Towards Embodied Intelligence, A Comprehensive review of Computer Vision Architectures

Renan Monteiro Barbosa[1]
[1]University of West Florida
rmb54@students.uwf.edu

Abstract

Computer Visions is a fundamental part in order to develop the ability to predict future outcomes given control actions is fundamental for understanding reasoning and eventually developing Embodied Intelligence. This study will explore how YOLO has become a central real-time object detection system for robotics. Then we will present a comprehensive analysis of YOLO's evolution, examining the innovations and contributions in each iteration from the original YOLO with focus on YOLOv4. We will describe how lots of the concepts and insights developed with Yolo carried on towards more sophisticated methods using Transformers and beyond. We will discuss the major changes in network architecture and training tricks for each model using nvidia TAO toolkit. Finally, we summarize with evaluation of the model performance and how to make it run in production.

1 Introduction

Computer Vision is an essential part in embodied robotics and behavioral learning. In order to develop new multi-modal models necessary for embodied intelligence it has become clear that we need a holistic understanding of the YOLO and ViT framework's evolution and its implications for object detection, segmentation and more.

The constant and fast paced evolution of the field of computer vision shows that it is necessary to constantly update and review the evolution of these frameworks. There are overlaps and key insights that might go underscored in face of new improvements but if put into context might highlight potential avenues for further research and development towards foundational models for embodied robotics and behavioral learning.

This project aims to provide a comprehensive review of the YOLO and ViT framework's development elucidating the key innovations, differences, improvements and overlaps across each of them to further understand new development towards multi-modal models required of embodied intelligence.

This paper aims to provide a comprehensive review of the YOLO framework's development, from the original YOLOv1 to the latest YOLOv8, elucidating the key innovations, differences, and improvements across each version.

Each method presents unique tradeoffs between speed, accuracy, and complexity, catering to different application needs and computational constraints. These innovations are important to understand more modern architectures in special how YOLOv4 and later Vision models started to be described in three parts: Backbone, Neck, and Head

Other great reviews include [[3], [13], [31]], and [69]. However, these previous reviews as [3] only covers until YOLOv3, and [13] covers until YOLOv4 using outdated code that not only doesnt work in modern hardware, when it does it has performance issues and is unsuitable to production, the more recent review [69] does a great job at covering all models in detail and it has references to

Preprint. Under review.

useful codebases, but it is still not suitable for production and it cannot be implemented with Nvidia TAO as well it dont implement benchmarks such as MLperf and it doesnt focus on quantization and optimization techniques.

This work will focus on using development framework compatible with production deployments and it will run evaluation likewise.

2 Methodology

PyTorch

Why I am using torch, version of torch, ...

2.1 Development Framework

At the time of writting this paper the code was run using the Nvidia PyTorch container nvcr.io/nvidia/pytorch:24.10-py3

Devcontainer

Describe here the VSCode Devcontainers and how it streamlines the devlopment workflow

Docker Container

Describe here what is a docker container

Singularity Container

Describe here what is a singularity container

HPC High Peformnace Computing

The scaling requirements of SOTA models in special for the FOundational models require it to be trained on HPCs which are massive computer clusters.

NGC - Nvidia GPU Cloud

Pytorch NGC containers

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch

Contents of PyTorch Container

https://docs.nvidia.com/deeplearning/frameworks/pytorch-release-notes/rel-24-10.html

TensorRT NGC containers

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorrt/tags

2.2 NVIDIA TensorRT

NVIDIA® TensorRT™ is an SDK for high-performance deep learning inference. It is designed to work in a complementary fashion with training frameworks such as TensorFlow, PyTorch, and MXNet. It focuses specifically on running an already-trained network quickly and efficiently on NVIDIA hardware.

TensorRT includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for deep learning inference applications. The core of NVIDIA TensorRT is a C++ library that facilitates high-performance inference on NVIDIA GPUs. TensorRT takes a trained network, which consists of a network definition and a set of trained parameters, and produces a highly optimized runtime engine that performs inference for that network. Refer to the following TensorRT product documentation for more information.

2.3 NVIDIA Triton Inference Server

Triton Inference Server is an open source inference serving software that streamlines AI inferencing.

Triton Inference Server enables teams to deploy any AI model from multiple deep learning and machine learning frameworks, including TensorRT, TensorFlow, PyTorch, ONNX, OpenVINO, Python, RAPIDS FIL, and more. Triton supports inference across cloud, data center, edge and embedded devices on NVIDIA GPUs, x86 and ARM CPU, or AWS Inferentia.

Hardware and objectives

3 Model Evaluation

Models should be evaluated to assess their performance, identify strengths and weaknesses, and this goes beyond whether or not they are accurate at prediction but also metrics to evaluate their performance like inference time, cost per inference, power consumption, and more ....

For simplicity this study will focus on simpler image classification models which have well understood metrics and well established Datasets, they being COCO and KITTI (Autonomous Vehicle Dataset)

4 MLPerf benchmark

This section Needs work

Besides model prediction accuracy, it is also important to evaluate performance metrics like inference time, cost per inference, power consumption, and more ....

The foundation for MLCommons began with the MLPerf benchmarks in 2018, which rapidly scaled as a set of industry metrics to measure machine learning performance and promote transparency of machine learning techniques.

4.1 MLPerf Inference Benchmark

MLPerf Inference v4.1 [55]

MLPerf Inference Benchmark is a widely recognized set of standardized benchmarks used to evaluate the performance of inference across a variety of hardware platforms, including CPUs, GPUs, and edge devices.

It provides a consistent framework for measuring the speed and efficiency of ML models in real-world applications, covering tasks such as image classification, object detection, and language processing.

MLPerf helps to optimize performance in production environments, where low latency and high throughput are crucial.

4.2 MLPerf Training benchmark

MLPerf Training v4.1 [52]

The MLPerf Training benchmark suite measures how fast systems can train models to a target quality metric.

4.3 MLPerf HPC benchmark

MLPerf HPC v3.0

5 Object Detection Metrics and Non-Maximum Suppression (NMS)

The Average Precision (AP), traditionally called Mean Average Precision (mAP), is the commonly used metric for evaluating the performance of object detection models. It measures the average precision across all categories, providing a single value to compare different models. The COCO dataset makes no distinction between AP and mAP. In the rest of this paper, we will refer to this metric as AP.

In YOLOv1 and YOLOv2, the dataset utilized for training and benchmarking was PASCAL VOC 2007, and VOC 2012 [15]. However, from YOLOv3 onwards, the dataset used is Microsoft COCO

(Common Objects in Context) [43]. The AP is calculated differently for these datasets. The following sections will discuss the rationale behind AP and explain how it is computed.

5.1 How AP works?

The AP metric is based on precision-recall metrics, handling multiple object categories, and defining a positive prediction using Intersection over Union (IoU).

Precision and Recall: Precision measures the accuracy of the model's positive predictions, while recall measures the proportion of actual positive cases that the model correctly identifies. There is often a trade-off between precision and recall; for example, increasing the number of detected objects (higher recall) can result in more false positives (lower precision). To account for this trade-off, the AP metric incorporates the precision-recall curve that plots precision against recall for different confidence thresholds. This metric provides a balanced assessment of precision and recall by considering the area under the precision-recall curve.

Handling multiple object categories: Object detection models must identify and localize multiple object categories in an image. The AP metric addresses this by calculating each category's average precision (AP) separately and then taking the mean of these APs across all categories (that is why it is also called mean average precision). This approach ensures that the model's performance is evaluated for each category individually, providing a more comprehensive assessment of the model's overall performance.

Intersection over Union: Object detection aims to accurately localize objects in images by predicting bounding boxes. The AP metric incorporates the Intersection over Union (IoU) measure to assess the quality of the predicted bounding boxes. IoU is the ratio of the intersection area to the union area of the predicted bounding box and the ground truth bounding box (see Figure 3). It measures the overlap between the ground truth and predicted bounding boxes. The COCO benchmark considers multiple IoU thresholds to evaluate the model's performance at different levels of localization accuracy.

Need to ADD some images here to show the Intersection over Union

5.2 Computing AP

Microsoft COCO Dataset

This dataset includes 80 object categories and uses a more complex method for calculating AP. Instead of using an 11-point interpolation, it uses a 101-point interpolation, i.e., it computes the precision for 101 recall thresholds from 0 to 1 in increments of 0.01. Also, the AP is obtained by averaging over multiple IoU values instead of just one, except for a common AP metric called AP50, which is the AP for a single IoU threshold of 0.5. The steps for computing AP in COCO are the following:

1. For each category, calculate the precision-recall curve by varying the confidence threshold of the model's predictions.
2. Compute each category's average precision (AP) using 101-recall thresholds.
3. Calculate AP at different Intersection over Union (IoU) thresholds, typically from 0.5 to 0.95 with a step size of 0.05. A higher IoU threshold requires a more accurate prediction to be considered a true positive.
4. For each IoU threshold, take the mean of the APs across all 80 categories.
5. Finally, compute the overall AP by averaging the AP values calculated at each IoU threshold.

The differences in AP calculation make it hard to directly compare the performance of object detection models across the two datasets. The current standard uses the COCO AP due to its more fine-grained evaluation of how well a model performs at different IoU thresholds.

KITTI Dataset

The KITTI dataset is a widely used collection of data for research in autonomous driving and computer vision. It was created by the Karlsruhe Institute of Technology and Toyota Technological Institute, containing real-world driving scenes recorded with a variety of sensors, including high-resolution color cameras, LIDAR, and GPS/IMU systems. The dataset includes several benchmark tasks such as

stereo vision, optical flow, visual odometry, 3D object detection, and semantic segmentation. KITTI provides both the raw sensor data (images, point clouds, and GPS/IMU information) and ground truth annotations, making it a valuable resource for developing and evaluating algorithms for autonomous navigation and perception in dynamic environments.

5.3 Non-Maximum Suppression (NMS)

Non-Maximum Suppression (NMS) is a post-processing technique used in object detection algorithms to reduce the number of overlapping bounding boxes and improve the overall detection quality. Object detection algorithms typically generate multiple bounding boxes around the same object with different confidence scores. NMS filters out redundant and irrelevant bounding boxes, keeping only the most accurate ones. Algorithm 1 describes the procedure. Figure 4 shows the typical output of an object detection model containing multiple overlapping bounding boxes and the output after NMS.

6 YOLO ARCHITECTURES Review

The YOLO (You Only Look Once) framework has been a massive step forward in computer vision for its remarkable balance of speed and accuracy, enabling the real-time identification of objects in images and video. Below we will review the most popular and impactfull YOLO architectures.

7 YOLO: You Only Look Once

YOLO by Joseph Redmon et al. was published in CVPR 2016 [56]. It presented for the first time a real-time end-to-end approach for object detection. The name YOLO stands for "You Only Look Once," referring to the fact that it was able to accomplish the detection task with a single pass of the network, as opposed to previous approaches that either used sliding windows followed by a classifier that needed to run hundreds or thousands of times per image or the more advanced methods that divided the task into two-steps, where the first step detects possible regions with objects or regions proposals and the second step run a classifier on the proposals. Also, YOLO used a more straightforward output based only on regression to predict the detection outputs as opposed to Fast R-CNN [23] that used two separate outputs, a classification for the probabilities and a regression for the boxes coordinates.

We will be doing a brief introduction to the main YOLO architectures up to YOLOv4 where we will do a more in depth review with code implementation as the YOLOv4 is a junction point that introduces concepts that carry on.

7.1 YOLOv1 Architecture

YOLOv1 architecture comprises 24 convolutional layers followed by two fully-connected layers that predict the bounding box coordinates and probabilities. All layers used leaky rectified linear unit activations [51] except for the last one that used a linear activation function. Inspired by GoogLeNet [65] and Network in Network [40], YOLO uses 1×1 convolutional layers to reduce the number of feature maps and keep the number of parameters relatively low. As activation layers, Table 1 describes the YOLOv1 architecture. The authors also introduced a lighter model called Fast YOLO, composed of nine convolutional layers.

From the original paper:

Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 X 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 X 224 input image) and then double the resolution for detection.

8 YOLOv2: Better, Faster, and Stronger

YOLOv2 was published in CVPR 2017 [58] by Joseph Redmon and Ali Farhadi. It included several improvements over the original YOLO, to make it better, keeping the same speed and also stronger —capable of detecting 9000 categories!—. The improvements were the following:

1. **Batch normalization** on all convolutional layers improved convergence and acts as a regularizer to reduce overfitting.
2. **High-resolution classifier.** Like YOLOv1, they pre-trained the model with ImageNet at 224×224. However, this time, they finetuned the model for ten epochs on ImageNet with a resolution of 448×448, improving the network performance on higher resolution input.
3. **Fully convolutional.** They removed the dense layers and used a fully convolutional architecture.
4. **Use anchor boxes to predict bounding boxes.** They use a set of prior boxes or anchor boxes, which are boxes with predefined shapes used to match prototypical shapes of objects as shown in Figure 7. Multiple anchor boxes are defined for each grid cell, and the system predicts the coordinates and the class for every anchor box. The size of the network output is proportional to the number of anchor boxes per grid cell.
5. **Dimension Clusters.** Picking good prior boxes helps the network learn to predict more accurate bounding boxes. The authors ran k-means clustering on the training bounding boxes to find good priors. They selected five prior boxes providing a good tradeoff between recall and model complexity.
6. **Direct location prediction.** Unlike other methods that predicted offsets [59], YOLOv2 followed the same philosophy and predicted location coordinates relative to the grid cell. The network predicts five bounding boxes for each cell, each with five values tx, ty, tw, th, and to, where to is equivalent to Pc from YOLOv1 and the final bounding box coordinates are obtained as shown in Figure 8.
7. **Finner-grained features.** YOLOv2, compared with YOLOv1, removed one pooling layer to obtain an output feature map or grid of 13×13 for input images of 416×416. YOLOv2 also uses a passthrough layer that takes the $26 \times 26 \times 512$ feature map and reorganizes it by stacking adjacent features into different channels instead of losing them via a spatial subsampling. This generates $13 \times 13 \times 2048$ feature maps concatenated in the channel dimension with the lower resolution $13 \times 13 \times 1024$ maps to obtain $13 \times 13 \times 3072$ feature maps. See Table 2 for the architectural details.
8. **Multi-scale training.** Since YOLOv2 does not use fully connected layers, the inputs can be different sizes. To make YOLOv2 robust to different input sizes, the authors trained the model randomly, changing the input size —from 320×320 up to 608×608— every ten batches.

Double check, try to located YOLOv2 AP scores

With all these improvements, YOLOv2 achieved an average precision (AP) of 78.6 on the PASCAL VOC2007 dataset compared to the 63.4 obtained by YOLOv1.

8.1 YOLOv2 Architecture

The backbone architecture used by YOLOv2 is called Darknet-19, containing 19 convolutional layers and five maxpooling layers. Similar to the architecture of YOLOv1, it is inspired in the Network in Network [40] using 1×1 convolutions between the 3×3 to reduce the number of parameters. In addition, as mentioned above, they use batch normalization to regularize and help convergence.

Table 2 shows the entire Darknet-19 backbone with the object detection head. YOLOv2 predicts five bounding boxes, each with five values and 20 classes when using the PASCAL VOC dataset.

The object classification head replaces the last four convolutional layers with a single convolutional layer with 1000 filters, followed by a global average pooling layer and a Softmax.

8.2 YOLO9000 is a stronger YOLOv2

The authors introduced a method for training joint classification and detection in the same paper. It used the detection labeled data from COCO [43] to learn bounding box coordinates and classification data from ImageNet to increase the number of categories it can detect. During training, they combined both datasets such that when a detection training image is used, it backpropagates the detection network, and when a classification training image is used, it backpropagates the classification part of

the architecture. The result is a YOLO model capable of detecting more than 9000 categories hence the name YOLO9000.

9 YOLOv3

YOLOv3 [57] was published in ArXiv in 2018 by Joseph Redmon and Ali Farhadi. It included significant changes and a bigger architecture to be on par with the state-of-the-art while keeping real-time performance. In the following, we described the changes with respect to YOLOv2.

1. **Bounding box prediction.** Like YOLOv2, the network predicts four coordinates for each bounding box tx, ty, tw, and th; however, this time, YOLOv3 predicts an objectness score for each bounding box using logistic regression. This score is 1 for the anchor box with the highest overlap with the ground truth and 0 for the rest anchor boxes. YOLOv3, as opposed to Faster R-CNN [59], assigns only one anchor box to each ground truth object. Also, if no anchor box is assigned to an object, it only incurs in classification loss but not localization loss or confidence loss.
2. **Class Prediction.** Instead of using a softmax for the classification, they used binary cross-entropy to train independent logistic classifiers and pose the problem as a multilabel classification. This change allows assigning multiple labels to the same box, which may occur on some complex datasets [36] with overlapping labels. For example, the same object can be a Person and a Man.
3. **New backbone.** YOLOv3 features a larger feature extractor composed of 53 convolutional layers with residual connections. Section 6.1 describes the architecture in more detail.
4. **Spatial pyramid pooling (SPP).** Although not mentioned in the paper, the authors also added to the backbone a modified SPP block [27] that concatenates multiple max pooling outputs without subsampling (stride = 1), each with different kernel sizes $k \times k$ where $k = 1$, 5, 9, 13 allowing a larger receptive field. This version is called YOLOv3-spp and was the best-performed version improving the AP50 by 2.7.
5. **Multi-scale Predictions.** Similar to Feature Pyramid Networks [41], YOLOv3 predicts three boxes at three different scales. Section 6.2 describes the multi-scale prediction mechanism with more details.
6. **Bounding box priors.** Like YOLOv2, the authors also use k-means to determine the bounding box priors of anchor boxes. The difference is that in YOLOv2, they used a total of five prior boxes per cell, and in YOLOv3, they used three prior boxes for three different scales.

9.1 YOLOv3 Architecture

The architecture backbone presented in YOLOv3 is called Darknet-53. It replaced all max-pooling layers with strided convolutions and added residual connections. In total, it contains 53 convolutional layers. Figure 9 shows the architecture details.

The Darknet-53 backbone obtains Top-1 and Top-5 accuracies comparable with ResNet-152 but almost 2× faster.

10 Backbone, Neck, and Head

The concepts of Backbone, Neck and Head are important to understand how the advancements learned in YOLO will later apply into VITs going towards DINOv2 and other foundational visual models that give origin to World Models which are of extreme importance to implement embodied intelligence.

At this time, the architecture of object detectors started to be described in three parts: the backbone, the neck, and the head. Figure 11 shows a high-level backbone, neck, and head diagram.

The backbone is responsible for extracting useful features from the input image. It is typically a convolutional neural network (CNN) trained on a large-scale image classification task, such as ImageNet. The backbone captures hierarchical features at different scales, with lower-level features

(e.g., edges and textures) extracted in the earlier layers and higher-level features (e.g., object parts and semantic information) extracted in the deeper layers.

The neck is an intermediate component that connects the backbone to the head. It aggregates and refines the features extracted by the backbone, often focusing on enhancing the spatial and semantic information across different scales. The neck may include additional convolutional layers, feature pyramid networks (FPN) [41], or other mechanisms to improve the representation of the features.

The head is the final component of an object detector; it is responsible for making predictions based on the features provided by the backbone and neck. It typically consists of one or more task-specific subnetworks that perform classification, localization, and, more recently, instance segmentation and pose estimation. The head processes the features the neck provides, generating predictions for each object candidate. In the end, a post-processing step, such as non-maximum suppression (NMS), filters out overlapping predictions and retains only the most confident detections.

In the rest of the YOLO models, we will describe the architectures using the backbone, neck, and head.

11 YOLOv4

Two years passed, and there was no new version of YOLO. It was until April 2020 that Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao released in ArXiv the paper for YOLOv4 [4]. At first, it felt odd that different authors presented a new "official" version of YOLO; however, YOLOv4 kept the same YOLO philosophy —real-time, open source, single shot, and darknet framework— and the improvements were so satisfactory that the community rapidly embrace this version as the official YOLOv4.

YOLOv4 tried to find the optimal balance by experimenting with many changes categorized as bag-of-freebies and bag-of-specials. Bag-of-freebies are methods that only change the training strategy and increase training cost but do not increase the inference time, the most common being data augmentation. On the other hand, bag-of-specials are methods that slightly increase the inference cost but significantly improve accuracy. Examples of these methods are those for enlarging the receptive field [[27], [8], [46]], combining features [[28], [41], [25], [88]], and post-processing [[26], [51], [53], [5]] among others.

We summarize the main changes of YOLOv4 in the following points:

- **An Enhanced Architecture with Bag-of-Specials (BoS) Integration.** The authors tried multiple architectures for the backbone, such as ResNeXt50 [82], EfficientNet-B3 [67], and Darknet-53. The best-performing architecture was a modification of Darknet-53 with cross-stage partial connections (CSPNet) [74], and Mish activation function [53] as the backbone (see Figure 12. For the neck, they used the modified version of spatial pyramid pooling (SPP) [27] from YOLOv3-spp and multi-scale predictions as in YOLOv3, but with a modified version of path aggregation network (PANet) [45] instead of FPN as well as a modified spatial attention module (SAM) [79]. Finally, for the detection head, they use anchors as in YOLOv3. Therefore, the model was called CSPDarknet53-PANet-SPP. The cross-stage partial connections (CSP) added to the Darknet-53 help reduce the computation of the model while keeping the same accuracy. The SPP block, as in YOLOv3-spp increases the receptive field without affecting the inference speed. The modified version ofPANet concatenates the features instead of adding them as in the original PANet paper.

- **Integrating bag-of-freebies (BoF) for an Advanced Training Approach.** Apart from the regular augmentations such as random brightness, contrast, scaling, cropping, flipping, and rotation, the authors implemented mosaic augmentation that combines four images into a single one allowing the detection of objects outside their usual context and also reducing the need for a large mini-batch size for batch normalization. For regularization, they used DropBlock [22] that works as a replacement of Dropout [64] but for convolutional neural networks as well as class label smoothing [[66], [33]]. For the detector, they added CIoU loss [89] and Cross mini-bath normalization (CmBN) for collecting statistics from the entire batch instead of from single mini-batches as in regular batch normalization [32].

- **Self-adversarial Training (SAT).** To make the model more robust to perturbations, an adversarial attack is performed on the input image to create a deception that the ground truth object is not in the image but keeps the original label to detect the correct object.
- **Hyperparameter Optimization with Genetic Algorithms.** To find the optimal hyperparameters used for training, they use genetic algorithms on the first 10 of periods, and a cosine annealing scheduler [49] to alter the learning rate during training. It starts reducing the learning rate slowly, followed by a quick reduction halfway through the training process ending with a slight reduction.

Evaluated on MS COCO dataset test-dev 2017, YOLOv4 achieved an AP of 43.5 and AP50 of 65.7 at more than 50 FPS on an NVIDIA V100.

11.1 YOLOv4 Architecture

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

11.2 YOLOv4 Performance Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

12 YOLOv5

!!!! PLACE HOLDER !!!!

YOLOv5 [35] was released a couple of months after YOLOv4 in 2020 by Glen Jocher, founder and CEO of Ultralytics. It uses many improvements described in the YOLOv4 section but developed in Pytorch instead of Darknet. YOLOv5 incorporates an Ultralytics algorithm called AutoAnchor. This pre-training tool checks and adjusts anchor boxes if they are ill-fitted for the dataset and training settings, such as image size. It first applies a k-means function to dataset labels to generate initial conditions for a Genetic Evolution (GE) algorithm. The GE algorithm then evolves these anchors over 1000 generations by default, using CIoU loss [89] and Best Possible Recall as its fitness function. Figure 13 shows the detailed architecture of YOLOv5.

12.1 YOLOv5 Architecture

The backbone is a modified CSPDarknet53 that starts with a Stem, a strided convolution layer with a large window size to reduce memory and computational costs; followed by convolutional layers that extract relevant features from the input image. The SPPF (spatial pyramid pooling fast) layer and the following convolution layers process the features at various scales, while the upsample layers increase the resolution of the feature maps. The SPPF layer aims to speed up the computation of the network by pooling features of different scales into a fixed-size feature map. Each convolution is followed by batch normalization (BN) and SiLU activation [29]. The neck uses SPPF and a modified CSP-PAN, while the head resembles YOLOv3.

YOLOv5 uses several augmentations such as Mosaic, copy paste [21], random affine, MixUp [86], HSV augmentation, random horizontal flip, as well as other augmentations from the albumentations package [6]. It also improves the grid sensitivity to make it more stable to runaway gradients.

YOLOv5 provides five scaled versions: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra large), where the width and depth of the convolution modules vary to suit specific applications and hardware requirements. For instance, YOLOv5n and YOLOv5s

are lightweight models targeted for low-resource devices, while YOLOv5x is optimized for high performance, albeit at the expense of speed.

The YOLOv5 released version at the time of this writing is v7.0, including YOLOv5 versions capable of classification and instance segmentation.

YOLOv5 is open source and actively maintained by Ultralytics, with more than 250 contributors and new improvements frequently. YOLOv5 is easy to use, train and deploy. Ultralytics provide a mobile version for iOS and Android and many integrations for labeling, training, and deployment.

Evaluated on MS COCO dataset test-dev 2017, YOLOv5x achieved an AP of 50.7 with an image size of 640 pixels. Using a batch size of 32, it can achieve a speed of 200 FPS on an NVIDIA V100. Using a larger input size of 1536 pixels and test-time augmentation (TTA), YOLOv5 achieves an AP of 55.8.

12.2 YOLOv5 Performance Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

13 Scaled-YOLOv4

One year after YOLOv4, the same authors presented Scaled-YOLOv4 [72] in CVPR 2021. Differently from YOLOv4, Scaled YOLOv4 was developed in Pytorch instead of Darknet. The main novelty was the introduction of scaling-up and scaling-down techniques. Scaling up means producing a model that increases accuracy at the expense of a lower speed; on the other hand, scaling down entails producing a model that increases speed sacrificing accuracy. In addition, scaled-down models need less computing power and can run on embedded systems.

The scaled-down architecture was called YOLOv4-tiny; it was designed for low-end GPUs and can run at 46 FPS on a Jetson TX2 or 440 FPS on RTX2080Ti, achieving 22 AP on MS COCO.

The scaled-up model architecture was called YOLOv4-large, which included three different sizes P5, P6, and P7. This architecture was designed for cloud GPU and achieved state-of-the-art performance, surpassing all previous models [[68], [42], [48]] with 56 AP on MS COCO.

13.1 Scaled-YOLOv4 Architecture

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

13.2 Scaled-YOLOv4 Performance Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

14 YOLOR

YOLOR [76] was published in ArXiv in May 2021 by the same research team of YOLOv4. YOLOR stands for **You Only Learn One Representation**. In this paper, the authors followed a different approach; they developed a multi-task learning approach that aims to create a single model for various

tasks (e.g., classification, detection, pose estimation) by learning a general representation and using sub-networks to create task-specific representations. With the insight that the traditional joint learning method often leads to suboptimal feature generation, YOLOR aims to overcome this by encoding the implicit knowledge of neural networks to be applied to multiple tasks, similar to how humans use past experiences to approach new problems. The results showed that introducing implicit knowledge into the neural network benefits all the tasks.

Evaluated on MS COCO dataset test-dev 2017, YOLOR achieved a AP of 55.4 and AP50 of 73.3 at 30 FPS on an NVIDIA V100.

14.1 YOLOR Architecture

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

14.2 YOLOR Performance Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

15 YOLOX

YOLOX [18] was published in ArXiv in July 2021 by Megvii Technology. Developed in Pytorch and using YOLOV3 from Ultralytics as starting point, it has five principal changes: an anchor-free architecture, multiple positives, a decoupled head, advanced label assignment, and strong augmentations. It achieved state-of-the-art results in 2021 with an optimal balance between speed and accuracy with 50.1 AP at 68.9 FPS on Tesla V100. In the following, we describe the five main changes of YOLOX with respect to YOLOv3:

1. **Anchor-free.** Since YOLOv2, all subsequent YOLO versions were anchor-based detectors. YOLOX, inspired by anchor-free state-of-the-art object detectors such as CornerNet [37], CenterNet [14], and FCOS [70], returned to an anchor-free architecture simplifying the training and decoding process. The anchor-free increased the AP by 0.9 points concerning the YOLOv3 baseline.

2. **Multi positives.** To compensate for the large imbalances the lack of anchors produced, the authors use center sampling [70] where they assigned the center 3×3 area as positives. This approach increased AP by 2.1 points.

3. **Decoupled head.** In [[63], [80]], it was shown that there could be a misalignment between the classification confidence and localization accuracy. Due to this, YOLOX separates these two into two heads (as shown in Fig. 14), one for classification tasks and the other for regression tasks improving the AP by 1.1 points and speeding up the model convergence.

4. **Advanced label assignment.** In [19], it was shown that the ground truth label assignment could have ambiguities when the boxes of multiple objects overlap and formulate the assigning procedure as an Optimal Transport (OT) problem. YOLOX, inspired by this work, proposed a simplified version called simOTA. This change increased AP by 2.3 points.

5. **Strong augmentations.** YOLOX uses MixUP [86] and Mosaic augmentations. The authors found that ImageNet pretraining was no longer beneficial after using these augmentations. The strong augmentations increased AP by 2.4 points.

15.1 YOLOX Architecture

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

15.2 YOLOX Performance Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

16 YOLOv6

YOLOv6 [38] was published in ArXiv in September 2022 by Meituan Vision AI Department. The network design consists of an efficient backbone with RepVGG or CSPStackRep blocks, a PAN topology neck, and an efficient decoupled head with a hybrid-channel strategy. In addition, the paper introduces enhanced quantization techniques using post-training quantization and channel-wise distillation, resulting in faster and more accurate detectors. Overall, YOLOv6 outperforms previous state-of-the-art models on accuracy and speed metrics, such as YOLOv5, YOLOX, and PP-YOLOE.

Figure 15 shows the detailed architecture of YOLOv6.

The main novelties of this model are summarized below:

1. **A new backbone based on RepVGG** [12] called EfficientRep that uses higher parallelism than previous YOLO backbones. For the neck, they use PAN [45] enhanced with RepBlocks [12] or CSPStackRep [74] Blocks for the larger models. And following YOLOX, they developed an efficient decoupled head.
2. **Label assignment** using the Task alignment learning approach introduced in TOOD [17].
3. **New classification and regression losses.** They used a classification VariFocal loss [85] and an SIoU [20] /GIoU [60] regression loss.
4. **A self-distillation** strategy for the regression and classification tasks.
5. **A quantization scheme** for detection using RepOptimizer [11] and channel-wise distillation [62] that helped to achieve a faster detector.

The authors provide eight scaled models, from YOLOv6-N to YOLOv6-L6. Evaluated on MS COCO dataset test-dev 2017, the largest model, achieved an AP of 57.2 at around 29 FPS on an NVIDIA Tesla T4.

16.1 YOLOv6 Architecture

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

16.2 YOLOv6 Performance Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

17 YOLOv7

YOLOv7 [73] was published in ArXiv in July 2022 by the same authors of YOLOv4 and YOLOR. At the time, it surpassed all known object detectors in speed and accuracy in the range of 5 FPS to 160 FPS. Like YOLOv4, it was trained using only the MS COCO dataset without pre-trained backbones. YOLOv7 proposed a couple of architecture changes and a series of bag-of-freebies, which increased the accuracy without affecting the inference speed, only the training time.

Figure 16 shows the detailed architecture of YOLOv7.

The architecture changes of YOLOv7 are:

- **Extended efficient layer aggregation network (E-ELAN).** ELAN [75] is a strategy that allows a deep model to learn and converge more efficiently by controlling the shortest longest gradient path. YOLOv7 proposed E-ELAN that works for models with unlimited stacked computational blocks. E-ELAN combines the features of different groups by shuffling and merging cardinality to enhance the network's learning without destroying the original gradient path.
- **Model scaling for concatenation-based models.** Scaling generates models of different sizes by adjusting some model attributes. The architecture of YOLOv7 is a concatenation-based architecture in which standard scaling techniques, such as depth scaling, cause a ratio change between the input channel and the output channel of a transition layer which, in turn, leads to a decrease in the hardware usage of the model. YOLOv7 proposed a new strategy for scaling concatenation-based models in which the depth and width of the block are scaled with the same factor to maintain the optimal structure of the model.

The bag-of-freebies used in YOLOv7 include:

- **Planned re-parameterized convolution.** Like YOLOv6, the architecture of YOLOv7 is also inspired by re-parameterized convolutions (RepConv) [12]. However, they found that the identity connection in RepConv destroys the residual in ResNet [28] and the concatenation in DenseNet [30]. For this reason, they removed the identity connection and called it RepConvN.
- **Coarse label assignment for auxiliary head and fine label assignment for the lead head.** The lead head is responsible for the final output, while the auxiliary head assists with the training.
- **Batch normalization in conv-bn-activation.** This integrates the mean and variance of batch normalization into the bias and weight of the convolutional layer at the inference stage.
- **Implicit knowledge** inspired in YOLOR [76].
- **Exponential moving average** as the final inference model.

17.1 Comparison with YOLOv4 and YOLOR

In this section, we highlight the enhancements of YOLOv7 compared to previous YOLO models developed by the same authors.

Compared to YOLOv4, YOLOv7 achieved a 75 reduction in parameters and a 36 reduction in computation while simultaneously improving the average precision (AP) by 1.5.

In contrast to YOLOv4-tiny, YOLOv7-tiny managed to reduce parameters and computation by 39 and 49, respectively, while maintaining the same AP.

Lastly, compared to YOLOR, YOLOv7 reduced the number of parameters and computation by 43 and 15, respectively, along with a slight 0.4 increase in AP.

Evaluated on MS COCO dataset test-dev 2017, YOLOv7-E6 achieved an AP of 55.9 and AP50 of 73.5 with an input size of 1280 pixels with a speed of 50 FPS on an NVIDIA V100.

17.2 YOLOv7 Architecture

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

17.3 YOLOv7 Performance Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

18 DAMO-YOLO

The MAE-NAS citation for the TinyNAS from Alibaba needs fixing.

DAMO-YOLO [84] was published in ArXiv in November 2022 by Alibaba Group. Inspired by the current technologies, DAMO-YOLO included the following:

1. **A Neural architecture search (NAS).** They used a method called MAE-NAS [2] developed by Alibaba to find an efficient architecture automatically.
2. **A large neck.** Inspired by GiraffeDet [34], CSPNet [74], and ELAN [75], the authors designed a neck that can work in real-time called Efficient-RepGFPN.
3. **A small head.** The authors found that a large neck and a small neck yield better performance, and they only left one linear layer for classification and one for regression. They called this approach ZeroHead.
4. **AlignedOTA label assignment.** Dynamic label assignment methods, such as OTA [19] and TOOD [17], have gained popularity due to their significant improvements over static methods. However, the misalignment between classification and regression remains a problem, partly because of the imbalance between classification and regression losses. To address this issue, their AlignOTA method introduces focal loss [42] into the classification cost and uses the IoU of prediction and ground truth box as the soft label, enabling the selection of aligned samples for each target and solving the problem from a global perspective.
5. **Knowledge distillation.** Their proposed strategy consists of two stages: the teacher guiding the student in the first stage and the student fine-tuning independently in the second stage. Additionally, they incorporate two enhancements in the distillation approach: the Align Module, which adapts student features to the same resolution as the teacher's, and Channel-wise Dynamic Temperature, which normalizes teacher and student features to reduce the impact of real value differences.

The authors generated scaled models named DAMO-YOLO-Tiny/Small/Medium, with the best model achieving an AP of 50.0 at 233 FPS on an NVIDIA V100.

19 YOLOv8

Need to create a citation for the ultralytics github repo, however it is crashing bibtex

YOLOv8 [113] was released in January 2023 by Ultralytics, the company that developed YOLOv5. YOLOv8 provided five scaled versions: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large) and YOLOv8x (extra large). YOLOv8 supports multiple vision tasks such as object detection, segmentation, pose estimation, tracking, and classification.

19.1 YOLOv8 Architecture

Figure 17 shows the detailed architecture of YOLOv8. YOLOv8 uses a similar backbone as YOLOv5 with some changes on the CSPLayer, now called the C2f module. The C2f module (cross-stage partial bottleneck with two convolutions) combines high-level features with contextual information to improve detection accuracy.

YOLOv8 uses an anchor-free model with a decoupled head to independently process objectness, classification, and regression tasks. This design allows each branch to focus on its task and improves the model's overall accuracy. In the output layer of YOLOv8, they used the sigmoid function as the activation function for the objectness score, representing the probability that the bounding box contains an object. It uses the softmax function for the class probabilities, representing the objects' probabilities belonging to each possible class.

YOLOv8 uses CIoU [89] and DFL [39] loss functions for bounding box loss and binary cross-entropy for classification loss. These losses have improved object detection performance, particularly when dealing with smaller objects.

YOLOv8 also provides a semantic segmentation model called YOLOv8-Seg model. The backbone is a CSPDarknet53 feature extractor, followed by a C2f module instead of the traditional YOLO neck architecture. The C2f module is followed by two segmentation heads, which learn to predict the semantic segmentation masks for the input image. The model has similar detection heads to YOLOv8, consisting of five detection modules and a prediction layer. The YOLOv8-Seg model has achieved state-of-the-art results on various object detection and semantic segmentation benchmarks while maintaining high speed and efficiency.

YOLOv8 can be run from the command line interface (CLI), or it can also be installed as a PIP package. In addition, it comes with multiple integrations for labeling, training, and deploying.

Evaluated on MS COCO dataset test-dev 2017, YOLOv8x achieved an AP of 53.9 with an image size of 640 pixels (compared to 50.7 of YOLOv5 on the same input size) with a speed of 280 FPS on an NVIDIA A100 and TensorRT.

20 PP-YOLO, PP-YOLOv2, and PP-YOLOE

PP-YOLO models have been growing parallel to the YOLO models we described. However, we decided to group them in a single section because they began with YOLOv3 and had been gradually improving upon the previous PP-YOLO version. Nevertheless, these models have been influential in the evolution of YOLO. PP-YOLO [48] similar to YOLOv4 and YOLOv5 was based on YOLOv3. It was published in ArXiv in July 2020 by researchers from Baidu Inc. The authors used the PaddlePaddle [50] deep learning platform, hence its PP name. Following the trend we have seen starting with YOLOv4, PP-YOLO added ten existing tricks to improve the detector's accuracy, keeping the speed unchanged. According to the authors, this paper was not intended to introduce a novel object detector but to show how to build a better detector step by step. Most of the tricks PP-YOLO uses are different from the ones used in YOLOv4, and the ones that overlap use a different implementation. The changes of PP-YOLO concerning YOLOv3 are:

1. **A ResNet50-vd backbone** replacing the DarkNet-53 backbone with an architecture augmented with deformable convolutions [10] in the last stage and a distilled pre-trained model, which has a higher classification accuracy on ImageNet. This architecture was called ResNet5-vd-dcn.
2. **A larger batch size** to improve training stability, they went from 64 to 192, along with an updated training schedule and learning rate.
3. **Maintained moving averages** for the trained parameters and use them instead of the final trained values.
4. **DropBlock** is applied only to the FPN.
5. **An IoU loss** is added in another branch along with the L1-loss for bounding box regression.
6. **An IoU prediction branch** is added to measure localization accuracy along with an IoU aware loss. During inference, YOLOv3 multiplies the classification probability and objec-

tiveness score to compute the final detection, PP-YOLO also multiplies the predicted IoU to consider the localization accuracy.

7. **Grid Sensitive approach** similar to YOLOv4 is used to improve the bounding box center prediction at the grid boundary.
8. **Matrix NMS** [83] is used, which can be run in parallel making it faster than traditional NMS.
9. **CoordConv** [44] is used for the 1×1 convolution of the FPN, and on the first convolution layer in the detection head. CoordConv allows the network to learn translational invariance improving the detection localization.
10. **Spatial Pyramid Pooling** is used only on the top feature map to increase the receptive field of the backbone.

20.1 PP-YOLO augmentations and preprocessing

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

20.2 PP-YOLOv2

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

20.3 PP-YOLOE

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

21 YOLO-NAS

YOLO-NAS [124] was released in May 2023 by Deci, a company that develops production-grade models and tools to build, optimize, and deploy deep learning models. YOLO-NAS is designed to detect small objects, improve localization accuracy, and enhance the performance-per-compute ratio, making it suitable for real-time edge-device applications. In addition, its open-source architecture is available for research use.

The novelty of YOLO-NAS includes the following:

- Quantization aware modules [9] called QSP and QCI that combine re-parameterization for 8-bit quantization to minimize the accuracy loss during post-training quantization.
- Automatic architecture design using AutoNAC, Deci's proprietary NAS technology.
- Hybrid quantization method to selectively quantize certain parts of a model to balance latency and accuracy instead of standard quantization, where all the layers are affected.
- A pre-training regimen with automatically labeled data, self-distillation, and large datasets.

The AutoNAC system, which was instrumental in creating YOLO-NAS, is versatile and can accommodate any task, the specifics of the data, the environment for making inferences, and the setting of performance goals. It assists users in identifying the most suitable structure that offers the perfect

blend of precision and inference speed for their particular use. This technology considers the data and hardware and other elements involved in the inference process, such as compilers and quantization. In addition, RepVGG blocks were incorporated into the model architecture during the NAS process for compatibility with Post-Training Quantization (PTQ). They generated three architectures by varying the depth and positions of the QSP and QCI blocks: YOLO-NASS, YOLO-NASM, and YOLO-NASL (S,M,L for small, medium, and large, respectively). Figure 19 shows the model architecture for YOLO-NASL.

The model is pre-trained on Objects365 [61], which contains two million images and 365 categories, then the COCO dataset was used to generate pseudo-labels. Finally, the models are trained with the original 118k train images of the COCO dataset.

At this writing, three YOLO-NAS models have been released in FP32, FP16, and INT8 precisions, achieving an AP of 52.2 on MS COCO with 16-bit precision.

22 YOLO with Transformers

This section is questionable as there are so many divergent studies implementing YOLO like models with transformers.

With the rise of the Transformer [71] taking over most Deep Learning tasks from Language and Audio Processing to Vision, it was natural for Transformers and YOLO to be combined. One of the first attempts at using transformers for object detection was You Only Look at One Sequence or YOLOS [16], turned a pre-trained Vision Transfomer (ViT) [1] from image classification to object detection, achieving 42.0 AP on MS COCO dataset. The changes made to ViT were two: 1) replace one [CLS] token used in classification with one hundred [DET] tokens for detection, and 2) replace the image classification loss in ViT with a bipartite matching loss similar to the End-to-end object detection with transformers [7].

Many works have combined transformers with YOLO-related architectures tailored to specific applications. For example, Zhang et al. [87], motivated by the robustness of Vision Transformers to occlusions, perturbations, and domain shifts, proposed ViT-YOLO, a hybrid architecture that combines CSP-Darknet [4] and multi-head self-attention (MHSA-Darknet) in the backbone along with bidirectional feature pyramid networks (BiFPN) [68] for the neck and multi-scale detection heads like YOLOv3. Their specific use case was for object detection in drone images. Figure 20 shows the detailed architecture of ViT-YOLO.

MSFT-YOLO [24] adds transformer-based modules to the backbone and detection heads intending to detect defects on the steel surface. NRT-YOLO [47] (Nested Residual Transformer) tries to address the problem of tiny objects in remote sensing images. Adding an extra prediction head, feature fusion layers, and a residual transformer module, NRT-YOLO improved YOLOv5l by 5.4 in the DOTA dataset [81].

In remote sensing applications, YOLO-SD [77] tried to improve the detection accuracy for small ships in synthetic aperture radar (SAR) images. They started with YOLOX [18] coupled with multi-scale convolution (MSC) to improve the detection at different scales and feature transformer modules to capture global features. The authors showed that these changes improved the accuracy of YOLO-SD compared with YOLOX in the HRSID dataset [78].

Another interesting attempt to combine YOLO with detection transformer (DETR) [7] is the case of DEYO [54] comprising two stages: a YOLOv5-based model followed by a DETR-like model. The first stage generates high-quality query and anchors that input to the second stage. The results show a faster convergence time and better performance than DETR, achieving 52.1 AP in the COCO detection benchmark.

23 Discussions

The YOLO models have significantly influenced the field of computer vision. The clever architecture is its core strength, it allows it to simultaneously predict the location and classification of objects in an image, making it extremely. This end-to-end approach allowed YOLO models to process images in a single forward pass, enabling real-time applications such as autonomous driving, surveillance,

and robotics. The innovation behind YOLO's speed and efficiency has made it a foundational piece towards multi-modal computer vision systems.

In summary, YOLO models, with their real-time detection capability, inspired the backbone (pun intended) of modern computer vision systems aimed at predicting and controlling future outcomes. As these models continue to evolve, their integration with predictive models and control frameworks will be key to the advancement of intelligent systems that can both understand and predict dynamic environments, ultimately pushing the boundaries of what autonomous systems can achieve.

24 Acknowledgments

Use unnumbered third level headings for the acknowledgments. All acknowledgments go at the end of the paper. Do not include acknowledgments in the anonymized submission, only in the final paper. This example was prepared by Dennis Núñez Fernández.

References

[1] Dosovitskiy Alexey. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv: 2010.11929*, 2020.

[2] Alibaba. TinyNAS, May 2020.

[3] B Bhavya Sree, V Yashwanth Bharadwaj, and N Neelima. An inter-comparative survey on state-of-the-art detectors—r-cnn, yolo, and ssd. In *Intelligent Manufacturing and Energy Sustainability: Proceedings of ICIMES 2020*, pages 475–483. Springer, 2021.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[5] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017.

[6] Alexander Buslaev, Vladimir I Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A Kalinin. Albumentations: fast and flexible image augmentations. *Information*, 11(2):125, 2020.

[7] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[9] Xiangxiang Chu, Liang Li, and Bo Zhang. Make repvgg greater again: A quantization-aware approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11624–11632, 2024.

[10] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.

[11] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Kaiqi Huang, Jungong Han, and Guiguang Ding. Re-parameterizing your optimizers rather than architectures. *arXiv preprint arXiv:2205.15242*, 2022.

[12] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021.

[13] Tausif Diwan, G Anirudh, and Jitendra V Tembhurne. Object detection using yolo: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, 82(6):9243–9275, 2023.

[14] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6569–6578, 2019.

[15] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

[16] Yuxin Fang, Bencheng Liao, Xinggang Wang, Jiemin Fang, Jiyang Qi, Rui Wu, Jianwei Niu, and Wenyu Liu. You only look at one sequence: Rethinking transformer in vision through object detection. *Advances in Neural Information Processing Systems*, 34:26183–26197, 2021.

[17] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R Scott, and Weilin Huang. Tood: Task-aligned one-stage object detection. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3490–3499. IEEE Computer Society, 2021.

[18] Z Ge. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[19] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and Jian Sun. Ota: Optimal transport assignment for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 303–312, 2021.

[20] Zhora Gevorgyan. Siou loss: More powerful learning for bounding box regression. *arXiv preprint arXiv:2205.12740*, 2022.

[21] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2918–2928, 2021.

[22] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. *Advances in neural information processing systems*, 31, 2018.

[23] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[24] Zexuan Guo, Chensheng Wang, Guang Yang, Zeyuan Huang, and Guo Li. Msft-yolo: Improved yolov5 based on transformer for detecting defects of steel surface. *Sensors*, 22(9):3467, 2022.

[25] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 447–456, 2015.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[29] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[30] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[31] Muhammad Hussain. Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11(7):677, 2023.

[32] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[33] Md Amirul Islam, Shujon Naha, Mrigank Rochan, Neil Bruce, and Yang Wang. Label refinement network for coarse-to-fine semantic segmentation. *arXiv preprint arXiv:1703.00551*, 2017.

[34] Yiqi Jiang, Zhiyu Tan, Junyan Wang, Xiuyu Sun, Ming Lin, and Hao Li. Giraffedet: A heavy-neck paradigm for object detection. *arXiv preprint arXiv:2202.04256*, 2022.

[35] Glenn Jocher. YOLOv5 by Ultralytics, May 2020.

[36] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Andreas Veit, et al. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github. com/openimages*, 2(3):18, 2017.

[37] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European conference on computer vision (ECCV)*, pages 734–750, 2018.

[38] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. Yolov6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022.

[39] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. *Advances in Neural Information Processing Systems*, 33:21002–21012, 2020.

[40] M Lin. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[41] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[42] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[43] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[44] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *Advances in neural information processing systems*, 31, 2018.

[45] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[46] Songtao Liu, Di Huang, et al. Receptive field block net for accurate and fast object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 385–400, 2018.

[47] Yukuan Liu, Guanglin He, Zehu Wang, Weizhe Li, and Hongfei Huang. Nrt-yolo: Improved yolov5 based on nested residual transformer for tiny remote sensing object detection. *Sensors*, 22(13):4953, 2022.

[48] Xiang Long, Kaipeng Deng, Guanzhong Wang, Yang Zhang, Qingqing Dang, Yuan Gao, Hui Shen, Jianguo Ren, Shumin Han, Errui Ding, et al. Pp-yolo: An effective and efficient implementation of object detector. *arXiv preprint arXiv:2007.12099*, 2020.

[49] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[50] Yanjun Ma, Dianhai Yu, Tian Wu, and Haifeng Wang. Paddlepaddle: An open-source deep learning platform from industrial practice. *Frontiers of Data and Domputing*, 1(1):105–115, 2019.

[51] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[52] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David A. Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debojyoti Dutta, Udit Gupta, Kim M. Hazelwood, Andrew Hock, Xinyuan Huang, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. Mlperf training benchmark. *CoRR*, abs/1910.01500, 2019.

[53] Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[54] Haodong Ouyang. Deyo: Detr with yolo for step-by-step object detection. *arXiv preprint arXiv:2211.06588*, 2022.

[55] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark. *CoRR*, abs/1911.02549, 2019.

[56] J Redmon. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[57] Joseph Redmon. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[58] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[59] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 91–99, Cambridge, MA, USA, 2015. MIT Press.

[60] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 658–666, 2019.

[61] Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. Objects365: A large-scale, high-quality dataset for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8430–8439, 2019.

[62] Changyong Shu, Yifan Liu, Jianfei Gao, Zheng Yan, and Chunhua Shen. Channel-wise knowledge distillation for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5311–5320, 2021.

[63] Guanglu Song, Yu Liu, and Xiaogang Wang. Revisiting the sibling head in object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11563–11572, 2020.

[64] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[65] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[66] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[67] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[68] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020.

[69] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, 2023.

[70] Z Tian, C Shen, H Chen, and T He. Fcos: Fully convolutional one-stage object detection. arxiv 2019. *arXiv preprint arXiv:1904.01355*, 2019.

[71] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[72] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 13029–13038, 2021.

[73] 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. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023.

[74] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

[75] Chien-Yao Wang, Hong-Yuan Mark Liao, and I-Hau Yeh. Designing network design strategies through gradient path analysis. *arXiv preprint arXiv:2211.04800*, 2022.

[76] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. You only learn one representation: Unified network for multiple tasks. *arXiv preprint arXiv:2105.04206*, 2021.

[77] Simin Wang, Song Gao, Lun Zhou, Ruochen Liu, Hengsheng Zhang, Jiaming Liu, Yong Jia, and Jiang Qian. Yolo-sd: Small ship detection in sar images by multi-scale convolution and feature transformer module. *Remote Sensing*, 14(20):5268, 2022.

[78] Shunjun Wei, Xiangfeng Zeng, Qizhe Qu, Mou Wang, Hao Su, and Jun Shi. Hrsid: A high-resolution sar images dataset for ship detection and instance segmentation. *Ieee Access*, 8:120234–120254, 2020.

[79] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[80] Yue Wu, Yinpeng Chen, Lu Yuan, Zicheng Liu, Lijuan Wang, Hongzhi Li, and Yun Fu. Rethinking classification and localization for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10186–10195, 2020.

[81] 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. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3974–3983, 2018.

[82] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[83] Wang Xinlong, Zhang Rufeng, Kong Tao, Li Lei, and Shen Chunhua. Solov2: Dynamic, faster and stronger. In *Proc. NIPS*, 2020.

[84] Xianzhe Xu, Yiqi Jiang, Weihua Chen, Yilun Huang, Yuan Zhang, and Xiuyu Sun. Damo-yolo: A report on real-time object detection design. *arXiv preprint arXiv:2211.15444*, 2022.

[85] Haoyang Zhang, Ying Wang, Feras Dayoub, and Niko Sunderhauf. Varifocalnet: An iou-aware dense object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8514–8523, 2021.

[86] Hongyi Zhang. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[87] Zixiao Zhang, Xiaoqiang Lu, Guojin Cao, Yuting Yang, Licheng Jiao, and Fang Liu. Vit-yolo: Transformer-based yolo for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2799–2808, 2021.

[88] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 9259–9266, 2019.

[89] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020.