**You Only Look Once: Unified, Real-Time Object Detection**

*Joseph Redmon∗ , Santosh Divvala∗†, Ross Girshick¶ , Ali Farhadi∗*

* Source: <https://arxiv.org/pdf/1506.02640.pdf>
* Current detection systems repurpose classification models to perform detection.

The basic idea is that you apply the classifier to various locations and scales in the input image rather than applying it to the whole image.

* More recent approaches, like R-CNN, use region proposals methods to generate regions of interest and then run a classifier only on those regions.
* YOLO treats object detection as a single regression problem, predicting bounding boxes and their corresponding classes directly from image pixels.
* Three characteristics:
  + Fast: Since the model doesn’t have complex pipeline or region proposal module, it is very fast. Base YOLO runs at 45 frames/sec and Fast YOLO runs at 155 frames/sec.

Because of this streaming videos can be processed by YOLO, with a small latency.

* + Contextual Information: YOLO takes a whole image as input, so it gets all the contextual information, unlike sliding window and region-proposal approaches.
  + Generalizability: Authors trained YOLO on images of various objects and tested it on artwork containing these objects. The model worked well even on artwork, so YOLO learns generalizable features.
* YOLO still lags behind SOTA detection systems in accuracy. It is fast but struggles to localize objects, especially small ones.
* The model **logically** divides an image into an S\*S grid (S can be changed as per requirement).
* An object in an image is assigned to the grid cell that contains the center of the object. So, that cell is responsible for detecting that object.
* **Each grid cell** predicts B bounding boxes and confidence scores for all these boxes.
* Confidence score specifies whether there is any object in the predicted box or not, and if there is an object, how confident the model is about it. It is defined as below:

, if there is no object, and , if there is an object.

(<https://datascience.stackexchange.com/a/87370/120153>

It seems the network outputs only which is then multiplied by to get the **confidence score**)

* Each bounding box prediction has five numbers: x, y, w, h, confidence.

x and y specify the center of a bounding box **relative to the bounds of the grid cell**.

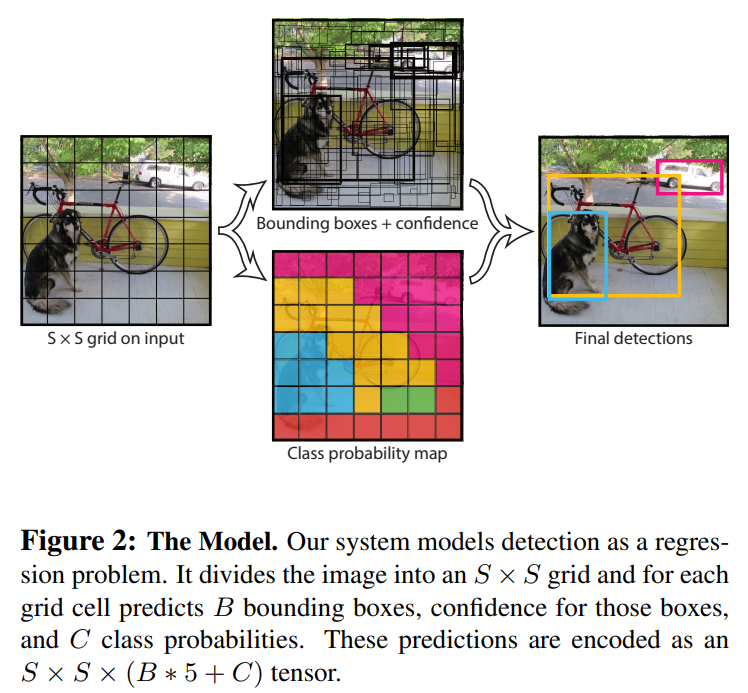
w and h specify the width and height of the bounding box **relative to the whole image**.

Confidence score represents the IOU between the predicted bounding box and any ground-truth box.

* **Each cell** also predicts C conditional probabilities, – probability that a cell contains an object of class given that there is an object.

**Note: There are B bounding boxes per grid cell, but there is a single set of class probabilities per cell.**

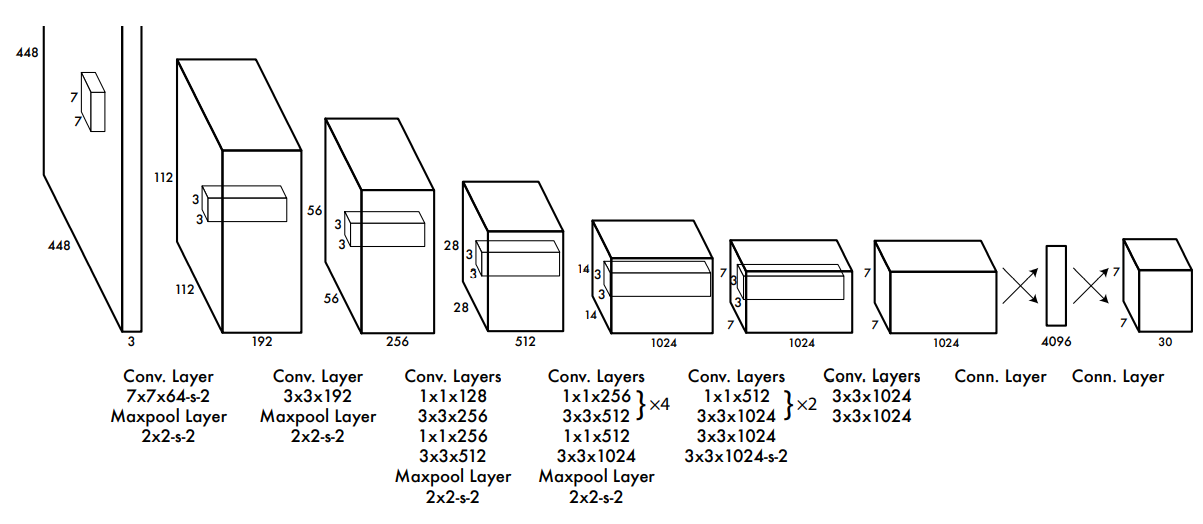
* To get the class-specific scores for each bounding box, we multiply the bounding box confidence score with the conditional class probabilities of the cell.

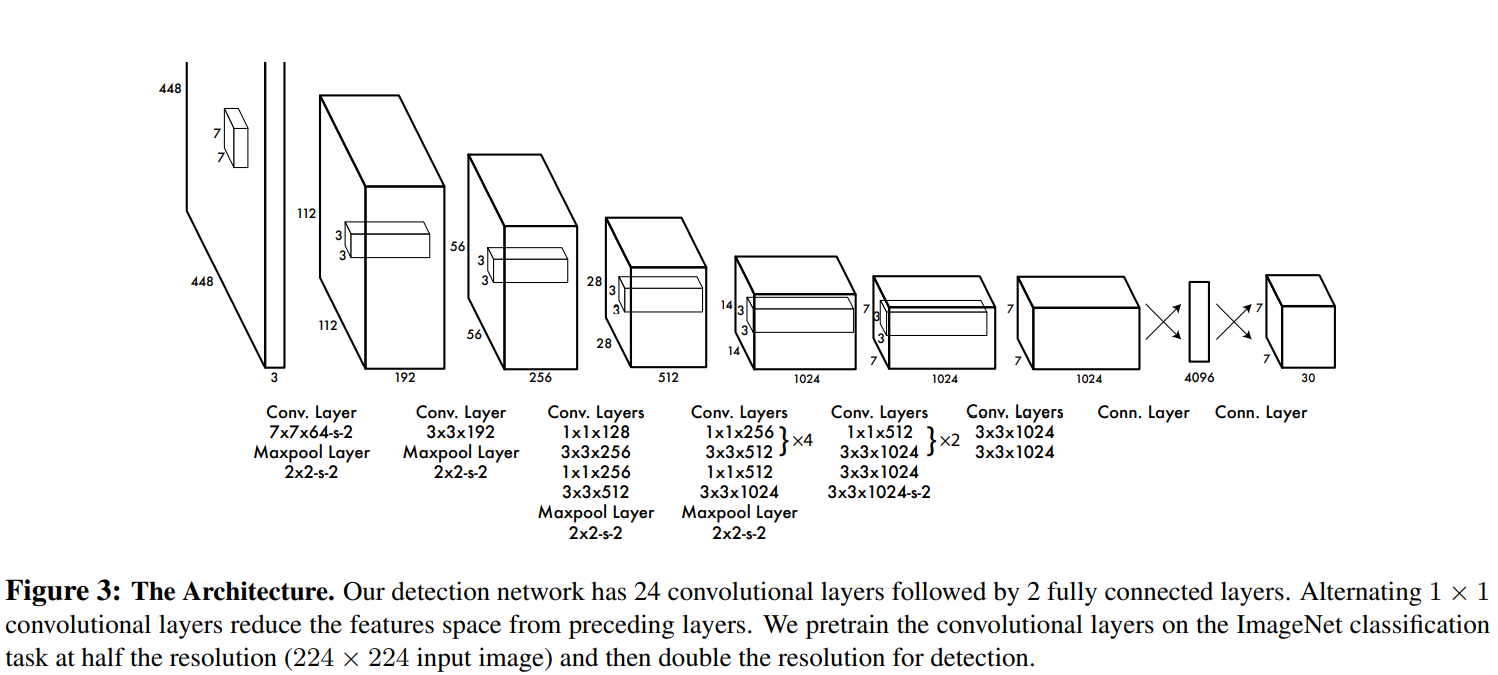


* On Pascal VOC, which has 20 classes, authors used S=7 and B=2.

So, the output size becomes 7\*7\*30, where 30 = 20 (i.e. #classes) + 5\*2 (i.e. B)

* YOLO has 24 conv. layers followed by 2 FC layers
* It is inspired by GoogLeNet, but it doesn’t use inception modules; it uses 1\*1 conv layer after 3\*3 conv. layer
* Fast YOLO has 9 conv. layers and fewer filters in those layers.





* Some clarifications about the architecture above:
  + To get 112\*112 output after the first conv-max pool layer, the input 448\*448 must be padded by 3
  + The output of the first conv-max pool layer has shape 112\*112\*64 and not 112\*112\*192 as shown in the above figure.
  + Similarly, the output of the second conv-max pool layer has shape 56\*56\*192 and not 56\*56\*256
  + Each 3\*3 conv. layer in the above network uses ‘SAME’ padding (i. e. padding of 1 because filter size is 3)
  + The output of the last conv layer 7\*7\*1024 is flattened and then passed to a FC layer having 4096 neurons.
  + Then, there is one more FC layer having 1470 neurons, whose output is reshaped to 7\*7\*30
  + Since output has shape 7\*7\*30, it means we have grid size of 7\*7 and 30 output values per grid cell. 30 comes from 20 class probabilities and two boxes, each having 5 values.
  + Also note that YOLOv1 assumes that there is at max one object per cell. Therefore, we have only one set of probabilities per cell even though we have two bounding boxes per cell.

*(Below is just for intuition)*

So, during training, we hope one box will adapt to detect wide objects and the other box will adapt to detect tall objects.

* Training:
  + Create a network containing 20 conv. layers from the above architecture, which are then followed by an average-pooling layer and a FC layer.
  + Train this model on ImageNet dataset having 1000 classes.
  + Authors stopped training after getting 88% top-5 accuracy.
  + After pre-training, add 4 conv. layers and 2 FC layers with random initialization.

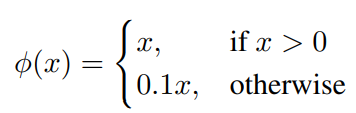
(Note that there is no softmax layer because YOLO treats the whole detection problem as a regression task)

* + Increase the input image resolution from 224\*224 to 448\*448
  + The final layer predicts the class probabilities as well as bounding box coordinates.
  + Bounding box width and height are normalized using image width and height. So, they are in the range [0, 1]
  + Similarly, center of a bounding box is specified as an offset from that grid cell’s location, so its coordinates are also in the range [0, 1]

(Because of the above normalization, you may need to transform ground-truth boxes that are present in your dataset.

Generally, a dataset contains all the ground-truth box coordinates relative to the whole image. This is because depending on the grid size that you use, normalized coordinates will differ. So, once you choose your grid size, you need to transform the ground-truth box coordinates according to the grid size chosen to be able to use them for training/testing)

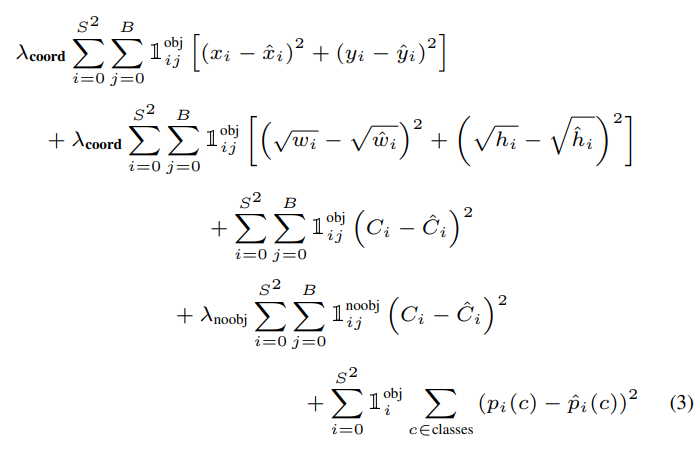
* + Final layer uses linear activation function, while all the remaining layers use leaky ReLU defined as:



* + Using sum-squared error gives equal weightage to localization error and classification error, which may not be ideal. Moreover, there may be cells that don’t contain any object.

To deal with these issues, increase the loss from bounding box coordinate predictions and decrease the loss from confidence scores for boxes that don’t contain objects. For this, use two parameters and

* + YOLO predicts two bounding boxes per cell. Because of this, multiple bounding boxes in an image may detect an object, even though we want only single bounding box per object. To deal with this, we find the box that has highest IOU with the ground-truth box. Doing this will lead to specialization of the bounding box predictors. Each predictor gets better at predicting object of certain sizes, aspect ratios, or classes.
  + Loss function:



where denotes if object appears in cell and denotes that the th bounding box predictor in cell is “responsible” for that prediction.

Loss function only penalizes classification error if an object is present in that grid cell.

It also only penalizes bounding box coordinate error if that predictor is “responsible” for the ground truth box.

* Some clarifications about the loss function above:
  + is an identity function that evaluates to 1 if the bounding box in the cell is responsible for predicting the object in the cell.
  + Since we can have large as well as small bounding boxes, taking square of the difference between w’s and h’s will be very large for large bounding boxes compared to that for small bounding boxes. This means that even a small difference in the large bounding boxes w’s and h’s will affect the loss much more than the amount of difference in the small box’s case.

Thus, authors take square root of w and h before subtracting and squaring.

* + The third term in the loss indicates that when there is an object in the cell , the square of the difference between the confidence scores is calculated **only for bounding box that is responsible for detecting the object.**
  + However, when there is no object in a cell, the above won’t execute; the fourth term will get executed. This term is executed **for all the bounding boxes** in the cell
  + Batch size: 64, momentum: 0.9, decay: 0.0005, and epochs: 135
  + Use dropout and data augmentation to prevent overfitting. Dropout of 0.5 after the first FC layer prevents *co-adaptation* between layers.
* Inference
  + For Pascal VOC, model predicts 98 (=7\*7\*2) bounding boxes and their confidence scores per image. It also predicts 49 (=7\*7) sets of class probabilities.
  + Generally, it is clear which grid cell an object falls in and the network predicts only one bounding box per object. However, some large objects or objects near the edges of multiple cells may be detected by multiple cells. Use non-max suppresion to deal with this.
* Limitations:
  + **Each grid cell only predicts two boxes and can have only one class.** This spatial constraint limits the number of nearby objects that the model can predict. It struggles with small objects that appear in groups, such as flocks of birds. (increasing grid size can alleviate the problem to some extent, but it leads to more computations)
  + Since the model learns from training data, it struggles to generalize to objects in new or unusual aspect ratios.
  + The loss function treats errors in small and large bounding boxes the same, which is not ideal. A small error in large bounding box is benign compared to the same error in small bounding box.
* Error analysis:
  + YOLO struggles with localization.
  + Localization errors are more than all other errors combined.
* Combining Fast R-CNN and YOLO:
  + Since YOLO makes far fewer background errors compared to Fast R-CNN, we can use YOLO to eliminate background errors from Fast R-CNN.
  + This increases accuracy.
  + So, if you want lower localization errors and time is not much of a concern, you should use Fast R-CNN, and to eliminate the background errors in Fast R-CNN, you should use YOLO. Since YOLO is very fast it wouldn’t introduce much time difference but will eliminate few of the errors made by Fast R-CNN
* Authors connected YOLO to a webcam and verified that it is real-time indeed.
* Even though YOLO processes images individually, it can be used for tracking since it is very fast.