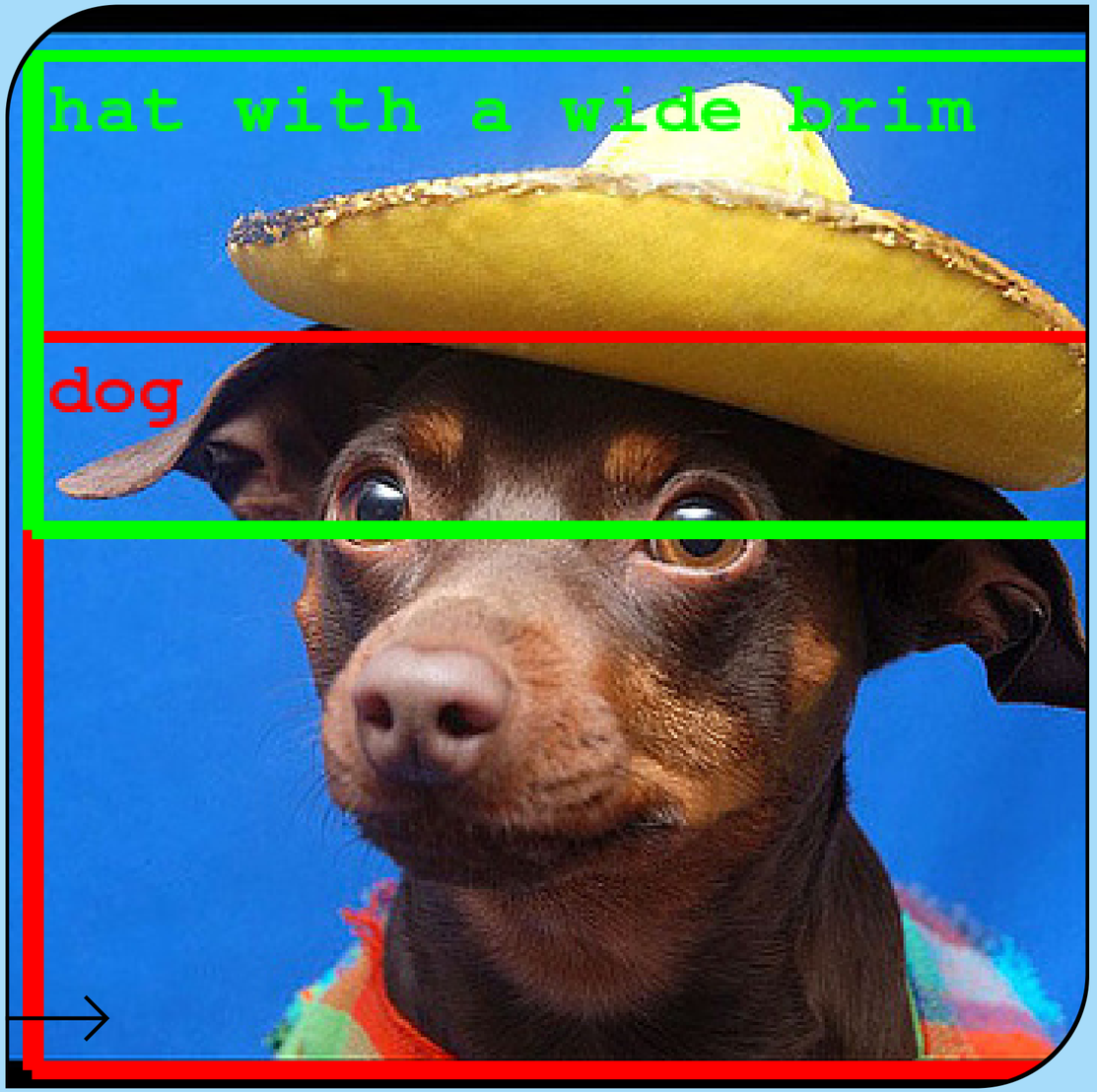
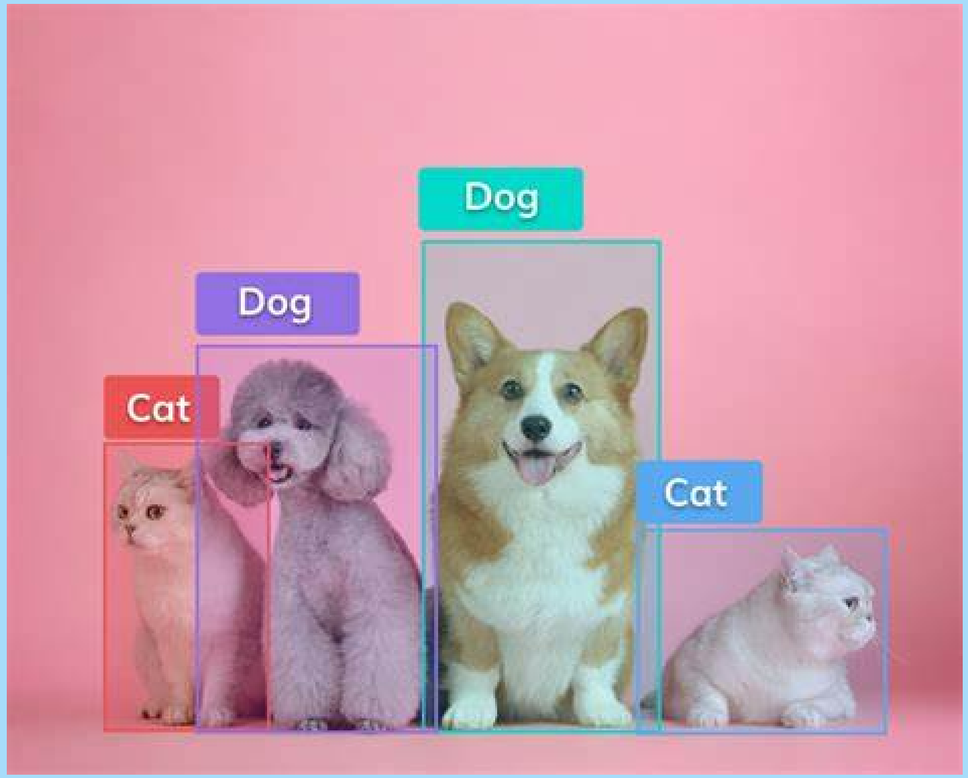
# YOLOX

Exceeding YOLO series in 2021

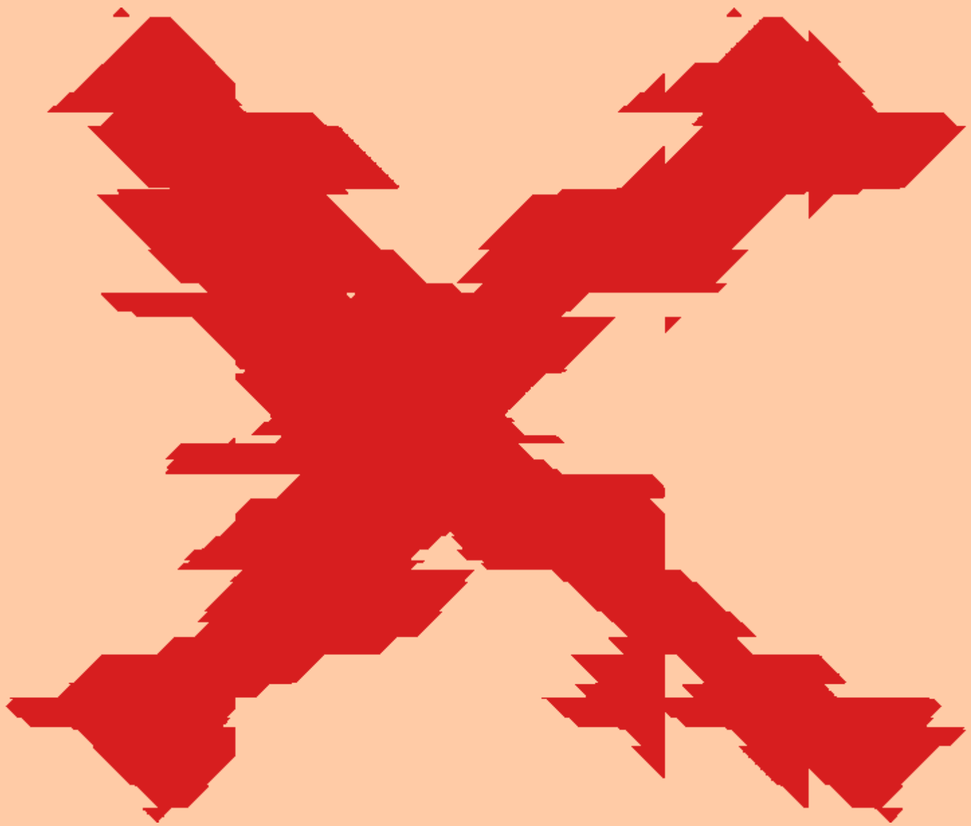
VISUAL LEARNERS



# Introduction




- YOLOX is YOLO’s new high-performance object detection algorithm.
- Unlike previous YOLO algorithms, it includes three fundamental innovations.
- These are decoupled head, anchor-free, and advanced label assigning strategy.
- With these innovations, **YOLOX provides a better performance in speed and performance than its counterparts.**



- Anchor free detectors, advanced label assignment strategies and end-to-end detectors are the most important focus points in object detection lately.

- These new focal points are still not integrated into the YOLO al[illegible] and YOLOv4,YOLOv5 are still anchor based detector and uses hand-crafted assigning rules for training.

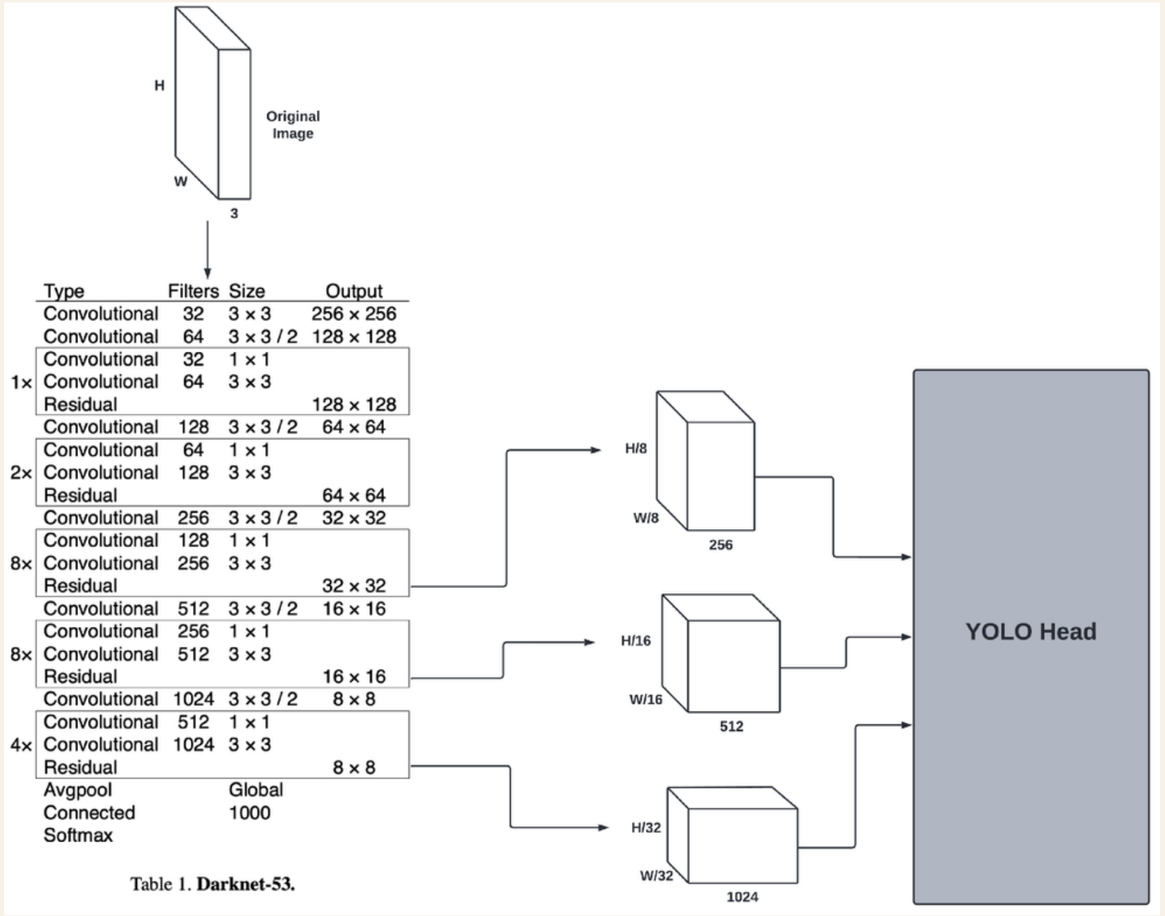
- It is an important reason for the development of the YOLOX algorithm.

| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 × 3 | 256 × 256 |
| | Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| 1× | Convolutional | 32 | 1 × 1 | |
| | Convolutional | 64 | 3 × 3 | |
| | Residual | | | 128 × 128 |
| | Convolutional | 128 | 3 × 3 / 2 | 64 × 64 |
| 2× | Convolutional | 64 | 1 × 1 | |
| | Convolutional | 128 | 3 × 3 | |
| | Residual | | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 | 32 × 32 |
| 8× | Convolutional | 128 | 1 × 1 | |
| | Convolutional | 256 | 3 × 3 | |
| | Residual | | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 | 16 × 16 |
| 8× | Convolutional | 256 | 1 × 1 | |
| | Convolutional | 512 | 3 × 3 | |
| | Residual | | | 16 × 16 |
| | Convolutional | 1024 | 3 × 3 / 2 | 8 × 8 |
| 4× | Convolutional | 512 | 1 × 1 | |
| | Convolutional | 1024 | 3 × 3 | |
| | Residual | | | 8 × 8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |

Table 1. **Darknet-53.**

# Darknet-53 — The YOLOX Backbone

- The **YOLOv3 algorithm** is the basis for many object detection algorithms and is also what YOLOX uses.
- **YOLOv3** uses a large backbone called **Darknet-53**.
- This backbone is also what YOLOX uses and it has the following architecture.

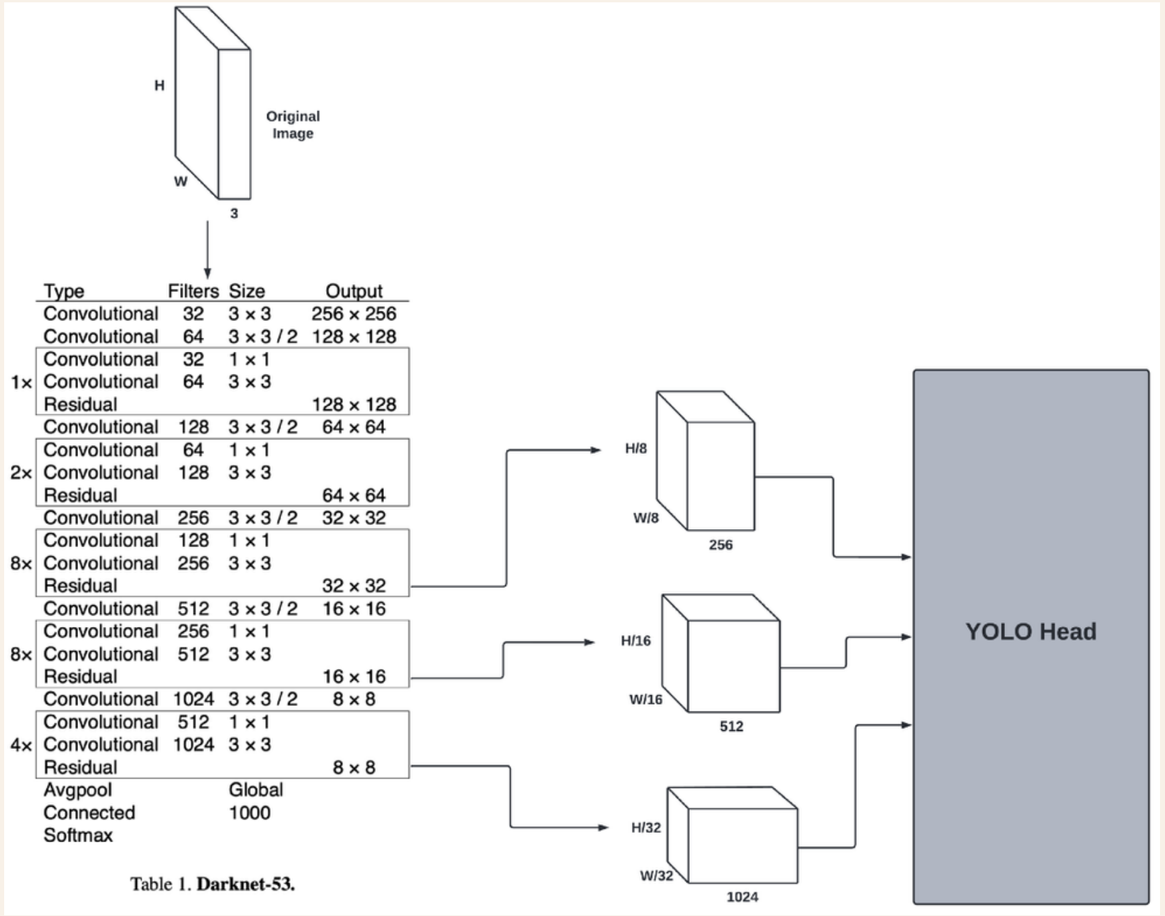



# The Darknet backbone and its three outputs

**"Essentially, the Darknet-53 backbone (which I may call the FPN from now on) outputs three different predictions at different scales:**

- 256 channels (first transition output)
- 512 channels (second transition output)
- 1024 channels (third transition output)

# The Darknet backbone and its three outputs




| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 × 3 | 256 × 256 |
| | Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| 1× | Convolutional | 32 | 1 × 1 | |
| | Convolutional | 64 | 3 × 3 | |
| | Residual | | | 128 × 128 |
| | Convolutional | 128 | 3 × 3 / 2 | 64 × 64 |
| 2× | Convolutional | 64 | 1 × 1 | |
| | Convolutional | 128 | 3 × 3 | |
| | Residual | | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 | 32 × 32 |
| 8× | Convolutional | 128 | 1 × 1 | |
| | Convolutional | 256 | 3 × 3 | |
| | Residual | | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 | 16 × 16 |
| 8× | Convolutional | 256 | 1 × 1 | |
| | Convolutional | 512 | 3 × 3 | |
| | Residual | | | 16 × 16 |
| | Convolutional | 1024 | 3 × 3 / 2 | 8 × 8 |
| 4× | Convolutional | 512 | 1 × 1 | |
| | Convolutional | 1024 | 3 × 3 | |
| | Residual | | | 8 × 8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |

Table 1. **Darknet-53.**

**Each of these outputs extracts information at a different scale. Notice how as the number of channels increases, the length and width of the image decreases.**

So the **256 channel** transition output extract features at a smaller scale while the **1024 channel** output extracts features at a larger scale since the **1024 channel** output has less information from the original image to work with.

# YOLOv3 head vs. YOLOX head

Although the YOLOv3 backbone and the YOLOX backbone are the same, the models begin to differ from their heads. Below is an image showing the difference between the two heads.

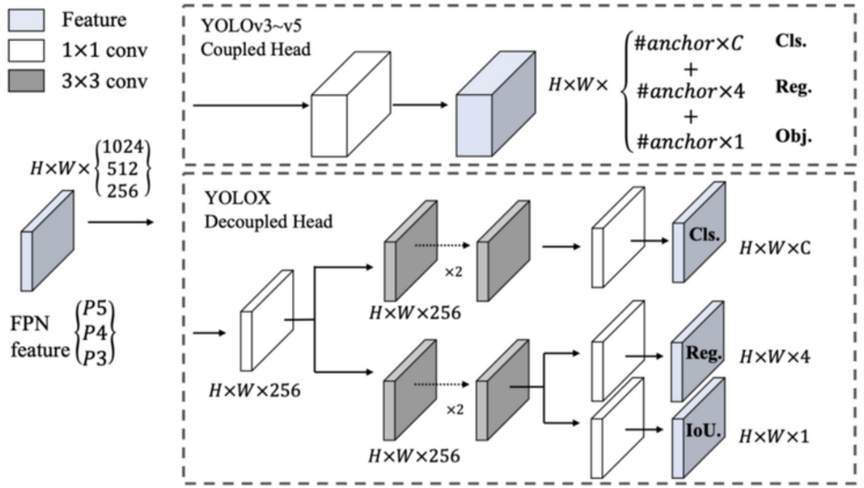



Figure 2: Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, we first adopt a $1 \times 1$ conv layer to reduce the feature channel to 256 and then add two parallel branches with two $3 \times 3$ conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch.

# YOLOv3 head vs. YOLOX head

When I first saw this diagram, I found it a bit confusing, so below is my marked-up version of the diagram which I hope makes it a little easier to understand.

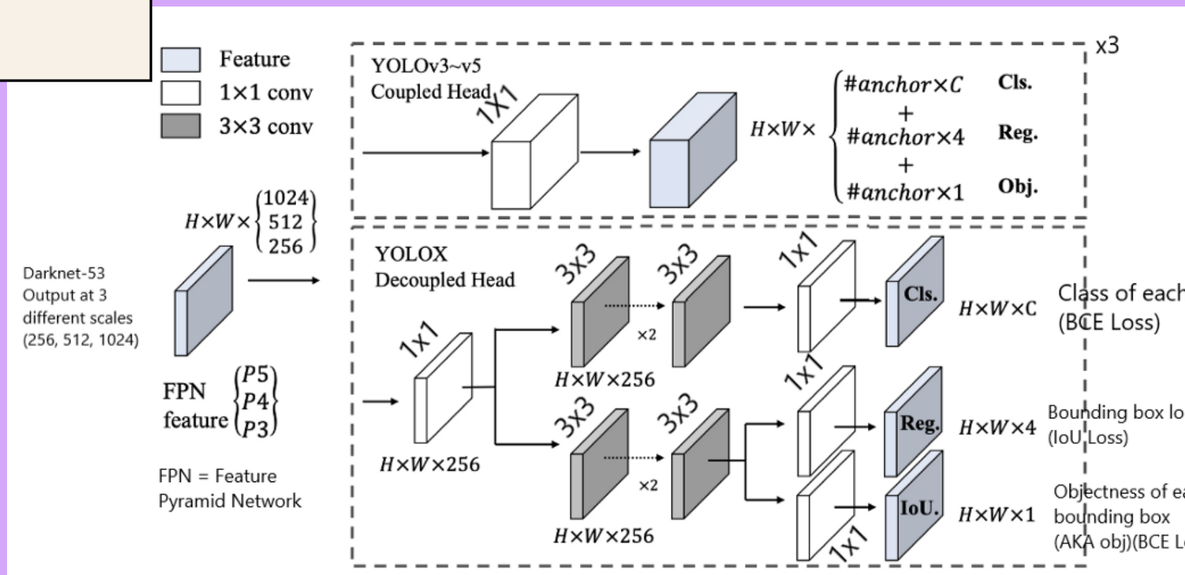



Figure 2: Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of feature, we first adopt a $1 \times 1$ conv layer to reduce the feature channel to 256 and then add two parallel branches with $3 \times 3$ conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch

# YOLOv3 head vs. YOLOX head

This diagram is stating that the input into both the YOLOv3 head and the YOLOX head is the 3 outputs from the FPN (darknet) backbone at three different scales – **1024, 512, 256 channels**.

The output of the two heads is essentially the exact same with dimensions **(H×W×features)** which is just like the original YOLO.

The difference between the two heads is that **YOLOv3** uses a **coupled head** and **YOLOX** uses a **decoupled head**.

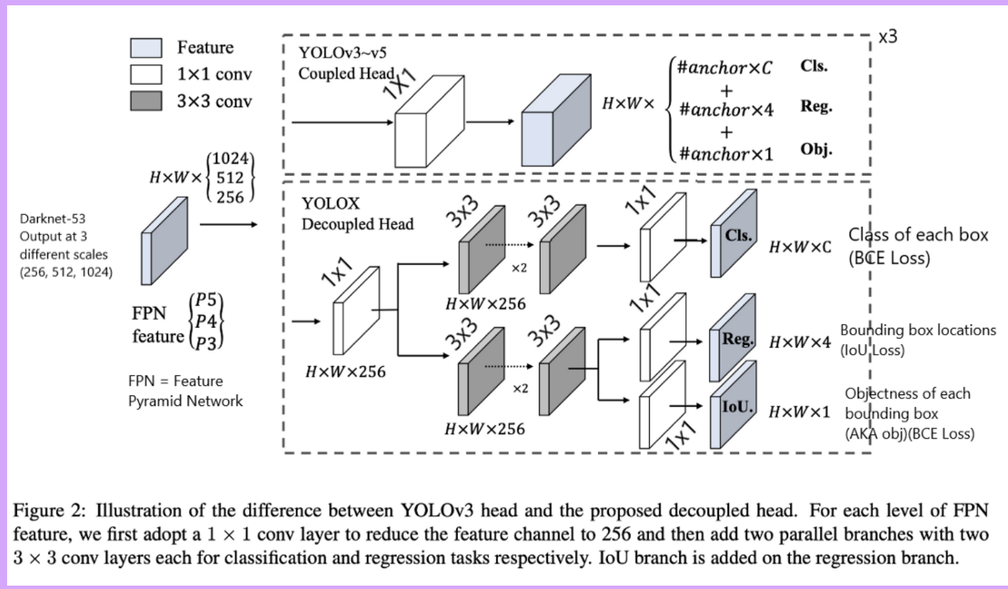



Figure 2: Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, we first adopt a $1 \times 1$ conv layer to reduce the feature channel to 256 and then add two parallel branches with two $3 \times 3$ conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch.

# YOLOv3 head vs. YOLOX head

So, the output of YOLOX is actually 3 tensors each holding different information instead of 1 massive tensor with all information.

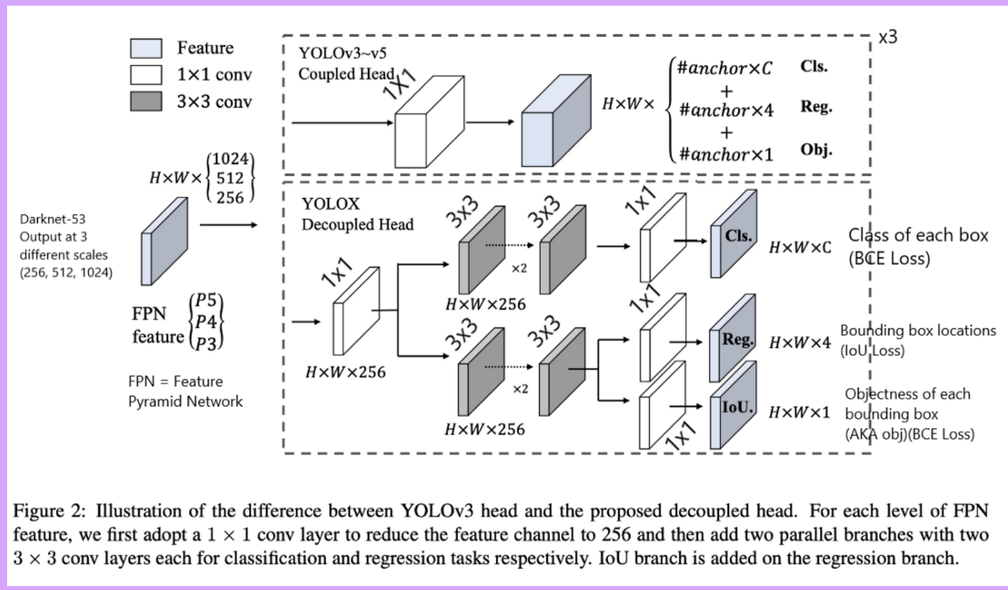



Figure 2: Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, we first adopt a $1 \times 1$ conv layer to reduce the feature channel to 256 and then add two parallel branches with two $3 \times 3$ conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch.

The three tensors YOLOX outputs hold the same information as the massive tensor YOLOv3 outputs:

1. **Cls:** The class of each bounding box
2. **Reg:** The 4 parts to the bounding box **(x, y, w, h)**
1. **IoU (Obj):** For some reason, the authors use IoU instead of Obj, but this output is just how confident is the network that there's an object in the bounding box (objectness)

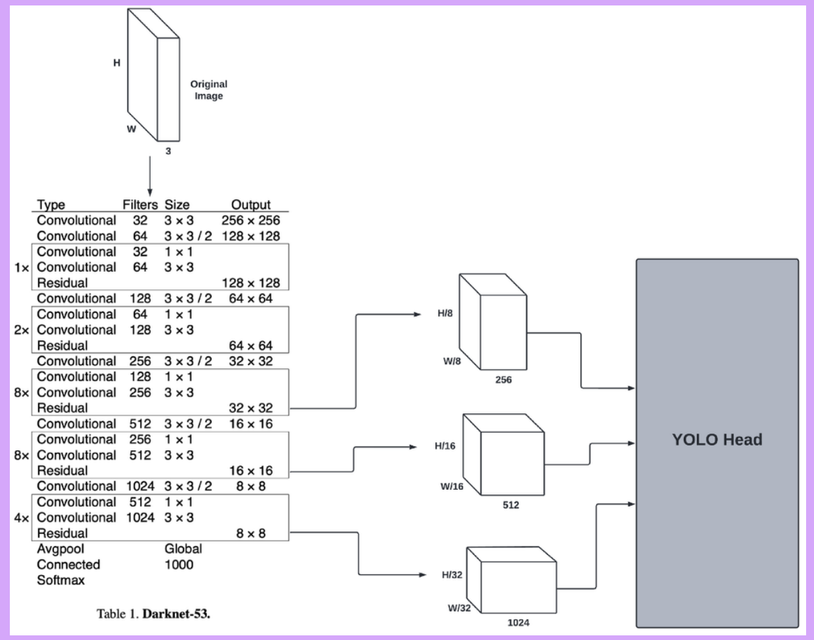



Table 1. **Darknet-53.**

# YOLOv3 head vs. YOLOX head

Just like with the original output, each **“pixel”** in the **height and width** of the output is a **different bounding box prediction**. So, there are **H*W different predictions**.

The outputs listed above are only for **a single output** in the **FPN**. Remember there are **three outputs** from the **FPN** which are fed into the heads of **YOLOv3 and YOLOX**.

This means there are actually **three different outputs** from each of the heads **instead of 1**. So, the **output of YOLOv3** is actually **(3×H×W×features)** and the output of **YOLOX** is actually **3 of each of the Cls, Reg, and IoU (obj) outputs** making 9 totals outputs.

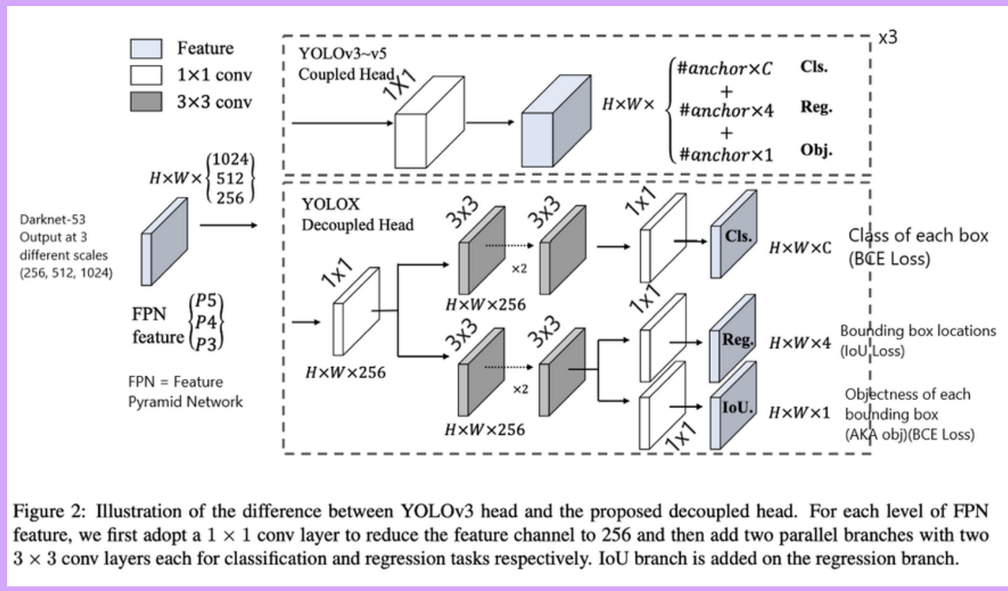



Figure 2: Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, we first adopt a $1 \times 1$ conv layer to reduce the feature channel to 256 and then add two parallel branches with two $3 \times 3$ conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch.

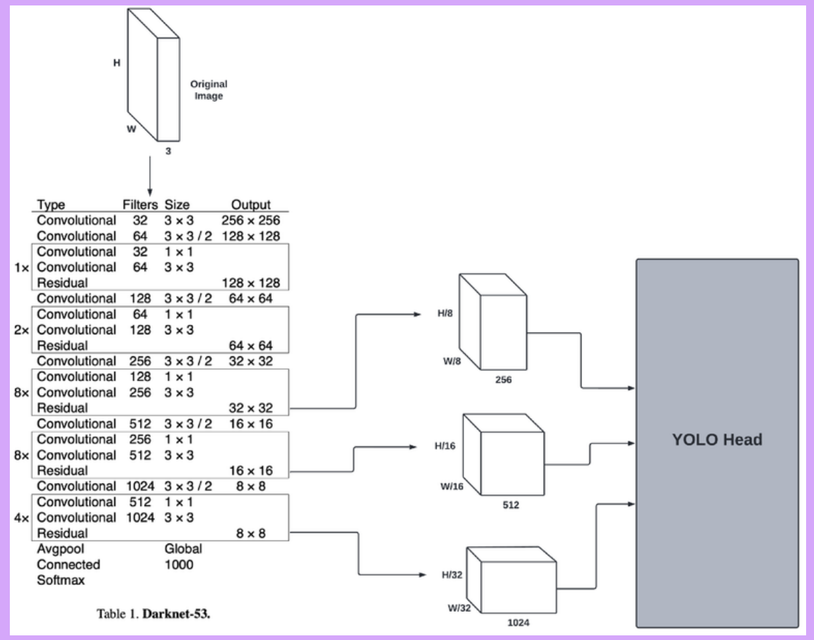


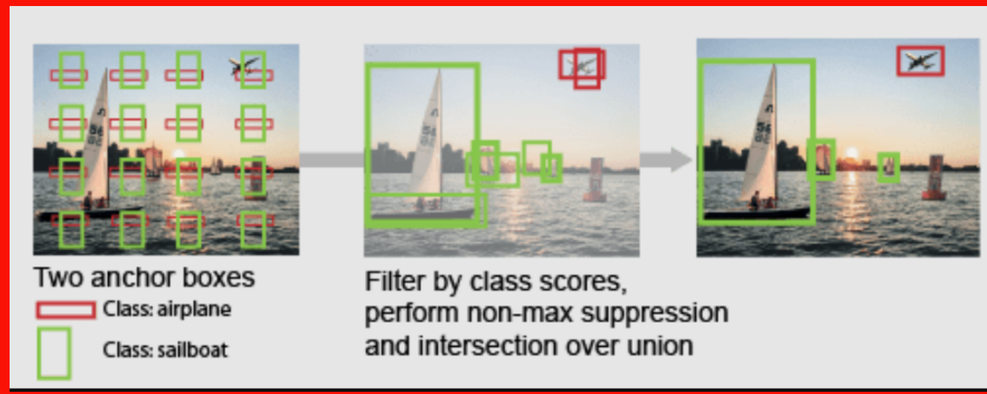

Table 1. **Darknet-53.**

# Switching To An Anchor-free Mode

One of the most important changes YOLOX made was not using anchors whereas YOLOv3 heavily relies on anchors.

Basically, an **anchor box** is a way to help the model so it doesn't have to directly predict a bounding box.

Although anchor boxes allow us to find more than one object in the same grid, they have some disadvantages.



# The Problem With Anchor Boxes

First of all, **optimal anchor boxes** should be determined for an **optimal object detection**. For this reason, we try to find **optimal anchor boxes** with **clustering analysis** before training. This is an **extra workload and waste of time for us**.

Another problem is that the anchor box structure mechanism **increases the number of predictions** to be made for each image. This can cause **delays and slowdowns on some systems**.

In recent years, it has been determined that **anchor free systems** are as successful as **anchor based systems**. With anchor free systems, similar success has been achieved and the design parameters have decreased and the speed has increased.

# How Does YOLOX Fix The Anchor Box Problem?

**YOLOX** simply has the model **directly predict the bounding box dimensions** as opposed to **predicting an offset from an anchor box**.

To directly predict a bounding box, YOLOX uses a **decoupled head** which was explained above. Additionally, it uses something called **striding.**

YOLOX was not just based on YOLOv3, it was also based on **FCOS**, which is another **bounding box model**, but it's **not part of the YOLO series** making it not very cool.

# Striding

**FCOS** uses **striding** to help the model out. Imagine the model has to learn to predict a bounding box anywhere from the top left of an image at **0,0** to the bottom right of an image at **1024,1024**.

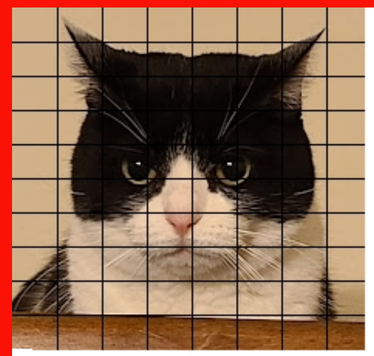
In a discrete space, the model has **1,048,576 possible locations** to predict from and it will likely not be able to learn anything due to this wide range of predictions.

**Striding fixes this issue and allows the model to predict from an offset as opposed to from the top-left of the image.**

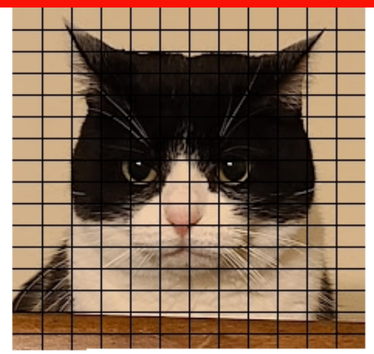
# Striding

Basically, we can split the image up into a **grid based on the three different scales** the model will make predictions at. For example, the grids may look something like the image shown below.

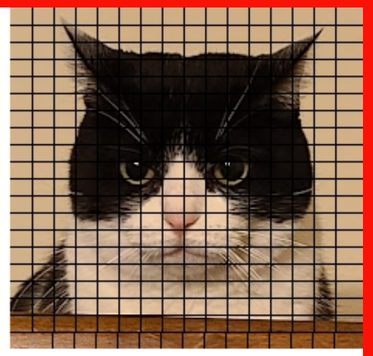
Using these grids, we can assign **each prediction** to each of the **intersection points** on the grid. The nice part about the predictions from the YOLOX model is they're already in a **length×width format**. So we can directly map each of the **outputs to a unique point on the grid** and then use that **grid point as an offset to scale the bounding box**.



63 Predictions



165 Predictions



285 Predictions

# Striding

The grids above can be created by defining a certain stride which is the distance between each of the intersection points on the grid.

In the YOLOX algorithm, strides of **32, 16, and 8** are used for each **FPN level** respectively. If a **stride of 32** is used on a **256×256 image**, then there will be a total of **256/32 = 8 intersection points** on each dimension totaling **64 intersection points**.

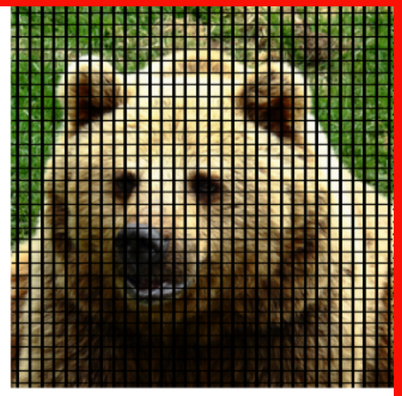
# Striding

TThe following image has the grid overlay for the bear image at **three different levels with different stride**.

Each intersection point on the image is called an **anchor point**. An **anchor point** is an **offset** to move the **x,y location** of a prediction while the **anchor box** explained previously **(which is what YOLOX gets rid of)** is a predefined box that is used as an offset for the **w, h** parts of a prediction.

Level 1
(Stride of 32)

Level 2
(Stride of 16)



Level 3
(Stride of 8)

# Striding

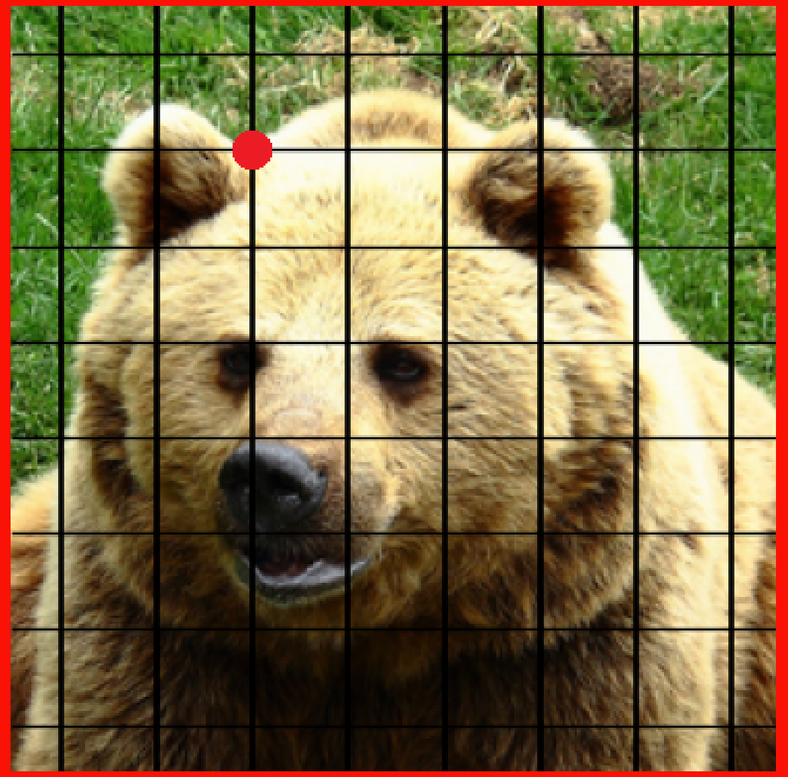
The anchor location on the image can be obtained with the following formulas:

```
x = s/2 + s*i
y = s/2 + s*j
```

Where s is the stride, i is the ith intersection point on the x-axis and j is the jth intersection point on the y-axis

# Striding

For example, let’s go back the bear image with a **stride of 32**. If the anchor point for this prediction was **(i, j) = (2, 1)** meaning intersection point **2 on the x-axis and 1 on the y-axis**, I would be looking at the following point on the image:



Note: The point is at **(2, 1)** on the grid, but pixel-wise, it is at:

```
x = 32/2 + 32*2 = 16 + 64 = 80
y = 32/2 + 32*1 = 16 + 32 = 48
```

# Striding

For YOLOX, we use the grid points as **top-left offsets** of the bounding box. The following formulas are used to map a predicted bounding box **(p_x, p_y, p_w, p_h)** to the actual location on the image **(l_x, l_y, l_w, l_h)** if **(x, y)** is the intersection point on the grid which the prediction belongs to and **s** is the stride at the current FPN level:

```
l_x = p_x + x
l_y = p_y + y
l_w = s*e^(p_w)
l_h = s*e^(p_h)
```

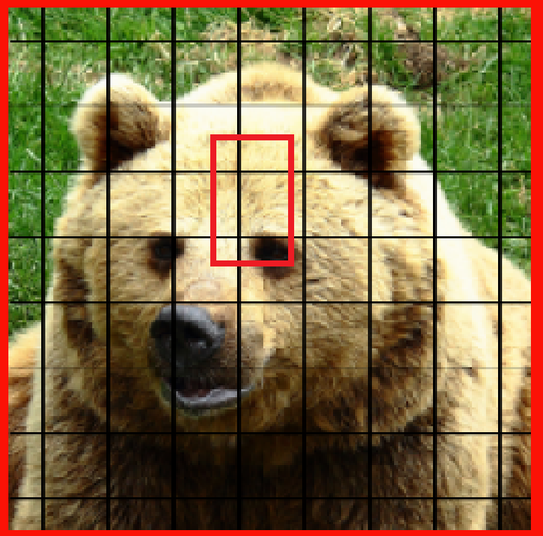
We move the predicted point **by adding the prediction to the anchor (the x,y point assigned to this prediction)**. We also **denormalize the width and height** by ensuring it's **not negative** with **an exponential function** and moving it based on the **stride of an image**.

# Striding

If the model gave me the prediction of **(20, 15, 0.2, 0.3)**, then we can calculate the box as:

```
l_x = 20 + 80 = 100
l_y = 15 + 48 = 63
l_w = 32*e^(0.2) = 39
l_h = 32*e^(0.3) = 43
```

So, the final image may look like the following:



# Label Assignment

Not all predictions are **equal**. Some are clearly garbage and we don't even want our model to optimize them. To differentiate between good and bad predictions, YOLOX uses something called **SimOTA** which is used for **dynamic label assignment**.

Predictions that are thought of as "**good**" **(which bound a ground truth object)** are labeled as **positive** and those that are "**bad**" **(which bound the background)** are labeled as **negative**.

**SimOTA** doesn't just assign **positive/negative labels**, but it also assigns the **ground truth bounding boxes** to each **positively labeled anchor** in the image. If it selects a single positive sample, it ignores other predictions, even if they are of high quality.

# Loss Functions – Evaluating YOLOX

There are **three outputs** to the **YOLOX model** and each output has **its own loss function** as they need to be **optimized in different ways**.

The three different losses include **Class loss, Regression loss, Object loss**. The **final loss function** is a combination of the three losses stated above and is defined as follows:

$$L = \frac{1}{N_{pos}} L_{cls} + reg_{weight} * \frac{1}{N_{pos}} L_{reg} + \frac{1}{N_{pos}} L_{obj}$$

The loss function is basically **the sum of all losses averaged over the number of positive labels**. Remember, we used **SimOTA** to assign labels to each prediction. **reg_weight** is a balancing term used to weigh the regression loss over the other losses as it's most important to optimize. The authors use a weight of 5.0.

# Data Augmentations

Along with YOLOX, **RandomHorizontalFlip, ColorJitter and multi-scale** were used in the data augmentation part. **RandomResizedCrop**, which was used for data augmentation in previous YOLO versions, is **no longer used**. YOLOX does not use **RandomResizedCrop** as the **RandomResizedCrop strategy** is countered by **Mosaic augmentation**.

Another data augmentation method added to YOLOX is **MixUp**. Although **MixUp** was originally a data augmentation method developed for **image classification**, it has **become usable for object detection with BoF**.

As an important point, after the **MixUp and Mosaic data augmentation methods**, **ImageNet pre-training**, which was used before, has been abandoned, train all the following models from scratch.

# Data Augmentations

Along with YOLOX, **RandomHorizontalFlip, ColorJitter and multi-scale** were used in the data augmentation part. **RandomResizedCrop**, which was used for data augmentation in previous YOLO versions, is **no longer used**. YOLOX does not use **RandomResizedCrop** as the **RandomResizedCrop strategy** is countered by **Mosaic augmentation**.

Another data augmentation method added to YOLOX is **MixUp**. Although **MixUp** was originally a data augmentation method developed for **image classification**, it has **become usable for object detection with BoF**.

As an important point, after the **MixUp and Mosaic data augmentation methods**, **ImageNet pre-training**, which was used before, has been abandoned, train all the following models from scratch.

# End to End Learning

One of the most important points in **machine learning** is **to decide which problem to solve with which algorithm**. There may be **more than one algorithm** to **solve a problem and we need to decide which one to use**. **End-to-end learning** is a solution to a **comprehensive goal** using a **single and complex deep learning model**.

YOLOX added **2 more convolutional layers** to YOLO. The first one is **one-to-one label assignment** and the second one is **stop gradient**. These ensure that the detector is **end-to-end**.

However, since they cause a **slight decrease in performance and inference speed**, they are not added to the **final version of YOLOX** and are offered as **optional**.

| Models | Coupled Head | Decoupled Head |
|---|---|---|
| Vanilla YOLO | 38.5 | 39.6 |
| End-to-end YOLO | 34.3 (-4.2) | 38.8 (-0.8) |

Table 1: The effect of decoupled head for end-to-end YOLO in terms of AP (%) on COCO.

# Other Backbones

In addition to YOLOX Darknet-53, it also performs feature extraction using backbones of different sizes. In addition to YOLOX Darknet-53, it also performs feature extraction using backbones of different sizes.

You can check the different YOLOX versions and their performance in the table below.

| Methods | AP (%) | Parameters | GFLOPs | Latency | FPS |
|---|---|---|---|---|---|
| YOLOv3-ultralytics[2] | 44.3 | 63.00 M | 157.3 | 10.5 ms | 95.2 |
| YOLOv3 baseline | 38.5 | 63.00 M | 157.3 | 10.5 ms | 95.2 |
| +decoupled head | 39.6 (+1.1) | 63.86 M | 186.0 | 11.6 ms | 86.2 |
| +strong augmentation | 42.0 (+2.4) | 63.86 M | 186.0 | 11.6 ms | 86.2 |
| +anchor-free | 42.9 (+0.9) | 63.72 M | 185.3 | 11.1 ms | 90.1 |
| +multi positives | 45.0 (+2.1) | 63.72 M | 185.3 | 11.1 ms | 90.1 |
| +SimOTA | **47.3** (+2.3) | 63.72 M | 185.3 | 11.1 ms | 90.1 |
| +NMS free (optional) | 46.5 (-0.8) | 67.27 M | 205.1 | 13.5 ms | 74.1 |

Table 2: Roadmap of YOLOX-Darknet53 in terms of AP (%) on COCO *val*. All the models are tested at $640 \times 640$ resolution, with FP16-precision and batch=1 on a Tesla V100. The latency and FPS in this table are measured without post-processing.

| Models | AP (%) | Parameters | GFLOPs | Latency |
|---|---|---|---|---|
| YOLOv5-S | 36.7 | 7.3 M | 17.1 | 8.7 ms |
| YOLOX-S | **39.6** (+2.9) | 9.0 M | 26.8 | 9.8 ms |
| YOLOv5-M | 44.5 | 21.4 M | 51.4 | 11.1 ms |
| YOLOX-M | **46.4** (+1.9) | 25.3 M | 73.8 | 12.3 ms |
| YOLOv5-L | 48.2 | 47.1 M | 115.6 | 13.7 ms |
| YOLOX-L | **50.0** (+1.8) | 54.2 M | 155.6 | 14.5 ms |
| YOLOv5-X | 50.4 | 87.8 M | 219.0 | 16.0 ms |
| YOLOX-X | **51.2** (+0.8) | 99.1 M | 281.9 | 17.3 ms |

Table 3: Comparison of YOLOX and YOLOv5 in terms of AP (%) on COCO. All the models are tested at $640 \times 640$ resolution, with FP16-precision and batch=1 on a Tesla V100.

| Models | AP (%) | Parameters | GFLOPs |
|---|---|---|---|
| YOLOv4-Tiny [30] | 21.7 | 6.06 M | 6.96 |
| PPYOLO-Tiny | 22.7 | 4.20 M | - |
| YOLOX-Tiny | **32.8** (+10.1) | 5.06 M | 6.45 |
| NanoDet[3] | 23.5 | 0.95 M | 1.20 |
| YOLOX-Nano | **25.3** (+1.8) | 0.91 M | 1.08 |

Table 4: Comparison of YOLOX-Tiny and YOLOX-Nano and the counterparts in terms of AP (%) on COCO *val*. All the models are tested at $416 \times 416$ resolution.

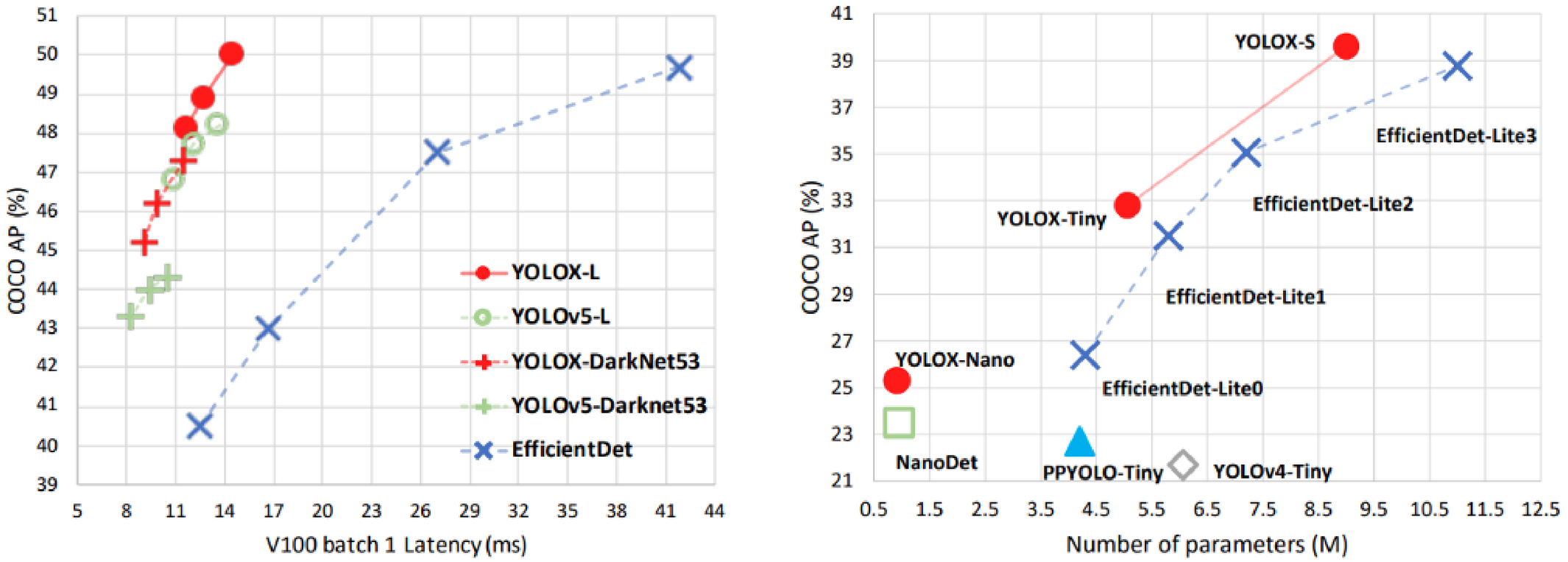



Figure 1: Speed-accuracy trade-off of accurate models (top) and Size-accuracy curve of lite models on mobile devices (bottom) for YOLOX and other state-of-the-art object detectors.