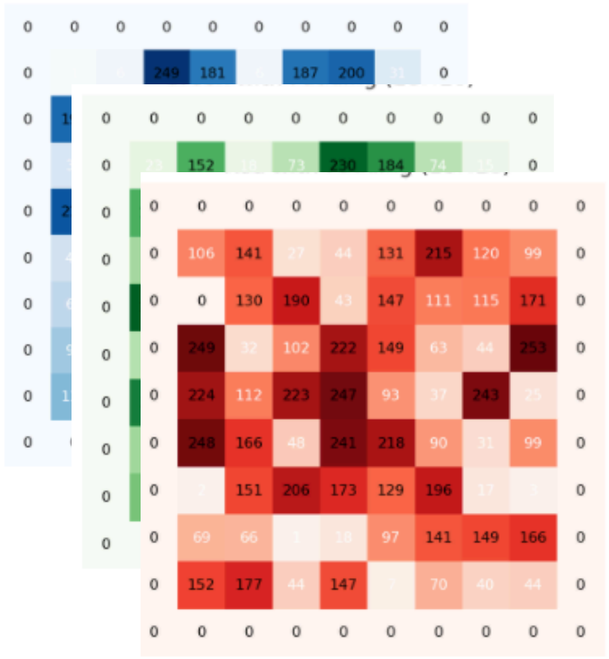
Output Dimensions in Convolution

Input size:



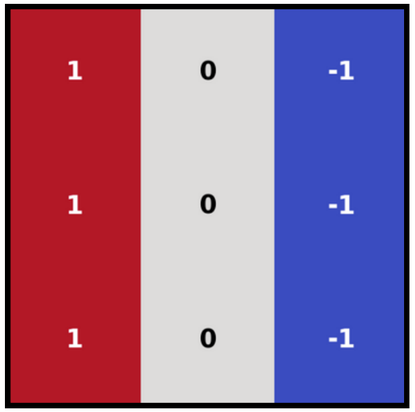
(8*8*3)

Padding:



(1)

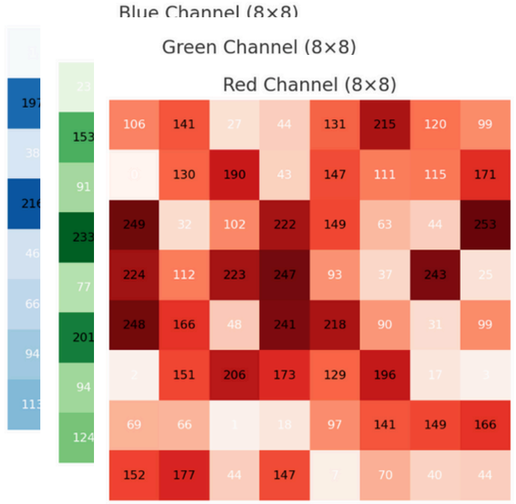
Kernel size:



(3*3)

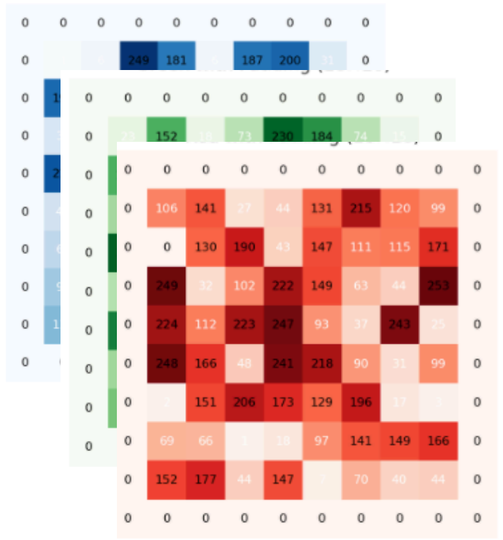
Output Dimensions in Convolution

Input size:



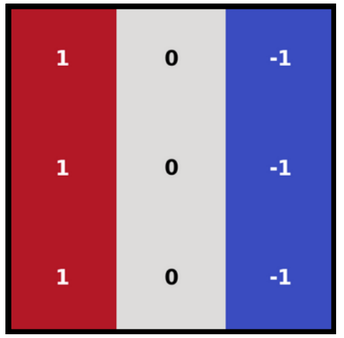
(8*8*3)

Padding:



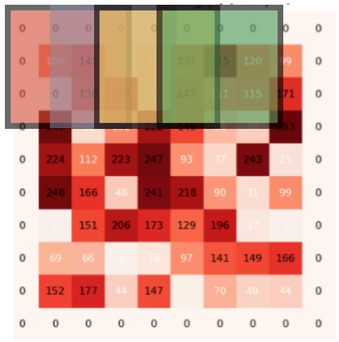
(1)

Kernel size:



(3*3)

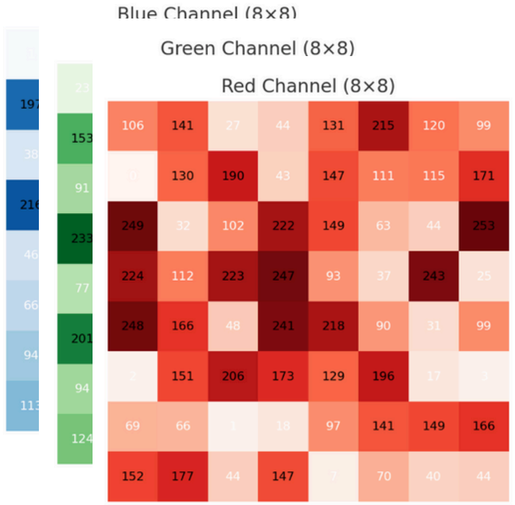
Stride:



(1)

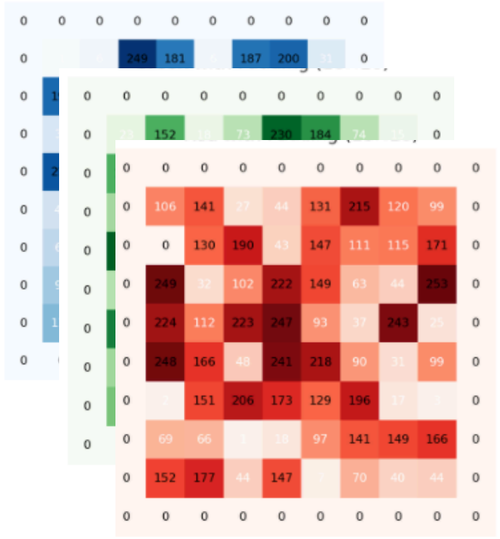
Output Dimensions in Convolution

Input size:



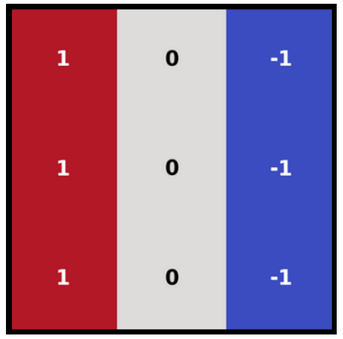
(8*8*3)

Padding:



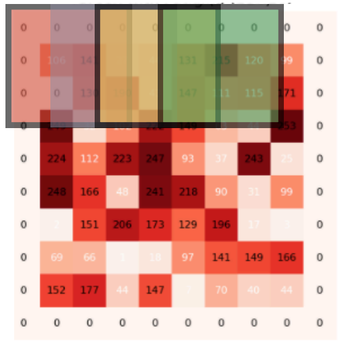
(1)

Kernel size:



(3*3)

Stride:



(1)

$$n_H[l] = \frac{n_H[l-1] + 2p[l] - f[l]}{s[l]} + 1$$

$$n_{out} = \frac{8 + 2(1) - 3}{1} + 1 = \frac{7}{1} + 1 = 8$$

Output Dimensions in Convolution

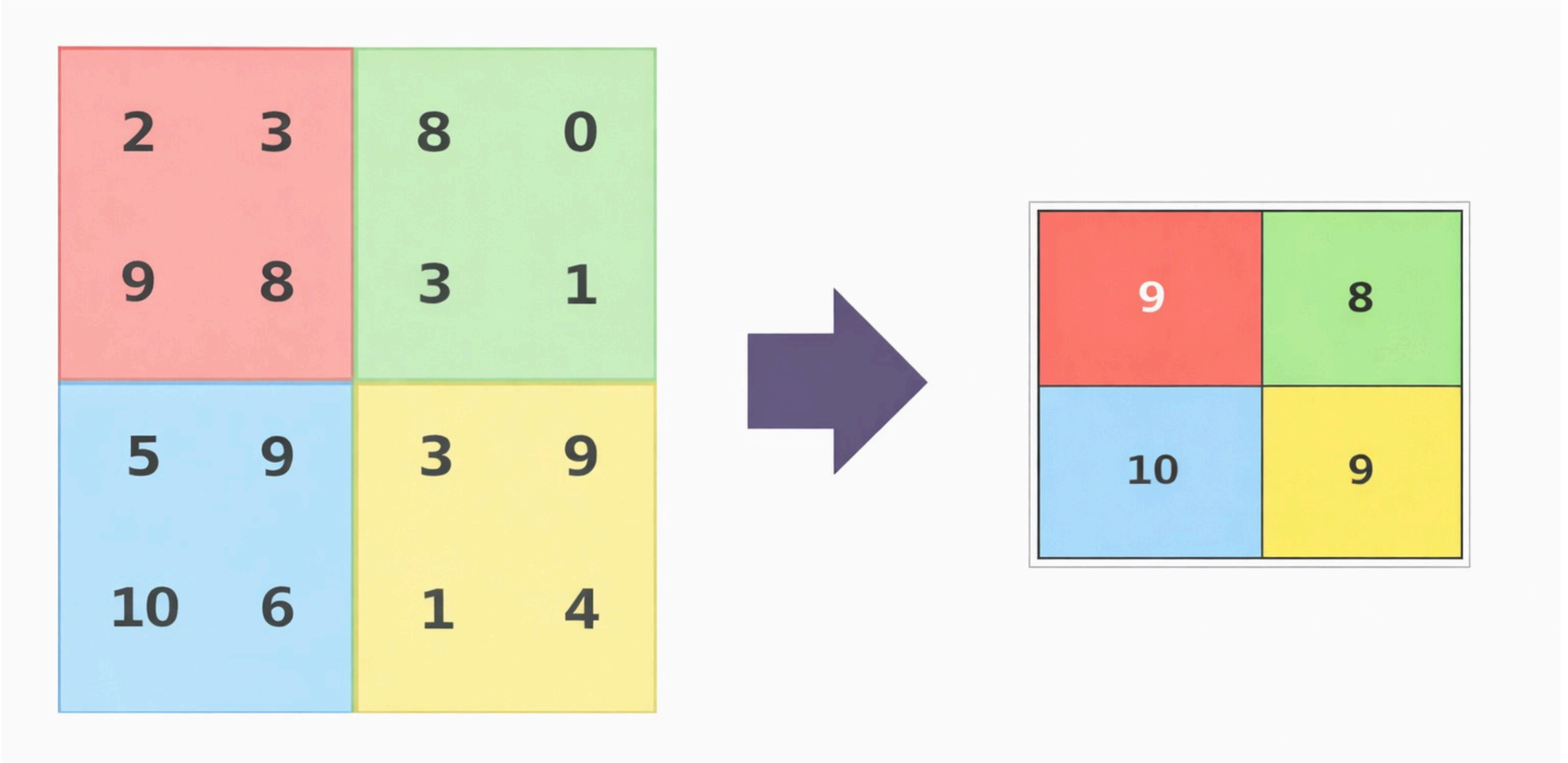
Input image size: 10 × 10 × 3
Kernel size: 5 × 5
Padding: 2
Stride: 2
Number of filters: 16

$$n_{\text{out}} = \left\lfloor \frac{n_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

Pooling Layer

- Pooling is used for dimensionality reduction in CNNs.
  It reduces the spatial size of feature maps while keeping important information.

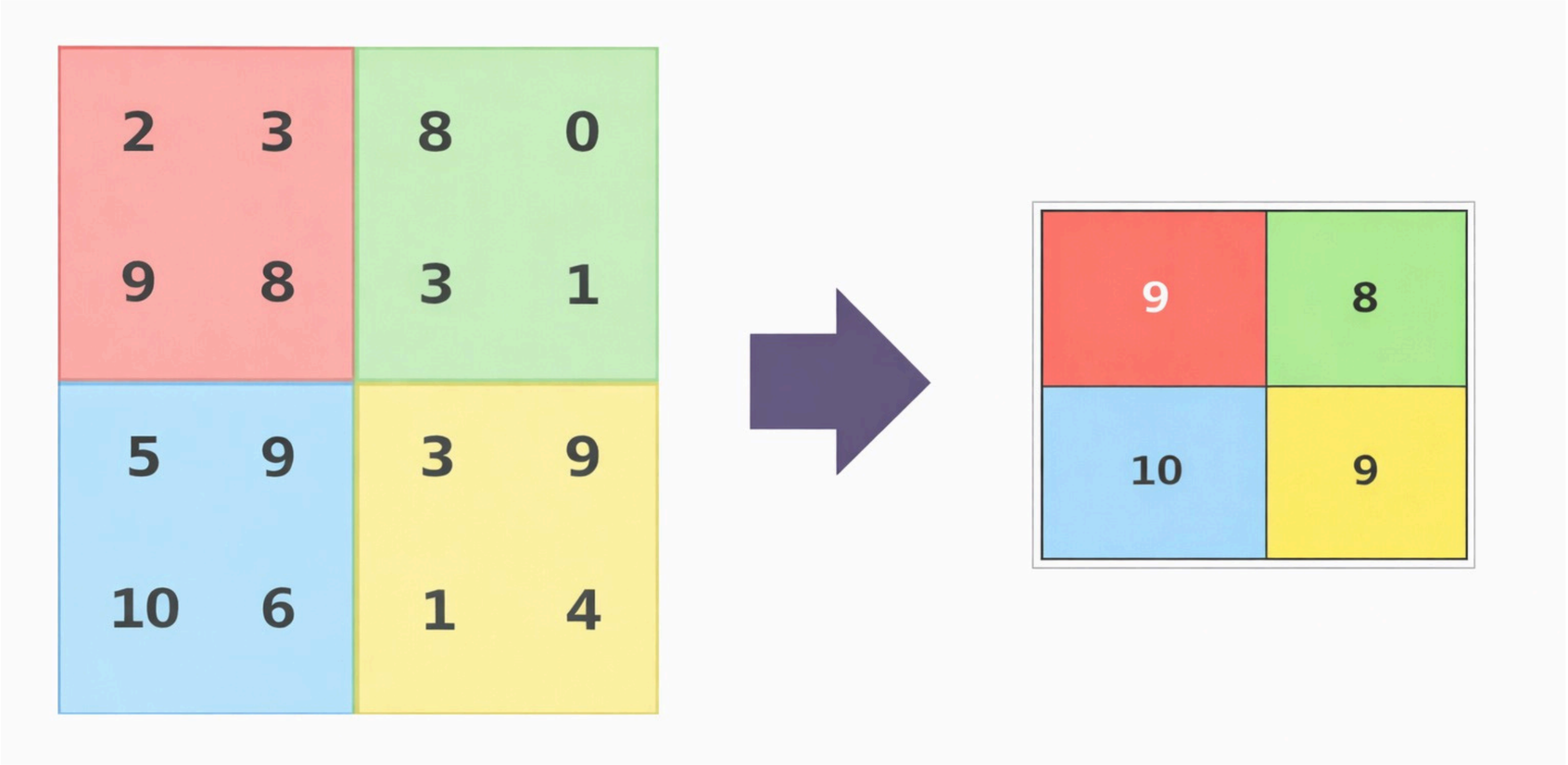
- Max Pooling



Pooling Layer

- Pooling is used for dimensionality reduction in CNNs.
  It reduces the spatial size of feature maps while keeping important information.

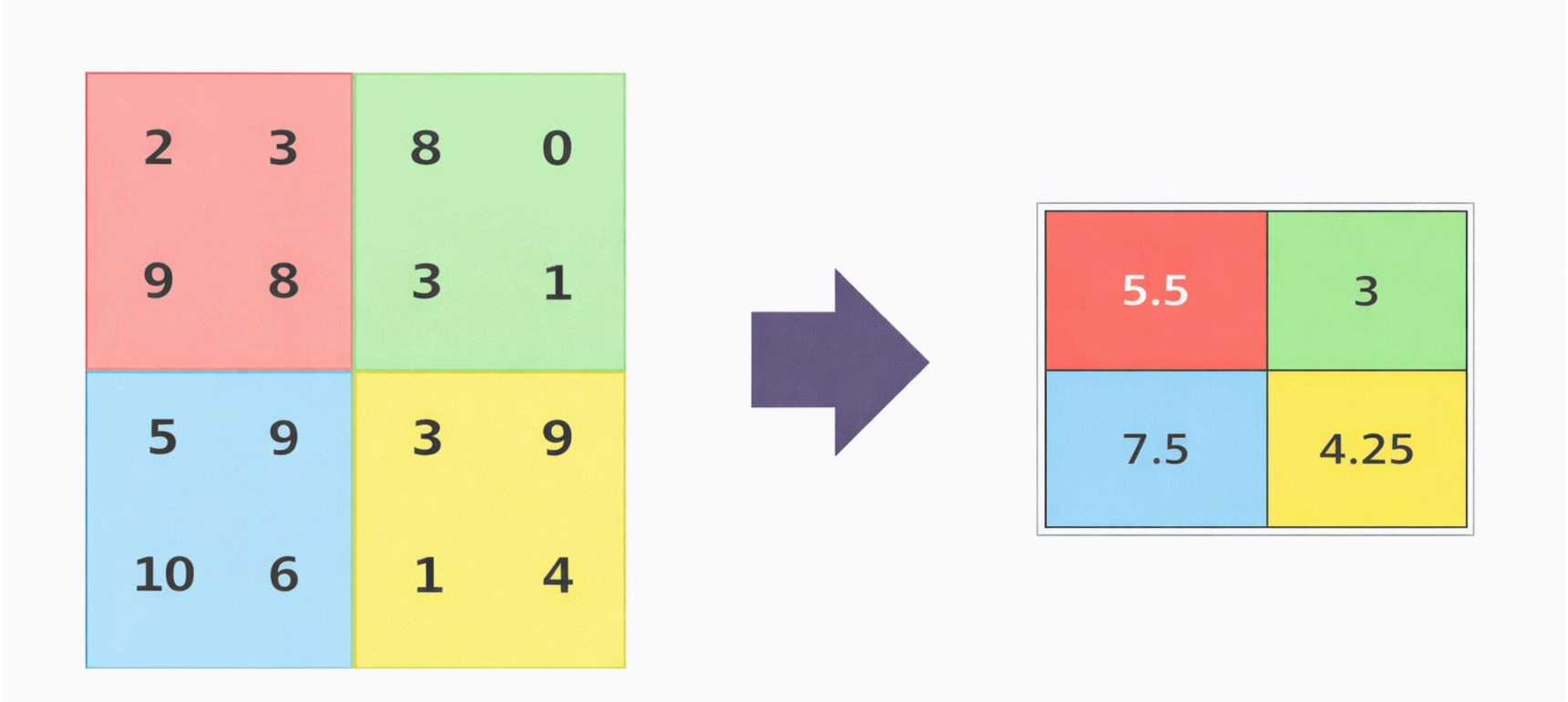
- Max Pooling



Pooling Layer

- Pooling is used for dimensionality reduction in CNNs.
  It reduces the spatial size of feature maps while keeping important information.

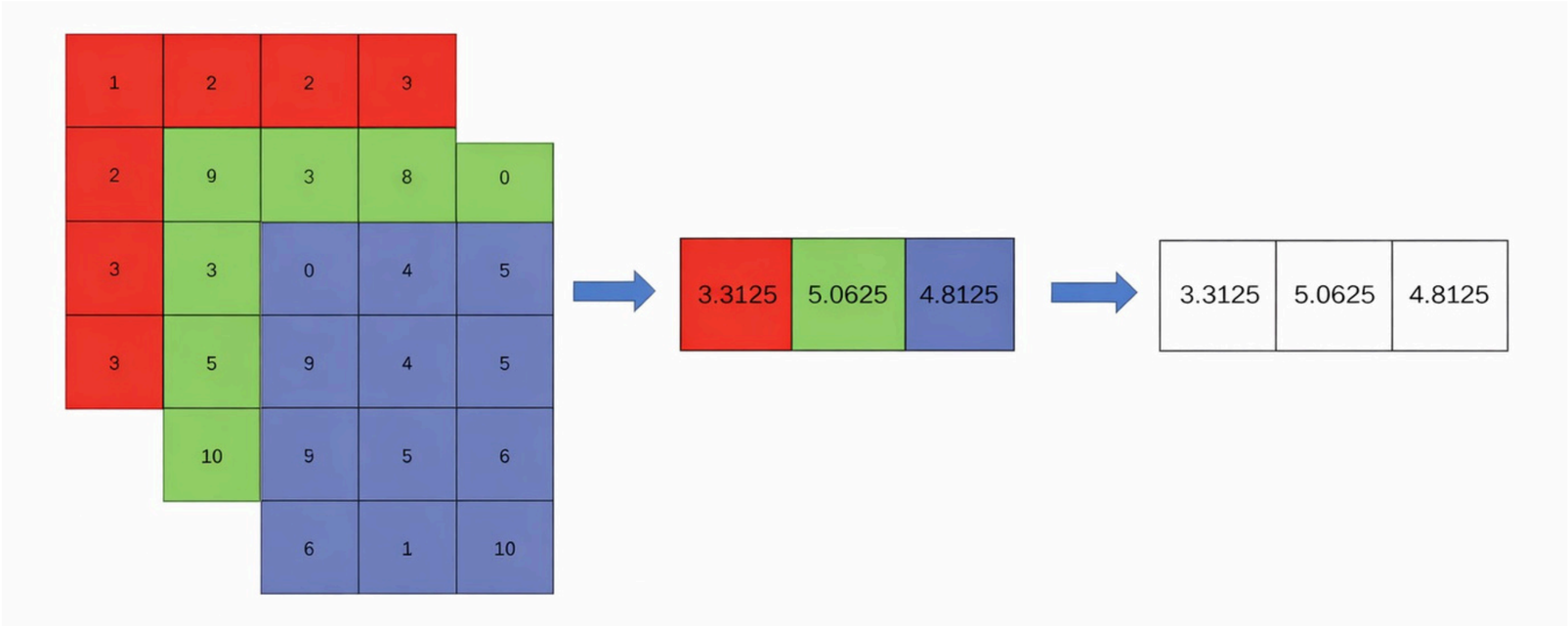
- Average Pooling



Pooling Layer

- Pooling is used for dimensionality reduction in CNNs.
  It reduces the spatial size of feature maps while keeping important information.

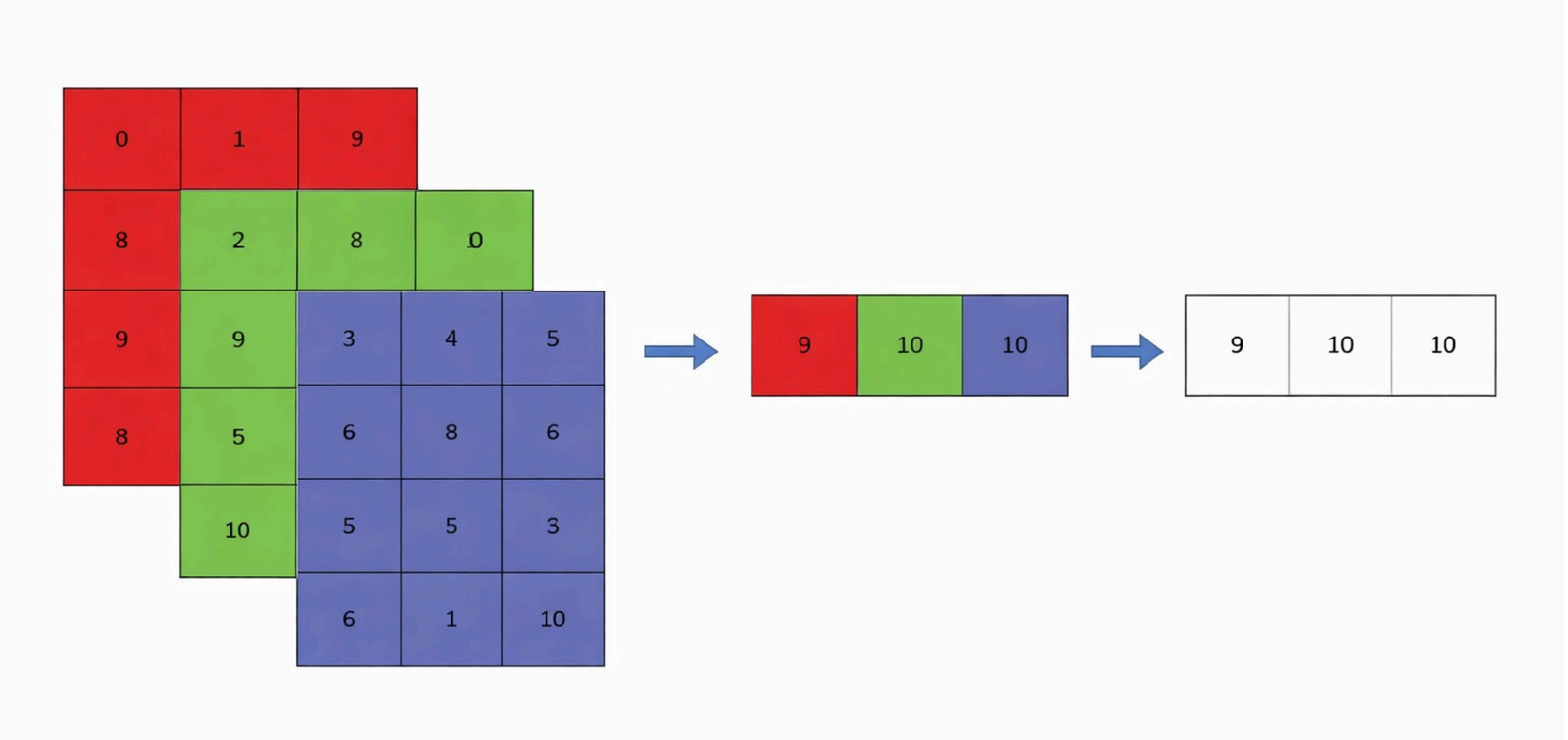
- Global Average Pooling



Pooling Layer

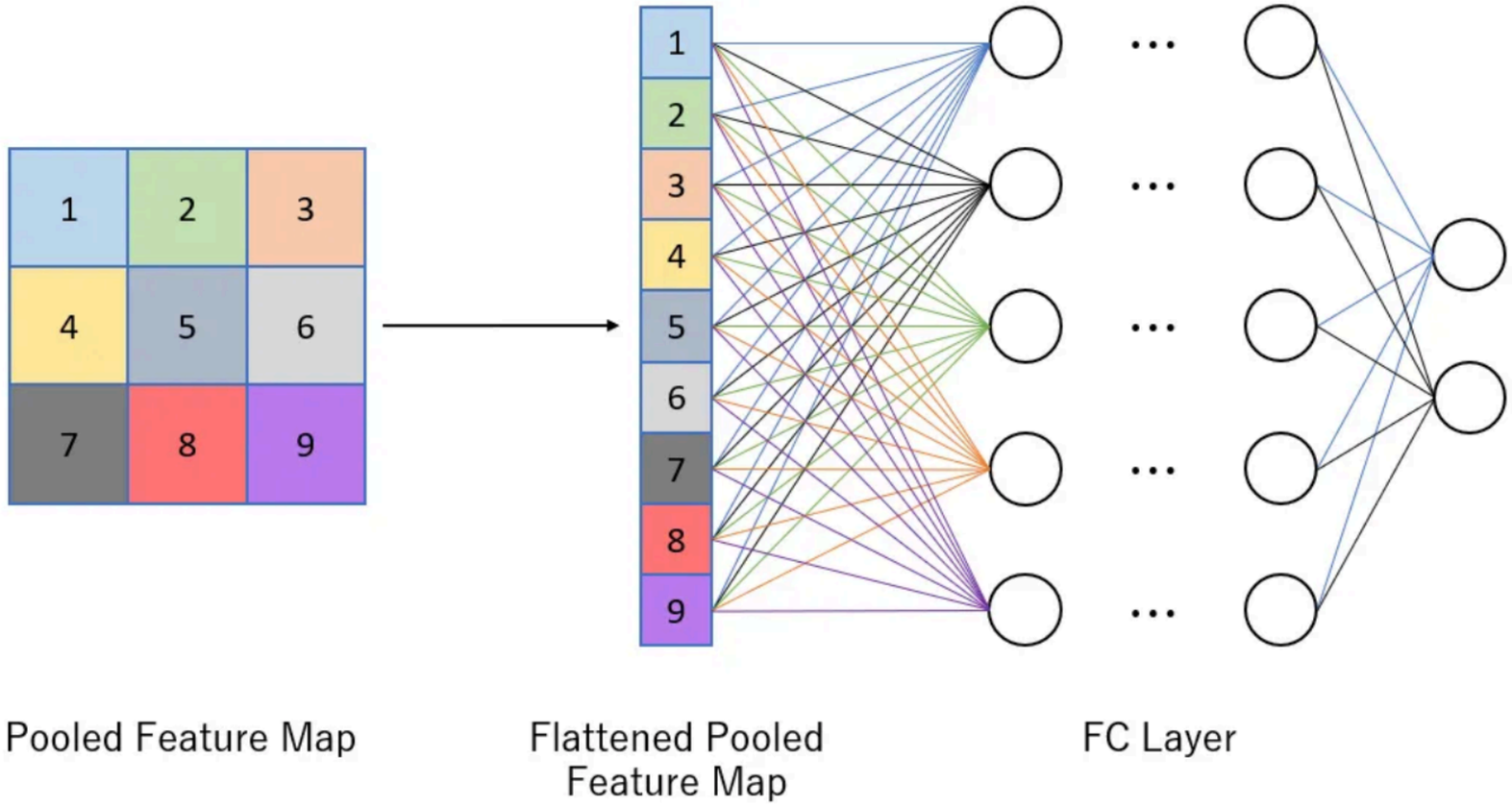
- Pooling is used for dimensionality reduction in CNNs.
  It reduces the spatial size of feature maps while keeping important information.

- Global Max Pooling



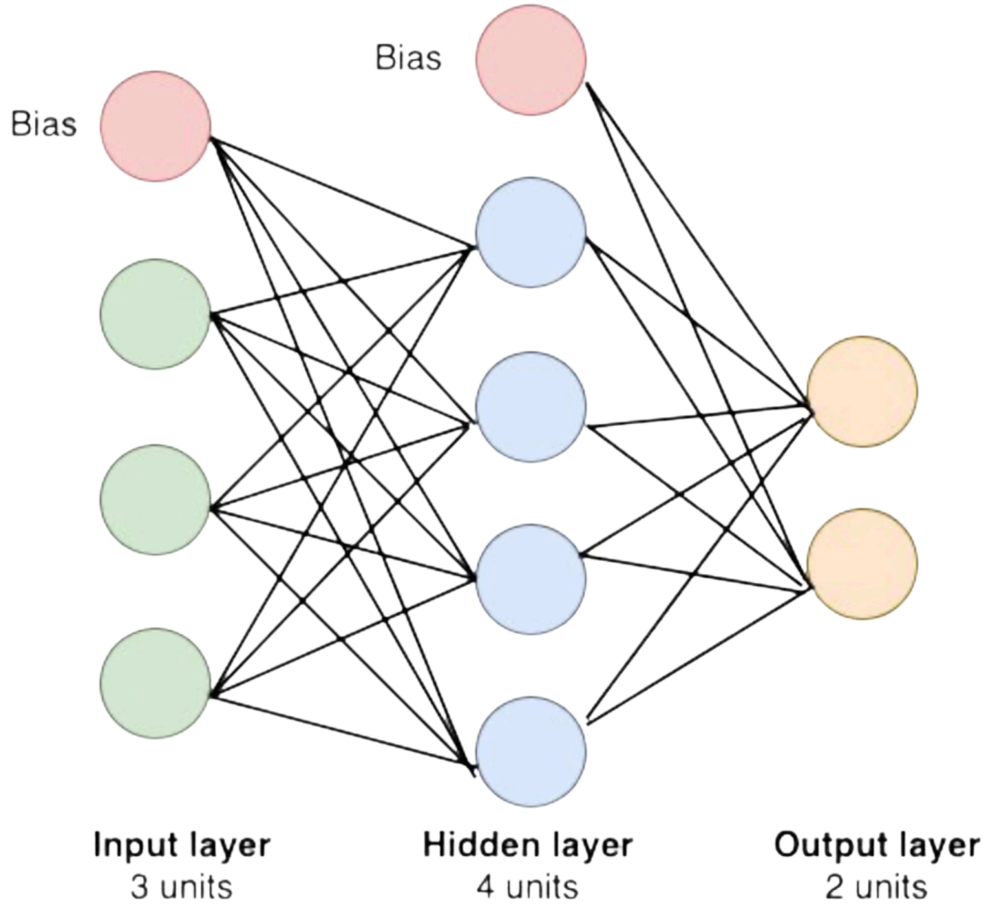
Dense Layer

- It combines the features learned by earlier convolutional and pooling layers to perform high-level reasoning or classification.



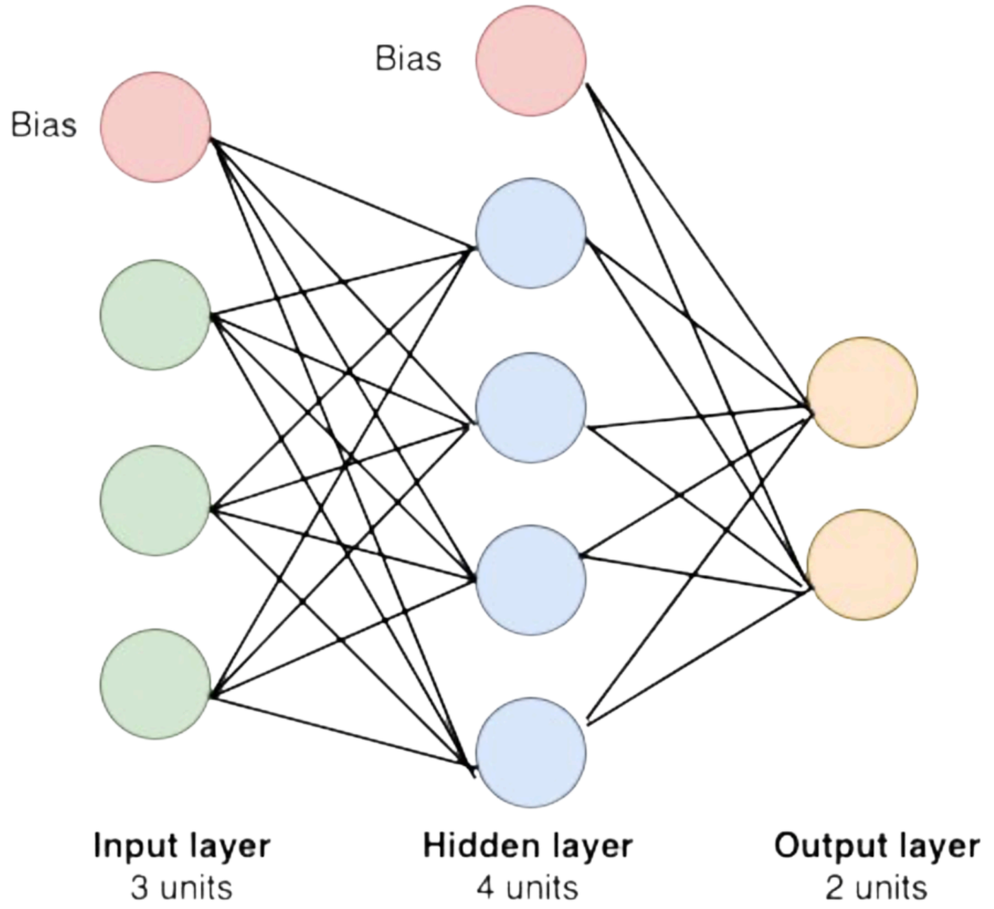
Learnable Parameters in NN

Learnable Parameters in a Neural Network: $= (n_{\text{in}} \times n_{\text{out}}) + n_{\text{out}}$



Learnable Parameters in NN

Learnable Parameters in a Neural Network: $= (n_{\text{in}} \times n_{\text{out}}) + n_{\text{out}}$



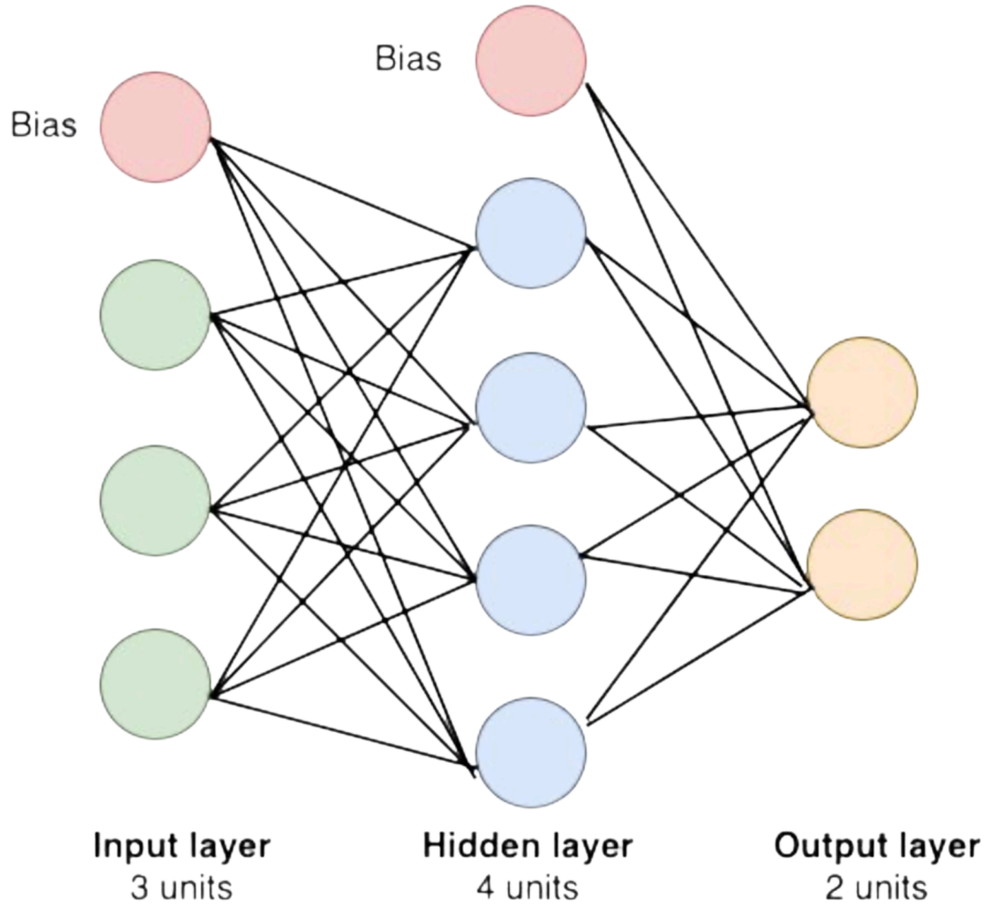
$$\text{Weights} = 3 \times 4 = 12$$

$$\text{Biases} = 4$$

$$\text{Total}_{\text{IH}} = 12 + 4 = \mathbf{16}$$

Learnable Parameters in NN

Learnable Parameters in a Neural Network: $= (n_{\text{in}} \times n_{\text{out}}) + n_{\text{out}}$



$$\text{Weights} = 3 \times 4 = 12$$

$$\text{Biases} = 4$$

$$\text{Total}_{\text{IH}} = 12 + 4 = \mathbf{16}$$

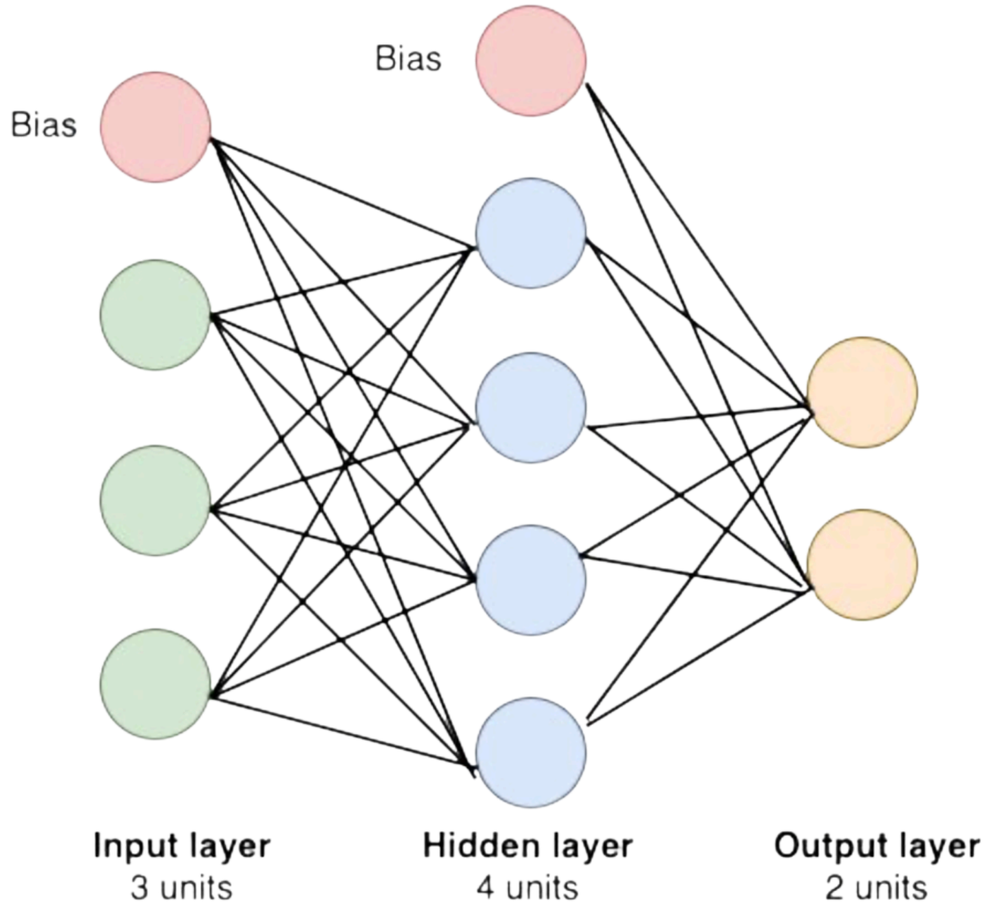
$$\text{Weights} = 4 \times 2 = 8$$

$$\text{Biases} = 2$$

$$\text{Total}_{\text{HO}} = 8 + 2 = \mathbf{10}$$

Learnable Parameters in NN

Learnable Parameters in a Neural Network: $= (n_{\text{in}} \times n_{\text{out}}) + n_{\text{out}}$



$$\text{Weights} = 3 \times 4 = 12$$

$$\text{Biases} = 4$$

$$\text{Total}_{\text{IH}} = 12 + 4 = \mathbf{16}$$

$$\text{Weights} = 4 \times 2 = 8$$

$$\text{Biases} = 2$$

$$\text{Total}_{\text{HO}} = 8 + 2 = \mathbf{10}$$

$$\text{Total} = 16 + 10 = \mathbf{26}$$

Learnable Parameters in CNN

Learnable Parameters in a CNN:

$$\textbf{Params} = (K_h \times K_w \times C_{in} + 1) \times C_{out}$$

A convolution layer with:

- Kernel size: 3×3
- Input channels: 3 (RGB)
- Output channels: 16 filters

Learnable Parameters in CNN

Learnable Parameters in a CNN:

$$\textbf{Params} = (K_h \times K_w \times C_{in} + 1) \times C_{out}$$

A convolution layer with:

- Kernel size: 3×3
- Input channels: 3 (RGB)
- Output channels: 16 filters

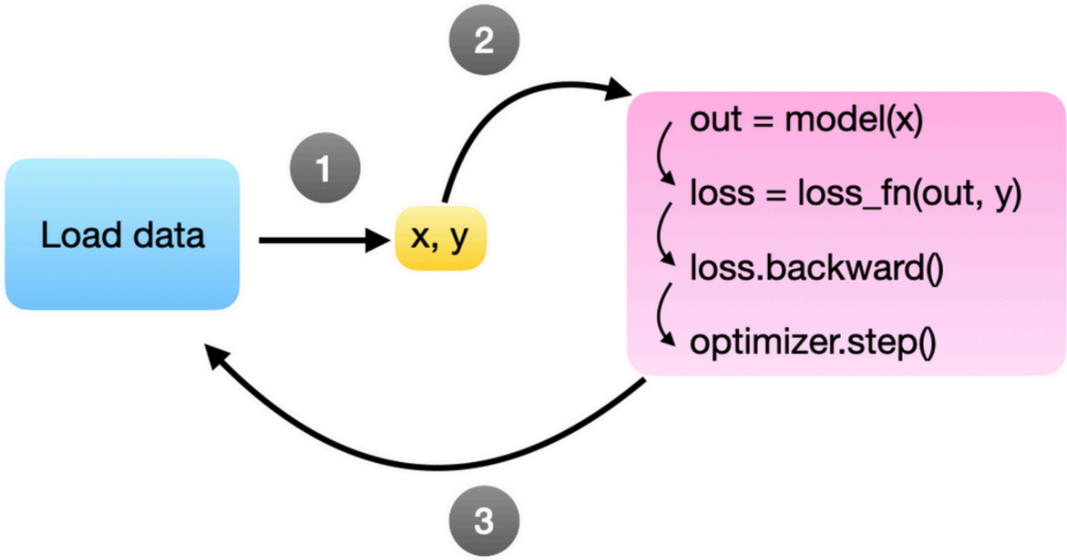
$$(3 \times 3 \times 3 + 1) \times 16$$

$$28 \times 16 = 448$$

Efficient Data Loading

- Training speed depends not only on the model
- GPU should never wait for data

```
for epoch in range(num_epochs):
    inputs = []
    for f in filename_minibatch:
        img = Image.open(os.path.join(img_dir, f))
        img_tensor = o_tensor(img).to('cuda')
        inputs.append(img_tensor)
    inputs = torch.cat(inputs)
    logits = model(inputs)
    loss.backward()
    optimizer.step()
```

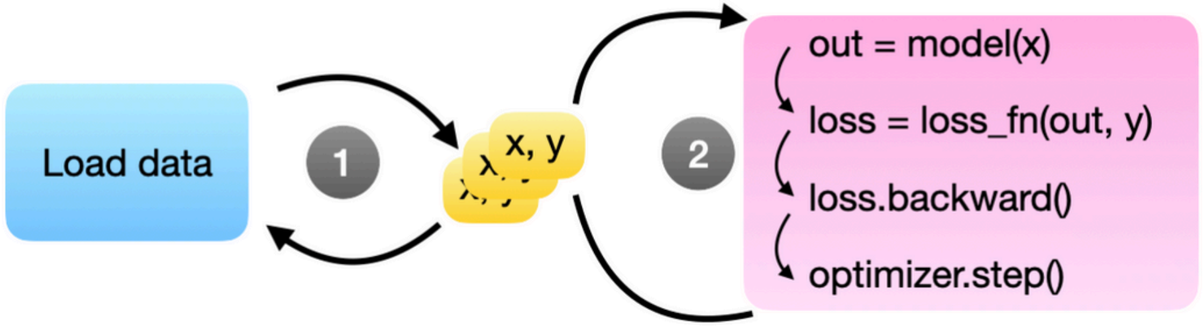


Efficient Data Loading

- Load next mini-batch in background
- Supports multiple worker processes
- Automatic shuffling, batching, and collation

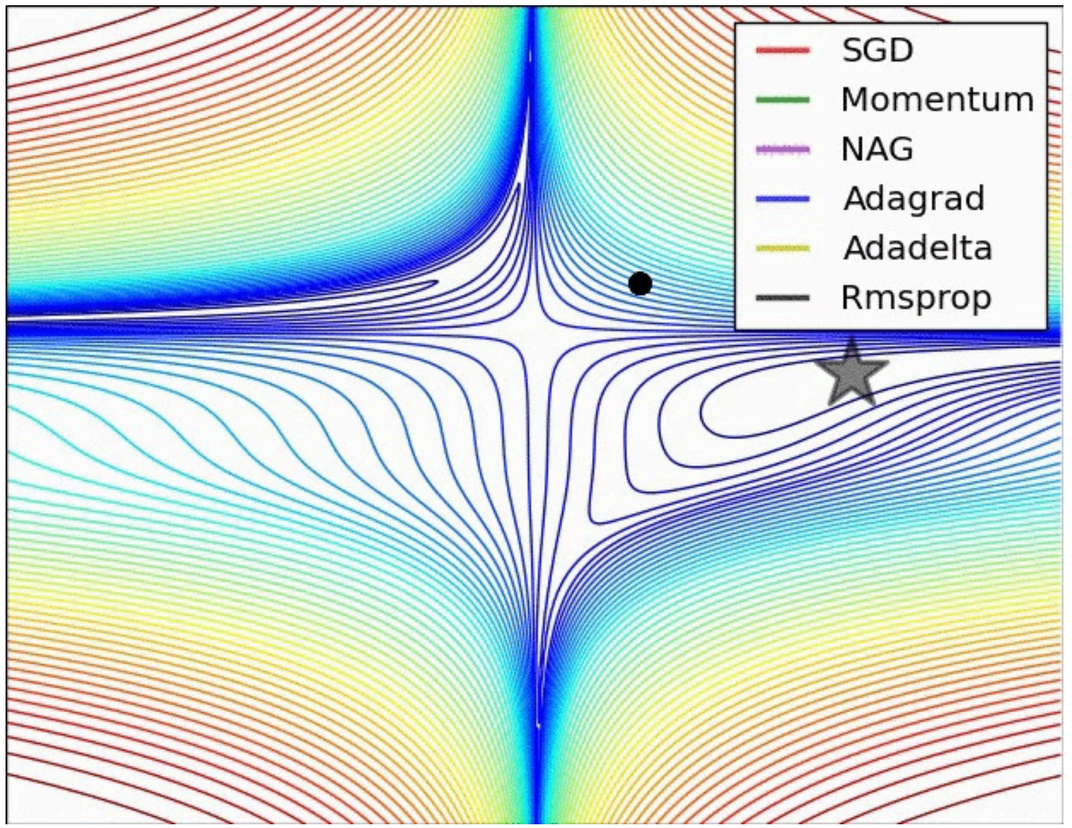
```
from torch.utils.data import Dataset, DataLoader

train_loader = DataLoader(dataset, batch_size=64,
                          shuffle=True, num_workers=4)
```



Optimization

- Optimizers are algorithms that update a model's parameters in order to minimize the loss function during training.

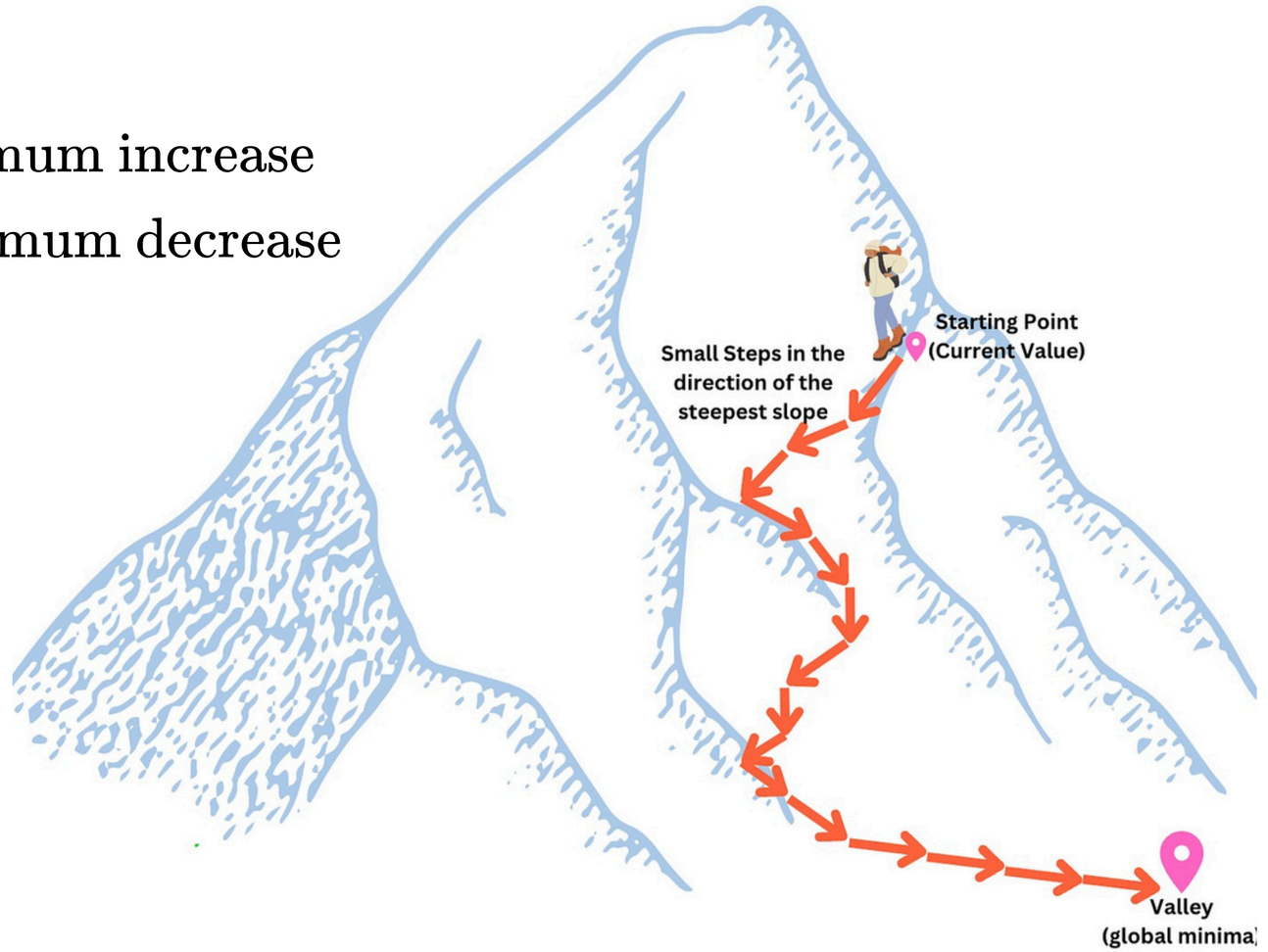


Batch Gradient Descent

- Direction of Maximum Increase and Decrease

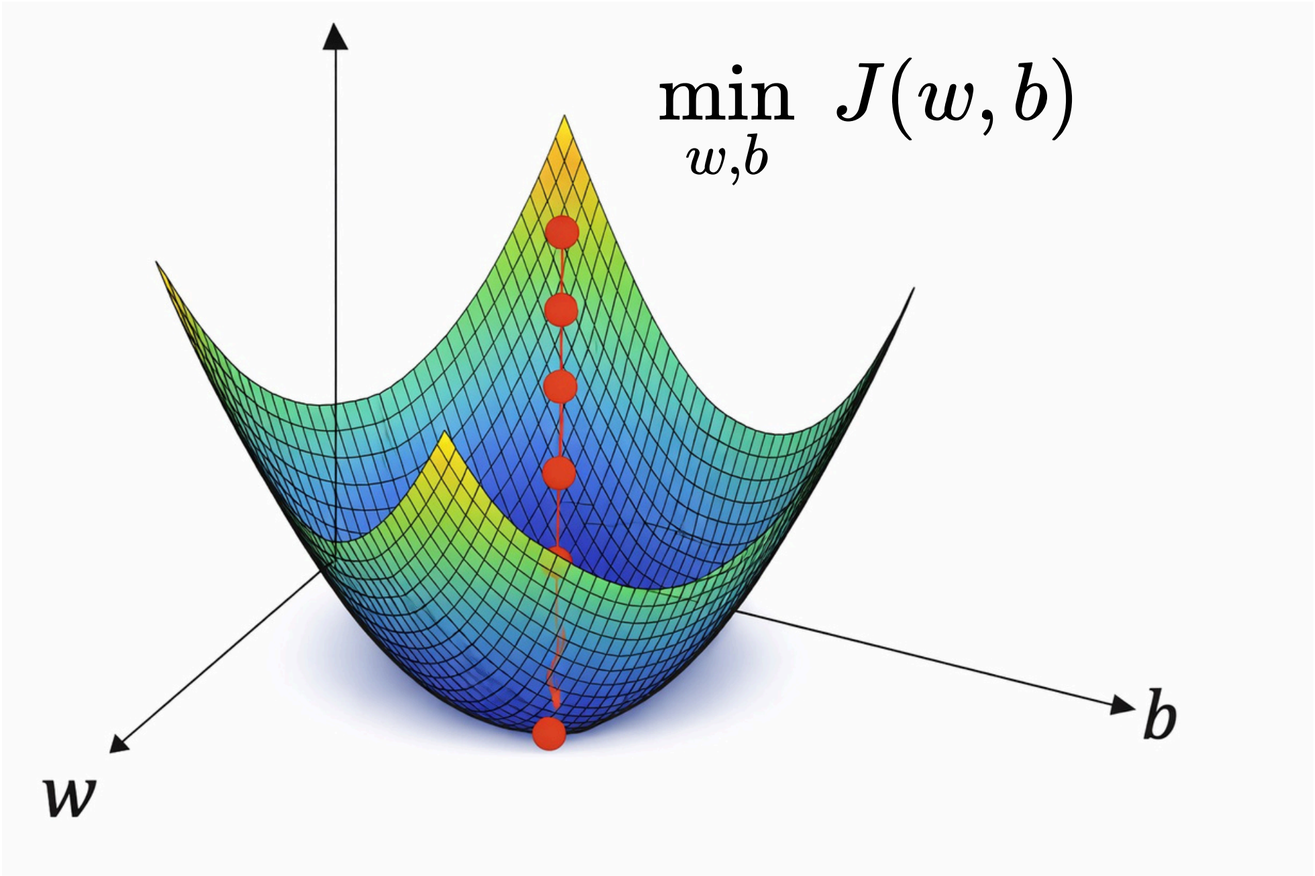
$\nabla f(x)$: direction of maximum increase

$-\nabla f(x)$: direction of maximum decrease



Gradient Descent

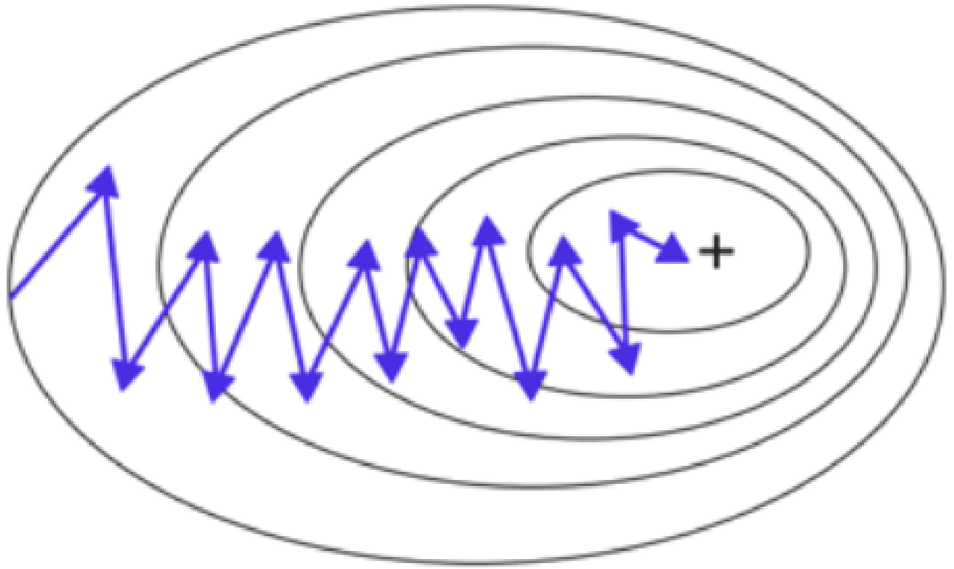
- Moving in the direction of steepest loss decrease



Stochastic Gradient Descent

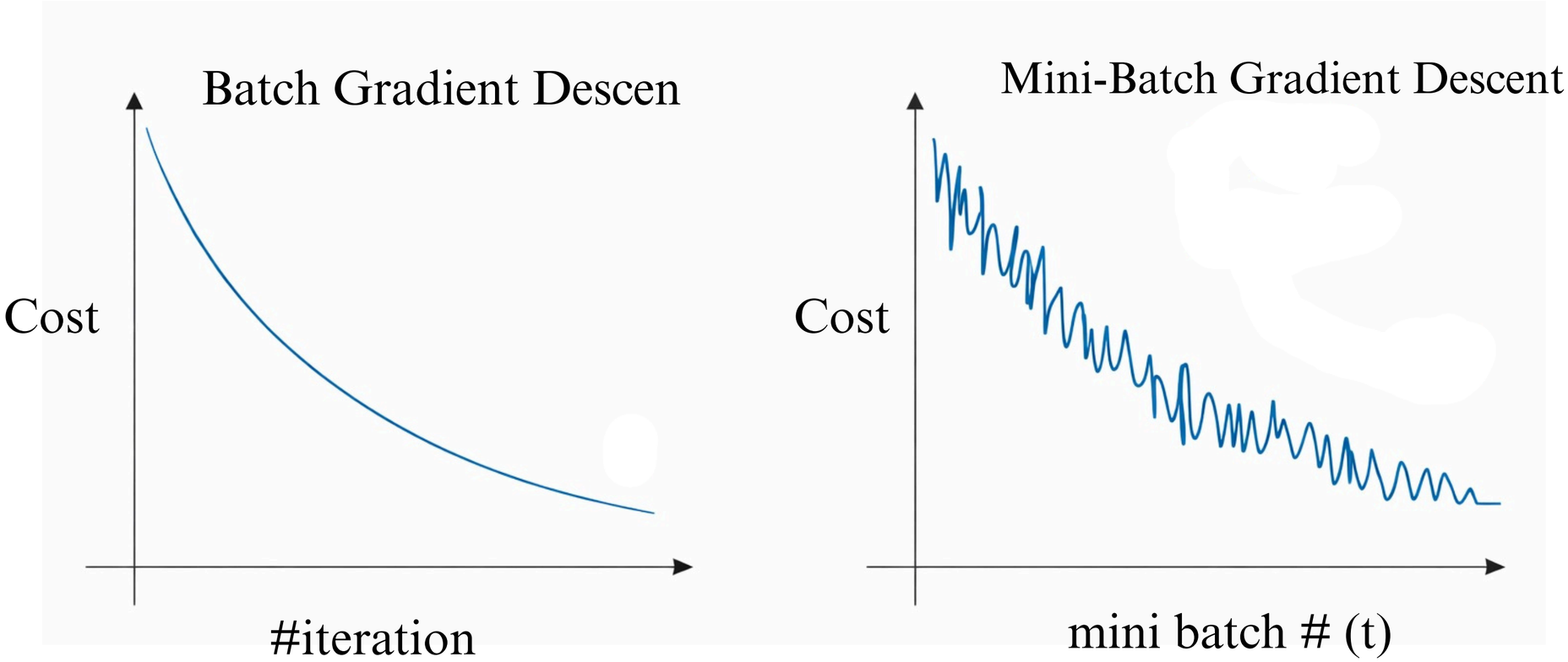
- Updates model parameters using one training sample at a time

$$\theta = \theta - \eta \cdot \nabla L(x_i, y_i)$$



Mini-Batch Gradient Descent

- Instead of using all training examples or just one, we update parameters using a small batch of samples.



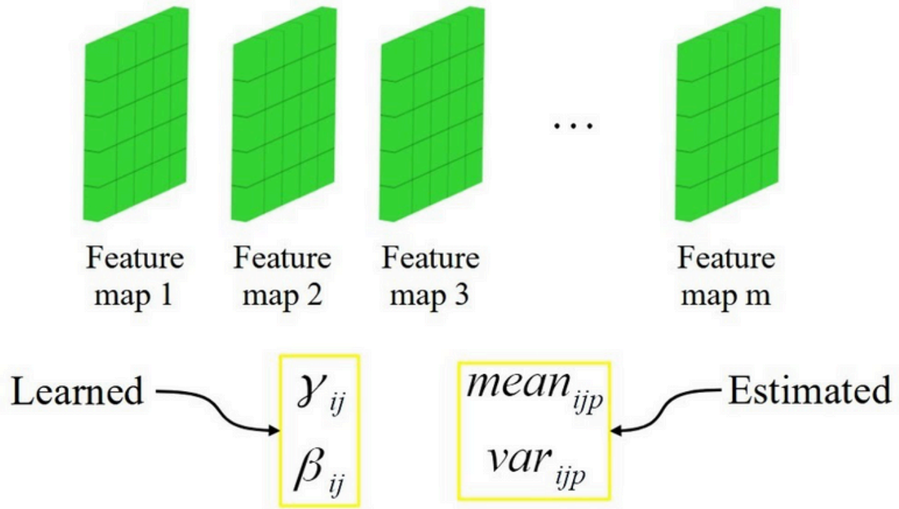
Common mini-batch sizes are 32/64/128 examples


Mini-batch gradient descent
Batch gradient descent
Stochastic gradient descent

Batch Normalization

- Uses batch statistics: mean (μ) and variance (σ^2) of the current mini-batch
- has 2 learnable parameters:
    - γ (gamma) controls the scale (how stretched the distribution is)
    - β (beta) controls the shift (moves the center left or right)

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \qquad y = \gamma\hat{x} + \beta$$



Batch Normalization

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

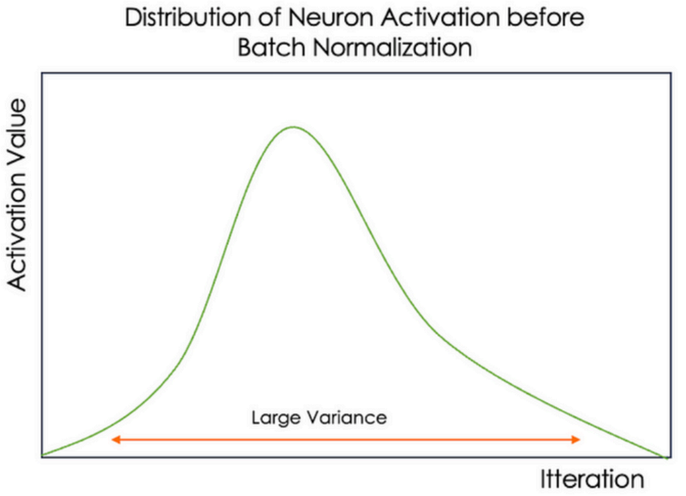
$$x = [2,\ 4,\ 6,\ 8]$$

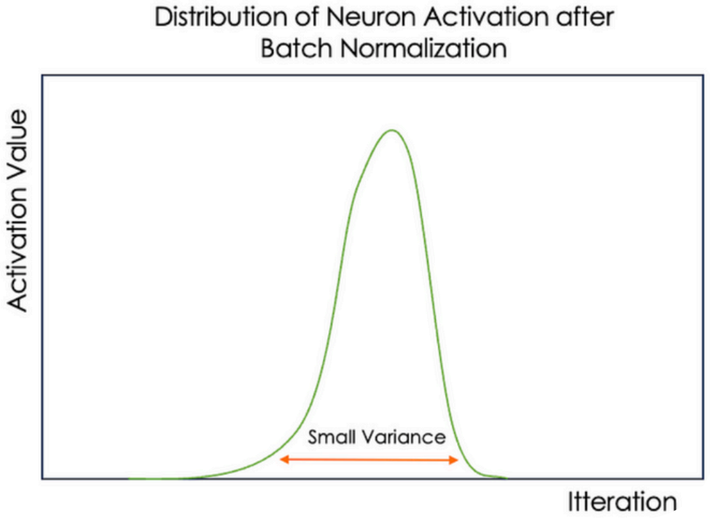
$$\mu = \frac{2 + 4 + 6 + 8}{4} = 5$$

$$\sigma^2 = \frac{(2-5)^2 + (4-5)^2 + (6-5)^2 + (8-5)^2}{4} = \frac{9 + 1 + 1 + 9}{4} = 5$$

$$\hat{x} = [-1.342,\ -0.447,\ 0.447,\ 1.342]$$

$$\mu_{\hat{x}} = 0 \qquad \sigma^2_{\hat{x}} = 1$$





Batch Normalization

$$y = \gamma \hat{x} + \beta$$

$$\gamma = 2,\ \beta = 1$$

$$y = 2 \cdot \hat{x} + 1$$

$$y = [-1.684,\ 0.106,\ 1.894,\ 3.684]$$

$$Var(y) = \gamma^2 = 4$$

- Applying γ (scale) stretches the distribution.
- Applying β (shift) moves the entire distribution left/right.
- The distribution is now 4× wider than before scaling
- β does not affect variance only shifts the mean

*BatchNorm learns statistics during training and reuses them during inference.

Dropout

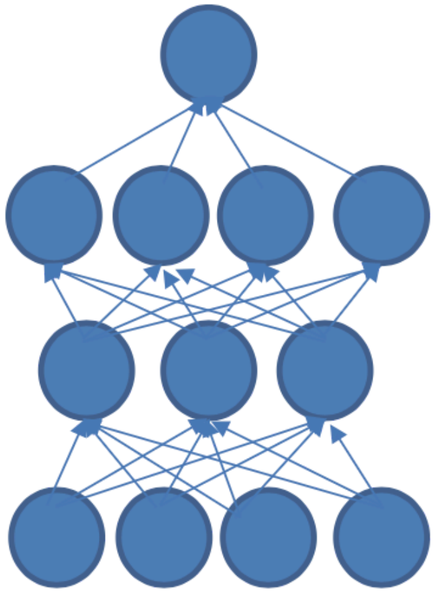
Input x = {1, 2, 3, 4, 5},

Dropout p = 0.2, keep probability = 0.8.

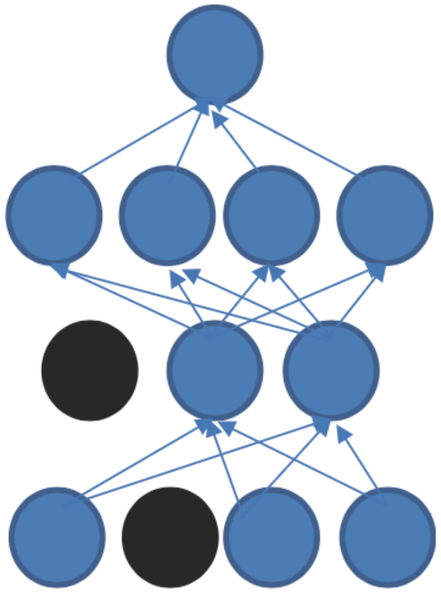
Randomly dropped version may be

{1, 0, 3, 4, 5} or {1, 2, 0, 4, 5}.

Hidden layer example: 1000 neurons, p = 0.5 → 500 neurons removed each iteration.

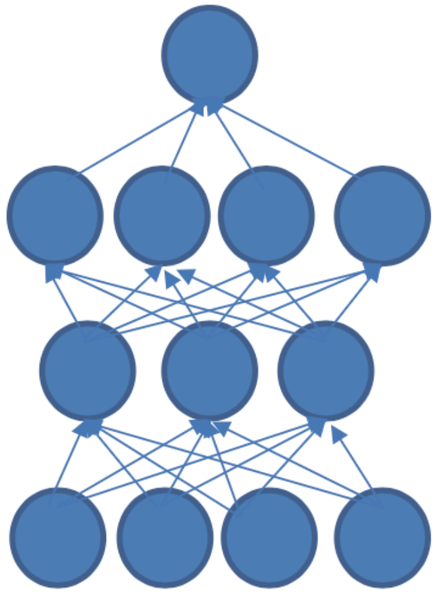


Architecture without dropouts

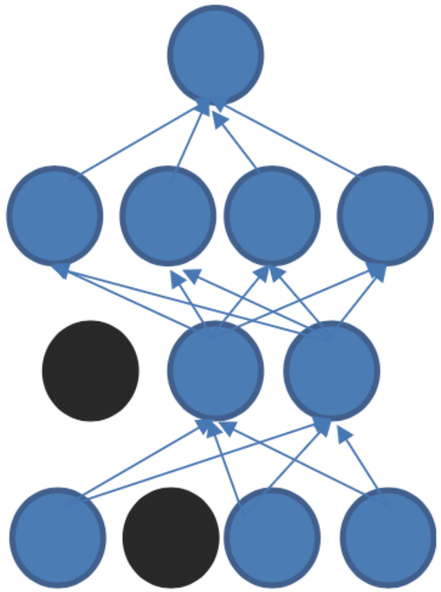


Architecture with dropouts

Dropout



Architecture without dropouts



Architecture with dropouts

$$x = [2,\ 4,\ 6,\ 8]$$

$$w = [1,\ 2,\ 3,\ 4]$$

$$y = x \cdot w^T = (2)(1) + (4)(2) + (6)(3) + (8)(4)$$

$$y = 2 + 8 + 18 + 32 = 60$$

$$p = 0.5$$

Dropout

Training

$$r = [1,\ 0,\ 1,\ 0]$$

$$x_{\text{drop}} = x \odot r = [2,\ 0,\ 6,\ 0]$$

$$y_{\text{train}} = (2)(1) + (0)(2) + (6)(3) + (0)(4)$$

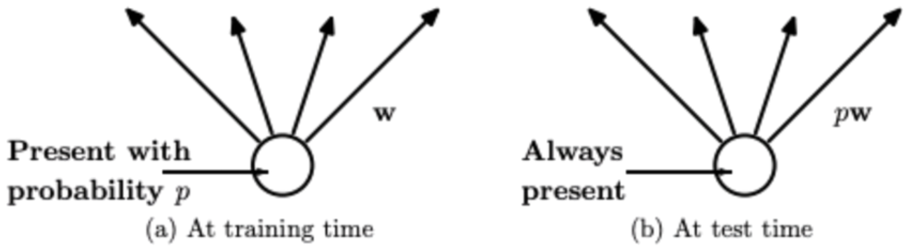
$$= 2 + 0 + 18 + 0 = 20$$

Inference

$$w' = p \cdot w = [0.5,\ 1,\ 1.5,\ 2]$$

$$y_{\text{test}} = x \cdot w'^{T} = (2)(0.5) + (4)(1) + (6)(1.5) + (8)(2)$$

$$y_{\text{test}} = 1 + 4 + 9 + 16 = 30$$

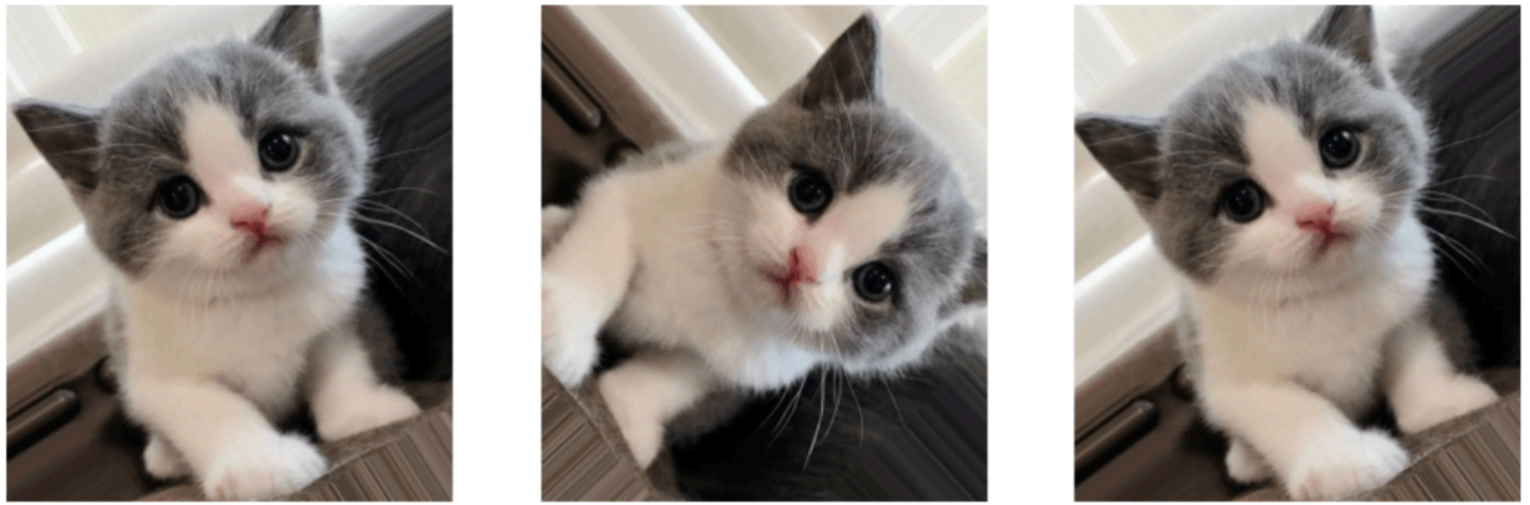


(a) At training time (b) At test time

- Training avg = 30, Testing = 30

Data Augmentation

Random Rotation



Rotation technique

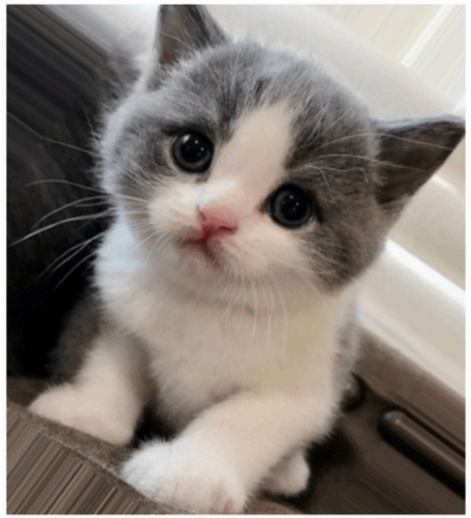
Data Augmentation

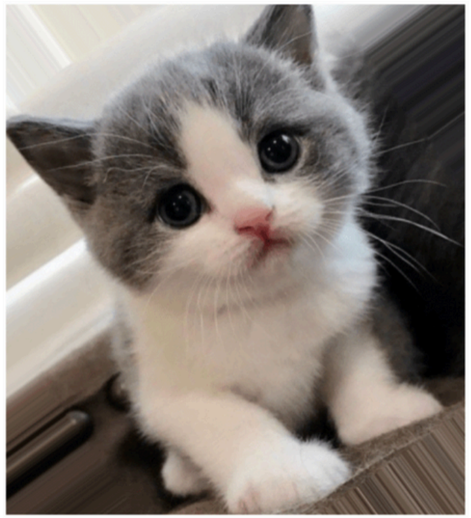
Brightness



Data Augmentation

Flip





Horizontal flip technique

Vertical flip technique

Data Augmentation

Zoom

Zoom technique

Random Shift

Width shift technique


Q
A

Thank You