CSC 486B Final Project
TensorFlow Implementation of Yolo9000

Jake Runzer V00797175 Benjamin Benetti V00193398 Daniel Truong V00795971 Tristan Partridge V00804280

Abstract

Yolo9000 is an advanced object detection system. It is fast enough to predict objects and bounding boxes in video frames in real-time with accuracy comparable to much slower state of the art systems. To accomplish this, the network only looks at the image a single time to make its prediction and uses a technique of anchor boxes which enables the network to learn object dimensions and locations. Our implementation of Yolo9000 is similar to the original, but was developed using the TensorFlow machine learning framework instead of darknet. After many hours of training our network was able to make inaccurate bounding box predictions, indicating that more training time is needed.

1. Introduction

The goal of this project is to implement a working version of the Yolo9000 [5] object detection network using TensorFlow [1] for CSC486b. The motivation for this project was out of interest. Real-time object detection is required to continue the development of many other advanced systems such as self-driving cars, facial recognition, and image search. The Yolo9000 model is one of the fastest models to date. The first version of Yolo was released in 2015 [3] and the second version, Yolo9000, was released in 2016 [5]. Yolo stands for ”You only look once” and is one of the reasons this network has become so popular. The input image need only be looked at once for a prediction to be made. This makes the network much faster than other object detection models, which required multiple passes of the image, one for bounding box detection and another for classification of the objects inside the bounding boxes. Yolo9000 accomplishes with its clever use of *anchor boxes*, which will be discussed later in this paper.

2. Contributions of the original paper

Yolo9000 is a system able to detect and classify objects in images in real-time. The first released version of the system was simply called Yolo, with Yolo9000 being an improved version of Yolo. The Yolo9000 model processes images fast enough to be able to process video streams in real-time (on suitable hardware) and identify all objects known to the system in every frame. Compared to the original Yolo network, Yolo9000 can process frames faster than its predecessor, performing predictions on over 9000 object categories [5]. One of the biggest improvements introduced by Yolo9000 is the improved, multi-scale training method. This allows for easy tradeoff between speed and accuracy [5].

2.1. Network Architecture

The Yolo architecture is similar to a fully convolutional neural network. The whole image is processed using a single neural network. The CNN architecture is illustrated in Figure 1.

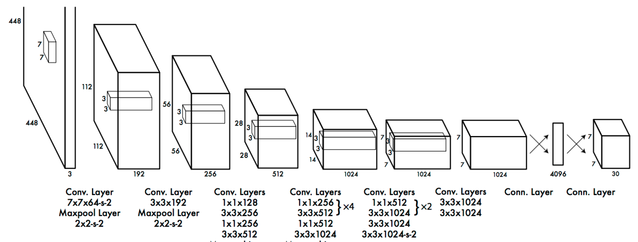
448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 1. The CNN behind Yolo [7].

Each image is processed once, hence the name you only look once. First, Yolo re-sizes the image. It than splits the current input image into an *13 X 13* grid. Each cell in the grid will predict 5 bounding boxes, five confidence scores, and a probability value for each possible class. The five confidence scores represent how sure the network is that the corresponding predicted bounding box encloses some type of object. The class probabilities represent how sure the network is that the bounding box encloses an object of that specific class. The probabilities are multiplied with the confidence values resulting in the bounding boxes weighted by their probabilities for the predicted objects. Boxes with low scores under a determined threshold are removed. The resulting shape of the output is $13 \times 13 \times 125$ and can be seen in Figure 2.

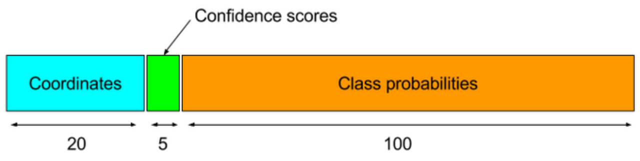

Figure 2. The output of the Yolo Network [2].

2.2. Anchor Boxes

Anchor boxes are an innovative aspect of how the Yolo9000 network makes detection predictions. One possible way a network could make bounding box predictions is by predicting the center x, center y, width, and height of each object. However, this approach is prone to overfitting and will take a very long time to train. The width and height of objects are very inconsistent and not conducive to consistent, accurate predictions for images the network has never seen before. Yolo gets around this by using *anchor boxes*. Each cell in the 13×13 grid has 5 anchors associated with it. These anchors are centered on the cell and are defined by a width and height relative to the size of the cell, each anchor having a different aspect ratio. For each image the network needs to learn the x and y offset from the center of the ground truth box to the center of the cell, as well as the width and height relative to the anchor box width and height. The exponential of the width and height ratios and sigmoid of x and y offsets are actually used as they are easier for the network to learn. Figure 3 shows an example of a ground truth bounding box offset from an anchor. When training, the anchor box with the lowest intersection over union (IOU) is associated with each ground truth bounding box. This technique of using anchor boxes keeps the bounding box predictions that the network needs to make much more consistent with each other and is more robust to a large variation in object sizes.

The exact number of anchors and width and height ratios that are used is determined by running k-means clustering on the dimensions of ground truth bounding boxes [5]. $k = 5$ was chosen in the original paper because it gives the best trade off between recall and complexity. In our implementation we use the 5 anchor boxes that give the lowest IOU on the VOC dataset.

2.3. Yolo loss function

The Yolo network uses a rather complex loss function. This function is comprised of multiple parts, designed to describe both the loss of the predicted bounding box(s) and class(es). The beauty of this loss function comes from the fact that only a single forward and backward pass of the network needs to be made in order for detection model to

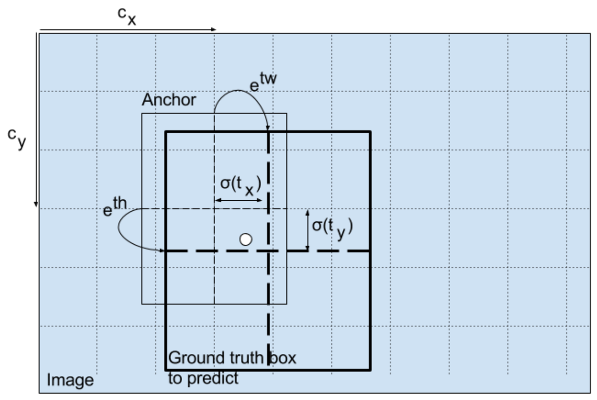

Figure 3. Yolo Anchor Boxes [2].

improve. This is in contrast to several other detection models which require multiple passes of the image.

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

$\mathbb{1}^{obj}$ indicates a mask value used to ignore objects that are not present within a given cell. i.e. if an object appears $\mathbb{1}^{obj}$ produces a 1. On the other hand if no object is present it produces a 0. In the above loss, $\sum_{i=0}^{S^2}$ is a sum over the 13×13 grid cells and $\sum_{i=0}^{B}$ is a sum over all 5 anchors.

Bounding Box Coordinate Loss

The two top most equations of the loss function describe the, amount of inaccuracy between the predicted bounding boxes x/y offsets and width/height ratios, when compared against that of the true bounding box.

Classification Loss

The next two equations in the loss penalise incorrect classification and superfluous classification respectively. This part pushes the network to the correct class for each object.

Confidence Score Loss

The final equation simply describes the difference between the predicted ”confidence score” and the true class probability.

Lambda Hyper-parameters

There are two lambda values, coord and noobj. These values are used adjust the relative weight of the loss. increasing λ_{coord} will increase the bounding box's contribution to the overall model loss. Where as λ_{noobj} controls the contribution of ”false objects” to the loss. As with many deep learning models, Yolo's loss is the most complex and most important part of the model.

2.4. Training

One of the reasons for the success of the Yolo network is the fact that an existing network trained for classification can be modified to perform object detection. In the original implementation the authors trained a network on ImageNet for about a week before modifying it to perform object detection. Pretraining for classification allows the network to start with reasonable weights before attempting the more complicated task of bounding box predictions. A classification network is modified for detection by removing the last convolutional layer and adding more convolutional layers with a shape dependent on the number of object classes. Another innovative aspect of Yolo9000 is that they can train for detection or classification, and backpropagate differently depending on what the input image is. For example, an input image without any bounding boxes will only update with respect to the classification part of the loss.

2.5. Results

Yolo9000 is able to perform ”fast, accurate detection”. Figure 4 shows the original implementations mAP and frames per second on the VOC2007 dataset in comparison to several other state of the art object detection systems. The speed of Yolo9000 contributes a lot to its success. There is even progress being made to implement the Yolo network on embedded systems. Using a similar network configuration with fewer parameters, Fast Yolo was able to achieve 18FPS on an embedded system [6].

3. Contributions of this project

For this paper we implemented a copy of the Yolo9000 network. Said copy contains a partial implementation of the Yolo9000 network, loss function, and optimiser. This implementation adds to the body of knowledge, pertaining to the implementation of Yolo in the TensorFlow deep learning framework. Unlike the original paper which trained not only on VOC 2012, COCO, and ImageNet [5], we only trained our network on the VOC 2012 data set. This decision was made not only to reduce the outrages week long training times reported in the Yolo paper [3] but, also to reduce the chance of a mistake in the design of the network. As for the rest of the network architecture, our model was designed to be as close as possible to the original model put forward in the Yolo9000 paper. However, this duplication of the model did not go as planed, yielding mostly incorrect predictions due to not enough training time. Even with our subpar results, we have produced a reference implementation that will be helpful to future researchers and students pursuing the goal of running a Yolo9000 network using t the TensorFlow framework. Perhaps, with a little luck, they may succeed where so many others have failed.

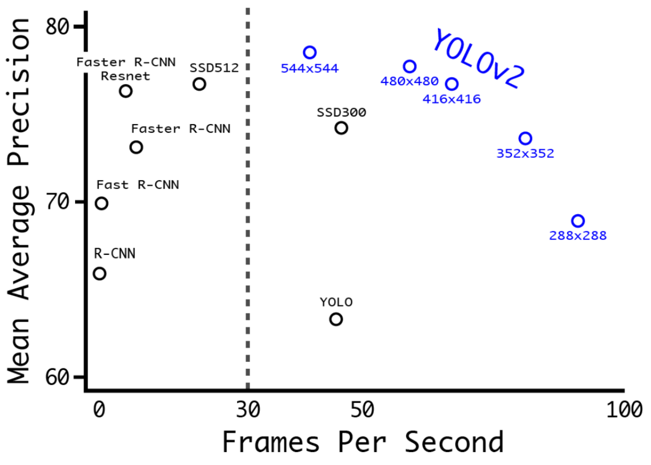

Figure 4. Yolo9000 Mean Average Precision and Frames per Second on VOC2007 [5]

3.1. Training

The computational resources available to us did not allow us to pretrain a classification convolutional network on ImageNet. Instead our network started from stracth. We trained using the VOC2012 data which consists of 5700 training images and objects belonging to 20 classes. We used a Nvidia Telsa with 12GB of VRAM as that was required for a batch size of 32 using batch normalization. Our preprocessing of the images was fairly simple and consisted of resizing the images to $416x416$ and performing random horizontal flipping of objects inside the bounding boxes. Finally, we used a batch size of 32, 45000 iterations, and a learning rate of 3e-5. Adam optimization was also used so the learning rate would slowly decrease with momentum. However, we ran into several issues during training which required us to restart the training process. We expected the training to take 48 hours before any reasonable results appeared, but with the restarts we were unable to accomplish this with a bug free network.

3.2. Results

Our preliminary results can be seen in Figure 5. This figure shows the first bounding box predictions our network made. This was accompanied by a large spike in loss, as seen in Figure 6. We were initially concerned with this spike, but upon closer inspection we realised it corresponded to the network finally making bounding box predictions. The loss spiked because the IOU of these predictions was finally being taken into account.

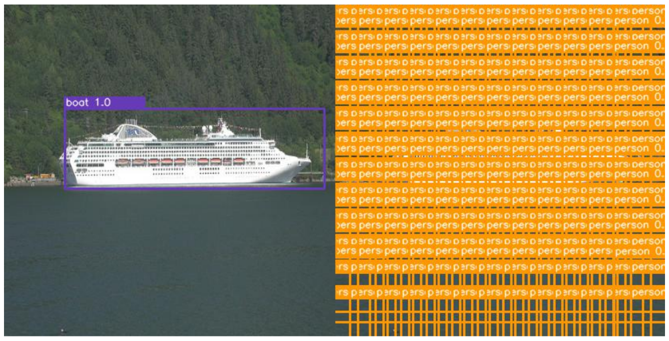

Figure 5. First bounding box predictions of our network during training. [Left] ground truth [Right] predictions

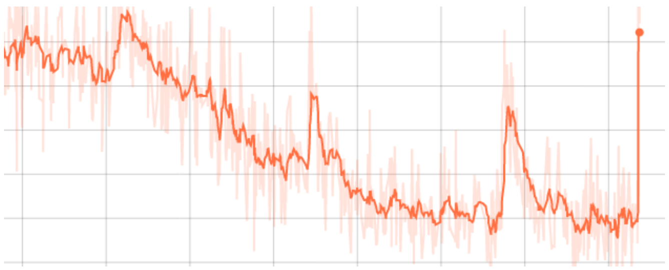
Figure 6. Loss of our network after 28k iterations

3.3. Issues

The most challenging part of implementing Yolo9000 is getting the training data into the correct format and implementing the loss function. The use of anchor boxes is ingenious, yet difficult to implement. The loss function expects the input data to be in a specific format in a specific shape. The bounding boxes imported from the VOC2012 dataset is in the [xmin, ymin, xmax, ymax] format, but needs to be converted to offsets relative to its closest grid cell and anchor box with the lowest IOU. The long training times made catching bugs with our implementation difficult.

One of the main issues we had was training time. Since training the network to make somewhat accurate predictions takes upwards of 48 hours, any modifications we made, big or small, would take hours to be noticeable. On a few occasions we found small bugs in our implementation when we were already multiple hours into training. The only option was to stop training and start again. Another issue we had was the computational resources required to train the network. Training the network with a single image in a batch was almost impossible and painfully slow on anything but a powerful GPU. This increased the time for the compile-run-test loop and prevented us from quickly testing and iterating on our code.

Another issue we ran into was the size of the Tensorflow summaries. While developing we thought it was a good idea to have the preprocessed images along with their bounding boxes appear in Tensorboard. Initially this let us know we were preprocessing the images correctly. However, after 35k iterations of training the Tensorflow summary file grew to an unmanageable size (20GB) and forced us to stop training. These issues were learned from and will be avoided in the future.

4. Discussion and future directions

Our implementation of Yolo9000 is currently erroneous and does not produce meaningful results, as can be seen in figure 5. That being said, with a full length training period, we believe that we will begin to see performance comparable to that reported by Redmon et al. With a full training period we should end up with a state of the art object detection system. However, that is not to say that the Yolo system is without weakness. The main benefit of Yolo is the speed at which it can evaluate an image. This speed, however, is favoured at the cost of accuracy. Other versions of Yolo, such as Tiny Yolo, are designed to be even faster with less accuracy. The various forms of Yolo are thus well-suited to applications which require rapid evaluation, such as when processing video streams. However, this solution is likely not the ideal option if your intended application favours accuracy over speed. Another potential deterrent from using the Yolo system is the relatively long required pre-training. To achieve the reported performance, Redmon et al. first pre-trained their network for classification on ImageNet for around a week. From this, a modified version of the pretrained network is trained for detection, with the input resolution being doubled[4]. This results in a very time consuming training process, particularly when compared to networks which do not require pretraining.

Overall, the Yolo object detection system provides accurate, general purpose object classification and detection at an extremely fast speed. Though not perfect for all situations, this system has the potential to be used in a plethora of applications.

References

[1] Tensorflow. `https://www.tensorflow.org/`. (Accessed on 04/06/2018).

[2] C. Bourez. Bounding box object detectors: understanding yolo, you look only once, 2017.

[3] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[4] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[5] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[6] M. J. Shafiee, B. Chywl, F. Li, and A. Wong. Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *CoRR*, abs/1709.05943, 2017.

[7] F. Shaikh. 10 advanced deep learning architectures data scientists should know!, 2018.