Northeastern University
Khoury College of
Computer Sciences

Advanced Topics 1 - Object Detection

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

Spring 2026

Wednesday | April 1st, 2026

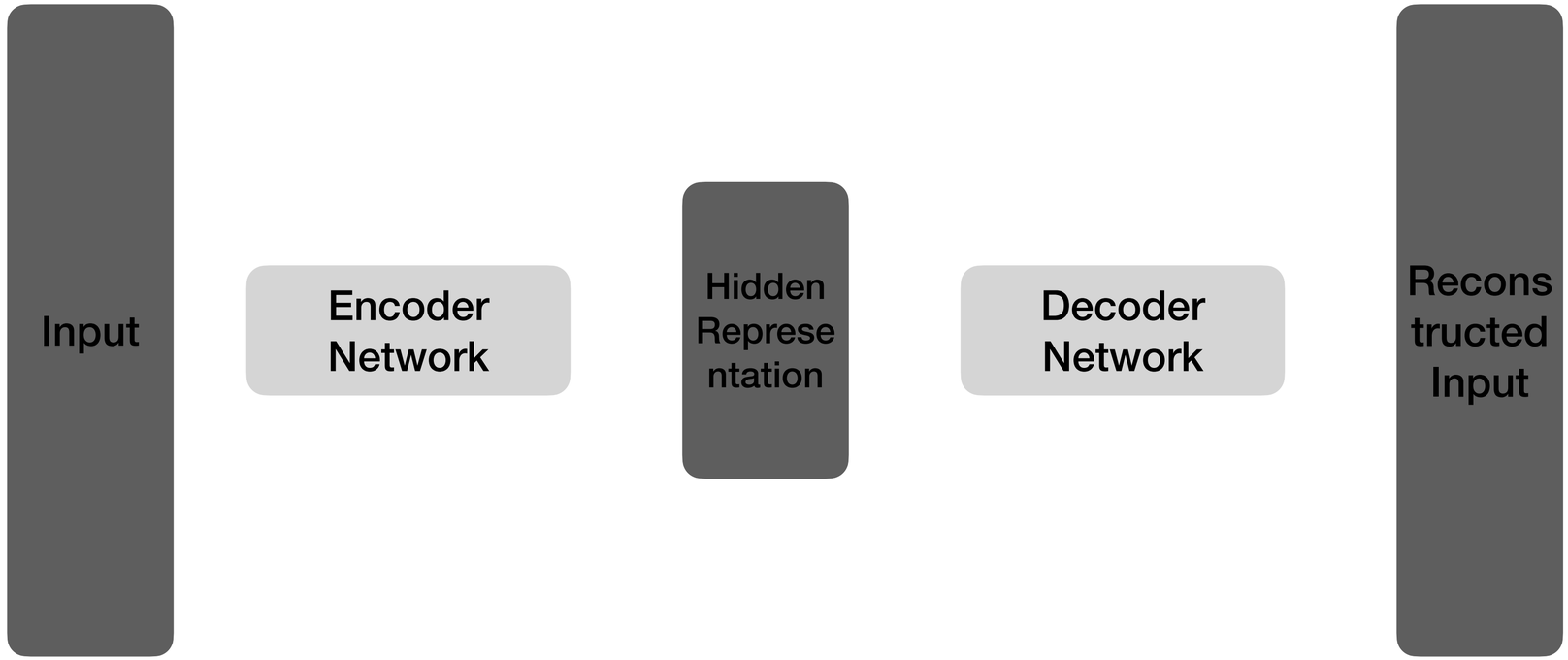
Today's Outline

- Variational Autoencoders
- Object Detection Models - YOLO

Autoencoders

Dimensionality Reduction using Deep Learning

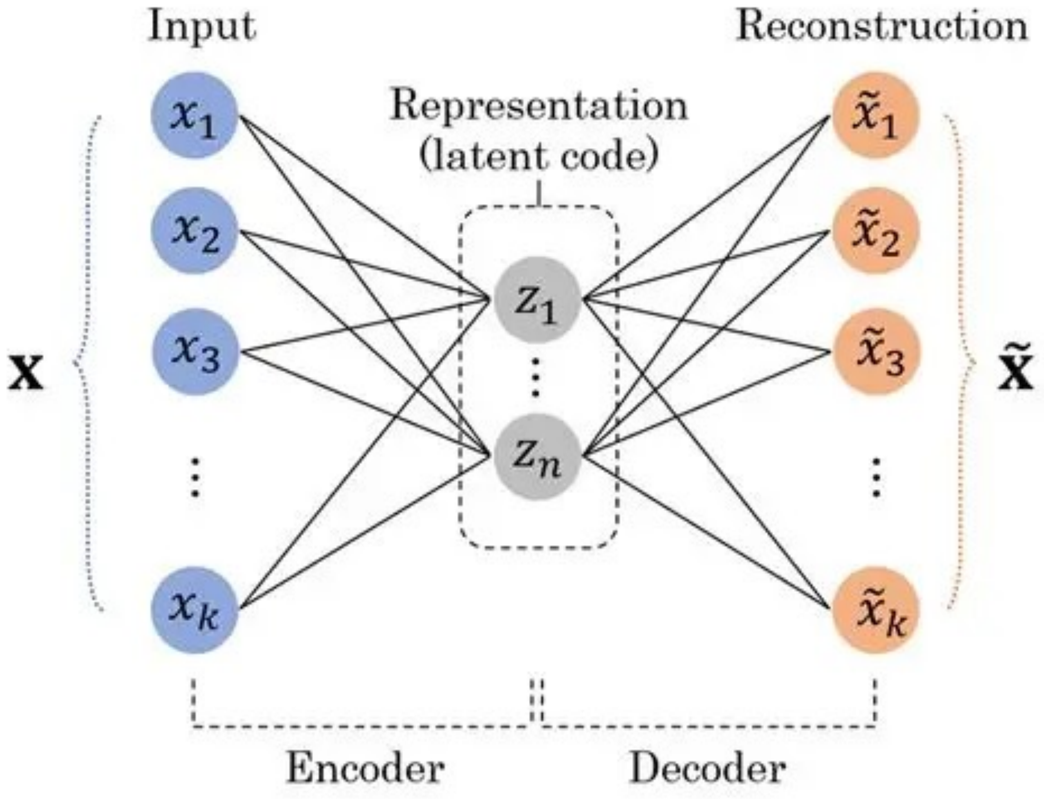
- An autoencoder is a neural network trained to **reconstruct** its input.

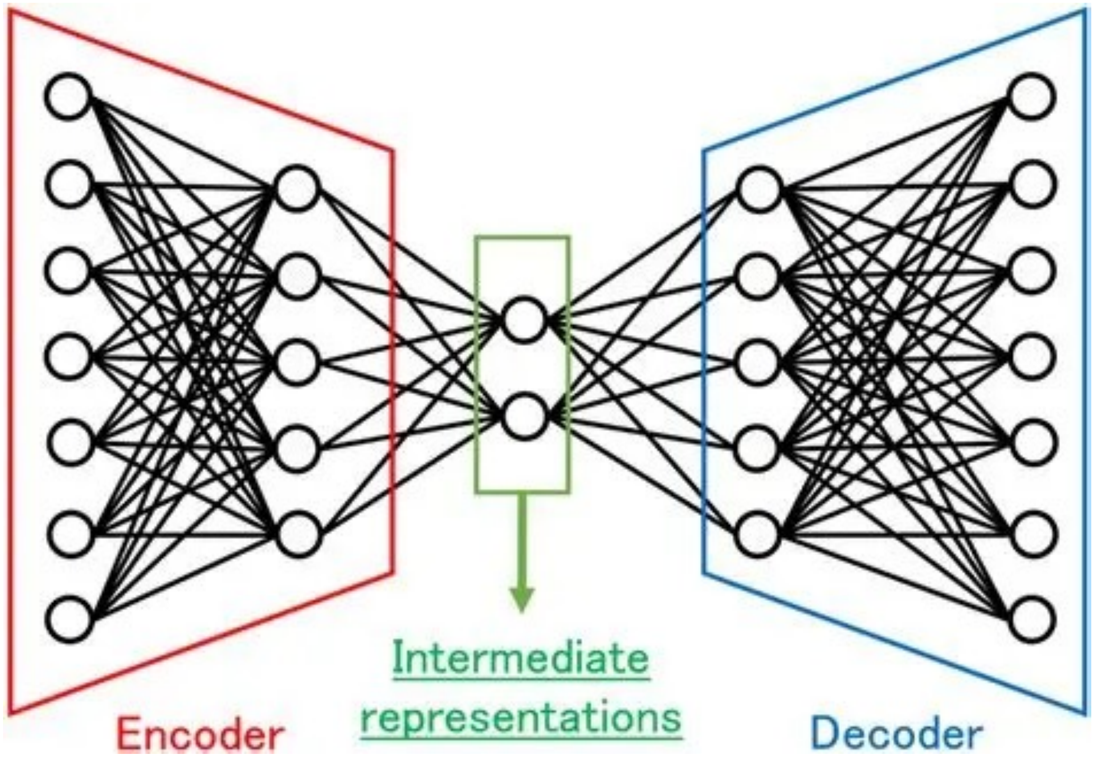


Autoencoders

Dimensionality Reduction using Deep Learning

- An autoencoder is a neural network trained to **reconstruct** its input.

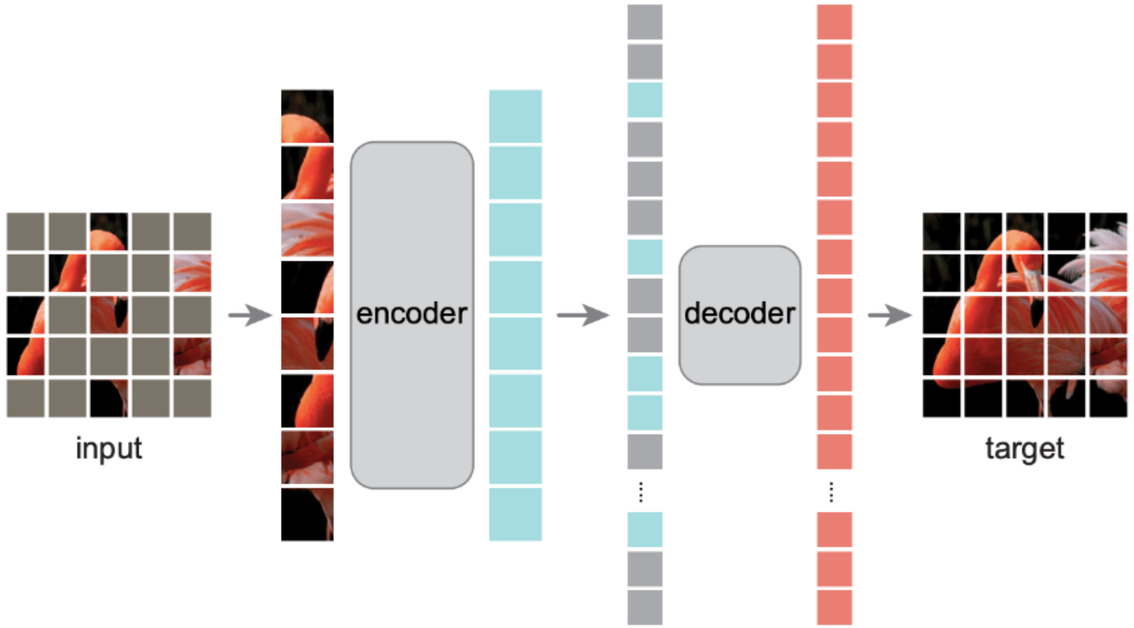


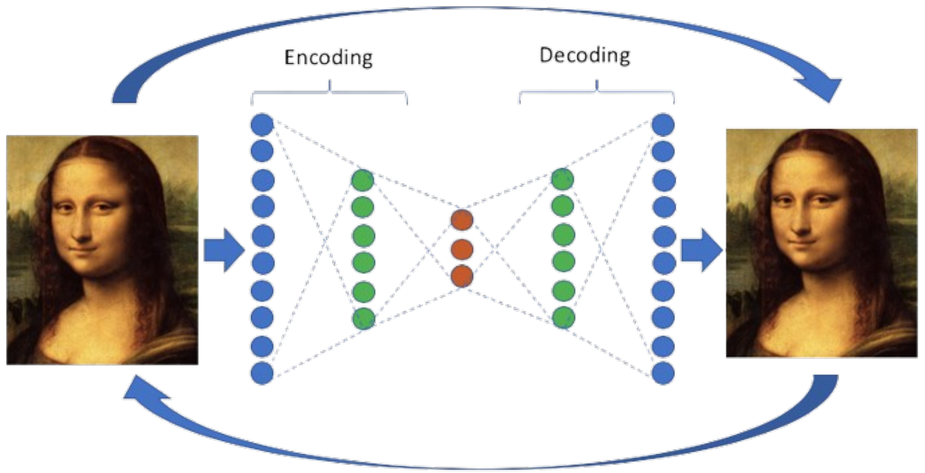


Autoencoders

Dimensionality Reduction using Deep Learning

- An autoencoder is a neural network trained to **reconstruct** its input.





Autoencoders

Dimensionality Reduction using Deep Learning

- An autoencoder is a neural network trained to **reconstruct** its input.
- **Encoder:** Compresses input to lower-dimensional latent representation
- **Latent space:** Compressed representation (bottleneck)
- **Decoder:** Reconstructs input from latent code
- **Training objective:** Minimize reconstruction error

$$L = ||x - \hat{x}||^2$$

Variational Autoencoders

Dimensionality Reduction using Deep Learning

- Standard autoencoders learn a **deterministic** mapping to latent space (this is true for all neural networks in general)
- The latent space may have “holes”, i.e., regions that don't correspond to valid data.
- You can't sample from it to generate new data.

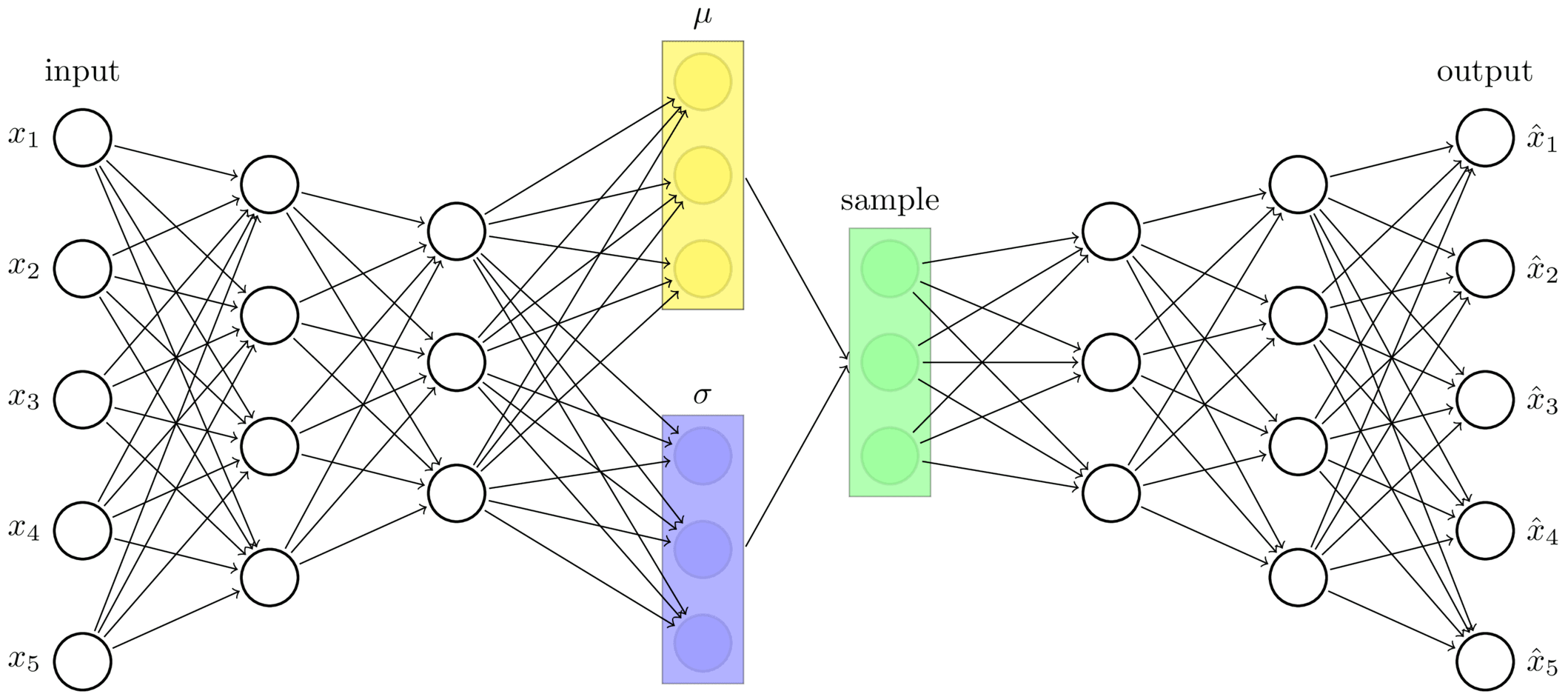
Variational Autoencoders

Dimensionality Reduction using Deep Learning

- Standard autoencoders learn a **deterministic** mapping to latent space (this is true for all neural networks in general)
- The latent space may have “holes”, i.e., regions that don't correspond to valid data.
- You can't sample from it to generate new data.
- **VAE Key Idea**
  - Instead of encoding to a point, encode to a **probability distribution.** The encoder outputs **parameters of a Gaussian distribution**, and we sample from it

Variational Autoencoders

Dimensionality Reduction using Deep Learning



Variational Autoencoders

Dimensionality Reduction using Deep Learning

- The Reparameterization Trick
  - **Problem**: Can't backpropagate through random sampling.
  - **Solution**: Reparameterize the sampling
    - $z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0,1)$
    - The randomness from sampling is now in ϵ and gradients can flow through μ and σ

Variational Autoencoders

Dimensionality Reduction using Deep Learning

- VAE Loss Function
  - Two Components
    - Reconstruction Loss - How well can the model reconstruct the input?

Variational Autoencoders

Dimensionality Reduction using Deep Learning

- VAE Loss Function
  - Two Components
    - Reconstruction Loss - How well can the model reconstruct the input?

$$L_{recon} = ||x - \hat{x}||^2$$

Variational Autoencoders

Dimensionality Reduction using Deep Learning

- VAE Loss Function
  - Two Components
    - Reconstruction Loss - How well can the model reconstruct the input?

$$L_{recon} = ||x - \hat{x}||^2$$

    - KL Divergence - How close is the learned distribution to the original?

Variational Autoencoders

Dimensionality Reduction using Deep Learning

- VAE Loss Function
  - Two Components
    - Reconstruction Loss - How well can the model reconstruct the input?

$$L_{recon} = ||x - \hat{x}||^2$$

    - KL Divergence - How close is the learned distribution to the original?

$$L_{KL} = KL(q(z|x) \parallel p(z))$$

  - Total Loss: $L = L_{recon} + L_{KL}$

Variational Autoencoders

Dimensionality Reduction using Deep Learning

- VAE Loss Function
  - Two Components
    - Reconstruction Loss - How well can the model reconstruct the input?

$$L_{recon} = ||x - \hat{x}||^2$$

    - KL Divergence - How close is the learned distribution to the original?

$$L_{KL} = KL(q(z|x) \parallel p(z))$$

  - Total Loss: $L = L_{recon} + L_{KL}$

The KL divergence **regularizes** the latent space:

- Encourages latent distributions to be close to $\mathcal{N}(0,1)$
- Creates a smooth, continuous latent space
- Enables **generation**: Sample$z \sim \mathcal{N}(0,1)$, decode to generate new data
- Without KL term, the encoder could **learn very narrow distributions** (essentially points), losing the generative property.

Variational Autoencoders

Applications

- **Image generation**: Generate new faces, digits, etc.
- **Data augmentation**: Generate variations of training data
- **Interpolation**: Morph between examples
- **Anomaly detection**: Unusual inputs have high reconstruction error
- **Drug discovery**: Generate novel molecular structures

Autoencoder

Summary

Aspect	K-Means	Hierarchical Clustering
Latent Space	Point (Deterministic)	Distribution (Probabilistic)
Generation	Poor (Gaps in space)	Good (Smooth Space)
Loss	Reconstruction Only	Recon + KL
Training	Simpler	More Complex
Sampling	Not Meaningful - Deterministic	Can sample and generate new data points

Today's Outline

- Variational Autoencoders
- Object Detection Models - YOLO

Object Detection

- Classification
  - Single Label
- Localization
  - Single Label + Bounding Box (eg: “Cat at (x, y, w, h)”
- Object Detection
  - Multiple Labels + Multiple Boxes (eg: “3 cats, 1 dog, positions”)
- Segmentation
  - Pixel level masks (eg: “these exact pixels are a cat”, no bounding boxes)

Object Detection

- Detection is harder than classification not just because of multiple objects, but because the number of outputs is variable
  - The model doesn't know **how many objects** are in the scene ahead of time.
  - The model doesn’t know **the shape** of the objects

Object Detection

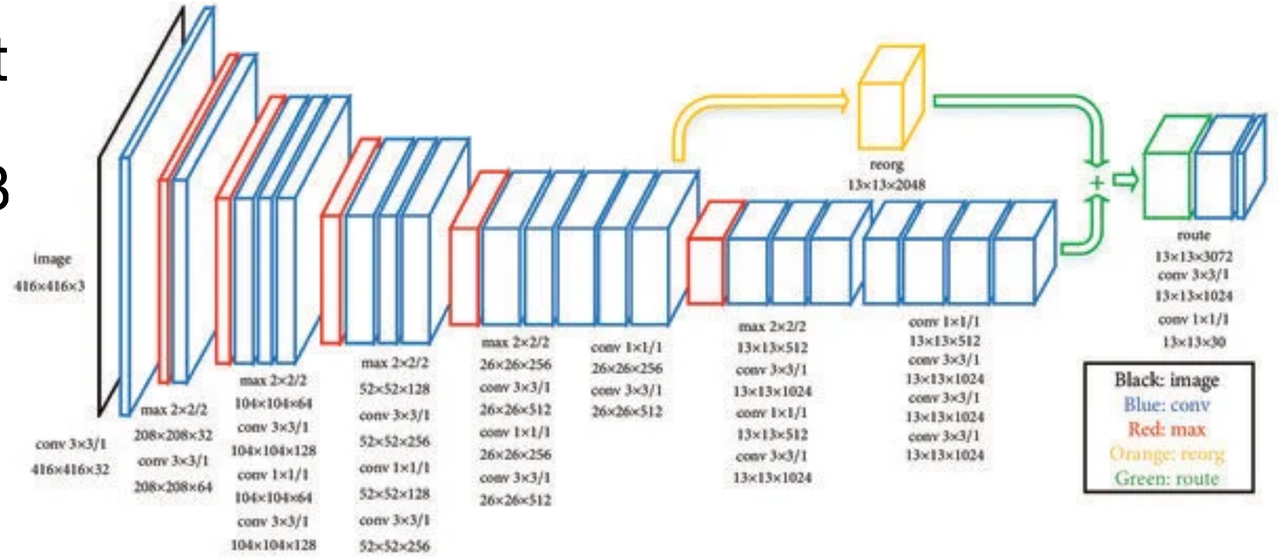
One-Stage Detectors: The YOLO Family (You Only Look Once)

- **Key Idea:**
  - Divide the image into an S×S grid (e.g., 13x13).
  - This division is **implicit in the way strides and convolutions work**
  - Each grid cell is responsible for detecting objects whose **center** falls in that cell.
- Each grid cell predicts:
  - B bounding boxes (or anchor boxes), each with (x, y, w, h, confidence)
  - C class probability scores (shared across the B boxes)

Object Detection

One-Stage Detectors: The YOLO Family (You Only Look Once)

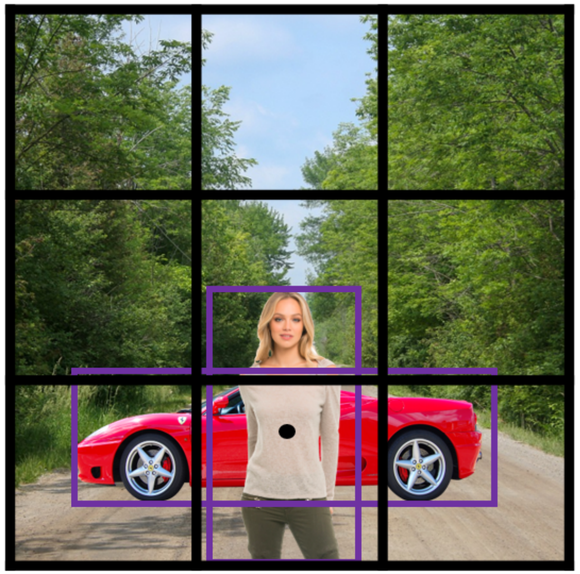
- Given an 416 x 416 image (as in **YOLOv2**):
  - Across 5 pooling operations, we get
  - 416 → 208 → 104 → 52 → 26 → 13



  - This means each of the 13×13 feature map has a **receptive field** centered on a 32×32 pixel region in the **original image**

Object Detection

One-Stage Detectors: The YOLO Family (You Only Look Once)



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

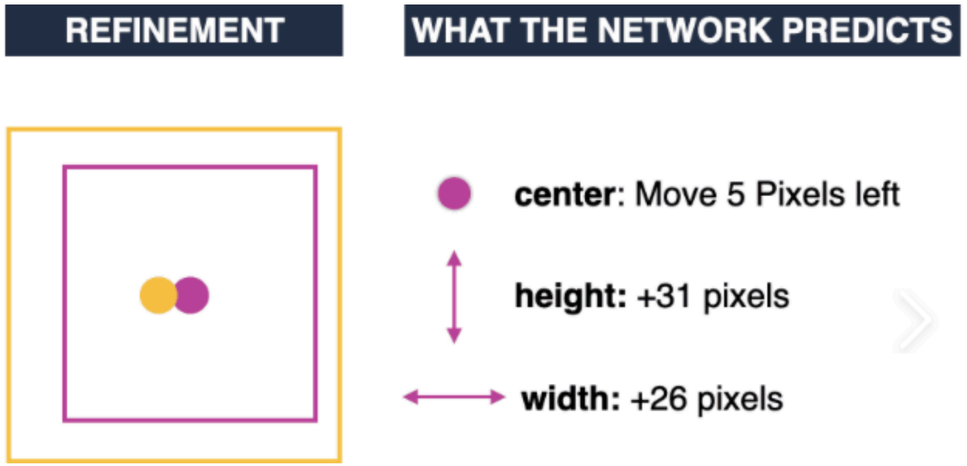
Total output tensor: S × S × (B×5 + C)

Object Detection

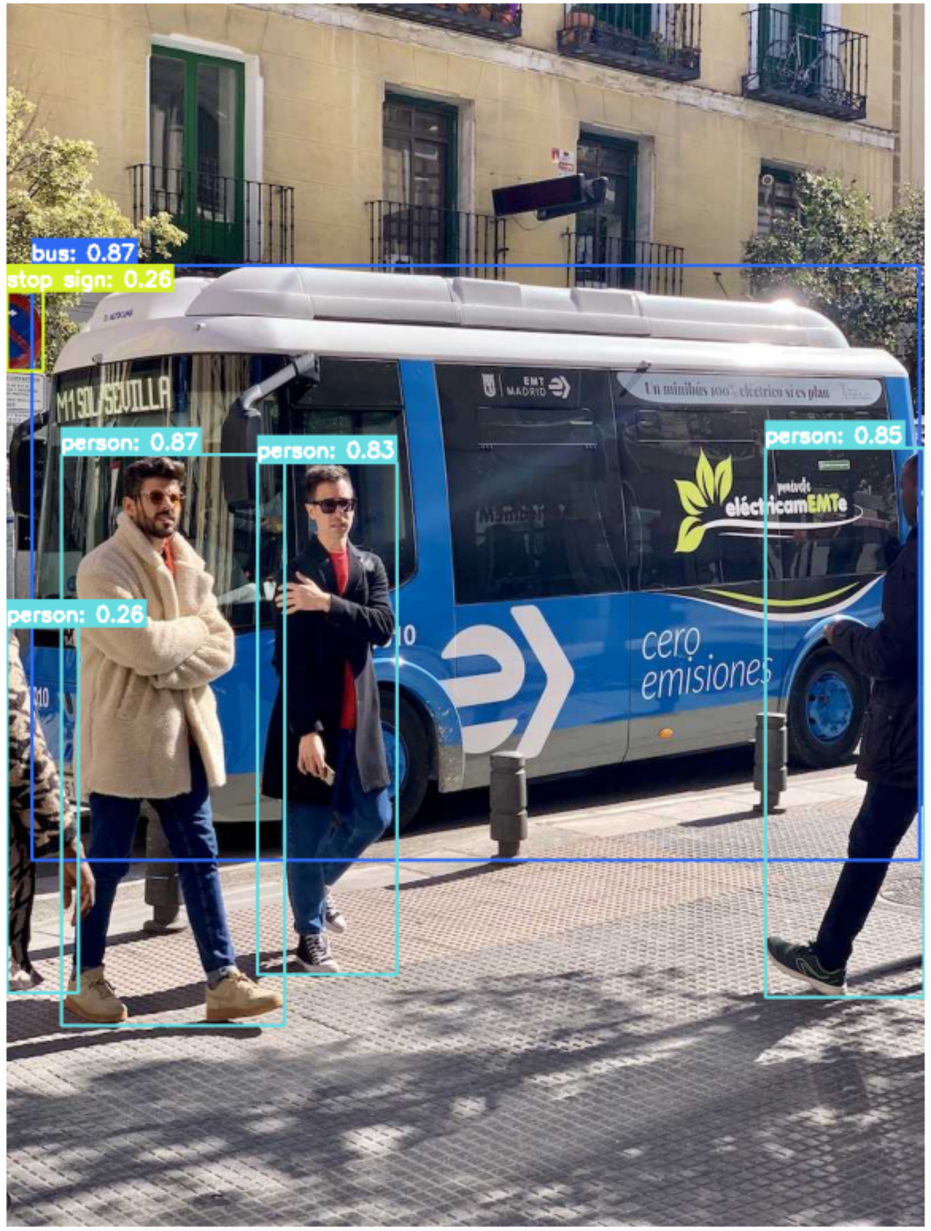
Bounding Box Parameterization

- A bounding box is represented as (x_c, y_c, w, h) - center coordinates, width, height - all typically normalized to [0, 1] relative to image dimensions.

- But, rather than predicting the exact sizes and centers, we predict **offsets**



Object Detection Results (Manual Visualization)



Object Detection

Bounding Box Parameterization

- t_x, t_y, t_w, t_h - What the network predicts (raw, unconstrained numbers)
- p_w, p_h - The anchor box's prior width and height (the fixed sizes you have)
- b_x, b_y, b_w, b_h - The final predicted box (what you actually use)

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w \cdot e^{t_w}$$
$$b_h = p_h \cdot e^{t_h}$$

Object Detection

Overall Flow of YOLOv2

- 20 Classes of Objects
  - Final layer output looks like - $13 \times 13 \times 5 \times 25$

Number of chunks of original image

Object Detection

Overall Flow of YOLOv2

- 20 Classes of Objects
  - Final layer output looks like - $13 \times 13 \times 5 \times 25$

5 pre-defined shapes - “bounding boxes” or “anchor boxes”
Each of the 13x13 cells could have any of the 5 anchors

Object Detection

Overall Flow of YOLOv2

- 20 Classes of Objects
  - Final layer output looks like - $13 \times 13 \times 5 \times 25$

    A 25 dimensional vector for each cell for each bounding box

Object Detection

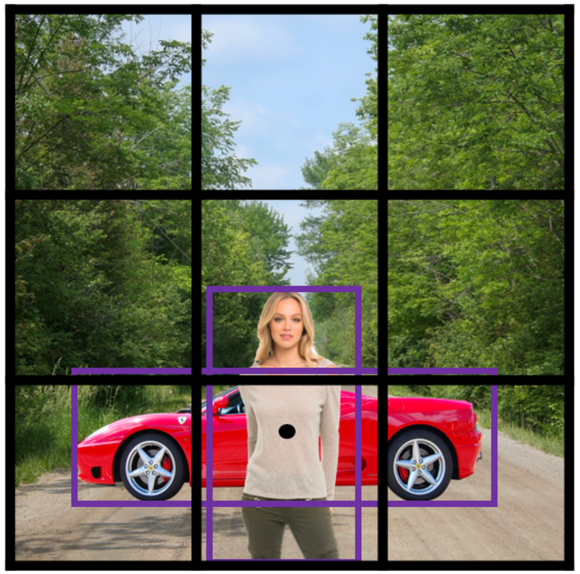
Overall Flow of YOLOv2

- 20 Classes of Objects
  - Final layer output looks like - $13 \times 13 \times 5 \times 25$

$$\begin{bmatrix} t_x & t_y & t_w & t_h & conf & c_1 & c_2 & c_3 & \dots & c_{20} \end{bmatrix}$$

Object Detection

One-Stage Detectors: The YOLO Family (You Only Look Once)



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Total number of bounding box predictions

13 x 13 x 5 = 845 bounding boxes

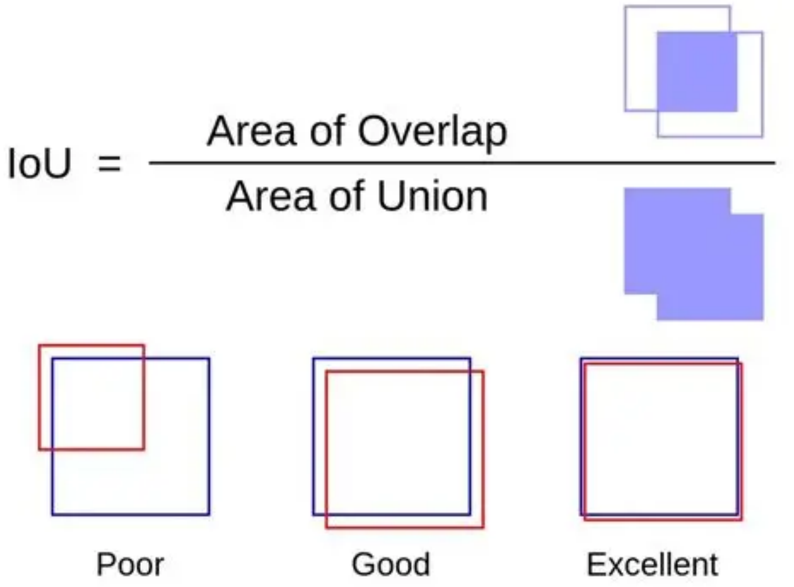
Object Detection

Intersection over Union

- IoU measures how well a predicted box overlaps a ground truth box:

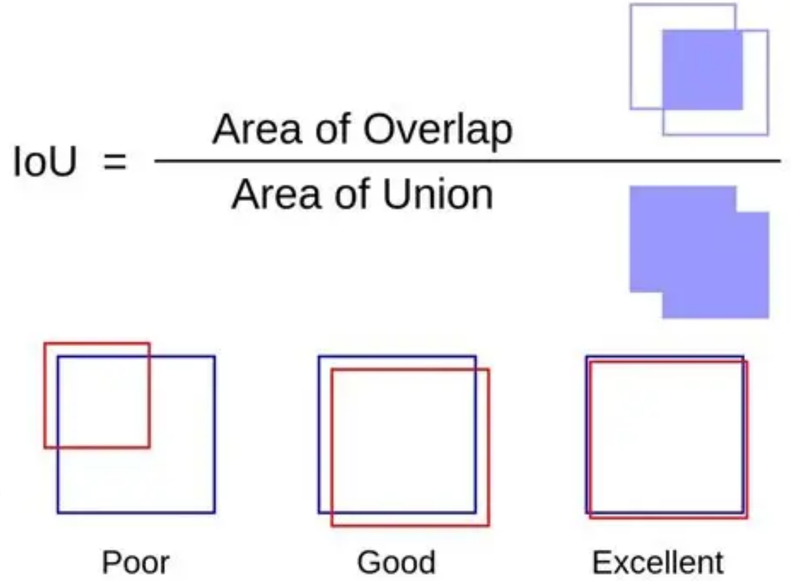
$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

- IoU = 1.0 → perfect overlap
- IoU = 0 → no overlap at all
- Common threshold: IoU ≥ 0.5 is a “correct” detection



Object Detection

Non-Maximum Suppression (NMS)



- Detectors produce many overlapping boxes for the same object. NMS resolves this:
  - Sort all predicted boxes by confidence score (descending)
  - Take the highest-confidence boxes - keep it
  - Discard any remaining box with IoU > threshold vs. the kept box
  - Repeat for the next remaining box
  - **Intuition**: You're greedily picking the most confident prediction and suppressing near-duplicates. **This is a post-processing step, not learned.**

YOLO v2

1. K-means on training data to generate fixed shapes of anchor boxes (5 anchors)
2. 5 anchor shapes - fixed, computed before training
3. Tiled across all 13×13 cells = 845 candidate boxes
4. Network predicts 25 values per candidate
5. Apply sigmoid/exp equations → 845 refined boxes with class scores
6. Filter by confidence threshold → maybe 20 boxes survive
7. NMS → maybe 5-10 final detections

YOLO v3

Multi-scale predictions: Predictions at 3 different scales. This directly addresses v1/v2's weakness on small objects:

Large feature map (52×52) → small objects

Medium feature map (26×26) → medium objects

Small feature map (13×13) → large objects

Anchor assignment: 9 anchors total, 3 per scale. Anchors still determined by k-means on training data.

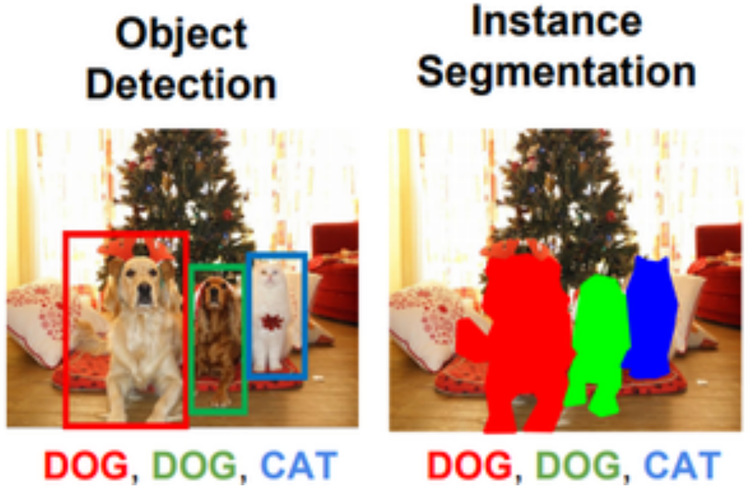
Class prediction: V3 replaces **softmax** with independent **sigmoid** classifiers for each class. This allows **multi-label predictions** - an object can be both "person" and "woman" simultaneously. Better for overlapping class hierarchies.

Segmentation Models

Semantic segmentation: Label every pixel with a class. No distinction between instances - all cars are “car."

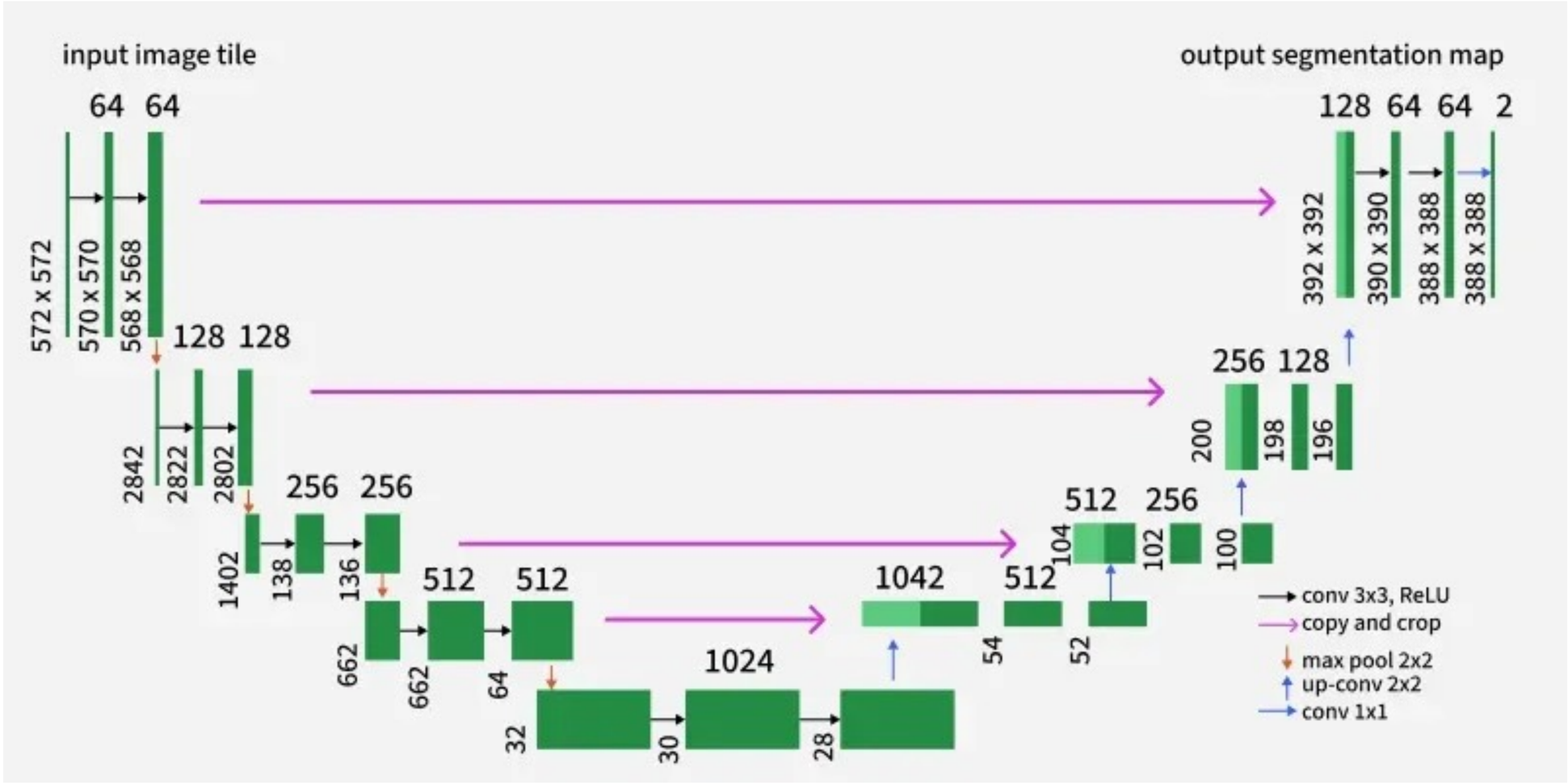
Instance segmentation: Detect and delineate each individual object. Two cars get two separate masks.

Panoptic segmentation: Unified - semantic for "stuff" (sky, road, grass) + instance for “things” (cars, people).



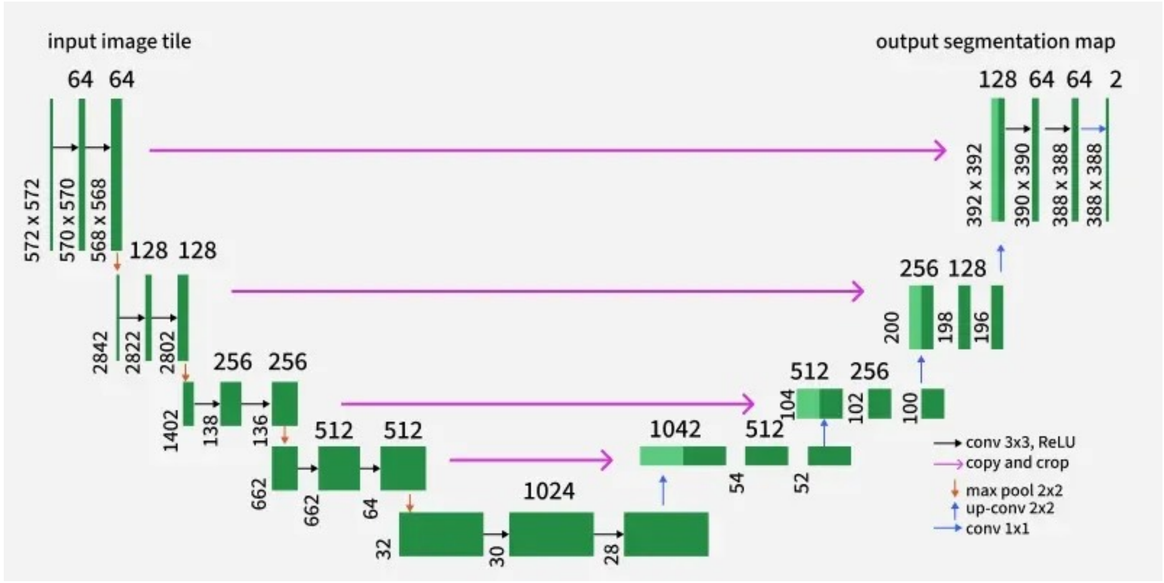
Segmentation Models

U-Net



Segmentation Models

U-Net



- The architecture is symmetric and has **three** key parts:

1. Contracting Path (Encoder):
    - Uses small filters (3×3 pixels) to scan the image and find features.
    - Uses max pooling (2×2 filters) to shrink the image size while keeping important information. This helps the network focus on bigger features.
2. Bottleneck
3. Expansive Path (Decoder):
    - Uses upsampling i.e increasing image size to get back the original image size.
    - Combines information from the encoder using **“skip connections”**. These connections help the decoder get spatial details that might have been lost when shrinking the image.