# Deep Learning based Object Detection using YOLOv3 with OpenCV

In this session, we will learn how to use YOLOv3 — a state of the art object detector — with OpenCV.

YOLOv3 is the latest variant of a popular object detection algorithm **YOLO – You Only Look Once**. The published model recognizes 80 different objects in images and videos, but most importantly it is super-fast and nearly as accurate as Single Shot MultiBox (SSD).

Starting with OpenCV 3.4.2, you can easily use YOLOv3 models in your own OpenCV application.

## **How does YOLO work?**

We can think of an object detector as a combination of a **object locator** and an **object recognizer**.

In traditional computer vision approaches, a sliding window was used to look for objects at different locations and scales. Because this was such an expensive operation, the aspect ratio of the object was usually assumed to be fixed.

Early Deep Learning based object detection algorithms like the R-CNN and Fast R-CNN used a method called [Selective Search](https://learnopencv.com/selective-search-for-object-detection-cpp-python/) to narrow down the number of bounding boxes that the algorithm had to test.

Another approach called Overfeat involved scanning the image at multiple scales using sliding windows-like mechanisms done convolutionally.

This was followed by Faster R-CNN that used a Region Proposal Network (RPN) for identifying bounding boxes that needed to be tested. By clever design the features extracted for recognizing objects, were also used by the RPN for proposing potential bounding boxes thus saving a lot of computation.

YOLO on the other hand approaches the object detection problem in a completely different way. It forwards the whole image only once through the network. SSD is another object detection algorithm that forwards the image once though a deep learning network, but YOLOv3 is much faster than SSD while achieving very comparable accuracy. YOLOv3 gives faster than real-time results on a M40, TitanX or 1080 Ti GPUs.

### **Lets see how YOLO detects the objects in a given image.**

First, it divides the image into a 13×13 grid of cells. The size of these 169 cells vary depending on the size of the input. For a 416×416 input size that we used in our experiments, the cell size was 32×32. Each cell is then responsible for predicting a number of boxes in the image.

For each bounding box, the network also predicts the confidence that the bounding box actually encloses an object, and the probability of the enclosed object being a particular class.

Most of these bounding boxes are eliminated because their confidence is low or because they are enclosing the same object as another bounding box with very high confidence score. This technique is called **non-maximum suppression**.

The authors of [YOLOv3](https://pjreddie.com/media/files/papers/YOLOv3.pdf), Joseph Redmon and Ali Farhadi, have made YOLOv3 faster and more accurate than their previous work [YOLOv2](https://pjreddie.com/media/files/papers/YOLO9000.pdf). YOLOv3 handles multiple scales better. They have also improved the network by making it bigger and taking it towards residual networks by adding shortcut connections.

## **Why use OpenCV for YOLO ?**

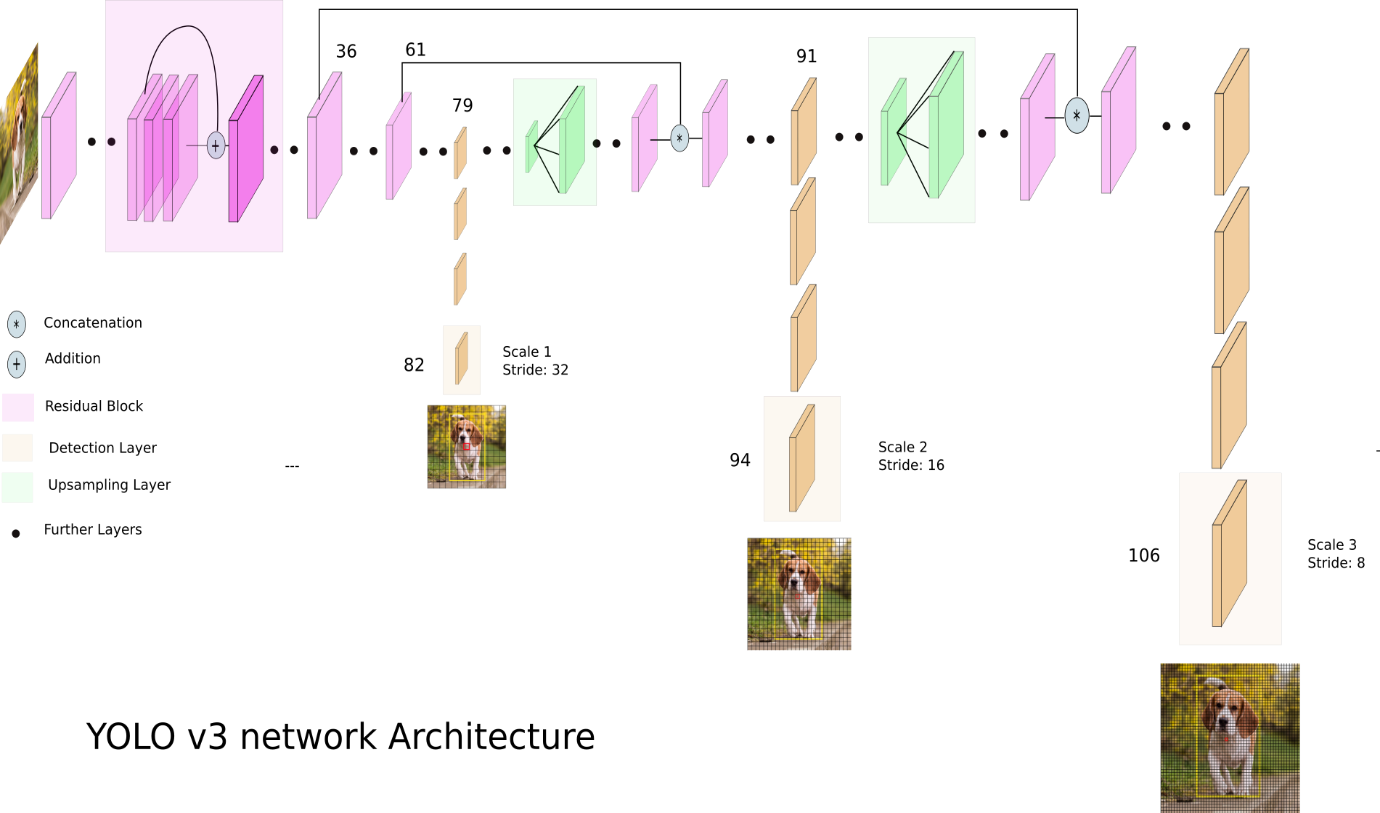
Here are a few reasons you may want to use OpenCV for YOLO

1. **Easy integration with an OpenCV application**: If your application already uses OpenCV and you simply want to use YOLOv3, you don’t have to worry about compiling and building the extra Darknet code.
2. **OpenCV CPU version is 9x faster**: OpenCV’s CPU implementation of the DNN module is astonishingly fast. For example, Darknet when used with OpenMP takes about 2 seconds on a CPU for inference on a single image. In contrast, OpenCV’s implementation runs in a mere 0.22 seconds! Check out table below.
3. **Python support**: Darknet is written in C, and it does not officially support Python. In contrast, OpenCV does. There are python ports available for Darknet though.

Links for your reference

<https://pjreddie.com/darknet/yolo/>

<https://cv-tricks.com/how-to/running-deep-learning-models-in-opencv/amp/>

****