Balancing Accuracy-Speed Trade-offs in Object Detection

William Tao
Elmore School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907 USA
tao135@purdue.edu

Abstract

Object detection has consistently evolved with transformer-based detectors and various methods to improve accuracy, speed, and reduce computational cost. This paper analyzes three recent approaches in CVPR: DynamicDet: A Unified Dynamic Architecture for Object Detection [7], SET (Spectral Enhancement for Tiny Object Detection) [10], and DETRs Beat YOLOs on Real-time Object Detection [14]. Each paper addresses a problem in object detection, namely, dynamic routing, tiny object detection, and real-time detectors, while also introducing new ideas about improving performance. DynamicDet will be the main focus, exploring dynamic neural networks with adaptive inference to reduce computational cost while still achieving high accuracy. It's able to accomplish this by exiting the network based off an image's complexity. This re-implementation is evaluated on a mini-COCO dataset containing 5,000 images, with results proving its accuracy-speed trade-offs are worthwhile.

1. Introduction

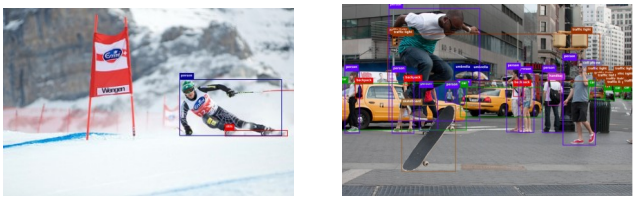
Object detection is an extremely important and in-demand area in computer vision that has many applications, including autonomous driving, surveillance, real-time detection, and many more. One of the main challenges is balancing the accuracy-speed trade-offs, as well as the computational cost that comes with it. This makes it difficult for a single model to be suitable for different situations and applications.

2. DynamicDet

Dynamic neural networks have recently emerged as a promising direction in the field of deep learning. They enable models to adapt their computation based on the input complexity. However, designing a dynamic detector remains challenging due to the difficulty of defining an appropriate exit criterion based for each input. In *DynamicDet: A Unified Dynamic Architecture for Object Detection* [7], Lin *et al.* (2023) introduces a unified dynamic architecture that dynamically selects between a lightweight and heavyweight detection backbone based on if an image is "easy" or "hard". The model was able to maintain a balanced accuracy speed trade-off, on the COCO [6] dataset, and performed well against modern YOLO [11] models.

2.1. Routing Detector

A human can easily identify the objects in an image and can also classify an image as "easy" or "hard". In the two examples shown in Figure 1, the distinction of difficulty between the two images are quick to spot, but can be confusing for a model to understand. Also, the time for a model to identify the objects in the "easy" image will be significantly faster than the "hard" image.



"Easy" image "Hard" image

Figure 1. Comparison of "Easy" and "Hard" images. Adapted from Lin *et al.* [7].

This idea can be applied to a model's pipeline where an image can be classified as "easy" or "hard" after an initial backbone pass, and can either exit or continue the pipeline based on the predicted difficulty. A "hard" image would go through a more intensive backbone, with the goal of extracting deeper features for better detection and localization.

To implement this, Lin *et al.* [7] introduces a router network that predicts a routing score $\phi \in (0, 1)$ for each input image x. This router is a lightweight classifier that operates on the features from the first backbone. The routing score is computed as:

$$\phi = \sigma(W_2(\delta(W_1 F + b_1)) + b_2, \tag{1}$$

where F denotes the compressed multi-scale features of the input image, W_1, W_2, b_1 and b_2 are the learnable parameters, and δ, σ denote the ReLU and sigmoid activation functions, respectively. ϕ represents the probability that the image is "hard" and should be forwarded to the heavyweight backbone.

The decision to exit or continue through the model pipeline is determined by a threshold τ_{val}. DynamicDet [7] bases the threshold value off of latency, using the following equation

$$k = \frac{lat_t - lat_1}{lat_2 - lat_1}, \; lat_1 \leq lat_t \leq lat_2, \tag{2}$$

where k serves as the maximum allowable proportion of "hard" images, lat_t represents the target latency, and lat_1, lat_2 is the latency of the first and second backbone, respectively. By adjusting the target latency lat_t, the dynamic model can achieve different accuracy-speed trade-offs. The threshold τ_{val} is

$$\tau_{val} = percentile(S_{val}, k) \tag{3}$$

where S_{val} is the difficulty scores of the validation set, and the percentile operation computes the k-th quantile of the data. With this strategy, one detector is able to balance the accuracy and speed trade-offs.

2.2. Backbone Design

The backbones play a crucial role in object detection models as they serve as the primary feature extractor, transforming images into high-level representations that can be used for classification and localization. Common backbones such as ResNet [2] and MobileNet [4] are all single-pass architectures, which means that they sequentially produce their features, and cannot be skipped. The choice of backbone can also significantly affect the trade-off between accuracy, speed, and computational cost.

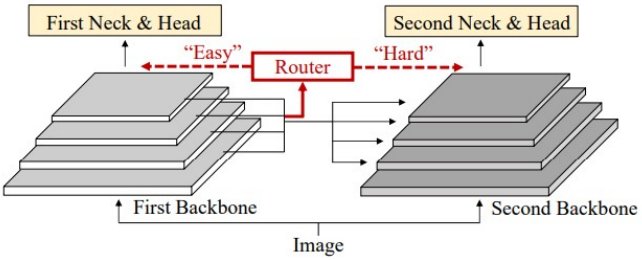


Figure 2. Two backbone and router architecture in DynamicDet. Reproduced from Lin *et al.* [7].

As seen in Figure 2, the dynamic architecture contains one lightweight backbone, one heavyweight backbone, and a router. For an input image x, its features are extracted by the lightweight backbone B_1 as

$$F_1 = B_1(x) = [f_1^1, f_1^2, ..., f_1^L] \tag{4}$$

where L denotes the number of multi-scale features. These features are then fed into the router R where it predicts a difficulty score ϕ, explained in 2.1. If the difficulty score is greater than the threshold τ_{val}, calculated in 3, then the heavyweight backbone B_2 will take the features from B_1 and continue processing. The goal is to have "easy" images exit after the B_1, while "hard" images follow through with B_2. This can be represented as

$$F_2 = B_2(x, H) = [f_2^1, f_2^2, ..., f_2^L] \tag{5}$$

where H is a composite connection module from the DHLC of CBNet [5]. This module helps connects previous layers from earlier layers to preserve information and prepare for further extraction in the heavyweight backbone. With this two-step structured architecture, accuracy and speed trade-offs can be achieved with variable difficulty score calculations, leading to early exits for images.

2.3. Optimization

The dynamic model is optimized in a unique, but well-thought out strategy. The two detectors are trained together with the following training objective

$$\min_{\Theta_1 \Theta_2} (L_{det}^1(x, y|\Theta_1) + L_{det}^2(x, y|\Theta_2), \tag{6}$$

where x, y denote the input image and the ground truth respectively, Θ_1 represents the learnable parameters of the lightweight/heavyweight detectors i, and L_{det}^i denotes the training loss (bounding box regression and classification loss) for detector i.

Optimizing the router to classify difficulty is more challenging, because if we are minimizing the loss function, the router will almost always choose the heavyweight backbone because it will have a lower training loss which makes sense because we expect it to perform better. However, this is not what we want with the router, balancing accuracy, speed and computation requires more constraints for the loss function. A hyperparameter-free optimization strategy was proposed which defines the difficulty criterion based on the training loss difference between the two detectors given an image. More specifically, if the loss difference between the two detectors is minimal, then the image can be classified as "easy", and "hard" if the difference is large.

2.4. Experiments

The model outperformed the baseline model YOLOv7 [11] when tested on the COCO [6] benchmark. They built multiple dynamic models with different settings for the router

threshold. The dynamic models achieved a higher AP (average precision) score and FPS (frames per second) than the standard YOLOv7 model. Specifically, their DyYOLOv7-W6/50 achieved 56.1% AP with 58 FPS, which was 0.1% more accurate and 12% faster than YOLOv7-E6.

2.5. Trade-offs

Generally speaking, most models are designed for a better accuracy-speed trade-off, prioritizing accuracy and speed. To achieve better accuracy, more channels and layers are added to improve feature extraction, but that slows the model down. Furthermore, this approach requires training multiple detectors for the best trade-off, which leads to a large amount of computational power. The DynamicDet [7] model architecture solves this problem by broadening its capabilities, allowing the model to perform well over a variety of datasets, while balancing accuracy-speed and computation trade-offs.

3. SET

Tiny object detection (TOD) is a challenging problem in computer vision applications due to limited and often noisy pixel information, along with weak feature extractions from those areas. The paper *SET: Spectral Enhancement for Tiny Object Detection* [10] proposes a spectral-based enhancement approach to amplify the frequency signatures of smaller objects in an image. There are two modules in this architecture, a Hierarchical Background Smoothing (HBS) module that suppresses high-frequency noise in the background, and an Adversarial Perturbation Injection (API) module that increases feature saliency and refines features in critical regions. The paper experiments on four datasets and demonstrates improvements in detecting tiny objects over baseline TOD methods.

3.1. Tiny Object Detection

Tiny objects are characterized by their limited pixel input, occupying less than or equal to 16×16 pixels. The main challenge is to preserve features after downsampling, as most operations such as max pooling keep the most important pixel value, so weak values will be completely overlooked. Furthermore, tiny objects can be missed from noisy backgrounds, making it harder to detect them.

Inspired from occlusion-based analysis, Sun *et al.* [10] uses a Feature Pyramid Network (FPN) to determine the impact of masking specific frequency bands for detection performance. The features are transformed to the frequency domain using an n-dimensional Fast Fourier Transform (FFT), and then fed to an Inverse Fast Fourier Transform (IFFT) after static filters are applied which generates masked spatial features. An example of these operations is shown in Figure 3, where they directly remove high-frequency features and enhance tiny objects. Working in the frequency domain separates the features by amplitude (strength of the feature) and phase (location of frequency). When the filters are applied to the amplitude, the phase remains the same, preserving the location of the tiny objects after inverting the Fourier Transform.

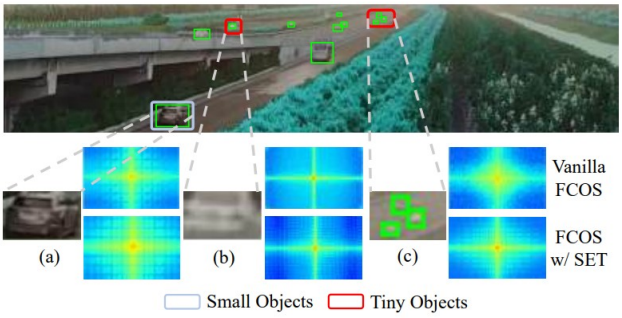


Figure 3. Different objects after feature encoding with FFT. Reproduced from Sun *et al.* [10].

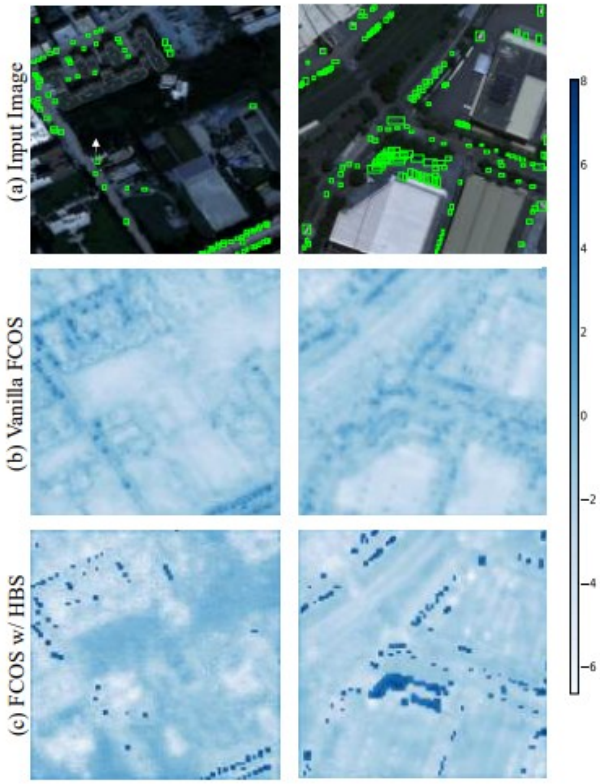


Figure 4. The difference between an input image with ground truth boxes applied with Vanilla FCOS and FCOS w/ HBS. Reproduced from Sun *et al.* [10].

3.2. Hierarchical Background Smoothing

The Hierarchical Background Smoothing (HBS) module is the first of two, and suppresses the high-frequency information in the background while preserving tiny details. Given an input image $X \in \mathbb{R}^{3\times W\times H}$, a binary mask $M \in 0, 1^{W\times H}$ is generated based off the ground truth box B with the following equation.

$$M_{i,j} = \mathbf{1}[(i,j) \in B], \tag{7}$$

The function $\mathbf{1}$ represents the value of location (i, j), which is 1 if the object belongs, and 0 otherwise. The mask decouples the image features and performs background smoothing. An adaptive smoothing operation $\phi_i(\cdot, r)$ is then utilized on the mask to compress information while retaining salient features. This makes it hard for high-frequency information to be recreated, resulting in a smoother background. As shown in Figure 4, applying HBS exemplifies clearer transitions between background and tiny objects compared to a Vanilla FCOS (Fully Convolutional One-Stage Object Detection).

3.3. Adversarial Perturbation Injection

The second module in the architecture, Adversarial Perturbation Injection (API), is designed to enhance the saliency (regions that stand out) in critical regions with adversarial perturbations. Adversarial perturbations are extremely small changes in an image that affects the prediction in the neural network. SET uses this as a training signal to enhance feature representations. We can represent the adversarial perturbation as ε_i^{adv} for each FPN layer as an expectation over M detection branches, where they are scaled by a balancing parameter λ_m. The equation is as follows:

$$\varepsilon_i^{adv} = \sum_{m=1}^{M} \lambda_m \cdot \varepsilon_{i,m}^{*}, \tag{8}$$

where $\varepsilon_{i,m}^{*}$ is the adversarial perturbation for the m-th branch. This design allows for well-balanced semantics across multiple branches, along with evenly distributed training.

3.4. Experiments

Sun *et al.* [10] evaluates the model on the following datasets: AI-TOD [13], VisDrone2019 [15], DOTA-v2.0 [12], and COCO [6]. They mainly tested on the AI-TOD dataset, where the average object size was 12.8 pixels, which is substantially smaller than objects in other datasets. The SET models enhanced all baselines by about 2% AP, which is a significant improvement proportional to the visual detection results.

3.5. Trade-offs

Performing the HBS and API modules sacrifices speed for accuracy. Their experiments resulted in an overall increased AP score, shown in Section 3.4. However, the HBS module increased training time, computational cost, and memory costs by 17%, 0.9%, and 0.9%, respectively. The API module had a similar impact on time due to adversarial training, but maintained the same computational costs and memory. Even with negative overhead, SET [10] makes up for these losses with significantly better detection results for tiny objects, making it a good trade-off.

4. DETR

Recent YOLO series models have become extremely popular and effective for real-time object detection because of their great speed and accuracy trade-offs. One of the main attributes that slows YOLO models down is Non-Maximum Suppression (NMS). The goal of NMS is to reduce overlapping bounding boxes using Intersection over Union (IOU) calculations on detected objects, keeping the most confident detection. End-to-end transformer-based detectors (DETRs) present an alternative to eliminate NMS, and reduce computational cost. The paper *DETRs Beat YOLOs on Real-time Object Detection* [14], proposes the **R**eal-**T**ime **DE**tection **TR**ansformer (RT-DETR), a real-time end-to-end object detector that addresses the NMS problem. There are two points that Zhao *et al.* focuses on, maintaining accuracy while improving speed, and maintaining speed while improving accuracy. This paper dives into how to design a hybrid model that attempts to push the limits of these trade-offs.

4.1. Disadvantages of NMS

Modern real-time object detectors utilize a CNN-based (Convolutional Neural Network) approach, most notably, YOLO [8] (You Only Look Once). These models have reasonable trade-offs between speed and accuracy, but require NMS for post-processing the bounding box detections. NMS not only slows down inference speed, but also introduces hyperparameters that can cause instability in these metrics. NMS can also be difficult to adapt to in different scenarios given the circumstances, which makes it hard when developing the detector. DETRs have recently received a lot of attention because they eliminate the NMS stage. However, their high-computational cost hinders real-time detection requirements, but brings a fresh perspective using transformer-based reasoning. RT-DETR [14] aims solve this with a hybrid encoder and improve real-time inference cost, making their approach appealing and competitive in the object detection field.

4.2. Hybrid Encoder

The biggest cost of the traditional DETR was the transformer encoder. Leveraging a hybrid encoder that switches between CNN and transformer blocks given the feature significantly improves inference speed. Their RT-DETR

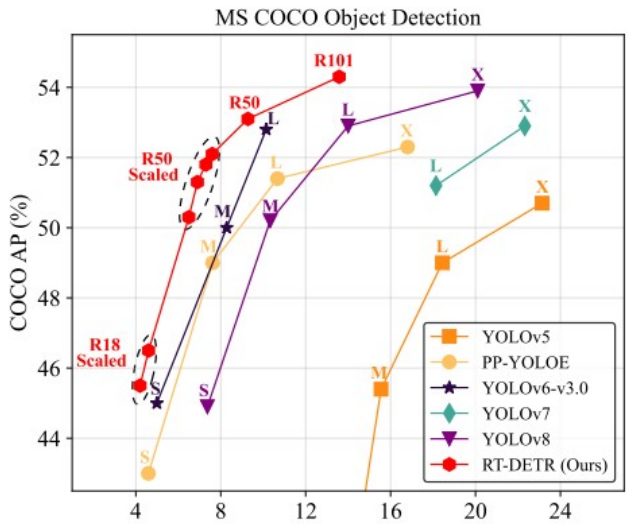


Figure 5. Comparing DETRs latency and AP performance against different YOLO series models. Reproduced from Zhao *et al.* [14].

two-step approach decouples intra-scale interactions and cross-scale features with different scales. Standard DETRs use the encoder to gather global attention, taking all features map tokens at all scales. However, by first processing each scale's features individually with Attention-based Intra-scale Feature Interaction (AIFI), (e.g, 1/8, 1/16, 1/32 of original image), local context is captured efficiently. Then, with Cross-Scale Feature Fusion (CCFF), it combines these contexts, allowing them to fit together without losing important information, while reducing computational cost and speeding up the process. Along with this, RT-DETR introduces Uncertainty-minimal Query Selection (UMQS), which provides high-quality initial queries for the decoder by optimizing uncertainty, therefore optimizing accuracy.

RT-DETR also supports flexible speed tuning to accommodate different real-time scenarios without retraining, due to its multi-layer decoder architecture. Later touched on in Section 4.4, Zhao *et al.* [14] designed variant RT-DETR models with different ResNet [2] backbones, targeting different trade-offs. As shown in Figure 5, the RT-DETR models score higher than the YOLO models on AP percentage while also staying ahead in end-to-end latency.

4.3. Uncertainty-minimal Query Selection

Query selection schemes use confidence scores to select the top K features from the encoder to initialize object queries. The confidence score of the feature is jointly correlated with the category and localization because of the detector's output. This leads to uncertainty for selected features, resulting in poor initialization and performance for the decoder. An Uncertainty-minimal Query Selection (UMQS) scheme uses the encoder's predicted localization uncertainty, and prioritizes the features with the lowest uncertainty, allowing for an improved set of queries. They use the following loss function in Equation 9 to optimize this,

$$L(X, \hat{Y}, Y) = L_{box}(\hat{b}, b) + L_{cls}(U(X), \hat{c}, c) \qquad (9)$$

where $\hat{Y}$ and Y denote the prediction and ground truth, $\hat{c}$ and $\hat{b}$ represent the category and bounding box, respectively, X being the encoder feature, and $U(X)$ denoting the discrepancy between the predicted distributions of localization and classification.

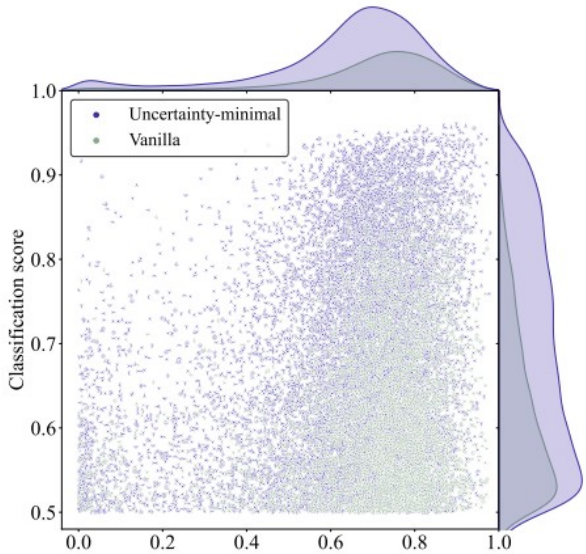


Figure 6. Comparing classification and IoU scores of UMQS and Vanilla query selected encoder features. Reproduced from Zhao *et al.* [14].

In Figure 6, the closer the dot is to the top right, the higher the quality feature, *i.e.*, the more likely the predicted category and bounding box are to the ground truth. The purple dots, representing the UMQS queries, clearly show the improvements with this approach.

4.4. Experiments

Multiple experiments were made comparing RT-DETR's performance against YOLO series models on the MSCOCO [6] dataset. The RT-DETR-R50 (ResNet50) achieved 53.1% AP and 108 FPS, outperforming YOLOv7 and YOLOv8 detectors by 1.9% and 0.2% AP, and increased FPS by 96.4% and 52.1%. This margin of improvement is significant and proves that RT-DETR models can compete and outperform modern YOLO models in both accuracy and speed.

4.5. Trade-offs

With RT-DETRs, their hybrid encoder improves on the original DETR's architecture, significantly improving accuracy

and speed. Zhao *et al.* [14] has set a fantastic foundation for their RT-DETR tradeoffs, with their heavyweight model RT-DETR-R101 (ResNet101) scoring the highest on AP percentage with above average FPS, and their RT-DETR-R50 model scoring slightly less, but with the highest FPS.

5. My Implementation

My reimplementation revolves around the DynamicDet [7] paper, where I'm using PyTorch's Faster RCNN MobileNetV3 [9] [3] as the lightweight backbone, and PyTorch's Faster RCNN ResNet50 [9] [2] as my heavyweight backbone. The dynamic router follows the paper's implementation, using two linear layers, separated by a ReLU and a sigmoid activation function.

5.1. Dataset

In the original DynamicDet [7] paper, they evaluate their model on the COCO [6] dataset, where they trained on 118k images, and tested on the 5k validation images and 20k test images. I don't have the computing resources or time to train and test on a dataset that large, so I used a Mini-COCO [1] dataset found from Kaggle containing 5k images from the original COCO. This custom dataset skipped 11 classes from the original 91, keeping only 80 classes. To prevent training errors (comparing missing classes from the annotations), I had to enumerate through the classes and edit the class IDs to ensure they fit with the model's labels.

5.2. Hyperparameters

From the 5,000 images, I created multiple subsets containing 100, 200, and 500 images to evaluate my model on. I used a standard 80/20 train and test split (400 train, 100 test), with a batch size of two, and also utilized two workers in the Data Loader for parallel computations. I trained both backbones for 50 epochs, and optimized the loss function with Stochastic Gradient Descent, using a learning rate of 5×10^{-3}, momentum of 0.9, and weight decay of 1×10^{-4}.

For the router, I also trained over 50 epochs, using the Adam optimizer instead with a learning rate of 1×10^{-3}. To evaluate if the loss difference between backbones was "minimal" or "large", I used a margin of 0.3. Equation 10 visualizes this relationship:

$$l = L_{B_2} + margin < L_{B_1}, \tag{10}$$

where L_{B_i} is the loss value for backbone i. I used a Binary Cross Entropy (BCE) loss function to measure the accuracy of the predicted label float value from the router's forward pass, with the marginalized difference label result $l \in \{0, 1\}$. I didn't use a specific latency-based threshold for the router score, but used a value of 0.3 to target a 70/30 split for lightweight and heavyweight backbone routings.

To determine if the dynamic model was performing as expected, I compared its performance against a standalone Faster RCNN MobileNetV3 [3] and Faster RCNN ResNet50 [9] model, simulating the lightweight and heavyweight backbones. Both models were initialized with the default weights, and finetuned over 50 epochs on the same training and testing set as the dynamic model.

5.3. Performance Evaluations

Model	Precision	Recall	F1	mAP	FPS	Inference (ms)
Dynamic	0.521	0.474	0.497	0.1774	6.12	163.28
Lightweight	0.581	0.398	0.472	0.1609	**13.98**	**71.54**
Heavyweight	**0.625**	**0.546**	**0.583**	**0.2405**	4.25	235.15

Table 1. Performance comparison of DynamicDet, Lightweight, and Heavyweight detectors.

As seen in Table 1, the dynamic model performs as expected, scoring between the lightweight and heavyweight detectors in all categories except precision. The dynamic model's router distribution was 71% and 29% for lightweight and heavyweight backbones, respectively. This means that the lightweight model kept its performance close to the heavyweight model. The most important metrics to pay attention to is F1 Score, mAP, and FPS because they quantify the quality, accuracy, and speed of detections, respectively. The F1 Score is the harmonic mean of Precision and Recall, calculated with Equation 11 and 12:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{11}$$

where Precision and Recall are calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \tag{12}$$

with TP, FP, and FN denoting True Positive, False Positive, and False Negative.

The Mean Average Precision (mAP) is a standard metric for detection accuracy, and gives an overall percentage of how well a detector performs. The heavyweight outperforms the other models as expected, with the dynamic model performing mildly worse, but slightly outperforming the lightweight model.

The FPS measures frames per second, and is a key measurement for real-time efficiency. The lightweight model leads in this metric, with the dynamic model being around 43.78% slower.

Overall, the dynamic model balances the accuracy-speed trade-offs well. The results don't look extremely impressive or on par with modern models because of a lack of resources and time, but they demonstrate the core idea of a dynamic neural network architecture and the benefits of utilizing them. Different hyperparameters and/or dataset size may show improved results for the dynamic model.

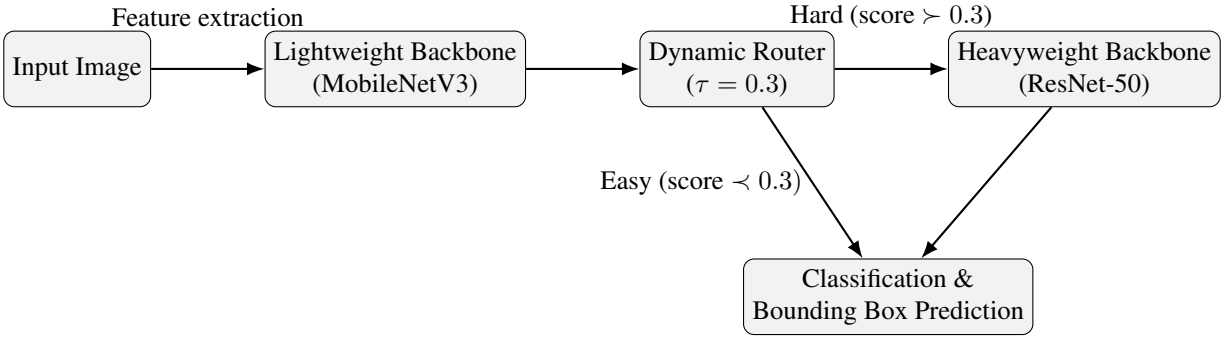
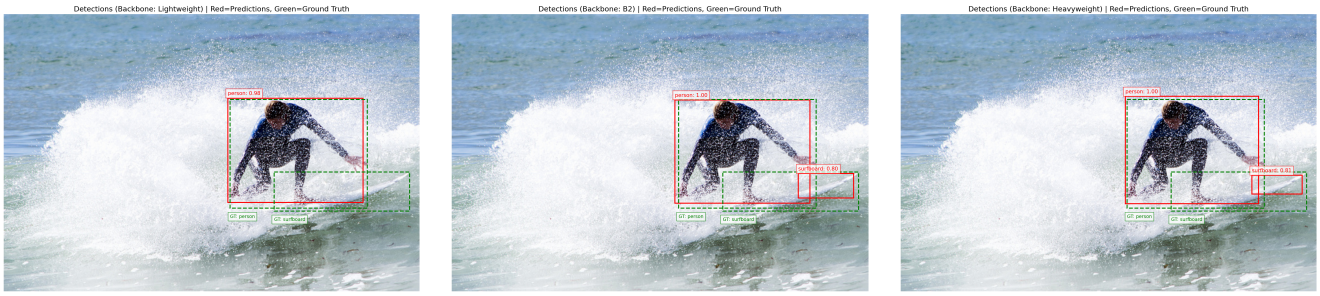


Figure 7. Workflow of my DynamicDet reimplementation. The dynamic router compares the routing score against threshold $\tau = 0.3$ to determine whether an image exits early or proceeds to the heavier backbone.



(a) Lightweight Detector (b) Dynamic Detector (c) Heavyweight Detector

Figure 8. Comparison of detection results across the three models on the same input image. The dynamic model continued with B2 (heavyweight) for further feature extraction

5.4. Results

In this section, I will share the detection results from the dynamic, lightweight, and heavyweight models. Across the majority of tested images, all of the models yield correct classifications and detections, with the incorrect ones being subtle and often tiny or hidden objects that are hard to see.

Figure 8 shows a simple image of a person surfing. The annotations mark a person and a surfboard as present in the image. All three models detect the person with extreme confidence, with the lightweight model failing to detect the surfboard. Granted, the surfboard blends in with the water, which is a big possibility for why the model might have missed that detection.

Figure 9 is an example of an image that can skew results as it contains over 20 objects that are tiny and next to each other, where the chances of a model grouping multiple of the same objects as one big detection. Each wine glass and book is individually annotated. Although the dynamic model misclassified some objects, the localization results remain strong. However, this still negatively affects mAP percentage.

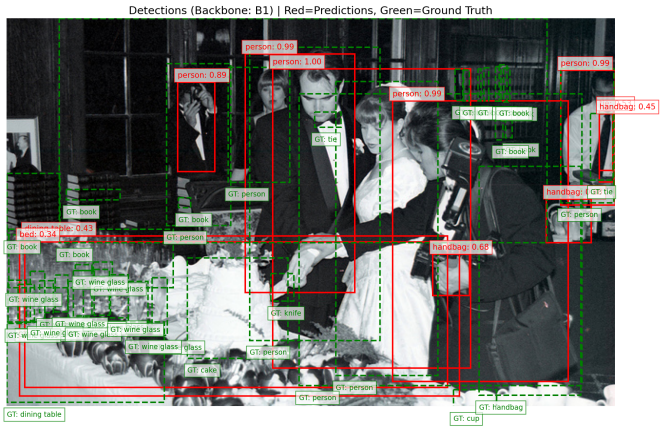


Figure 9. Example of an image with many annotations that are close together and tiny

6. Conclusion

Balancing trade-offs remains a huge challenge in computer vision, where it's hard for one model to fit all applications and situations. Dynamic models are a step in the right direction. The ability to exit in the middle of the pipeline

depending on the complexity of an image is extremely powerful and can save precious time and resources, especially when compute power is becoming more and more valuable as AI advances. The results in this paper demonstrate that it's possible to maintain high detection accuracy while minimizing latency. All in all, a dynamic model approach with real-time inference capabilities has proved to be proficient in accuracy and speed, creating new trade-off strategies.

References

[1] Bardia Ardakanian. Annotated 5k mscoco subset for object detection. https://www.kaggle.com/datasets/bardiaardakanian/mmsample, 2023. 6

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2, 5, 6

[3] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019. 6

[4] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. 2

[5] Tingting Liang, Xiaojie Chu, Yudong Liu, Yongtao Wang, Zhi Tang, Wei Chu, Jingdong Chen, and Haibin Ling. Cbnet: A composite backbone network architecture for object detection. *IEEE Transactions on Image Processing*, 31: 6893–6906, 2022. 2

[6] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. 1, 2, 4, 5, 6

[7] Zhihao Lin, Yongtao Wang, Jinhe Zhang, and Xiaojie Chu. Dynamicdet: A unified dynamic architecture for object detection, 2023. 1, 2, 3, 6

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016. 4

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. 6

[10] Huixin Sun, Runqi Wang, Yanjing Li, Linlin Yang, Shaohui Lin, Xianbin Cao, and Baochang Zhang. Set: Spectral enhancement for tiny object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4713–4723, 2025. 1, 3, 4

[11] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022. 1, 2

[12] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Dota: A large-scale dataset for object detection in aerial images, 2019. 4

[13] Chang Xu, Ruixiang Zhang, Wen Yang, Haoran Zhu, Fang Xu, Jian Ding, and Gui-Song Xia. Oriented tiny object detection: A dataset, benchmark, and dynamic unbiased learning, 2024. 4

[14] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. Detrs beat yolos on real-time object detection, 2024. 1, 4, 5, 6

[15] Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Heng Fan, Qinghua Hu, and Haibin Ling. Detection and tracking meet drones challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. 4