

TRAINING REPORT
On
OBJECT DETECTION USING YOLO

Submitted to MAHARAJA RANJIT SINGH PUNJAB TECHNICAL
UNIVERSITY in partial fulfillment of the requirement for the award of
the degree of

B.TECH
in
COMPUTER SCIENCE & ENGINEERING

Submitted By

KASHIF REZA
15110171

CHANDRA BHUSHAN
15110151

ANKIT KR GAUTAM
15110143



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GIANI ZAIL SINGH CAMPUS COLLEGE OF ENGINEERING
& TECHNOLOGY, BATHINDA-151001

May 2019

PREFACE

Project is an integral part of B.Tech. Each and every student has to undergo the project for last semester in the college.

This record is concerned about our project during the 8th semester of our B.Tech. We have taken our project in (Deep Learning, Computer Vision).

During this project, we got to learn many new things about the technology and the current requirements of skills. This project proved to be a milestone in our knowledge of present industry. Every moment was an experience in itself, an experience which theoretical study can't provide.

ACKNOWLEDGEMENT

It is our pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced our thinking, behaviour and acts during the course of study.

We express our sincere gratitude to ***Dr.Dinesh Kumar*** worthy HOD and ***Mrs. Jyoti Gill***, our project In-charge for providing us an opportunity to undergo final year project at GSZCCET.

We are thankful to ***Mrs. Jyoti Gill*** for support, cooperation, and motivation provided to us during the training for constant inspiration, presence and blessings.

We also extend our sincere appreciation to her who provided her valuable suggestions and precious time in accomplishing our project report.

Lastly, We would like to thank the almighty and our parents for their moral support and our friends with whom we shared our day-to day experience and received lots of suggestions that quality of work.

ANKIT KUMAR GAUTAM(15110143)

CHANDRA BHUSHAN(15110151)

KASHIF REZA(15110171)

CANDIDATE’S DECLARATION

We, Ankit Kumar Gautam Roll No. 15110143, Chandra Bhushan, Roll No. 15110151, Kashif Reza Roll No. 15110171, B.Tech(Semester-VII) of the **Giani Zail Singh Campus College of Engineering & Technology, Bathinda** hereby declare that Project Report entitled “**Object Detection Using YOLO**” is an original work and data provided in study is authentic to the best of our knowledge. This report has not been submitted to any other Institute for the award of any other degree.

Ankit Kumar Gautan

(Roll No 15110143)

Chandra Bhushan

(RollNo. 15110151)

Kashif Reza

(Roll No. 15110171)

Place: BATHINDA

Date: 28-05-2019

INDEX

S. No.	Topic	Page No.
1.	Introduction	10
2.	Unified Detection	12
	2.1 Design	13
	2.2 Bonding Box Prediction	14
	2.3 Class Prediction	15
	2.4 Prediction across Scales	16
	2.5 Feature Extractor	17
	2.6 Training	18
	2.7 Inference	20
	2.8 Limitation	21
3.	Comparison to other detection system	22
	3.1 Displaying accuracy/speed trade-off	23
4.	Problem Statement	25
	4.1 Comparison to other real time systems	25
	4.2 Combining Fast-RCNN and YOLO	26
	4.3 Real-Time detection in wild	27
6.	Conclusion and Future Work	28
7.	RESULT	29
	Reference	32

LIST OF FIGURES

Fig. No.	Figure Name	Page No.
2.1	System Models	13
2.2	Architecture	14
2.3	Bounding Box Prediction	15
4.1	Real-Time Object Detection	28
7.1	Loading YOLO architecture and Trained Weight	29
7.2	Showing YOLO Architecture	29
7.3	Detecting Object in Image	30
7.4	Output Image	30
7.5	Detecting Object from Live Webcam	31

LIST OF TABLES

Table. No.	Table Name	Page No.
2.1	Number of Classes	16
2.2	DarkNet-53	17
2.3	Comparison of backbones	18
4.1	Real-Time system on PASCAL VOC 2007	26
4.2	Model Combination experiments on VOC 2007	26
4.3	PASCAL VOC 2012 LeaderBoard	27

ABSTRACT

Efficient and accurate object detection has been an important topic in the advancement of computer vision systems. With the advent of deep learning techniques, the accuracy for object detection has increased drastically. The project aims to incorporate state-of-the-art technique for object detection with the goal of achieving high accuracy with a real-time performance. A major challenge in many of the object detection systems is the dependency on other computer vision techniques for helping the deep learning based approach, which leads to slow and non-optimal performance. In this project, we use a completely deep learning based approach to solve the problem of object detection in an end-to-end fashion. The network is trained on the most challenging publicly available dataset (DARKNET-53), on which a object detection challenge is conducted annually. The resulting system is fast and accurate, thus aiding those applications which require object detection.

We learn the parameters of the network and compare mean average precision computed from pre-trained network parameters. Furthermore, we propose a post-processing scheme to perform real-time object tracking in live video feeds(webcam).

1. INTRODUCTION

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate, algorithms for object detection would allow computers to drive cars in any weather without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems. Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image.

More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding box, eliminate duplicate detections, and rescore the box based on other objects in the scene. These complex pipelines are slow and hard to optimize because each individual component must be trained separately. We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are. YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection. First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems Second, YOLO reasons globally about the image when making predictions.

Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN. Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected input. All of our training and testing code is open source and available online at [removed for review]. A variety of pretrained models are also available to download.

2. Unified Detection

We unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision. Our system divides the input image into a $S \times S$ grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $\text{Pr}(\text{Object}) * \text{IOU}_{\text{truth pred}}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth. Each bounding box consists of 5 predictions: x , y , w , h , and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, $\text{Pr}(\text{Class}_i | \text{Object})$. These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B . At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\text{Pr}(\text{Class}_i | \text{Object}) * \text{Pr}(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \text{Pr}(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

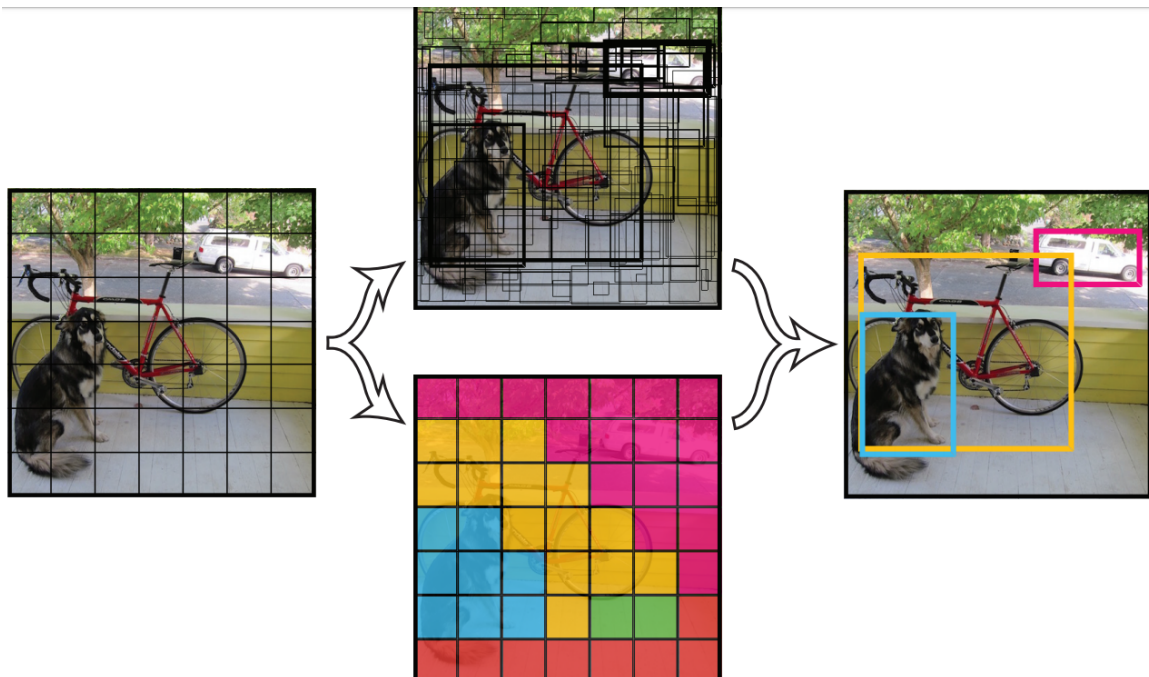


Figure 2.1: The System Model. Our system models detection as a regression problem. It divides the image into an even grid and simultaneously predicts bounding boxes, confidence in those boxes, and class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

For evaluating YOLO on PASCAL VOC, we use $S = 7$, $B = 2$. PASCAL VOC has 20 labelled classes so $C = 20$. Our final prediction is a $7 \times 7 \times 30$ tensor.

2.1. Design

We implement this model as a convolutional neural network and evaluate it on the PASCAL VOC detection dataset, The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. Our network architecture is inspired by the GoogLeNet model for image classification. Our network has 24 convolutional layers followed by 2 fully connected layers. However, instead of the inception modules used by GoogLeNet we simply use 1×1 reduction layers followed by 3×3 convolutional layers, similar to Lin et al. The full network is shown in Figure 3.

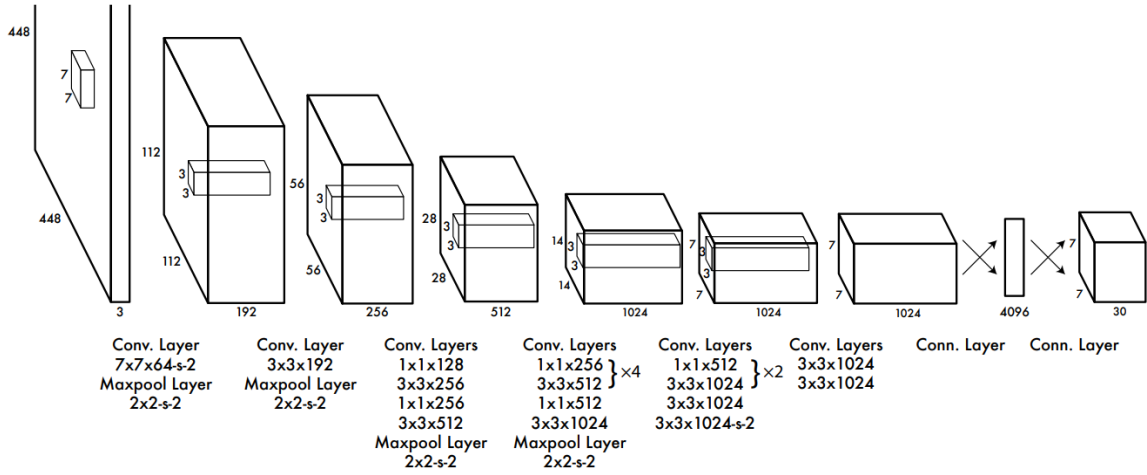


Figure 2.2: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

We also train a fast version of YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolutional layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO. The final output of our network is the $7 \times 7 \times 30$ tensor of predictions.

2.2 Bounding Box Prediction

Following YOLO9000 our system predicts bounding boxes using dimension clusters as anchor boxes. The network predicts 4 coordinates for each bounding box, t_x , t_y , t_w , t_h . If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w , p_h , then the predictions correspond to:

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

During training we use sum of squared error loss. If the ground truth for some coordinate prediction is \hat{t}^* our gradient is the ground truth value (computed from the ground truth box) minus our prediction: $\hat{t}^* - t^*$. This ground truth value can be easily computed by inverting the equations above.

YOLOv3 predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. If the bounding box prior is not the best but does overlap a ground truth object by more than some threshold we ignore the prediction, following

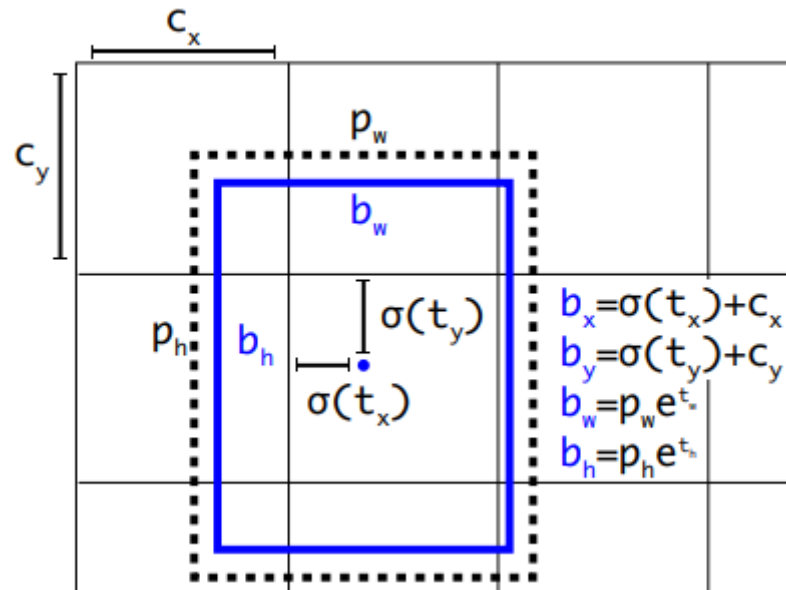


Figure 2.3. Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from

is not the best but does overlap a ground truth object by more than some threshold we ignore the prediction, following. We use the threshold of .5. Unlike our system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness.

2.3 Class Prediction

Each box predicts the classes the bounding box may contain using multilabel classification. We do not use a softmax as we have found it is unnecessary for good performance, instead we simply use independent logistic classifiers. During training we use binary cross-entropy loss for the class predictions.

This formulation helps when we move to more complex domains like the Open Images Dataset. In this dataset there are many overlapping labels (i.e. Woman and Person). Using

a softmax imposes the assumption that each box has exactly one class which is often not the case. A multilabel approach better models the data. As DarkNet-53 can classify into 80 classes given below:

Person	Bicycle	Car	Motorbike	Aeroplane
Bus	Train	Truck	Boat	Traffic
Light	Fire	Hydrant	Stop	Sign
Parking	Meter	Bench	Bird	Cat
Dog	Horse	Sheep	Cow	Elephant
Bear	Zebra	Giraffe	Backpack	Umbrella
Handbag	Tie	Suitcase	Frisbee	Skis
Snowboard	Sports	Ball	Kite	Baseball Bat
Baseball Glove	Skateboard	Surfboard	Tennis	Racket
Bottle	Wine Glass	Cup	Fork	Knife
Spoon	Bow	Banana	Apple	Sandwich
Orange	Broccoli	Carrot	Hot Dog	Pizza
Donut	Cake	Sofa	Potted plant	Bed
Dining Table	Toilet	TV Monitor	Laptop	Mouse
Remote	Keyboard	Cell Phone	Microwave	Oven
Toaster	Sink	Refrigerator	Book	Clock
Vase	Scissors	Teddy Bear	Hair Drier	Toothbrush

Table 2.1 Number of Classes (80 Classes)

2.4 Prediction Across Scales

YOLOv3 predicts boxes at 3 different scales. Our system extracts features from those scales using a similar concept to feature pyramid networks. From our base feature extractor we add several convolutional layers. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. In our experiments with COCO we predict 3 boxes at each scale so the tensor is $N \times N \times [3 * (4 + 1 + 80)]$ for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

Next we take the feature map from 2 layers previous and upsample it by $2\times$. We also take a feature map from earlier in the network and merge it with our upsampled features using concatenation. This method allows us to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. We then add a few more convolutional layers to process this combined feature map, and

eventually predict a similar tensor, although now twice the size. We perform the same design one more time to predict boxes for the final scale. Thus our predictions for the 3rd scale benefit from all the prior computation as well as finegrained features from early on in the network. We still use k-means clustering to determine our bounding box priors. We just sort of chose 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales. On the COCO dataset the 9 clusters were: (10×13) , (16×30) , (33×23) , (30×61) , (62×45) , (59×119) , (116×90) , (156×198) , (373×326) .

2.5. Feature Extractor

We use a new network for performing feature extraction. Our new network is a hybrid approach between the network used in YOLOv2, Darknet-19, and that newfangled residual network stuff. Our network uses successive 3×3 and 1×1 convolutional layers but now has some shortcut connections as well and is significantly larger. It has 53 convolutional layers so we call it.... wait for it..... Darknet-53! This new network is much more powerful than Darknet19 but still more efficient than ResNet-101 or ResNet-152. Here are some ImageNet results:

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

Table 2.2 Darknet-53

Each network is trained with identical settings and tested at 256×256 , single crop accuracy. Run times are measured on a Titan X at 256×256 . Thus Darknet-53 performs on par with state-of-the-art classifiers but with fewer floating point operations and more speed. Darknet-53 is better than ResNet-101 and $1.5\times$ faster. Darknet-53 has similar performance to ResNet-152 and is $2\times$ faster.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78

Table 2.3. Comparison of backbones. Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

Darknet-53 also achieves the highest measured floating point operations per second. This means the network structure better utilizes the GPU, making it more efficient to evaluate and thus faster. That's mostly because ResNets have just way too many layers and aren't very efficient.

2.6. Training

We pretrain our convolutional layers on the ImageNet 1000-class competition dataset. For pretraining we use the first 20 convolutional layers from Figure 3 followed by a average-pooling layer and a fully connected layer.

We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe's Model Zoo. We then convert the model to perform detection. Ren et al. show that adding both convolutional and connected layers to pretrained networks can improve performance .

Following their example, we add four convolutional layers and two fully connected layers with randomly initialized weights. Detection often requires fine-grained visual information so we increase the input resolution of the network from 224×224 to 448×448 . Our final layer predicts both class probabilities and bounding box coordinates.

We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1.

We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1. We use a linear activation

function for the final layer and all other layers use the following leaky rectified linear activation:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

We optimize for sum-squared error in the output of our model. We use sum-squared error because it is easy to optimize, however it does not perfectly align with our goal of maximizing average precision. It weights localization error equally with classification error which may not be ideal. Also, in every image many grid cells do not contain any object.

This pushes the “confidence” scores of those cells towards zero, often overpowering the gradient from cells that do contain objects. This can lead to model instability, causing training to diverge early on. To remedy this, we increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don’t contain objects.

We use two parameters, λ_{coord} and λ_{noobj} to accomplish this. We set $\lambda_{\text{coord}} = 5$ and $\lambda_{\text{noobj}} = .5$. Sum-squared error also equally weights errors in large boxes and small boxes. Our error metric should reflect that small deviations in large boxes matter less than in small boxes.

To partially address this we predict the square root of the bounding box width and height instead of the width and height directly. YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object.

We assign one predictor to be “responsible” for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes, aspect ratios, or classes of object, improving overall recall.

During training we optimize the following, multi-part loss function:

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

where 1_{obj_i} denotes if object appears in cell i and $1_{obj_{ij}}$ denotes that the j th bounding box predictor in cell i is “responsible” for that prediction.

Note that the loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability discussed earlier). It also only penalizes bounding box coordinate error if that predictor is “responsible” for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell).

We train the network for about 135 epochs on the training and validation data sets from PASCAL VOC 2007 and 2012. When testing on 2012 we also include the VOC 2007 test data. Throughout training we use a batch size of 64, a momentum of 0.9 and a decay of 0.0005. Our learning rate schedule is as follows: For the first epochs we slowly raise the learning rate from 10^{-3} to 10^{-2} .

If we start at a high learning rate our model often diverges due to unstable gradients. We continue training with 10^{-2} for 75 epochs, then decrease to 10^{-3} for 30 epochs, and finally decrease again to 10^{-4} for 30 epochs.

To avoid overfitting we use dropout and extensive data augmentation. A dropout layer with rate = .5 after the first connected layer prevents co-adaptation between layer.

For data augmentation we introduce random scaling and translations of up to 20% of the original image size. We also randomly adjust the exposure and saturation of the image by up to a factor of 1.5 in the HSV color space.

2.7. Inference

Just like in training, predicting detections for a test image only requires one network evaluation. On PASCAL VOC the network predicts 98 bounding boxes per image and class probabilities for each box. YOLO is extremely fast at test time since it only requires a single network evaluation, unlike classifier-based methods.

The grid design enforces spatial diversity in the bounding box predictions. Often it is clear which grid cell an object falls in to and the network only predicts one box for each object.

However, some large objects or objects near the border of multiple cells can be well localized by multiple cells. Non-maximal suppression can be used to fix these multiple detections.

While not critical to performance as it is for R-CNN or DPM, non-maximal suppression adds 2- 3% in mAP.

2.8. Limitations of YOLO

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict.

Our model struggles with small objects that appear in groups, such as flocks of birds. Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations

Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image.

Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations.

3. Comparison to Other Detection Systems

Object detection is a core problem in computer vision. Detection pipelines generally start by extracting a set of robust features from input images (Haar, SIFT, HOG, convolutional features). Then, classifiers or localizers are used to identify objects in the feature space. These classifiers or localizers are run either in sliding window fashion over the whole image or on some subset of regions in the image. We compare the YOLO detection system to several top detection frameworks, highlighting key similarities and differences.

Deformable parts models. Deformable parts models (DPM) use a sliding window approach to object detection. DPM uses a disjoint pipeline to extract static features, classify regions, predict bounding boxes for high scoring regions, etc. Our system replaces all of these disparate parts with a single convolutional neural network. The network performs feature extraction, bounding box prediction, nonmaximal suppression, and contextual reasoning all concurrently. Instead of static features, the network trains the features in-line and optimizes them for the detection task. Our unified architecture leads to a faster, more accurate model than DPM.

R-CNN. R-CNN and its variants use region proposals instead of sliding windows to find objects in images. Selective Search generates potential bounding boxes, a convolutional network extracts features, an SVM scores the boxes, a linear model adjusts the bounding boxes, and non-max suppression eliminates duplicate detections. Each stage of this complex pipeline must be precisely tuned independently and the resulting system is very slow, taking more than 40 seconds per image at test time. YOLO shares some similarities with R-CNN. Each grid cell proposes a potential bounding boxes and scores those boxes using convolutional features. However, our system puts spatial constraints on the grid cell proposals which helps mitigate multiple detections of the same object. Our system also proposes far fewer bounding boxes, only 98 per image compared to about 2000 from Selective Search. Finally, our system combines these individual components into a single, jointly optimized model.

Other Fast Detectors Fast and Faster R-CNN focus on speeding up the R-CNN framework by sharing computation and using neural networks to propose regions instead of Selective Search. While they offer speed and accuracy improvements over R-CNN, both still fall short of real-time performance. Many research efforts focus on speeding up the DPM pipeline. They speed up HOG computation, use cascades, and push computation to GPUs. However, only 30Hz DPM actually runs in real-time.

Instead of trying to optimize individual components of a large detection pipeline, YOLO throws out the pipeline entirely and is fast by design. Detectors for single classes like faces or people can be highly optimized since they have to deal with much less variation. YOLO is a general purpose detector that learns to detect a variety of objects simultaneously.

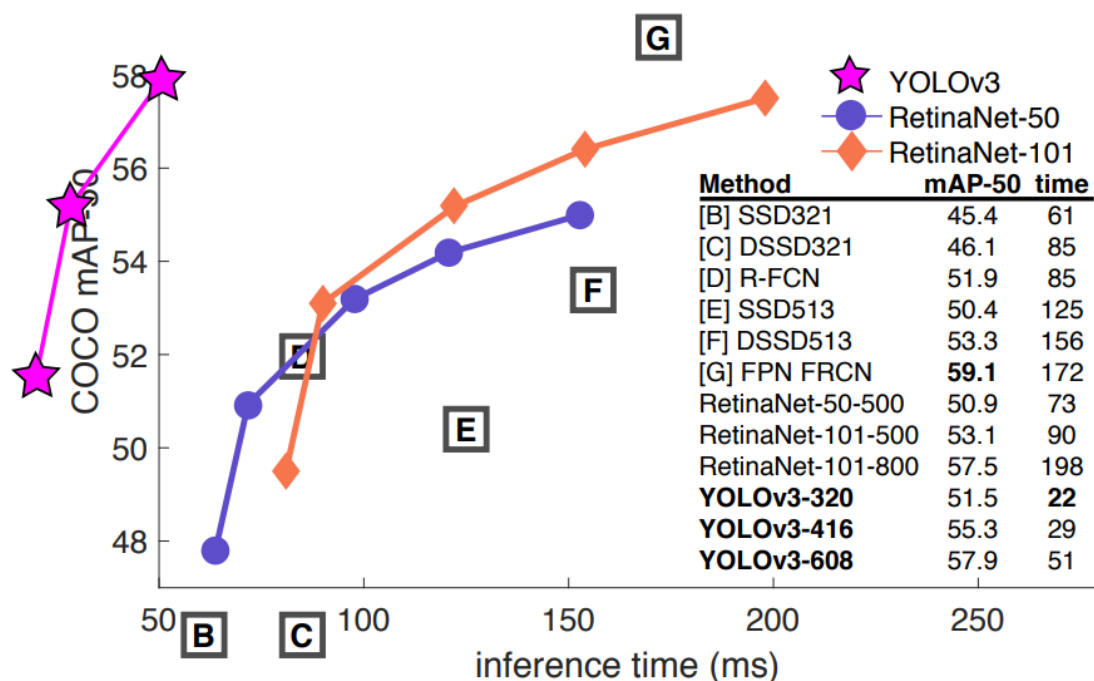


Figure 3.1 Displaying speed/accuracy tradeoff on the mAP at .5 IOU metric.

You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try, this guy. Oh, I forgot, we also fix a data loading bug in YOLOv2, that helped by like 2 mAP. Just sneaking this in here to not throw off layout.

Deep MultiBox. Unlike R-CNN, Szegedy et al. train a convolutional neural network to predict regions of interest instead of using Selective Search. MultiBox can also perform single object detection by replacing the confidence prediction with a single class prediction. However, MultiBox cannot perform general object detection and is still just a piece in a larger detection

pipeline, requiring further image patch classification. Both YOLO and MultiBox use a convolutional network to predict bounding boxes in an image but YOLO is a complete detection system.

OverFeat. Sermanet et al. train a convolutional neural network to perform localization and adapt that localizer to perform detection. OverFeat efficiently performs sliding window detection but it is still a disjoint system. OverFeat optimizes for localization, not detection performance. Like DPM, the localizer only sees local information when making a prediction. OverFeat cannot reason about global context and thus requires significant post-processing to produce coherent detections.

MultiGrasp. Our work is similar in design to work on grasp detection by Redmon et al. Our grid approach to bounding box prediction is based on the MultiGrasp system for regression to grasps. However, grasp detection is a much simpler task than object detection. MultiGrasp only needs to predict a single graspable region for an image containing one object. It doesn't have to estimate the size, location or boundaries of the object or predict it's class only find a region suitable for gripping. YOLO predicts both bounding boxes and class probabilities for multiple objects of multiple classes in an image.

4. Problem Statement

4.1 Methodology

First we compare YOLO with other real-time detection systems on PASCAL VOC 2007. To understand the differences between YOLO and R-CNN variants we explore the errors on VOC 2007 made by YOLO and Fast R-CNN, one of the highest performing versions of R-CNN. Based on the different error profiles we show that YOLO can be used to rescore Fast R-CNN detections and reduce the errors from background false positives, giving a significant performance boost. We also present VOC 2012 results and compare mAP to current state-of-the-art methods. Finally, we show that YOLO generalizes to new domains better than other detectors on two artwork datasets.

4.2. Comparison to Other Real-Time Systems

Many research efforts in object detection focus on making standard detection pipelines fast. However, only Sadeghi et al. actually produce a detection system that runs in real-time (30 frames per second or better). We compare YOLO to their GPU implementation of DPM which runs either at 30Hz or 100Hz. While the other efforts don't reach the real-time milestone we also compare their relative mAP and speed to examine the accuracy-performance tradeoffs available in object detection systems. Fast YOLO is the fastest object detection method on PASCAL; as far as we know, it is the fastest extant object detector. With 52.7% mAP, it is more than twice as accurate as prior work on real-time detection. YOLO pushes mAP by an additional 10% while still maintaining real-time performance. Fastest DPM effectively speeds up DPM without sacrificing much mAP but it still misses real-time performance by a factor of 2. It also is limited by DPM's relatively low accuracy on detection compared to neural network approaches. R-CNN minus R replaces Selective Search with static bounding box proposals. While it is much faster than RCNN, it still falls short of real-time and takes a significant accuracy hit from not having good proposals. Fast R-CNN speeds up the classification stage of R-CNN but it still relies on selective search which can take around 2 seconds per image to generate bounding box proposals.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

Table 4.1: Real-Time Systems on PASCAL VOC 2007.

Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

Thus it has high mAP but at 0.5 fps it is still far from realtime.

The recent Faster R-CNN replaces selective search with a neural network to propose bounding boxes, similar to Szegedy et al. In our tests, their most accurate model achieves 7 fps while a smaller, less accurate one runs at 18 fps. The VGG-16 version of Faster R-CNN is 10 mAP higher but is also 6 times slower than YOLO. The ZeilerFergus Faster R-CNN is only 2.5 times slower than YOLO but is also less accurate.

4.3. Combining Fast R-CNN and YOLO

	mAP	Combined	Gain
Fast R-CNN	-	71.8	-
Fast R-CNN (2007 data)	66.9	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	75.0	3.2

Table 2: Model combination experiments on VOC 2007.

We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.

YOLO makes far fewer background mistakes than Fast R-CNN. By using YOLO to eliminate background detections from Fast R-CNN we get a significant boost in performance. For every bounding box that R-CNN predicts we check to see if YOLO predicts a similar box. If it does, we give that prediction a boost based on the probability predicted by YOLO and the overlap between the two boxes. The best Fast R-CNN model achieves a mAP of 71.8% on the VOC 2007 test set. When combined with YOLO, its mAP increases by 3.2% to 75.0%. We also tried combining the top Fast R-CNN model with several other versions of Fast R-CNN. Those ensembles produced small increases in mAP between .3 and .6%, see Table 2 for details. The boost from YOLO is not simply a byproduct of model ensembling since there is little benefit from combining different versions of Fast R-CNN. Rather, it is specifically because YOLO makes different kinds of mistakes at test time that it is so effective at boosting Fast R-CNN's performance. Unfortunately, this combination doesn't benefit from the speed of YOLO since we run each model separately and then combine the results. However, since YOLO is so fast it doesn't add any significant computational time compared to Fast R-CNN.

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR.CNN.MORE.DATA [11]	73.9	85.5	82.9	76.6	57.8	62.7	79.4	77.2	86.6	55.0	79.1	62.2	87.0	83.4	84.7	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet.VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	79.8	87.7	49.6	74.9	52.1	86.0	81.7	83.3	81.8	48.6	73.5	59.4	79.9	65.7
HyperNet.SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	89.4	49.4	75.5	57.0	87.5	80.9	81.0	74.7	41.8	71.5	68.5	82.1	67.2
MR.CNN.S.CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [27]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP.ENS.COCCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	68.8	75.9	71.4
NoC [28]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH.FGS.STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS.NIN.C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS.NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [32]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Table 4.3: PASCAL VOC 2012 Leaderboard. YOLO compared with the full comp4 (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.

4.4. Real-Time Detection In The Wild

YOLO is a fast, accurate object detector, making it ideal for computer vision applications. We connect YOLO to a webcam and verify that it maintains real-time performance, including the time to fetch images from the camera and display the detections. The resulting system is interactive and engaging. While YOLO processes images individually, when attached to a webcam it functions like a tracking system, detecting objects as they move around and change in appearance.

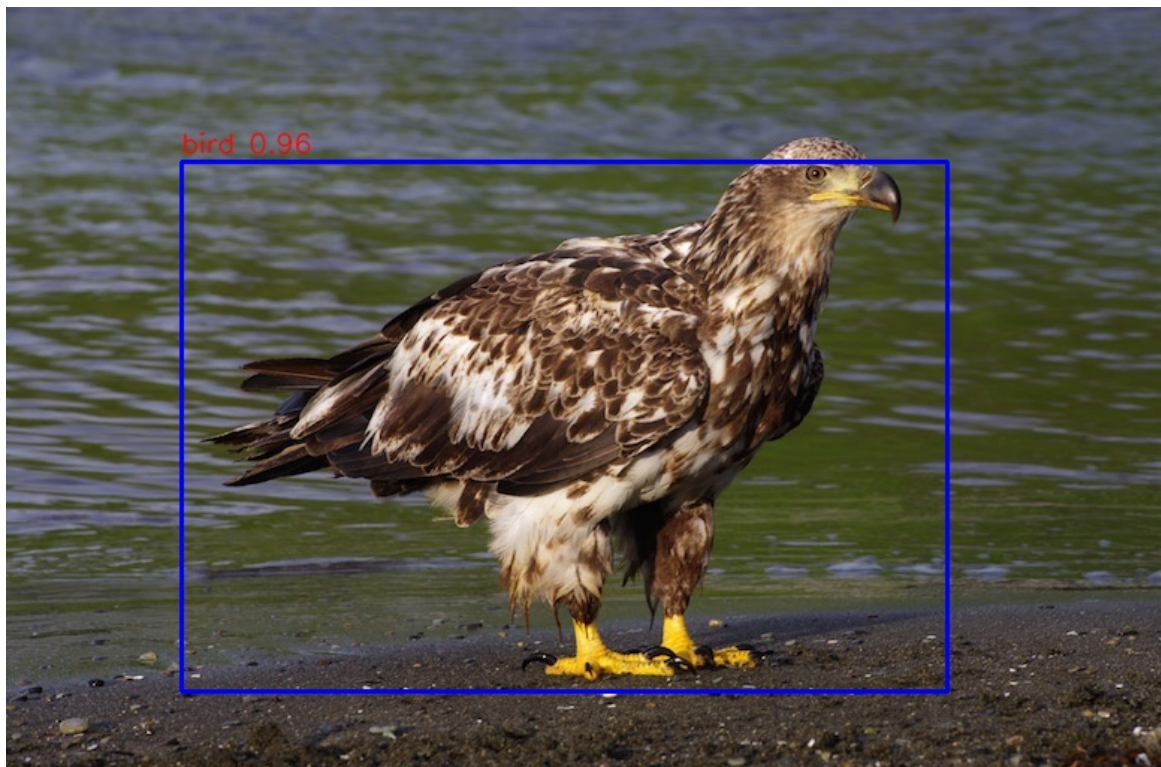


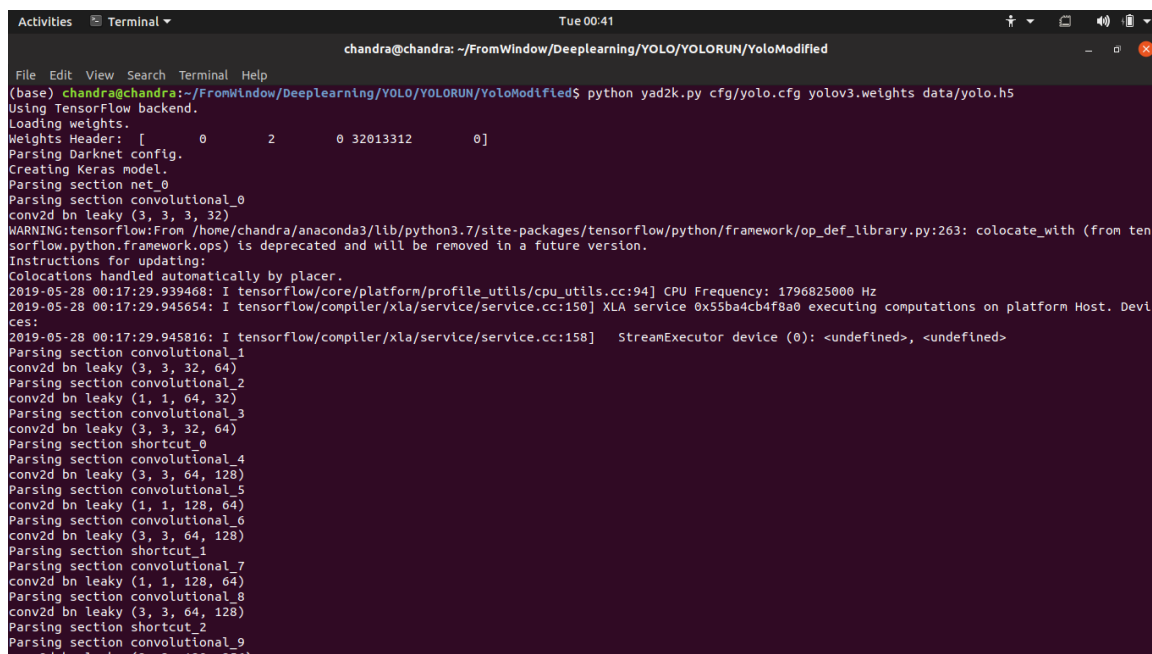
Fig 5.1 Real time object detection

5. Conclusion and Future Work

We introduce YOLO, a unified model for object detection. Our model is simple to construct and can be trained directly on full images. Unlike classifier-based approaches, YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly. Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection. YOLO also generalizes well to new domains making it ideal for applications that rely on fast, robust object detection.

We have demonstrated and verified a functional port of YOLO from darknet to TensorFlow. More work is needed to improve the robustness of the image clustering for real-time video tracking. For example, the existing implementation is limited in the scenarios it can successfully track; image fidelity and frame rate must be high, and objects cannot move at high velocity. More complex algorithms used to calculate image similarity as discussed by would improve tracking capability. In future work, we intend to resolve the ambiguities in the original loss function by contacting the original authors of and report the mAP across all data set splits of our self-trained model.

8.RESULT

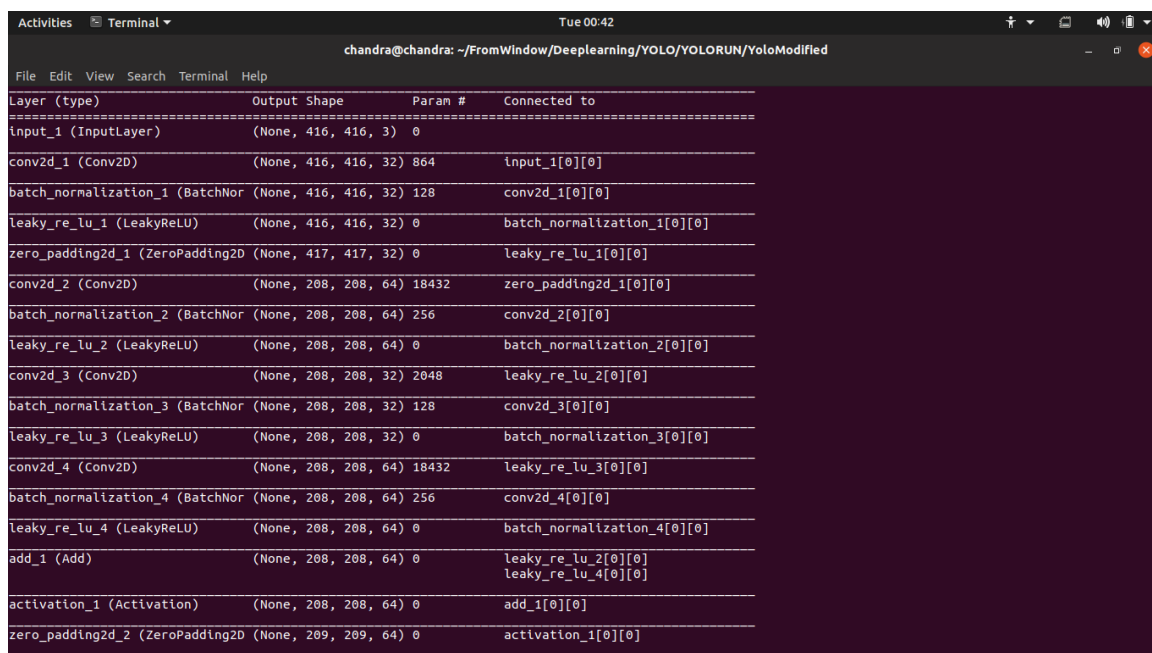


```

Activities Terminal Tue 00:41
chandra@chandra: ~/FromWindow/Deeplearning/YOLO/YOLORUN/YoloModified
File Edit View Search Terminal Help
(base) chandra@chandra:~/FromWindow/Deeplearning/YOLO/YOLORUN/YoloModified$ python yad2k.py cfg/yolo.cfg yolov3.weights data/yolo.hs
Using TensorFlow backend.
Loading weights.
Weights Header: [ 0 2 0 32013312 0]
Parsing Darknet config.
Creating Keras model.
Parsing section net_0
Parsing section convolutional_0
conv2d bn leaky (3, 3, 32)
WARNING:tensorflow:From /home/chandra/anaconda3/lib/python3.7/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
2019-05-28 00:17:29.939468: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 1796825000 Hz
2019-05-28 00:17:29.945654: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x55ba4cb4f8a0 executing computations on platform Host. Devices:
2019-05-28 00:17:29.945816: I tensorflow/compiler/xla/service/service.cc:158] StreamExecutor device (0): <undefined>, <undefined>
Parsing section convolutional_1
conv2d bn leaky (3, 3, 32, 64)
Parsing section convolutional_2
conv2d bn leaky (1, 1, 64, 32)
Parsing section convolutional_3
conv2d bn leaky (3, 3, 32, 64)
Parsing section shortcut_0
Parsing section convolutional_4
conv2d bn leaky (3, 3, 64, 128)
Parsing section convolutional_5
conv2d bn leaky (1, 1, 128, 64)
Parsing section convolutional_6
conv2d bn leaky (3, 3, 64, 128)
Parsing section shortcut_1
Parsing section convolutional_7
conv2d bn leaky (1, 1, 128, 64)
Parsing section convolutional_8
conv2d bn leaky (3, 3, 64, 128)
Parsing section shortcut_2
Parsing section convolutional_9
conv2d bn leaky (3, 3, 128, 256)

```

Fig 8.1 Loading Yolo Architecture and Trained Weight



Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 416, 416, 3)	0	
conv2d_1 (Conv2D)	(None, 416, 416, 32)	864	input_1[0][0]
batch_normalization_1 (BatchNormal	(None, 416, 416, 32)	128	conv2d_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 416, 416, 32)	0	batch_normalization_1[0][0]
zero_padding2d_1 (ZeroPadding2D)	(None, 417, 417, 32)	0	leaky_re_lu_1[0][0]
conv2d_2 (Conv2D)	(None, 208, 208, 64)	18432	zero_padding2d_1[0][0]
batch_normalization_2 (BatchNormal	(None, 208, 208, 64)	256	conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 208, 208, 64)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 208, 208, 32)	2048	leaky_re_lu_2[0][0]
batch_normalization_3 (BatchNormal	(None, 208, 208, 32)	128	conv2d_3[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 208, 208, 32)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 208, 208, 64)	18432	leaky_re_lu_3[0][0]
batch_normalization_4 (BatchNormal	(None, 208, 208, 64)	256	conv2d_4[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 208, 208, 64)	0	batch_normalization_4[0][0]
add_1 (Add)	(None, 208, 208, 64)	0	leaky_re_lu_2[0][0] leaky_re_lu_4[0][0]
activation_1 (Activation)	(None, 208, 208, 64)	0	add_1[0][0]
zero_padding2d_2 (ZeroPadding2D)	(None, 209, 209, 64)	0	activation_1[0][0]

Fig 8.2 Showing YOLO Architecture


```

Activities Terminal Tue 01:43
chandra@chandra: ~/FromWindow/Deeplearning/YOLO/YOLORUN/YoloModified

File Edit View Search Terminal Help
(base) chandra@chandra:~/FromWindow/Deeplearning/YOLO/YOLORUN/YoloModified$ python demo.py
Using TensorFlow backend.
WARNING:tensorflow:From /home/chandra/anaconda3/lib/python3.7/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
2019-05-28 01:37:59.011858: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 1796825000 Hz
2019-05-28 01:37:59.012890: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x558f6b3a89b0 executing computations on platform Host. Devices:
2019-05-28 01:37:59.012974: I tensorflow/compiler/xla/service/service.cc:158] StreamExecutor device (0): <undefined>, <undefined>
/home/chandra/anaconda3/lib/python3.7/site-packages/keras/engine/saving.py:292: UserWarning: No training configuration found in save file: the model was *not* compiled. Compile it manually.
  warnings.warn('No training configuration found in save file: ')
eagle.jpg
time: 7.18s
class: bird, score: 0.96
box coordinate x,y,w,h: [116.86000837 103.06972504 512.02361313 353.46265734]

horses.jpg
time: 3.34s
class: horse, score: 0.99
box coordinate x,y,w,h: [441.10483342 209.1317749 154.83146719 142.6069355 ]
class: horse, score: 0.97
box coordinate x,y,w,h: [ 6.2066293 193.34841919 301.04014944 223.11424828]
class: horse, score: 0.96
box coordinate x,y,w,h: [243.42719969 178.76040649 203.26054293 173.19878945]
class: horse, score: 0.90
box coordinate x,y,w,h: [ 0.83050664 180.769104 153.17852983 175.04206115]

giraffe.jpg
time: 3.33s
class: zebra, score: 0.97
box coordinate x,y,w,h: [271.48172259 197.0153749 157.88637847 260.55867884]
class: giraffe, score: 1.00
box coordinate x,y,w,h: [156.37391806 38.93317282 288.81168674 359.76752309]

takagaki.jpg
time: 3.34s

```

Fig 8.3 Detecting object in Image

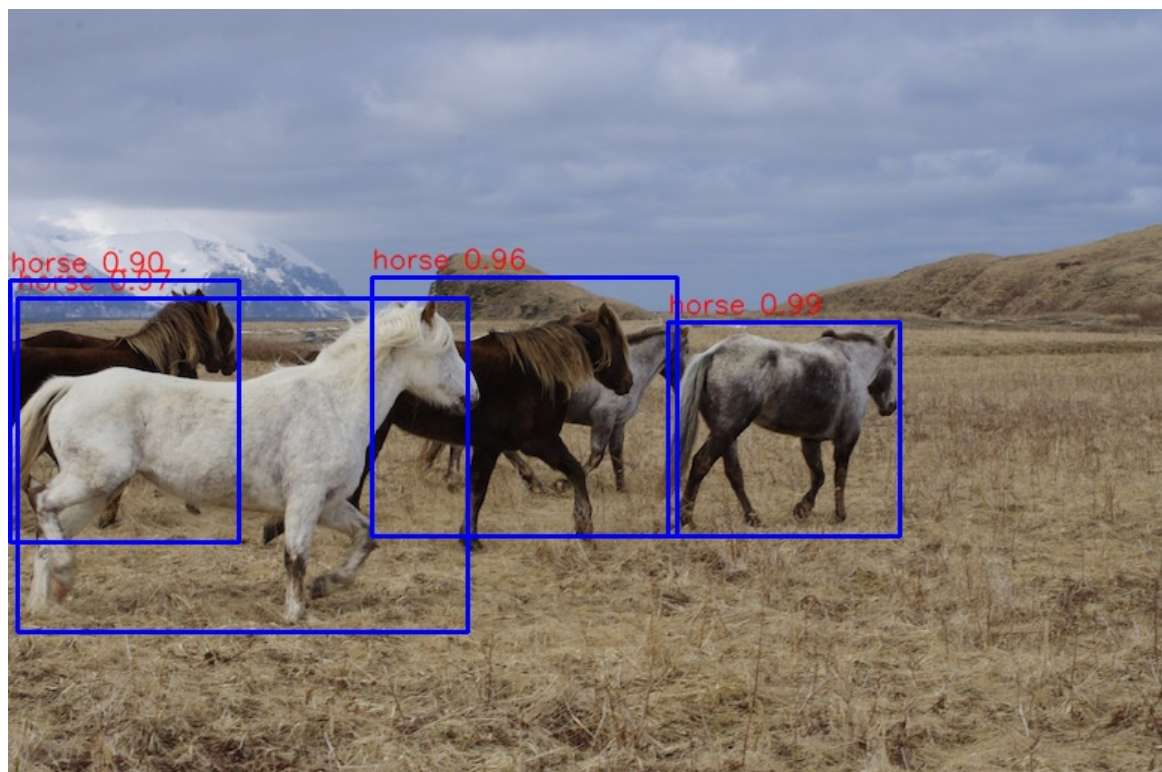


Fig 8.4 Output Result

```

OpenCV: FFMPEG: tag 0x6765706d/'mpeg' is not supported with codec id 2 and format 'mp4 / MP4 (MPEG-4 Part 14)'
OpenCV: FFMPEG: fallback to use tag 0x7634706d/'mp4v'
time: 3.39s
time: 3.31s
time: 3.29s
time: 3.29s
time: 3.36s
time: 3.40s
time: 3.36s
class: person, score: 0.98
box coordinate x,y,w,h: [235.26424408 170.37117004 395.61614972 316.08484351]
class: person, score: 0.97
box coordinate x,y,w,h: [ -4.83496189 148.22374821 296.91390038 335.50715749]

time: 3.36s
class: person, score: 0.99
box coordinate x,y,w,h: [236.66559219 183.51277828 396.66025785 305.72351575]
class: person, score: 0.95
box coordinate x,y,w,h: [ 1.99595451 169.59550381 303.84778976 314.85097069]

time: 3.40s
class: person, score: 0.98
box coordinate x,y,w,h: [241.55052185 181.58527851 387.88634282 308.47227106]
class: person, score: 0.97
box coordinate x,y,w,h: [ -4.12755489 167.07847595 316.6073513 313.0036929 ]

```

Fig 8.5 Detecting object from live Webcam

Reference

- [1] Analogy. Wikipedia, Mar 2018.
- [2] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.
- [3] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.
- [4] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. Iqa: Visual question answering in interactive environments. *arXiv preprint arXiv:1712.03316*, 2017.
- [5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [6] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors.
- [7] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. Dataset available from <https://github.com/openimages>, 2017.
- [8] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.
- [9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.
- [10] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [12] I. Newton. *Philosophiae naturalis principia mathematica*. William Dawson & Sons Ltd., London, 1687.

- [13] J. Parham, J. Crall, C. Stewart, T. Berger-Wolf, and D. Rubenstein. Animal population censusing at scale with citizen science and photographic identification. 2017.
- [14] J. Redmon. Darknet: Open source neural networks in c. <http://pjreddie.com/darknet/>, 2013–2016.
- [15] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 6517–6525. IEEE, 2017.
- [16] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv, 2018.
- [17] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. arXiv preprint arXiv:1506.01497, 2015.
- [18] Udemy OpenCV and Object Detection SSD
- [19] <https://TowardDataScience.com>
- [20] <https://pjreddie.com/darknet/>
- [21] <https://medium.com>