

# **ComputerVision**

**Week7**

2025-2

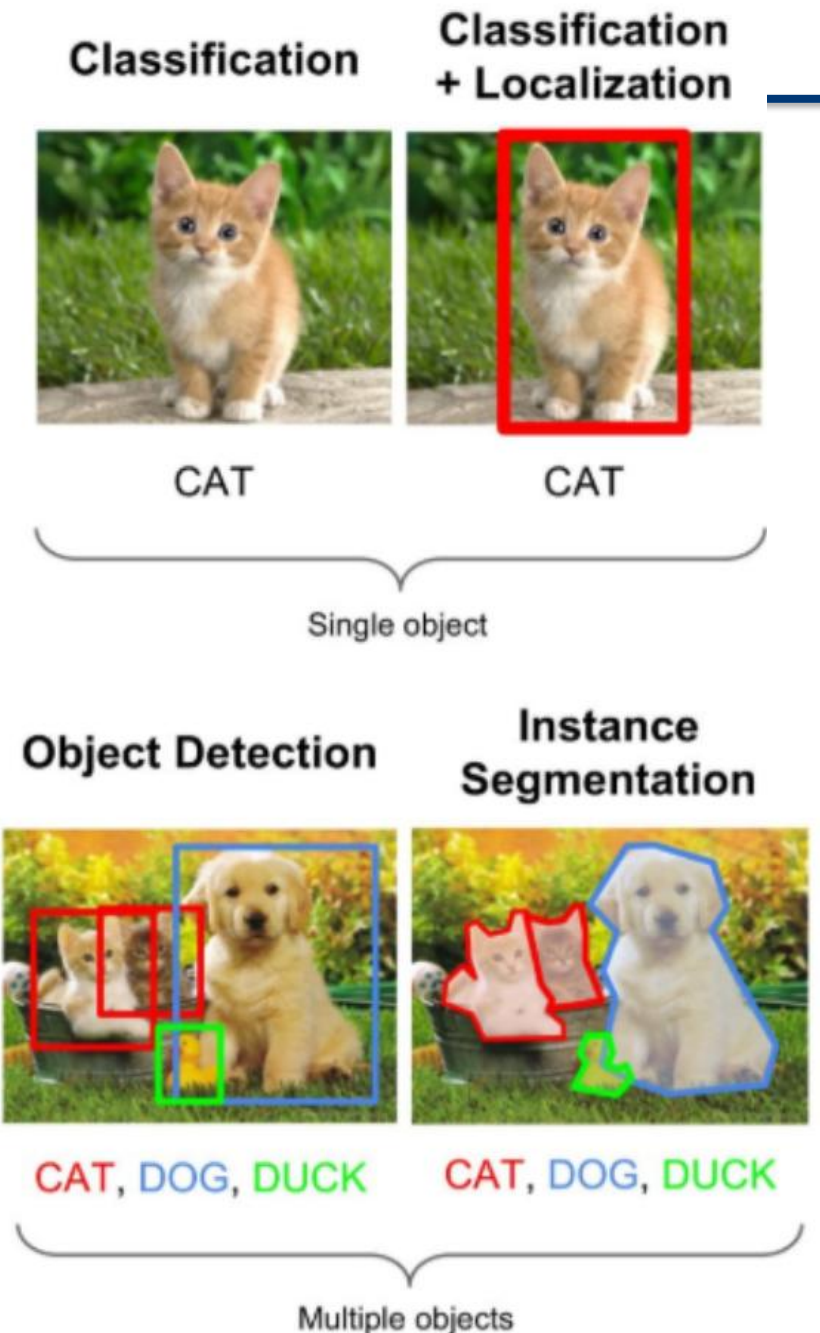
Mobile Systems Engineering

Dankook University

# From Classification to Segmentation

## ■ Understanding different vision tasks

- **1. Classification** – Translation Invariance Task
  - “What is in the image?”
  - Single label for the entire image.
- **2. Classification + Localization** – Translation Variance Task
  - “What is it, and where is it?”
  - Predicts class label and bounding box for **one object**.
- **3. Object Detection** – Translation Variance Task
  - “What objects are there, and where are they?”
  - Detects **multiple objects**, each with a bounding box and label.
- **4. Instance Segmentation** – Translation Variance Task
  - “Which pixels belong to each object?”
  - Pixel-level masks that separate even objects of the same class.

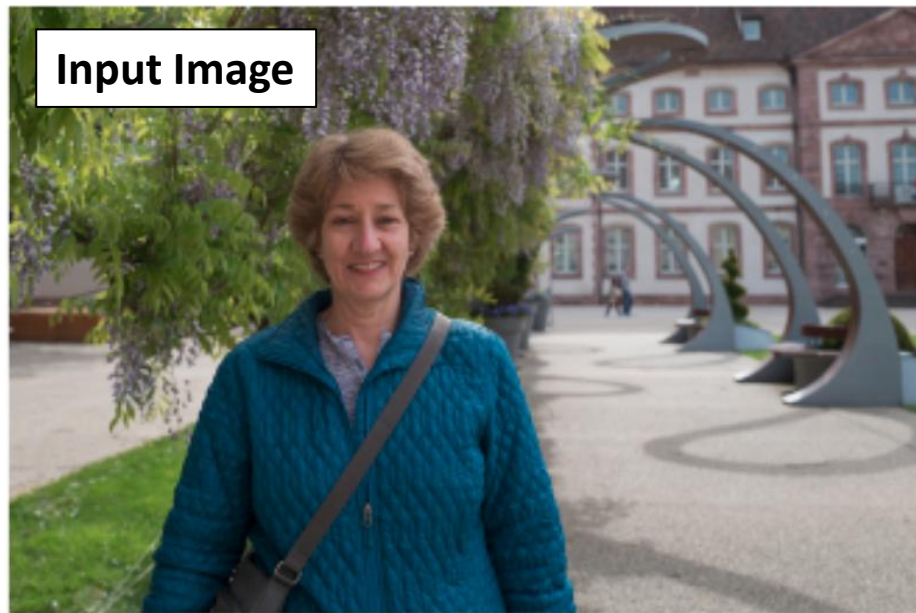


# From Classification to Segmentation

## ■ Segmentation Types: Semantic vs. Instance

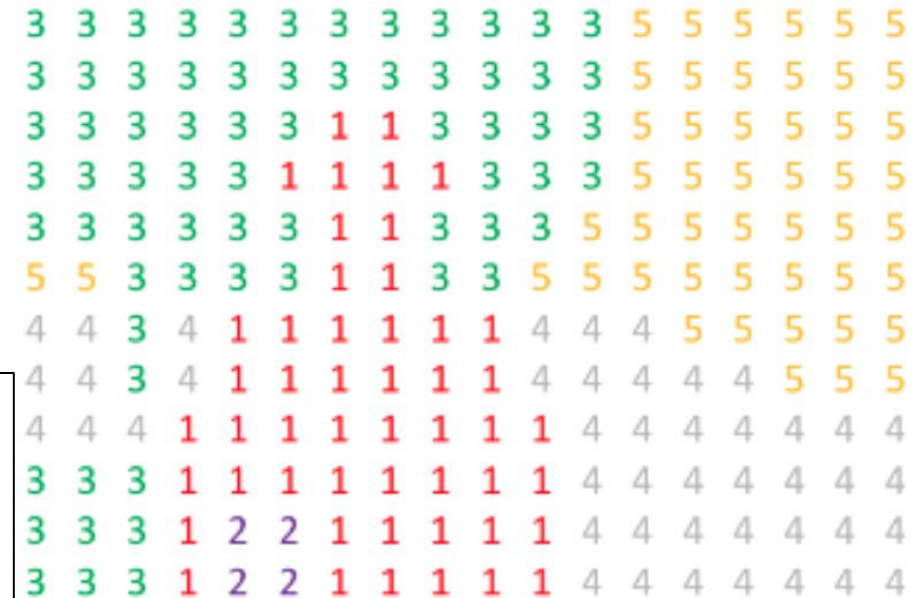
### • Segmentation Types in Computer Vision

#### ○ 1. Semantic Segmentation

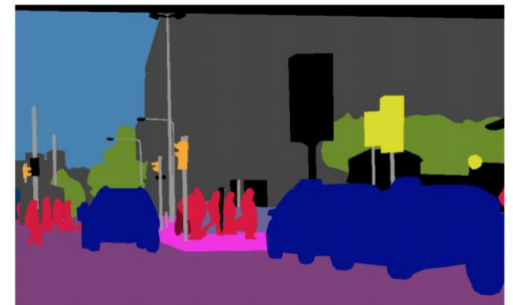


1: Person  
2: Purse  
3: Plants/Grass  
4: Sidewalk  
5: Building/Structures

Semantic Labels



- ✓ Assigns a **class label** to **each pixel** in the image.
- ✓ All objects of the same class share the **same label**.
  - Example: All cars in the image → labeled as “Car” (same color in mask).
- ✓ Limitation: Cannot distinguish **different instances** of the same class.

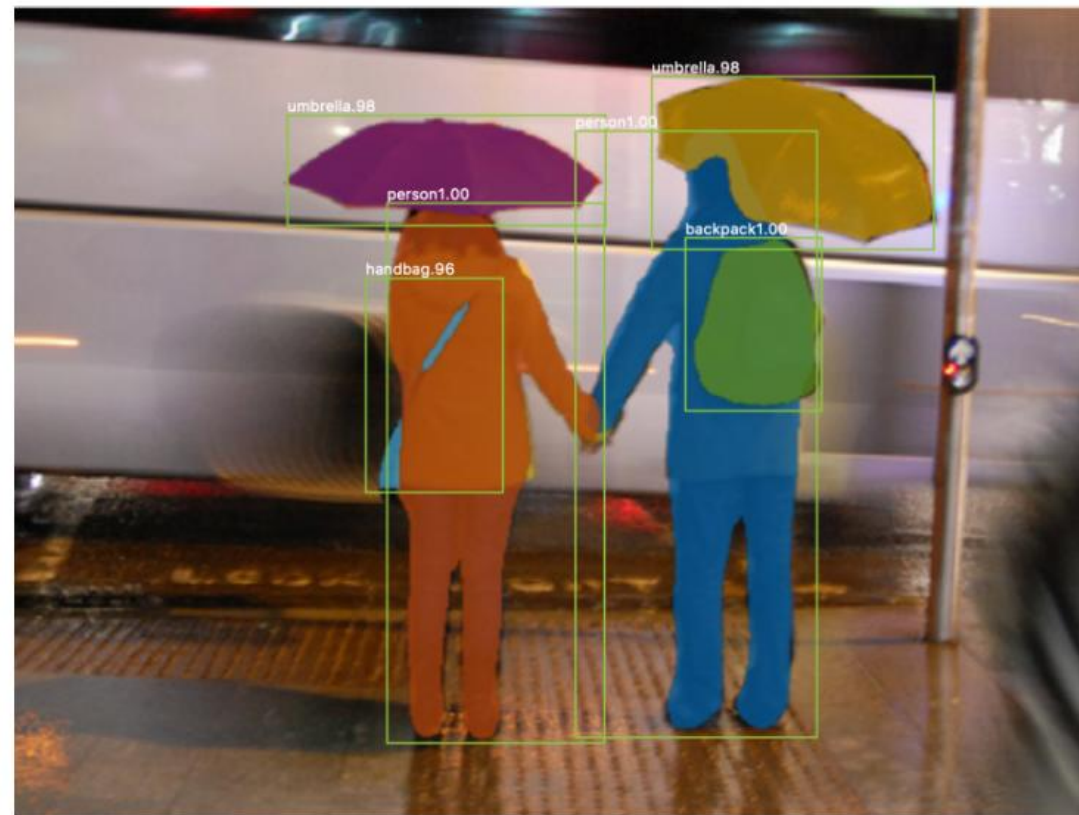


# From Classification to Segmentation

## ■ Segmentation Types: Semantic vs. Instance

### • Segmentation Types in Computer Vision

#### ○ 2. Instance Segmentation



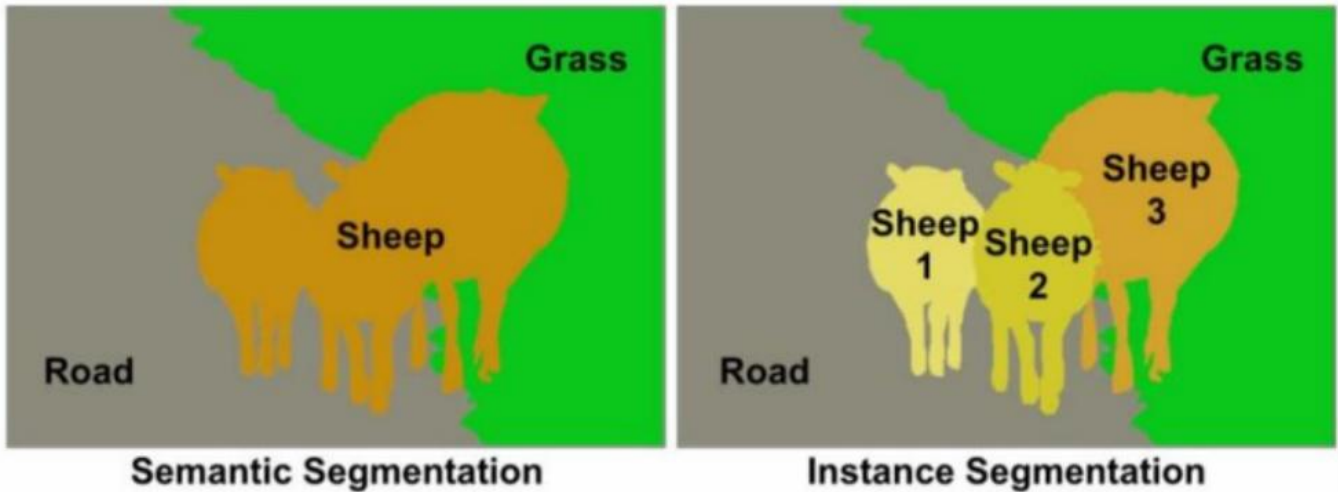
- ✓ Assigns **class + instance ID** to each pixel.
- ✓ Objects with the same class but different instances have **separate masks**.
  - Example: Two cars → both “Car” class, but with different colors/masks (i.e., different id – car1 and car2).
- ✓ Combines object detection’s **localization** with segmentation’s **pixel-level detail**.

# From Classification to Segmentation

- Segmentation Types: Semantic vs. Instance

  - Segmentation Types in Computer Vision

    - Key Differences



| Aspect             | Semantic Segmentation               | Instance Segmentation               |
|--------------------|-------------------------------------|-------------------------------------|
| Output Label       | Class ID per pixel                  | Class ID + Instance ID per pixel    |
| Same-class objects | Same label                          | Different labels (instance-aware)   |
| Example Use Cases  | Road scene parsing, medical imaging | Autonomous driving, object counting |



# From Classification to Segmentation

## ■ Why Standard CNN Architectures for Classification Fail at Segmentation

### • 1. How CNNs Learn Features

#### ○ Early Layers – Low-Level Features

✓ Learn basic visual patterns

➤ Edges, corners, simple colors.

✓ Capture fine-grained **spatial information**.

✓ Examples

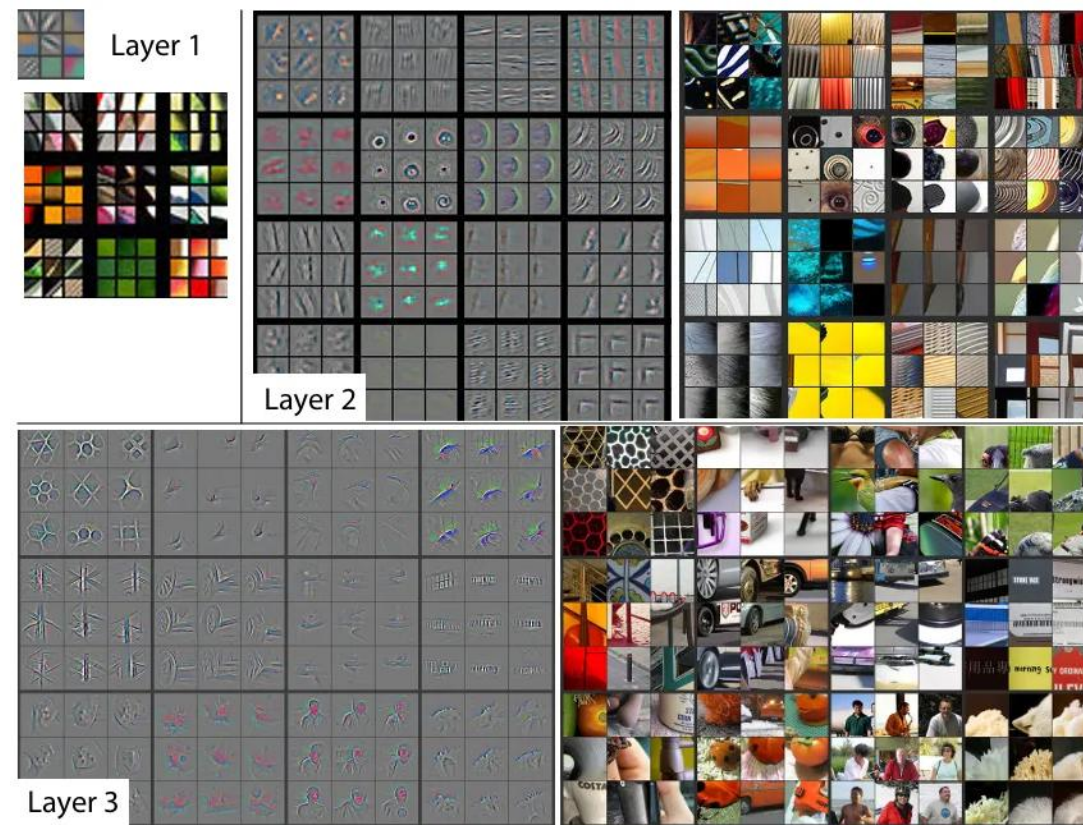
➤ Layer 1 detects straight lines, edges at different orientations.

#### ○ Middle Layers – Mid-Level Features

✓ Combine low-level patterns into **textures** and **object parts**.

✓ Example

➤ Layer 3 detects a dog's ear, a car wheel, or repeated surface patterns.



# From Classification to Segmentation

## ■ Why Standard CNN Architectures for Classification Fail at Segmentation

### • 1. How CNNs Learn Features

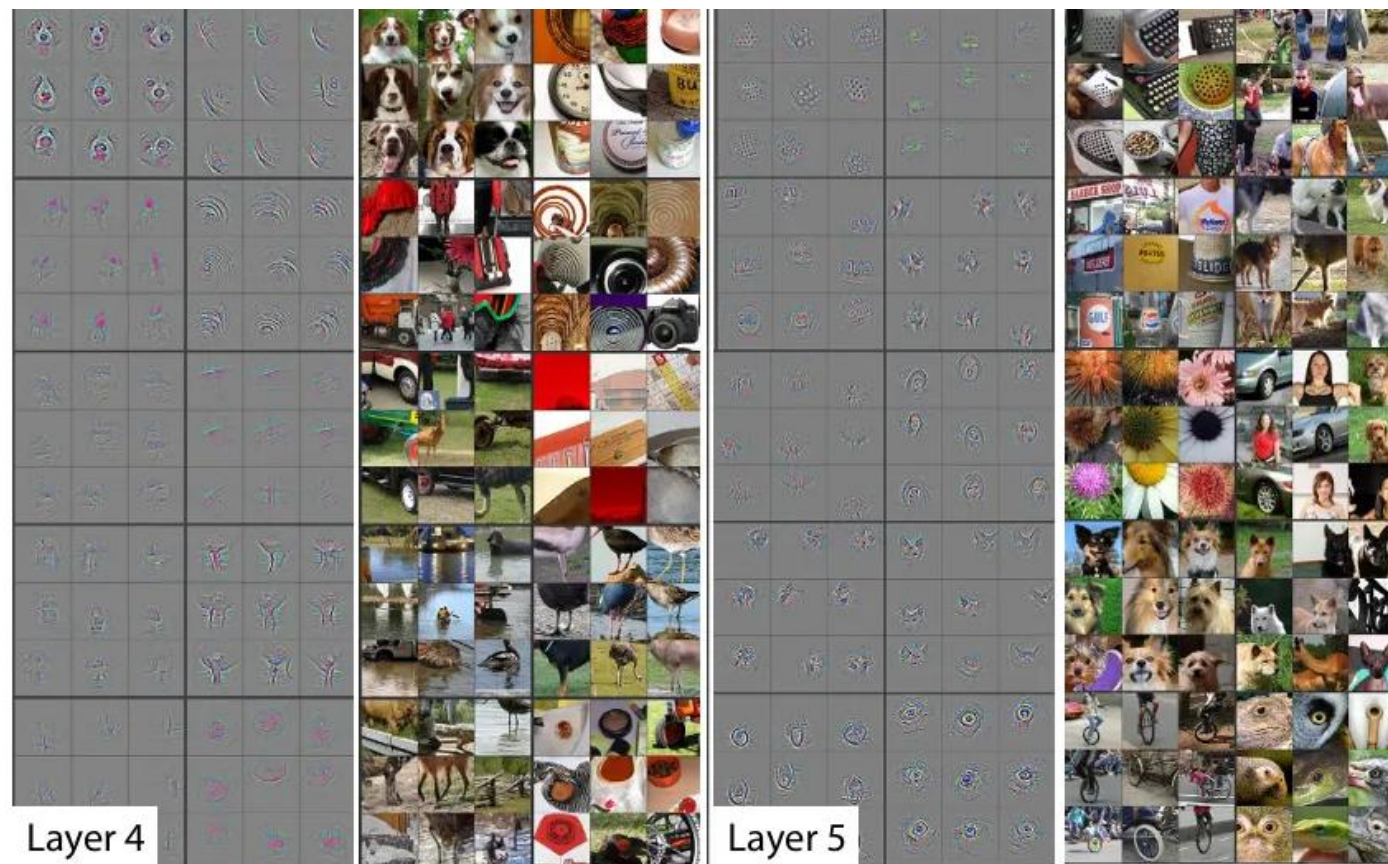
#### ○ Deep Layers – High-Level Features

✓ Detect **entire objects** or complex shapes.

✓ **Focus on semantic meaning**  
rather than precise location.

✓ Example

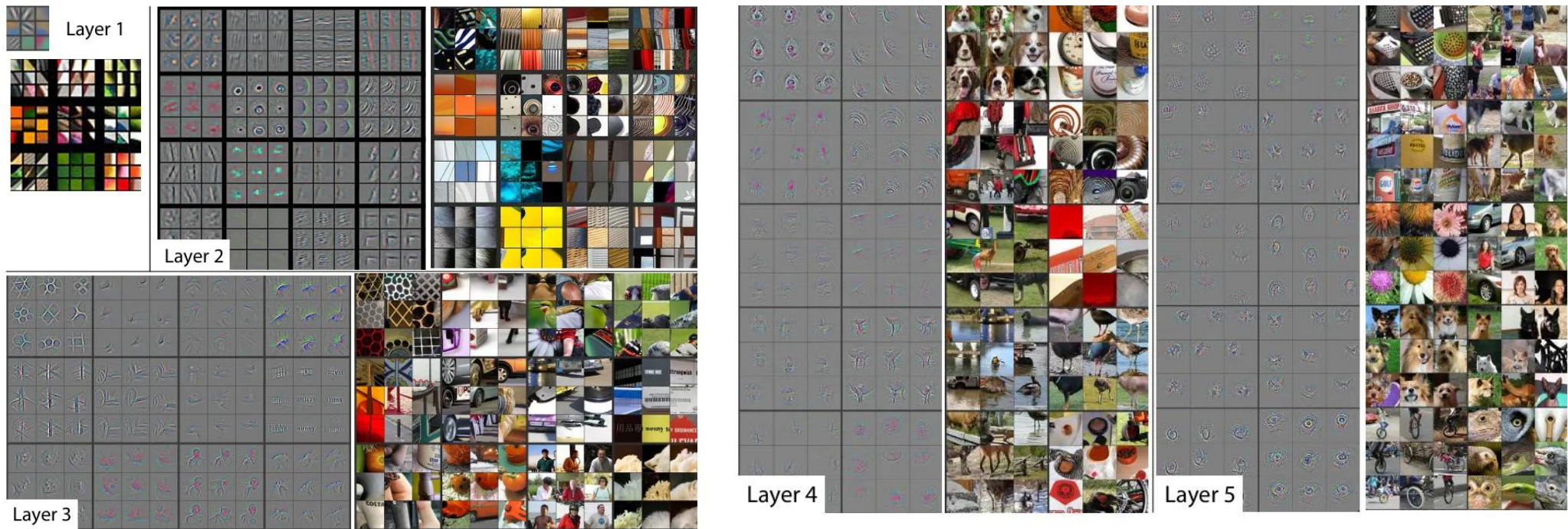
➤ Layer 5 activates  
for a whole dog or bicycle.





# From Classification to Segmentation

- Why Standard CNN Architectures for Classification Fail at Segmentation
  - 2. Important Observations for Segmentation



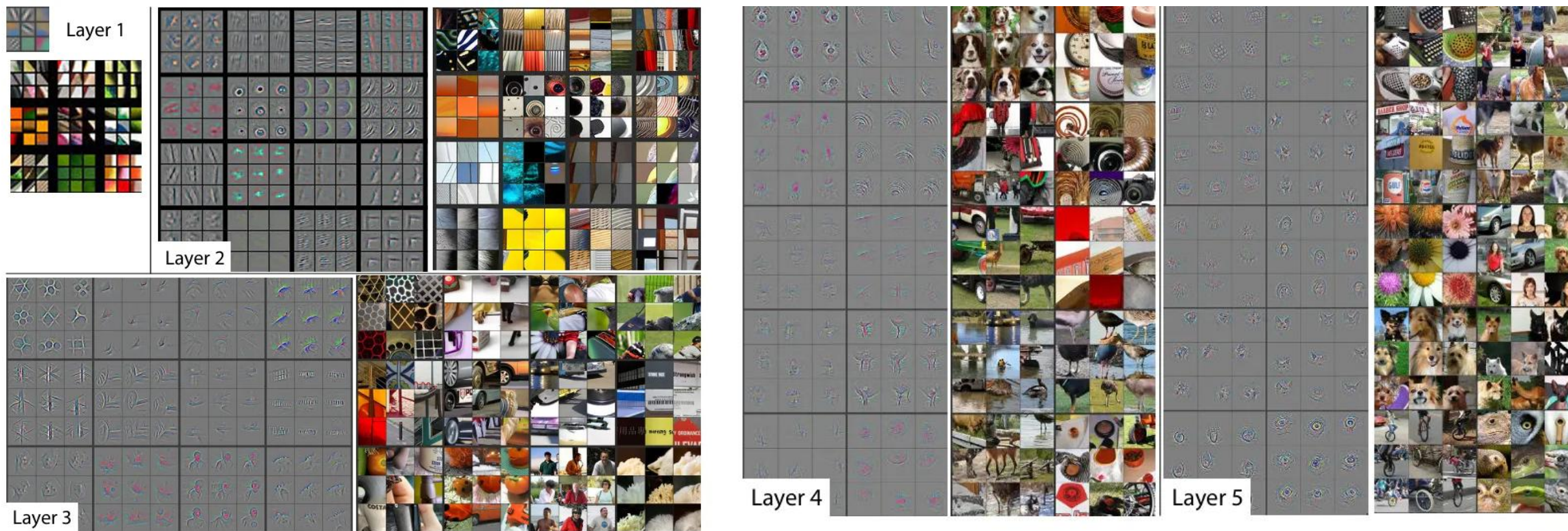
- Standard CNN is “Translation Invariance”!

- ✓ 1. As we move deeper, spatial resolution decreases due to pooling, stride, and downsampling.
  - This leads to loss of spatial information → harder to know exact pixel positions.
- ✓ 2. Fully connected layers discard almost all spatial info.



# From Classification to Segmentation

- Why Standard CNN Architectures for Classification Fail at Segmentation
  - 3. For segmentation, we need both for “Translation Variance”!



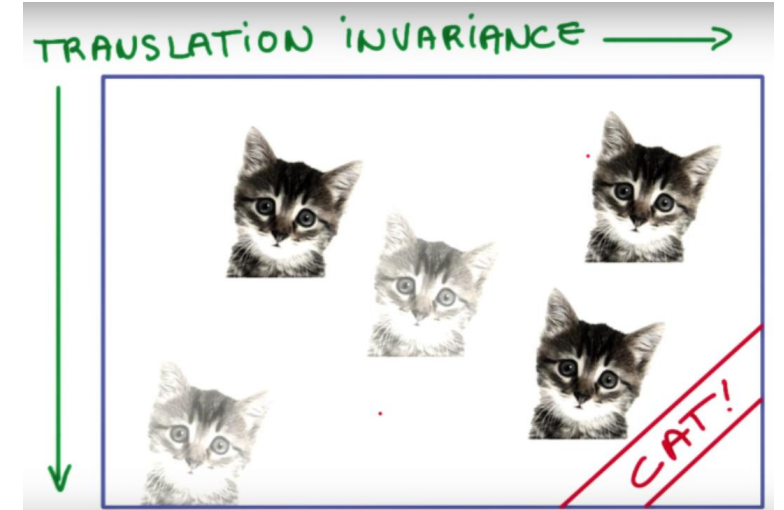
- 1. Low-level features → for precise boundaries (locality).
- 2. High-level features → for semantic understanding.

# From Classification to Segmentation

## ■ How CNN Achieves Translation Invariance

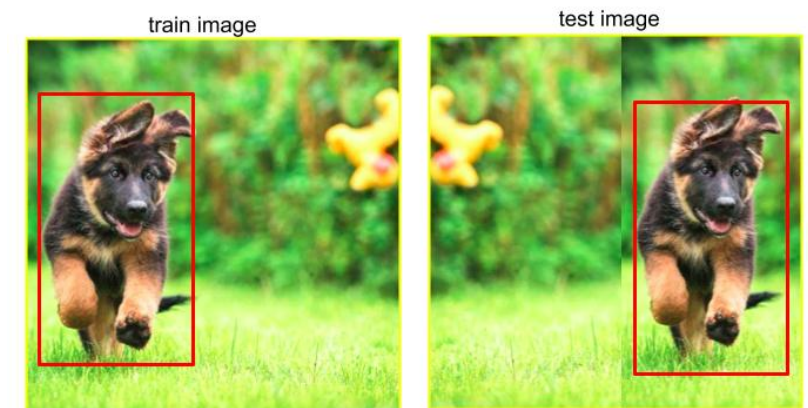
### • 1. Translation Equivariance vs. Translation Invariance

- **Equivariance:** Output changes position **in the same way** as input.
  - ✓ Example: If a cat's face moves in the input, the feature map also moves.
- **Invariance:** Output **remains the same** even if the input shifts.
  - ✓ Example: A cat is still labeled "cat" no matter where it appears.



### • 2. Why CNN Has Translation Invariance

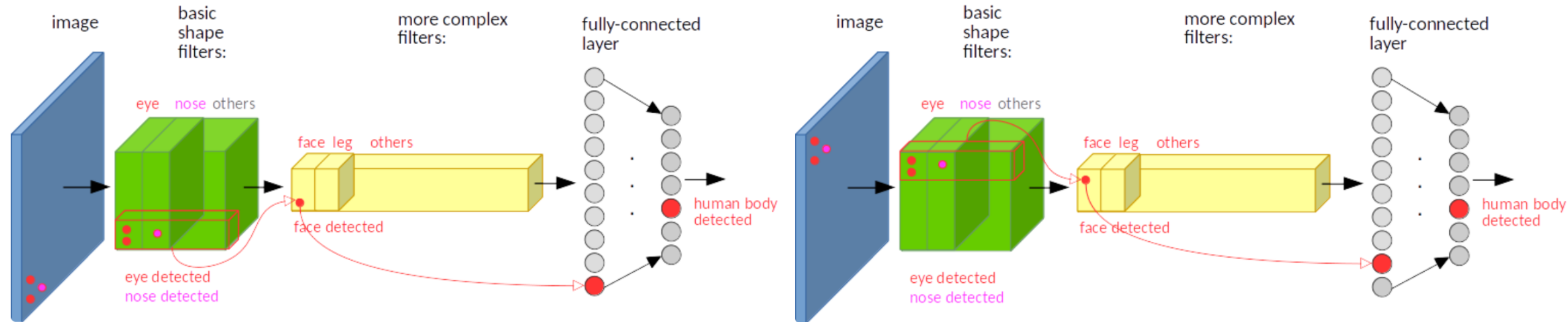
- Convolution operation → **translation equivariant** by nature.
- **Weight sharing:** Same filters detect patterns anywhere in the image.
- **Pooling:** Reduces sensitivity to small shifts (small-scale invariance).
- **Fully connected layers + Softmax:** Final classification ignores position.



# From Classification to Segmentation

## ■ How CNN Achieves Translation Invariance

### • 3. Example Flow in CNN



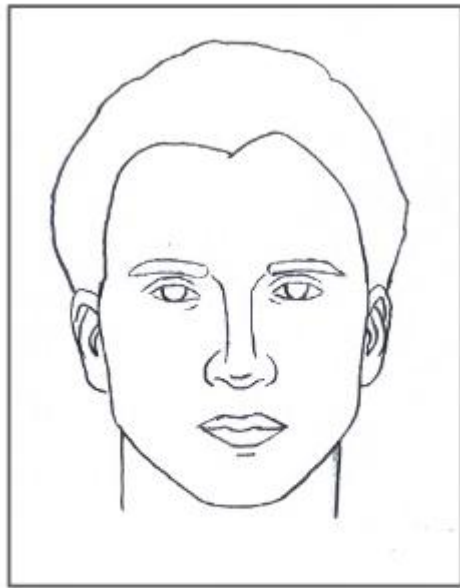
- **Early convolution layers:** Detect low-level features (eyes, nose) at exact positions → still equivariant.
- **Deeper convolution layers:** Combine features into higher-level patterns (face, leg).
- **FC layer & Softmax:** Output label probability is **position-independent** → invariant.



# From Classification to Segmentation

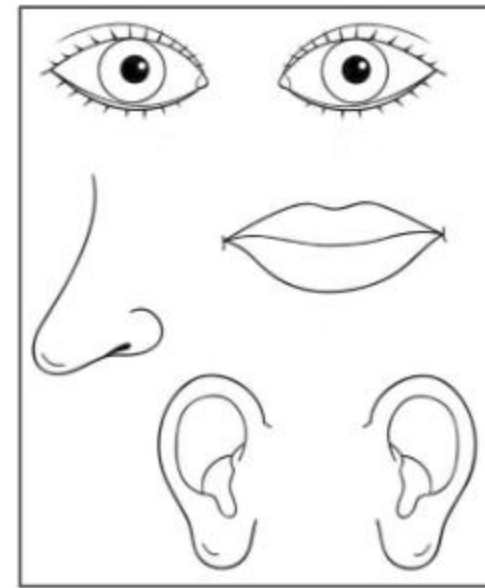
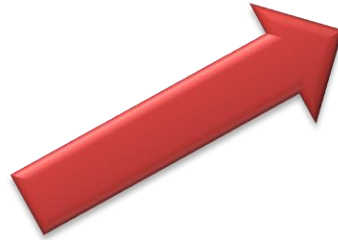
## ■ How CNN Achieves Translation Invariance

### • 4. Why This Matters

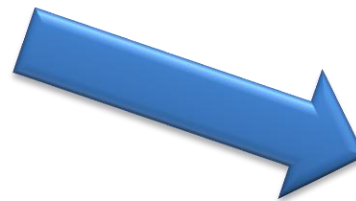


Face

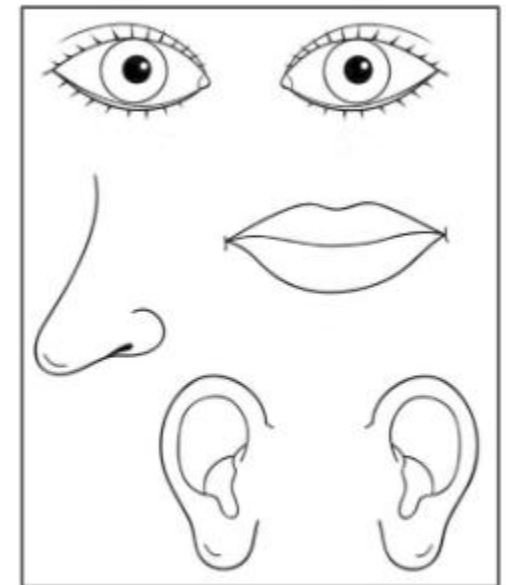
Classification Task



Face



Segmentation Task



Not Face

- For **classification**: Translation invariance is beneficial.
- For **segmentation**: It can cause *loss of locality*, harming pixel-level predictions.

# From Classification to Segmentation

---

## ■ Detection Needs Translation Variance

### • Why Detection Needs Translation Variance

- **Classification:** Output label does not depend on location → *translation invariant*.
- **Detection:** Must **localize** the object → output changes when the object moves.
- **Translation variance** means the network output must change according to the object's position.

### • Key Requirements for Detection

#### ○ Preserve Spatial Information

- ✓ **Spatial information** = exact positional coordinates of features in the image.
- ✓ Allows model to determine *where* the object is located.
- ✓ Loss of spatial info → bounding boxes drift or fail completely.

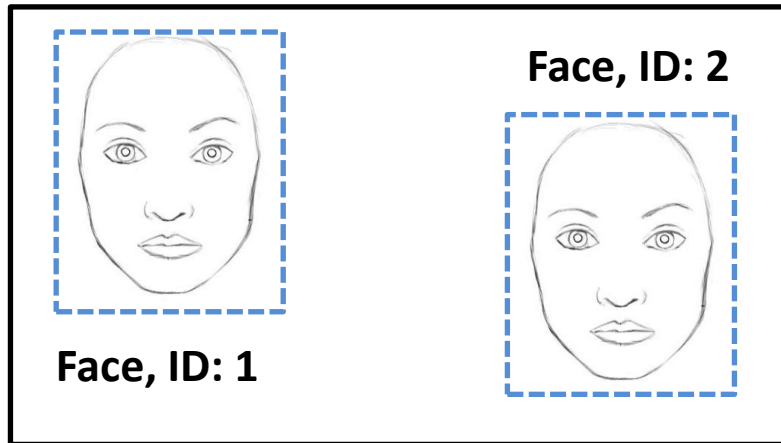
#### ○ Preserve Locality

- ✓ **Locality** = maintaining the relationship between features **within a region** of the image.
- ✓ Important for capturing an object's **internal structure** (e.g., face = eyes + nose).

# From Classification to Segmentation

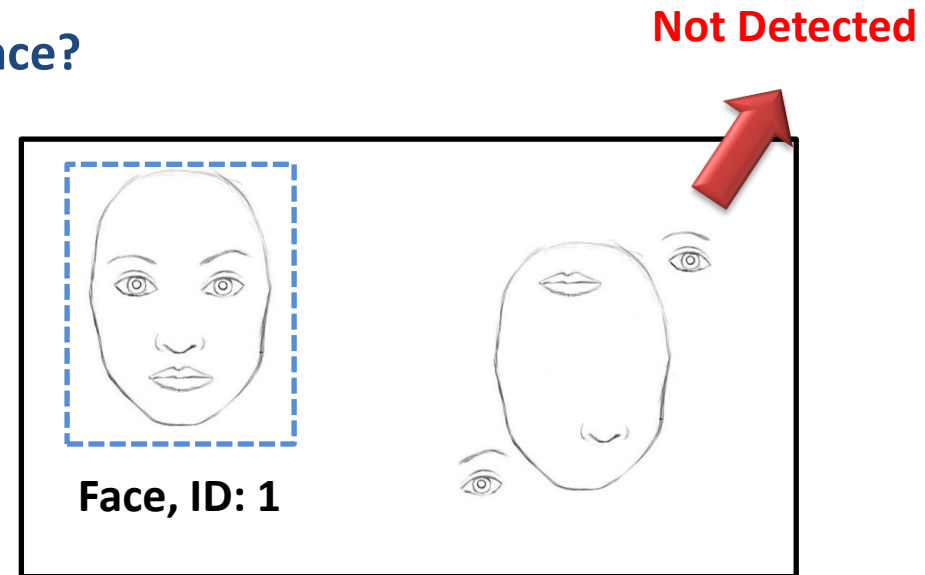
## ■ Detection Needs Translation Variance

- Spatial Information vs. Locality – What's the Difference?



 Detected Object

**Spatial Information-based Facial Identification**



 Detected Object

**Locality-based Facial Identification**

| Term                       | Meaning  | Example in Detection Task                                 |
|----------------------------|--|---|
| <b>Spatial Information</b> | Absolute and relative coordinates of features in the image | "The face is at (x=150, y=200)"                           |
| <b>Locality</b>            | Preservation of spatial relationships within a region      | "The ears are above the eyes, the nose is below the eyes" |



# From Classification to Segmentation

---

## ■ How to Preserve Spatial Information and Locality

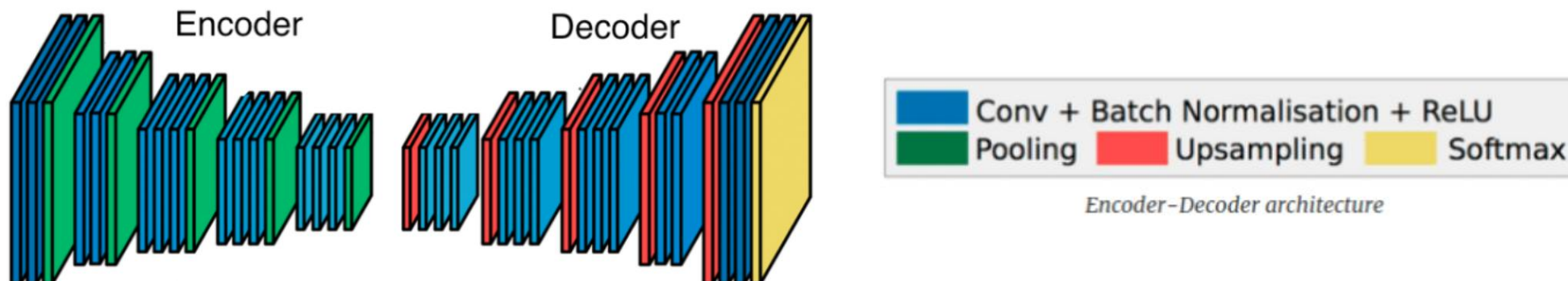
### • Problem Recap

- 1. Pooling & stride in CNN → **downsampling** → loss of spatial resolution.
- 2. **Spatial information**: Absolute position of features in the image.
- 3. **Locality**: Relative arrangement of features within a region.
- 4. Loss of either → blurry boundaries & inaccurate localization.

# From Classification to Segmentation

## ■ How to Preserve Spatial Information and Locality

### • Solution 1: Encoder–Decoder Structure



#### ○ Encoder

- ✓ Uses **convolutions + pooling** to extract high-level semantic features.
- ✓ Gradually reduces spatial dimensions ( $W \times H \rightarrow w \times h$ ).

#### ○ Decoder

- ✓ Uses **upsampling** to restore feature maps to original size ( $w \times h \rightarrow W \times H$ ).
- ✓ Generates segmentation map where each pixel has a class label.

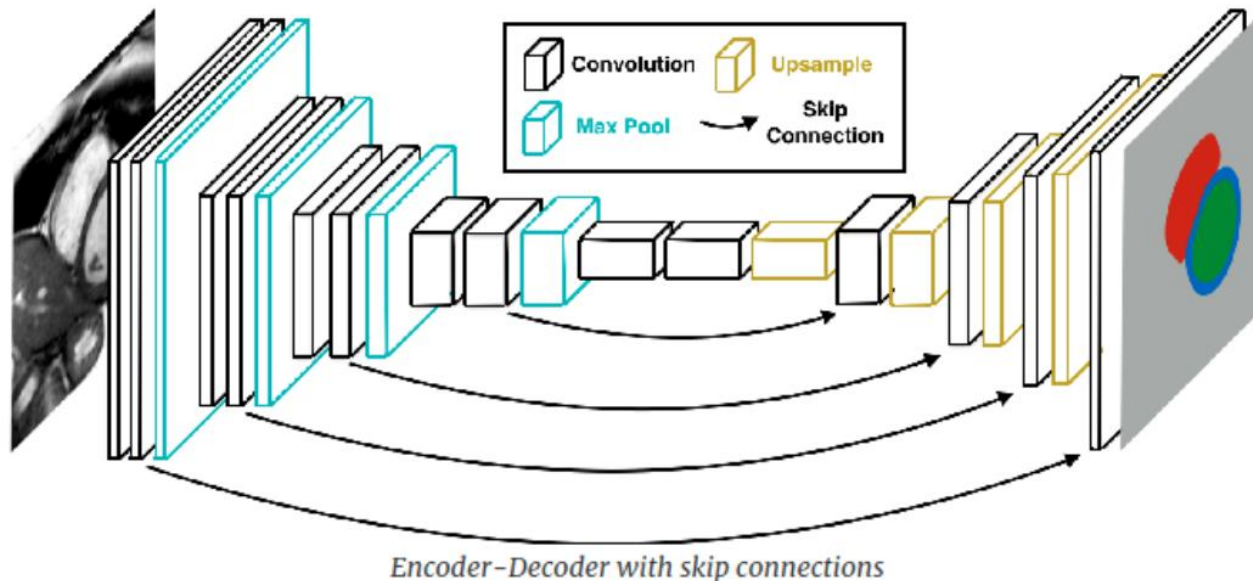
#### ○ Problem in Simple Encoder–Decoder

- ✓ High-level features from encoder lack **low-level details** (edges, fine structures).
- ✓ Upsampling alone cannot reconstruct precise boundaries.

# From Classification to Segmentation

## ■ How to Preserve Spatial Information and Locality

### • Solution 2: Skip Connections



- Directly connect encoder layers to corresponding decoder layers.
- Merge **low-level spatial details** from encoder with **high-level semantics** from decoder.
- Restores sharp object boundaries and preserves locality.
- **Benefits**
  - ✓ Retain **low-level details** (edges, textures).
  - ✓ Preserve **spatial resolution** across network.
  - ✓ Combine **locality** + **semantic context** for accurate segmentation.



# Introduction to Fully Convolutional Networks (FCNs)

## ■ Fully Convolutional Networks for Semantic Segmentation

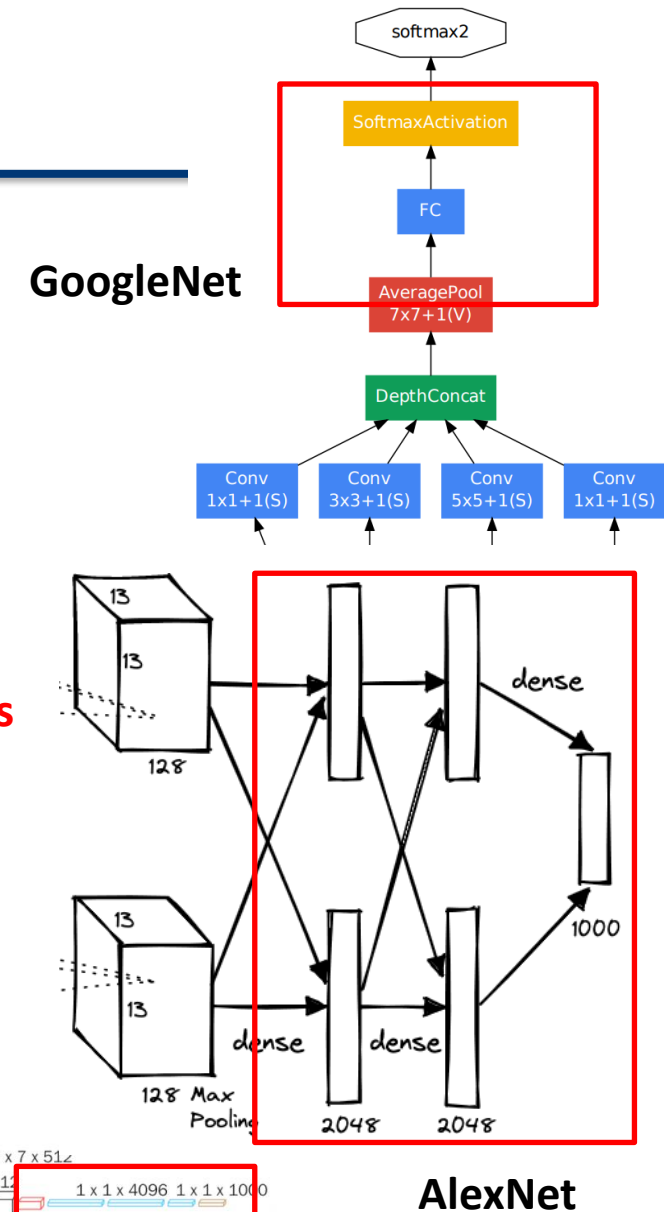
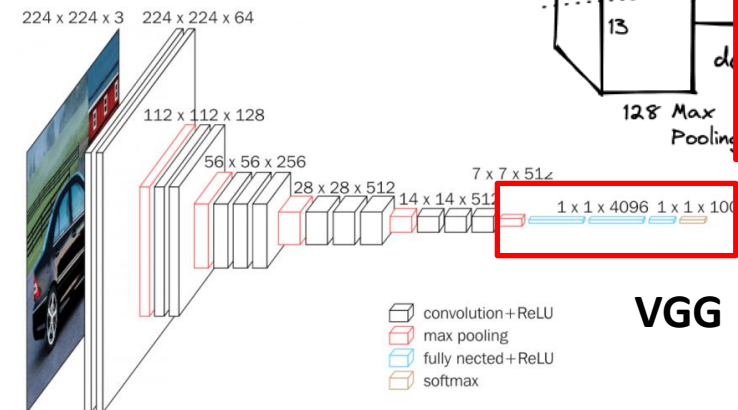
### • Goal

- Perform **semantic segmentation**: assign a **class label** to **every pixel** in an image.
- Achieve **end-to-end learning** from input image to dense output.
- Balance **semantic meaning** (*what is present*) with **spatial precision** (*where it is located*).

### • Key Idea

- Transform existing high-performing classification CNNs (e.g., AlexNet, VGG, GoogLeNet) into **fully convolutional networks**.
- Replace **fully connected layers** with **convolutional layers** to preserve spatial information.
- Adapt these networks for **dense prediction** tasks.

  Fully-connected layers



# Introduction to Fully Convolutional Networks (FCNs)

---

## ▪ Fully Convolutional Networks for Semantic Segmentation

### • Key Insight

#### ○ Fully Convolutional Design

- ✓ 1. Accepts **arbitrary-sized inputs**.
- ✓ 2. Produces outputs of **corresponding spatial size**.
- ✓ 3. Enables **efficient whole-image training & inference**.
- Avoids complex pre/post-processing (superpixels, proposals, CRFs).
- Fine-tune from supervised pre-training for strong initial feature representations.

### • Core Steps

#### ○ 1. Convolutionalization

- ✓ Replace fully connected layers with equivalent convolution layers.
- ✓ Maintain spatial correspondence and allow variable input sizes.

#### ○ 2. Upsampling (Deconvolution)

- ✓ Convert coarse feature maps into dense, high-resolution predictions.

#### ○ 3. Skip Architecture

- ✓ Fuse **deep, coarse, semantic** features with **shallow, fine, appearance** features for accurate boundaries.

# Convolutionalization: Motivation

## ■ Why Replace Fully Connected Layers for Segmentation?

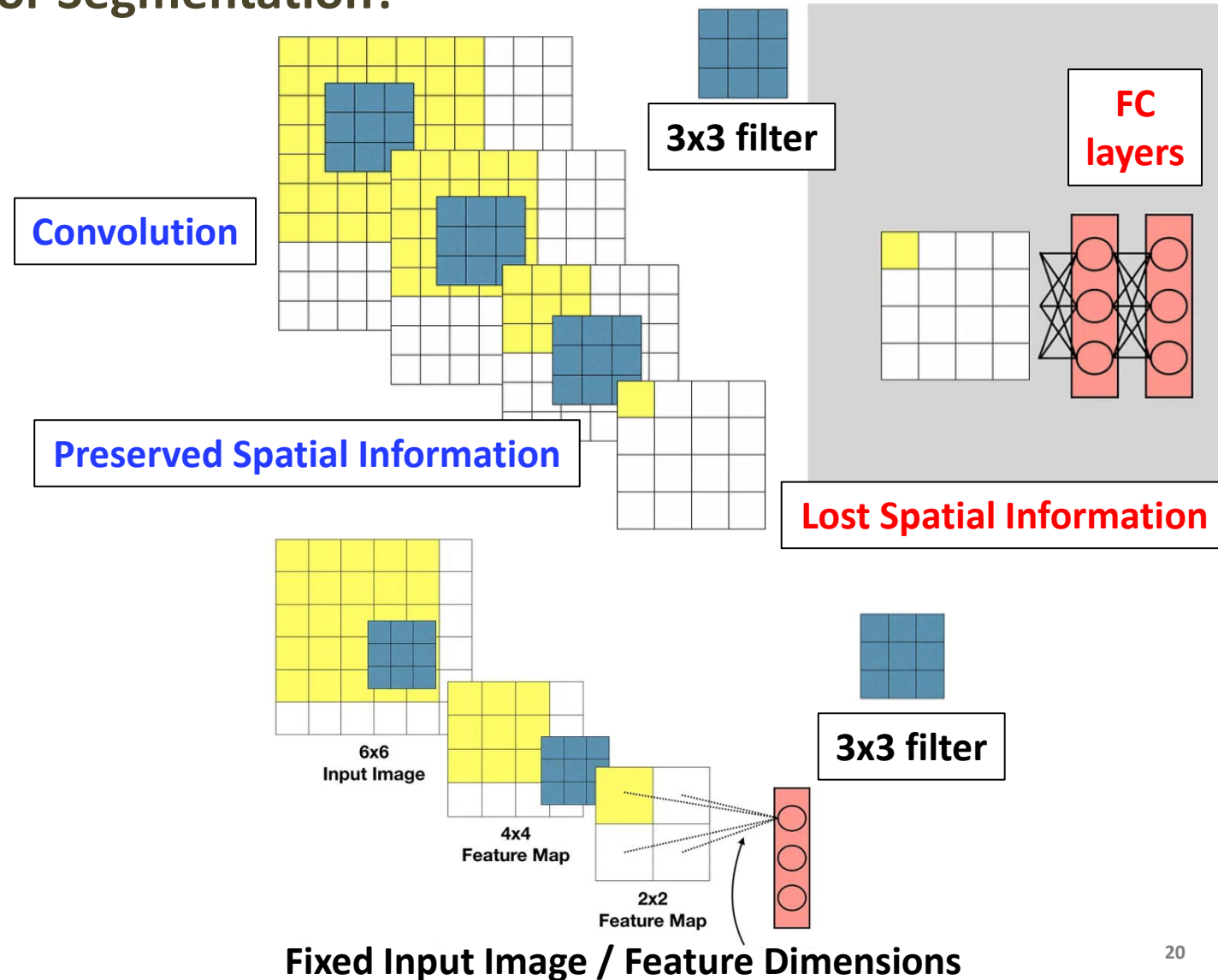
### • Problems with Fully Connected Layers

#### ○ 1. Loss of Spatial Information

- ✓ FC layers flatten feature maps, discarding pixel location data.
- ✓ Critical spatial correspondences (what is where) are lost.

#### ○ 2. Fixed Input Size Constraint

- ✓ FC layers require a fixed vector size → fixed input image dimensions.
- ✓ Limits flexibility for variable-sized images.



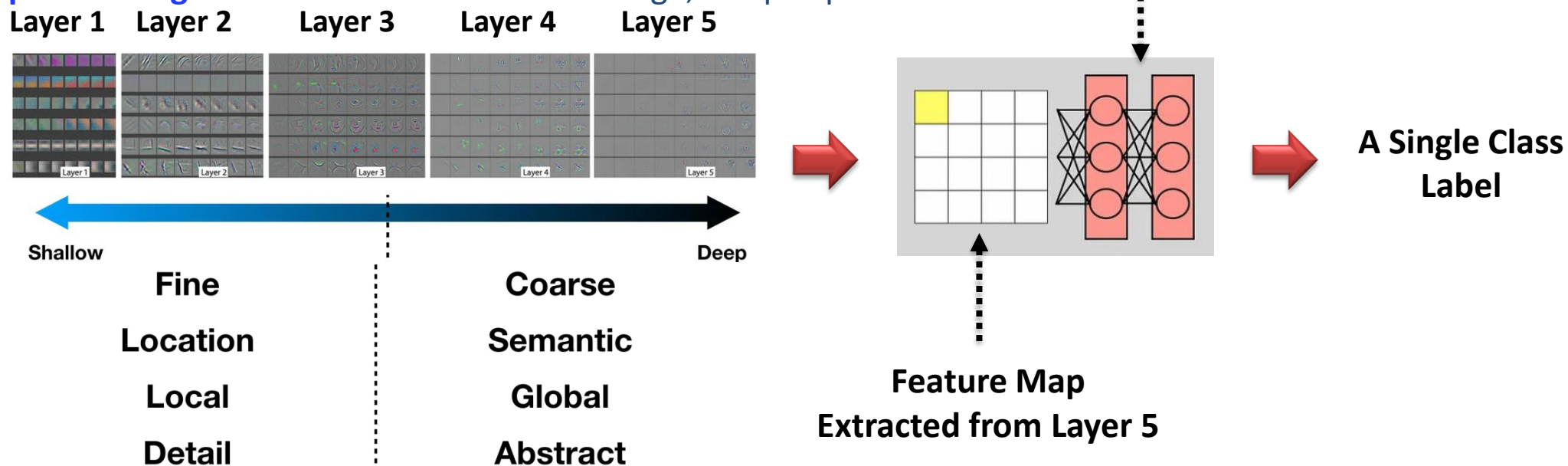
# Convolutionalization: Motivation

## ■ Why Replace Fully Connected Layers for Segmentation?

### • Problems with Fully Connected Layers

#### ○ 3. Coarse Outputs for Dense Prediction

✓ Output is a **single class label** for the whole image, not per-pixel.



#### ○ Summary – Why This is a Problem for Segmentation

- ✓ 1. Semantic segmentation needs **dense, pixel-to-pixel predictions**.
- ✓ 2. Requires **both semantic meaning** (what the object is) **and location information** (where it is).
- ✓ 3. FC layers destroy the "where" component.



# Convolutionalization: Motivation

## ■ Why Replace Fully Connected Layers for Segmentation?

### • The FC-to-Conv Transformation (Convolutionalization)

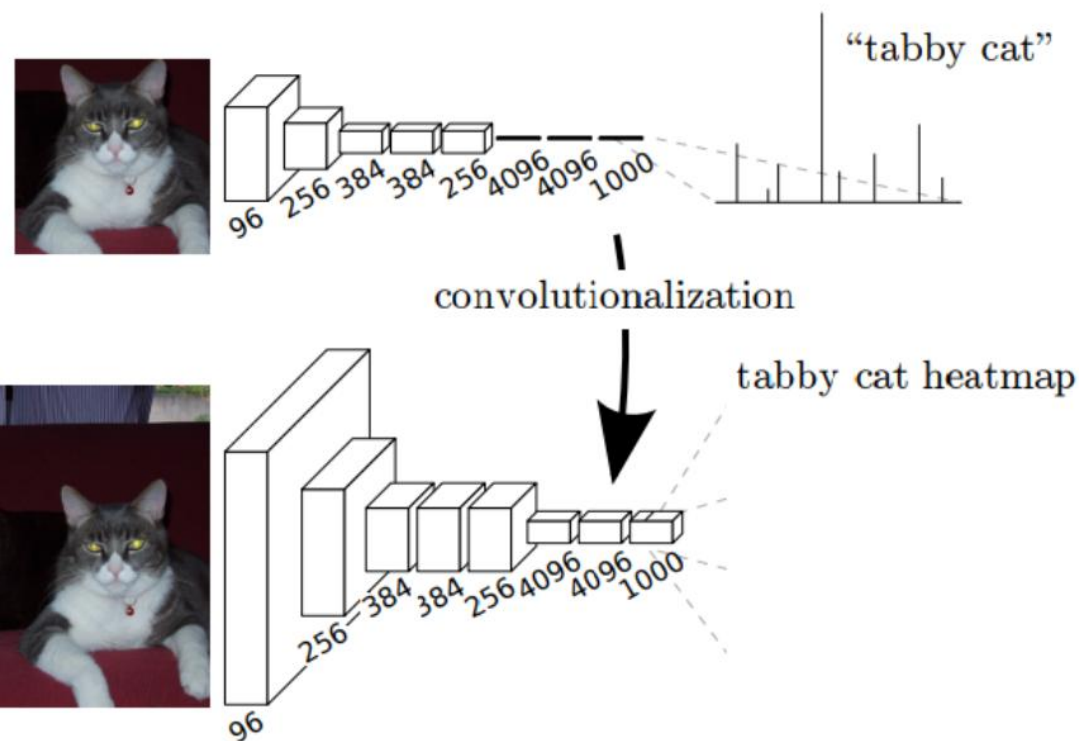
- Replace FC layers with **equivalent convolutional layers**.

- ✓ Example: VGG16's first FC layer  
→ 7×7 Conv layer with the same number of parameters.

- ✓ Final FC layer  
→ **1×1 Conv layer** with channels = number of classes.

### ○ Benefits

- ✓ (1) Accepts **arbitrary input sizes**
- ✓ (2) Produces **spatial output maps** that preserve location cues
- ✓ (3) Enables **end-to-end pixel-level learning**.



# Convolutionalization: Motivation

## ■ Why Replace Fully Connected Layers for Segmentation?

### • The FC-to-Conv Transformation (Convolutionalization)

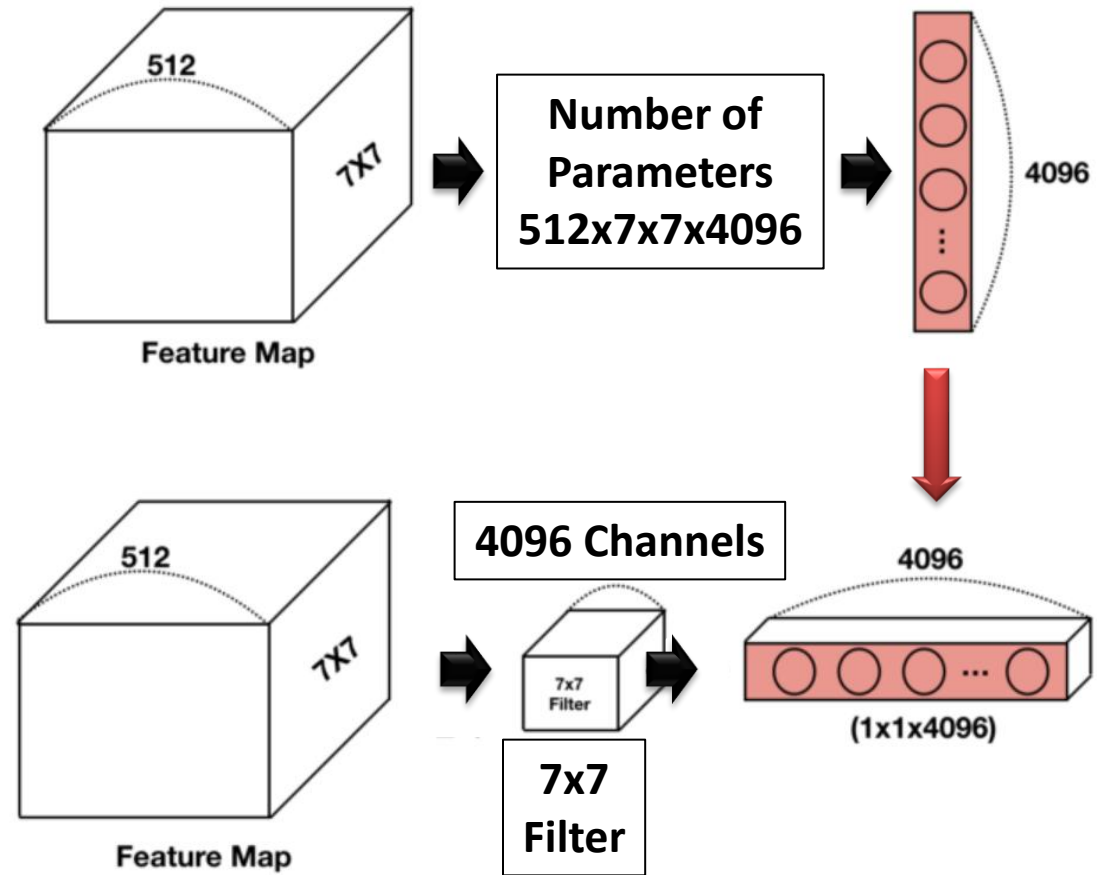
#### ○ Example: Replacing FC Layers with Convolutional Layers

##### ✓ Problem with FC Layers

- Fixed input dimension (e.g., VGG-16 requires a vector of length **4096**).
- Spatial coordinates are lost when flattening.

##### ✓ Key Idea

- Treat an FC layer as a **convolution** with a kernel covering the entire input feature map.
- Replace
  - First FC layer  
→ **7×7 Conv** with 4096 filters.
  - Final FC layer  
→ **1×1 Conv** with channels = number of classes.



# Convolutionalization: Motivation

## ■ Why Replace Fully Connected Layers for Segmentation?

### • The FC-to-Conv Transformation (Convolutionalization)

#### ○ Example: Replacing FC Layers with Convolutional Layers

##### ✓ VGG-16 Example

##### ➤ Before

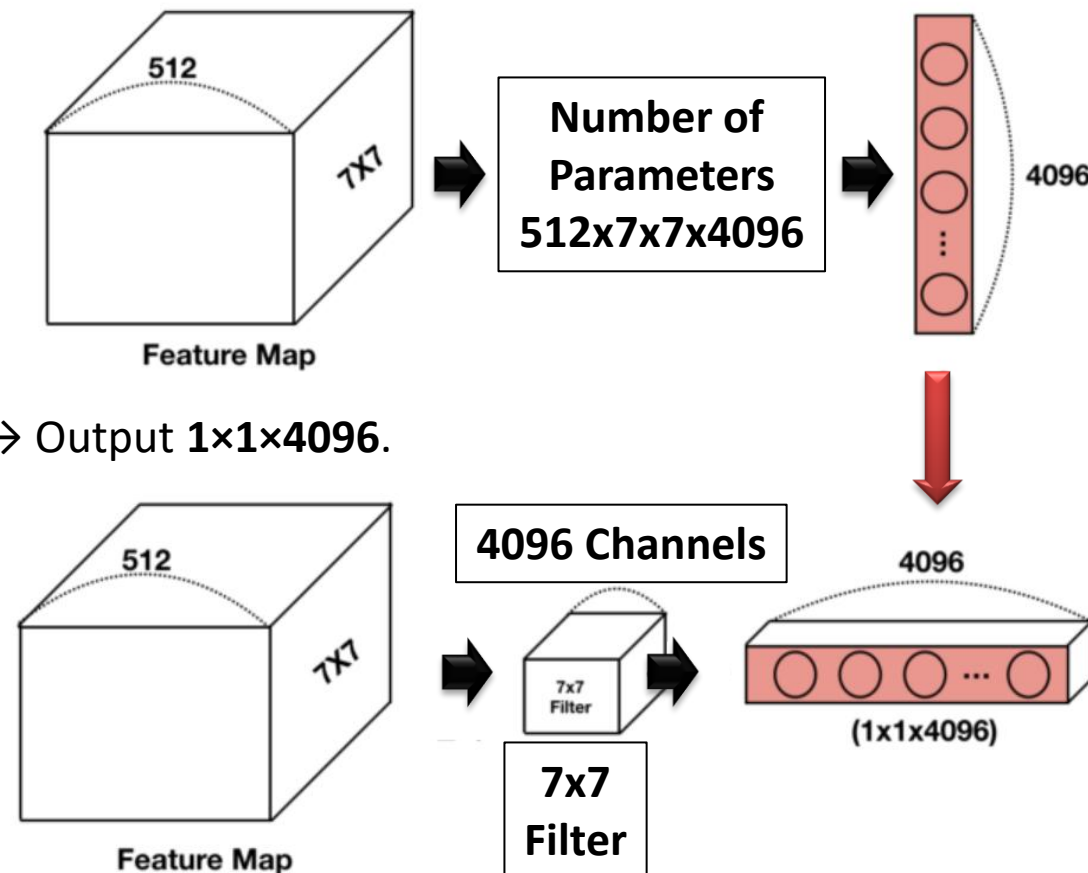
- Input feature map: **7×7×512**.
- Flatten to  $1 \times 1 \times (512 \times 7 \times 7) = 1 \times 1 \times 25088$ .
- Fully Connected to 4096 units.

##### ➤ After

- Apply **7×7 Conv** (512 input channels, 4096 filters) → Output **1×1×4096**.
- Spatial structure is preserved for larger inputs.

##### ✓ Advantages

- Works with **any input size** (no FC size constraint).
- Produces **spatial output maps**  
→ essential for segmentation.
- Computation is **faster** for large images  
→  $\approx 5\times$  speedup over patch-by-patch.

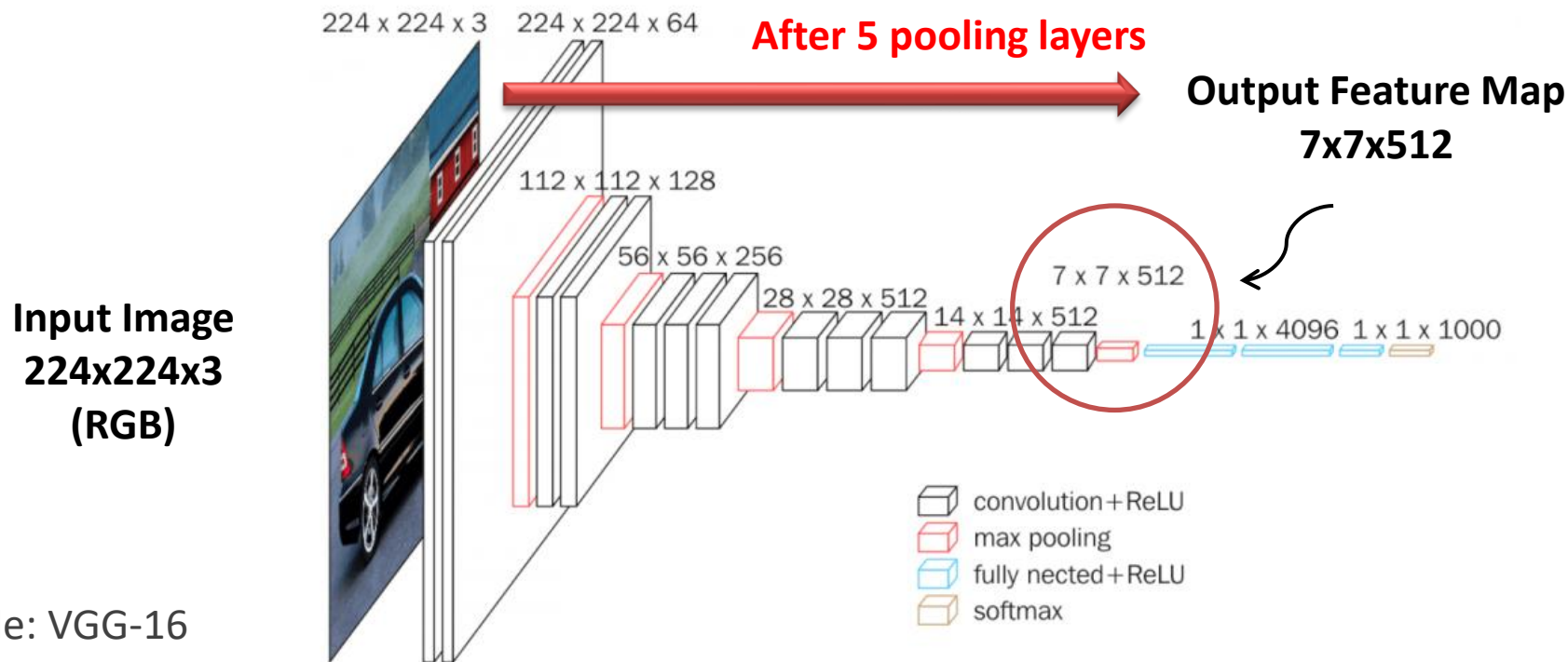


# Coarse Outputs and the Need for Upsampling

## ■ Why Upsampling is Essential for Segmentation

### • 1. Downsampling in CNNs

- Pooling layers and strided convolutions progressively reduce spatial resolution.



- Example: VGG-16

✓ Input: **224x224**

✓ After 5 pooling layers (stride 2 each) → Output feature map: **7x7**

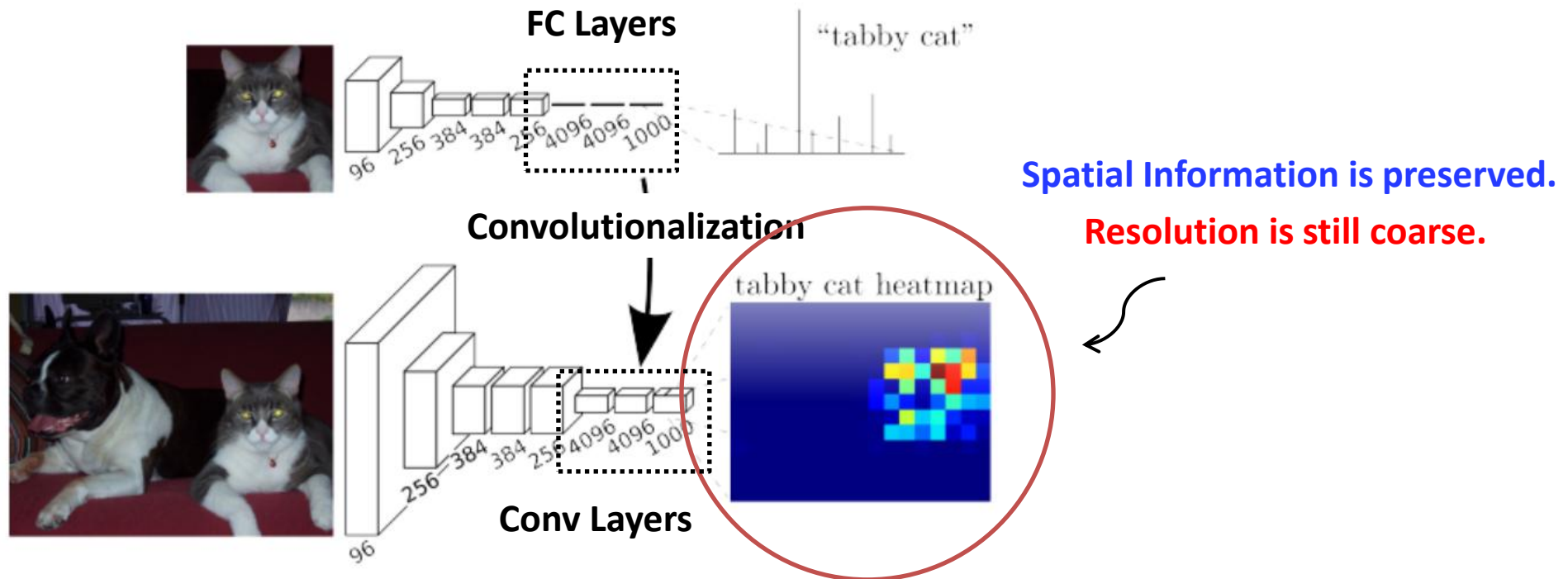
✓ This is a **stride 32** reduction.



# Coarse Outputs and the Need for Upsampling

## ■ Why Upsampling is Essential for Segmentation

### • 2. The Problem for Segmentation



- Fully convolutional conversion preserves spatial arrangement, but resolution is still coarse.
- For pixel-wise prediction, we need **near-original resolution** output.
- Without upsampling, fine details and object boundaries are lost.

# Upsampling with Learnable Deconvolution

## ■ From Coarse to Dense: Deconvolution Layers

### • 1. Upsampling Methods

#### ○ Fixed

##### ✓ Bilinear Interpolation

➤ Simple, non-learnable resizing.

##### ✓ Unpooling

➤ Reverses pooling using stored indices.

#### ○ Learnable

##### ✓ Deconvolution

(=Transposed Convolution, Backwards Convolution)

➤ Parameters learned during training.

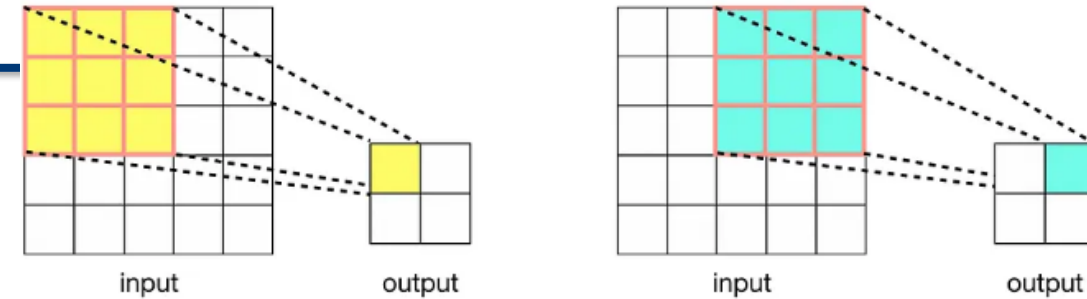
### • 2. Backwards Convolution Concept

○ Standard convolution (stride > 1) **reduces** spatial size (downsampling).

○ Reversing this process **increases** spatial size (upsampling).

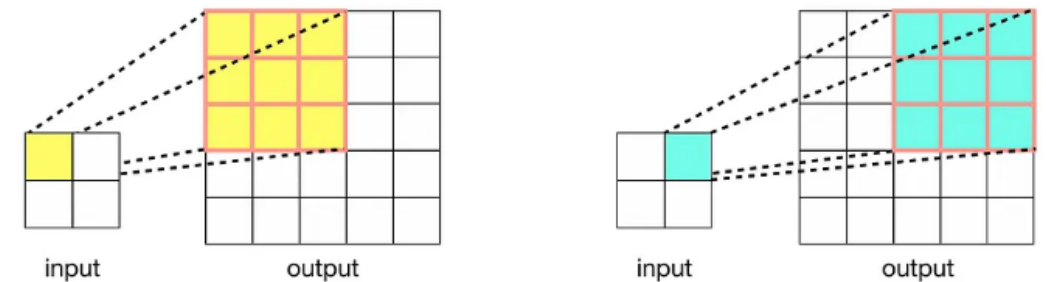
○ Uses **learnable filters** so that the upsampling is **task-optimized**.

### Convolution



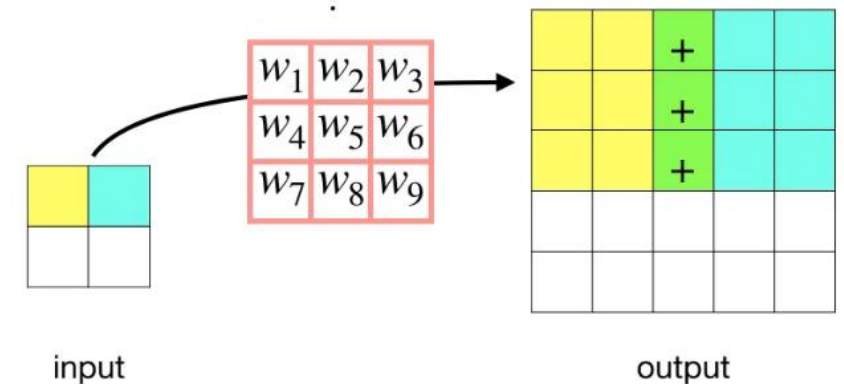
Filter: 3x3 Stride:2

### Deconvolution



Filter: 3x3 Stride:2

### Overlapped Parameters are added.



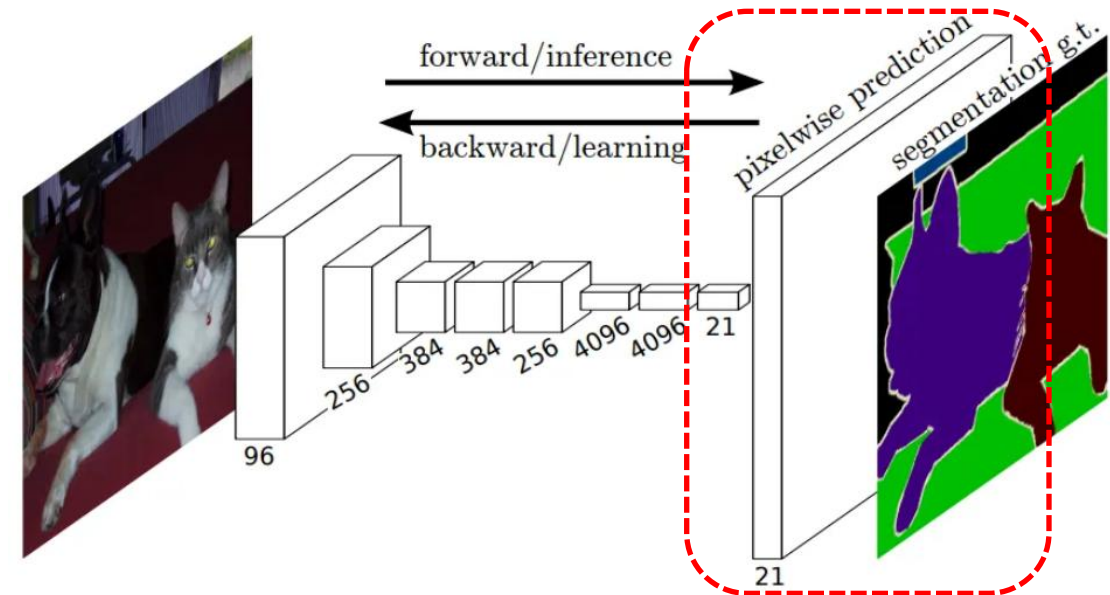
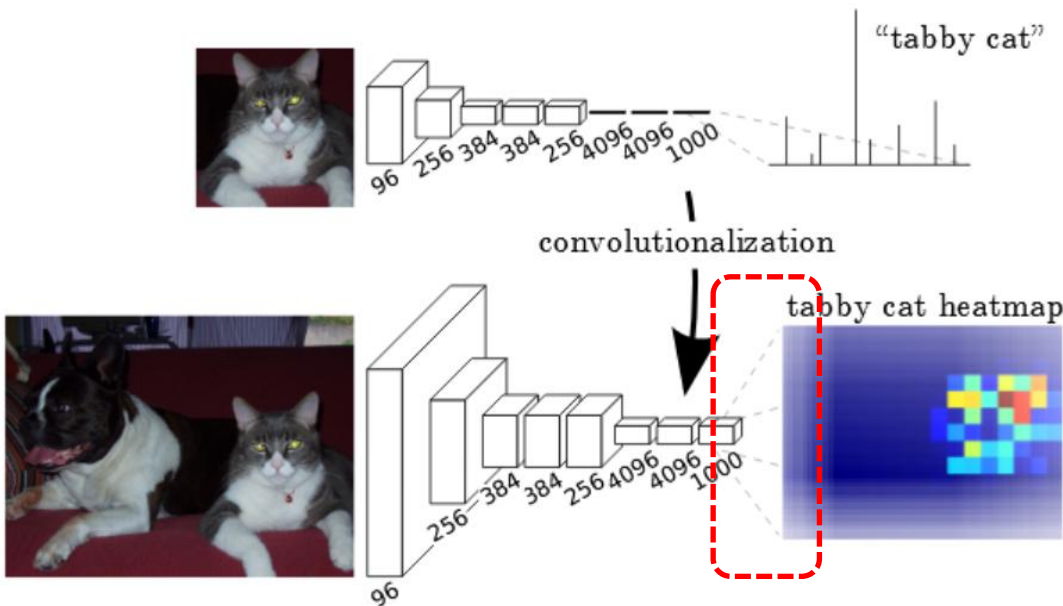
# Upsampling with Learnable Deconvolution

## ■ From Coarse to Dense: Deconvolution Layers

### • How FCN Uses It

- FCNs employ deconvolution layers for upsampling coarse feature maps to the input size.
- Initial weights are set to **bilinear interpolation** values.
- During training, these weights are updated to improve segmentation accuracy.
- Allows **end-to-end training** with pixel-wise loss.

**Deconvolution  
(i.e., upsampling)**



# The Problem with Coarse Outputs

## ■ Why Coarse Outputs are Not Enough

- **Deconvolution alone is insufficient**

- Even with trainable deconvolution, a coarse feature map (e.g., stride 32) lacks detailed spatial information.

- **Loss of detail**

- Large stride means much information is lost during down sampling, leading to blurry and imprecise boundaries.

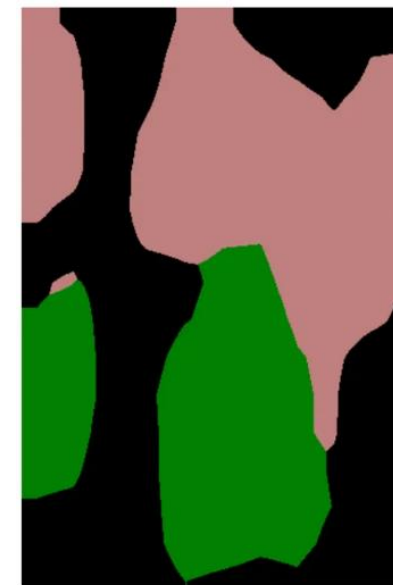
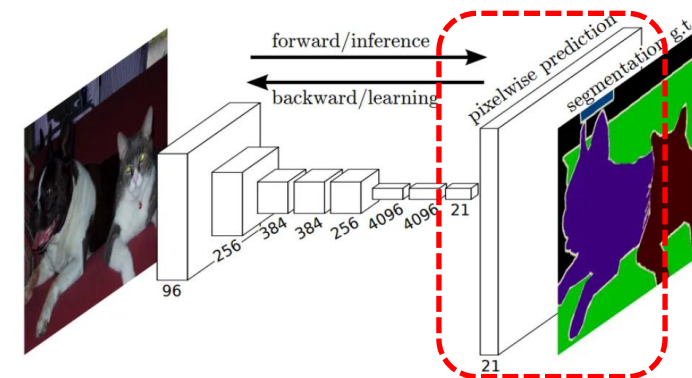
- **Example**

- FCN-32s often produces smooth, rounded edges rather than accurate object contours.

- **Reason**

- The receptive field covers a large area, so fine-grained boundaries from the original image are not preserved.

**Deconvolution  
(i.e., upsampling)**



**Prediction**



**Ground truth**

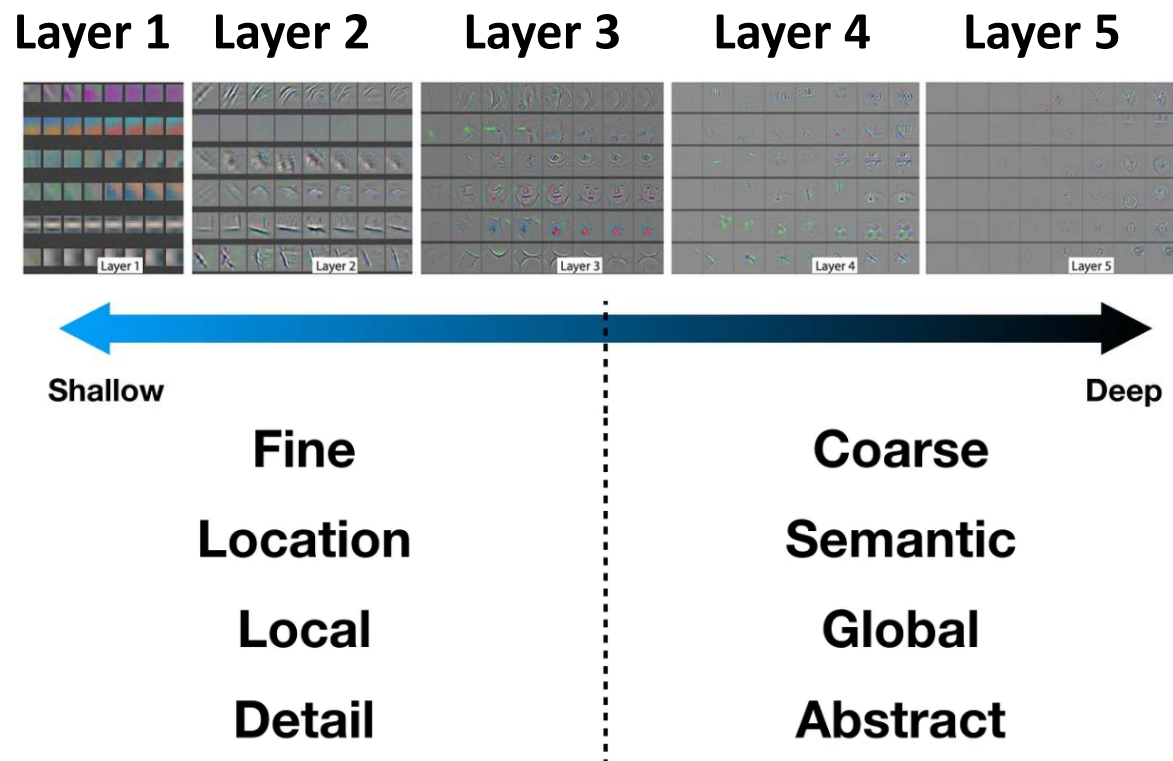
**Loss of detail**



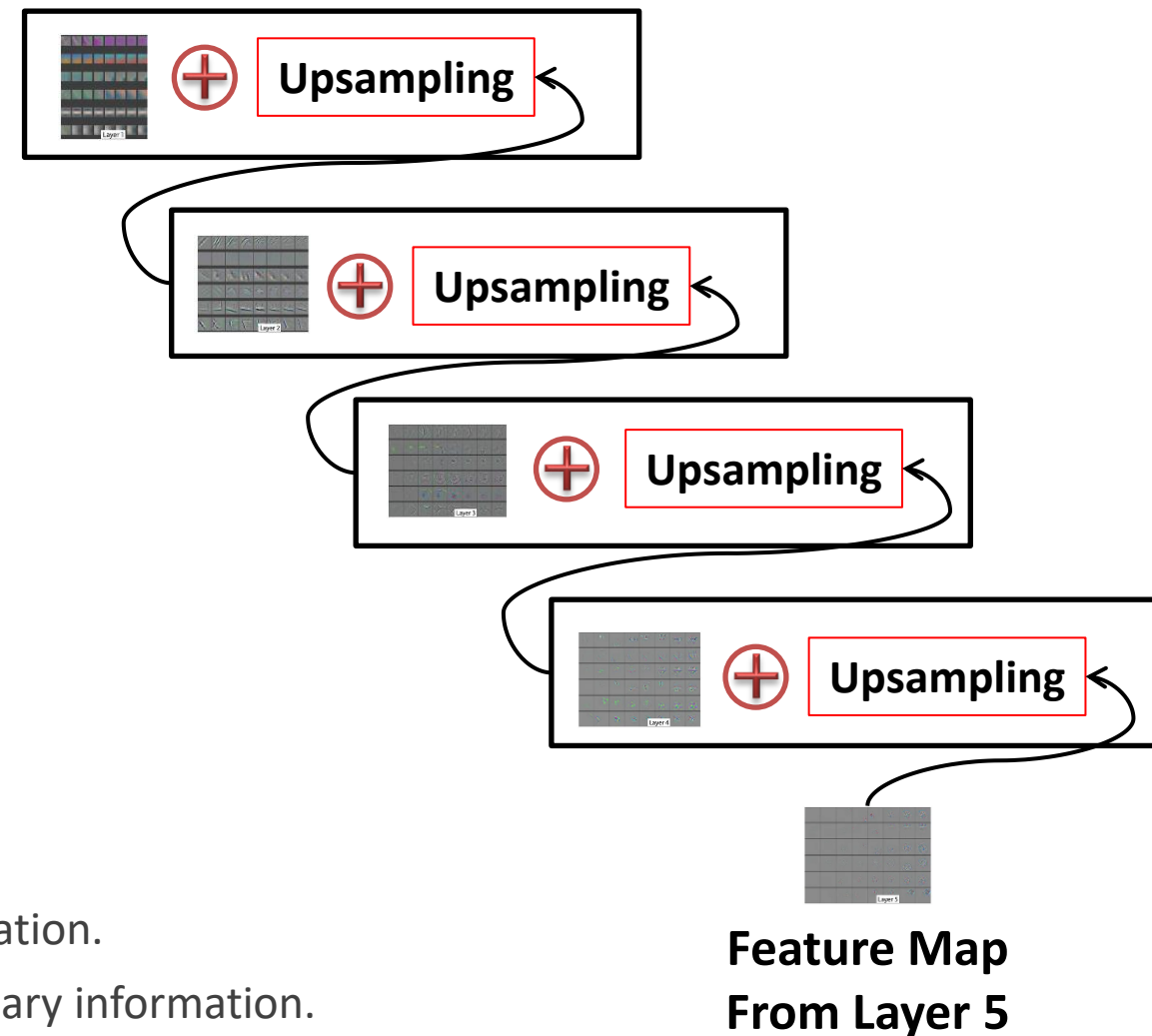
# Skip Architecture

## ■ Skip Architecture for Better Segmentation

- Core idea – Fuse



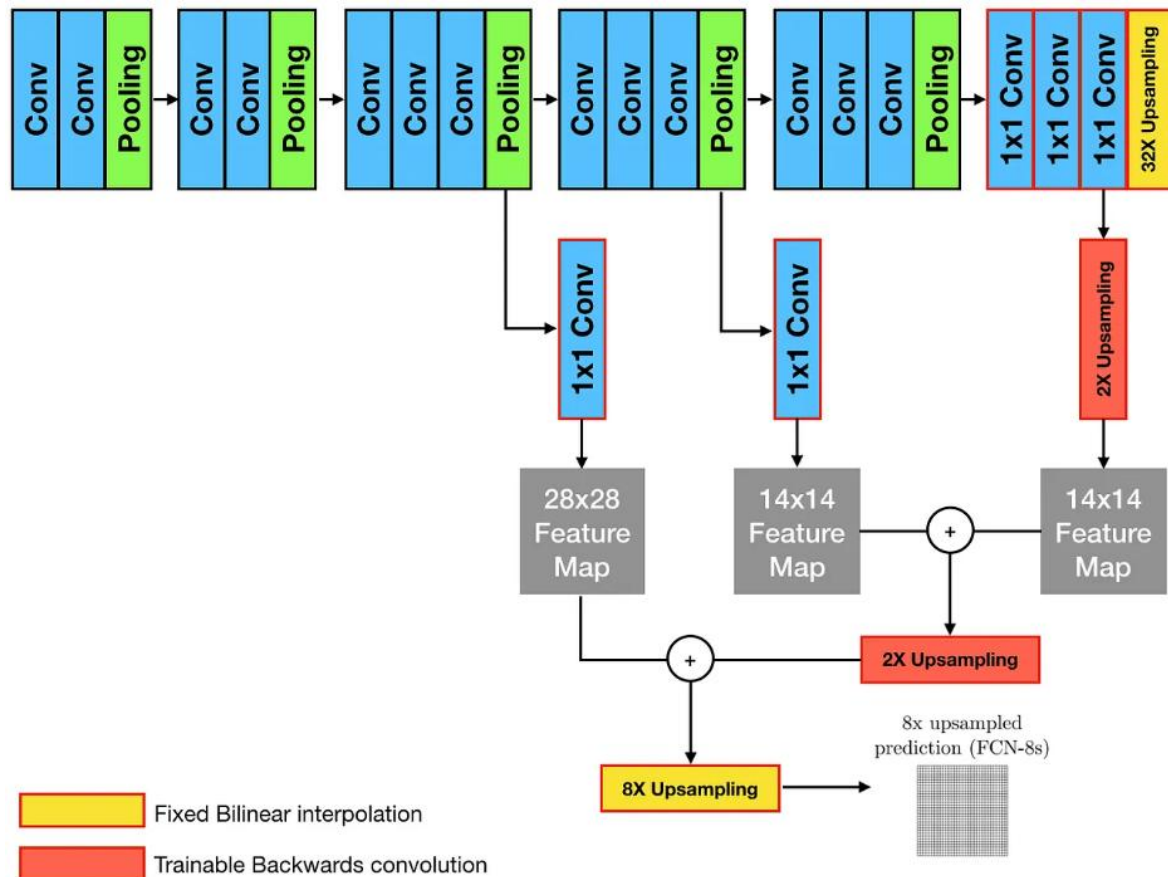
- **Deep & coarse layers** → rich semantic (meaning) information.
- **Shallow & fine layers** → detailed appearance and boundary information.



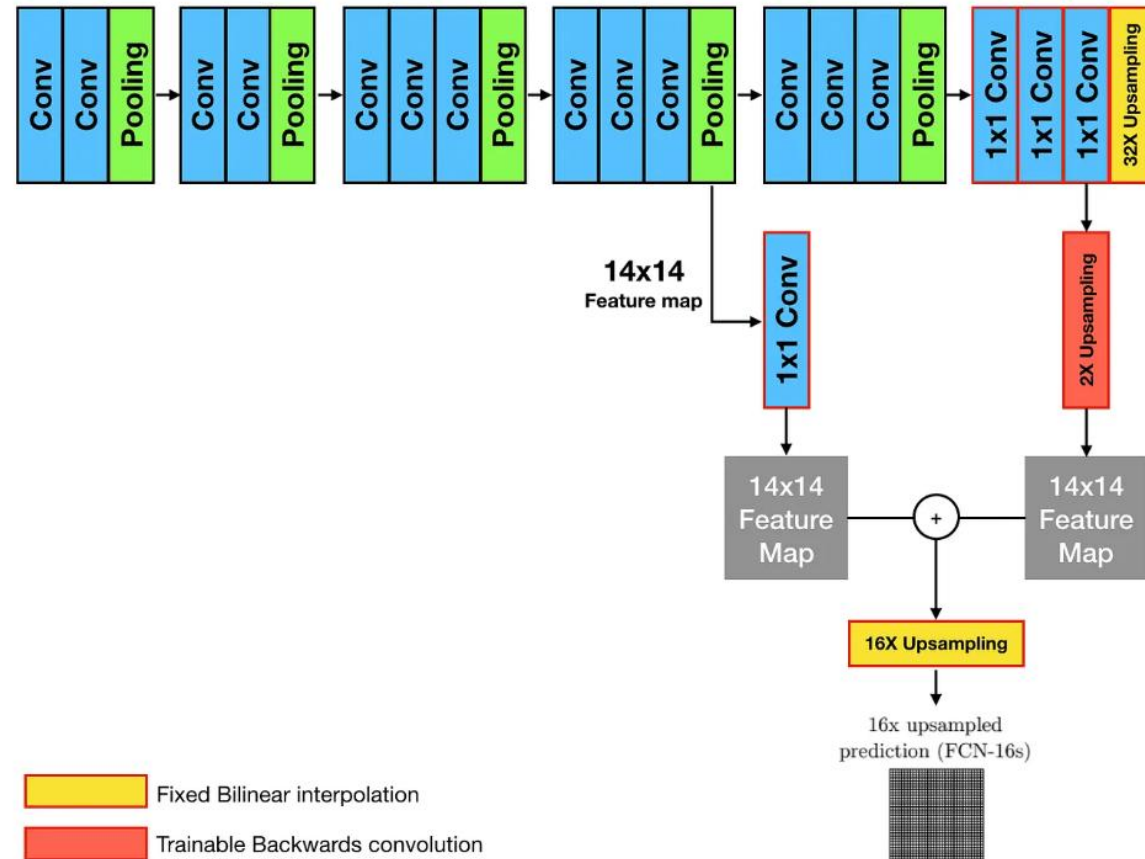
# Skip Architecture

## ■ Skip Architecture for Better Segmentation

### • Architecture variants

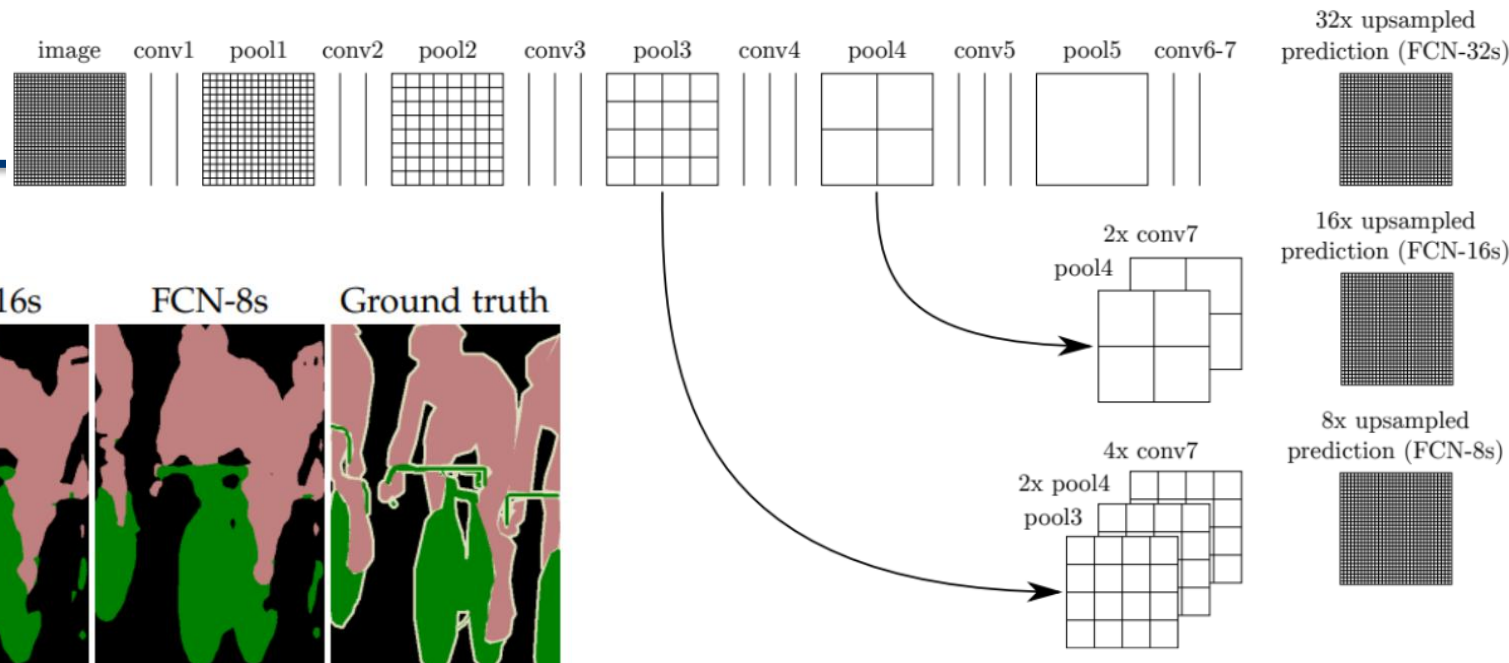


**FCN-8s:** Final + pool4 + pool3  
(stride 8) — most detailed boundaries.



**FCN-16s:** Final + pool4 layer  
(stride 16) — better boundaries.

# FCN Results and Key Takeaways



## FCN Performance Highlights

### Performance Trend

- **FCN-32s → FCN-16s → FCN-8s**: Steady improvement in segmentation accuracy.
- Finer strides capture more precise object boundaries.

### Quantitative Impact

- Significant mean IU improvement over prior state-of-the-art.
- ~20% relative gain on benchmarks such as **PASCAL VOC 2011/2012**, **NYUDv2**, and **SIFT Flow**.

### Legacy

- Provided the foundation for later architectures (U-Net, SegNet, DeepLab, etc.).

# Introduction to U-Net

---

- **U-Net: Convolutional Networks for Biomedical Image Segmentation**

- Published at **MICCAI 2015** by Olaf Ronneberger et al.
- Designed for **biomedical image segmentation** with **limited training data**.
- Key innovation: **U-shaped architecture** combining
  - **Contracting Path** → capture context
  - **Expansive Path** → enable precise localization
- Outperformed previous state-of-the-art on ISBI 2015 challenges.

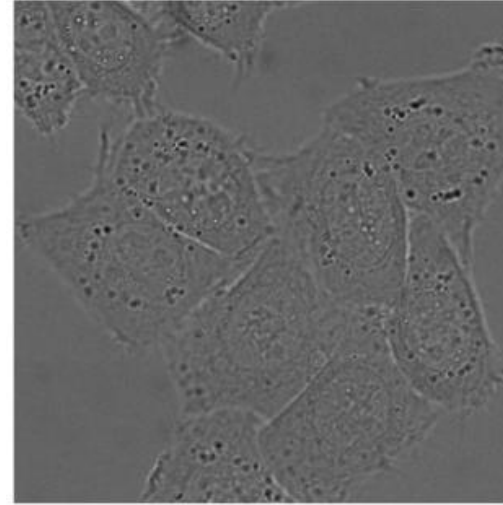


# Motivation – U-Net

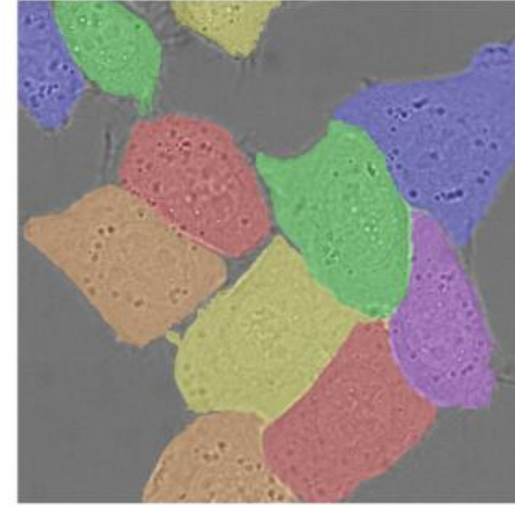
## ■ Why U-Net?

### • 1. Problem in Biomedical Segmentation

- Small, annotated datasets
- Need for **pixel-level** classification (semantic segmentation)
- Critical to preserve both
  - ✓ **Context**
    - Global semantic understanding (what is in the image)
  - ✓ **Localization**
    - Precise spatial boundaries (where the object is)



Biomedical Image



Ground-truth (Label)

### • 2. Limitations of Sliding-Window CNNs (Patch-Based Methods)

- **Inefficient computation**: neighboring patches overlap heavily → redundant convolution operations.
- **Context–Localization trade-off**
  - ✓ Large patch → captures context but loses fine detail.
  - ✓ Small patch → keeps detail but misses context.
- Cannot leverage **shared computation** across overlapping regions.

# Motivation – U-Net

## ■ Why U-Net?

### • 3. U-Net's Approach

#### ○ (1) Fully Convolutional

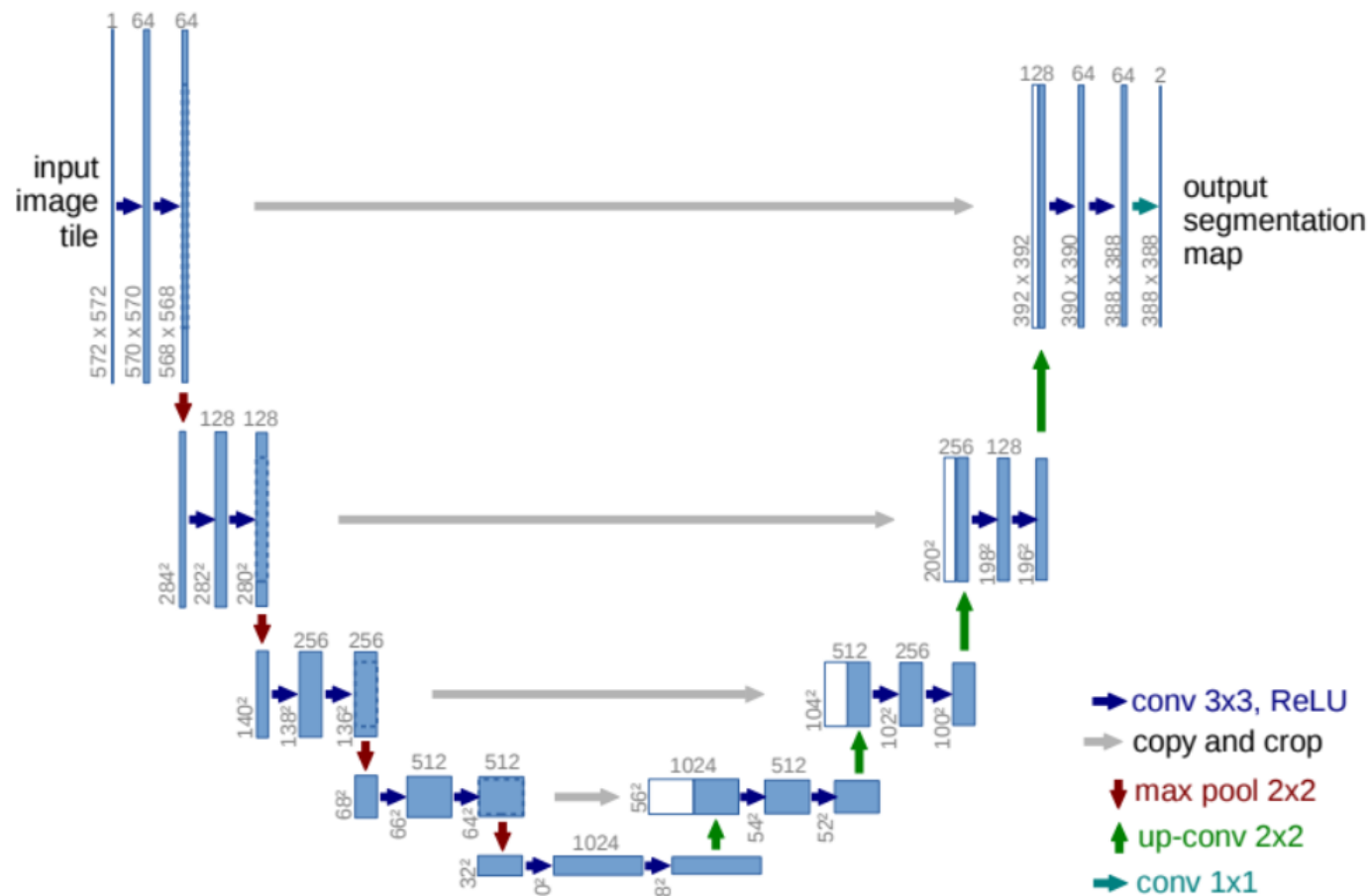
- ✓ Process the whole image at once.

#### ○ (2) Encoder–Decoder Structure

- ✓ Encoder (contracting path) captures context.
- ✓ Decoder (expansive path) restores spatial resolution.

#### ○ (3) Skip Connections

- ✓ Combine fine detail from early layers with deep semantic info.
- ✓ Designed to work **efficiently** with **small biomedical datasets**.



# Architecture Overview

## ■ The U-Shaped Architecture

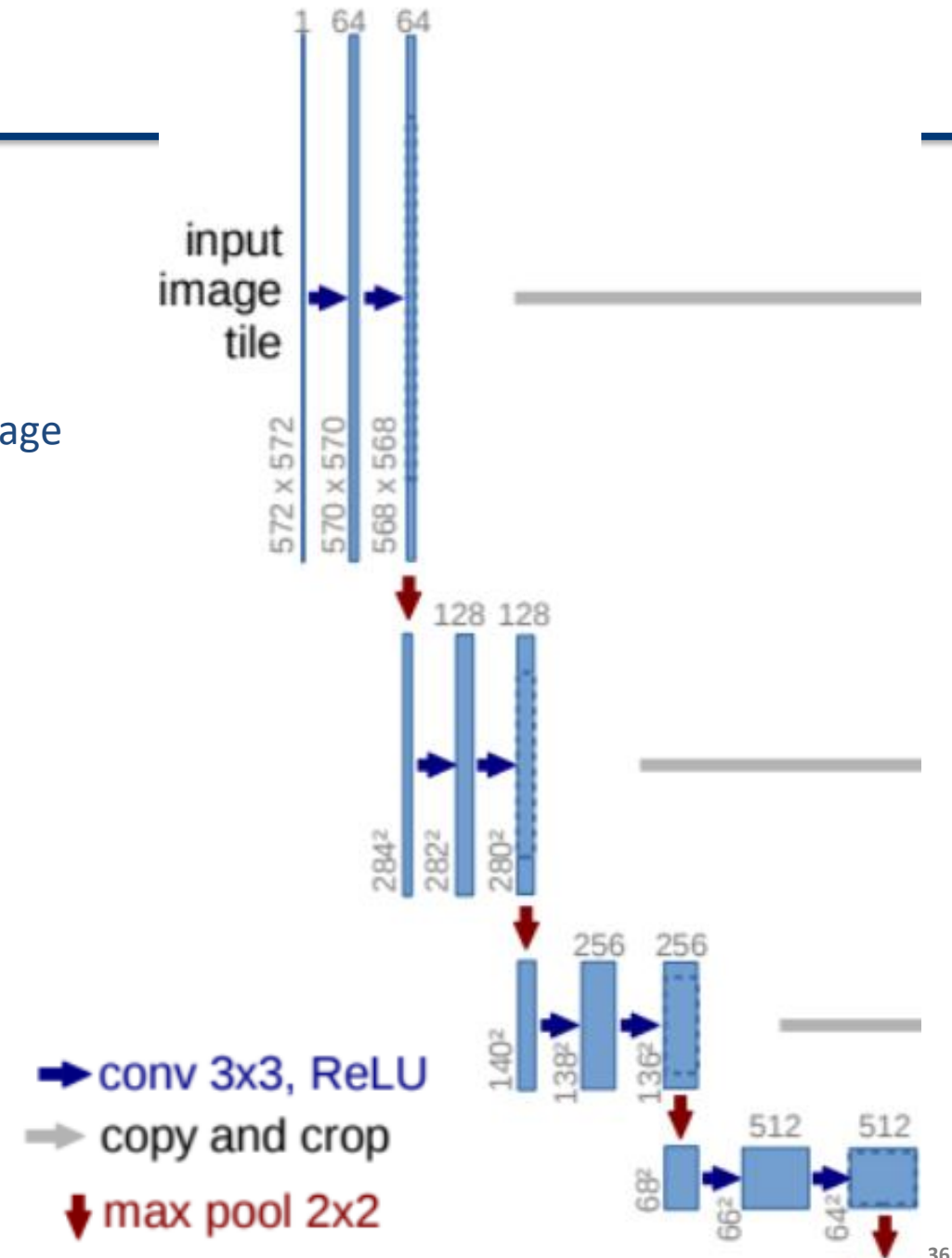
### • 1. Contracting Path (Encoder)

#### ○ Purpose

- ✓ **Capture context** by progressively downsampling the input image while increasing the number of feature channels.

#### ○ Structure per Downsampling Step

- ✓ **Step 1. 3×3 Convolution** (stride=1, valid padding) → ReLU  
→ *(optional) BatchNorm*
  - No padding → feature map shrinks by 2 pixels per conv in each spatial dimension.
- ✓ **Step 2. 3×3 Convolution** (stride=1, valid padding) → ReLU  
→ *(optional) BatchNorm*
- ✓ **Step 3. 2×2 Max Pooling** (stride=2) for downsampling.

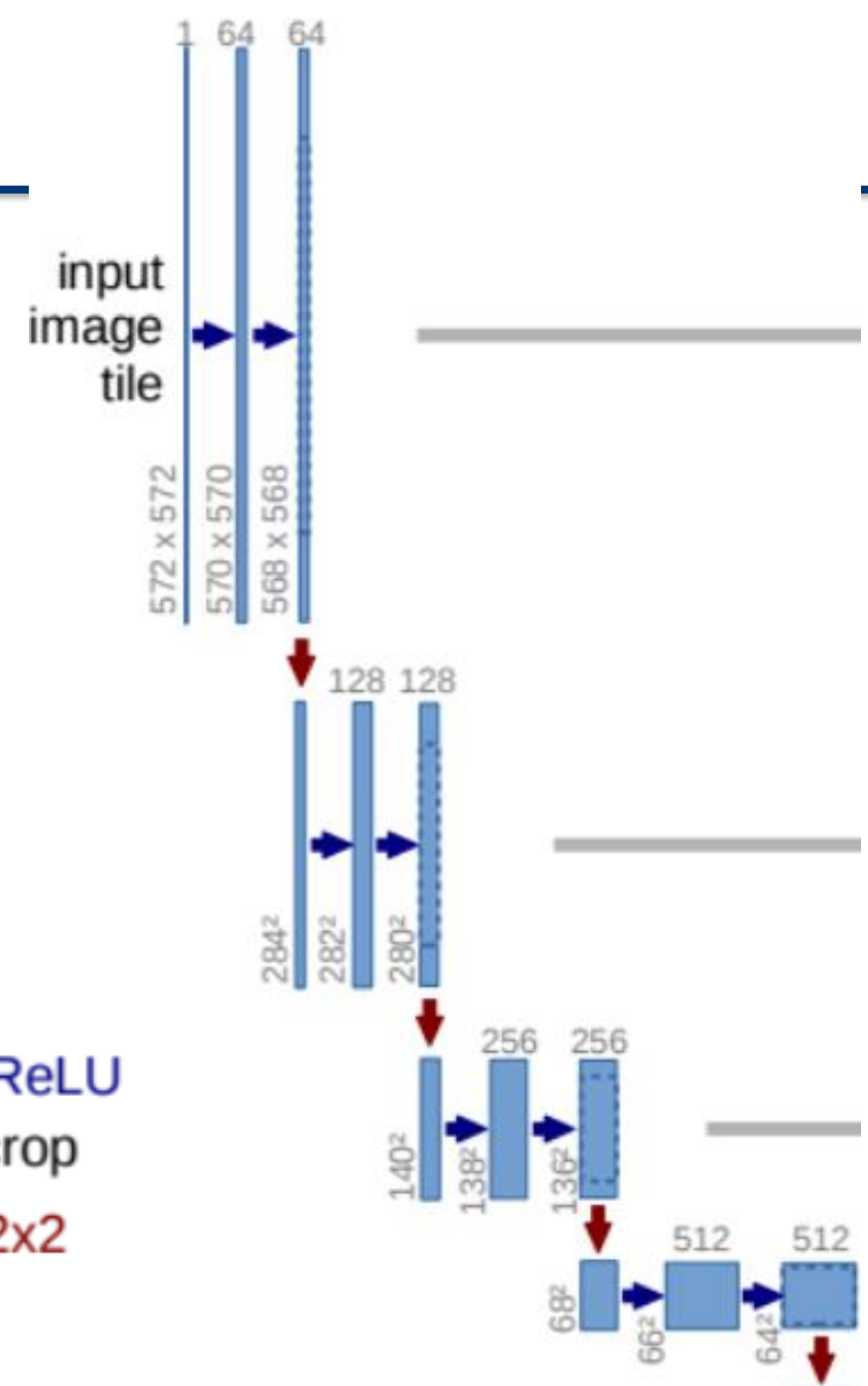
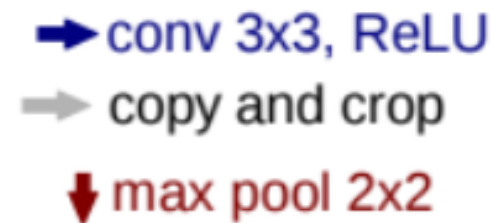


# Architecture Overview

## ■ The U-Shaped Architecture

### • 1. Contracting Path (Encoder)

- After **each** downsampling step, the number of channels **doubles**  
✓ E.g.,  $1 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 1024$  (bottleneck).
- Pooling **increases the receptive field**, allowing the network to capture more context from the input image.



# Architecture Overview

## ■ The U-Shaped Architecture

### • 2. Bottleneck

#### ○ Purpose

✓ **Bridge between encoder and decoder**, containing the most abstract representation of the input.

#### ○ Structure



✓ **3x3 Convolution** (stride=1, valid padding) → ReLU → *(optional) BatchNorm*

✓ **3x3 Convolution** (stride=1, valid padding) → ReLU → *(optional) BatchNorm*

#### ○ **Dropout Layer** (optional in modern versions; not in original)

✓ Helps prevent overfitting and improves robustness to noise.

#### ○ This layer has the **highest channel count** and smallest spatial resolution.



# Architecture Overview

## ■ The U-Shaped Architecture

### • 3. Expansive Path (Decoder)

#### ○ Purpose

✓ **Recover spatial resolution** and produce dense

- Pixel-level predictions by combining **contextual features** from the encoder with **localization features** from early layers.

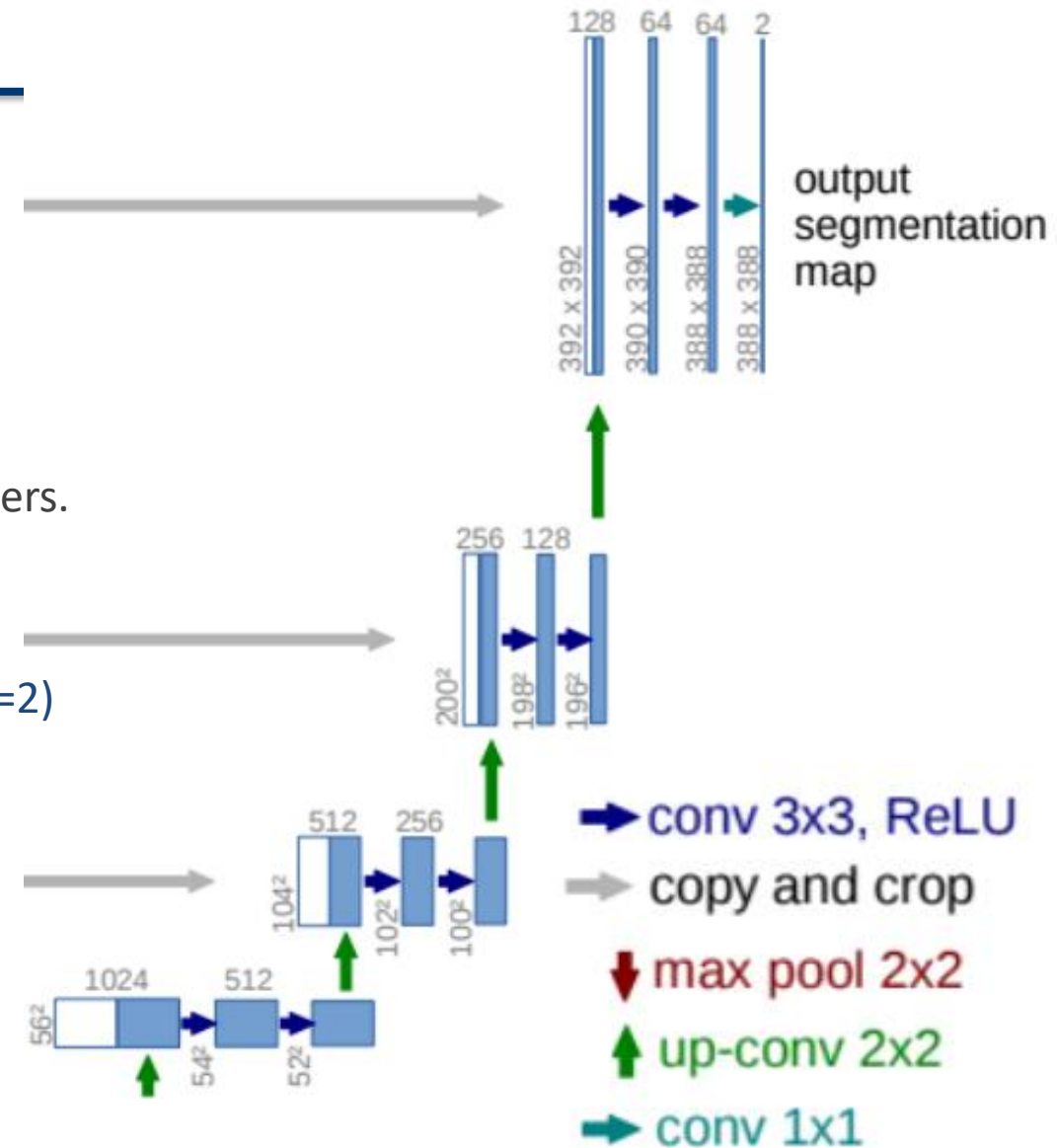
#### ○ Structure per Upsampling Step

✓ **Step 1. 2×2 Transposed Convolution (Deconvolution)** (stride=2)

- Upsamples the feature map by a factor of 2.
- Halves the number of channels.

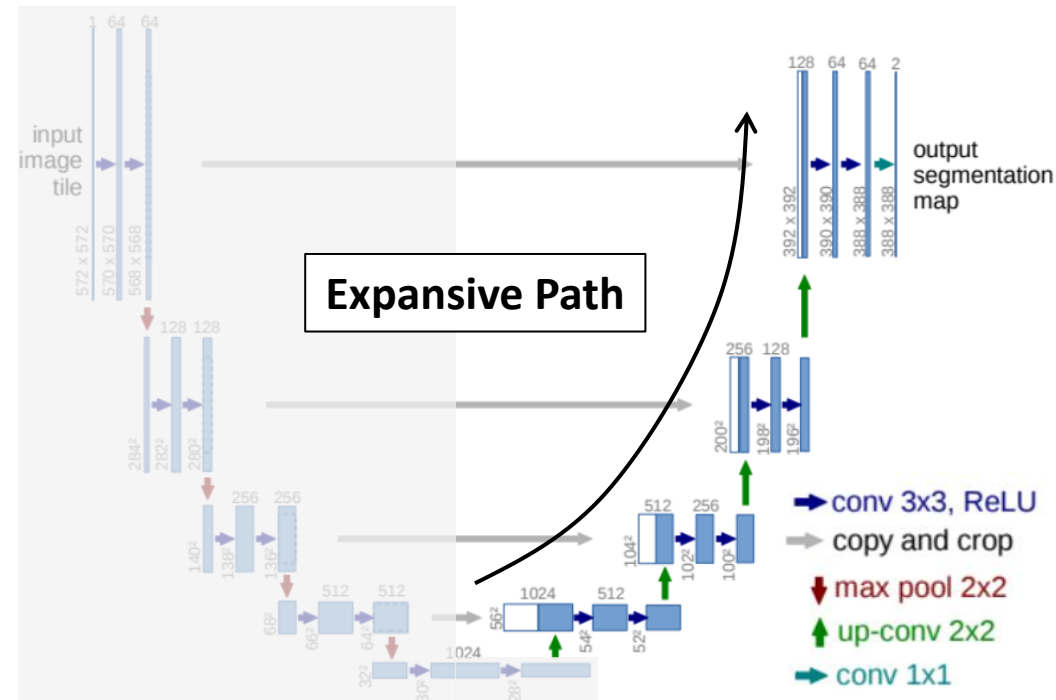
✓ **Step 2. Crop & Concatenate**

- Take the corresponding feature map from the encoder (same depth level).
- Because of **no padding** in 3×3 convolutions, encoder feature maps are slightly smaller → **crop** to match dimensions before concatenation.



# Architecture Overview

- The U-Shaped Architecture
  - 3. Expansive Path (Decoder)



## ○ Purpose

✓ **Recover spatial resolution** and produce dense

- Pixel-level predictions by combining **contextual features** from the encoder with **localization features** from early layers.

# Architecture Overview

## ■ The U-Shaped Architecture

### • 3. Expansive Path (Decoder)

#### ○ Structure per Upsampling Step

##### ✓ Step 1. 2×2 Transposed Convolution (Deconvolution) (stride=2)

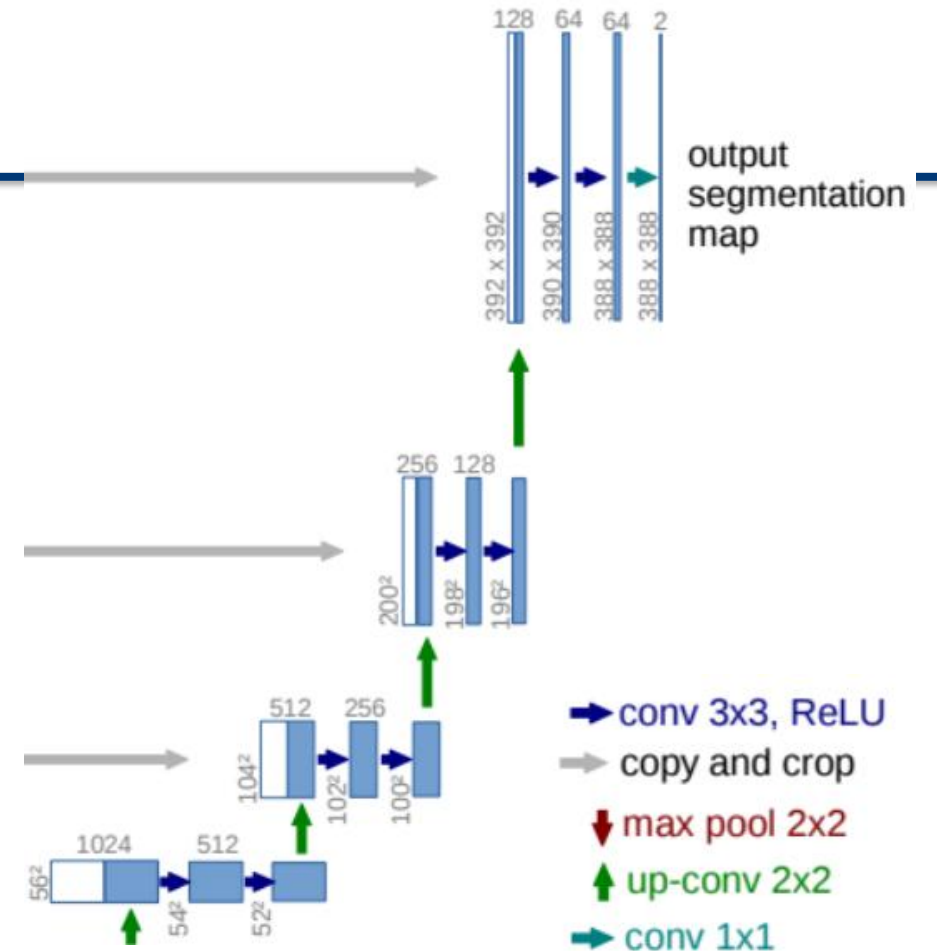
- Upsamples the feature map by a factor of 2.
- Halves the number of channels.

##### ✓ Step 2. Crop & Concatenate

- Take the corresponding feature map from the encoder (same depth level).
- Because of **no padding** in 3×3 convolutions, encoder feature maps are slightly smaller → **crop** to match dimensions before concatenation.

##### ✓ Step 3. 3×3 Convolution (stride=1, valid padding) → ReLU → (optional) BatchNorm

##### ✓ Step 4. 3×3 Convolution (stride=1, valid padding) → ReLU → (optional) BatchNorm

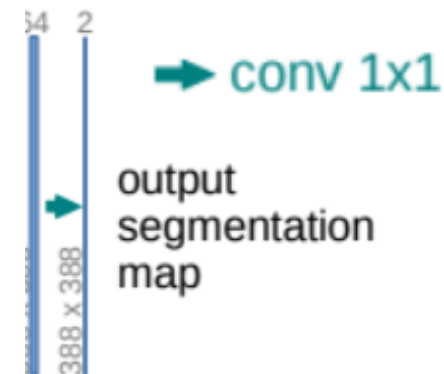


# Architecture Overview

## ■ The U-Shaped Architecture

### • 4. Output Layer

- **1x1 Convolution** with **C filters**, where  $C$  = number of segmentation classes.
- Produces a  $(H \times W \times C)$  score map, where each pixel has a vector of class probabilities.



### • 5. Skip Connections

#### ○ Purpose

✓ Preserving Spatial Detail with Skip Connections

✓ Without skips

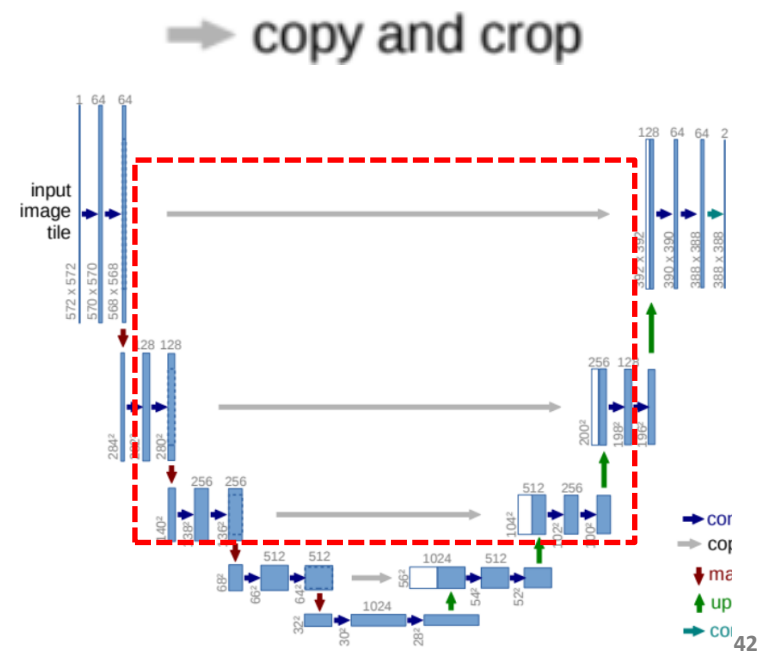
➤ decoder relies solely on deep, coarse features → blurry outputs.

✓ With skips – combine

➤ Low-level appearance features (from early layers)

➤ High-level semantic features (from deep layers)

- Requires **crop & copy** due to valid convolutions reducing feature map size.

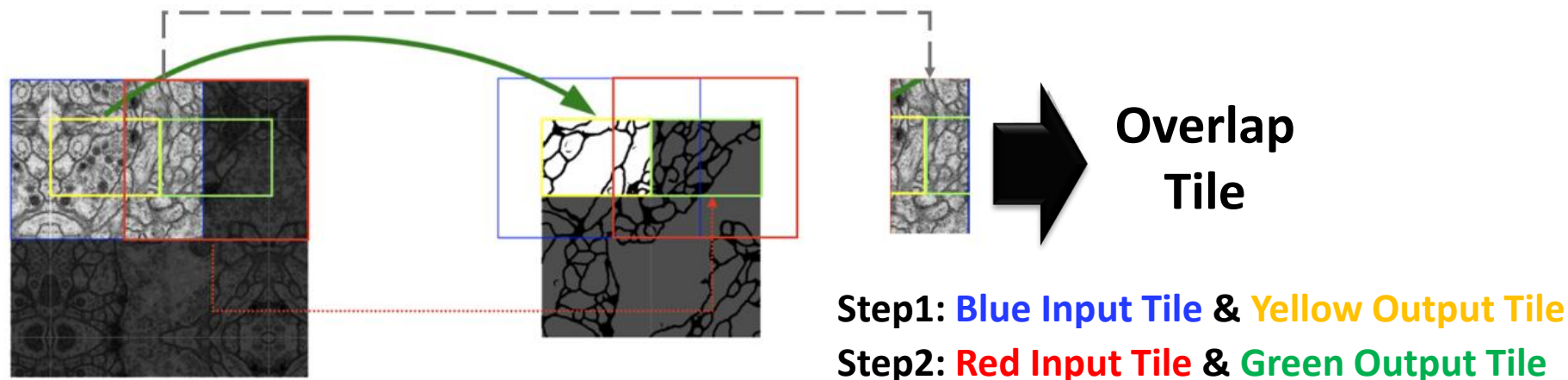


# Training Strategies in U-Net

## ■ Key Training Strategies for High-Quality Segmentation

### • Strategy 1. Overlap-Tile Strategy

- **Purpose:** Handle large images and ensure predictions are accurate at boundaries.



- **How it works**

- ✓ Split large input images into smaller tiles with overlapping regions.
- ✓ Each tile is fed into the network, producing a smaller output region due to valid padding in convolutions.
- ✓ Overlap ensures full coverage and avoids loss of context at the borders.

- **Key Benefit**

- ✓ Maintains context for edge regions and prevents border artifacts.

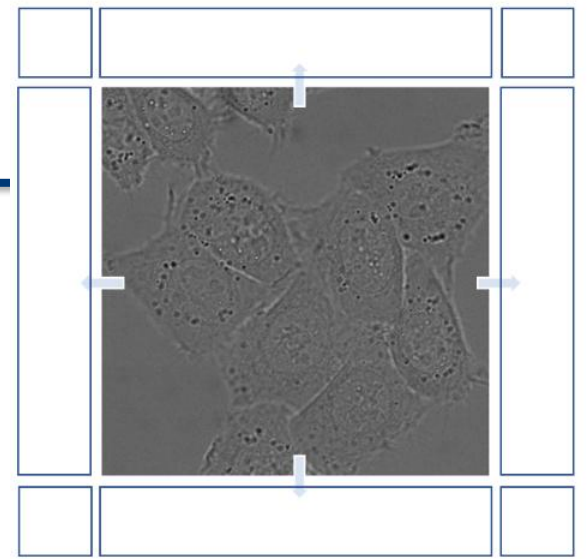


# Training Strategies in U-Net

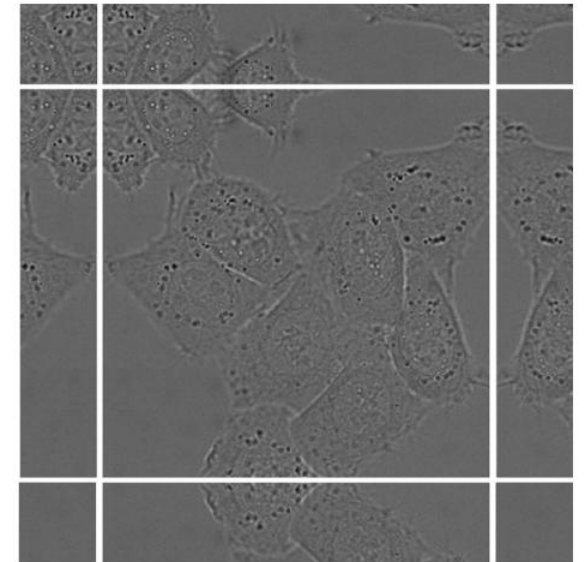
## ■ Key Training Strategies for High-Quality Segmentation

### • Strategy 2. Mirroring Extrapolation

- **Purpose:** Provide meaningful context for border pixels without introducing artificial padding values.
- **How it works**
  - ✓ Extend the input image by mirroring its borders.
  - ✓ This mirrored extension surrounds the original image before tiling.
- **Why it's useful in biomedical images**
  - ✓ Many biological structures (e.g., cells) have symmetrical patterns.
  - ✓ Mirroring preserves realistic textures at the borders, helping the network learn better boundary representations.



Origin Image



After Mirroring

# Training Strategies in U-Net

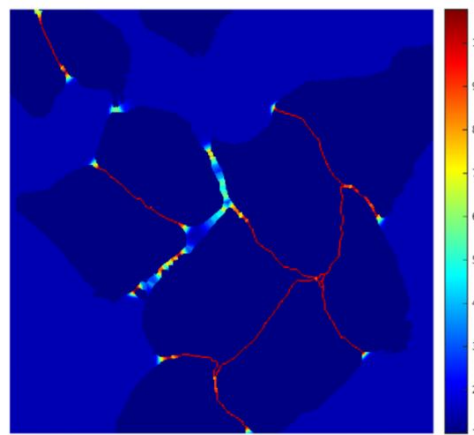
## ■ Key Training Strategies for High-Quality Segmentation

### • Strategy 3. Weighted Loss for Boundary Separation

- **Purpose:** Improve segmentation in regions where objects touch or have thin boundaries.

- **How it works**

Segmented Image



Visualized Weight Map

- ✓ Compute a **weight map** from ground truth

- Higher weights near boundaries between touching objects.
- Lower weights elsewhere.

- ✓ Use this weight map in the pixel-wise cross-entropy loss to emphasize difficult boundary pixels.

- **Effect**

- ✓ The network learns to focus on separating closely packed or touching objects.

# Summary & Key Takeaways

---

## ■ FCN & U-Net: Foundations of Modern Segmentation

### • 1. FCN (Fully Convolutional Network)

- **Key Idea:** Replace fully connected layers with convolution layers → accept arbitrary input size.
- **Upsampling:** Learnable deconvolution layers (backwards convolution) to restore resolution.
- **Skip Architecture:** Combines deep semantic info with shallow spatial info for sharper boundaries (FCN-32s → FCN-16s → FCN-8s).
- **Impact:** First end-to-end trainable CNN for dense prediction; influenced most later segmentation models.

### • 2. U-Net

- **Architecture:** Symmetric **U-shape** with **Contracting Path** (context) and **Expansive Path** (localization).
- **Skip Connections:** Concatenate encoder features to decoder features at matching resolution for precise segmentation.
- **Specialized Training Strategies:**
  - ✓ **Overlap-Tile** for large images
  - ✓ **Mirroring Extrapolation** for border pixels
  - ✓ **Weighted Loss** to separate touching objects
- **Impact:** State-of-the-art in biomedical segmentation, strong performance with limited data.

# Summary & Key Takeaways

---

## ■ FCN & U-Net: Foundations of Modern Segmentation

### • 3. Key Insights

- **Preserving spatial information** is crucial for pixel-wise tasks → skip connections are a common solution.
- **Upsampling quality** directly affects boundary accuracy.
- **Transfer learning** and **augmentation** help overcome limited labeled data.