Understanding YOLO and tiny-YOLO for hardware accelerated object detection

Syed Tihaam Ahmad and M.Hanzalah Zia

***Abstract*— The project aimed at understanding the state of the art for the problem of real-time object detection. The algorithm You-only-look-one also known as YOLO .On using Pascal Titan X YOLO gives images at 30 FPS simultaneously giving mean precision accuracy (mAP) of 57.9.To meet the desired improved results in terms of real time detection, we will be looking at tinier version of YOLO which have much more efficient inference on a FPGA. The investigation is focused on answering the various questions that will occur to those seeking to understand the code for YOLO and tinyYOLO.We enjoyed the process of co-relating things taught in theory and then implemented. Understanding YOLO was somewhat easy because of the solid background built during the machine learning course taught in this semester.We learnt multivariate probability and a bit statistical machine learning in the first half of the course which helped us understanding the theory of YOLO and the parameters used in YOLO. It made easy for us to tune the learning parameters in it.We knew how to avoid overfitting and what activation functions to use.The second part taught in the course was based on neural networks like MLP and CNN's which helped us in understanding and implementing the core architecture of YOLO.Now implementing them in a hardware accelerated version was done on a FPGA which was easy for us because of our FYP on the very same thing,which is deployment of different deep learning architecture on FPGA's in an accelerated manner .It was our first time going through machine learning research papers they helped us to get to know the main theory and architecture of the respective models.**

I. Introduction

Object detection has always remain a hot topic in machine learning domain.It is one of the areas of computer vision that is rapidly growing since the boom of DNN in 2012.Every year new architectures outperform the last ones.In the recent 5 years, YOLO,RCNN,Fast RCNN,Mask RCNN have been were built and were state-of-the-art models. All of the models have a basis somehow on CNN as they are the basic building blocks for object detection and computer vision areas.The basic problem of object detection starts from cat or not problem.The problem has been very well solved using even basic models built on CNN. With the evolution of deep learning era,we have jumped from image classification to object classification and localization ,and then to multiple object detection.Multiple objects detection along with their localization can also be done using ConvNet by cropping the image into several images and then running ConvNet for all of the images produced.But this can be very computationally expensive.We can improve the computational power of sliding window method by replacing the FC layers with 1x1 Conv layers moreover we pass the image once for a given windows size.Hence we donot pass the cropped image but we pass the complete image in the actual implementation.We usually slide a stride on the whole image.But the problem with this is the stride we defined may not fit on object of different sizes all over the image.So a better solution in form of YOLO was provided.YOLO is the kind of architecture which is based on multiple object detection along localization of the objects. YOLO involves the following main key steps :

1) Divide image into multiple grids
2) Implement the object detection algorithm on each grid cell made and then localize the object in grids and label data in them

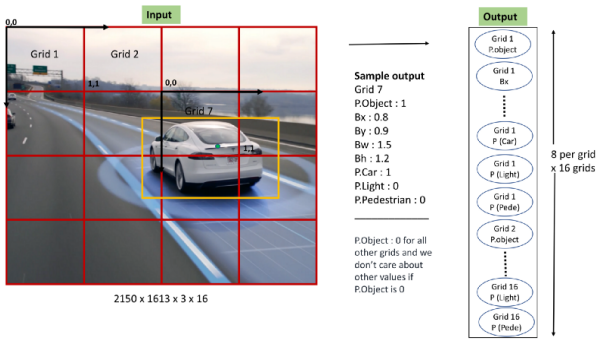

Fig. 1. Grids in YOLO

A. Dataset and Training

- ImageNet-1000 was used initially to pre-train YOLO ConvLayers.In case of inference and training,Darknet framework was used.
- Object detection usually requires high reslution and fine-grained images. Hence,the input resolution of 224x224 images were enhanced to a resolution of 448X448.
- The classifier usually achieves a top-1 accuracy of 76.5and a top-5 accuracy of 93.3after training it in the following manner:
- For 160 epochs we trained the network using Imagenet-1000 dataset along with stochastic gradient decent with weight decay of 5x10E-4 , a initial learning rate of 0.1, and momentum of 0.9.
- Initially,we trained YOLO using 224x224 dataset then we fine tuned the model for high resolution images of 448x448 for 10 epochs.
- After pre-training of Convlayers using imagenet-1000, the FClayers and the last Convlayer is removed for a detector.
- We add a pass-through layer.
- YOLO adds three 3 3 Convlayers with 1024 filters each followed by a final 1 1 Convlayer with 125 output channels.

II. Model Architecture

YOLO architecture consist of 24 Convlayers followed by 2 FC layers.1x1 reduction layers were used followed by 3x3 convlayers. The final tensor has dimensions of 7x7x30

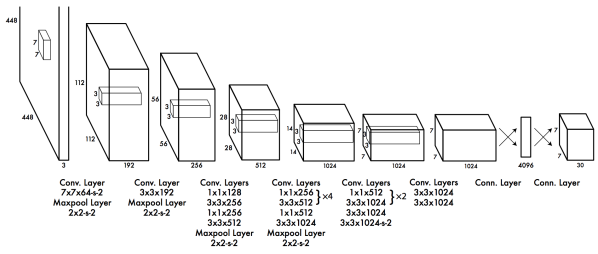

Fig. 2. Architecture of YOLO

III. Limitations

A. Incorrect localization

Incorrect localization can be one of the limitations of YOLO. It has defined bounding box dimensions so spatial constraints as each grid cell only predicts two boxes and can have only one class. For example , it will struggle detecting small objects very close to each other like flocks of birds.It cannot detect multiple objects in same grid but we can choose a smaller grid to solve this problem. But still object detection can fail to work in cases where objects are very close to each other, like in the case of image of herd of sheep.These problems can somehow be rectified using anchor boxes.

B. What are anchor boxes?

In case of multiple objects in same grid cell, one object is assigned to one anchor box in one grid, the others are assigned to the other anchor boxes in the very same grid

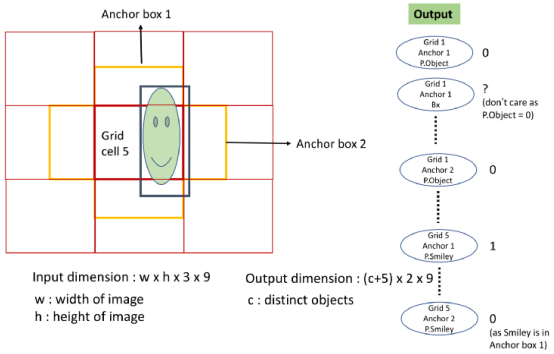

Fig. 3. Anchor boxes

IV. Comparison to Other Models

A. R-CNN

R-CNN uses segmentation and region proposal instead of sliding stride to detect objects in an image.R-CNN involves selective searching drawing bounding boxes,convolutional neural network extracting features, SVM scoring the bounding boxes,Linear model adjusting them and non-max suppression removing the duplicates. This whole lot of steps are tuned independently and hence the real-time object detection becomes very slow in this case.YOLO on the other hand is faster it even suggest less number of bounding boxes due to spatial constraints.

B. Faster R-CNN

Fast R-CNN is a huge improvement over R-CNN in terms of inference or real-time scenes. But it still falls short in real time performance to YOLO.Instead of trying to optimize individual components of a large detection pipeline, YOLO is such a general purpose detector which throws out the pipeline entirely and is fast by design and can be used to detect different objects simultaneously

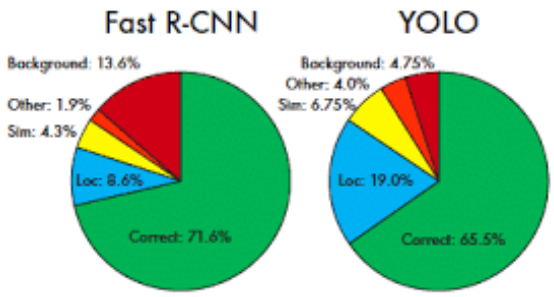

Fig. 4. Fast-RCNN vs YOLO

V. Accelerated Implementation on Hardware like FPGA

A. Learning to use FPGA

We chose a FPGA board called PYNQ-Z1 to test tinyYOLO based on ZYNQ.ZYNQ integrates the power of an ARM-based processor with the parallelization power of an FPGA in terms of hardware, enabling key analytics and hardware acceleration.PYNQ is an open-source project from Xilinx. We can use Python to implement high performance embedded applications with hardware accelerated algorithms,parallel hardware execution,high frame-rate video processing ,high bandwidth IO,low latency control,real-time signal processing etc

B. QNN and tinyYOLO

QNN also known quantized neural network were introduced for training neural networks with low precision weights and activations eg(1-bit weights and activations).So,when we are training, the quantized weights and activations are used for computing the parameter gradients. During the forward pass, QNNs drastically reduce memory size and accesses, and replace most arithmetic operations with bit-wise operations like XNOR.As a result, power consumption is expected to be drastically reduced.Xilinx a fpga vendor also provides QNN library with tinyYOLO example which is just like YOLO but with a smaller architecture and is trained on quantized weights but gives faster inference than YOLO.

C. What is XNOR-Count?

The most basic mathematical operations in a CNN is convolution and matrix multiplication.They are replaced by an XNOR-popcount.The usual multiply-accumulate consists of a dot product between two vectors and adding the resulting numbers. This is replaced by bitwise XNOR of the two vectors which is very fast on hardware level.

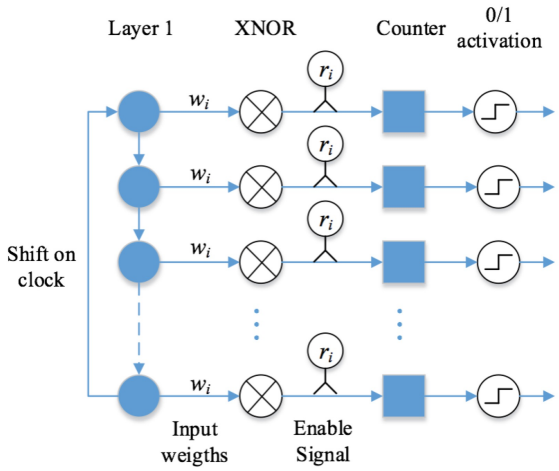

Fig. 5. XNOR operation

VI. Workflow for Implementing TinyYOLO

- Prepare dataset in Darknet format.You can use it to first obtain baseline Float accuracy using Darknet and later use the same dataset in Lightnet.Build Darknet provided by AlexeyAB on GitHub.Place train and test images in different folders, and with corresponding annotations as.txt with the same name as the images. You can use YOLO-Mark by AlexeyAB to confirm your annotation.Create the.namesfile for object classes.Use the scripts present in Darknet to obtain anchors for your own dataset.Create the.cfg and.datafiles needed by Darknet.Launch the training. Weights are saved every few epochs. Use mapmode to get mAP on your test dataset and testing mode to visually see results on a few test images.We will use the saved weights for starting training in Lightnet later.
- Now for Lightnet training.Download modified version of Lightnet framework for quantized training. The changes we made are related to TinyTinier YOLO, quantization, and fixing a few scripts to work.We will change theyolo-voc example to train on our own dataset. Copy dataset into this location.Run make-pickle.py to create pickle annotations.Modify cfg/tinyyolo.py to suit your network parameters.Launch training. The weights are saved every few epochs as.pt.
- Export weights from the.pt with best mAP into.npz using given script.Feed into finnthesizer to get binary weights packed. The thresholds are saved

as.mat files.Use given MATLAB script to save thresholds as.bin properly.Copy the initial and last layer floating convolutional weights from.npz into the binparamfolder.Copy weights onto FPGA SD Card.Create.json,.names and.cfg files on the board SD card.Call given python script with appropriate paths for inference.

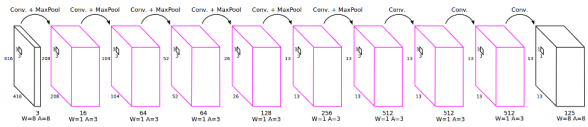

Fig. 6. architecture of tiny yolo

VII. YOLO9000

We came across YOLO9000 paper which is YOLO for 9000 different classes.We found it interesting.Here is a short summary of what we learnt from it. When we compared YOLO to Fast-RCNN we came to know that YOLO has a lot of localization errors.Moreover,in comparison to region proposal based strategies YOLO has a low recall.Hence,we tend to focus chiefly on improving the above described localization errors and recall and simultaneously we maintain/improve mAP. One of the technique used is batch normalisation on all of the convolutional layers in YOLO.From this, we tend to get quite improvement in mAP. Batch normalisation is a regularization technique which helps overcome overfitting. With batch normalisation we are able to take away dropout layers from the model while not overfitting. In case of YOLOv2,we fine tune the model of YOLOv2 using 448x448 images from ImageNet for 10 epochs.This offers the network time to regulate its filters to work better on higher resolution input. We then fine tune the ensuing network on detection.We get an increase in mAP by using this high resolution classification network model. The coordinates of the bounding boxes are predicted using FC layers on top of Conv feature extractor.It becomes easier for model to learn by predicting offsets rather than coordinates .We then remove the FC layers from YOLO and use anchor boxes to predict bounding boxes. using anchor boxes we tend to get a small decrease in accuracy. YOLO solely predicts ninety eight boxes per image however with anchor boxes our model predicts more than one thousand. while not anchor boxes our intermediate model gets 69.5 mAP with a recall of 81. With anchor boxes our model gets 69.2 mAP with a recall of 880. Even though the mAP decreases, the rise in recall means our model has additional room to enhance. Another thing we studied was fine grained technique.Fine-Grained options changed YOLO predicts detections on a 13x13 feature map,whereas this can be sufficient for large objects, it may benefit from finer grained features for localizing smaller objects. Faster R-CNN and SSD each learn at different resolutions at varied feature maps within the network. However,we tend to take a different approach, merely adding a passthrough layer that brings features from an earlier layer at 26x26 resolution. Like we do identity mapping in Resnet model,the passthrough layer is used to concatenate the higher resolution features with the low resolution features by stacking adjacent features into completely different channels rather than spatial locations. This turns the 26x26x512 feature map into a 13x13x2048 feature map, which might be concatenated with the initial features. Our detector runs on top of this expanded feature map in order that it's access to fine grained features. We combine pictures from both detection and classification datasets during training our model.Once our network sees an image labeled for detection we are able to backpropagate based on the complete YOLOv2 loss operate.

VIII. Implementation

It was our first time going through these research papers they helped us to get to know the main theory and architecture of the respective models.After going through the papers,we went for the implementation of the models as described in the papers.We did get help from many github repos and already implemented codes.In the process we first wanted to learn tensorflow and keras. But got to know about pytorch which we found more easy and updated.So,we used pytorch framework to implement one of the YOLO variants YOLOv3.Other libraries which were required to install were python3.5,opencv and pytorch.Due to unavailibilty of a GPU we used pre-trained weights to implement the algorithm.So we built the model using pytorch and then used the weights and inputs to infer our results which we were happy with. It was able to detect different objects in the picture.We did not performed it real time but we did feed the model with the images to test it and it did predicted well. We also tested tinyYOLO on PYNQ which is a FPGA board.Xilinx provides QNN library which includes tinyYOLO we implemented the algorithm on hardware too.We did not notice significant drops in accuracy.

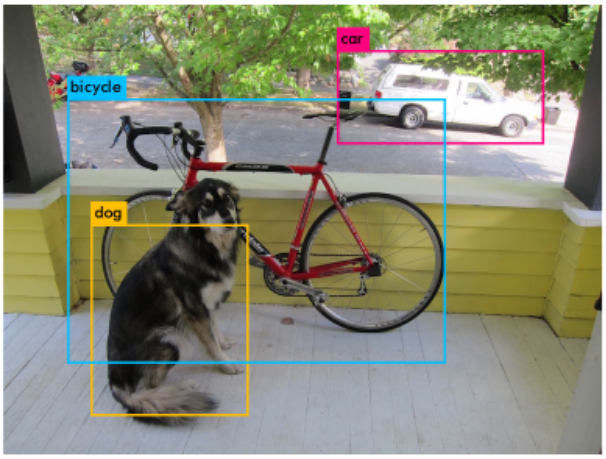

Fig. 7. Result on test image

IX. What is the significance of this project?

One may ask why is this project important.Why are we implementing the 4 years old technique.The answer to that is that as we know that YOLO was state-of-the-art model and we till use it and its variants for object detection.We implemented this model on hardware in an accelerated manner because in real-time object detection latency of inference is of primary importance and we cannot use power hungry devices like GPUs on our real-time systems.Hence we utilize the power of parallelization of layers of model on hardware like FPGA to decrease the latency of inference for real-time object detection. It is extensively used in missile and runway detection.Hence it can be of primary importance in security of a country.

X. Conclusion

The project was inspired from our FYP idea which is deployment of deep neural networks on FPGAs. Hence we can conclude that we were able to completely understand YOLO and its variants. We understood the architecture,data flow,training,and inference of different variants of YOLO. We realized the importance of implementing tinyYOLO with quantized weights and activation bits on a FPGA to get better inference.

XI. Acknowledgements

Different images used above where taken from google.We acknowledge datasets,images,and helped used during the project.We acknowledge medium.com's article to help us begin writing the report in a professional manner.We used overleaf.com to write this report in lateX.We are thankful to them for providing us with free IEEE paper template in lateX.We also are thankful to muhammad umar and ussama zahid from bee-7 who had done the same work last year ,to supervise us.And now we are continuing their work as our FYP

References

[1] You Only Look Once: Unified, Real-Time Object Detection: arXiv:1506.02640
[2] Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, Yoshua Bengio
[3] BNN+: Improved Binary Network Training Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, Vahid Partovi Nia
[4] FINN: A Framework for Fast, Scalable Binarized Neural Network Inference Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, Kees Vissers
[5] YOLO9000: Better, Faster, Stronger Joseph Redmon, Ali Farhadi
[6] YOLOv3: An Incremental Improvement Joseph Redmon, Ali Farhadi
[7] FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks Michaela Blott, Thomas Preusser, Nicholas Fraser, Giulio Gambardella, Kenneth O'Brien, Yaman Umuroglu
[8] XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, Ali Farhadi