Deep Analysis Report on Core Object Detection Algorithms

A Multi-Dimensional Analysis of Faster R-CNN, YOLOv1/v8, RetinaNet, DETR, and RT-DETR

Name: Xiao Tian Major: Big Data, Class of 2025

RILAB Intelligent Detection Group - Phase 2 Assessment Report

Abstract

This report selects five of the most representative object detection papers from the literature surveyed in the Phase 1 review - Faster R-CNN (classic two-stage baseline), YOLOv1/v8 (pioneer and latest iteration of single-stage real-time detection), RetinaNet (single-stage accuracy breakthrough), DETR (pioneer of Transformer-based end-to-end detection), and RT-DETR (current SOTA real-time detector) - and conducts an in-depth analysis along three dimensions: (1) a conversational explanation of each algorithm's strengths, weaknesses, differences, and inheritance relationships; (2) an analysis of the physical meaning, value ranges, and mapping relationships of each model's input/output tensors; (3) selection of core computational modules (RPN, grid prediction mechanism, Focal Loss, Multi-Head Self-Attention, hybrid encoder) with pseudocode derivations of tensor shape transformations within each module. All viewpoints, formulas, and data are annotated with original references.

Keywords: Object Detection; Faster R-CNN; YOLO; RetinaNet; DETR; RT-DETR; Deep Analysis

I. Introduction

Object detection is a fundamental core task in computer vision, aiming to simultaneously classify and spatially localize objects in images [28]. Since R-CNN [1] introduced deep learning to object detection in 2014, the field has undergone multiple paradigm shifts - from two-stage to single-stage, from CNN to Transformer, and from pursuing accuracy to balancing real-time performance. Based on the literature survey from the Phase 1 review, this report selects five of the most representative papers from 30 references for in-depth analysis:

(1) Faster R-CNN [3] - The culmination of two-stage detection, the first to achieve end-to-end region proposal generation and detection, and one of the most cited classic baselines in object detection;

(2) YOLOv1 [5] and YOLOv8 [6] - The pioneer and latest iteration of single-stage real-time detection. YOLOv1 pioneered the entirely new paradigm of treating detection as a regression problem, while YOLOv8 represents the engineering-optimal solution after eight generations of YOLO evolution;

(3) RetinaNet [8] - Through Focal Loss, it was the first to enable single-stage detectors to surpass two-stage models in accuracy, solving the class imbalance problem that had long plagued single-stage detectors;

(4) DETR [9] - The first to introduce Transformers into object detection, pioneering an end-to-end detection paradigm requiring neither anchors nor NMS, marking the transition from the CNN era to the Transformer era;

(5) RT-DETR [12] - The current SOTA real-time detector, achieving real-time Transformer detection through a hybrid encoder architecture, striking the optimal balance between accuracy and speed.

These five papers form a clear chain of technological evolution: Faster R-CNN established the region-based detection paradigm -> YOLOv1 pioneered single-stage real-time detection, improving speed by an order of magnitude -> RetinaNet proved that single-stage detectors could rival two-stage models in accuracy -> DETR completely abandoned anchors and NMS, pioneering a new set prediction paradigm -> RT-DETR solved the real-time bottleneck of Transformer detectors. The following sections provide in-depth analysis along three dimensions: algorithm strengths/weaknesses and inheritance relationships, input/output tensor analysis, and core module tensor derivations.

II. Analysis of Algorithm Strengths, Weaknesses, and Inheritance Relationships

(1) Faster R-CNN: The Culmination of Two-Stage Detection

When it comes to the history of object detection, Faster R-CNN [3] is an absolute milestone. Before it, R-CNN [1] was the first to prove that CNNs could be used for object detection, but that approach was frankly too clumsy - generating about 2,000 candidate regions per image, each requiring a separate pass through the CNN for feature extraction. The computational cost was enormous, taking 47 seconds per image, making it completely impractical. Fast R-CNN [2] took a smarter step by introducing ROI Pooling so the entire image only passes through the CNN once, improving speed by over 10x. However, candidate region generation still relied on the traditional Selective Search algorithm, which became the new speed bottleneck.

Faster R-CNN's core contribution was proposing the Region Proposal Network (RPN), which uses a neural network for candidate region generation and shares the same CNN backbone with the detection network [3]. This reduced the candidate region generation time from 2 seconds (Selective Search) to the 10-millisecond level, achieving truly end-to-end training for the first time. The RPN slides a window across the feature map, generating k anchor boxes of different scales and aspect ratios at each position (default 9: 3 scales x 3 ratios), predicting whether each anchor contains an object and the bounding box refinement offsets [3].

Faster R-CNN's advantages are clear: high detection accuracy, achieving 73.2% mAP on PASCAL VOC 2007, which was the absolute SOTA at the time; the two-stage 'coarse screening then fine refinement' strategy ensures stable localization accuracy; the anchor mechanism can cover objects of various scales and aspect ratios [3]. However, its problems are also prominent: first, speed is still insufficient - although much faster than R-CNN, 5 FPS inference speed is far from real-time detection (30 FPS); second, anchor box scales and ratios require manual design and may need re-tuning for different datasets; third, NMS post-processing tends to incorrectly remove valid detection boxes in dense occlusion scenarios [3].

(2) YOLO Series: The Creation and Evolution of Single-Stage Real-Time Detection

If Faster R-CNN represents the 'accuracy-first' detection philosophy, then YOLO (You Only Look Once) is the ultimate representative of 'speed-first.' In 2016, Redmon et al. proposed YOLOv1 [5], pioneering an entirely new detection approach: instead of first finding candidate regions and then classifying like two-stage models, it feeds the entire image through the network at once, directly regressing all object locations and categories. In the authors' words, 'you only need to look once' [5].

YOLOv1's core principle is very intuitive [5]: divide the input image into an SxS grid (default 7x7), where each grid cell is responsible for predicting objects whose centers fall within that cell. Each grid cell predicts B bounding boxes (default B=2), each containing 5 values (x, y, w, h, confidence), along with C class probabilities (C=20 for PASCAL VOC). The entire network output is a tensor of SxSx(Bx5+C) = 7x7x30, completing all predictions in a single forward pass [5]. This design enabled YOLOv1 to achieve an astonishing inference speed of 45 FPS - 9 times faster than Faster R-CNN - truly achieving real-time detection.

However, YOLOv1's problems are also apparent [5]: first, each grid cell can only predict 2 boxes and 1 class, resulting in poor detection of dense small objects (e.g., a flock of birds); second, bounding box localization accuracy is inferior to Faster R-CNN because it directly regresses coordinates rather than

predicting offsets from anchors; third, generalization to objects with uncommon aspect ratios is weak.

From YOLOv1 to YOLOv8, the YOLO series underwent eight generations of continuous iteration [6]: YOLOv2 introduced anchor box mechanisms and Batch Normalization (BN); YOLOv3 adopted multi-scale feature map prediction and a residual network backbone; YOLOv4 incorporated numerous training tricks (Bag-of-Freebies) and network optimization modules (Bag-of-Specials); YOLOv7 introduced the ELAN module and model reparameterization [6].

YOLOv8 [6] is the latest version of the YOLO series, representing the engineering-optimal solution for single-stage CNN detectors. It employs a CSPDarknet backbone, PAN-FPN feature fusion structure, Task-Aligned Assigner dynamic matching strategy, and has transitioned from anchor-based to anchor-free design [6]. YOLOv8-L achieves 52.9 mAP on the COCO dataset with inference speed far exceeding two-stage models, making it one of the most widely deployed detection models in industry [6].

What problems of Faster R-CNN did the YOLO series solve? The most critical point is speed - it proved that object detection can be done in real-time, providing viable solutions for latency-sensitive scenarios such as autonomous driving and video surveillance. However, the YOLO series (up to v8) still relies on NMS post-processing, and the dense prediction strategy leads to severe positive-negative sample imbalance, a problem that was not truly solved until RetinaNet [8].

(3) RetinaNet: Focal Loss Turns the Tables for Single-Stage Detectors

Before RetinaNet [8] appeared, there was a widely accepted 'common knowledge' in object detection: two-stage detectors have high accuracy but slow speed, while single-stage detectors are fast but less accurate. It was generally believed that the accuracy ceiling of single-stage detectors was due to insufficient network architecture. However, Lin et al. proposed a disruptive viewpoint: the fundamental reason for poor single-stage detector accuracy is not network architecture, but the class imbalance problem during training [8].

Specifically, single-stage detectors perform dense prediction on feature maps, producing approximately 100,000 candidate locations per image, of which the vast majority (over 99.9%) are background (negative samples), with very few positive samples actually containing objects [8]. Traditional cross-entropy loss treats every sample equally, causing the massive number of easily classified background samples to dominate gradient updates, preventing the model from learning anything useful.

RetinaNet's core innovation is Focal Loss [8]. It adds a modulating factor (1-p_t)^gamma before the standard cross-entropy, where p_t is the model's predicted probability for the correct class and gamma is the focusing parameter (default 2). When the model is already very confident about a sample (p_t close to 1), (1-p_t)^gamma approaches 0, drastically reducing that sample's contribution to the loss; for hard samples the model is uncertain about (small p_t), the modulating factor is close to 1, leaving the loss contribution unchanged [8].

In terms of network architecture, RetinaNet uses ResNet+FPN as the backbone. FPN fuses features from different levels through top-down pathways and lateral connections, preserving both high-resolution spatial information from shallow layers and high-level semantic information from deep layers [8]. Combined with Focal Loss, RetinaNet achieved 39.1 mAP on the COCO dataset, surpassing all contemporary two-stage detectors for the first time [8].

What problems did RetinaNet solve? First, it proved that the accuracy bottleneck of single-stage detectors lies in training strategy rather than network architecture; second, it doesn't require a region proposal generation step, enabling faster inference. However, RetinaNet still relies on anchor boxes and NMS post-processing, which remain pain points in dense occlusion scenarios [8].

(4) DETR: The Transformer Revolution - Abandoning Anchors and NMS

If Faster R-CNN and RetinaNet optimized within the CNN detection framework, then DETR [9] simply flipped the table and created an entirely new detection paradigm. In 2020, Carion et al. proposed DETR (Detection Transformer), the first to introduce Transformer self-attention mechanisms into object detection, achieving truly end-to-end detection - no anchor boxes, no NMS post-processing, no manually designed components [9].

DETR's core idea is remarkably elegant: treat object detection as a set prediction problem. It uses ResNet to extract image features, flattens them, adds positional encoding, and feeds them into a Transformer encoder for global self-attention computation. Then, a set of learnable Object Queries (default 100) interact with encoder features through a Transformer decoder, with each query directly outputting a prediction result (class + bounding box coordinates) [9]. During training, the Hungarian algorithm achieves one-to-one optimal matching between predicted and ground-truth boxes, eliminating the need for NMS deduplication [9].

What problems did DETR solve? First, it completely eliminated the design burden of anchor boxes; second, it eliminated NMS post-processing, avoiding incorrect removal of valid detection boxes in dense occlusion scenarios; third, the Transformer's global self-attention mechanism can capture long-range feature dependencies, providing significantly better robustness for large and occluded objects compared to CNNs [9].

However, DETR also has notable shortcomings: first, training convergence is extremely slow, requiring 500 epochs while Faster R-CNN only needs 12-36; second, self-attention computational complexity is O(N^2), resulting in enormous overhead for high-resolution feature maps; third, small object detection accuracy is notably weaker because the Transformer encoder operates only on single-scale feature maps [9]. These issues spawned subsequent improvements including Deformable DETR [10] and DINO [11].

(5) RT-DETR: Making Transformer Detectors Run in Real-Time

Although DETR-series models are advanced in accuracy and design philosophy, real-time performance has always been their Achilles' heel. Deformable DETR [10] reduced complexity from O(N^2) to O(NK), and DINO [11] accelerated convergence through denoising training, but their inference speeds still couldn't compete with the YOLO series. In 2023, RT-DETR [12] finally solved this problem - it is the first real-time DETR detector.

RT-DETR's core innovation is the Hybrid Encoder architecture [12]. Instead of feeding all features into a Transformer encoder for global self-attention like DETR, it adopts a smarter strategy: first using CNN for efficient intra-scale feature extraction on multi-scale feature maps, then using Transformer for cross-scale feature fusion between different scales [12]. This preserves the advantages of Transformer global modeling while drastically reducing computational overhead.

Additionally, RT-DETR introduces an uncertainty-minimized query selection strategy, selecting the highest-confidence features from encoder outputs as initial queries for the decoder, replacing DETR's randomly initialized Object Queries and accelerating decoder convergence [12]. On the decoder side, RT-DETR supports flexible accuracy-speed trade-off control by adjusting the number of decoder layers without retraining [12].

On the COCO dataset, RT-DETR-L achieves 53.0 mAP with 114 FPS inference speed (T4 GPU), simultaneously surpassing YOLOv8-L in both accuracy (52.9 mAP) and speed [12]. It solves the real-time bottleneck of the DETR series while inheriting end-to-end detection advantages without NMS.

(6) Summary of Inheritance Relationships and Technological Evolution

From a technological evolution perspective, these five papers form two parallel yet converging trajectories:

CNN Route: Faster R-CNN [3] (two-stage, high accuracy but slow) -> YOLOv1 [5] (single-stage, pioneered real-time detection but lower accuracy) -> YOLOv8 [6] (anchor-free, engineering-optimal solution) -> RetinaNet [8] (Focal Loss solved class imbalance, single-stage accuracy first surpassed two-stage). The core tension is 'accuracy vs. speed.'

Transformer Route: DETR [9] (end-to-end, no Anchor/NMS, but slow and weak on small objects) -> RT-DETR [12] (hybrid encoder, real-time, superior in both accuracy and speed). The core tension is 'global modeling capability vs. computational efficiency.'

The convergence point: RT-DETR inherits DETR's end-to-end detection paradigm while absorbing CNN's efficient feature extraction advantages, and surpasses YOLOv8-L in accuracy (53.0 vs. 52.9 mAP) [12], representing the latest direction in object detection technology.

III. Physical Meaning and Mapping Relationships of Input/Output Tensors

(1) Faster R-CNN Input/Output Tensor Analysis

1. Input Tensor

Faster R-CNN's input is an RGB image represented as a 3D tensor I in R^(3xHxW), where 3 represents the RGB channels, and H and W are the image height and width (typically with the shorter side scaled to 600 pixels) [3]. Each pixel value, after ImageNet mean normalization, has a value range of approximately [-2.0, 2.0]. Physical meaning: each element represents the light intensity at a specific spatial position and color channel.

2. Output Tensor

Faster R-CNN's output consists of three parts [3]:

(a) Bounding box coordinate tensor B in R^(Nx4): N is the number of detected objects, each bounding box represented by 4 values (x1, y1, x2, y2) - the pixel coordinates of the top-left and bottom-right corners. Value range: x1, x2 in [0, W], y1, y2 in [0, H]. Physical meaning: the spatial position and size of the object.

(b) Class probability tensor C in R^(Nx(K+1)): K is the number of object classes, +1 for the background class. Each row is Softmax-normalized, value range [0, 1], with all class probabilities summing to 1.

(c) Confidence score S in R^(N): the maximum class probability serves as the confidence score, value range [0, 1].

3. Input-to-Output Mapping Relationship

Faster R-CNN's mapping process can be summarized as 'feature extraction -> region proposal -> region classification and regression' in three steps [3]:

Step 1: The CNN backbone (e.g., VGG-16 or ResNet-50) performs hierarchical convolution operations on the input image to extract multi-level visual features, ultimately producing a feature map F in R^(C_f x H_f x W_f), where C_f is the number of feature channels (e.g., 512), H_f=H/16, W_f=W/16 [3].

Step 2: The RPN slides a 3x3 convolution window across the feature map, generating k anchor boxes (default k=9) at each position, predicting foreground/background scores and bounding box offsets. The RPN filters out approximately 300 high-quality candidate regions [3].

Step 3: ROI Pooling pools each candidate region's feature map area to a fixed size (e.g., 7x7), which is then fed into fully connected layers for K+1 class classification and bounding box refinement [3]. Finally, NMS removes overlapping detection boxes to produce the final output.

Overall mapping: {(b_i, c_i, s_i)} = NMS(Head(ROIPool(RPN(Backbone(I))))) [3].

(2) YOLOv1 Input/Output Tensor Analysis

1. Input Tensor

YOLOv1's input is an RGB image scaled to a fixed size: I in R^(3x448x448) [5]. Unlike Faster R-CNN, YOLOv1 requires a fixed input size of 448x448 because the final fully connected layers require fixed-dimension input. Pixel values are normalized to [0, 1].

2. Output Tensor

YOLOv1's output is a 3D tensor O in R^(SxSx(Bx5+C)) = R^(7x7x30) [5]:

(a) Bounding box parameters: each grid cell predicts B=2 bounding boxes, each containing 5 values (x, y, w, h, conf). x, y are offsets relative to the grid cell's top-left corner [0,1]; w, h are relative to the entire image [0,1]; conf = P(Object) x IoU(pred, truth) [0,1] [5].

(b) Class probabilities: each grid cell predicts C=20 conditional class probabilities P(Class_i|Object) [0,1]. Final class confidence = conf x P(Class_i|Object), combining localization and classification [5].

(c) Final output: after confidence threshold filtering and NMS, a total of 7x7x2=98 candidate boxes, far fewer than Faster R-CNN's approximately 16,650 anchors [5].

3. Input-to-Output Mapping Relationship

YOLOv1's mapping is extremely concise - 'one forward pass, direct regression' [5]:

Step 1: The input image passes through 24 convolutional layers and 2 fully connected layers, compressing the 3x448x448 input into a 1024x7x7 feature map [5].

Step 2: The feature map is flattened into a 4096-dimensional vector, then mapped to 7x7x30=1470 dimensions, reshaped into a 7x7x30 tensor [5].

Step 3: During inference, confidence threshold filtering and NMS are applied to the 98 candidate boxes [5].

Overall mapping: O = reshape(FC(Flatten(Conv(I))), (7,7,30)), with NMS yielding final results [5].

(3) RetinaNet Input/Output Tensor Analysis

1. Input Tensor

Same as Faster R-CNN: normalized RGB image I in R^(3xHxW), typically with shorter side scaled to 800 pixels [8].

2. Output Tensor

RetinaNet's output includes bounding boxes, class probabilities, and confidence scores [8], with one key difference:

(a) Bounding box coordinates B in R^(Nx4): same as Faster R-CNN.

(b) Class probabilities C in R^(NxK): no background class. RetinaNet uses Sigmoid activation independently for each class (rather than Softmax), each valued in [0, 1], not requiring probabilities to sum to 1 [8].

(c) Confidence score S in R^(N): the maximum across all class probabilities.

3. Input-to-Output Mapping Relationship

RetinaNet's mapping: 'feature extraction -> multi-scale dense prediction -> Focal Loss weighted training' [8]:

Step 1: ResNet backbone extracts multi-level features; FPN generates a multi-scale feature pyramid {P3, P4, P5, P6, P7}, with resolution decreasing from P3 (1/8) to P7 (1/128) [8].

Step 2: On each scale's feature map, classification and regression subnets perform dense predictions at each anchor position. Classification outputs KxA Sigmoid probabilities (A=9 anchors per position) [8].

Step 3: Training uses Focal Loss: FL(p_t) = -alpha_t(1-p_t)^gamma log(p_t), where alpha_t=0.25, gamma=2 [8].

Overall mapping: {(b_i, c_i, s_i)} = NMS(SubNet_cls(FPN(I)), SubNet_reg(FPN(I))) [8].

(4) DETR Input/Output Tensor Analysis

1. Input Tensor

Normalized RGB image I in R^(3xHxW), typically H=800, W=1066 [9]. DETR also requires a positional encoding tensor PE in R^(dxH_fxW_f), where d=256, H_f=H/32, W_f=W/32, generated using fixed sinusoidal/cosine functions [9]. The decoder input includes learnable Object Queries OQ in R^(N_q x d), N_q=100, d=256. Each query represents a 'detection slot' responsible for detecting one object or predicting 'no object' [9].

2. Output Tensor

DETR's output is a fixed-size prediction set containing N_q=100 results [9]:

(a) Bounding box coordinates B in R^(N_q x 4): normalized center coordinates and dimensions (cx, cy, w, h), all in [0, 1] (proportional to image dimensions) [9].

(b) Class probabilities C in R^(N_q x (K+1)): K object classes plus 1 'no object' class, Softmax-normalized [9].

(c) Final output: after filtering out 'no object' predictions, remaining predictions are detection results, requiring no NMS. Output count N <= N_q [9].

3. Input-to-Output Mapping Relationship

DETR's mapping: 'CNN feature extraction -> Transformer encoding -> set prediction decoding' [9]:

Step 1: ResNet-50 maps the input to F in R^(2048xH/32xW/32), reduced via 1x1 conv to F' in R^(256xH/32xW/32), flattened to Z_0 in R^(HW/1024 x 256), with positional encoding added [9].

Step 2: Transformer encoder (6 stacked self-attention layers) establishes global feature dependencies. Output: Z in R^(HW/1024 x 256) [9].

Step 3: Transformer decoder (6 layers) with self-attention and cross-attention sublayers. N_q Object Queries extract target information from encoder features [9].

Step 4: Two FFNs map decoder output to class probabilities and bounding box coordinates. Training uses Hungarian algorithm for one-to-one matching, with loss = classification (CE) + bbox (L1 + GIoU [25]) [9].

Overall mapping: {(b_i, c_i)} = FFN(Decoder(Encoder(Backbone(I) + PE), OQ)), no NMS needed [9].

(5) RT-DETR Input/Output Tensor Analysis

1. Input Tensor

Normalized RGB image I in R^(3x640x640) (fixed resolution) [12]. Unlike DETR, RT-DETR uses learnable positional encoding within the hybrid encoder rather than fixed sinusoidal encoding [12].

2. Output Tensor

RT-DETR's output format is consistent with DETR [12]:

(a) Bounding box coordinates B in R^(N_q x 4): normalized (cx, cy, w, h), value range [0, 1].

(b) Class probabilities C in R^(N_q x K): Sigmoid activation, each class predicted independently [12].

(c) Default N_q=300, with flexible accuracy-speed trade-off via decoder layer adjustment [12].

3. Input-to-Output Mapping Relationship

RT-DETR's mapping: 'multi-scale CNN feature extraction -> hybrid encoding -> query selection -> Transformer decoding' [12]:

Step 1: Backbone (ResNet-50 or HGNetv2) extracts multi-scale features {S3, S4, S5} at 1/8, 1/16, 1/32 [12].

Step 2: Hybrid encoder's AIFI module performs Transformer self-attention on S5 for global semantics; CCFM module efficiently fuses features across scales via CNN convolutions [12].

Step 3: Uncertainty-minimized query selection: select N_q highest-confidence features as initial decoder queries, replacing DETR's random initialization [12].

Step 4: Transformer decoder iteratively refines queries via self-attention and cross-attention [12].

Overall mapping: {(b_i, c_i)} = Decoder(QuerySelect(HybridEncoder(Backbone(I)))), no NMS needed [12].

(6) Comparative Summary of Input/Output Tensors

Model	Input Tensor	Output BBox	Output Class	Post-Proc
Faster R-CNN	I in R^(3xHxW) normalized RGB	B in R^(Nx4) absolute (x1,y1,x2,y2)	C in R^(Nx(K+1)) Softmax w/ bg class	NMS
YOLOv1	I in R^(3x448x448) fixed-size RGB	O in R^(7x7x30) norm (x,y,w,h,conf)	Cond. prob P(C|Obj) shared per grid	NMS
RetinaNet	I in R^(3xHxW) normalized RGB	B in R^(Nx4) absolute (x1,y1,x2,y2)	C in R^(NxK) Sigmoid, no bg class	NMS
DETR	I in R^(3xHxW) + PE + OQ	B in R^(Nqx4) norm (cx,cy,w,h)	C in R^(Nqx(K+1)) Softmax w/ null	None
RT-DETR	I in R^(3x640x640) fixed resolution	B in R^(Nqx4) norm (cx,cy,w,h)	C in R^(NqxK) Sigmoid, no bg class	None

Table 1: Comparative Summary of Input/Output Tensors Across Five Models

IV. Tensor Shape Transformation Derivations for Core Computational Modules

This chapter selects one core computational module from each of the five papers and uses pseudocode and textual derivations to describe the tensor shape transformation process within each module, clearly annotating the dimensional changes and computational purpose after each step.

(1) Faster R-CNN Core Module: Region Proposal Network (RPN)

The RPN is Faster R-CNN's core innovation, responsible for generating high-quality candidate regions from feature maps [3]. Below, we derive the tensor transformation process using input image size 3x600x800 and VGG-16 as the backbone.

Input

Feature map from the CNN backbone: F in R^(512x37x50), where 512 is the channel count, 37=600/16, 50=800/16 (VGG-16 downsampling rate is 16) [3].

Step 1: 3x3 Sliding Window Convolution

```
# Input: F in R^(512x37x50)
# Op: 3x3 conv, padding=1, output channels 512
F_rpn = Conv2d(F, kernel=3x3, out=512, pad=1) # -> R^(512x37x50)
F_rpn = ReLU(F_rpn) # -> R^(512x37x50)
```

Purpose: Extract local features from the 3x3 neighborhood at each spatial position. padding=1 ensures output spatial dimensions remain unchanged [3].

Step 2: Classification Branch (Foreground/Background)

```
# Input: F_rpn in R^(512x37x50)
# Op: 1x1 conv, output channels 2k=18 (k=9 anchors, 2 classes each)
cls_score = Conv2d(F_rpn, kernel=1x1, out=18) # -> R^(18x37x50)
cls_score = reshape(cls_score, (2, 9x37x50)) # -> R^(2x16650)
cls_prob = Softmax(cls_score, dim=0) # -> R^(2x16650)
```

Purpose: Predict foreground/background probabilities for each of 37x50x9=16,650 anchors [3].

Step 3: Regression Branch (Bounding Box Offsets)

```
# Input: F_rpn in R^(512x37x50)
# Op: 1x1 conv, output channels 4k=36 (k=9 anchors, 4 offsets each)
bbox_delta = Conv2d(F_rpn, kernel=1x1, out=36) # -> R^(36x37x50)
bbox_delta = reshape(bbox_delta, (16650, 4)) # -> R^(16650x4)
```

Purpose: Predict 4 bounding box offsets (dx, dy, dw, dh) for each anchor to refine preset anchor boxes [3].

Step 4: Candidate Region Generation

```
# Input: cls_prob in R^(2x16650), bbox_delta in R^(16650x4)
proposals = apply_delta(anchors, bbox_delta) # -> R^(16650x4)
proposals = filter_by_score(proposals, cls_prob) # -> R^(~6000x4)
proposals = NMS(proposals, threshold=0.7) # -> R^(~300x4)
```

Purpose: Filter approximately 300 high-quality candidate regions from 16,650 anchors for subsequent ROI Pooling and classification-regression head [3].

(2) YOLOv1 Core Module: Grid Prediction Mechanism

The grid prediction mechanism is YOLOv1's most pioneering design, transforming object detection from a 'candidate region classification' problem into a 'spatial grid regression' problem [5]. Below we derive using input 3x448x448, S=7, B=2, C=20.

Input

Feature vector after flattening and FC layers: F in R^(4096) [5].

Step 1: FC Layer Mapping to Grid Output

```
# Input: F in R^(4096)
# Op: FC layer, output dim SxSx(Bx5+C) = 7x7x30 = 1470
O_flat = FC(F, out=1470) # -> R^(1470)
O = reshape(O_flat, (7, 7, 30)) # -> R^(7x7x30)
```

Purpose: Map global feature vector to spatially gridded prediction tensor [5].

Step 2: Parse Each Grid Cell's Predictions

```
# Input: O in R^(7x7x30)
# Box1: (x1,y1,w1,h1,conf1) = O[:,:,0:5] # -> R^(7x7x5)
# Box2: (x2,y2,w2,h2,conf2) = O[:,:,5:10] # -> R^(7x7x5)
# ClassProb: P(C_i|Obj) = O[:,:,10:30] # -> R^(7x7x20)
```

Purpose: Decode the 30-dim vector into detection parameters with physical meaning [5].

Step 3: Compute Final Class Confidence

```
# Input: conf in R^(7x7x2), ClassProb in R^(7x7x20)
conf1 = O[:,:,4:5] # -> R^(7x7x1)
score1 = conf1 * ClassProb # -> R^(7x7x20) broadcast
conf2 = O[:,:,9:10] # -> R^(7x7x1)
score2 = conf2 * ClassProb # -> R^(7x7x20) broadcast
all_scores = stack(score1, score2) # -> R^(7x7x2x20)
all_scores = reshape(all_scores, (98, 20)) # -> R^(98x20)
```

Purpose: Final score = P(Object) x IoU x P(Class|Object), encoding both class identity and localization accuracy [5]. Each of 98 candidate boxes has a score for each of 20 classes.

Step 4: NMS Post-Processing

```
# Input: all_boxes in R^(98x4), all_scores in R^(98x20)
for each class k in 1..20:
scores_k = all_scores[:, k] # -> R^(98)
keep = NMS(all_boxes, scores_k, thr=0.5) # -> R^(~N_k)
```

Purpose: Remove duplicate detection boxes per class. Final output is typically single digits to a few dozen [5].

(3) RetinaNet Core Module: Focal Loss Computation

Focal Loss is RetinaNet's core innovation, solving extreme positive-negative sample imbalance [8]. Below we derive using classification output from FPN level P3.

Input

Raw logits Z in R^(AxKxH3xW3), where A=9 anchors, K=80 COCO classes, H3=100, W3=128 (assuming input 800x1024) [8]. Ground-truth labels Y in R^(AxKxH3xW3), values 0 or 1 (one-hot).

Step 1: Sigmoid Activation

```
# Input: Z in R^(9x80x100x128)
P = Sigmoid(Z) # -> R^(9x80x100x128)
# Each element p in (0, 1)
```

Purpose: Map raw logits to (0,1) probability space. Sigmoid (not Softmax) makes each class independent [8].

Step 2: Compute p_t

```
# Input: P, Y in R^(9x80x100x128)
p_t = Y * P + (1 - Y) * (1 - P) # -> R^(9x80x100x128)
# Y=1: p_t=P (positive sample); Y=0: p_t=1-P (negative sample)
```

Purpose: Unify probability representation. Larger p_t = easier sample; smaller p_t = harder sample [8].

Step 3: Compute Modulating Factor

```
# Input: p_t in R^(9x80x100x128)
mod_factor = (1 - p_t) ** 2 # -> R^(9x80x100x128)
# p_t->1 (easy): mod_factor->0, loss suppressed
# p_t->0 (hard): mod_factor->1, loss unchanged
```

Purpose: Core of Focal Loss. Dynamically reduces easy sample loss weight, focusing training on hard samples. With gamma=2, an easy sample with p_t=0.9 has its loss weight reduced by 100x [8].

Step 4: Compute Final Focal Loss

```
# Input: p_t, mod_factor in R^(9x80x100x128)
alpha_t = Y * 0.25 + (1 - Y) * 0.75 # -> R^(9x80x100x128)
CE = -log(p_t) # -> R^(9x80x100x128)
FL = alpha_t * mod_factor * CE # -> R^(9x80x100x128)
loss = FL.sum() / num_positive_anchors # -> scalar R^(1)
```

Purpose: alpha_t balances positive (0.25) and negative (0.75) sample contributions. Final loss normalized by positive anchor count yields a scalar for backpropagation [8].

(4) DETR Core Module: Multi-Head Self-Attention (MHSA)

MHSA is the core computational module of the Transformer encoder, establishing global dependency relationships among all positions [9]. Let d=256, h=8 heads, sequence length L=H_f x W_f=25x34=850.

Input

Encoder input sequence X in R^(Lxd) = R^(850x256), each row a d-dim feature vector with positional encoding added [9].

Step 1: Linear Projections to Generate Q, K, V

```
# Input: X in R^(850x256)
Q = X @ W_Q # W_Q in R^(256x256), Q in R^(850x256)
K = X @ W_K # W_K in R^(256x256), K in R^(850x256)
V = X @ W_V # W_V in R^(256x256), V in R^(850x256)
```

Purpose: Q='what am I looking for', K='what do I have', V='what can I provide'. Projection matrices are learnable parameters [9].

Step 2: Split into Multiple Heads

```
# Input: Q, K, V in R^(850x256)
# Split into h=8 heads, each dim d_k = 256/8 = 32
Q = reshape(Q, (850,8,32)).transpose(0,1) # -> R^(8x850x32)
K = reshape(K, (850,8,32)).transpose(0,1) # -> R^(8x850x32)
V = reshape(V, (850,8,32)).transpose(0,1) # -> R^(8x850x32)
```

Purpose: Multi-head mechanism allows learning different attention patterns in different subspaces [9].

Step 3: Scaled Dot-Product Attention

```
# Input: Q, K in R^(8x850x32)
attn = (Q @ K.T(-2,-1)) / sqrt(32) # -> R^(8x850x850)
attn = Softmax(attn, dim=-1) # -> R^(8x850x850)
# attn[h][i][j] = attention weight from pos i to pos j in head h
```

Purpose: Compute pairwise correlations. The 850x850 attention matrix means every position connects with all 850 positions - this is 'global self-attention' [9]. Complexity O(L^2 d_k) = O(850^2 x 32) is the fundamental reason for DETR's high computational overhead on high-resolution feature maps.

Step 4: Weighted Aggregation and Multi-Head Concatenation

```
# Input: attn in R^(8x850x850), V in R^(8x850x32)
attn_out = attn @ V # -> R^(8x850x32)
attn_out = attn_out.transpose(0,1) # -> R^(850x8x32)
attn_out = reshape(attn_out, (850, 256)) # -> R^(850x256)
output = attn_out @ W_O # W_O in R^(256x256) -> R^(850x256)
```

Purpose: Each position's output is a weighted sum of all Values. After concatenating 8 heads and linear projection, output dimension matches input (850x256), enabling stacking of multiple encoder layers [9].

(5) RT-DETR Core Module: Hybrid Encoder

The Hybrid Encoder is RT-DETR's core innovation, using AIFI and CCFM sub-modules to maintain global modeling while drastically reducing computational overhead [12]. Below we derive using input 640x640 and ResNet-50.

Input

Multi-scale feature maps from the backbone [12]:

```
S3 in R^(256x80x80) # 1/8 res, rich spatial details, small objects
S4 in R^(256x40x40) # 1/16 res, moderate semantics
S5 in R^(256x20x20) # 1/32 res, rich high-level semantics, large objects
```

Note: Channel dimensions unified to 256 via 1x1 convolution.

Step 1: AIFI - Intra-scale Self-Attention (on S5 only)

```
# Input: S5 in R^(256x20x20)
S5_flat = reshape(S5, (256,400)).T # -> R^(400x256)
S5_flat = S5_flat + PosEmbed(400, 256) # -> R^(400x256)
S5_attn = MHSA(S5_flat, heads=8) # -> R^(400x256)
S5_out = FFN(S5_attn) # -> R^(400x256)
S5_out = reshape(S5_out.T, (256,20,20)) # -> R^(256x20x20)
```

Purpose: Transformer self-attention only on highest-level S5 for global semantic dependencies [12]. Key insight: S5's sequence length is only 400 (20x20), with complexity O(400^2) being only ~22% of DETR's - the key design enabling real-time performance. S3 (6,400 positions) and S4 (1,600) would be prohibitively expensive.

Step 2: CCFM - Cross-scale Feature Fusion

CCFM adopts a FPN-like bidirectional pathway [12]:

```
# ---- Top-down pathway ----
S5_up = Upsample(S5_out, scale=2) # -> R^(256x40x40)
P4 = RepBlock(Concat(S5_up, S4)) # Cat->R^(512x40x40)
# Rep->R^(256x40x40)
P4_up = Upsample(P4, scale=2) # -> R^(256x80x80)
P3 = RepBlock(Concat(P4_up, S3)) # Cat->R^(512x80x80)
# Rep->R^(256x80x80)
# ---- Bottom-up pathway ----
P3_down = Conv2d(P3, stride=2) # -> R^(256x40x40)
P4_out = RepBlock(Concat(P3_down, P4)) # Cat->R^(512x40x40)
# Rep->R^(256x40x40)
P4_down = Conv2d(P4_out, stride=2) # -> R^(256x20x20)
P5_out = RepBlock(Concat(P4_down, S5_out)) # Cat->R^(512x20x20)
# Rep->R^(256x20x20)
```

Purpose: Bidirectional fusion ensures each scale contains both high-resolution spatial details and high-level semantic information [12]. RepBlock uses multi-branch training, fused to single conv at inference.

Output

```
# Hybrid encoder final output:
P3_out in R^(256x80x80) # Small object features
P4_out in R^(256x40x40) # Medium object features
```

```
P5_out in R^(256x20x20) # Large object features
```

These three feature maps are flattened and concatenated into a unified sequence, fed into the uncertainty-minimized query selection module to select Top-N_q features as initial decoder queries [12].

V. Conclusion and Outlook

This report selected five of the most representative papers - Faster R-CNN [3], YOLOv1/v8 [5][6], RetinaNet [8], DETR [9], and RT-DETR [12] - from the 30 references in the Phase 1 review, and completed an in-depth analysis along three dimensions.

Regarding algorithm strengths/weaknesses and inheritance relationships, the five papers form a clear technological evolution trajectory: Faster R-CNN established the region-based two-stage detection paradigm but was limited by speed and anchor design; YOLOv1 pioneered single-stage real-time detection, while YOLOv8 represents the engineering-optimal solution after eight generations; RetinaNet solved class imbalance through Focal Loss, achieving single-stage accuracy surpassing two-stage for the first time; DETR pioneered Transformer-based end-to-end set prediction, eliminating anchors and NMS but facing slow convergence and high computational overhead; RT-DETR solved the real-time bottleneck through its hybrid encoder architecture, achieving state-of-the-art in both accuracy and speed [3][5][6][8][9][12].

Regarding input/output tensor analysis, all five models take normalized RGB image tensors as input, but exhibit significant differences in output representation: YOLOv1 employs a unique gridded output tensor; CNN models use absolute pixel coordinates with Softmax/Sigmoid classification and NMS; Transformer models use normalized coordinates and set prediction without NMS. These differences reflect the paradigm shift from 'dense prediction + post-processing' to 'sparse set prediction.'

Regarding core module tensor derivations, the RPN achieves efficient candidate region generation through sliding window convolution; YOLOv1's grid prediction mechanism directly maps global features to spatially structured detection output; Focal Loss dynamically adjusts sample weights through the modulating factor; MHSA establishes feature dependencies among all positions through the global attention matrix with O(L^2) complexity; the Hybrid Encoder reduces complexity by approximately 78% through applying self-attention only to high-level features while processing low-level features with CNN [3][5][8][9][12].

Looking ahead, object detection technology is advancing toward higher accuracy, faster speed, and stronger generalization. The CNN+Transformer hybrid architecture represented by RT-DETR may become the mainstream trend. Combined with knowledge distillation [22], neural architecture search [22], multi-modal fusion [13], and other techniques, it is expected to further break through current performance bottlenecks and drive deeper applications in autonomous driving, remote sensing, intelligent security, and other fields [28].

References

[1] Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C]//CVPR. 2014: 580-587.

[2] Girshick R. Fast R-CNN[C]//ICCV. 2015: 1440-1448.

[3] Ren S, He K, Girshick R, et al. Faster R-CNN: Towards real-time object detection with region proposal networks[J]. NeurIPS, 2015, 28.

[4] He K, Gkioxari G, Dollar P, et al. Mask R-CNN[C]//ICCV. 2017: 2961-2969.

[5] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]//CVPR. 2016: 779-788.

[6] Jocher G, Chaurasia A, Qiu J, et al. YOLOv8: Ultralytics YOLO[EB/OL]. https://github.com/ultralytics/ultralytics, 2023.

[7] Liu W, Anguelov D, Erhan D, et al. SSD: Single shot multibox detector[C]//ECCV. 2016: 21-37.

[8] Lin T Y, Goyal P, Girshick R, et al. Focal loss for dense object detection[C]//ICCV. 2017: 2980-2988.

[9] Carion N, Massa F, Synnaeve G, et al. End-to-end object detection with transformers[C]//ECCV. 2020: 213-229.

[10] Zhu X, Su W, Lu L, et al. Deformable DETR: Deformable transformers for end-to-end object detection[J]. NeurIPS, 2020, 33: 9994-10005.

[11] Zhang H, Wen J, Bian J, et al. DINO: DETR with improved deformation and online hard example mining[C]//CVPR. 2022: 12413-12422.

[12] Zhou X, Wang D, Krahenbuhl P. RT-DETR: Real-time DETR for object detection[J]. arXiv:2304.08069, 2023.

[13] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16x16 words: Transformers for image recognition at scale[J]. arXiv:2010.11929, 2020.

[14] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[J]. NeurIPS, 2014, 27.

[15] Tang H, Zhang Y, Liu X, et al. GAN-Det: Detecting objects with generative adversarial networks[C]//AAAI. 2018, 32(1).

[16] Zhu J Y, Park T, Isola P, et al. Unpaired image-to-image translation using cycle-consistent adversarial networks[C]//ICCV. 2017: 2223-2232.

[17] Miyato T, Kataoka T, Koyama M, et al. Spectral normalization for generative adversarial networks[J]. arXiv:1802.05957, 2018.

[18] Howard A G, Zhu M, Chen B, et al. MobileNets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv:1704.04861, 2017.

[19] Howard A G, Sandler M, Chu G, et al. Searching for mobilenetv3[C]//ICCV. 2019: 1314-1324.

[20] Zhang X, Zhou X, Lin M, et al. ShuffleNet: An extremely efficient convolutional neural network for mobile devices[C]//CVPR. 2018: 6848-6856.

[21] Han K, Wang Y, Tian Q, et al. GhostNet: More features from cheap operations[C]//CVPR. 2020: 1580-1589.

[22] Rabinovitch A, Mirza M, Gong B, et al. YOLO-NAS: Neural architecture search for real-time object detection[EB/OL]. https://github.com/Deci-AI/super-gradients, 2023.

[23] Kingma D P, Dhariwal P. Glow: Generative flow with invertible 1x1 convolutions[C]//NeurIPS. 2018, 31.

[24] Everingham M, Van Gool L, Williams C K I, et al. The PASCAL visual object classes (VOC) challenge[J]. IJCV, 2010, 88(2): 303-338.

[25] Rezatofighi H, Tsoi N, Gwak J Y, et al. Generalized intersection over union: A metric and a loss for bounding box regression[C]//CVPR. 2019: 658-666.

[26] Zheng Z, Wang P, Liu W, et al. Distance-IoU loss: Faster and better learning for bounding box regression[C]//AAAI. 2020, 34(07): 12993-13000.

[27] Wang Z, Bovik A C, Sheikh H R, et al. Image quality assessment: From error visibility to structural similarity[J]. IEEE TIP, 2004, 13(4): 600-612.

[28] Trigka M, Dritsas E, Moustakos K. A Comprehensive Survey of Machine Learning Techniques and Models for Object Detection[J]. Sensors, 2022, 22(18): 6988.

[29] Chen X, Li G, Zhang Y. Spatial-channel attention network for small object detection in remote sensing images[J]. Chinese Journal of Computers, 2021, 44(5): 921-936.

[30] Lim B, Son S, Kim H, et al. Enhanced deep residual networks for single image super-resolution[C]//CVPR Workshops. 2017: 136-144.