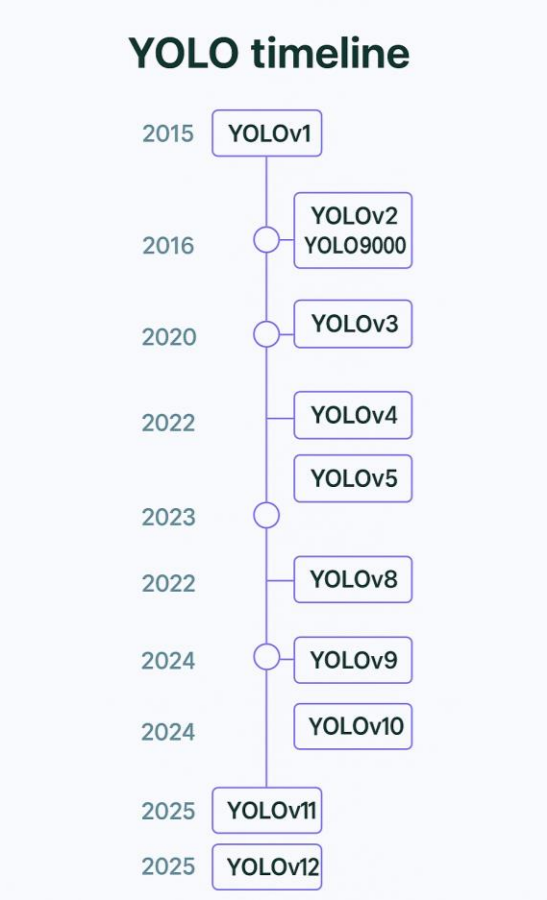
Object Detection Models

Contents

1. One-Stage, Anchor-Based Detectors
   - YOLO Family
   - EfficientDet Series
   - SSD & Variants
   - RetinaNet
2. Two-Stage Detectors
   - Faster R-CNN
   - Mask R-CNN
   - Cascade R-CNN
3. Transformer-Based Detectors (DETR Variants)
   - DETR
   - DINO
   - Grounding DINO
4. Vision–Language (Multi-Modal) Models
   - CLIP

YOLO Family



What Is YOLO

"You Only Look Once" (YOLO) proposes using an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once, rather than repurposing classifiers for detection.

- **Single Iteration, End-to-End**

YOLO performs all its predictions with a single fully connected layer, requiring just one forward pass over the image to produce detections.

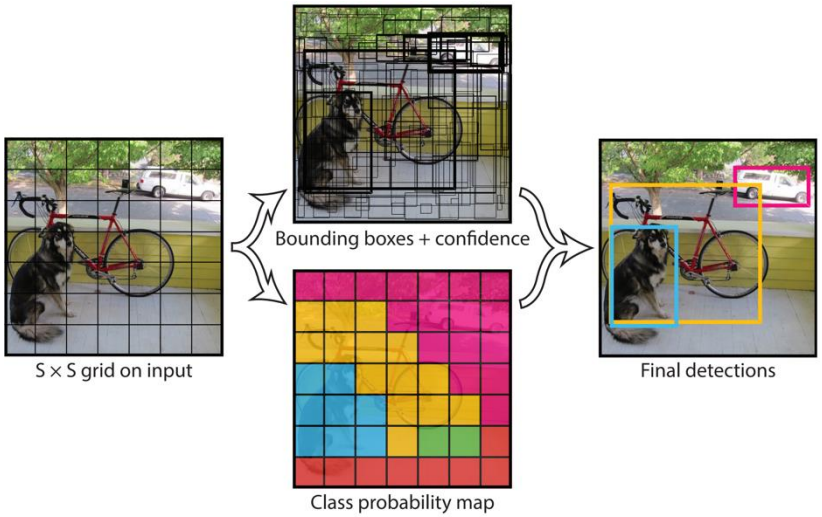
- **Fast Inference for Real-Time**

By unifying the entire detection pipeline into one network, YOLO processes an image in a single pass, enabling real-time speeds

- **Global Reasoning**

By seeing the full image during both training and inference, YOLO encodes contextual information—reducing background false positives compared to sliding-window or proposal-based methods.

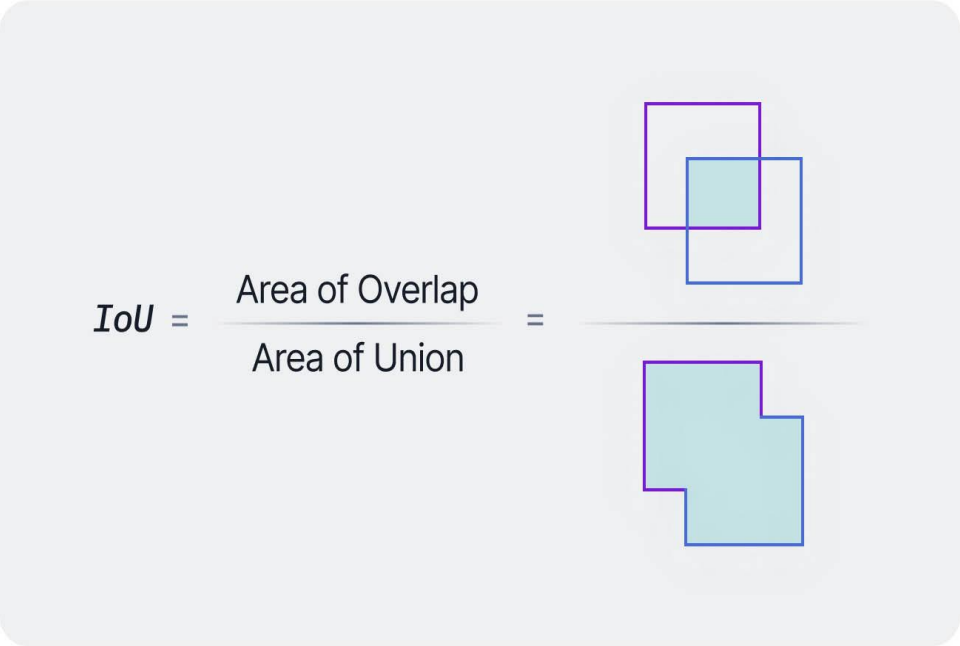
YOLO Working



How does YOLO work ?

- The YOLO algorithm works by dividing the image into N grids, each having an equal dimensional region of SxS.
- Each of these N grids is responsible for the detection and localization of the object it contains.
- Correspondingly, these grids predicts B bounding box coordinates relatives to their cell coordinates, along with the object label and probability of the object being present in the cell.

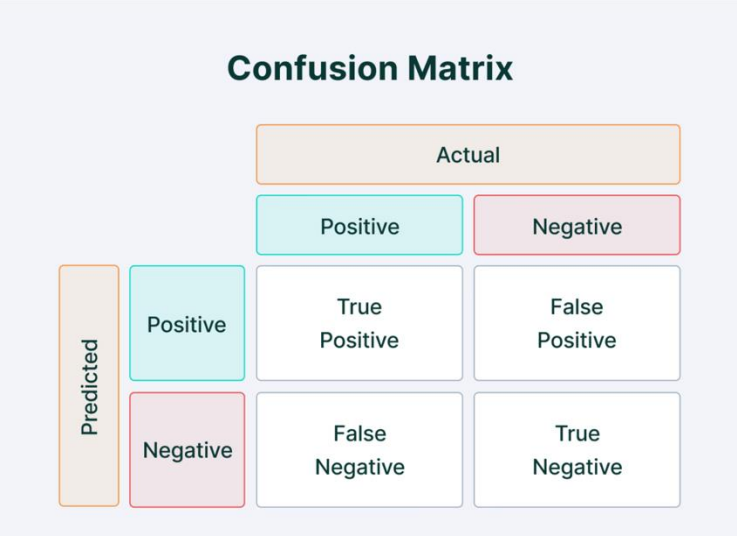
Object detection models performance evaluation metrics



Intersection of Union

- Intersection over Union is a popular metric to measure localization accuracy an
- To calculate the IoU between the predicted and the ground truth bounding boxes, we first take the intersecting area between the two corresponding bounding boxes for the same object. Following this, we calculate the total area covered by the two bounding boxes— also known as the “Union” and the area of overlap between them called the “Intersection.”
- The intersection divided by the Union gives us the ratio of the overlap to the total area, providing a good estimate of how close the prediction bounding box is to the original bounding box.

Average Precision



- Average Precision (AP) is calculated as the area under a precision vs. recall curve for a set of predictions.
- Recall is calculated as the ratio of the total predictions made by the model under a class with a total of existing labels for the class. Precision refers to the ratio of true positives with respect to the total predictions made by the model.
- Recall and precision offer a trade-off that is graphically represented into a curve by varying the classification threshold. The area under this precision vs. recall curve gives us the Average Precision per class for the model. The average of this value, taken over all classes, is called mean Average Precision (mAP).

YOLO Architecture

- **Key Features of YOLOv1 Architecture**
- **Single-pass detection:** The whole image is processed in one forward pass.
- **1×1 Convolutions:** Reduce computation and add non-linearity (a technique from GoogLeNet).
- **Unified detection:** Predicts class probabilities and bounding boxes together.
- **Pretraining:** The conv layers are pretrained on ImageNet at 224×224, then fine-tuned on 448×448 for detection.

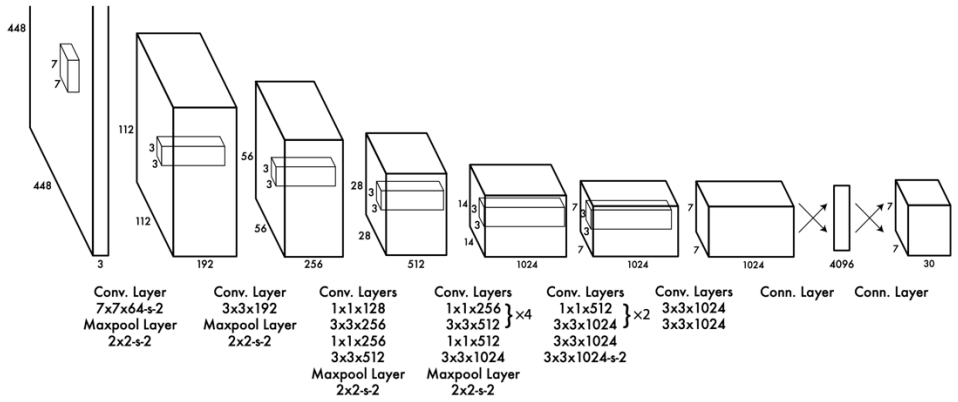
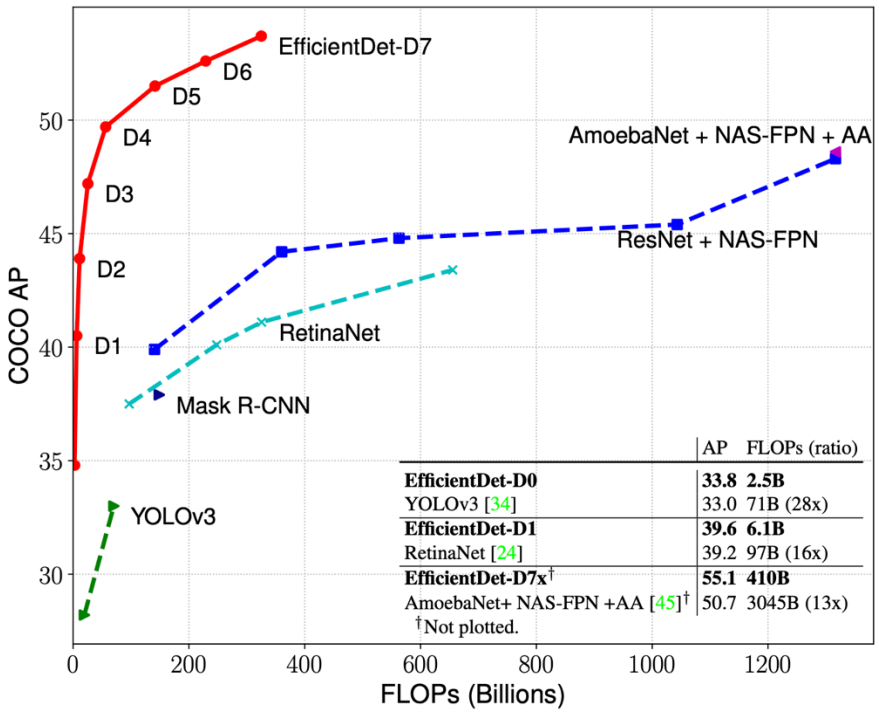


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Stage	Size	What Happens
Input	448×448×3	Image input
Initial Conv	7×7, stride 2	Broad feature extraction
Pooling	2×2, stride 2	Downsampling
Conv Blocks	1×1, 3×3, maxpool	Repeated for depth and spatial size
Deep Conv	7×7×1024	Complex, high-level features
Fully Connected	4096	Dense representation
Output	7×7×30	Grid of bounding boxes + classes

EfficientDet



	AP	FLOPs (ratio)
EfficientDet-D0	**33.8**	**2.5B**
YOLOv3 [34]	33.0	71B (28x)
EfficientDet-D1	**39.6**	**6.1B**
RetinaNet [24]	39.2	97B (16x)
EfficientDet-D7x†	**55.1**	**410B**
AmoebaNet+ NAS-FPN +AA [45]†	50.7	3045B (13x)

†Not plotted.

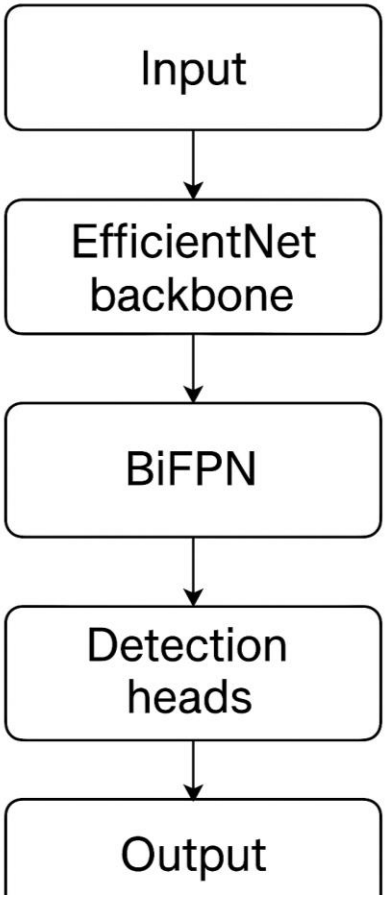
Figure 1: **Model FLOPs vs. COCO accuracy –** All numbers are for single-model single-scale. Our EfficientDet achieves new state-of-the-art 55.1% COCO AP with much fewer parameters and FLOPs than previous detectors. More studies on different backbones and FPN/NAS-FPN/BiFPN

- EfficientDet is a family of object detection models designed for both speed and accuracy, making it suitable for real-world applications on everything from mobile devices to cloud servers. By rethinking how detectors handle multi-scale features and how models are scaled up, EfficientDet delivers state-of-the-art accuracy using far fewer parameters and FLOPs than previous object detectors.

Key Points:

- Achieves strong detection results with much smaller model size and computation.
- Introduces optimizations that address the limitations of large, slow detectors like NAS-FPN and RetinaNet.
- Enables efficient deployment in resource-constrained environments.

EfficientDet- Workflow



EfficientDet follows a one-stage detection paradigm, meaning it predicts object classes and bounding boxes in a single forward pass through the network.

How EfficientDet Works:

- Uses EfficientNet as the backbone to extract rich feature maps from the input image with minimal computation.
- Features from multiple scales are fused together using a Bi-directional Feature Pyramid Network (BiFPN), which combines information from different resolutions more effectively through learnable weights.
- Employs compound scaling to jointly scale the model's resolution, depth, and width, enabling the creation of small, fast models or larger, more accurate ones as needed.
- The final detection heads simultaneously output bounding box coordinates and class probabilities for every relevant region.

EfficientDet Architecture

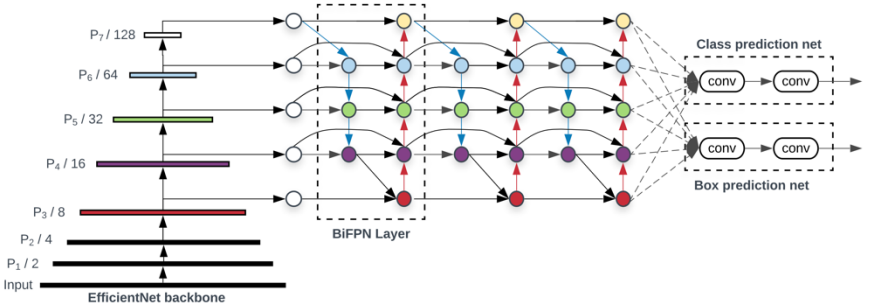


Figure 3: **EfficientDet architecture –** It employs EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

Table 1: **Scaling configs for EfficientDet D0-D6 –** ϕ is the compound coefficient that controls all other scaling dimensions; *BiFPN, box/class net, and input size* are scaled

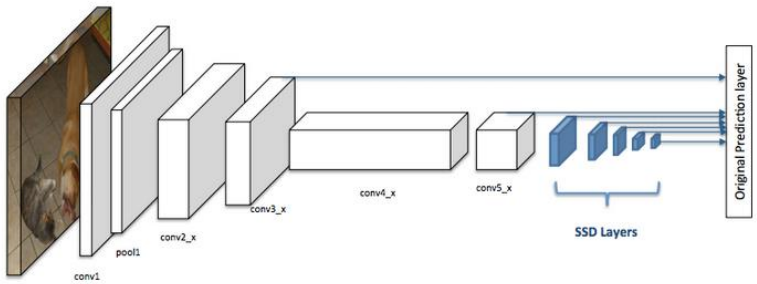
The architecture of EfficientDet is streamlined for efficiency and scalability, while still supporting complex detection tasks.

Input: The image is fed into the EfficientNet backbone for feature extraction.

Multi-scale Features: Feature maps from different levels (P3–P7) are combined using the BiFPN, which allows for repeated top-down and bottom-up fusion and assigns importance to each feature using learnable weights.

- **Prediction Heads**: Shared class and box heads make predictions from each fused feature map, producing final bounding boxes and class labels.
- **Compound Scaling**: The entire pipeline is scalable, so you can choose the right EfficientDet model size for your hardware and accuracy needs.

Single Shot MultiBox Detector (SSD)

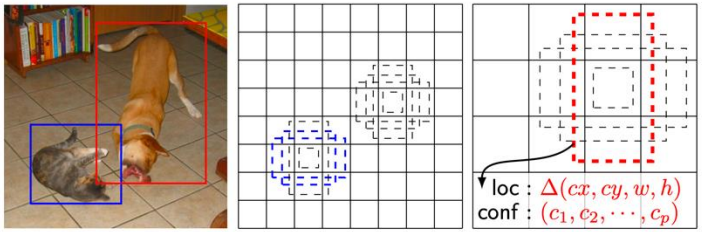


- SSD is a deep learning object detection model that streamlines the detection pipeline by eliminating the need for separate region proposal and feature resampling stages.
- It predicts bounding boxes and class scores directly from images in a single forward pass, making it both accurate and extremely fast, suitable for real-time applications.

Key points:

- Directly predicts categories and bounding box offsets from multiple feature maps.
- Efficiently handles objects at different scales and aspect ratios.
- Outperforms earlier real-time detectors like YOLO in both speed and accuracy.

SSD Workflow

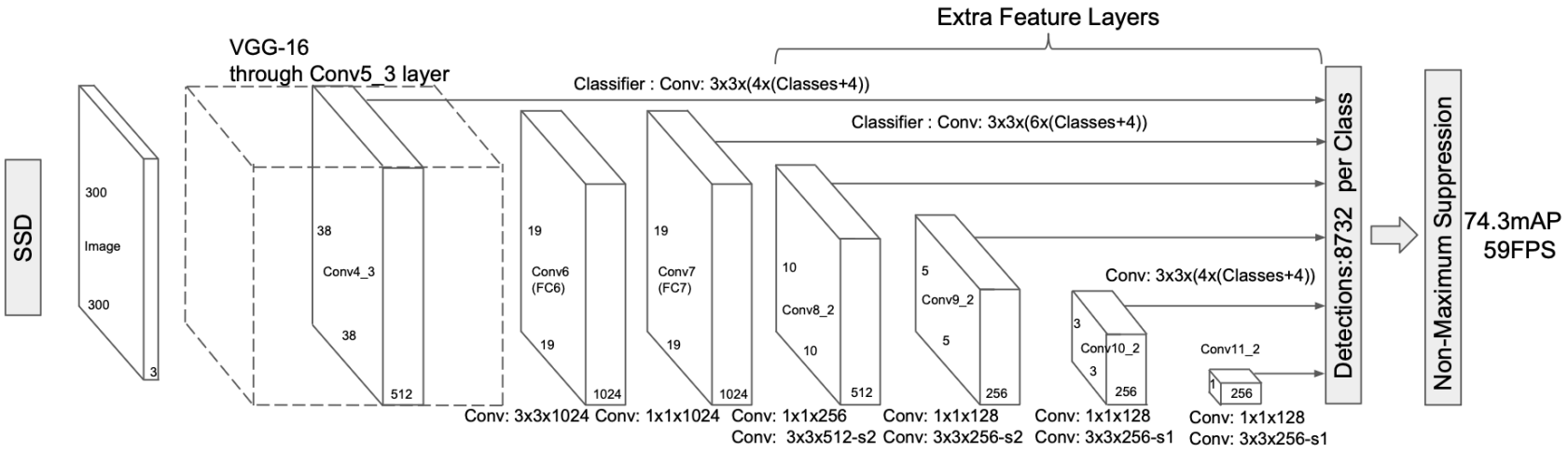


(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Fig. 1: **SSD framework.** (a) SSD only needs an input image and ground truth boxes for each object during training. In a convolutional fashion, we evaluate a small set (e.g. 4) of default boxes of different aspect ratios at each location in several feature maps with different scales (e.g. 8×8 and 4×4 in (b) and (c)). For each default box, we predict both the shape offsets and the confidences for all object categories ($(c_1, c_2, \cdots, c_p)$). At training time, we first match these default boxes to the ground truth boxes. For example, we have matched two default boxes with the cat and one with the dog, which are treated as positives and the rest as negatives. The model loss is a weighted sum between localization loss (e.g. Smooth L1 [6]) and confidence loss (e.g. Softmax).

The SSD workflow revolves around discretizing the output space of bounding boxes into a set of default boxes at various aspect ratios and scales for each feature map location.

- The input image is processed by a base CNN (typically VGG-16, truncated before classification).
- Multiple additional convolutional layers generate feature maps at decreasing spatial sizes.
- At each location in these maps, a fixed set of default bounding boxes (of different shapes) is considered.
- For each default box, the network predicts:
- Bounding box offsets (location refinement).
- Confidence scores for each object category.
- Non-maximum suppression (NMS) is applied to produce the final set of detections.

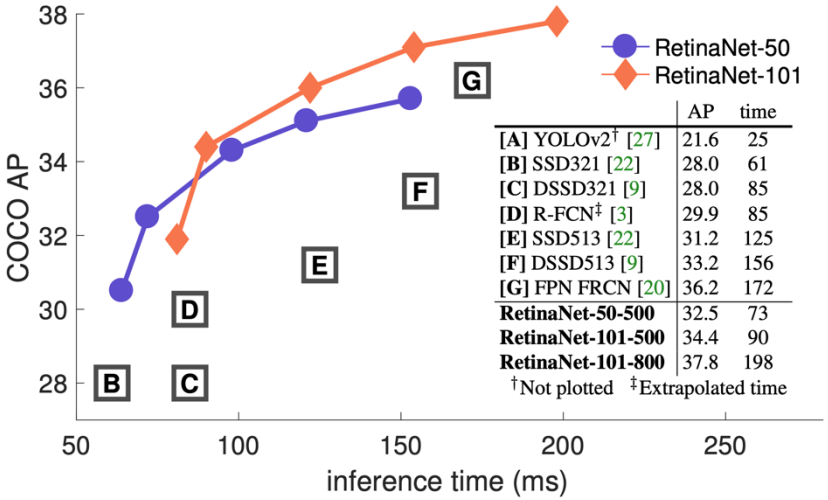


SSD's architecture is built on a feed-forward convolutional network augmented by additional layers for multi-scale detection

SSD Architecture

- Base Network: Truncated VGG-16 is commonly used for initial feature extraction.
- Extra Feature Layers: Additional convolutional layers produce lower-resolution feature maps for detecting larger objects.
- Multi-scale Detection: Predictions are made from multiple layers (conv4_3, conv7, conv8_2, etc.), each responsible for objects at different sizes.
- Convolutional Predictors: Small 3×3 convolutional filters applied to each feature map location for predicting class scores and bounding box offsets.
- Default Boxes: Each feature map cell uses multiple default boxes with various aspect ratios and scales.

RetinaNet



	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN‡ [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

†Not plotted ‡Extrapolated time

Figure 2. Speed (ms) versus accuracy (AP) on COCO `test-dev`. Enabled by the focal loss, our simple one-stage *RetinaNet* detector outperforms all previous one-stage and two-stage detectors, including the best reported Faster R-CNN [28] system from [20]. We show variants of RetinaNet with ResNet-50-FPN (blue circles) and ResNet-101-FPN (orange diamonds) at five scales (400-800 pixels). Ignoring the low-accuracy regime (AP<25), RetinaNet forms an upper envelope of all current detectors, and an improved variant (not shown) achieves 40.8 AP. Details are given in §5.

- RetinaNet is a one-stage object detector that achieves both high accuracy and fast inference by addressing the extreme foreground-background class imbalance in dense detection. It introduces the Focal Loss to focus learning on hard, misclassified examples, enabling one-stage detectors to rival or surpass two-stage detectors like Faster R-CNN in accuracy.

Key points:

- Closes the accuracy gap between one-stage (fast) and two-stage (accurate) detectors.
- Well-suited for real-time applications due to simple and efficient design.

RetinaNet Workflow

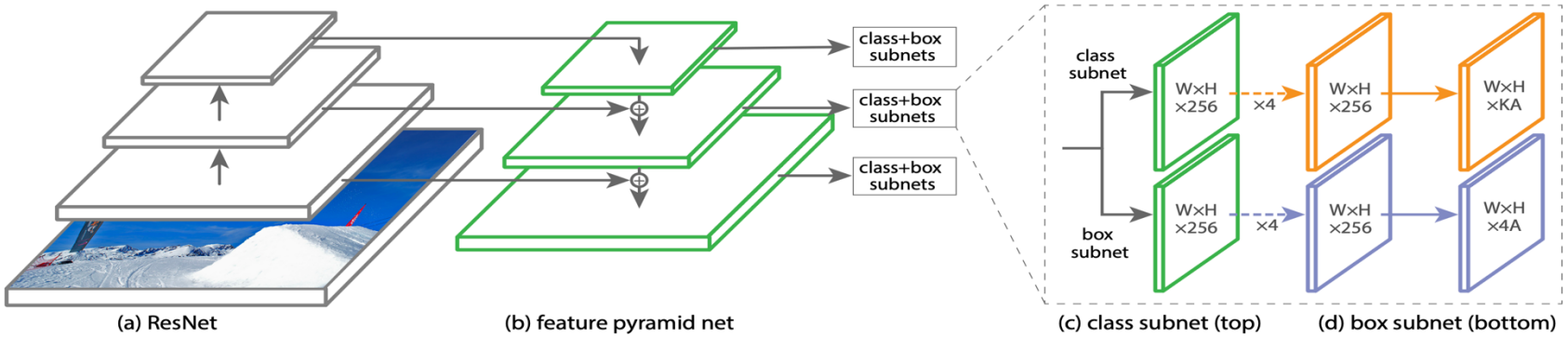


Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN [20] while running at faster speeds.

RetinaNet processes images in a single shot, predicting classes and bounding boxes for a dense grid of anchors across multiple feature map levels.

Workflow steps:

- Input image passes through a ResNet+FPN backbone, generating a pyramid of multi-scale feature maps.
- At each location on every feature map, multiple anchor boxes are generated with different scales and aspect ratios.
- Two subnetworks process each anchor:
- Classification subnet: predicts probability of each class for each anchor.
- Box regression subnet: predicts bounding box coordinates for each anchor.
- All predictions are post-processed with non-maximum suppression to remove duplicates.

RetinaNet Architecture

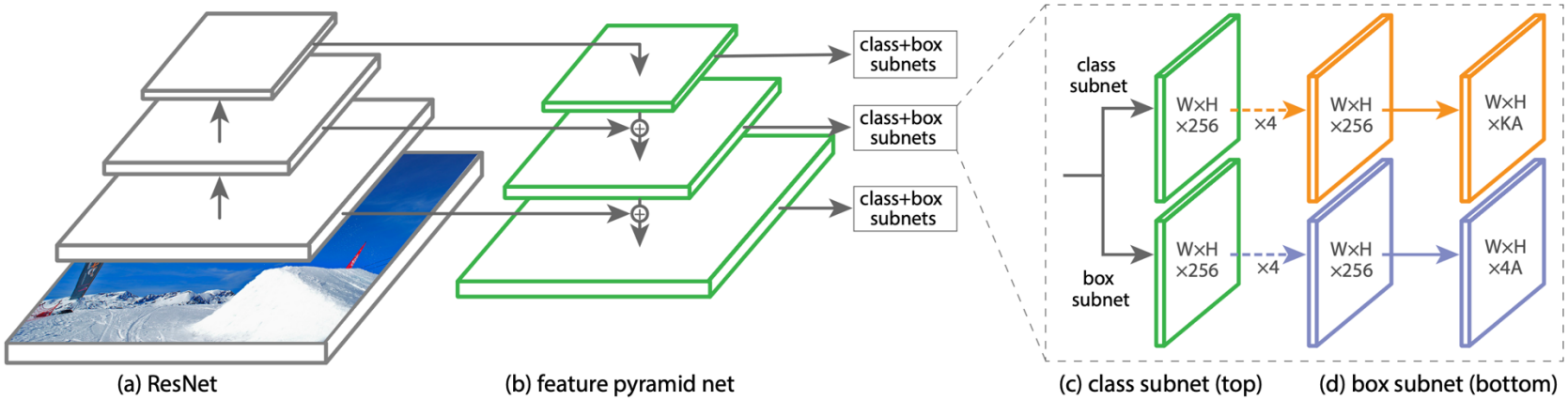


Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN [20] while running at faster speeds.

RetinaNet is built around three main components:

- ResNet + Feature Pyramid Network (FPN) backbone: Extracts rich, multi-scale features from the image.
- Classification subnet: Deep, small fully-convolutional network that outputs class probabilities for every anchor at every spatial position.
- Box regression subnet: Predicts box offsets for each anchor.
- Uses anchor boxes of multiple scales and aspect ratios at each feature map location.

RetinaNet: Focal Loss

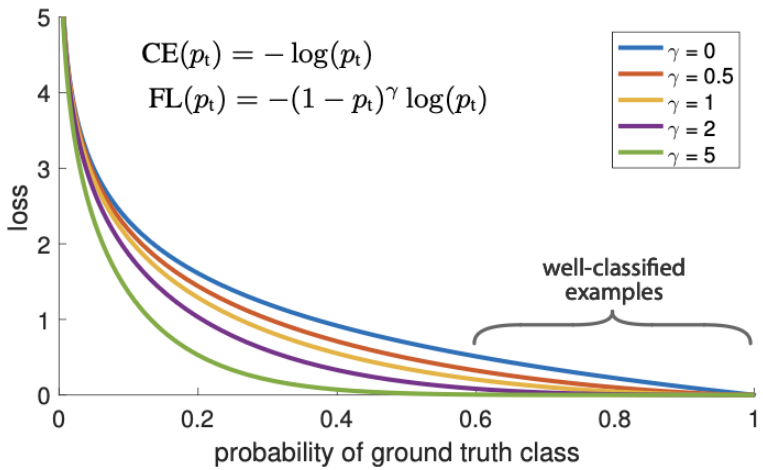
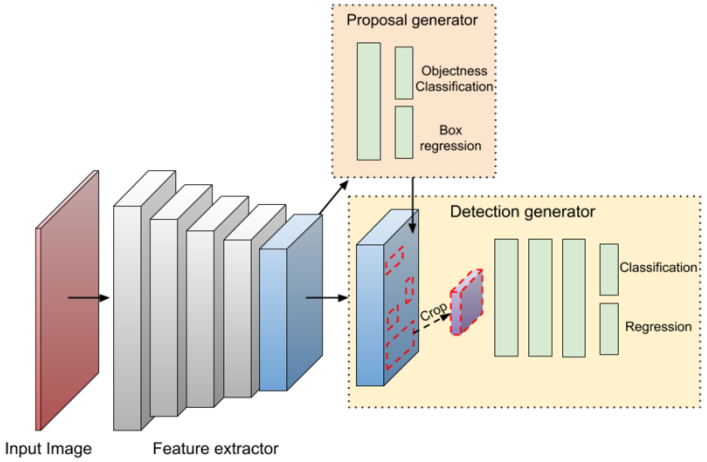


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^{\gamma}$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

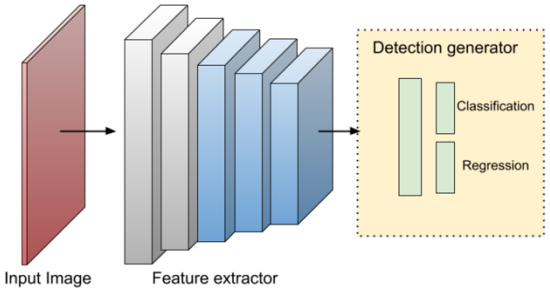
FL(pt) =−(1−pt)γlog(pt)

- Every sample is weighted according to its error!
- Modulating factor added
- Focusing parameter γ smoothly adjusts the rate at which easy examples are downweighted
- Misclassified, pt is small, modulating factor is near 1, loss is unaffected.
- As pt → 1, the factor goes to 0 and the loss for well-classified examples is down-weighted..
- with γ = 2, example classified with pt = 0.9 would have 100× lower loss compared to CE and with pt ≈ 0.968 it would have 1000× lower loss. This in turn increases the importance of correcting misclassified examples!

Two-Stage Object Detectors



(a) Two-stage Faster R-CNN



(b) One-stage RetinaNet

Two-stage detectors break the object detection process into two distinct steps:

1. Region Proposal:

- The first stage generates a set of candidate object regions (region proposals) within the image, which are likely to contain objects.
- Traditional methods (like Selective Search) use hand-crafted features and are computationally expensive, while modern two-stage detectors (like Faster R-CNN) use neural networks (Region Proposal Networks, RPNs) to generate these proposals efficiently

2.Region Classification and Refinement:

- The second stage classifies each proposed region and refines its bounding box.
- A convolutional neural network extracts features from each proposal and predicts the object class and bounding box adjustments.

Faster R-CNN

Faster R-CNN is a two-stage object detection framework that integrates deep learning for both generating region proposals and performing object classification and bounding box regression. It builds upon previous R-CNN models by introducing a Region Proposal Network (RPN), making the process much faster and more accurate.

- Combines region proposal and detection in a single, unified network.
- Achieves state-of-the-art accuracy with improved speed compared to earlier detectors.

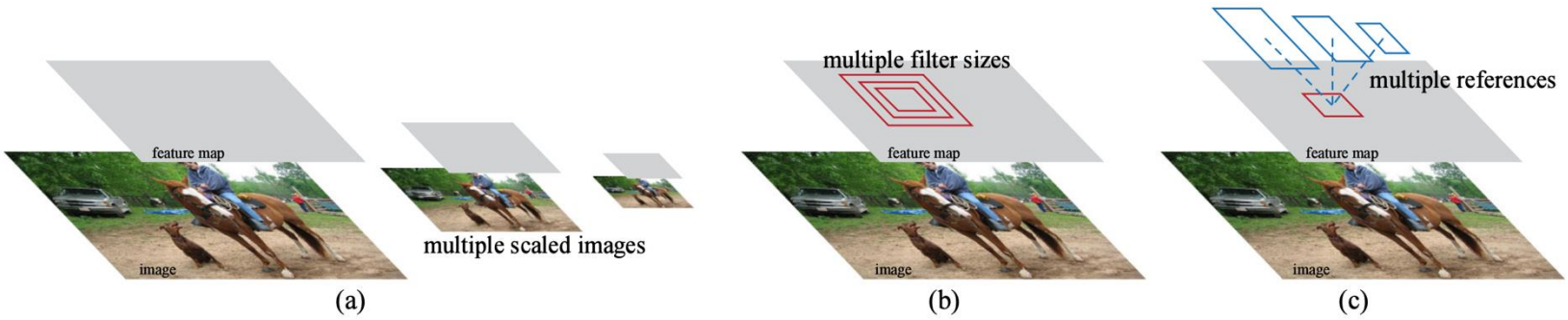
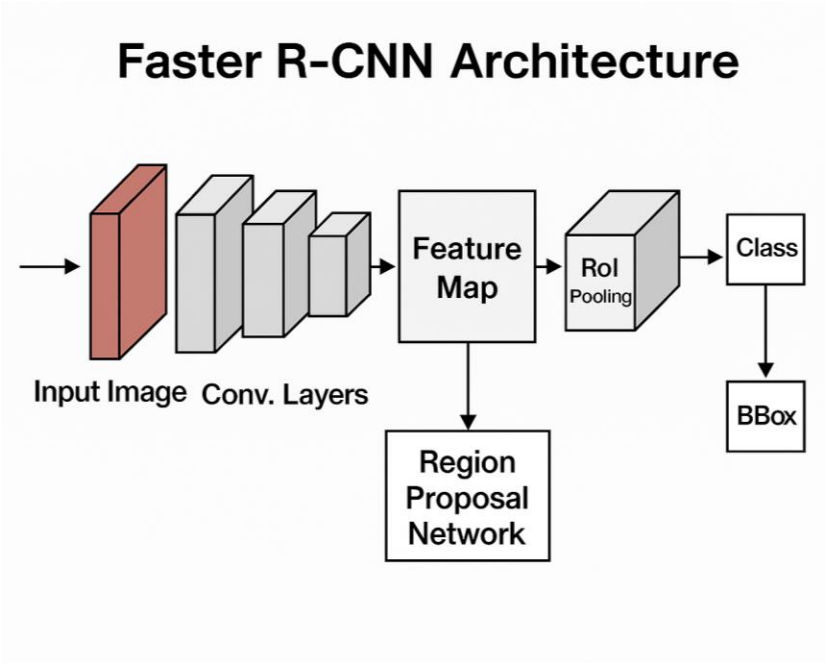


Figure 1: Different schemes for addressing multiple scales and sizes. (a) Pyramids of images and feature maps are built, and the classifier is run at all scales. (b) Pyramids of filters with multiple scales/sizes are run on the feature map. (c) We use pyramids of reference boxes in the regression functions.

Faster R-CNN - workflow



Faster R-CNN breaks the object detection process into two stages:

1. Region Proposal:

Uses a Region Proposal Network (RPN) to scan the feature map and generate candidate object regions (proposals) likely to contain objects.

Proposals are generated efficiently via sliding windows and anchor boxes at each position.

2.Region Classification & Refinement:

For each proposal, ROI pooling extracts a fixed-size feature map.

The network classifies the region (object category or background) and refines the bounding box coordinates.

- The RPN and the detection network share convolutional features, making the system efficient.

Faster R-CNN - Architecture

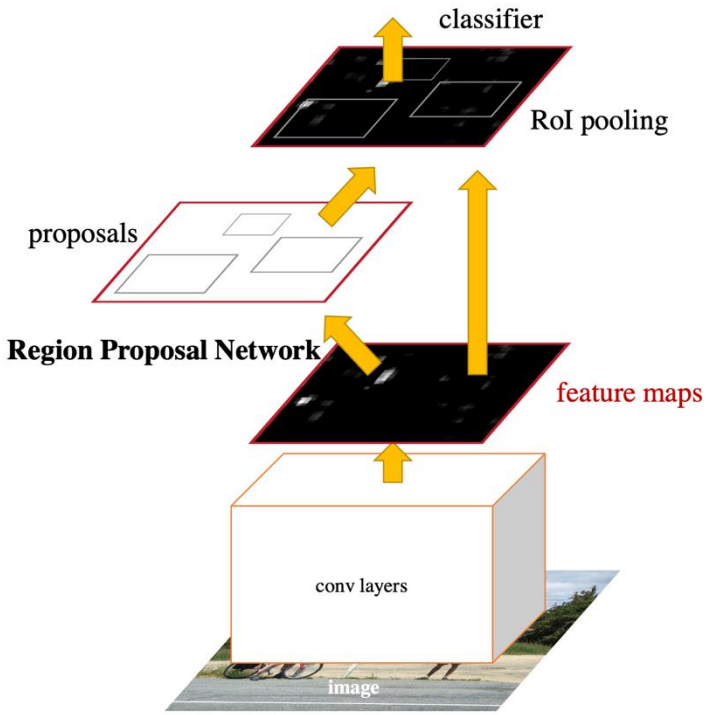


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

The architecture of Faster R-CNN is built around the following components:

- Convolutional Backbone: Extracts feature maps from the input image (e.g., VGG-16, ResNet).
- **Region Proposal Network (RPN):**

Slides over the shared feature map.

Proposes object regions using anchor boxes of different scales and aspect ratios.

- **ROI Pooling Layer:**

Extracts fixed-size feature maps from proposals for classification.

- **Fully Connected Layers:**

Classifies each region and regresses bounding box coordinates.

- **Unified Network:**

Shares convolutional features between RPN and detection head for speed and memory efficiency.

Masked R-CNN

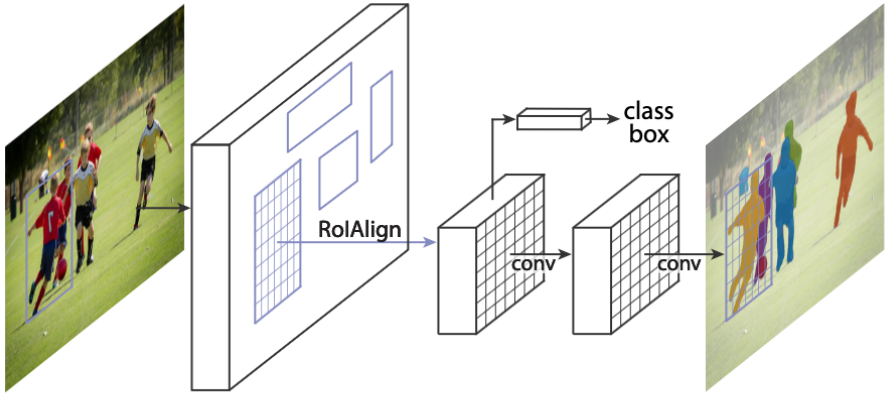


Figure 1. The **Mask R-CNN** framework for instance segmentation.

A flexible framework for instance segmentation that extends Faster R-CNN by adding a parallel branch to predict a pixel-level mask for each detected object .

- **RoIAlign**: Eliminates quantization in RoI pooling via bilinear interpolation, improving mask boundary accuracy by up to 50% .
- **Decoupled Mask & Class**: Predicts one binary mask per class (via per-pixel sigmoid) without competition between classes, while the detection branch handles classification and box regression .

Masked R-CNN Workflow

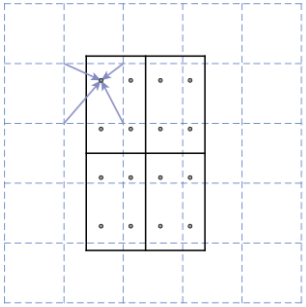


Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

- **Backbone + RPN:**

Extract multi-scale features using ResNet-C4 or FPN; generate region proposals via RPN sliding over shared feature maps .

- **RoIAlign:**

For each RoI, sample exact feature values at predefined points using bilinear interpolation—avoiding any quantization to preserve spatial alignment .

- **Parallel Heads:**

Classification Head: predicts object class.

Box Head: refines bounding-box coordinates.

Mask Head: a small fully convolutional network that outputs an m×m binary mask for each detected object .

- **Post-Processing:**

Apply NMS on boxes; upsample each m×m mask to the original RoI size and threshold at 0.5 to produce final instance masks .

Masked R-CNN Architecture

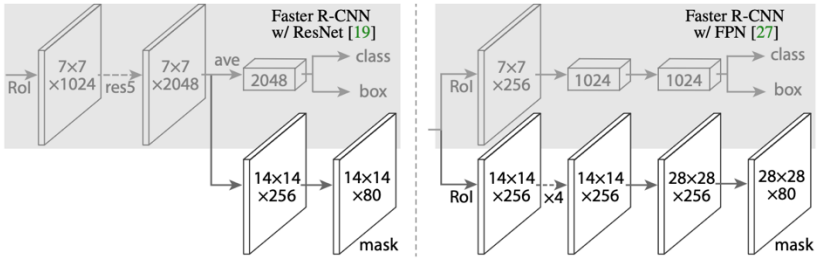


Figure 4. **Head Architecture**: We extend two existing Faster R-CNN heads [19, 27]. Left/Right panels show the heads for the ResNet C4 and FPN backbones, from [19] and [27], respectively, to which a mask branch is added. Numbers denote spatial resolution and channels. Arrows denote either conv, deconv, or *fc* layers as can be inferred from context (conv preserves spatial dimension while deconv increases it). All convs are 3×3, except the output conv which is 1×1, deconvs are 2×2 with stride 2, and we use ReLU [31] in hidden layers. *Left*: 'res5' denotes ResNet's fifth stage, which for simplicity we altered so that the first conv operates on a 7×7 RoI with stride 1 (instead of 14×14 / stride 2 as in [19]). *Right*: '×4' denotes a stack of four consecutive convs.

Backbone Variants:

- ResNet-50/101-C4 or ResNet-50/101-FPN (for multi-scale feature fusion) .

Region Proposal Network (RPN):

- 3×3 conv over feature map → two sibling 1×1 convs for objectness (2k scores) and bbox deltas (4k values) per anchor .

RoIAlign Layer:

- Samples features with bilinear interpolation into fixed-size bins (e.g., 7×7 for detection, 14×14 for mask) to feed subsequent heads .

Head Details:

- Box & Class Head: for each RoI, uses either the ResNet res5 block (C4 variant) or a 4-conv stack (FPN variant), followed by two FC layers for classification and box regression.
- **Mask Head:** 4×3×3 conv layers → 2×2 deconv (stride 2) → 1×1 conv to produce K masks of resolution 28×28 .

Training Loss:

- $L = L_{\_cls} + L_{\_box} + L_{\_mask}$

  where L_{\text{mask}} is the per-pixel binary cross-entropy on the true-class mask only

Cascade R-CNN

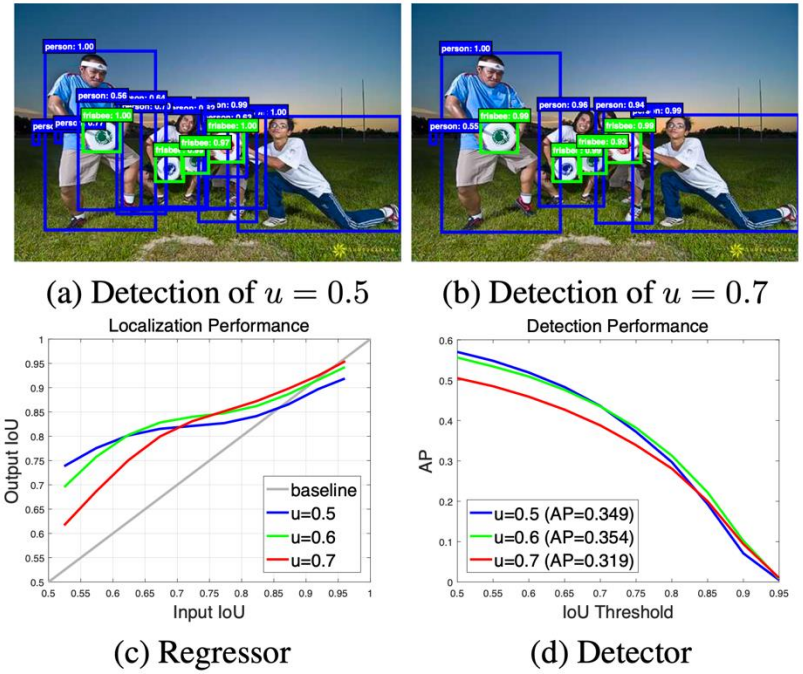


Figure 1. The detection outputs, localization and detection performance of object detectors of increasing IoU threshold u.

A multi-stage extension of the two-stage R-CNN framework that trains a sequence of detectors with increasing IoU thresholds to progressively reject close false positives and improve localization quality .

- Overfitting at High IoU: Raising the IoU threshold in a single detector leads to too few positives and overfitting.
- Quality Mismatch: A detector trained for one IoU level underperforms when evaluated on hypotheses of different quality .

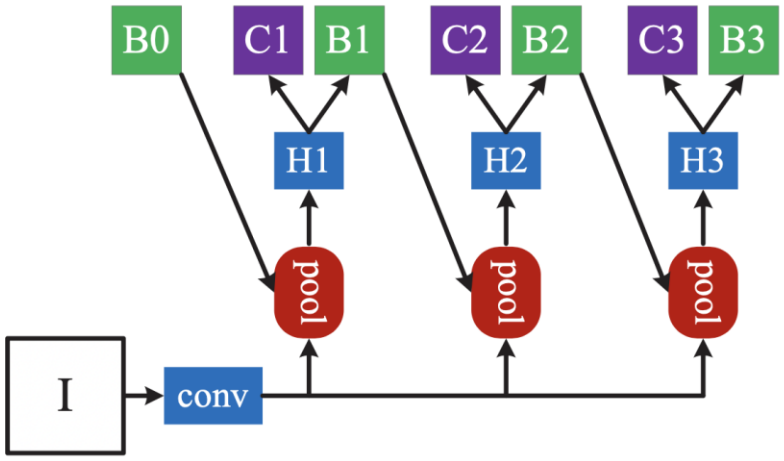
Core Idea:

- Train stages for u = 0.5, 0.6, 0.7 in sequence, using each stage's regressed outputs to resample higher-quality proposals for the next—maintaining ample positives and matching detector quality to proposal quality .

Impact:

- Surpasses all single-model detectors on COCO, with consistent AP gains (2–4 pts) over diverse baselines at marginal extra cost

Cascade R-CNN Workflow



(d) Cascade R-CNN

- **Stage 0 – RPN Proposals:**

Generate initial object proposals (b_0) via the Region Proposal Network sliding over the shared backbone features .

- **Stage 1 ($u_1 = 0.5$):**

Train detector head H_1 on (x, b_0), classify and regress to produce refined boxes b_1.

- **Stage 2 ($u_2 = 0.6$):**

Resample proposals: use b_1 as new hypotheses (x, b_1), train H_2 with IoU threshold 0.6 → outputs b_2.

- **Stage 3 ($u_3 = 0.7$):**

Repeat on b_2 with threshold 0.7 → final b_3.

- **Inference**:

Apply the same sequence: RPN → H_1 → H_2 → H_3, yielding progressively higher-quality detections.

Cascade R-CNN Architecture

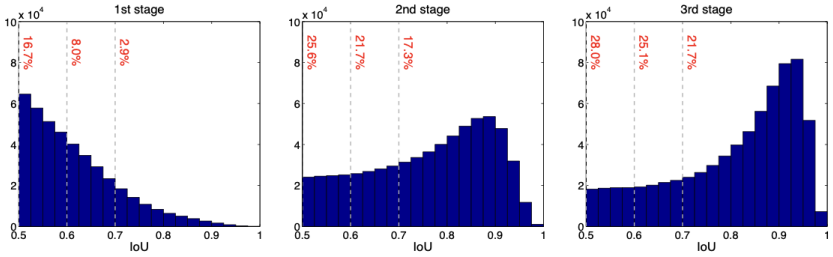


Figure 4. The IoU histogram of training samples. The distribution at 1st stage is the output of RPN. The red numbers are the positive percentage higher than the corresponding IoU threshold.

Backbone & RPN (Stage 0):

- Any CNN backbone (e.g., ResNet-50/101 with FPN); RPN head: 3×3 conv → sibling 1×1 convs for objectness and bbox deltas .

Cascade Heads (Stages 1–3):

- Each head t has the same architecture as the baseline detection head:
  - Box & Class Subnet: RoIAlign → 2 FC layers → softmax (K+1 classes) + 4-D bbox regression.
- IoU Thresholds: u_1=0.5, u_2=0.6, u_3=0.7.

Cascaded Regression:

- Regression functions f_1, f_2, f_3 composed as

$$f(x,b) = f_3(x,f_2(x,f_1(x,b)))$$

- Each f_t optimized on the resampled b_{t-1} distribution to avoid overfitting .

Multi-Task Loss per Stage:

- $L_t = L_{cls}(h_t(x), y_t) + [y_t >= 1]L_{loc}(f_t(x,b_{t-1}), g)$

where y_t labels proposals by IoU$\geq u_t$; enables joint classification and regression with balanced positive samples .

Transformer-Based Detection Models

DETR (DEtection TRansformer)

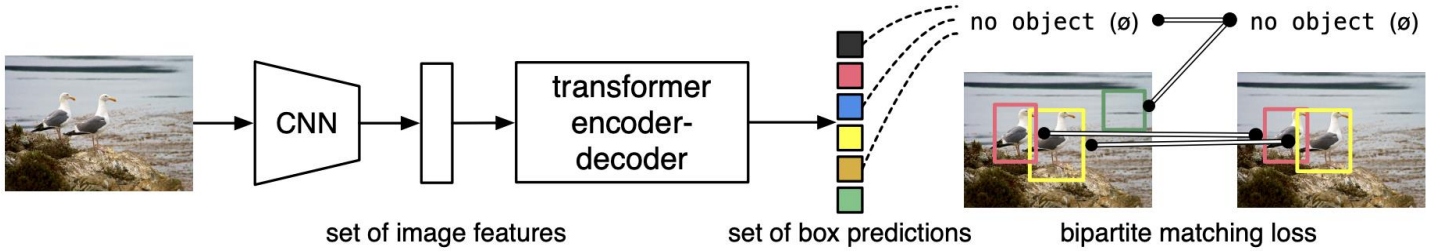


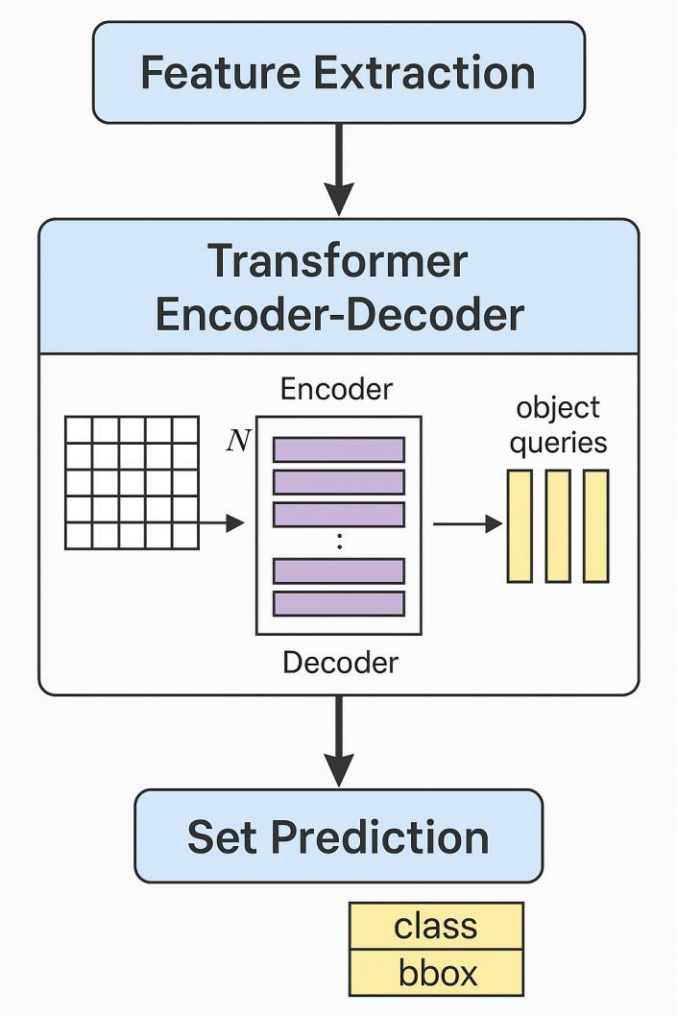
Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

DETR (DEtection TRansformer) is a novel object detection model that applies the transformer architecture, commonly used in natural language processing, directly to object detection. Unlike traditional detectors that rely on hand-crafted components like anchor boxes and non-maximum suppression, DETR simplifies detection by formulating it as a direct set prediction problem.

Key points:

- First fully end-to-end object detector using transformers.
- Eliminates the need for region proposal networks and post-processing steps.
- Achieves competitive accuracy with a streamlined, unified pipeline.

DETR Workflow



DETR processes an image through the following workflow:

- **Feature Extraction**: An input image is passed through a convolutional backbone (like ResNet) to generate a compact feature map.
- **Transformer Encoder-Decoder**: The feature map is flattened and passed into a transformer encoder, which models global context. The transformer decoder receives a fixed number of learned object queries and predicts bounding boxes and class labels for each query.
- **Set Prediction**: DETR outputs a fixed-size set of predictions. During training, Hungarian matching is used to optimally match predictions to ground truth objects.

DETR - Architecture

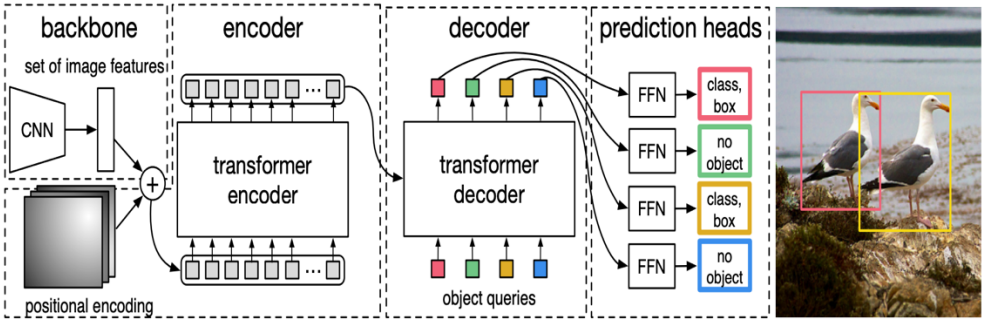


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

DETR architecture includes:

- **CNN Backbone**: Extracts dense feature maps from the input image.
- **Transformer Encoder**: Processes the feature map, attending globally to all image locations.
- **Transformer Decoder**: Receives learned object queries, decodes them into object representations.
- **Prediction Heads**: Each query is passed through feed-forward layers to produce a class label and bounding box coordinates.
- **End-to-End Loss**: The model is trained with a bipartite matching loss, allowing direct supervision without NMS or anchors.

DINO (Self-Distillation with No Labels)

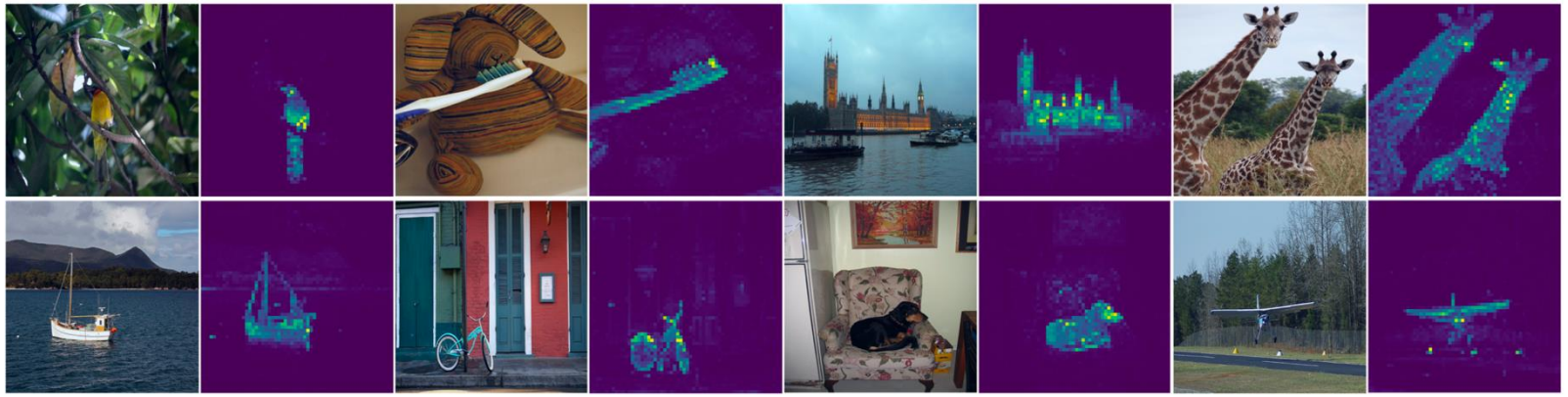


Figure 1: **Self-attention from a Vision Transformer with 8×8 patches trained with no supervision.** We look at the self-attention of the `[CLS]` token on the heads of the last layer. This token is not attached to any label nor supervision. These maps show that the model automatically learns class-specific features leading to unsupervised object segmentations.

DINO (Self-Distillation with No Labels) is a self-supervised learning method designed for Vision Transformers (ViTs) and convolutional networks. DINO enables models to learn rich, transferable features without any labeled data, and leads to unique properties—such as automatically discovering object boundaries and scene layout from images.

Key points:

- Learns high-quality visual representations *without supervision*.
- Achieves strong performance in classification, retrieval, and segmentation.
- Especially effective with Vision Transformers (ViTs), unlocking emergent properties not seen in supervised training.

DINO -Workflow

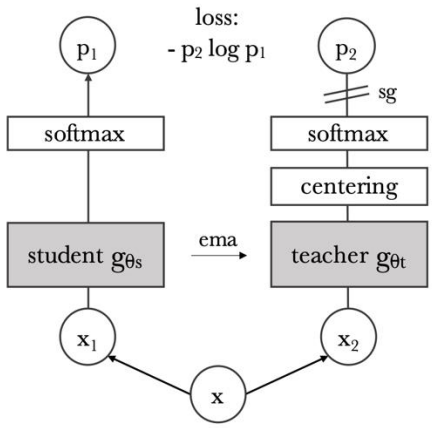
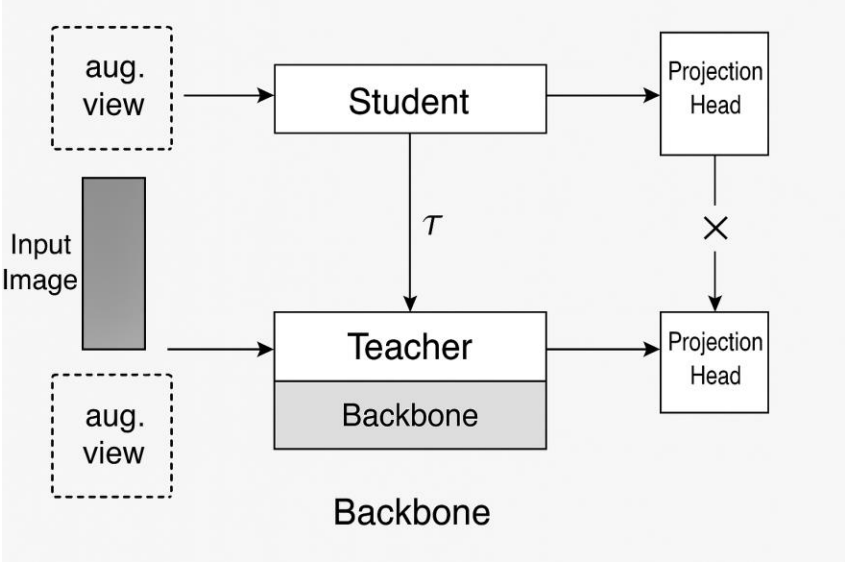


Figure 2: **Self-distillation with no labels.** We illustrate DINO in the case of one single pair of views (x_1, x_2) for simplicity. The model passes two different random transformations of an input image to the student and teacher networks. Both networks have the same architecture but different parameters. The output of the teacher network is centered with a mean computed over the batch. Each networks outputs a K dimensional feature that is normalized with a temperature softmax over the feature dimension. Their similarity is then measured with a cross-entropy loss. We apply a stop-gradient (sg) operator on the teacher to propagate gradients only through the student. The teacher parameters are updated with an exponential moving average (ema) of the student parameters.

The DINO workflow is based on a self-distillation framework with no labels:

- An input image is transformed into multiple different crops (global and local views).
- Each view is passed through two networks: a student and a teacher (both with the same architecture).
- The teacher is updated as an exponential moving average of the student (momentum encoder).
- The student is trained to match the teacher's output (using a cross-entropy loss), with no labels involved.
- This process uses *centering* and *sharpening* to avoid collapse (trivial solutions).

DINO - Architecture



DINO's architecture is flexible and can be applied to ViTs or CNNs.

- **Backbone**: Can be a Vision Transformer (ViT) or ResNet.
- **Projection Head**: Multilayer perceptron (MLP) on top of the backbone to produce embedding features.
- **Student & Teacher**: Identical architectures, but the teacher is updated via momentum (not gradients).
- **Input Augmentations**: Multi-crop strategy (two global crops and several local crops).
- **Output**: Student and teacher produce normalized feature vectors, and the student is trained to match the teacher.

Grounding DINO

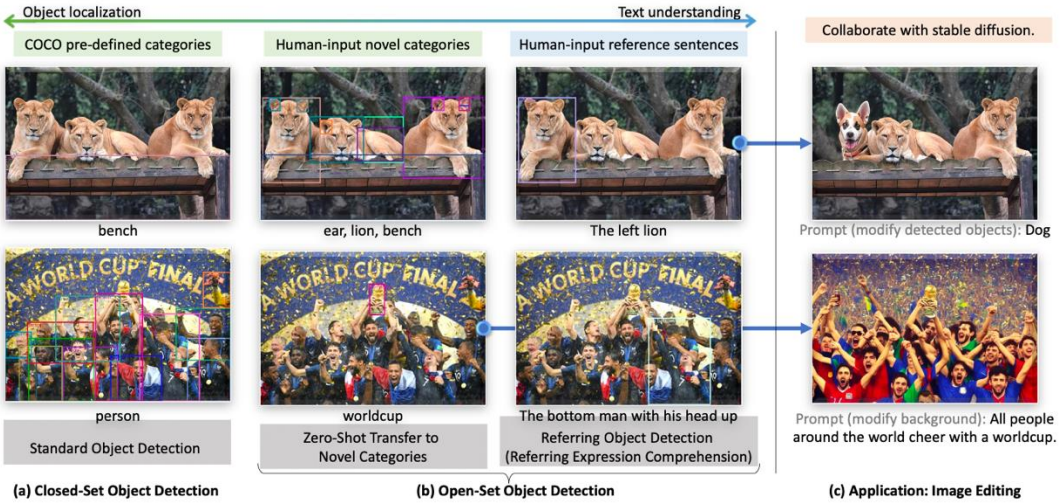


Fig. 1: (a) Closed-set object detection requires models to detect objects of pre-defined categories. (b) We evaluate models on novel objects and standard Referring expression comprehension (REC) benchmarks for model generalizations on novel objects with attributes. (c) We present an image editing application by combining Grounding DINO and Stable Diffusion [41]. Best viewed in colors.

An open-set object detector that marries the end-to-end DETR-based DINO architecture with grounded pre-training to detect arbitrary objects specified via language inputs (category names or referring expressions) .

Key Principles:

- Tight Modality Fusion: integrates vision and language across three phases—neck (feature enhancer), query initialization, and decoder—to learn language-aware region embeddings .
- Large-Scale Grounded Pre-Training: uses a mixture of detection, grounding (phrase-to-region), and caption data to generalize to novel categories without fine-tuning .

Grounding DINO - Workflow

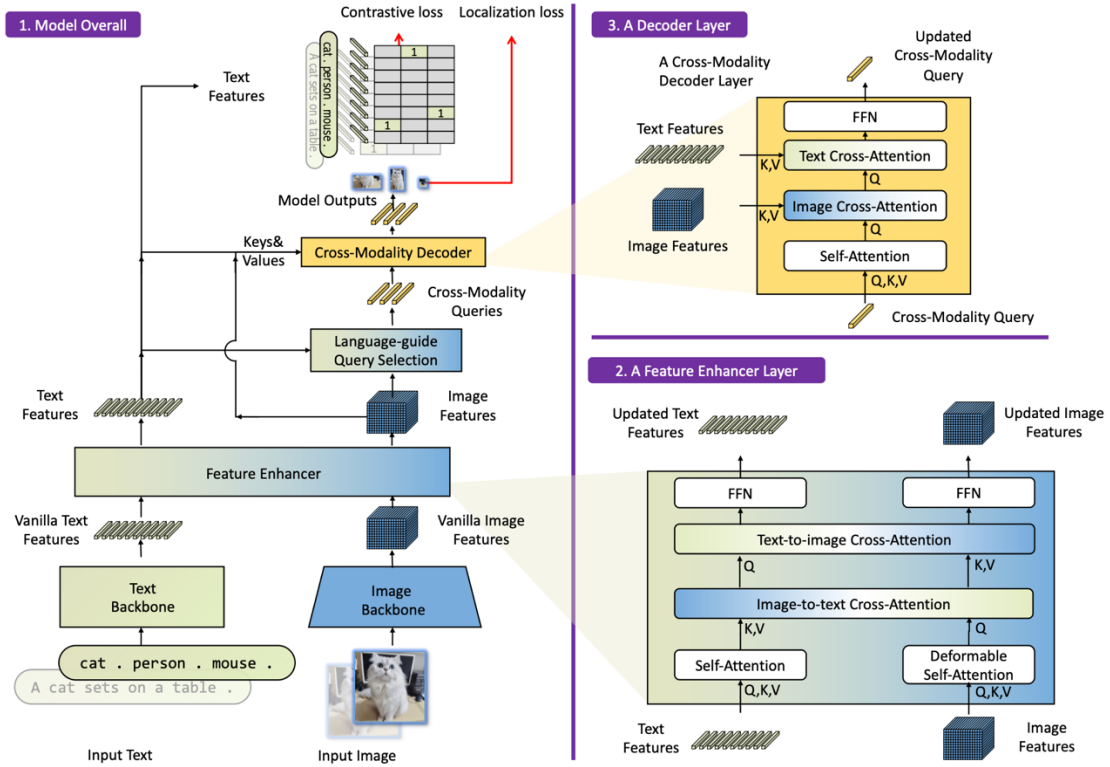


Fig. 3: The framework of Grounding DINO. We present the overall framework, a feature enhancer layer, and a decoder layer in block 1, block 2, and block 3, respectively.

Feature Extraction:

- Image Backbone (e.g., Swin Transformer) → multi-scale image tokens.
- Text Backbone (e.g., BERT) → token embeddings for input prompt.

Feature Enhancer (Neck):

- Stack of layers combining self-attention, image→text and text→image cross-attention, plus deformable self-attention to fuse modalities .

Language-Guided Query Selection:

- Compute similarity between image tokens and text features; select top N_q image tokens as cross-modality queries for the decoder .

Cross-Modality Decoder:

- Each decoder layer applies: self-attention → image cross-attention → text cross-attention → FFN, refining queries to capture joint semantics .

Prediction:

- Final queries predict bounding boxes (via box regression head) and ground to text spans (via contrastive classification loss).

Grounding DINO - Architecture

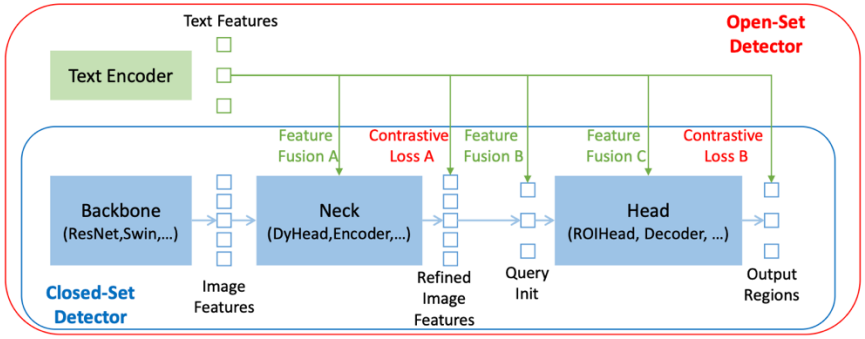


Fig. 2: Extending closed-set detectors to open-set scenarios.

Dual-Encoder–Single-Decoder Design:

- Image Encoder (Swin-T/L): produces multi-scale feature maps.
- Text Encoder (BERT): yields sub-sentence level embeddings to avoid cross-category interference .

Feature Enhancer Module:

L layers of:

- Deformable Self-Attention on image features
- Image→Text Cross-Attention
- Text→Image Cross-Attention
- FFNdecoder layer.

Language-Guided Query Selection:

- Select top N_q=900 image tokens via $\text{Top}N_q(\max_t(X_I, X_T))$ to initialize decoder queries .

Cross-Modality Decoder:

D layers of:

- Self-Attention → Image Cross-Attention → Text Cross-Attention → FFN
- Final queries feed into heads for box regression (L1 + GIoU loss) and text contrastive classification (focal loss) .

Training Loss:

- $L = L_{loc}^{(L1+GIoU)} + L_{cls}^{contrastive}$

with auxiliary losses after each decoder layer.

Vision–Language (Multi-Modal) Models

CLIP (Contrastive Language–Image Pre-training)

CLIP (Contrastive Language–Image Pre-training) is a vision-language model that learns to connect images and natural language through contrastive learning on a massive dataset of image–text pairs. Unlike traditional computer vision models that require labeled datasets and task-specific fine-tuning, CLIP can recognize and classify visual concepts using just a text description—enabling powerful zero-shot learning across many domains.

Key Points:

- Trained on 400 million (image, text) pairs from the internet.
- Can perform a wide variety of visual classification tasks without task-specific training.
- Enables zero-shot transfer: the model can classify new concepts just by providing their text descriptions.

CLIP - Workflow

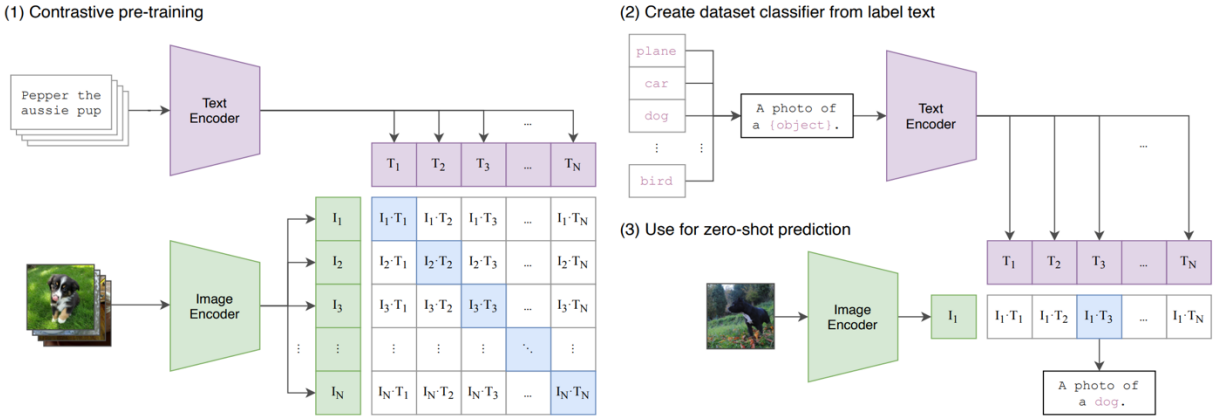


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

The CLIP workflow consists of three main steps:

1. Contrastive Pre-Training:

- The model is jointly trained to match images and their paired text descriptions in a shared embedding space.
- Two separate encoders (for images and text) are trained with a contrastive loss to bring paired image–text embeddings close, and push non-matching pairs apart.

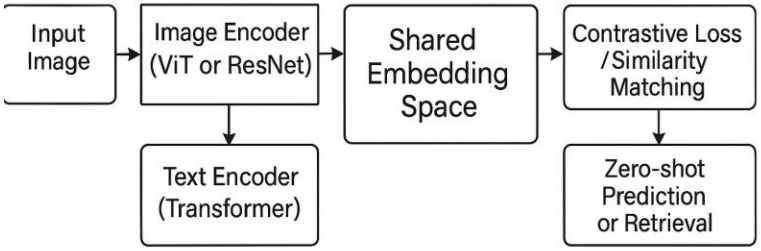
2. Zero-Shot Classification:

- To classify an image, CLIP computes similarities between the image embedding and a set of candidate text embeddings (e.g., "a photo of a cat," "a photo of a dog," etc.).
- The class with the highest similarity is chosen as the prediction.

3. Generalization:

- The model's text interface allows it to adapt to new tasks or classes simply by providing appropriate prompts—no additional training is required.

CLIP - Architecture



CLIP model architecture

CLIP uses a dual-encoder architecture:

Image Encoder:

- Can be a Vision Transformer (ViT) or ResNet; maps images to feature vectors.

Text Encoder:

- A Transformer-based network that maps tokenized text to feature vectors.

Shared Embedding Space:

- Both encoders are trained so that matching image–text pairs are close in this space.

Contrastive Loss:

- Ensures that only the correct image–text pairs are aligned.

Scalability:

- Trained on web-scale data (400M pairs), the architecture generalizes well to many downstream vision tasks.

References

YOLO (You Only Look Once)

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A.

You Only Look Once: Unified, Real-Time Object Detection

arXiv:1506.02640

EfficientDet

Tan, M., Pang, R., & Le, Q. V.

EfficientDet: Scalable and Efficient Object Detection

arXiv:1911.09070

SSD (Single Shot MultiBox Detector)

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed,

S., Fu, C. Y., & Berg, A. C.

SSD: Single Shot MultiBox Detector

arXiv:1512.02325

RetinaNet

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P.

Focal Loss for Dense Object Detection

arXiv:1708.02002

Faster R-CNN

Ren, S., He, K., Girshick, R., & Sun, J.

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

arXiv:1506.01497

References

Mask R-CNN (Mask Region-based Convolutional Neural Network)

He, K., Gkioxari, G., Dollár, P., & Girshick, R.

Mask R-CNN

https://arxiv.org/abs/1703.06870

Cascade R-CNN: Delving into High Quality Object Detection

Cai, Z., & Vasconcelos, N.

https://arxiv.org/abs/1712.00726

Grounding DINO (Grounded Detection with DINO)

Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Jiang, Q., Li, C., Yang, J., Su, H., Zhu, J., & Zhang, L.

Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection

https://arxiv.org/abs/2303.05499

DETR (DEtection TRansformer)

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S.

End-to-End Object Detection with Transformers

arXiv:2005.12872

DINO (Self-Distillation with No Labels)

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., & Joulin, A.

Emerging Properties in Self-Supervised Vision Transformers

arXiv:2104.14294

CLIP (Contrastive Language-Image Pre-training)

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., … & Sutskever, I.

Learning Transferable Visual Models From Natural Language Supervision

arXiv:2103.00020

Thank You