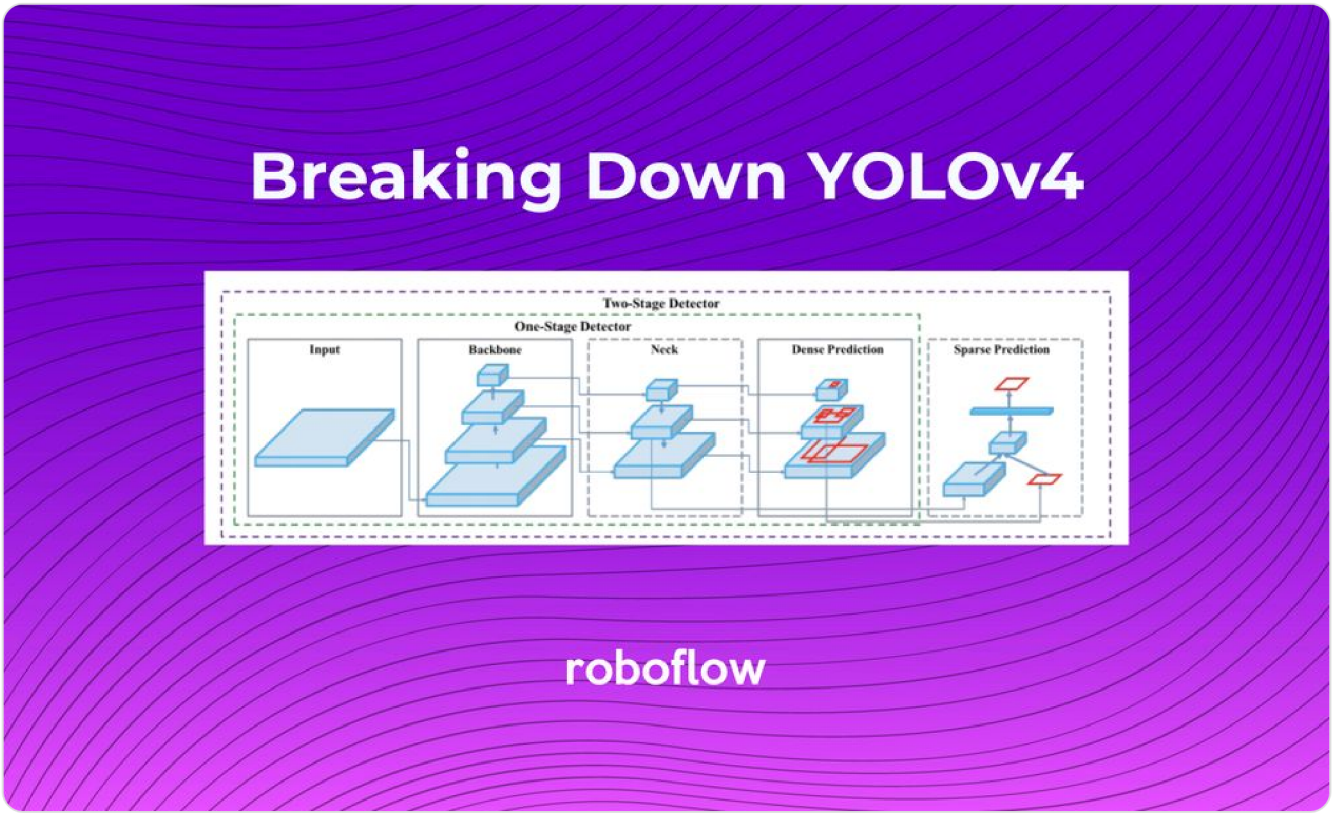
EDUCATION

What is YOLOv4? A Detailed Breakdown.

Jacob Solawetz
JUN 4, 2020 10 MIN READ



The realtime object detection space remains hot and moves ever forward with the publication of YOLO v4. Relative to inference speed, YOLOv4 outperforms other object detection models by a significant margin.

We have recently been amazed at the performance of YOLOv4 on **custom object detection** tasks and have published tutorials on **how to train YOLOv4 in Darknet** and **how to train YOLOv4 in PyTorch**.

In this article, we're going to talk about:

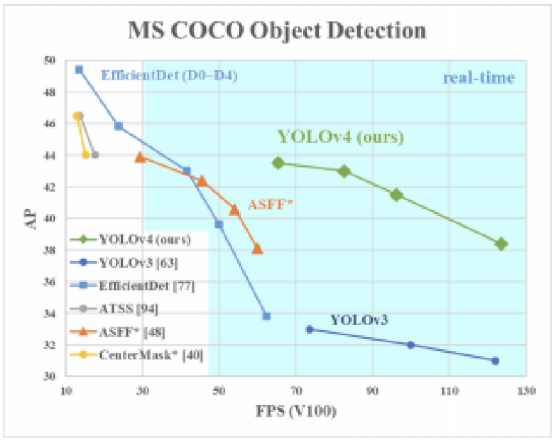
- What YOLOv4 is;
- How YOLOv4 works and;
- Significant features of YOLOv4.

Let's get started!

YOLOv5 has been released

You may want to also see our post on **YOLOv5 vs YOLOv4**. This post will explain some of the pros of the new YOLOv5 framework.

What's new in YOLOv5



*YOLOv4 is both performant and fast (**citation**)*

In this post, we take a deep dive into the research contributions of YOLOv4 and put them in the context of previous work on object detection.

In summary, YOLOv4 is a series of additions of computer vision techniques that are known to work with a few small novel contributions. The main contribution is to discover how all of these techniques can be combined to play off one another effectively and efficiently for object detection.

Looking to train a model?

Skip this post and jump straight to our YOLOv4 tutorial. You'll have a trained YOLOv4 model on your custom data in minutes.

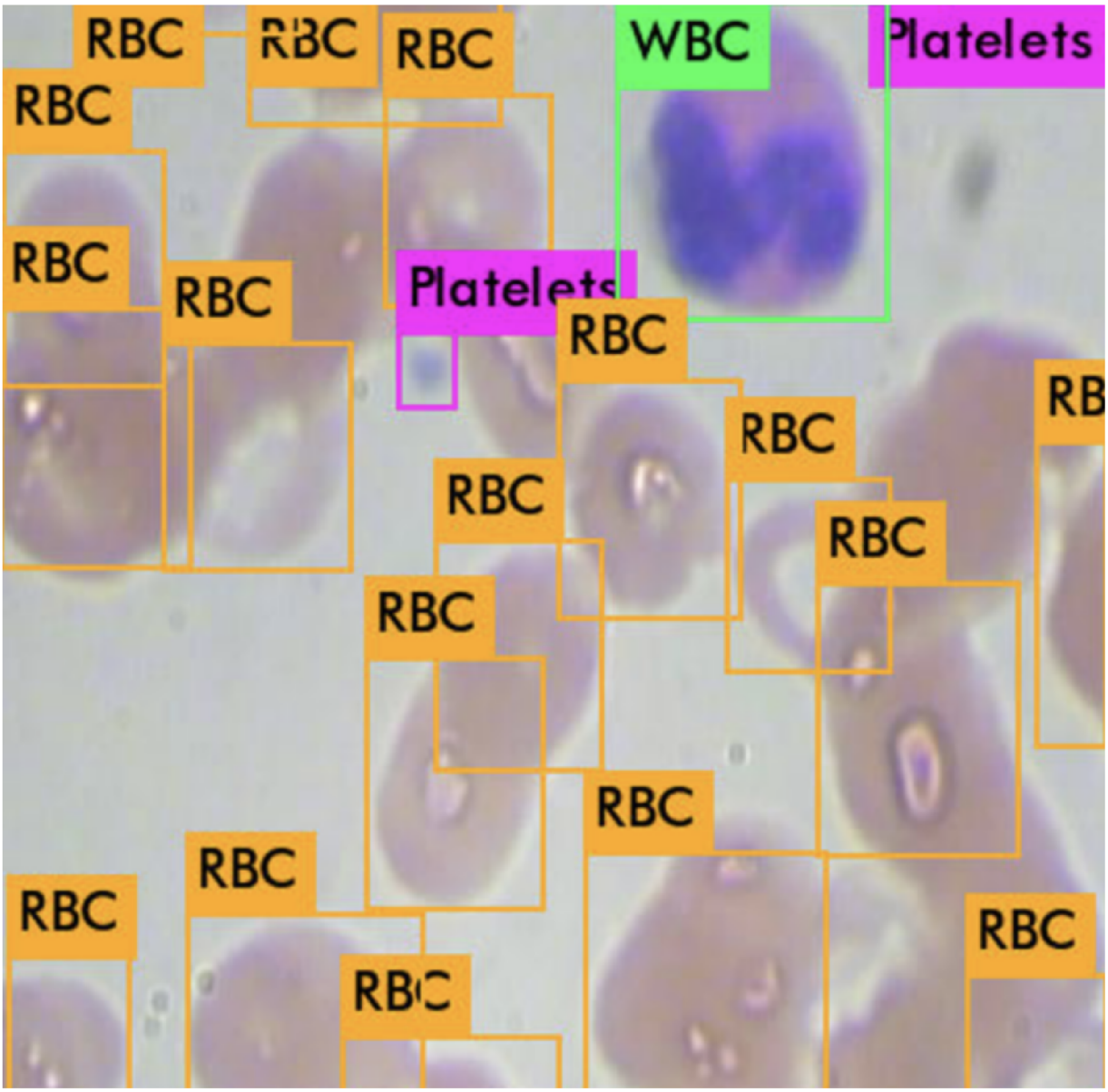
Train YOLOv4 on Custom Data

What is YOLOv4?

YOLOv4 is the fourth version in the You Only Look Once family of models. YOLOv4 makes realtime detection a priority and conducts training on a single GPU. The authors' intention is for vision engineers and developers to easily use their YOLOv4 framework in custom domains.

YOLO and Object Detection Models

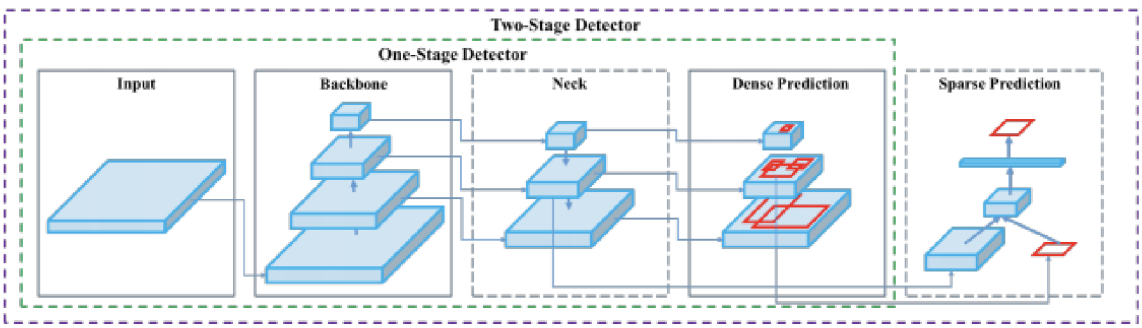
All of the YOLO models are object detection models. Object detection models are trained to look at an image and search for a subset of object classes. When found, these object classes are enclosed in a bounding box and their class is identified. Object detection models are typically trained and evaluated on the COCO dataset which contains a broad range of 80 object classes. From there, it is assumed that object detection models will generalize to new object detection tasks if they are exposed to new training data. Here is an example of me using YOLOv4 to detect cells in the bloodstream.



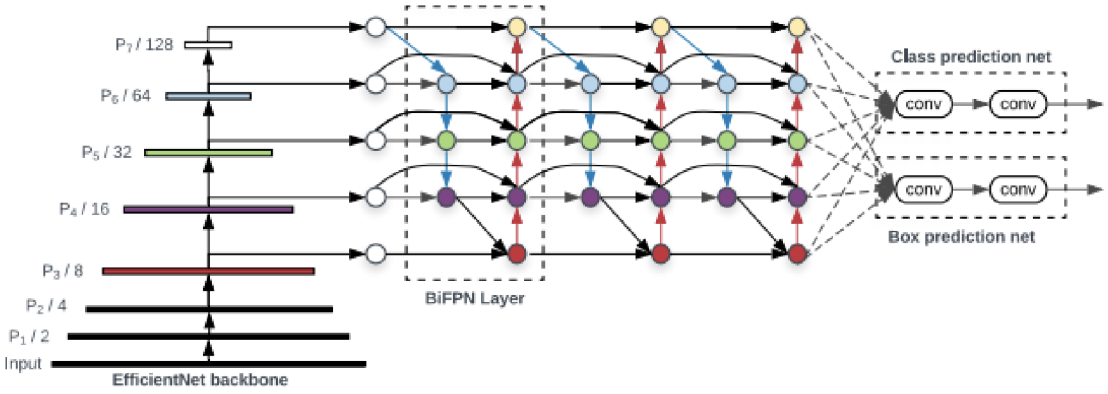
YOLOv4 making inferences on cells in the bloodstream

Realtime is particularly important for object detection models that operate on video feeds, such as self driving cars. The other advantage of realtime object detection models is that they are small and easy to wield by all developers.

The Anatomy of an Object Detector



(citation)



(citation)

All object detectors take an image in for **input** and compress features down through a convolutional neural network backbone. In image classification, these backbones are the end of the network and prediction can be made off of them. In object detection, multiple bounding boxes need to be drawn around images along with classification, so the feature layers of the convolutional backbone need to be mixed and held up in light of one another. The combination of backbone feature layers happens in **the neck.**

It is also useful to split object detectors into two categories: one-stage detectors and two stage detectors. Detection happens in the **head**. Two-stage detectors decouple the task of object localization and classification for each bounding box. One-stage detectors make the predictions for object localization and classification at the same time. YOLO is a one-stage detector, hence, You Only Look Once.

A Brief History of YOLOs

The original YOLO (You Only Look Once) was written by Joseph Redmon (now retired from CV) in a custom framework called Darknet. Darknet is a very flexible research framework written in low level languages and has produced a series of the best realtime object detectors in computer vision: YOLO, YOLOv2, YOLOv3, and now, YOLOv4.

The Original YOLO - YOLO was the first object detection network to combine the problem of drawing bounding boxes and identifying class labels in one end-to-end differentiable network.

YOLOv2 - YOLOv2 made a number of iterative improvements on top of YOLO including BatchNorm, higher resolution, and anchor boxes.

YOLOv3 - YOLOv3 built upon previous models by adding an objectness score to bounding box prediction, added connections to the backbone network layers, and made predictions at three separate levels of granularity to improve performance on smaller objects.

And now onto YOLOv4!

YOLOv4 Deep Dive: The Key Features

How does YOLOv4 work? That's a great question! In this section, we're going to talk about how YOLOv4 works and the main features that comprise the model.

YOLOv4 Backbone Network: Feature Formation

The backbone network for an object detector is typically pretrained on ImageNet classification. Pretraining means that the network's weights have already been adapted to identify relevant features in an image, though they will be tweaked in the new task of object detection.

The authors considered the following backbones for the YOLOv4 object detector

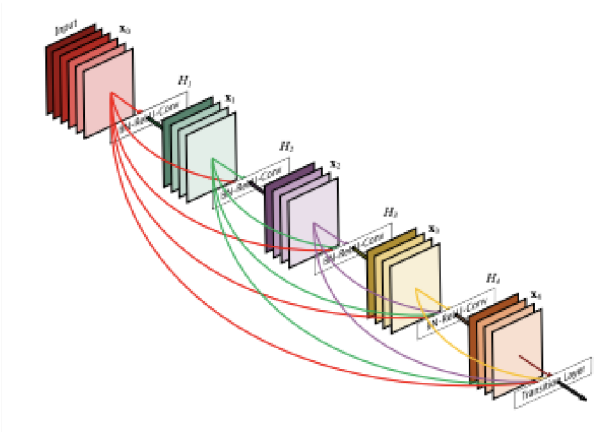
- CSPResNext50
- CSPDarknet53
- EfficientNet-B3

Table 1: Parameters of neural networks for image classification.

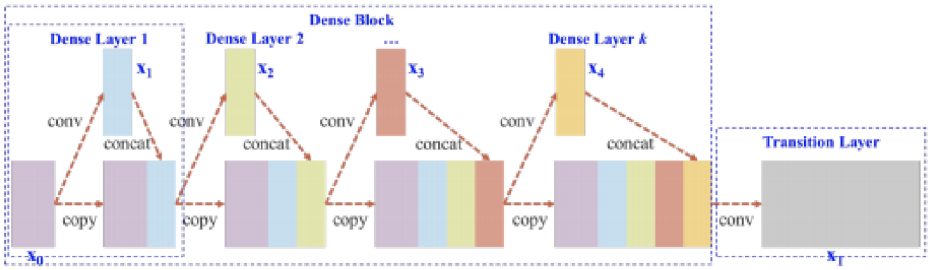
Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	**1058 K**	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	**27.6 M**	950 K	52 (26.0 FMA)	**66**
EfficientNet-B3 (ours)	512x512	**1311x1311**	12.0 M	668 K	11 (5.5 FMA)	26

(citation)

The CSPResNext50 and the CSPDarknet53 are both based on DenseNet. DenseNet was designed to connect layers in convolutional neural networks with the following motivations: to alleviate the vanishing gradient problem (it is hard to backprop loss signals through a very deep network), to bolster feature propagation, encourage the network to reuse features, and reduce the number of network parameters.



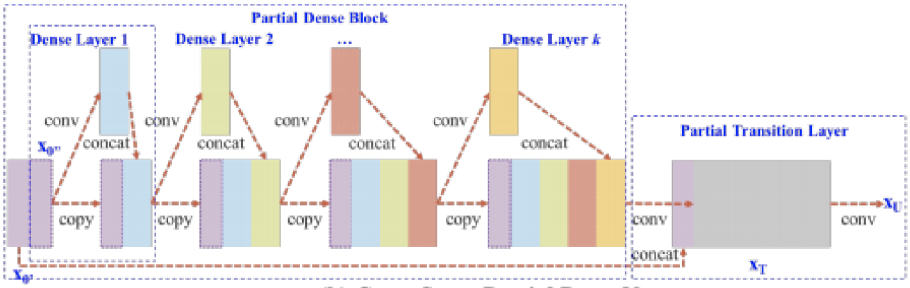
(citation)



(a) DenseNet

(citation)

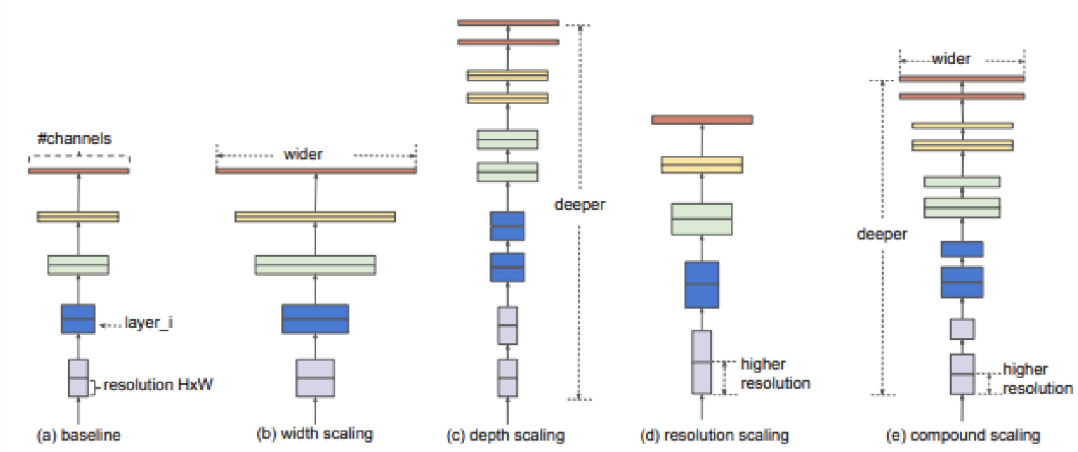
In CSPResNext50 and CSPDarknet53, the DenseNet has been edited to separate the feature map of the base layer by copying it and sending one copy through the dense block and sending another straight on to the next stage. The idea with the CSPResNext50 and CSPDarknet53 is to remove computational bottlenecks in the DenseNet and improve learning by passing on an unedited version of the feature map.



(b) Cross Stage Partial DenseNet

(citation)

EfficientNet was designed by Google Brain to primarily study the scaling problem of convolutional neural networks. There are a lot of decisions you can make when scaling up your ConvNet including input size, width scaling, depth scaling, and scaling all of the above. The EfficientNet paper posits that there is an optimal point for all of these and through search, they find it.



(citation)

EfficientNet outperforms the other networks of comparable size on image classification. The YOLOv4 authors posit, however, that the other networks may work better in the object detection setting and decide to experiment with all of them.

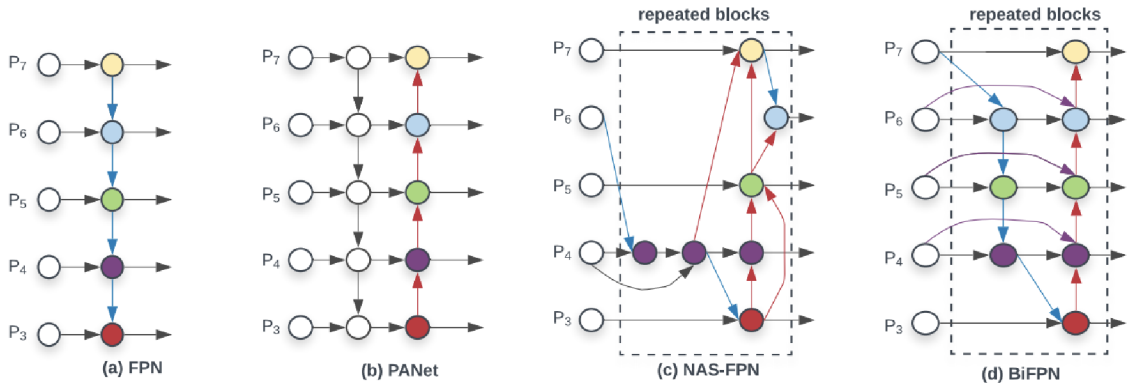
Based on their intuition and experimental results (aka A LOT of experimental results), the final YOLOv4 network implements **CSPDarknet53** for the backbone network.

YOLOv4 Neck: Feature Aggregation

The next step in object detection is to mix and combine the features formed in the ConvNet backbone to prepare for the detection step. YOLOv4 considers a few options for the neck including:

- FPN
- PAN
- NAS-FPN
- BiFPN
- ASFF
- SFAM

The components of the neck typically flow up and down among layers and connect only the few layers at the end of the convolutional network.



(cite)

Each one of the P_i above represents a feature layer in the **CSPDarknet53** backbone.

The image above comes from YOLOv4's predecessor, EfficientDet. Written by Google Brain, EfficientDet uses neural architecture search to find the best form of blocks in the neck portion of the network, arriving at NAS-FPN. The EfficientDet authors then tweak it slightly to the make the more architecture more intuitive (and probably perform better on their development sets).

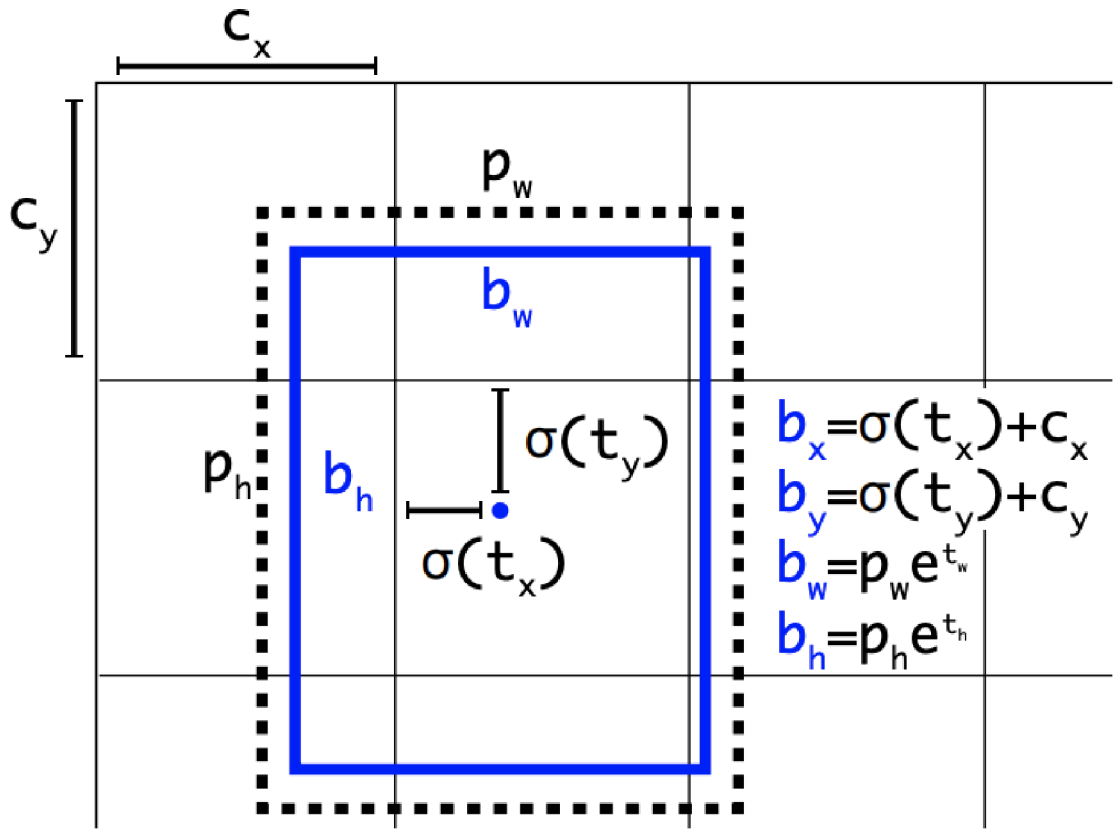
YOLOv4 chooses **PANet** for the feature aggregation of the network. They don't write much on the rationale for this decision, and since NAS-FPN and BiFPN were written concurrently, this is

presumably an area of future research.

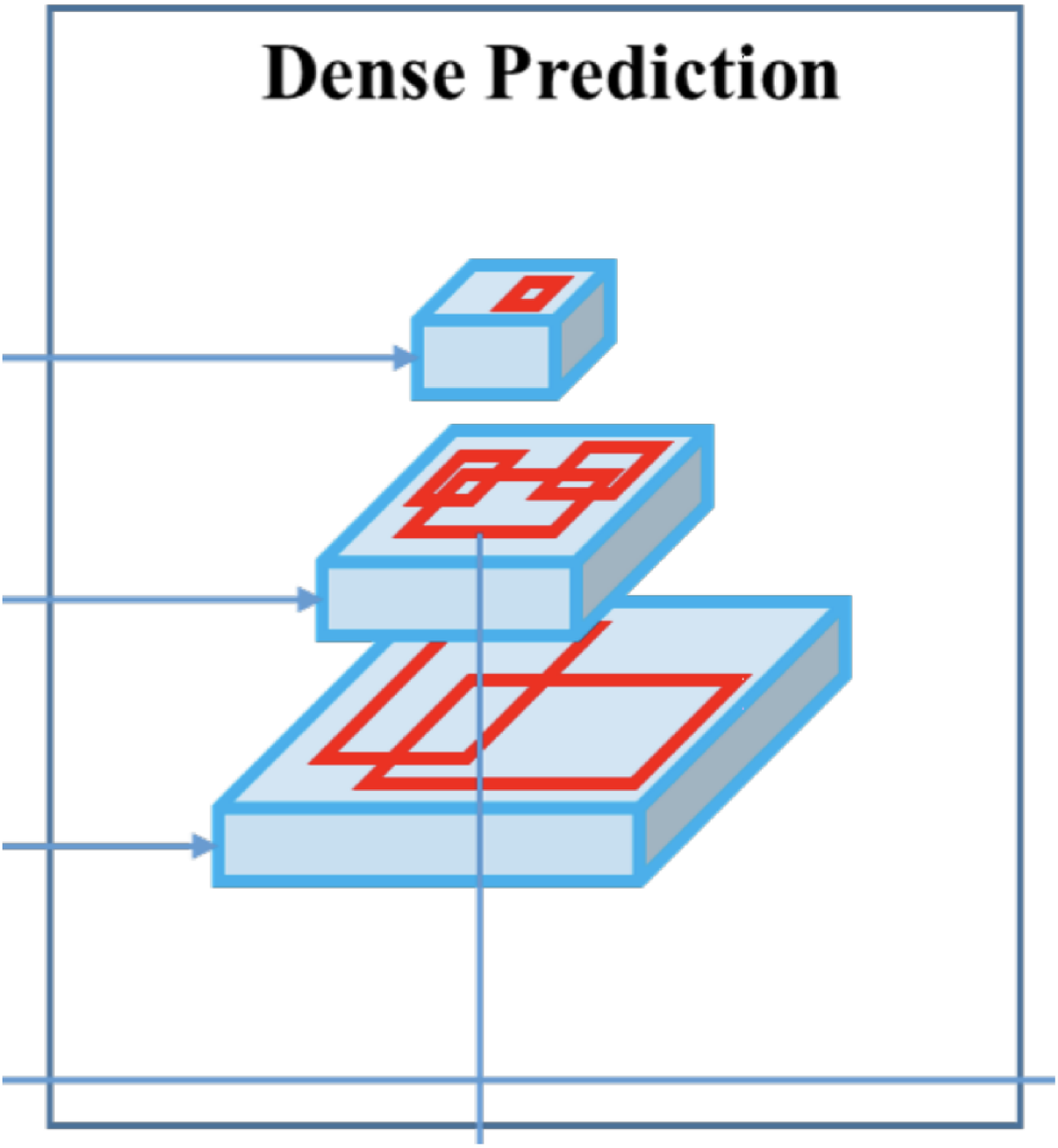
Additionally, YOLOv4 adds a **SPP block** after CSPDarknet53 to increase the receptive field and separate out the most important features from the backbone.

YOLOv4 Head: The Detection Step

YOLOv4 deploys the same **YOLO head** as YOLOv3 for detection with the anchor based detection steps, and three levels of detection granularity.



(citation)



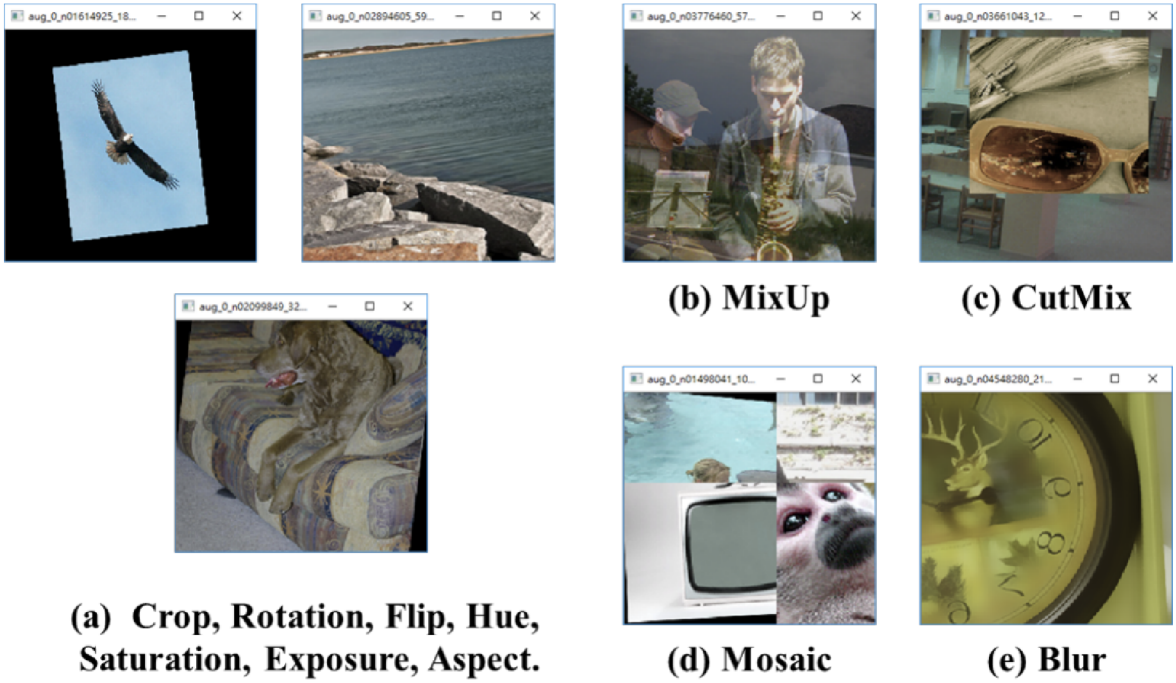
(Citation)

That's all for architecture! Now onto the YOLOv4 Bag of Freebies and Bag of Specials.

YOLOv4 Bag of Freebies

YOLOv4 employs a "**Bag of Freebies**" so termed because they improve performance of the network without adding to inference time in production. Most of the Bag of Freebies have to do with data augmentation. We wrote an in depth dive into the specifics of Data Augmentation in YOLOv4, and will summarize the techniques here. Using data augmentation in computer vision is very important, and we highly recommend it to get the most performance out of your models.

The authors of YOLOv4 use data augmentation to expand the size of their training set and expose the model to semantic situations that it would not have otherwise seen.



(citation)

Many of these strategies were already know to the computer vision community, and YOLOv4 is simply verifying their effectiveness. The new contribution is mosaic data augmentation which tiles four images together, teaching the model to find smaller objects and pay less attention to surrounding scenes that are not immediately around the object.

(citation)

Another unique contribution the authors make in data augmentation is **Self-Adversarial Training (SAT)**. SAT aims to find the portion of the image that the network most relies on during training, then it edits the image to obscure this reliance, forcing the network to generalize to new features that can help it with detection.

The YOLOv4 authors provide an ablation study justifying the data augmentations that they used.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.9%	94.0%
✓							77.2%	**94.0%**
	✓						**78.0%**	**94.3%**
		✓					**78.1%**	**94.5%**
			✓				77.5%	93.8%
				✓			**78.1%**	**94.4%**
					✓		64.5%	86.0%
						✓	**78.9%**	**94.5%**
	✓	✓		✓			**78.5%**	**94.8%**
	✓	✓		✓		✓	**79.8%**	**95.2%**

(citation)

This is what worked best on COCO, but for your own dataset, it is important to think critically about what augmentations might help and deploy augmentations experimentally.

Another freebie is **CIoU loss** to edit the loss function. The YOLOv4 authors use **CIoU loss**, which has to do with the way the predicted bounding box overlaps with the ground truth bounding box. Basically, it is not enough to just look at the overlap, because in the event of no overlap, you also want to look at how close the box was to the ground truth box and encourage the network to pull over the predicted box closer to the ground truth box. Of course, there's a lot of mathematical engineering to that.

YOLOv4 Bag of Specials

YOLOv4 deploys strategies called a "**Bag of Specials**", so termed because they add marginal increases to inference time but significantly increase performance, so they are considered worth it.

The authors experiment with various activation functions. Activation functions transform features as they flow through the network. With traditional activation functions like ReLU, it can be difficult to get the network to push feature creations towards their optimal point. So the research has been done to produce functions that marginally improve this process. **Mish** is an activation function designed to push signals to the left and right.

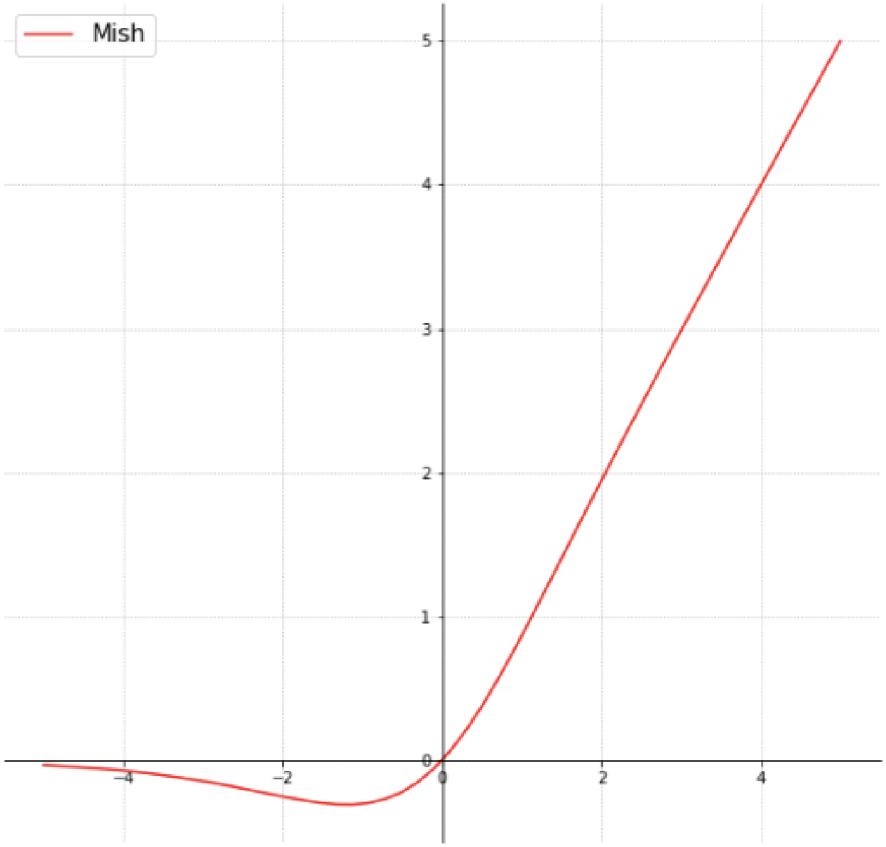
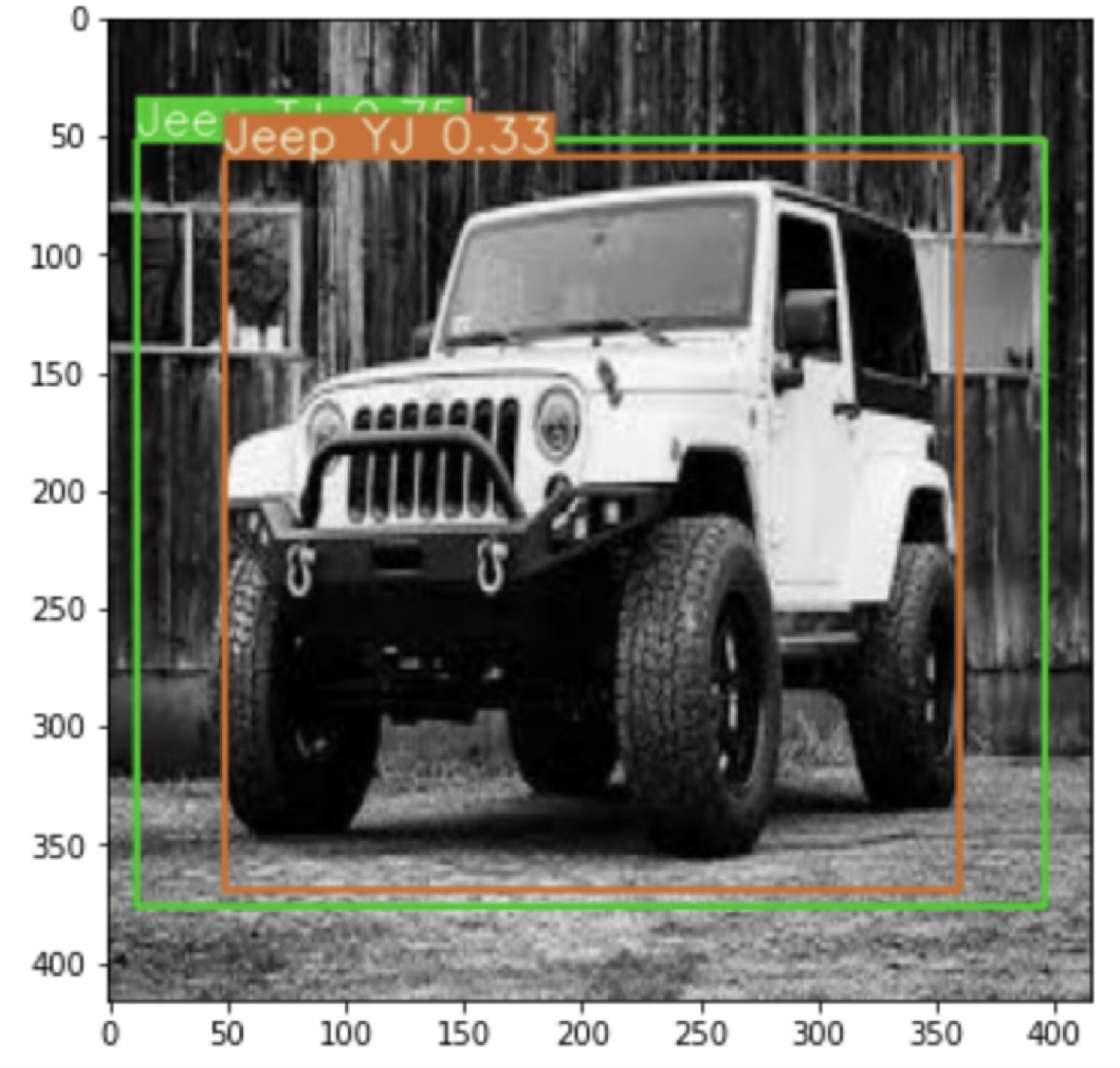


Figure 1. Mish Activation Function

(citation)

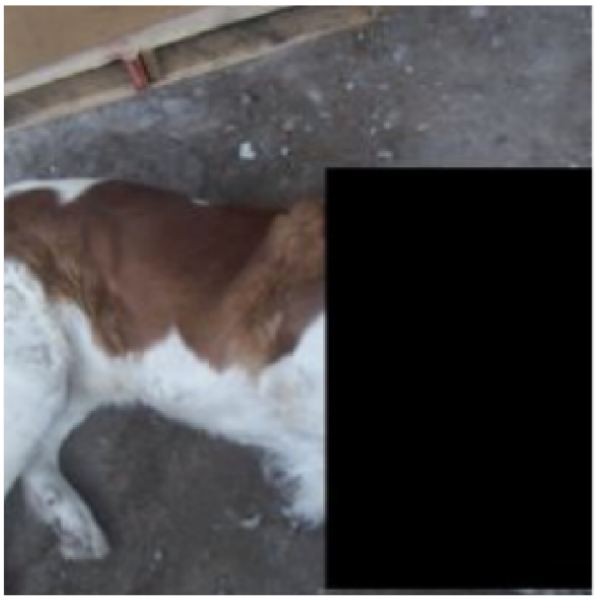
The authors use **DIoU NMS** to separate out predicted bounding boxes. The network may predict multiple bounding boxes over a single object, and it would be useful to efficiently pick the best one.



Using YOLOv3 and it needs some better NMS like YOLOv4 - this cannot be two kinds of jeeps at once (green label is Jeep TJ, brown label is Jeep YJ)

For batch normalization, the authors use **Cross mini-Batch Normalization (CmBN)** with the idea that this can be run on any GPU that people use. Many batch normalization techniques require multiple GPUs operating in tandem.

YOLOv4 uses **DropBlock regularization**. In DropBlock, sections of the image are hidden from the first layer. DropBlock is a technique to force the network to learn features that it may not otherwise rely upon. For example, you can think of a dog with its head hidden behind a bush. The network should be be able to identify the dog from its torso as well as its head.



(citation)

YOLOv4: Experimental Results

The techniques in YOLOv4 were thoroughly proved out via experimentation on MS COCO. COCO contains 80 object classes and is meant to represent a broad range of object detection scenarios that a detector may need to encounter in the wild.

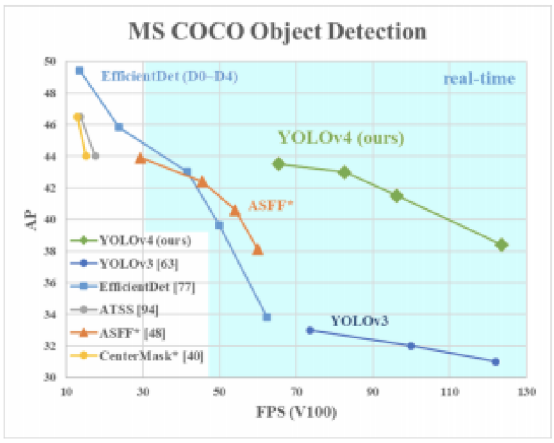
YOLOv4 does an in depth ablation study on the techniques tried in the paper. An ablation study seeks to remove additions sequentially to prove which additions are improving the network.

Table 4: Ablation Studies of Bag-of-Freebies. (CSPResNeXt50-PANet-SPP, 512x512).

S	M	IT	GA	LS	CBN	CA	DM	OA	loss	AP	AP_{50}	AP_{75}
									MSE	38.0%	60.0%	40.8%
✓									MSE	37.7%	59.9%	40.5%
	✓								MSE	**39.1%**	**61.8%**	**42.0%**
		✓							MSE	36.9%	59.7%	39.4%
			✓						MSE	**38.9%**	**61.7%**	**41.9%**
				✓					MSE	33.0%	55.4%	35.4%
					✓				MSE	**38.4%**	**60.7%**	**41.3%**
						✓			MSE	**38.7%**	**60.7%**	**41.9%**
							✓		MSE	35.3%	57.2%	38.0%
✓									GIoU	**39.4%**	59.4%	**42.5%**
✓									DIoU	**39.1%**	58.8%	**42.1%**
✓									CIoU	**39.6%**	59.2%	**42.6%**
✓	✓	✓	✓						CIoU	**41.5%**	**64.0%**	**44.8%**
	✓		✓					✓	CIoU	36.1%	56.5%	38.4%
✓	✓	✓	✓					✓	MSE	**40.3%**	**64.0%**	**43.1%**
✓	✓	✓	✓					✓	GIoU	**42.4%**	**64.4%**	**45.9%**
✓	✓	✓	✓					✓	CIoU	**42.4%**	**64.4%**	**45.9%**

(citation)

And with the final configuration, YOLOv4 achieves state of the art performance for object detection. The paper examines inference time on many different GPUs, though we just display one here.



YOLOv4 achieves state of the art for object detection (citation)

MORE ABOUT EDUCATION VIEW ALL

YOLOv4: Let's get it out there

4 is [illegible] suite of tec [illegible] detection in computer vision. [illegible] improved to [illegible] the lightweight and easy to use.

OCT 6, 2023

YOLOv4 was designed with proliferation in mind. You can train YOLOv4 on your custom objects, easily on your own GPU or on Google Colab.

[illegible] quick and easy to use tutorials to ge [illegible] LOv4 including:

- How to train YOLOv4 in the darknet framework and;
- How to train YOLOv4 in PyTorch.

[illegible] these tutorials, you will have a train [illegible] can do inference in the domain [illegible]. Stay tuned for future tutorials like h [illegible] LOv4 onto an NVIDIA Jetson and how to train YOLOv4 in TensorFlow.

Thank you for reading and happy detecting!

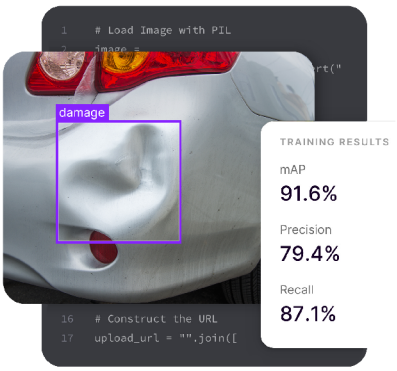
Frequently Asked Questions

What is YOLOv4 used for?

Like all other models in the YOLO family, the YOLOv4 model is used for object detection. Object detection problems are where you need to identify where particular objects are in a given image (i.e. where a car is in an image).

How many convolutional layers does YOLOv4 have?

YOLOv4 uses a CSPDarknet53 backbone. This backbone is comprised of 53 convolutional layers.



Build and deploy computer vision models with Roboflow

Join over 100,000 developers and top-tier companies from Walmart to Cardinal Health building computer vision models with Roboflow.

Get started

Jacob Solawetz

Founding Engineer @ Roboflow - ascending the 1/loss

VIEW MORE POSTS

TOPICS:

Education

Safety & Security

Transportation

All Industries