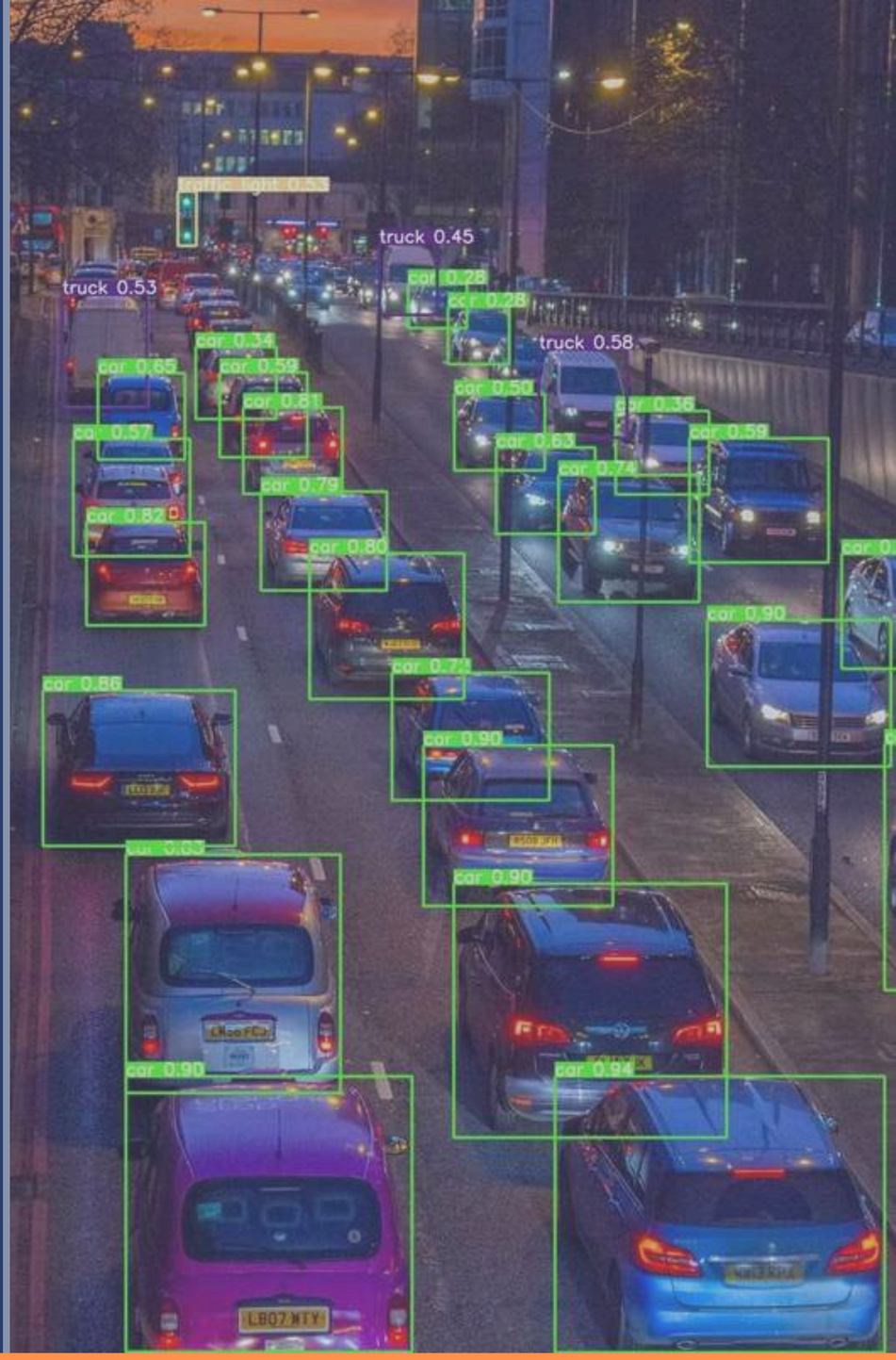
Rumah Coding

Technical Webinar Series

# Analyzing Object Detection Architectures

A Strategic Guide to Choosing Between YOLO, SSD, and Faster R-CNN.




**Lhuqita Fazry**

Founder Rumah Coding

Lhuqita Fazry

**Lhuqita Fazry, S.Si., M.Kom.**

(Founder Rumah Coding)

Pendidikan:

- S1: Math, Universitas Indonesia
- S2: Ilmu Komputer, Universitas Indonesia

Pengalaman Riset:

- Intelligent Robots and Systems (IRoS) Lab, Faculty of Computer Science, Universitas Indonesia
- Human-AI Interaction Laboratory, Japan Advance Institute of Technology (JAIST)

Lhuqita Fazry

Rumah Coding

# Introducing: **Room**

Rumah Coding

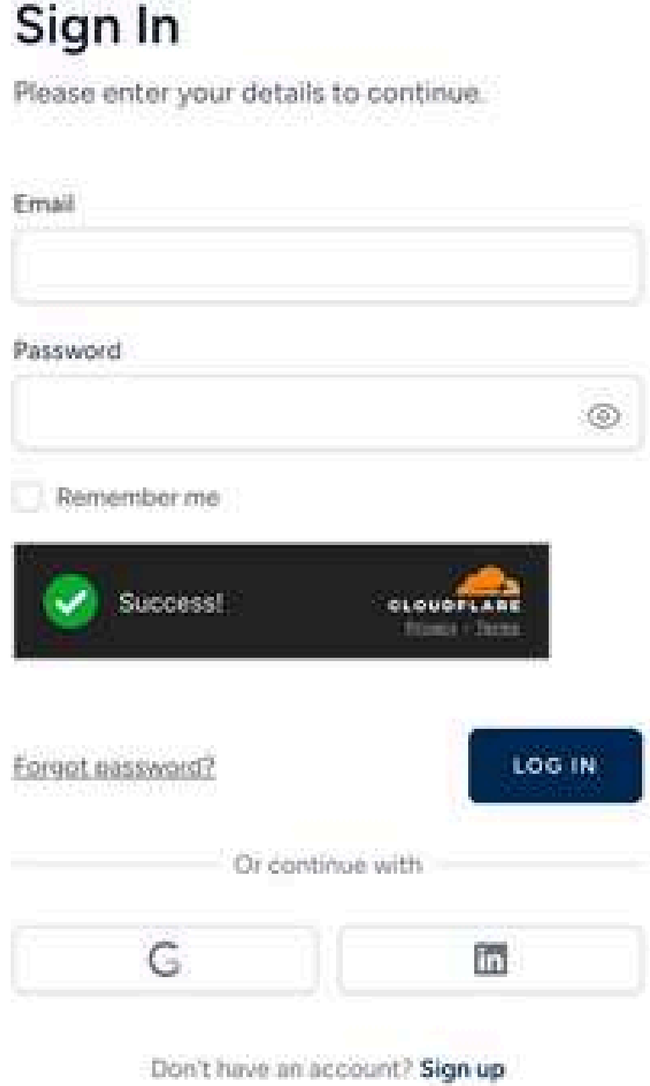
room.rumahcoding.id

## Welcome Back to Rumah Coding

"Programming isn't about what you know; it's about what you can figure out."

- CHRIS PINE





# Contents

- The Foundation of Object Detection
- Deep Dive into Region-Based Detectors (Two-Stage)
- Deep Dive into Single-Shot Detectors (One-Stage)
- Comparative Analysis
- Implementation Walkthrough

# The Foundation of Object Detection

# Defining the Problem Space

# Classification, Localization and Detection

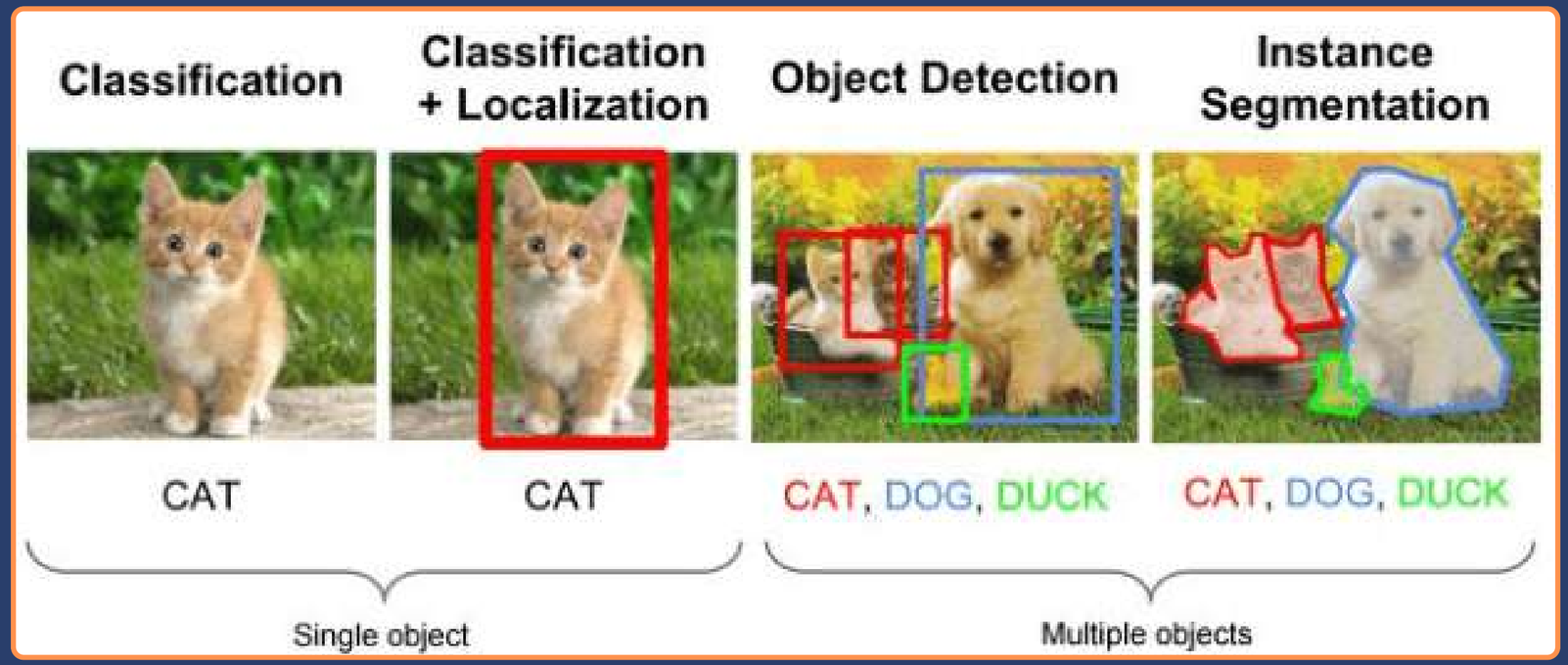
Rumoh Coding

## Image Classification

- **Goal**: Assign a single label to an image (e.g., "Car").
- **Question**: "What is the dominant object?"
- **Output**: Class Probability.

## Object Localization

- **Goal**: Locate the single main object with a bounding box.
- **Question**: "Where is the object?"
- **Output**: Class + Coordinates ($x$, $y$, $w$, $h$).

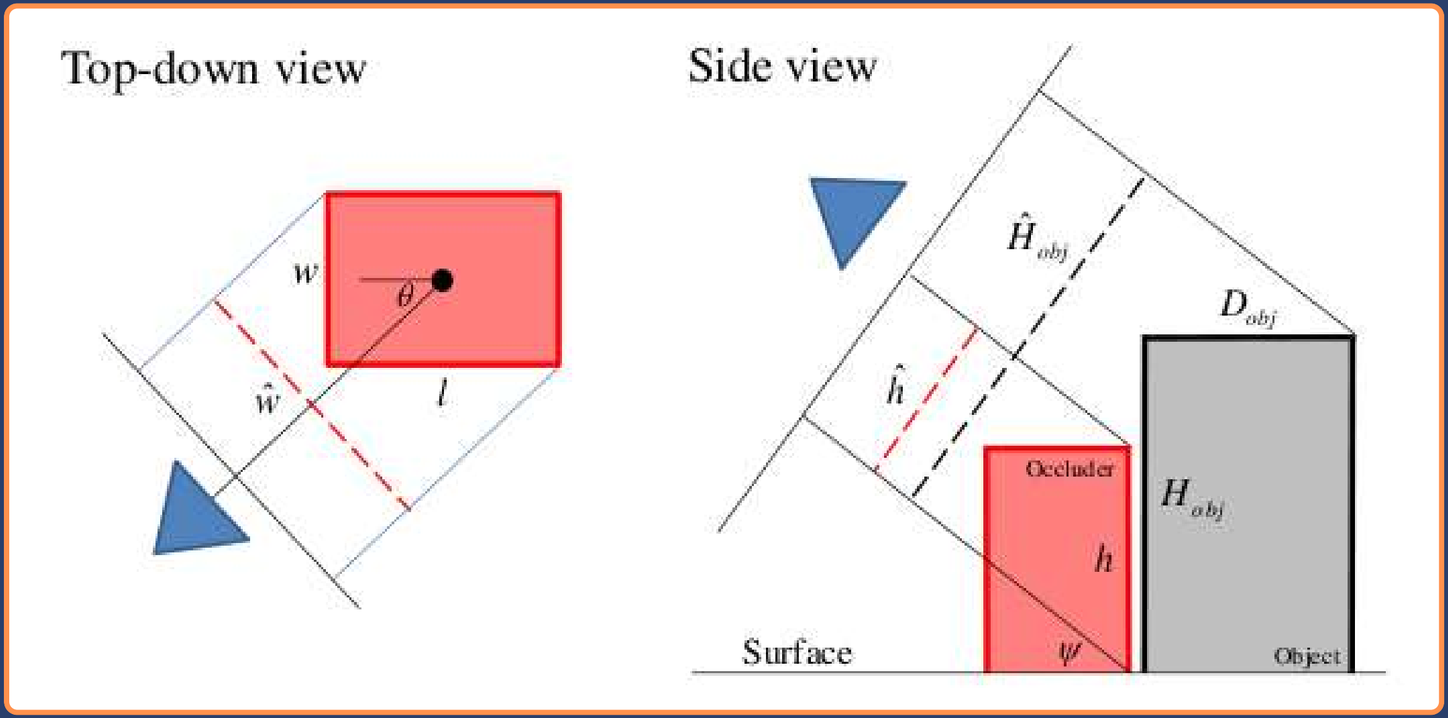



## Object Detection

- **Goal**: Locate and classify variable numbers of objects in a single image.
- **Complexity**: Simultaneous classification (labeling) and regression (box positioning) for multiple instances.

https://www.researchgate.net/figure/Difference-between-classification-localization-detection-and-segmentation-Jaiswal-et_fig1_355467756

# Core Challenges in Detection

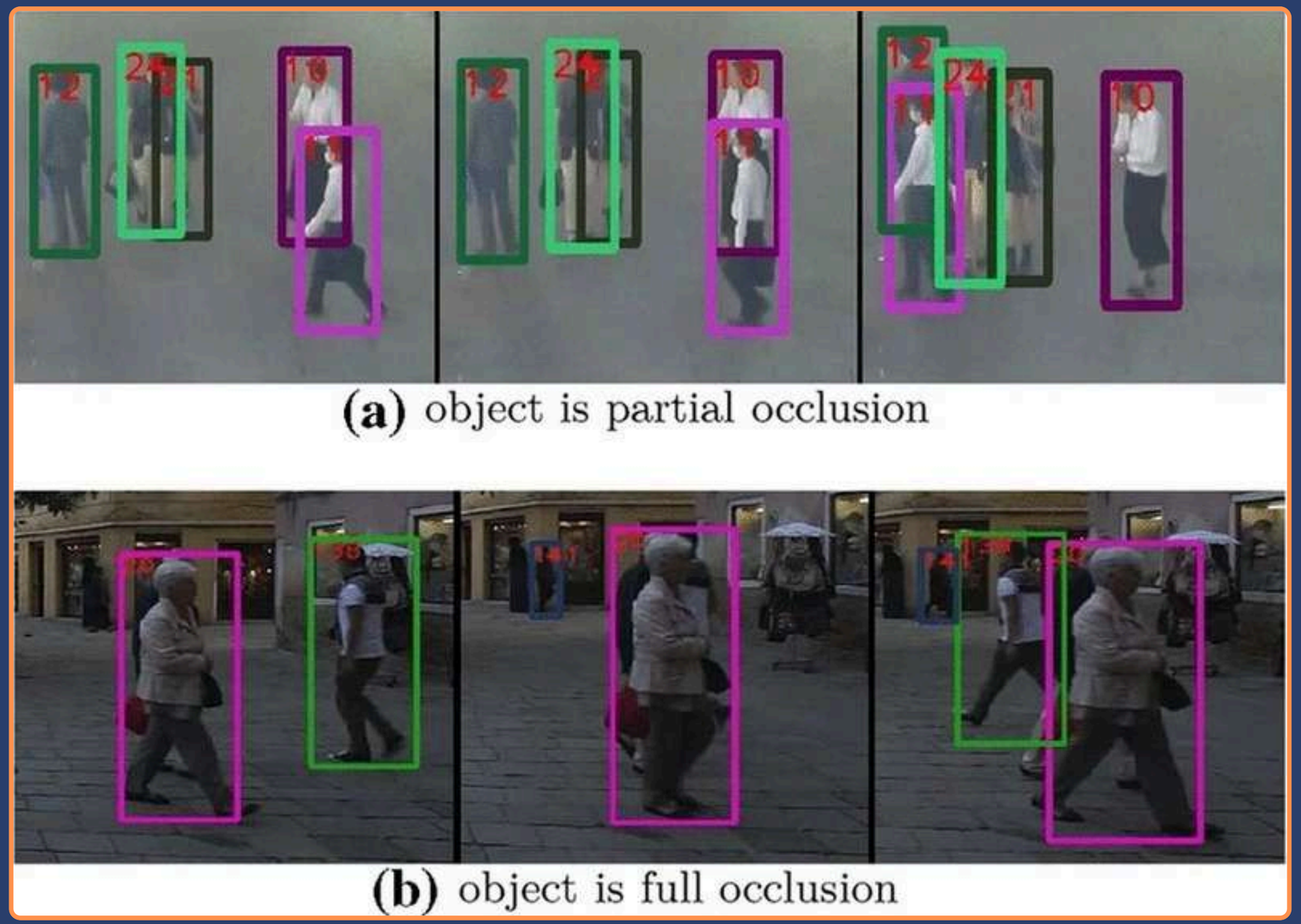



- Unlike controlled datasets, real-world images contain noise and unpredictable layouts.
- The Three Major Hurdles
  - **Occlusion**: Objects are partially hidden.
  - **Scale Variation**: Objects appear in vastly different sizes.
  - **Multiple Instances**: Crowded objects of the same class.

https://www.cs.cmu.edu/~hebert/occarbview.html

# Challenge I - Occlusion

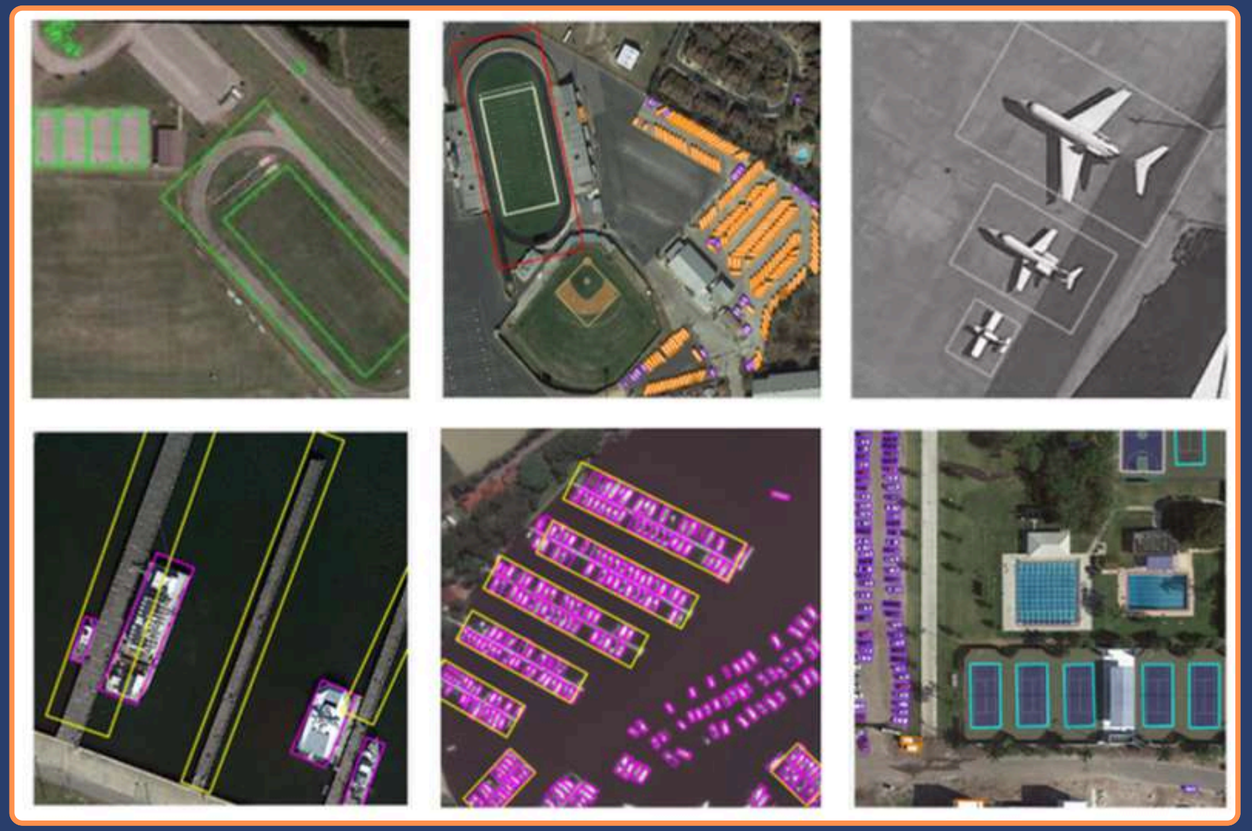
- The target object is partially obstructed by other objects or background elements.
- Critical features may be missing (e.g., a person standing behind a desk, only the upper body is visible).
- The detector must infer the complete object boundary based on incomplete visual data, avoiding "hallucinations" (false positives).




https://www.researchgate.net/figure/Some-examples-of-tracking-when-object-occluded-using-proposed-method_fig5_341770094

# Challenge II - Scale Variation

- The pixel size of an object varies drastically based on its distance from the camera.
  - **Large Objects**: Rich in texture and detail.
  - **Small Objects**: Few pixels, lacking detail, easily lost in down-sampling layers.
- A robust architecture must be "scale-invariant" capable of detecting both a 500px car and a 20px car effectively.



https://www.researchgate.net/figure/Large-scale-variation-in-objects-within-remote-sensing-images_fig1_381084722

# Challenge III - Multiple Instances

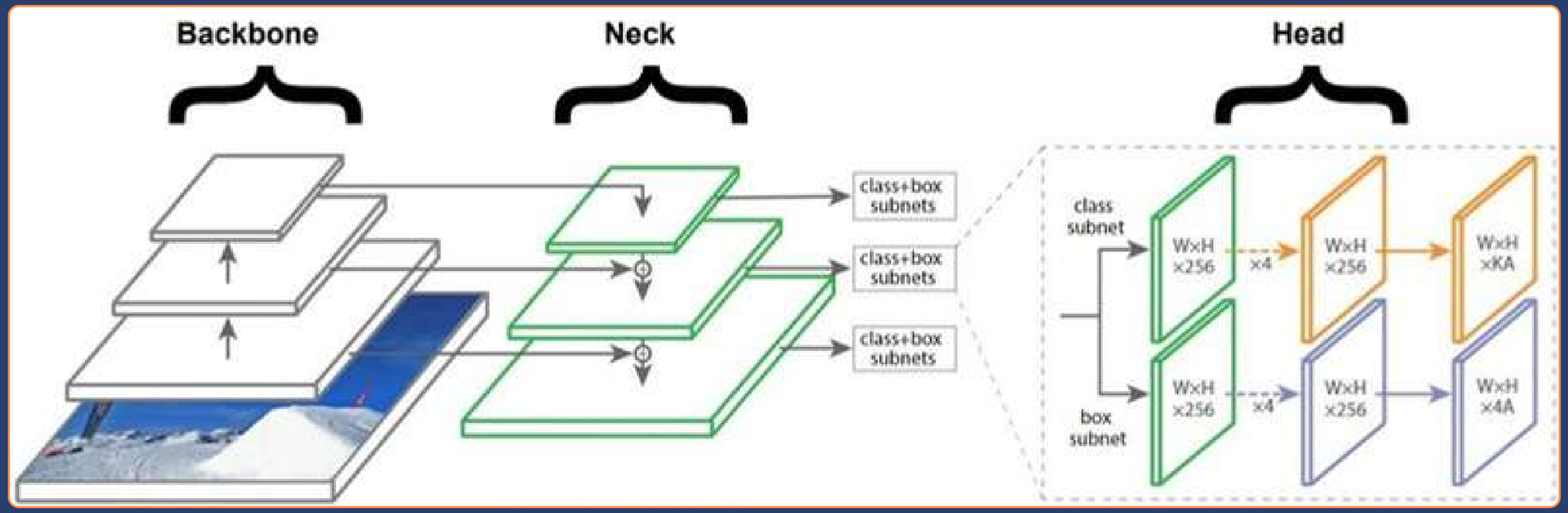
- Objects of the same category (e.g., humans) have different poses, clothing, and shapes.
- In dense scenarios, bounding boxes overlap significantly.
- The model must distinguish individual instances (e.g., Person A vs. Person B) rather than merging them into one large detection.


https://github.com/ultralytics/yolov5/issues/1779

# The Universal Anatomy of Modern Object Detectors

# The Modern Detector Architecture

- Most modern detectors (YOLO, Faster R-CNN, RetinaNet) share a common three-stage architectural pattern.




Extracts visual features

Refines and combines features across scales

Predicts bounding boxes and classes

- Understanding this structure allows us to mix and match components for specific performance goals (e.g., swapping a heavy Backbone for a lighter one for mobile deployment).

https://velog.io/@hewas1230/ObjectDetection-Architecture

# Backbone as The Feature Extractor

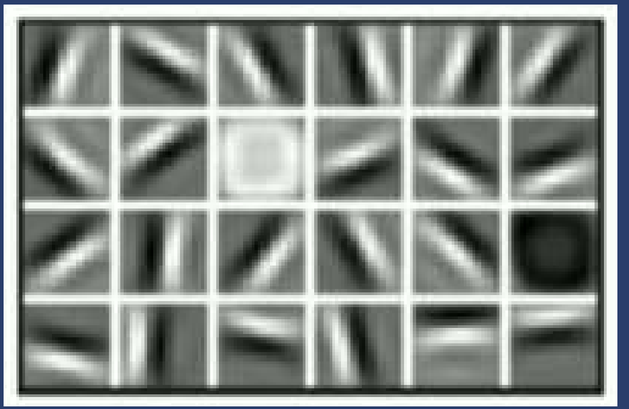
Rumah Coding

- Backbone takes the image as input and extracts hierarchical feature maps.

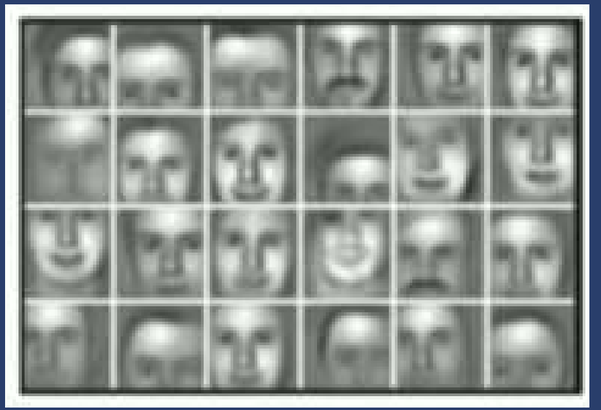
| Low-Level Features | High-Level Features |
| --- | --- |
| Edges, corners, textures | Semantic patterns (e.g., wheels, eyes, text) |

- Typically a Convolutional Neural Network (CNN) pre-trained on ImageNet, with the final classification layers removed.
- Popular architectures:
  - **ResNet / VGG**: Standard backbones for high accuracy.
  - **CSPDarknet**: Utilized by YOLO for optimized gradient flow.
  - **MobileNet / EfficientNet**: Lightweight backbones optimized for edge devices and CPU inference.

Input Image



Low level features

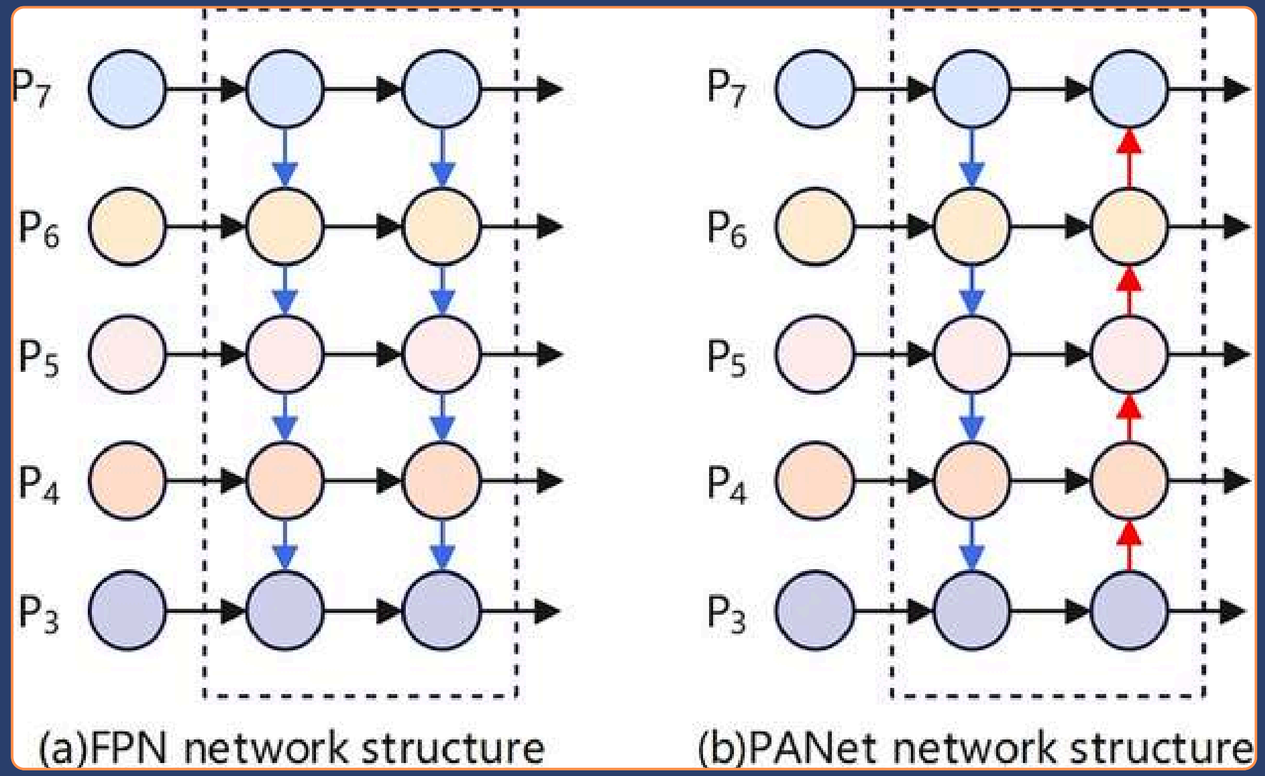


High level features

https://aiml.com/what-is-deep-learning/

# Neck as Feature Aggregation & Multi-Scale Handling

- **Problem:** Backbones reduce image resolution
- Neck connects the Backbone to the Head. It mixes features from different layers to create representation that is both semantically strong and spatially precise.
- Common approaches:
  - **FPN (Feature Pyramid Network)**: Builds a top-down pathway to pass semantic information to high-resolution layers.
  - **PANet (Path Aggregation Network)**: Adds a bottom-up pathway to further enhance localization signals (common in YOLO).




https://journals.plos.org/plosone/article/figures?id=10.1371/journal.pone.0312112

# Head as The Prediction Layer

Rumoh Coding

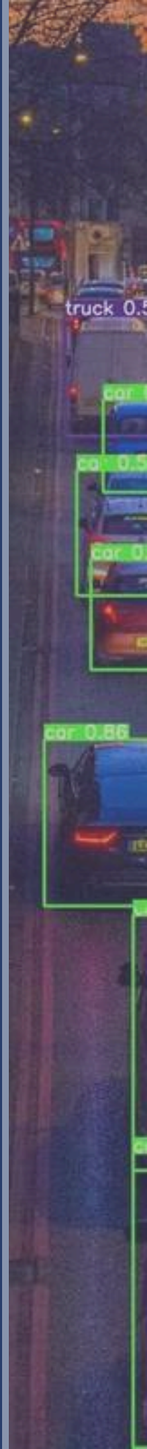
- Consumes the features from the Neck and outputs the final detection results.
  - **Regression Branch**: Predicts coordinates (x, y, w, h).
  - **Classification Branch**: Predicts the class probability.
- Two Main Paradigms

| Sparse Prediction |
| --- |
| Processes only specific candidate regions |

| Dense Prediction |
| --- |
| Processes the entire image grid in a single pass |

Two-stage

One-stage




https://journals.plos.org/plosone/article/figures?id=10.1371/journal.pone.0312112

# Deep Dive into Region-Based Detectors

Rumah Coding

# "Look Twice" Philosophy

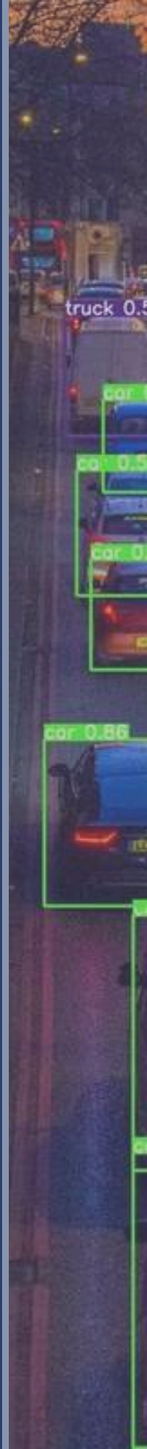
- **Stage 1 (Proposal)**: "Where might the objects be?" (Generates candidate regions)
- **Stage 2 (Classification)**: "What exactly is in this region?" (Refines the box and assigns a label).
- **Advantage**: Extremely high localization accuracy and better performance on small objects.
- **Disadvantage**: Higher computational cost and latency compared to one-stage detectors.

**Stage 1**

Where?

**Stage 2**

What?

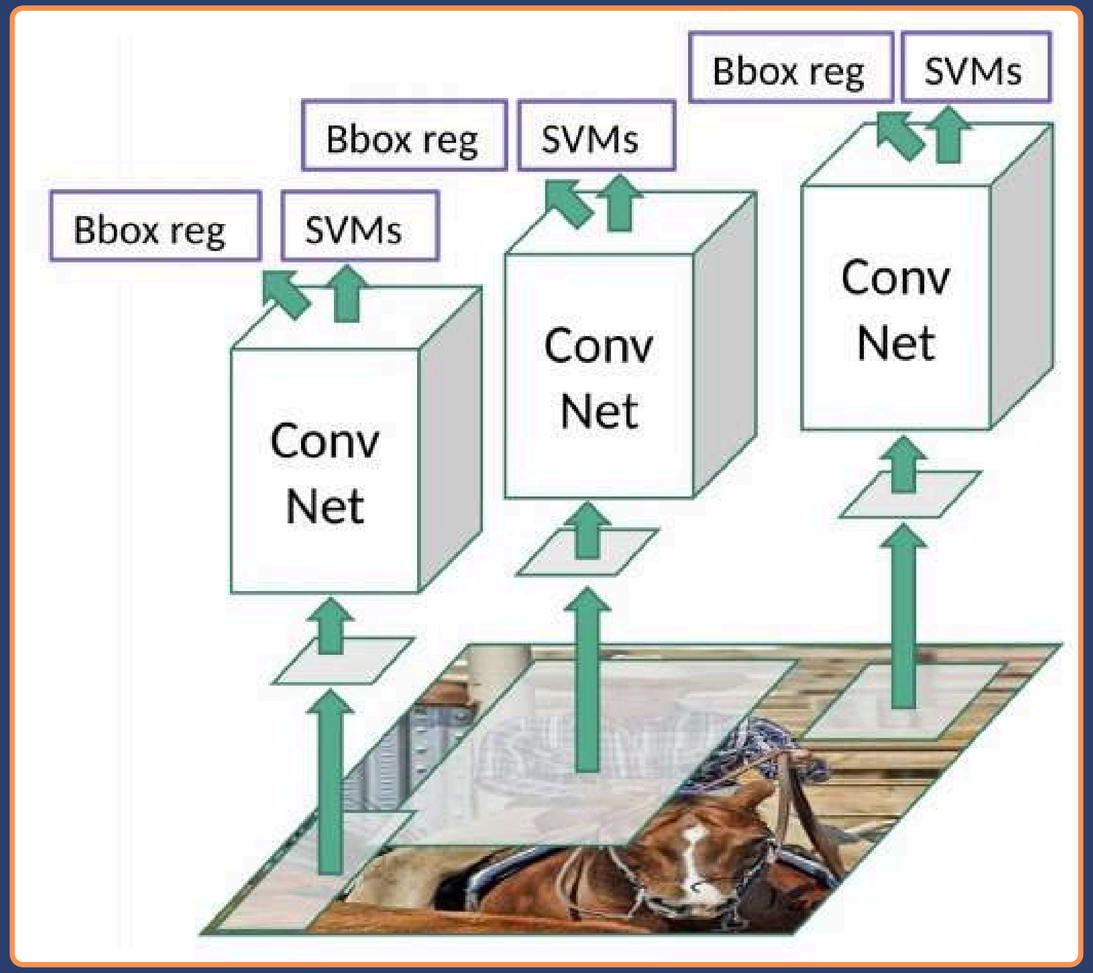


# The Evolution:
# From R-CNN to Faster R-CNN

# R-CNN (2014)

R-CNN (Regions with CNN Features)

- **Selective Search**: Uses a CPU-based algorithm to extract ~2,000 region proposals.
- **Warping**: Resizes every region to a fixed input size.
- **ConvNet**: Feeds each of the 2,000 regions independently into a CNN (e.g., AlexNet).
- **SVM & Regressor**: Classifies features and refines coordinates.

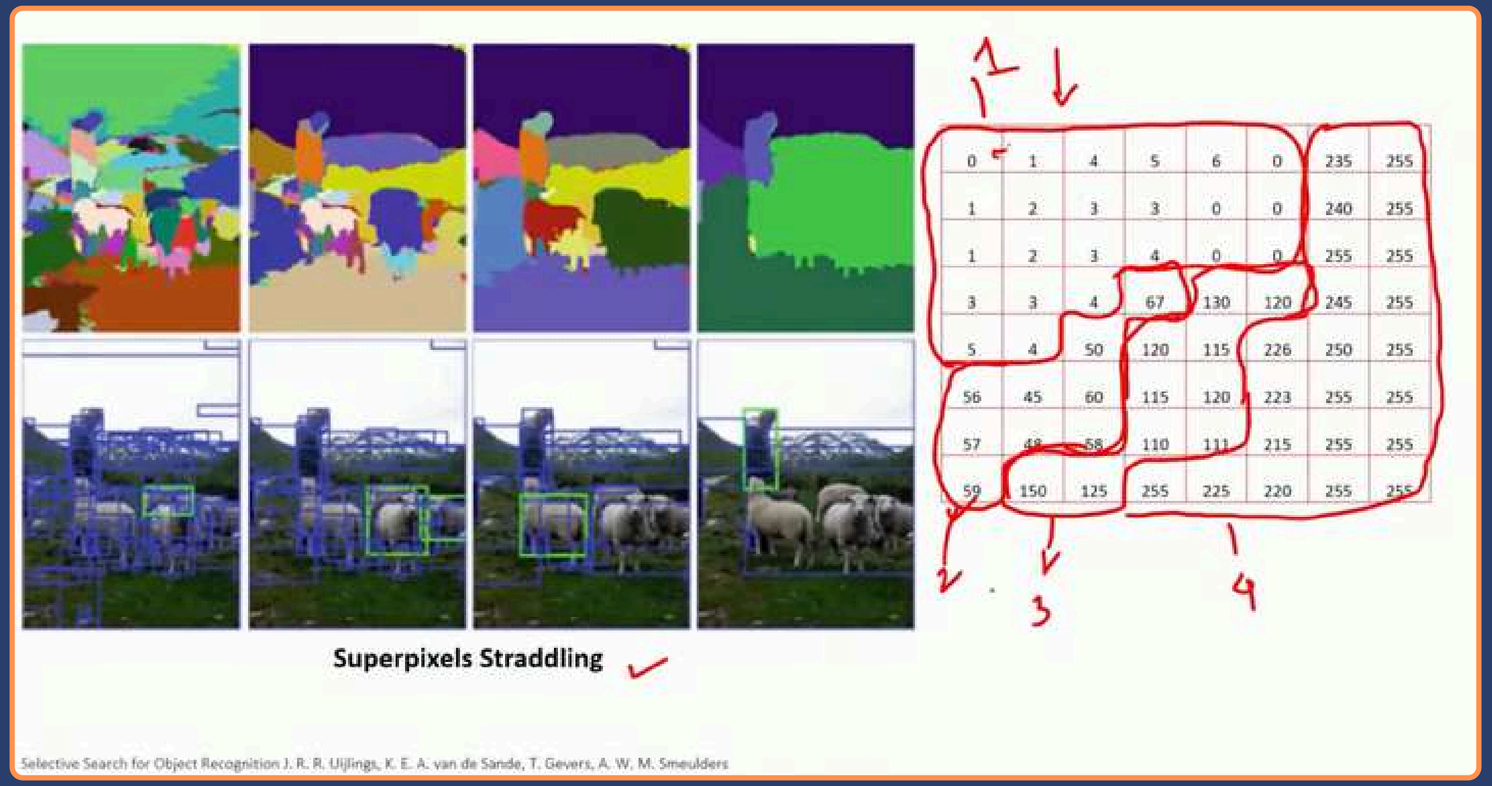



Major Drawbacks:

- **Slow Inference**: ~47 seconds per image (on GPU).
- **Redundant Computation**: The CNN processes overlapping regions multiple times.
- **Complex Training**: Multi-stage pipeline (ConvNet, SVM, and Regressor trained separately).

https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e/

# How Selective Search Works in R-CNN?



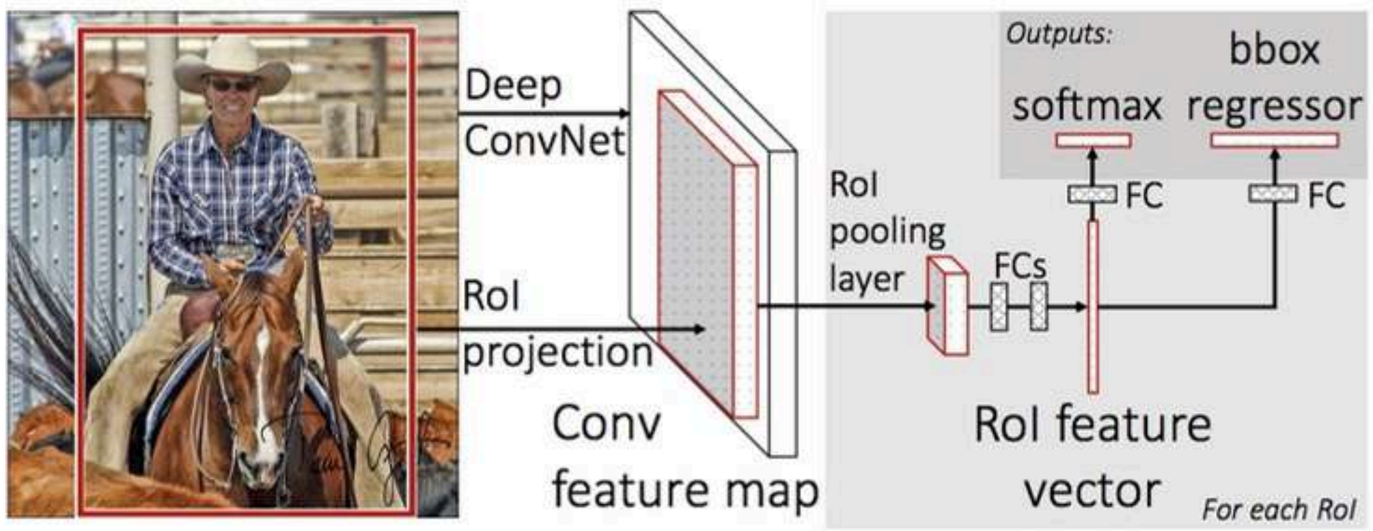

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 4 | 5 | 6 | 0 | 235 | 255 |
| 1 | 2 | 3 | 3 | 0 | 0 | 240 | 255 |
| 1 | 2 | 3 | 4 | 0 | 0 | 255 | 255 |
| 3 | 3 | 4 | 67 | 130 | 120 | 245 | 255 |
| 5 | 4 | 50 | 120 | 115 | 226 | 250 | 255 |
| 56 | 45 | 60 | 115 | 120 | 223 | 255 | 255 |
| 57 | 48 | 58 | 110 | 111 | 215 | 255 | 255 |
| 59 | 150 | 125 | 255 | 225 | 220 | 255 | 255 |

- **Initial Segmentation**: Over-segments the image into thousands of small superpixels based on pixel intensity.
- **Bottom-Up Merging**: Iteratively groups adjacent segments together to form larger, meaningful regions.

- **Merge Criteria:**
  - **Color**: Merges regions with similar color histograms.
  - **Texture**: Merges regions with similar visual patterns/gradients.
  - **Size**: Prioritizes merging small regions first to prevent them from being swallowed by large ones.
  - **Fill**: Merges regions that fit well into each other to ensure compactness.

# Fast R-CNN (2015)

- Mechanism
  - Processes the entire image through the CNN once to get a global feature map.
  - Projects region proposals (from Selective Search) onto this feature map.
  - **ROI Pooling**: Extracts fixed-length feature vectors from the feature map for each proposal.




Improvements

- **Speed**: Significantly faster training and inference than R-CNN.
- **Single Stage Training**: Uses a multi-task loss (classification + regression) in one network.

Still relies on Selective Search (external CPU algorithm) for generating proposals, which limits real-time performance.

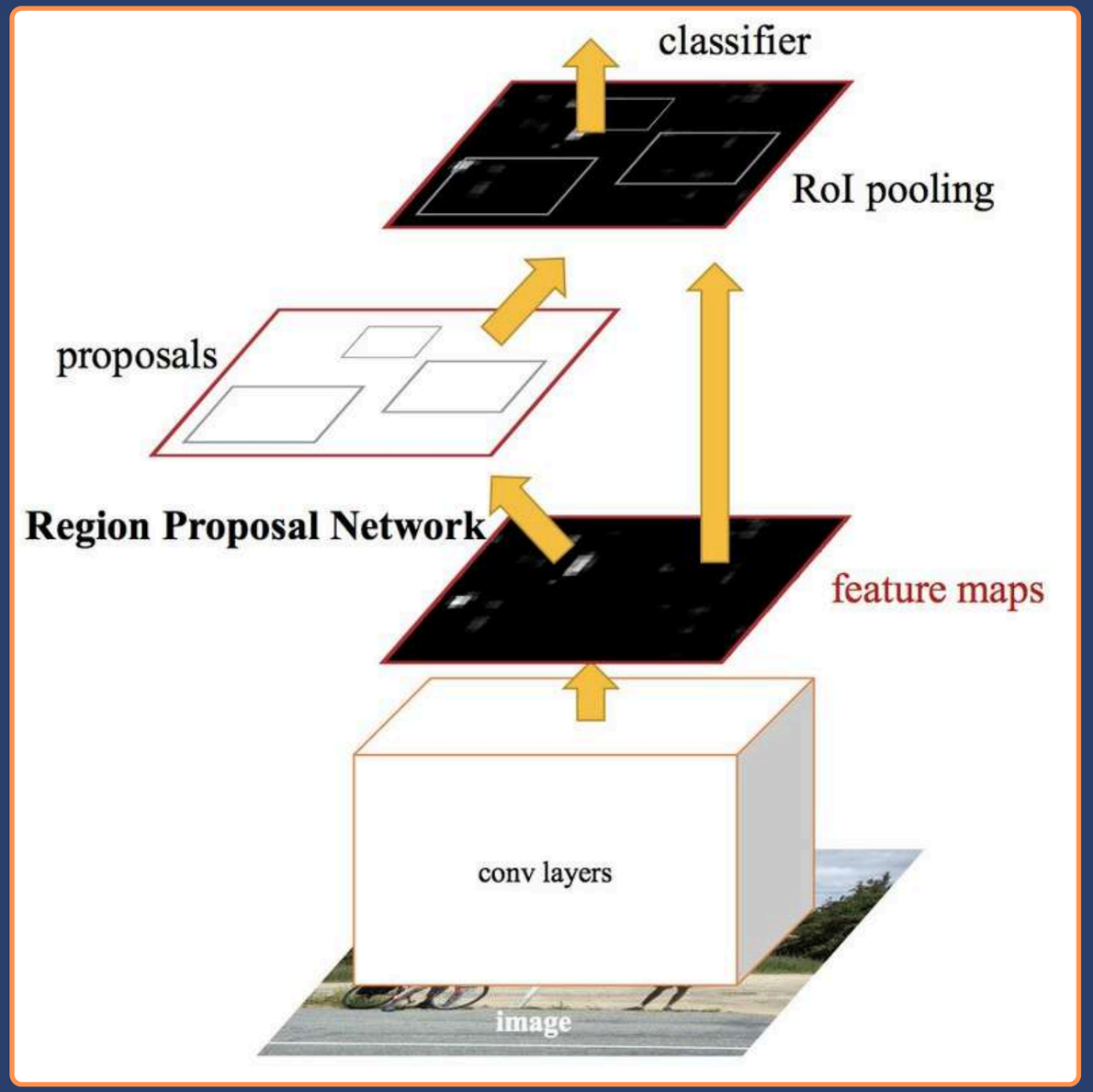
Key Innovation:
ROI (Region of Interest) Pooling

https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e/

# Faster R-CNN (2015)

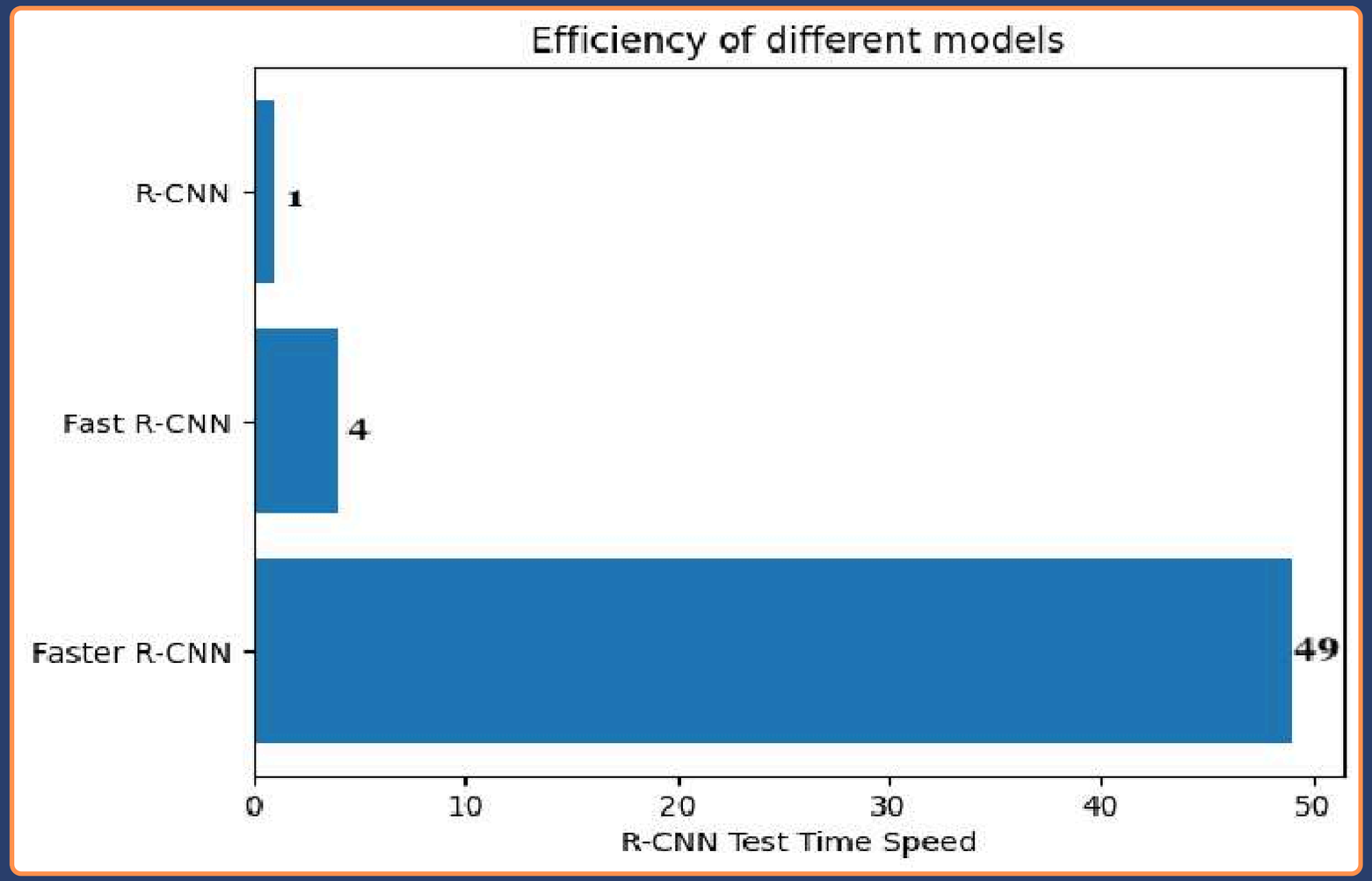
Truly End-to-End Object Detection

- Replaces "Selective Search" with a Region Proposal Network (RPN) running on the GPU.
- The RPN shares the same backbone features as the classifier (almost zero cost for proposals).
- Mechanism
  - **Backbone**: Extracts Feature Maps.
  - **RPN**: Slides over features to predict "objectness" and propose regions.
  - **ROI Pooling**: Aligns features.
  - **Head**: Final Classification and Bounding Box Regression.




https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e/

# Speed Comparison




https://www.researchgate.net/figure/Comparison-of-R-CNN-Fast-R-CNN-and-Faster-R-CNN_fig2_365019554

# The Authors of R-CNN Family

Rumoh Coding

## Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

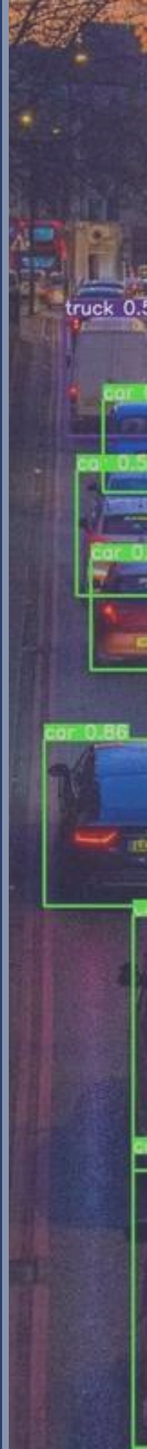
Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

The author of legendary ResNet

The author of R-CNN and Fast R-CNN

**Abstract**—State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet [1] and Fast R-CNN [2] have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a *Region Proposal Network* (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with "attention" mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model [3], our detection system has a frame rate of 5fps (*including all steps*) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.

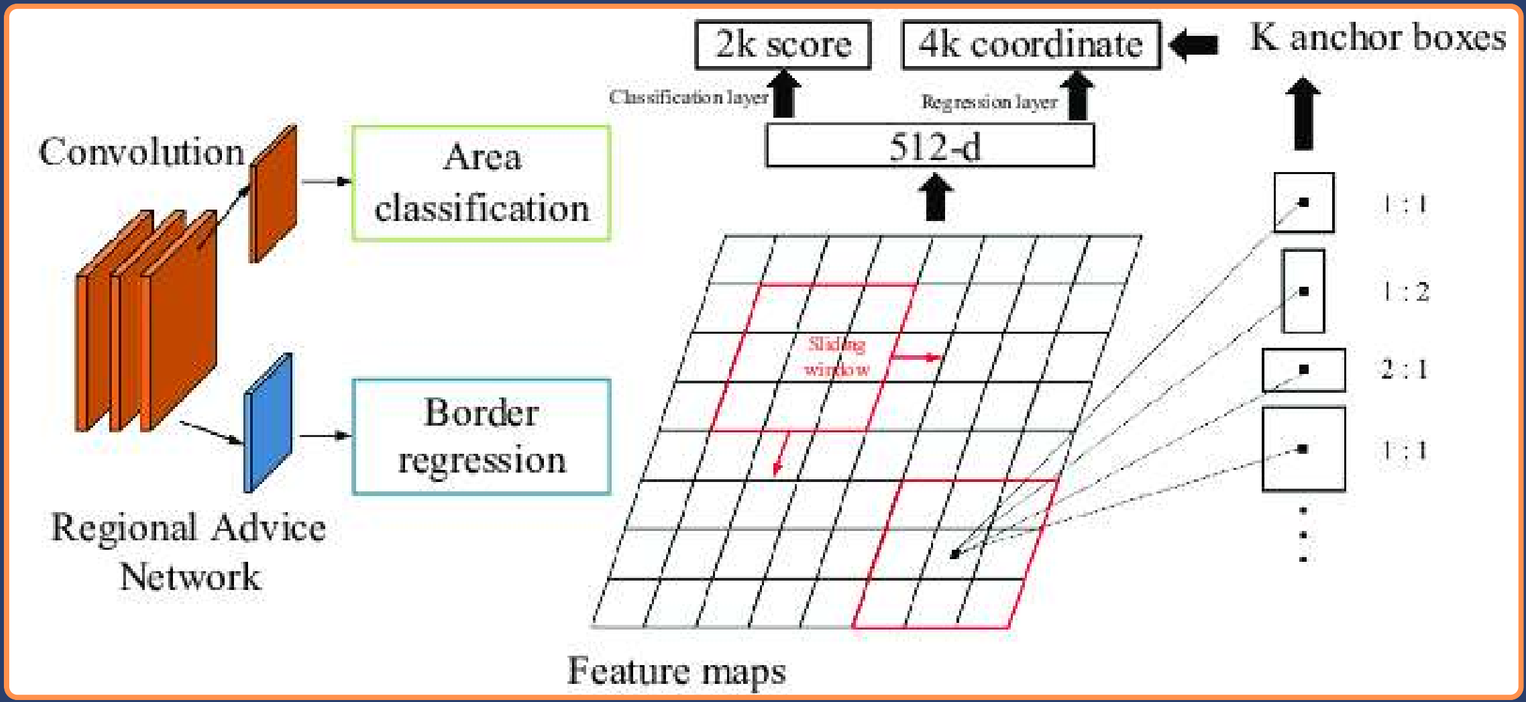
**Index Terms**—Object Detection, Region Proposal, Convolutional Neural Network.



# Dissecting Faster R-CNN

Rumoh Coding

# The Region Proposal Network (RPN)




A fully convolutional network that acts as the "first stage" of detection.

Output

- **Objectness Score**: Probability that a region is an object vs. background.
- **Box Regressors**: Rough coordinate adjustments for the proposed box.

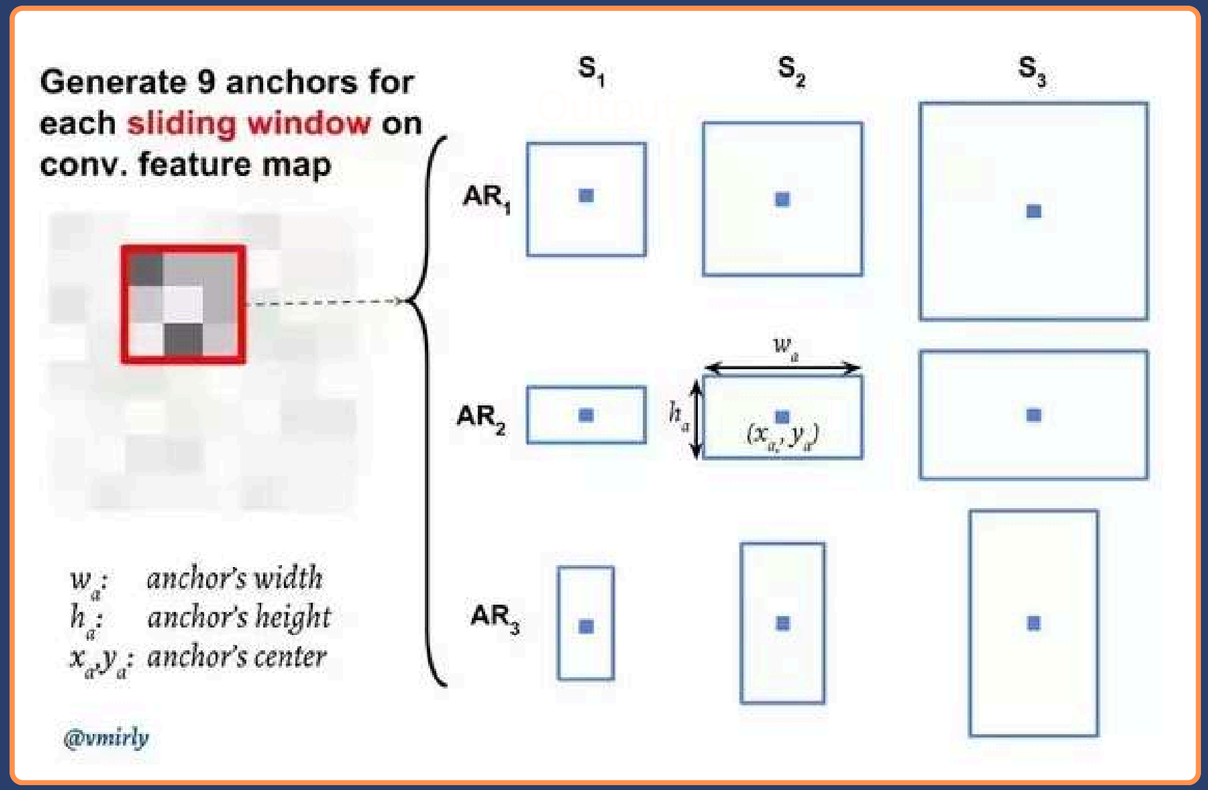
- **Goal**: Efficiently generate candidate bounding boxes (proposals) that are likely to contain objects.
- Mechanism
  - Slides a small window (e.g., 3 x 3) over the convolutional feature map generated by the backbone.
  - It uses the same feature map as the final classifier, making the proposal process computationally "cheap."

https://www.researchgate.net/figure/Region-proposal-Network-RPN_fig3_348772741

# The Concept of Anchor Boxes

How does a fixed-size sliding window detect objects of different shapes (e.g., a tall pedestrian vs. a wide bus)?

The Solution: Predefined reference boxes called Anchors.

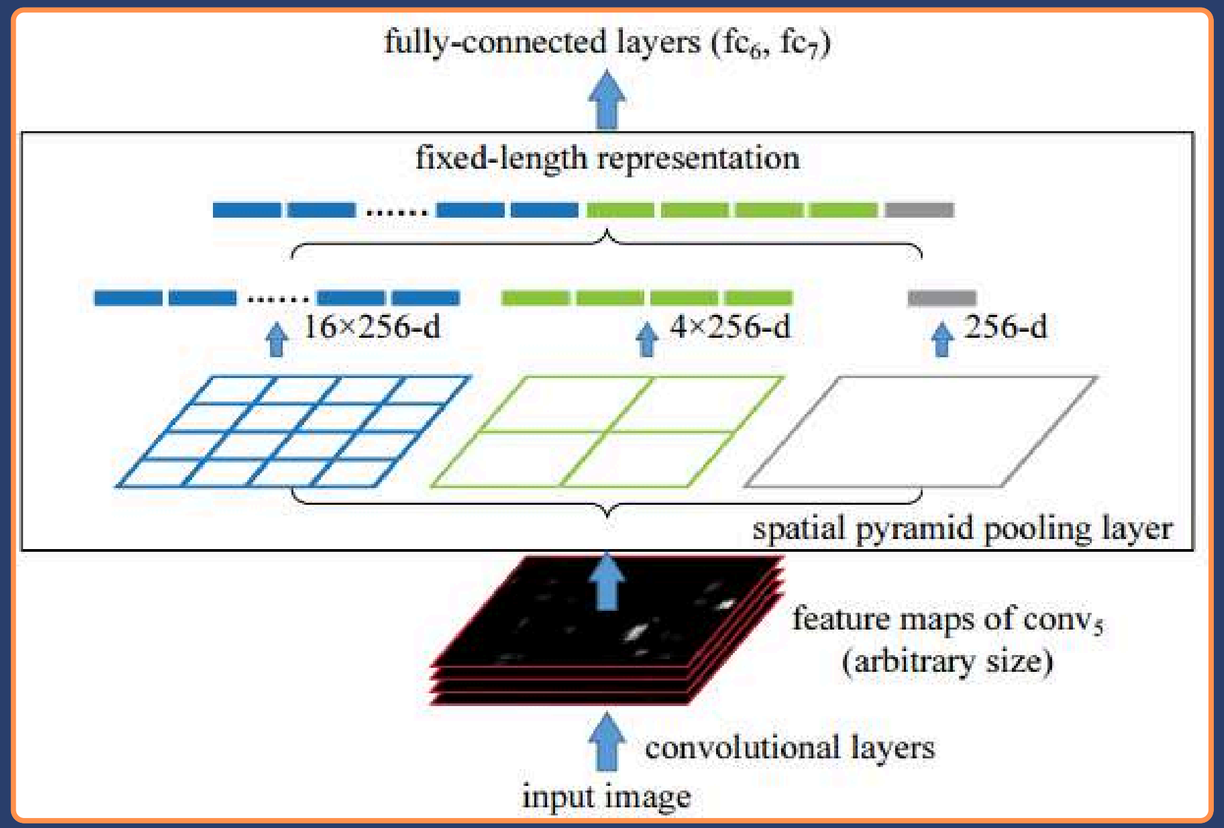



- Implementation
  - At every location on the feature map, the network predicts multiple boxes simultaneously.
  - Typically uses combinations of scales (e.g., 128^2, 256^2, 512^2) and aspect ratios (e.g., 1:1, 1:2, 2:1).
  - The model does not predict absolute coordinates from scratch; it learns the offset (delta) required to shift and resize the anchor box to match the ground truth.

https://datascience.stackexchange.com/questions/27277/faster-rcnn-how-anchor-work-with-slider-in-rpn-layer

# ROI Pooling

The RPN produces proposals of varying sizes (different widths/heights), but the final Fully Connected (FC) layers require a fixed input size (e.g., 7 x 7 vector).

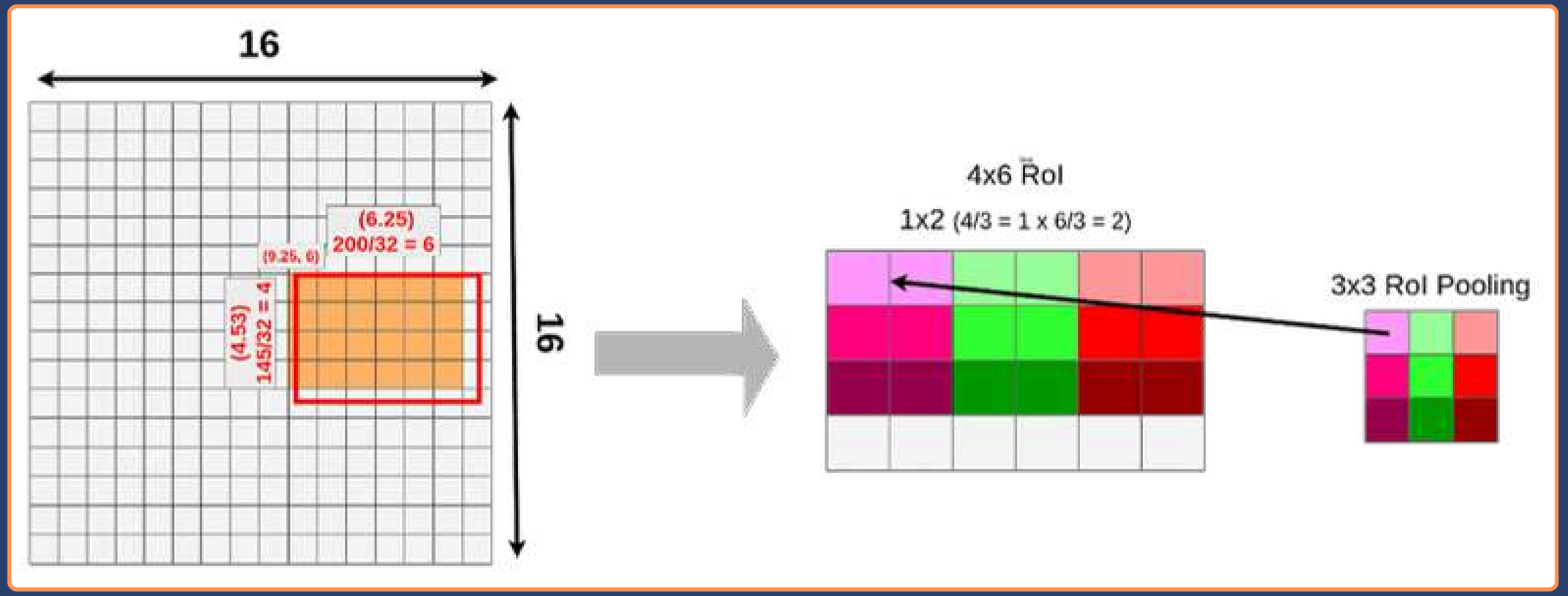



Use a distinct pooling layer that extracts features for a specific proposal from the global feature map.

- Mechanism
  - Project the proposal coordinates onto the feature map.
  - Divide the region into a fixed grid (e.g., 7 x 7).
  - Apply Max Pooling to each grid cell.

https://stackoverflow.com/questions/43430056/what-is-the-purpose-of-the-roi-layer-in-a-fast-r-cnn

# ROI Pooling Example

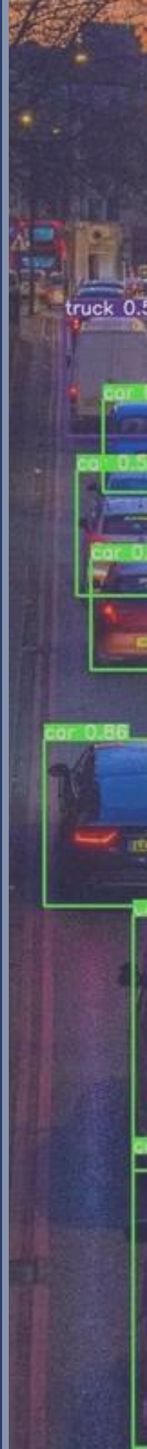



https://erdem.pl/2020/02/understanding-region-of-interest-part-2-ro-i-align

# Simple ROI Pooling Algorithm

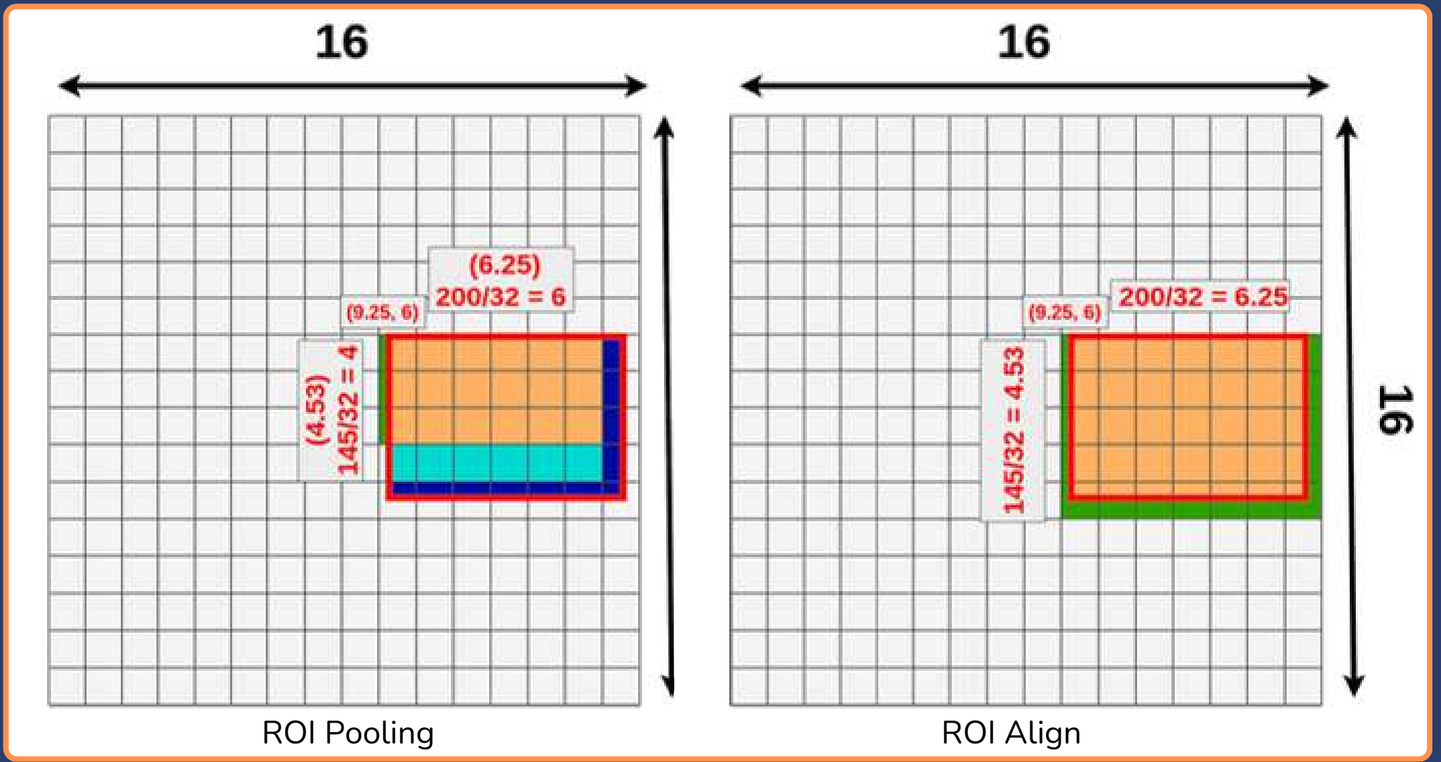
Rumah Coding

```python
import numpy as np
import math

def simple_roi_pooling(feature_map, roi, output_size):
    x, y, w, h = roi
    out_h, out_w = output_size

    # 1. Crop feature map sesuai ROI
    region = feature_map[y:y+h, x:x+w]

    # Persiapkan output kosong
    output = np.zeros(output_size)

    # Hitung ukuran setiap bin (kotak grid)
    bin_h = h / out_h
    bin_w = w / out_w

    # 2. Grid Division & Max Pooling
    for i in range(out_h):
        for j in range(out_w):
            # Tentukan batas awal dan akhir untuk setiap bin
            start_y = int(i * bin_h)
            end_y = int((i + 1) * bin_h)
            start_x = int(j * bin_w)
            end_x = int((j + 1) * bin_w)

            # Ambil potongan bin
            bin_region = region[start_y:end_y, start_x:end_x]

            # 3. Lakukan Max Pooling pada bin tersebut
            max_val = np.max(bin_region) if bin_region.size > 0 else 0
            output[i, j] = max_val

            print(f"Bin ({i},{j}) slice [{start_y}:{end_y}, {start_x}:{end_x}] -> Max:
{max_val}")
    return output
```



# ROI Align (Alternatif to ROI Polling)

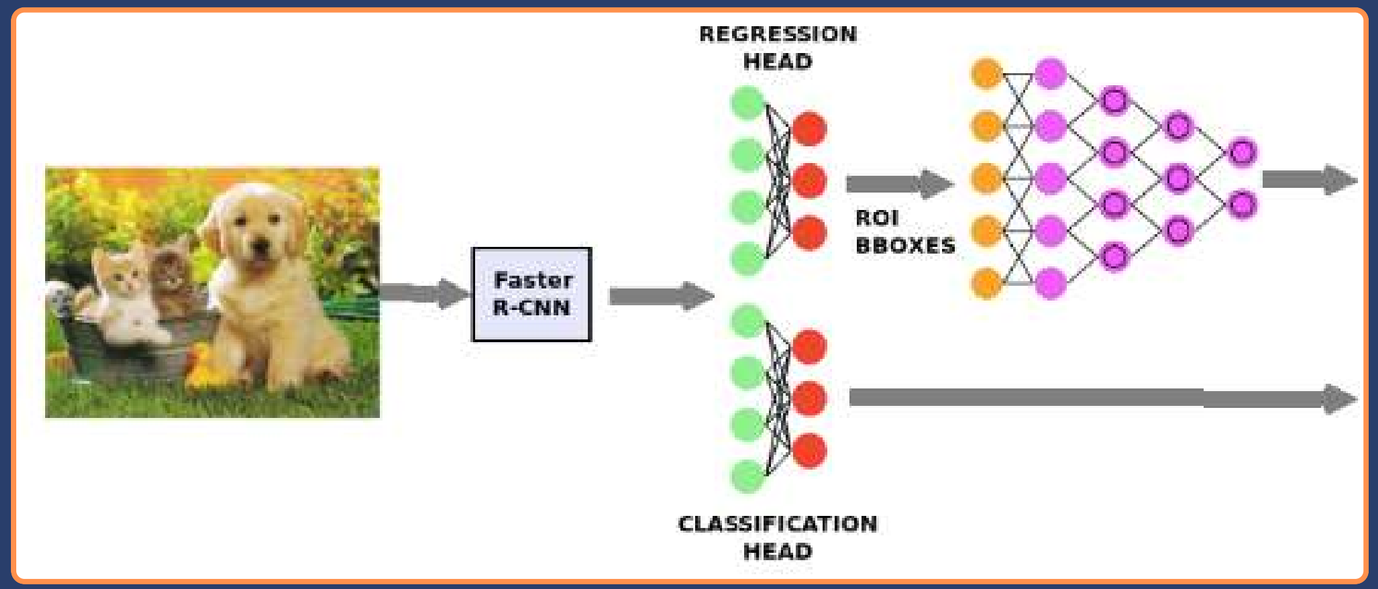
- ROI Pooling uses "quantization" (rounding off floating-point coordinates to integers).
  - **Example**: 100 / 16 = 6.25 => Rounded to 6.
  - Causes misalignment between the feature map and the original image, hurting accuracy for small objects or pixel-perfect tasks.
- ROI Align removes the rounding operation.
- It uses Bilinear Interpolation to calculate the exact values of the features at floating-point coordinates.




ROI Align was introduced in Mask R-CNN (2017)

https://erdem.pl/2020/02/understanding-region-of-interest-part-2-ro-i-align

# The Final Head - Classification and Regression

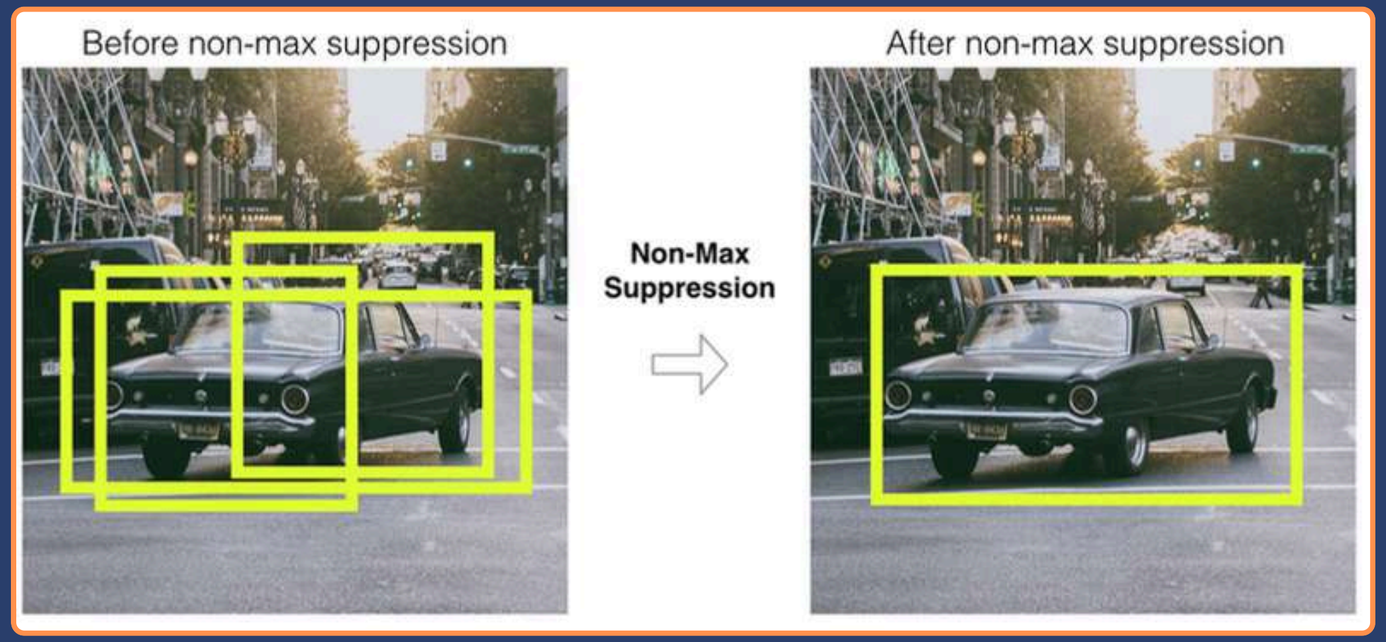



- **Input**: Fixed-size feature vectors from ROI Pooling.
- Dual Branches:
  - **Classifier**: Uses Softmax to predict the specific class (e.g., "Car", "Dog") or "Background".
  - **Bounding Box Regressor**: Fine-tunes the coordinates (x, y, w, h) one last time for maximum precision.

- **NMS (Non-Maximum Suppression)**: The final post-processing step to remove overlapping duplicate boxes, keeping only the one with the highest confidence score.

https://opendatascience.com/using-the-cnn-architecture-in-image-processing/

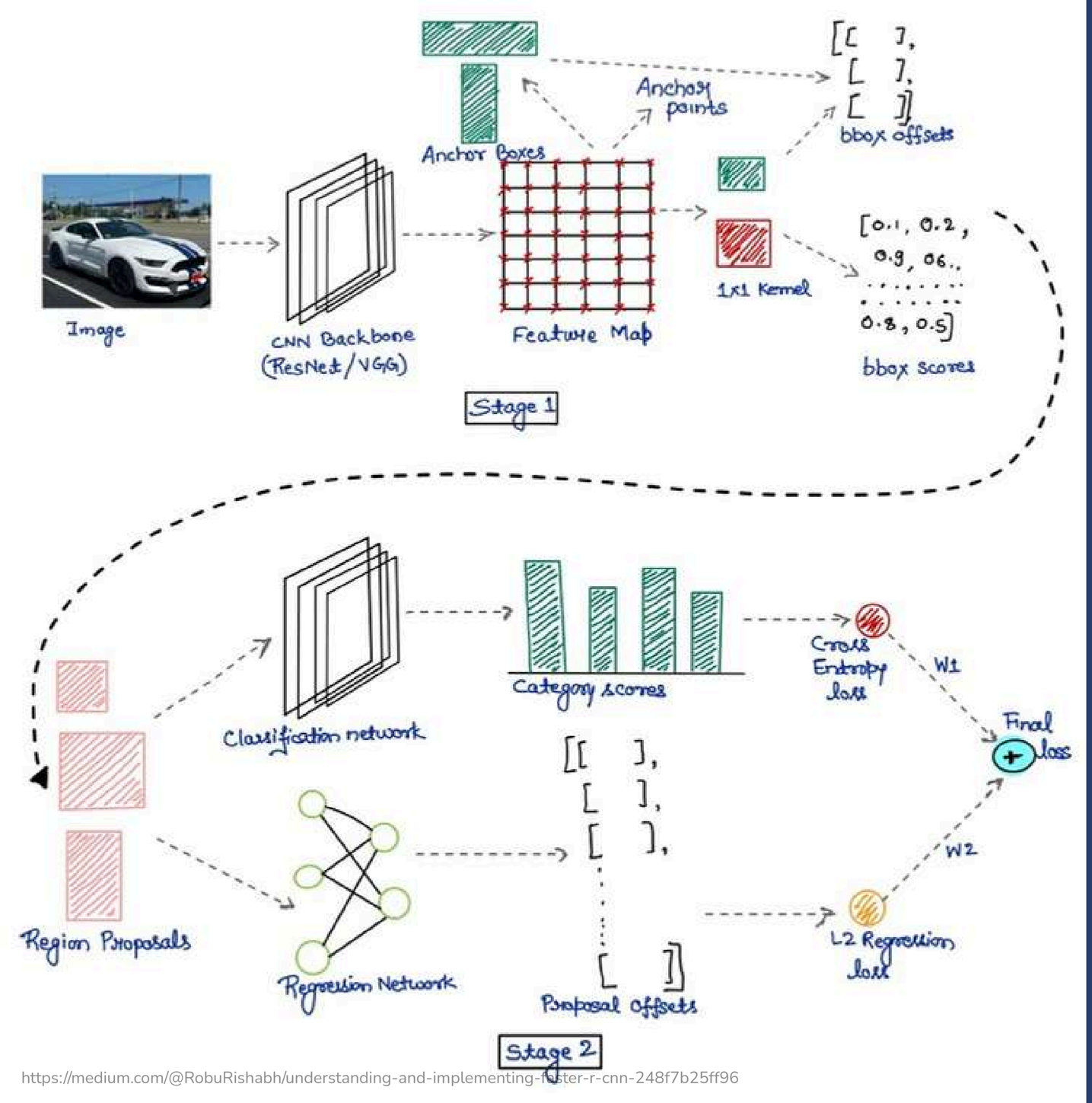
# How Non-Maximum Suppression Works




- Object detectors are designed to be spatially invariant. They often predict multiple bounding boxes for the same object with slightly different coordinates and scores.

- **Mechanism:**
  1. **Sort**: Rank all boxes by confidence score (High => Low).
  2. **Select**: Pick the box with the highest score as a valid detection (The "Maximum").
  3. **Compare** (IoU): Calculate the Intersection over Union (IoU) between this "Max" box and all other remaining boxes.
  4. **Suppress**: Discard any box that has an overlap (IoU) greater than a threshold (e.g., 0.5) with the Max box.
  5. **Repeat**: Move to the next highest scoring box and repeat until the list is processed.

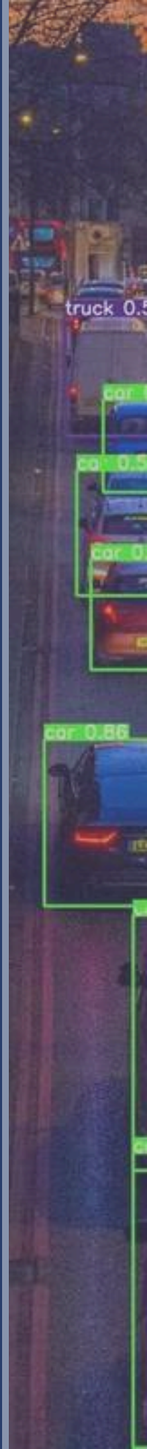
https://www.researchgate.net/figure/Non-Maximal-Suppression_fig5_345061606




https://medium.com/@RobuRishabh/understanding-and-implementing-faster-r-cnn-248f7b25ff96

Rumoh Coding

# Summary



# Pros & Cons

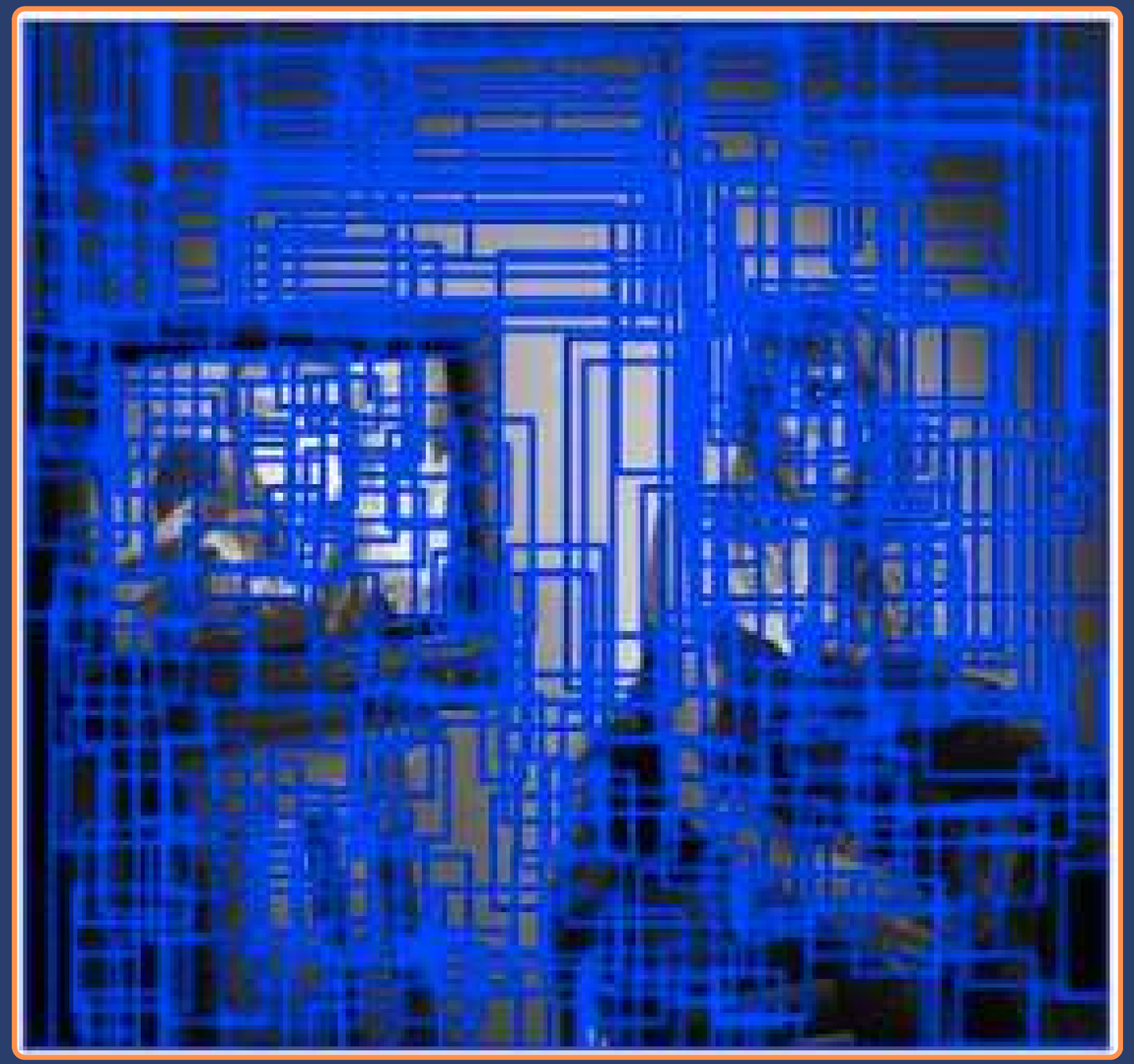
# The Strengths


- **Superior Accuracy**: The two-step process allows for "double-checking," resulting in lower false positives.
- **Small Object Detection**: ROI Align preserves spatial details better than one-stage grids, making it ideal for detecting small or distant objects.
- **Better Localization**: The regression head refines the box twice (once in RPN, once in the final stage), leading to tighter bounding boxes (high IoU).

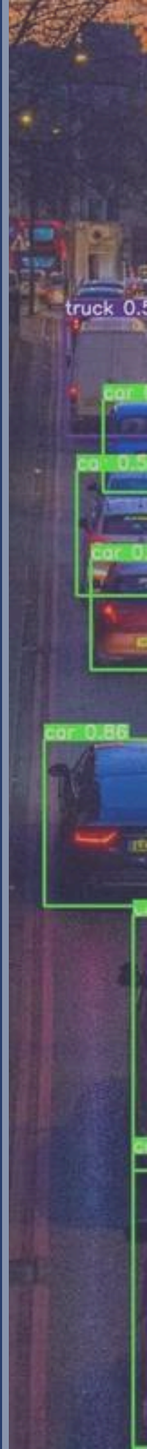
https://debuggercafe.com/object-detection-using-pytorch-faster-rcnn-resnet50-fpn-v2/

# The Limitations

- **Computational Bottleneck**: Generating and processing thousands of proposals creates high latency.
- **Lower FPS**: Typically runs between 5–15 FPS on standard GPUs (slower than real-time requirements).
- **Complex Training**: Often requires more memory (VRAM) and longer training times due to the complexity of the pipeline.

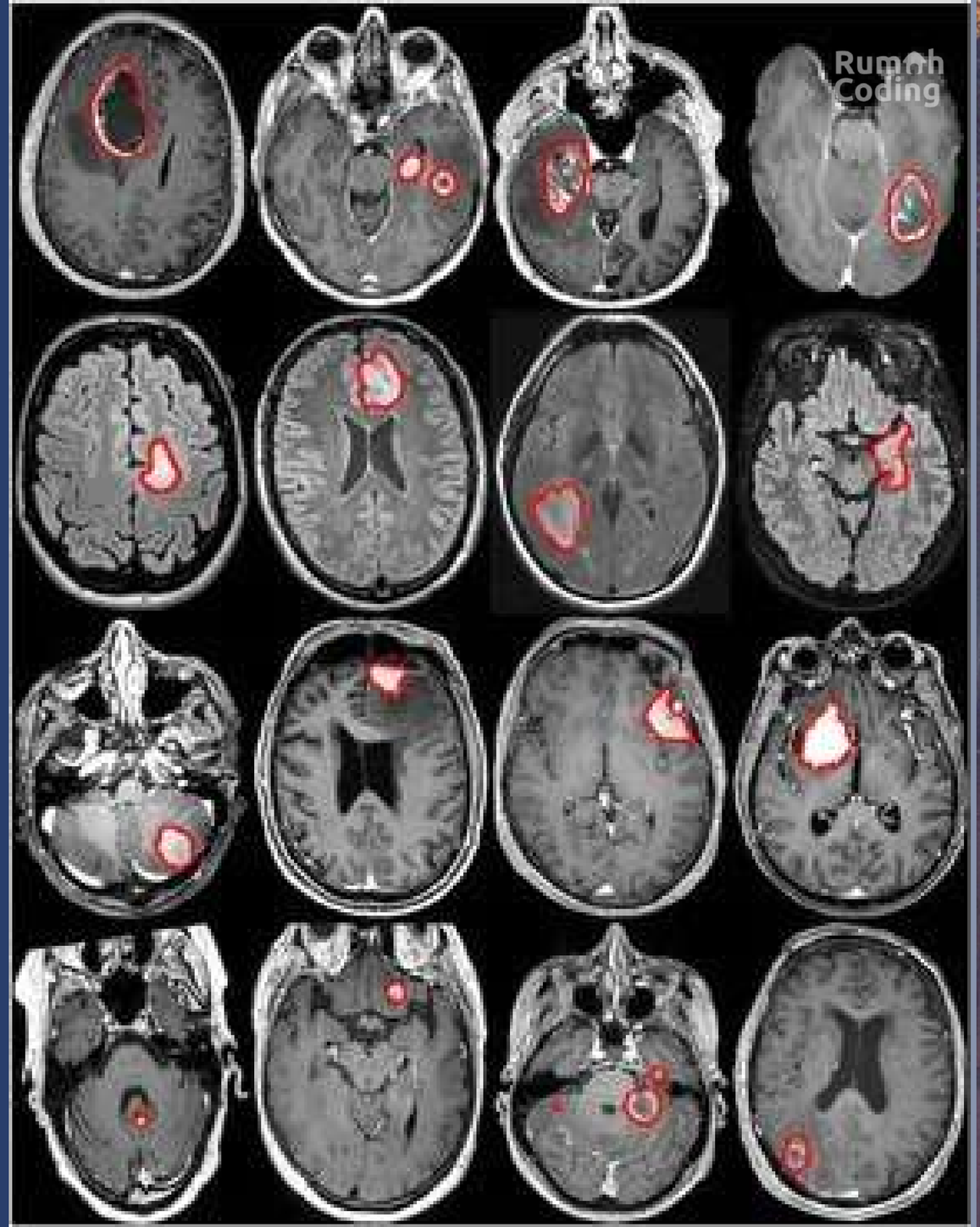


https://jamiekang.github.io/2017/05/28/faster-r-cnn/



# Ideal Use Cases

- **Medical Imaging**: Detecting tumors or anomalies where precision is critical.
- **Forensics/Security**: Analyzing high-resolution footage where speed is secondary to accuracy.
- **Robotics (Grasping)**: Where precise object boundaries are needed for manipulation.




https://www.frontiersin.org/journals/neurology/articles/10.3389/fneur.2022.932219/full

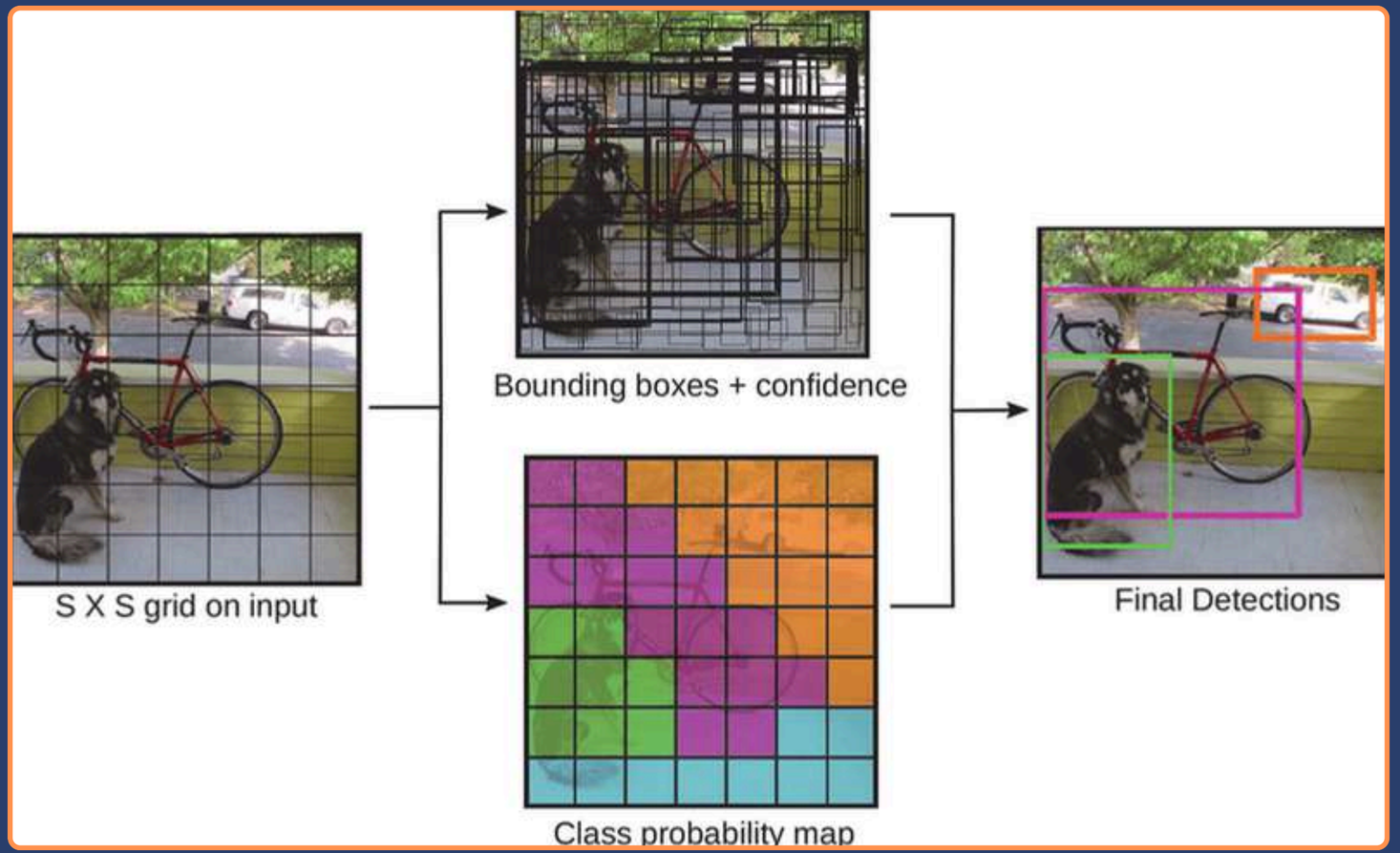
# Deep Dive into Single-Shot Detectors

Rumoh Coding

# Detection as a Single Regression Problem

- Instead of asking "Is there an object here?", the model looks at the whole image and asks "What objects are where?" in a single mathematical step.
- Mechanism
  - **Input**: Raw pixels.
  - **Process**: A single Convolutional Neural Network (CNN).
  - **Output**: Direct prediction of bounding box coordinates and class probabilities.

Eliminates the latency bottleneck caused by generating and resampling thousands of region proposals.

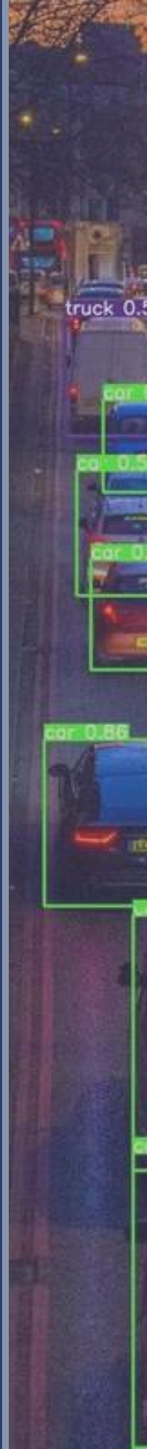



https://www.researchgate.net/figure/YOLO-model-detection-as-a-regression-problem-17-Thus-the-input-image-is-divided-into-a_fig5_337146307

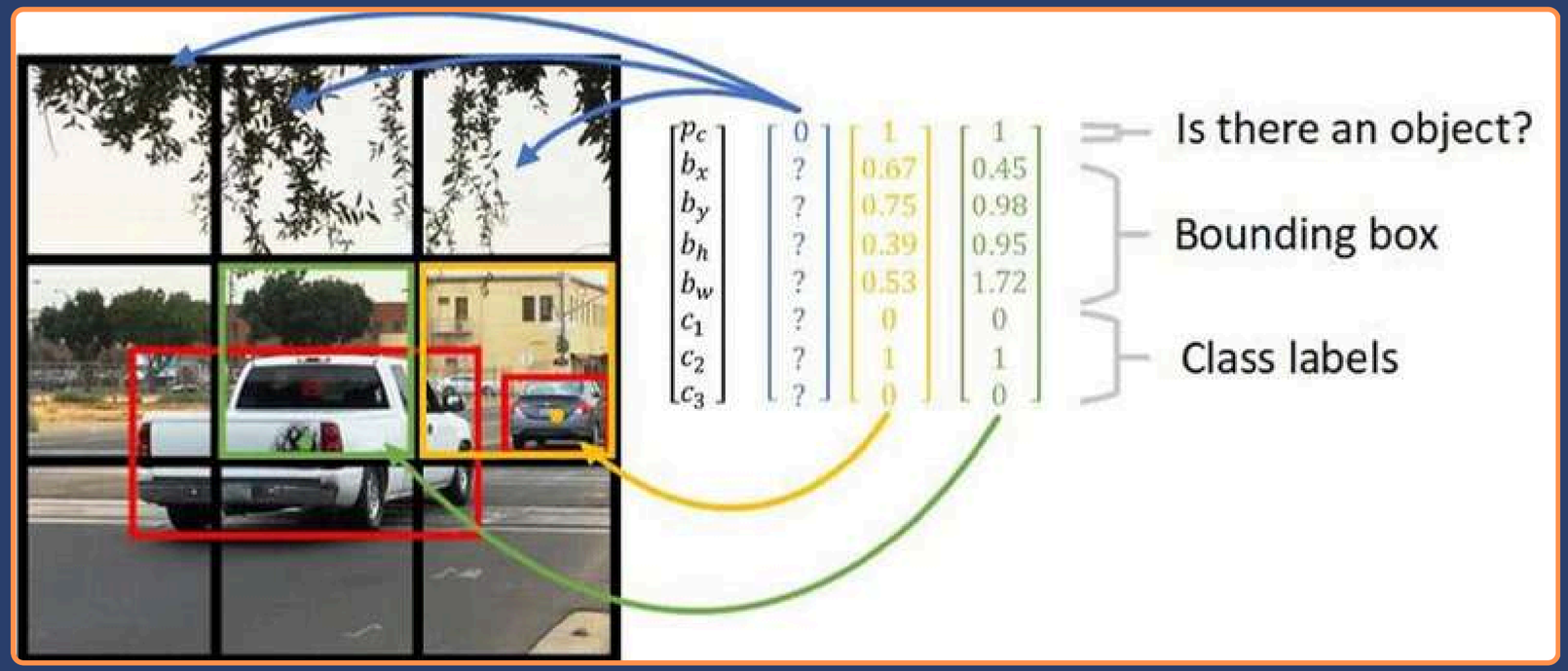
# Inside YOLO

Rumah Coding

Big Question:
Where to search without proposed region?



# The Grid System in YOLO




- The input image is divided into an S x S grid (e.g., 7 x 7).
- If the center of an object falls into a specific grid cell, that specific cell is responsible for detecting that object.

Note: In YOLO v1, each cell can only detect one object at a time

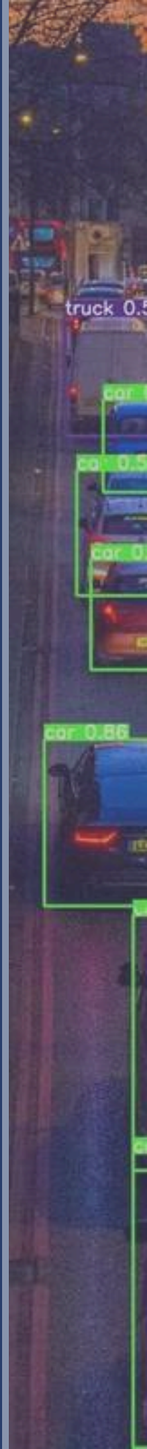
- Unlike sliding windows that process regions sequentially, the grid cells are processed simultaneously in a single forward pass of the network.

https://stackoverflow.com/questions/50575301/yolo-object-detection-how-does-the-algorithm-predict-bounding-boxes-larger-than

# Object to Cell Asignment

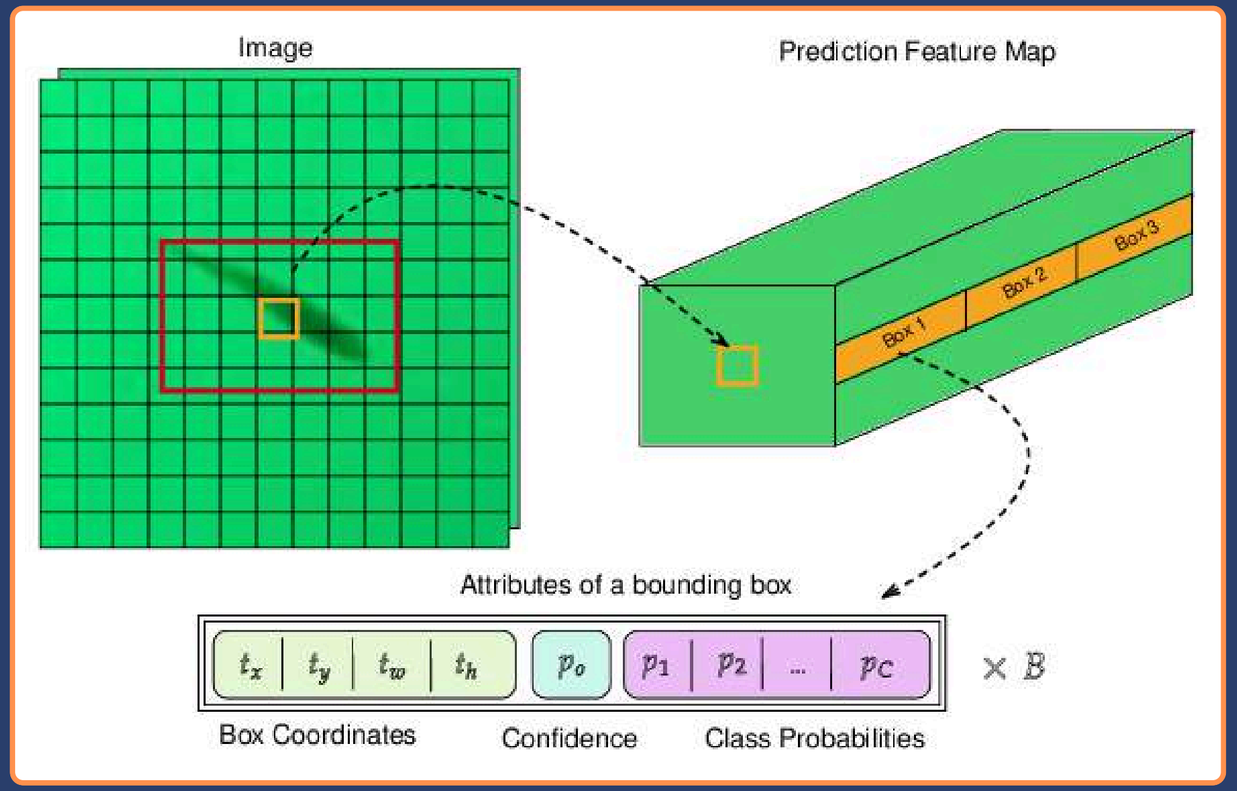
Rumah Coding

```python
import numpy as np

def encode_yolo_grid(image_size, grid_num, objects):
    # 1. Create an Empty Grid (S x S)
    grid_map = np.full((grid_num, grid_num), "Empty", dtype=object)

    cell_size = image_size / grid_num

    for obj in objects:
        name = obj['label']
        cx = obj['x'] # Center X
        cy = obj['y'] # Center Y

        # 2. Determine Grid Index (i, j)
        grid_x = int(cx / cell_size)
        grid_y = int(cy / cell_size)

        # 3. Calculate Relative Coordinates to the Top-Left Corner of that Cell
        x_relative_to_cell = (cx % cell_size) / cell_size
        y_relative_to_cell = (cy % cell_size) / cell_size

        # Assign object to the cell
        grid_map[grid_y, grid_x] = name
        print("-" * 30)

    return grid_map
```



# Unified Detection

- Each grid cell predicts a fixed number of Bounding Boxes (B) and Class Probabilities (C) at the exact same time.
- For each box, the network predicts 5 values:
  - **x, y** (Center coordinates relative to the grid cell).
  - **w, h** (Width and height relative to the image).
  - **Confidence** (IoU between predicted box and ground truth).




Because the network sees the entire image at once, it encodes contextual information better than region-based methods (e.g., fewer false positives for background errors).

https://www.researchgate.net/figure/Explanation-of-YOLOs-output-tensor-Illustration-inspired-by-13_fig1_339641783

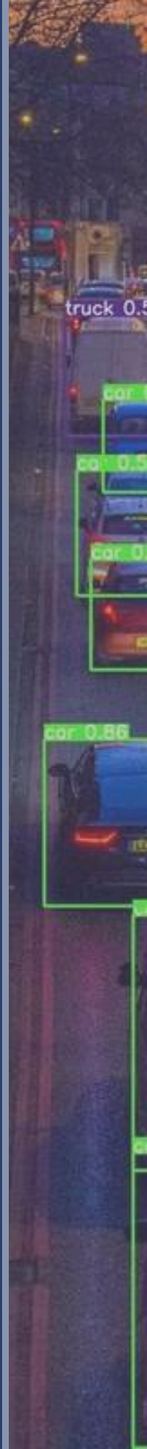
# The Anchor Era (YOLOv2 - YOLOv5)

Rumoh Coding

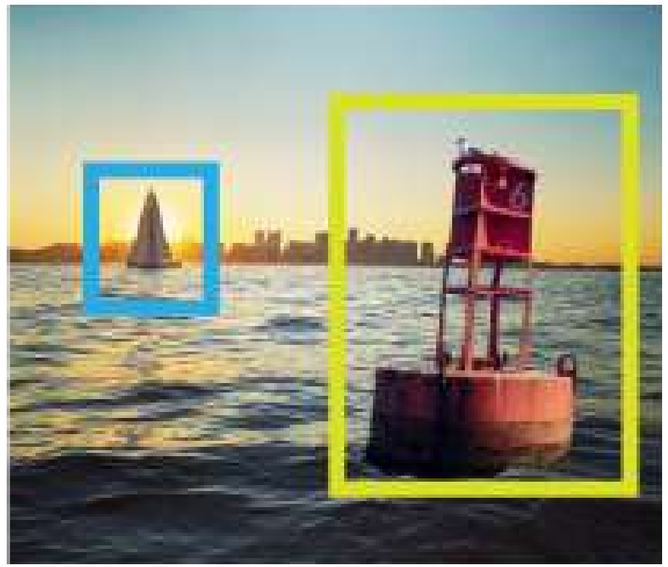
- **YOLOv2 (Better)**: Introduced Anchor Boxes (priors) to help the model learn object shapes, similar to Faster R-CNN. Introduced Batch Normalization.
- **YOLOv3 (Stronger)**: Introduced Multi-Scale Detection (FPN-like structure) to solve the "small object" problem. Predictions happen at three different scales.
- **YOLOv4/v5 (Faster)**: Focus on data augmentation (Mosaic, MixUp) and activation functions (Mish, SiLU) to maximize inference speed without sacrificing accuracy.

| Version | Release Year | Key Innovations and Features |
|---|---|---|
| **YOLOv2 (YOLO9000)** | 2016 | Improved localization and recall using anchor boxes, batch normalization, and a higher resolution classifier. Introduced the custom Darknet-19 backbone, which was faster and more accurate than its predecessor. |
| **YOLOv3** | 2018 | Enhanced performance with a deeper Darknet-53 backbone, predictions across multiple scales (for better small object detection), and a more sophisticated loss function. |
| **YOLOv4** | 2020 | Focused on achieving an optimal balance of speed and accuracy by integrating numerous modern techniques ("Bag of Freebies" and "Bag of Specials"), such as Mosaic data augmentation, Spatial Pyramid Pooling (SPP), and Path Aggregation Network (PANet) into a CSPDarknet backbone. |
| **YOLOv5** | 2020 | Developed by Ultralytics, this version emphasized ease of use, deployment, and a PyTorch implementation. It included further optimization with hyperparameter optimization, integrated experiment tracking, and easy export to various formats, offering different model sizes (small, medium, large) for various use cases. |

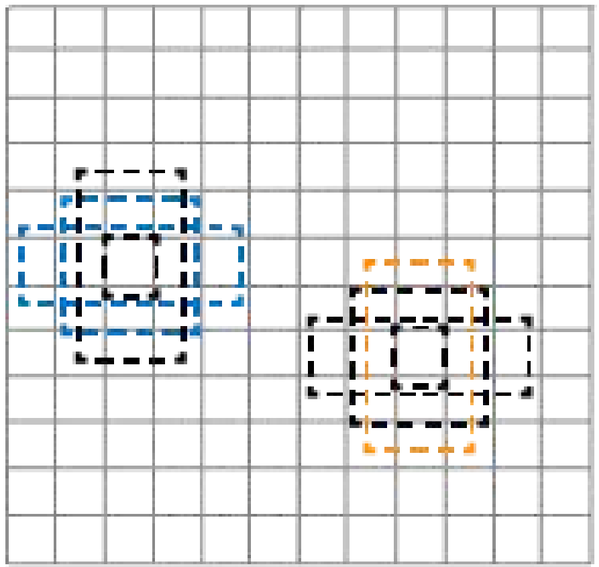
https://www.kaggle.com/code/vikramsandu/yolo-1-through-5-a-complete-and-detailed-overview/code



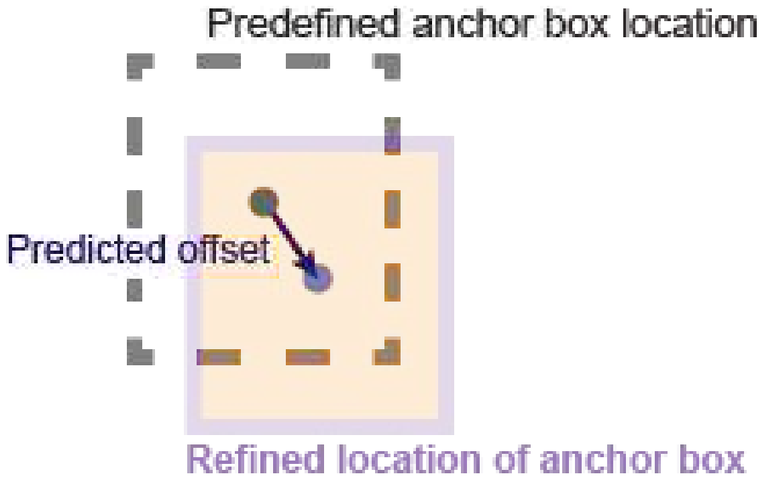
# The Anchor Box



Ground truth image and bounding boxes



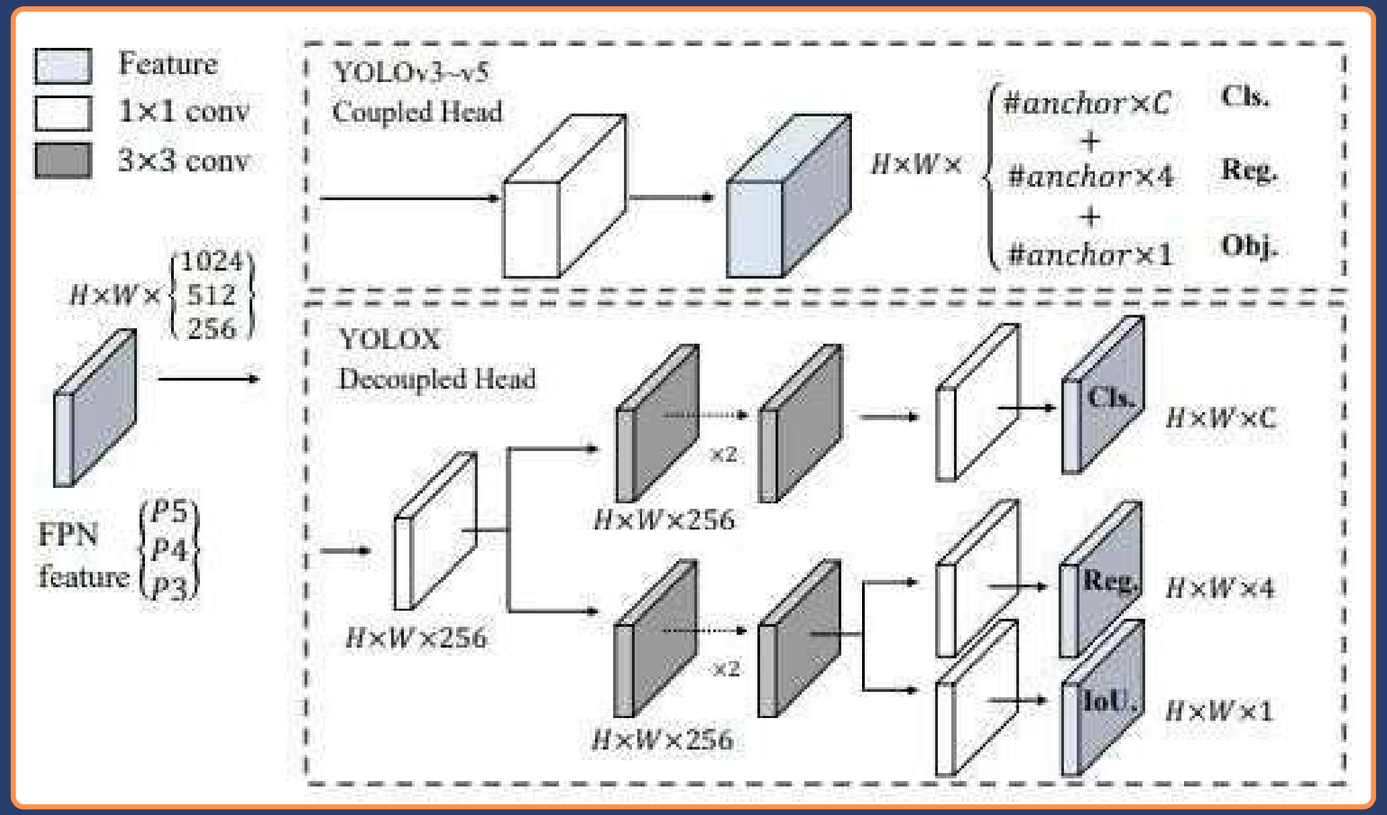
Anchor boxes at each predefined location in each feature map




https://www.mathworks.com/help/vision/ug/getting-started-with-yolo-v2.html

# The Modern Era (YOLOv8 - v11)

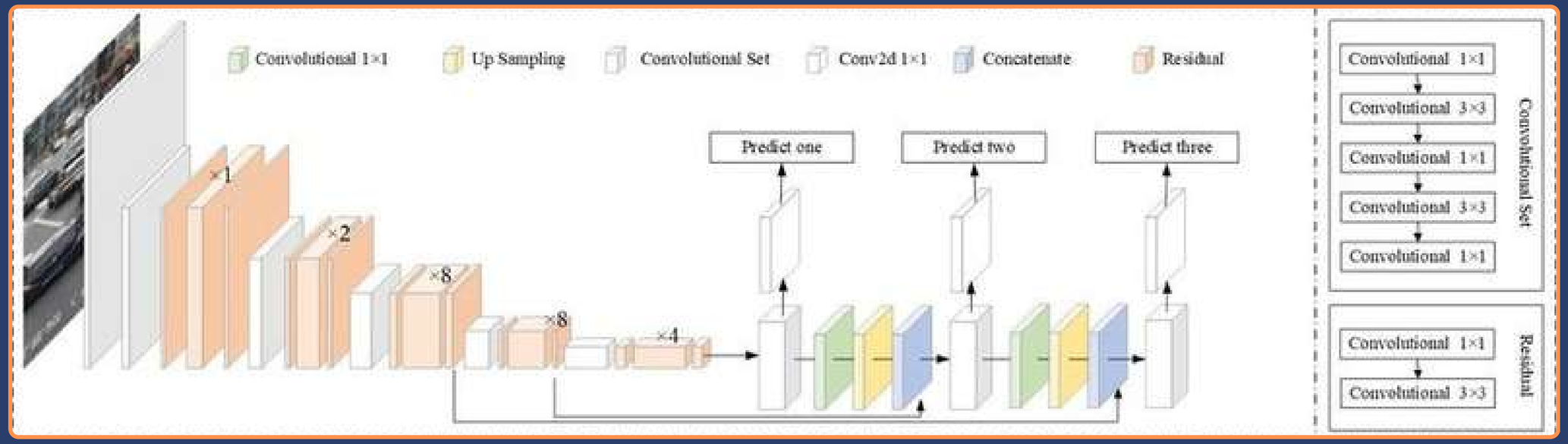
- Modern YOLOs (e.g., YOLOv8) removed Anchor Boxes to simplify the architecture and improve generalization on diverse datasets.
- **Decoupled Heads**: Separating the Classification branch and Regression branch in the final layer to allow each task to learn independently.
- **Focus**: State-of-the-Art (SOTA) trade-off between latency (ms) and accuracy (mAP), specifically optimized for Edge AI and mobile deployment.




https://www.researchgate.net/figure/The-head-layer-structure-of-YoloX-decoupled-classification-and-regression-40_fig3_373056795

# The YOLO Backbone

# The Classic Era - Darknet (v1 - v3)

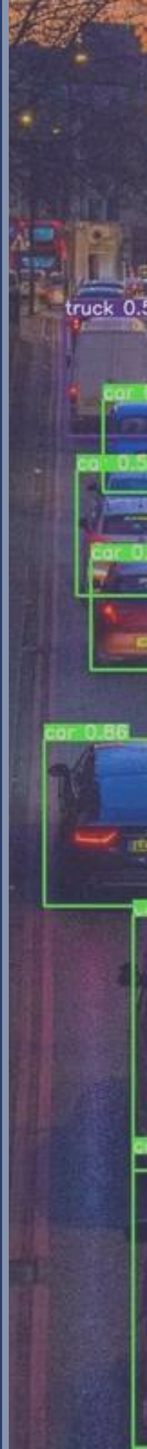



- Custom-designed by Joseph Redmon (creator of YOLO) because existing backbones (like VGG) were too slow for real-time applications.
- **Darknet-19 (YOLOv2)**: Utilized 19 convolutional layers. Extremely fast, offering moderate accuracy.
- **Darknet-53 (YOLOv3):**
  - Adopted Residual Connections (similar to ResNet) to stack layers deeper.
  - Achieved higher accuracy than ResNet-152 while being 2x faster.
  - Proved that we do not need massive, inefficient networks for accurate detection.

https://www.researchgate.net/figure/Structure-detail-of-YOLOv3It-uses-Darknet-53-as-the-backbone-network-and-uses-three_fig1_335865923

# ResNet vs. DarkNet

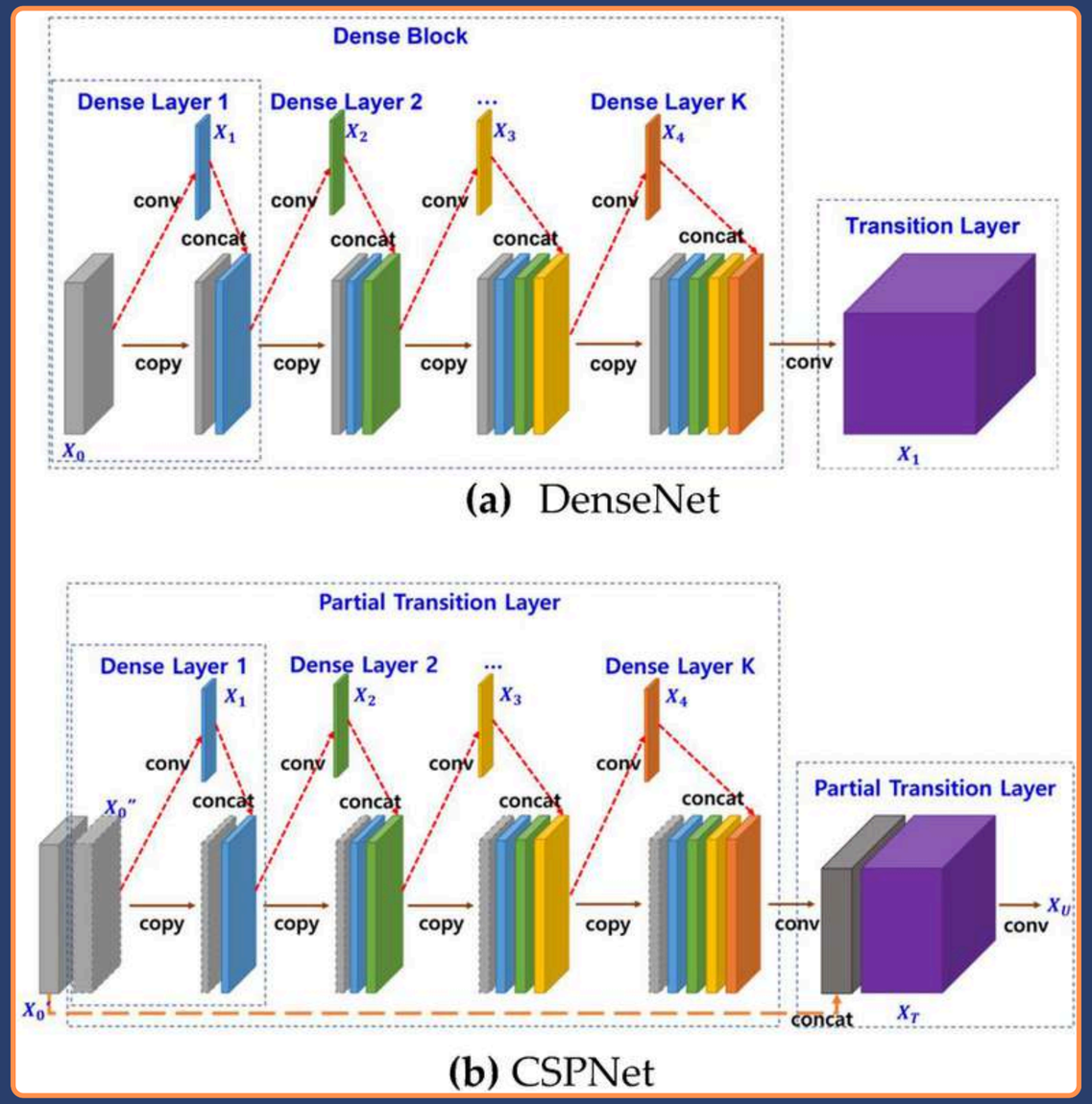
| Feature | ResNet (e.g., ResNet-50/101) | Darknet (e.g., Darknet-53) |
|---|---|---|
| Primary Design Goal | Deep Image Classification (ImageNet Winner). | Real-time Object Detection (YOLO Backbone). |
| Downsampling Method | Uses Max Pooling (aggressive data reduction) + Strided Convolution. | Uses Convolution with Stride 2 only (learnable downsampling, no pooling layers). |
| Building Block | Bottleneck Block (1x1 → 3x3 → 1x1). Compresses channels to save compute. | Basic Block (1x1 → 3x3). Maintains channel width for better information flow. |
| Activation Function | Standard ReLU (Rectified Linear Unit). | Leaky ReLU (Prevents "dying neurons" by allowing small negative values). |
| Depth vs. Width | Very Deep (up to 152+ layers) but "thin" bottlenecks. | Shallower (53 layers) but "wider" filters. |
| Performance (GPU) | Lower FLOPs, but higher latency (slower FPS). | Higher FLOPs, but higher FPS (optimized for parallel GPU processing). |



# The Efficiency Era - CSPNet (v4 - v5)

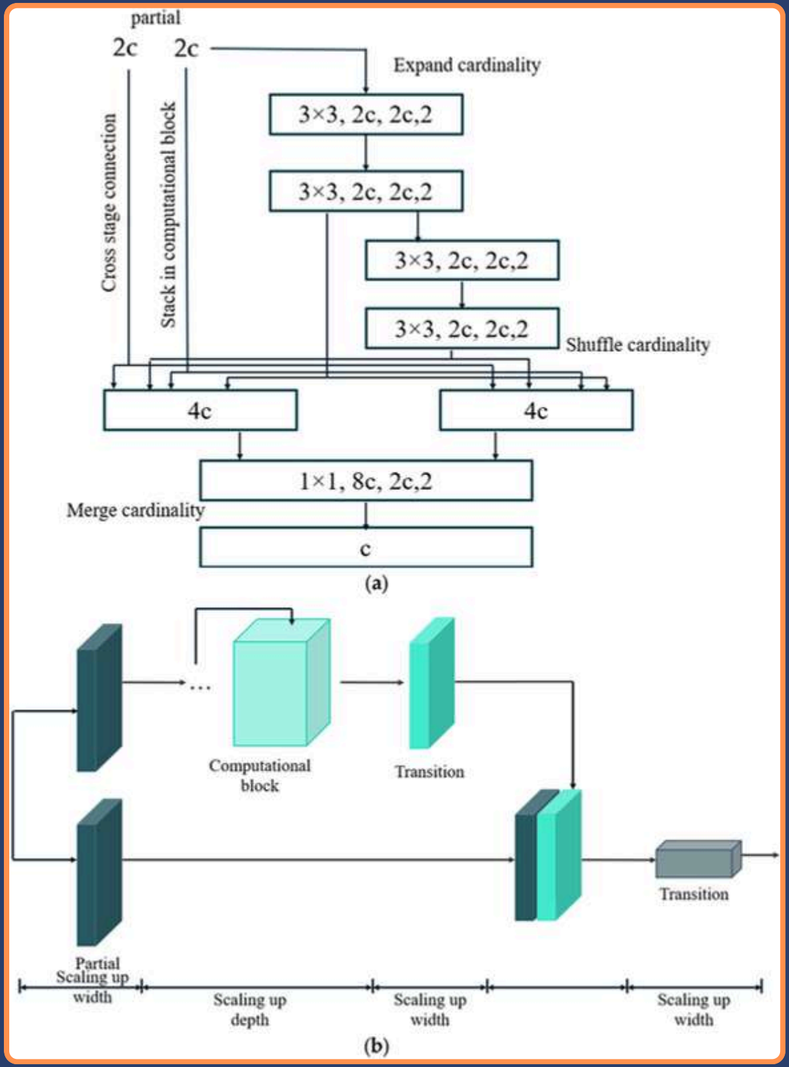
- In deep networks, duplicate gradient computations often occur, wasting memory and processing time.
- **CSPDarknet**
  - Splits the input Feature Map into two parts.
  - Part 1: Goes through the convolutional block (Dense Block).
  - Part 2: Bypasses the block entirely (skip connection) to the end.
  - Merge: Both parts are concatenated.

Reduces computation by ~20% without sacrificing accuracy. This became the de-facto standard for YOLOv4 and YOLOv5.




https://www.researchgate.net/figure/Structures-of-DenseNet-and-CSPNet-CSP-DenseNet_fig9_367209059

# The Modern Era - ELAN & C2f (v7 - v8)




- **YOLOv7 (E-ELAN)**: Introduced the Extended Efficient Layer Aggregation Network. Focuses on controlling the shortest and longest gradient paths to allow the network to learn efficiently from diverse data.

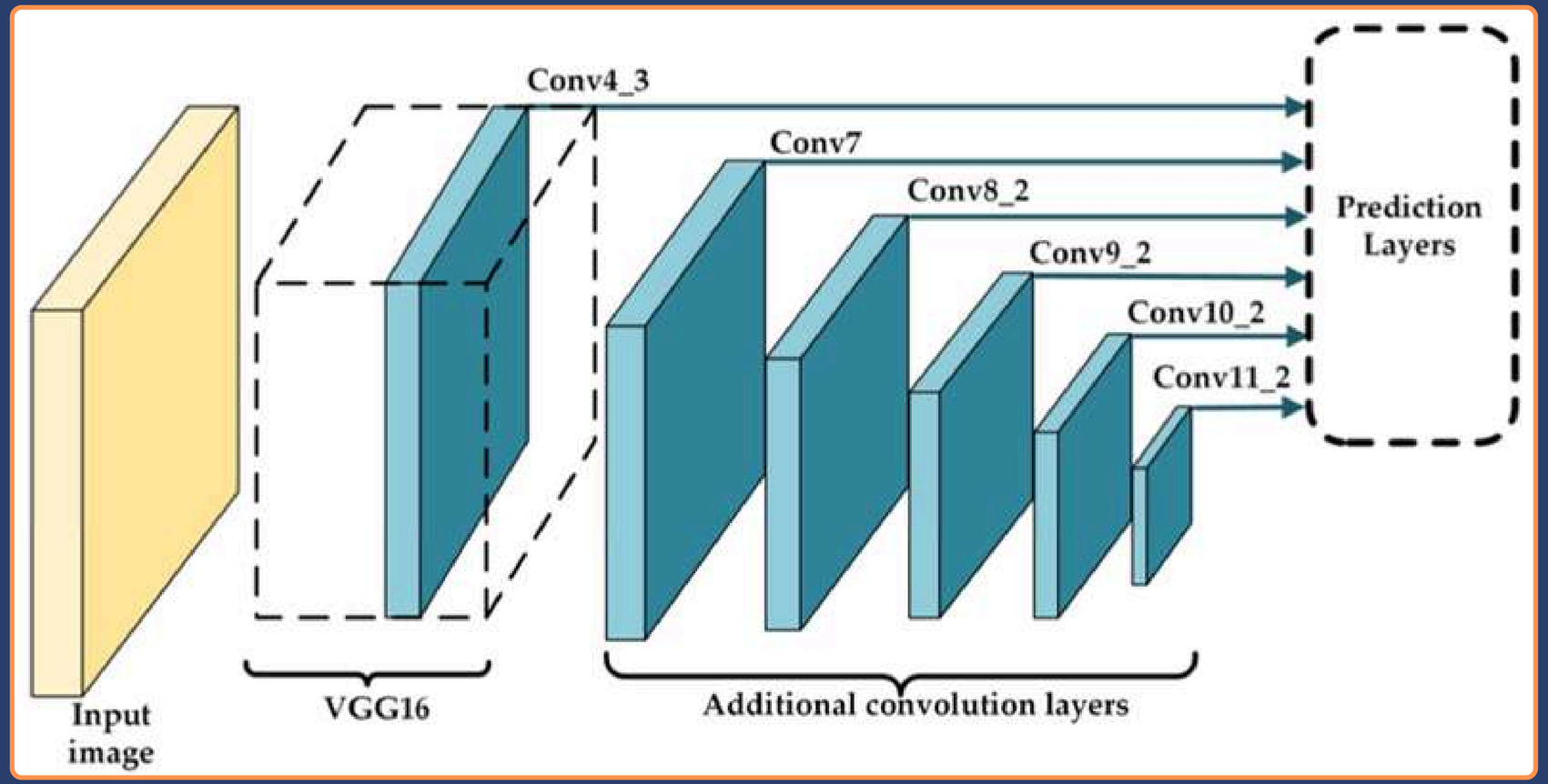
- **YOLOv8 (C2f Module)**
  - An evolution of the C3 module (CSP) from YOLOv5.
  - Adds more Skip Connections within the block.
  - Provides richer Information Flow, allowing even smaller models (Nano/Small) to detect complex features with low latency.

https://www.researchgate.net/figure/The-key-components-of-YOLOV7-where-a-are-the-E-ELAN-structures-and-b-is-the-model_fig3_376729860

# Inside SSD

# Single Shot MultiBox Detector (SSD)

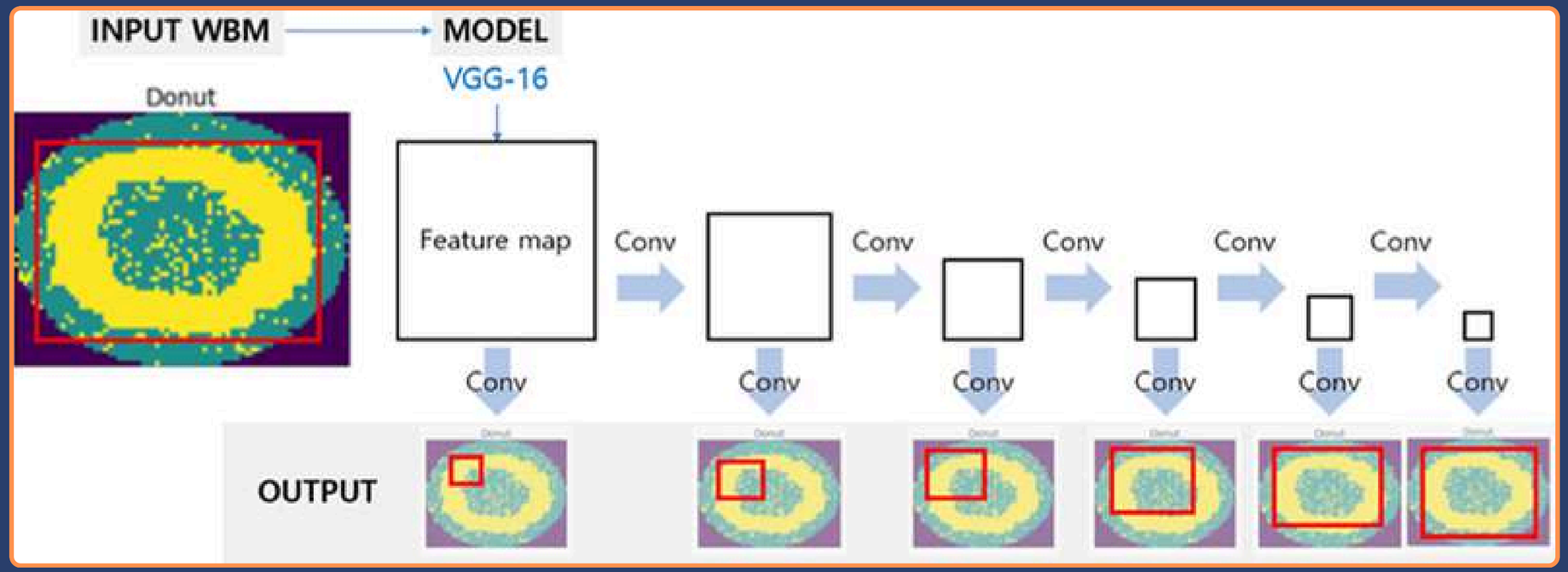
- Combine the speed of YOLO with the accuracy of Region-Based detectors (specifically for different object sizes).
- It does not rely on a single final feature map to make predictions.
- It adds a series of auxiliary convolutional layers to the end of the backbone, progressively decreasing in size.




- The result is a pyramidal structure where predictions happen at multiple layers of the network, not just the end.

https://medium.com/aimonks/ssd-neural-network-revolutionizing-object-detection-f655d8b4b521

# Multi-Scale Detection

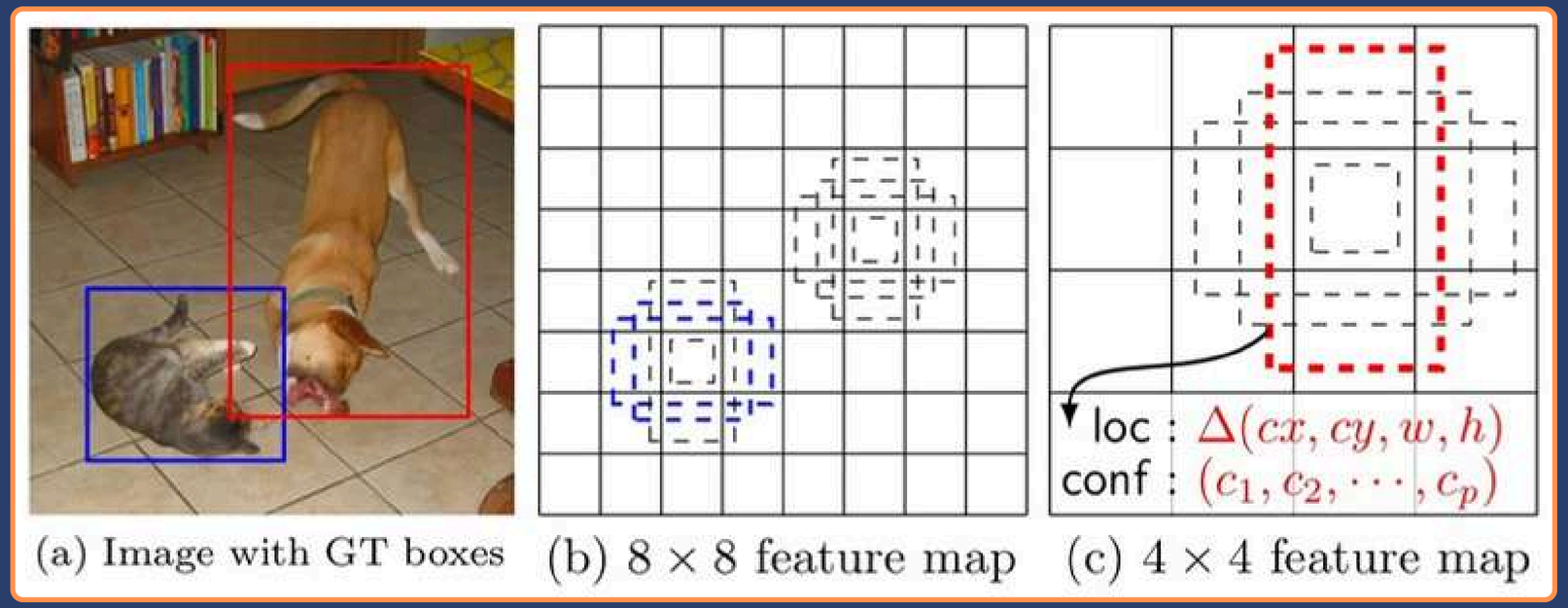



- The size of the feature map determines the size of the objects it can detect.
  - Early Layers (Large Resolution): Contain fine-grained details. Responsible for detecting small objects.
  - Deep Layers (Small Resolution): Contain high-level semantic info. Responsible for detecting large objects.
- As the image passes through the network, the system outputs bounding boxes at every stage of down-sampling.

https://www.researchgate.net/figure/Multi-scale-feature-maps-for-detection-in-SSD_fig4_350159612

# The "MultiBox" Concept

- SSD uses a set of pre-defined boxes at every grid cell of every feature map.
- Each feature map layer is assigned specific box scales and aspect ratios (e.g., 1:1, 1:2, 2:1, 3:1).
- For each default box, the network predicts:
  - **Offsets**: Adjustments to the box location and size (Localization).
  - **Confidences**: Scores for every object class (Classification).




- SSD produces significantly more predictions per image (e.g., 8,732 boxes) compared to early versions of YOLO (e.g., 98 boxes), increasing the chance of catching objects.

https://medium.com/@jesse419419/single-shot-multibox-detector-ssd-explained-by-a-j-dda10ba42a29

# SSD vs. The Competition

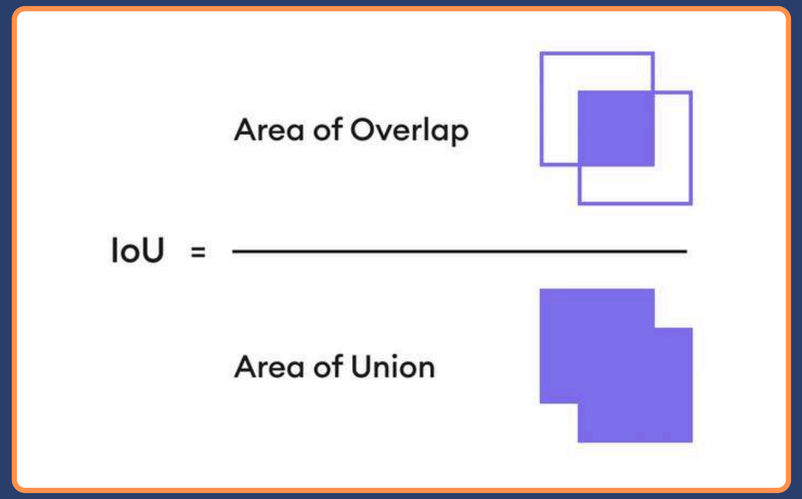
- **vs. Faster R-CNN**: SSD removes the Region Proposal Network and pixel resampling stage, resulting in much faster inference.
- **vs. Original YOLO**: By using lower-level feature maps, SSD historically performed much better on small objects than the original YOLO, which only predicted from the final (coarse) layer.

While modern YOLOs (v3+) have adopted multi-scale predictions, SSD was the architecture that pioneered this hierarchical approach for one-stage detectors.

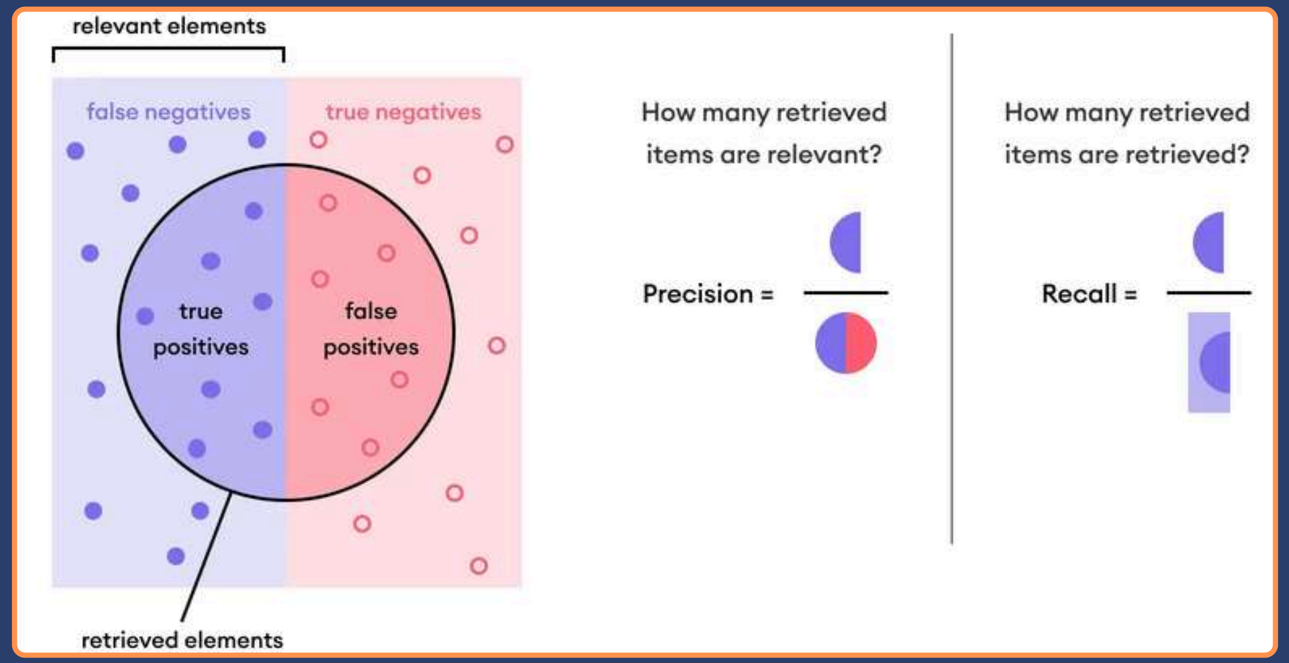
# Comparative Analysis

# Accuracy (mAP)

- Intersection over Union (IoU):
  - The metric for localization quality.
  - A detection is "correct" only if IoU > Threshold (e.g., 0.5 or 0.75)



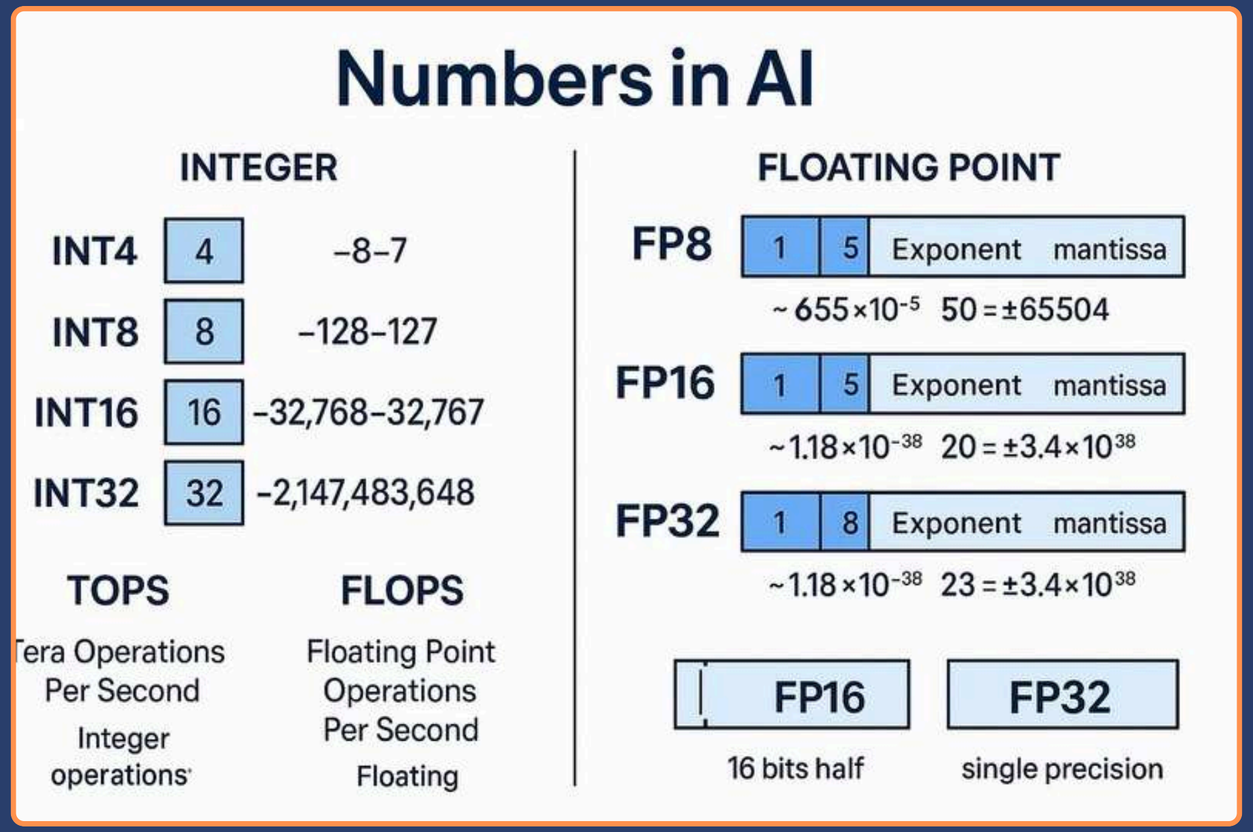

- mAP (mean Average Precision)
  - The Gold Standard metric. It calculates the area under the Precision-Recall curve.
  - mAP@0.5: Lenient metric (Good for general object counting).
  - mAP@0.5:0.95 (COCO Standard): Strict metric (Averages performance across 10 different IoU thresholds).




https://www.superannotate.com/blog/mean-average-precision-and-its-uses-in-object-detection

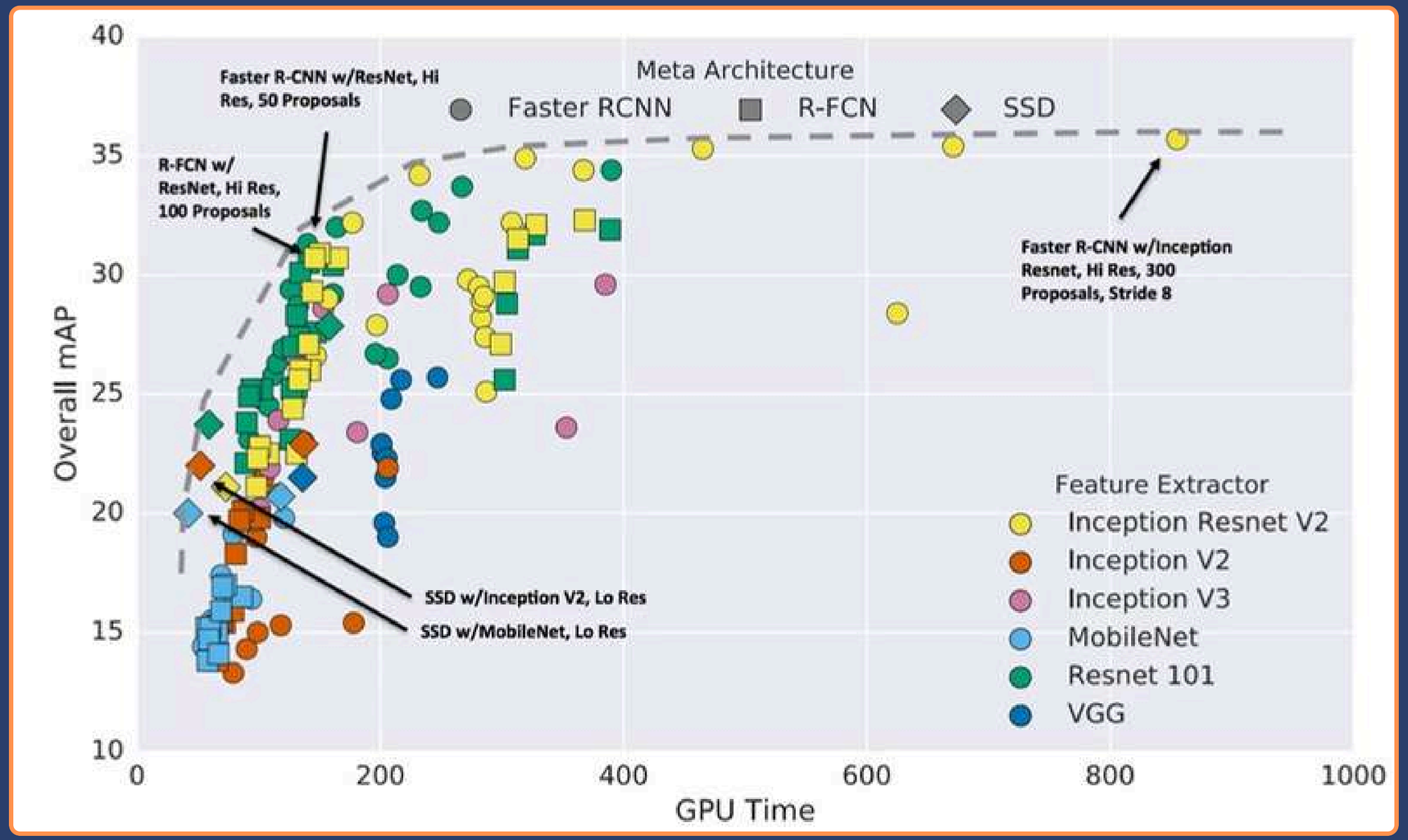
# Speed (FPS & Latency)

- Inference Speed (FPS)
  - **Frames Per Second**: Measures throughput. Critical for video processing.
  - **Target**: >30 FPS for real-time video, >60 FPS for smooth interaction.
- Latency (ms)
  - **Milliseconds per Image**: Measures the delay from input to output.
  - For autonomous systems (e.g., braking systems), low latency is safer than high FPS.
- FLOPs (Floating Point Operations)
  - Measures computational load. Higher FLOPs = Higher battery drain and hardware requirements.



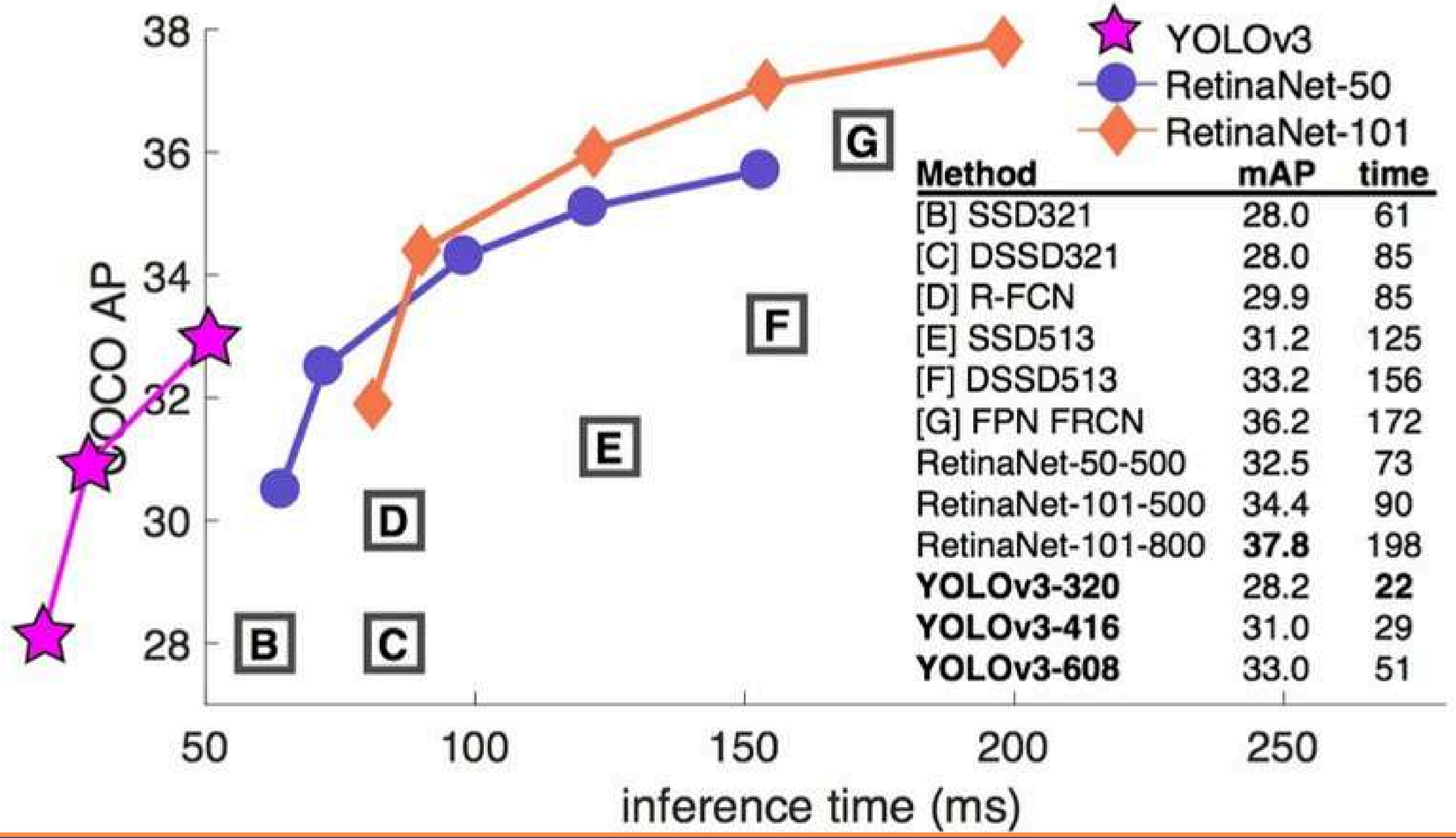

https://medium.com/@aquaone/what-are-tops-flops-int4-float-32-854cc79de75d

# Accuracy Comparison



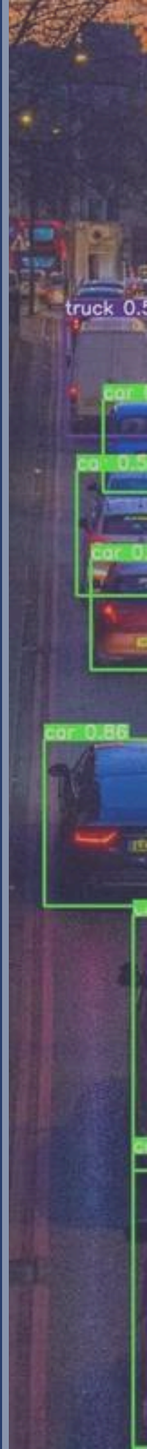

https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

# Accuracy Comparison




| Method | mAP | time |
|---|---|---|
| [B] SSD321 | 28.0 | 61 |
| [C] DSSD321 | 28.0 | 85 |
| [D] R-FCN | 29.9 | 85 |
| [E] SSD513 | 31.2 | 125 |
| [F] DSSD513 | 33.2 | 156 |
| [G] FPN FRCN | 36.2 | 172 |
| RetinaNet-50-500 | 32.5 | 73 |
| RetinaNet-101-500 | 34.4 | 90 |
| RetinaNet-101-800 | **37.8** | 198 |
| **YOLOv3-320** | 28.2 | **22** |
| **YOLOv3-416** | 31.0 | 29 |
| **YOLOv3-608** | 33.0 | 51 |

https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359



## 🔗 How to Cite

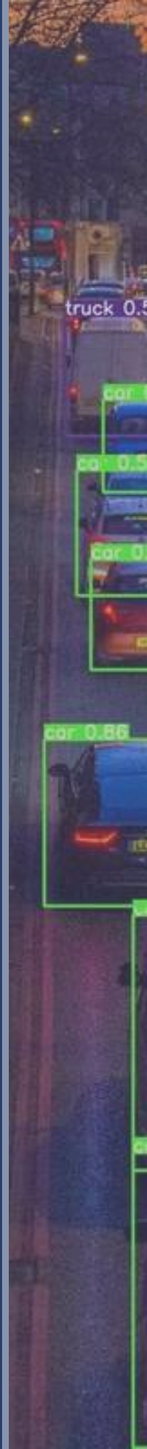
If you find these materials useful for your research or study, please cite this repository:

**APA Style:**

> Fazry, L. (2026). *Analyzing Object Detection Architectures*. GitHub. https://github.com/lhfazry/Analyzing-Object-Detection-Architectures

**BibTeX:**

```
@misc{fazry2026analyzing,
  author = {Fazry, Lhuqita},
  title = {Analyzing Object Detection Architectures},
  year = {2026},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{[https://github.com/lhfazry/Analyzing-Object-Detection-Architectures](https://github.com/lhfa
}
```



# Terima Kasih

Lhuqita Fazry