ECE 57000 Term Paper: YOLO Object Detection Comparison and Tuning

Anonymous Authors[1]

Abstract

YOLO object detection is a novel approach to computer vision object detection which applies a single neural network to the entire image. YOLO's approach enables extremely efficient processing, resulting in real-time object detection with high accuracy. In this paper, we will explore the third iteration of YOLO, YOLOv3, and implement it from scratch using a popular Python framework. We will then test our implementation on publicly available images as well as dashcam video to observe its effectiveness. Finally, we will compare our own implementation to a highly regarded existing implementation to further observe its strengths and shortcomings.

1. Review and Critique

Before jumping into YOLO, we will begin by reviewing and critiquing some relevant artificial intelligence papers. It will be shown by comparison that the YOLO paper is, in the opinion of the author, superior to the others, and most fit for further exploration.

1.1. Efficient Neural Architecture Search via Parameter Sharing (Pham et al., 2018)

Summary: This paper proposes "Efficient Neural Architecture Search" as a potentially breakthrough approach to automated artificial intelligence model design. While traditional "Neural Architecture Search" has been very successful, it is extremely computationally expensive to run. Therefore, its practicality is limited in some situations, such as embedded devices and low-power notebooks and other computers. ENAS uses new techniques to speed up NAS by 1000x with comparable rates of error, truly an exciting development in artificial intelligence.

Key Contributions: ENAS achieves this speedup by parameter sharing, a process whereby all candidate models are forced to share optimization parameters (or "weights") in order to avoid having to train each model from scratch. The ENAS paper shows that this sharing of parameters is very possible and can be highly performant when used properly. The paper goes into depth on how the ENAS algorithm searches for models, trains them all while using parameter sharing, and derives an optimal architecture. It also includes sufficient experimental results that back up its central points and further argue for the effectiveness of ENAS.

Review: Overall, this paper presents a very exciting new approach to neural architecture search. The paper transitions well from intuitive introduction to in-depth step-by-step breakdown of how ENAS does its work. Examples are used frequently to aid in understanding. However, the training details in the experiments section feel a bit glossed-over. Nevertheless, the results on common datasets speak for themselves. The 50,000x reduction in GPU hours when compared to NAS is very exciting, all while maintaining good testing error. A question that could be posed to the authors is what situations ENAS would be more suitable for, as they mention other similar networks and their benefits and drawbacks, but don't elaborate much on when / why ENAS should NOT be used.

1.2. You Only Look Once: Unified, Real-Time Object Detection (Redmon et al., 2016)

Summary: This paper proposes YOLO or "You Only Look Once" as a potentially breakthrough approach to object detection in computer vision (and is also the topic of this paper). With YOLO, object detection is treated as an end-to-end regression problem, and the result is an extremely fast, optimized object detection pipeline. In general, YOLO is also better at domain generalization than other object detection approaches, although it does exhibit slightly more localization errors.

Key Contributions: Typically, traditional classifiers are simply re-purposed for computer vision. YOLO re-frames object detection entirely by using regression to go from full images to bounding boxes and classification probabilities. The simplicity and speed YOLO brings are its main advantages, opening up a whole new avenue of applications for computer vision object detection technology. However, YOLO lags behind state-of-the-art systems in identifying some objects, especially smaller ones. The experiments included in the paper do a good job of showing these trade-offs.

Review: Overall, this paper's slightly less-formal style and

simpler language makes it a much easier read than other papers. The paper is very clear about the advantages and drawbacks of YOLO, and even proposes a combination of YOLO with other techniques to mitigate its drawbacks. The examples and visuals given give the reader a clear idea of all aspects of YOLO's performance, both good and bad. One potentially negative aspect of the paper is the fact that because YOLO is relatively simple, more time is spent comparing YOLO to other classifiers than detailing YOLO's operation. One question that could be posed to the authors is if there is any possible algorithm combination that could preserve YOLO's speed while improving accuracy, as there is one combination given that improves accuracy but not at YOLO's typical speeds, which is its primary advantage as the only real-time object detection classifier in the top 10 VOC 2012 leaderboards. YOLO is clearly a great option for performance-sensitive contexts, such as in low-power computing systems or fast-paced applications.

1.3. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer (Chen et al., 2019)

Summary: This paper proposes an Interpolation-based Differentiable Renderer as a potential breakthrough technique in computer vision and machine learning. In this paper, the idea of recognizing objects from simple 2D images is questioned, and the important point that images are 2D projections formed by interactions between 3D objects and light is used to inspire a novel new approach to image rendering and object detection that accounts for this. The authors show that their approach is highly accurate over a variety of conditions.

Key Contributions: The new approach to differentiable rendering, dubbed "DIB-R", views rasterization (rendering of 3D to 2D image) in a way that allows it to compute gradients "analytically over the entire image". Foreground and background pixels are treated differently to enable superior capturing of shape and occlusion information. DIB-R is then wrapped around a simple neural network to predicts the properties of "an initial polygon sphere" given some input. Numerical and visual results are found to be "state-of-the-art".

Review: Overall, this paper describes its proposals in great detail but doesn't do a great job of gradually and intuitively building up to those details. Non-experts on the subject matter may get quite confused at times when reading. The results of their differentiable renderer are also somewhat glossed-over, although applications and experiments are explained in-depth, which is a bright spot. Their two applications could have probably been split into two different papers in order to allow for more space for explanation for both. One question that could be posed to the authors is what real-world applications the DIB-R differentiable renderer might have. This would also help readers better understand what contexts the renderer would perform well and not well in.

1.4. Image-to-Image Translation with Conditional Adversarial Networks (Isola et al., 2017)

Summary: This paper investigates adversarial networks as a potentially viable tool for solving image-to-image translation problems. It is demonstrated that teaching the network a loss function to train the mappings between input to output images allows for effective synthesis of completely transformed images from simple input images. Mapping functions are no longer hand-engineered, and this paper seeks to show that loss functions don't have to be either to achieve good results.

Key Contributions: This paper seeks to develop a common framework to solve all kinds image-to-image translation problems. It explores the use of Generative Adversarial Networks (GANs), which learns a loss function that adapts to input data, in image-to-image translation, avoiding the manual optimization of loss functions that usually requires expert knowledge. Conditional GANs are integrated into the neural network architecture in order to enable this, and experiments show promising results.

Review: Overall, this paper takes the relatively familiar concept of Generative Adversarial Networks (GANs) for image-to-image translation, and explores them in a conditional setting. That is, conditioning on an input image instead of a random noise vector and generating a corresponding output image). This helps in making the paper more understandable, and the methods more generally applicable. The methods are well-explained and the experiments appear sound. The paper also shows how cGAN can be combined with other methods to improve performance. Positive and negative aspects of results are well-analyzed and discussed. The section showing off user-submitted applications of their methods is especially interesting to read. One question that could be posed to the authors is what specific situations CNNs might produce better results than cGANs, but overall it appears that cGANs are applicable in a very wide variety of contexts.

1.5. Feature Pyramid Networks for Object Detection (Lin et al., 2017)

Summary: This paper aims to show that by taking advantage of the inherent pyramidal structure of CNNs, feature pyramids can be constructed with very little additional cost as compared to traditional feature pyramid computation. Their proposed "Feature Pyramid Network" or FPN performs much better than traditional feature extractors, and when integrated into a complete detection system, displays

state-of-the-art detection while running in real-time on a single GPU.

Key Contributions: This paper shows that it is highly viable to "naturally leverage" the pyramidal shape of existing feature hierarchies to create a strong feature pyramid. These "in-network" feature pyramids can replaced traditional featurized image pyramids, resulting in big improvements in accuracy without sacrificing on cost and performance. The proposed network is laid out quite intuitively and experiments show very strong results.

Review: Overall, this paper makes a fairly intuitive connection between the natural pyramidal structure of deep convolutional networks and featurized image pyramids, creating a "Feature Pyramid Network" that enjoys benefits of featurized image pyramids without the slowdowns involved. This is a great idea in theory, and it is demonstrated to be fairly practical and performant in a variety of applications through their experiments. Users will especially appreciate the comparisons to other high-performance image pyramid techniques. The paper also does a good job of explaining the importance of different aspects of their proposed network structure. Diagrams of its structure are also well-appreciated. One question that could be posed to the authors is if there is some combination of FPNs and traditional feature pyramids that would give a "best of both worlds" result, however it appears that overall, FPNs are suitable and effective for a wide range of applications as it stands.

2. The Experiment

We will now move on to discuss the experiment. The experiment will be described in three parts: implementation, evaluation, and discussion.

2.1. Implementation

The original plan for this experiment involved implementing multiple versions of the YOLO network, and comparing their performance. Over the course of the experiment, the focus was shifted to instead consist of implementing YOLOv3 from scratch in PyTorch and then comparing it to the YOLOv3 implementation available in the widely-known "Darknet" framework (Redmon, 2021).

The YOLOv3 python implementation we created is based on an online YOLO guide (Kathuria, 2018). This guide walked through the process of incrementally building up the YOLOv3 network, forward pass, prediction processing, and input and output pipelines. Ultimately, we were able to re-implement the YOLOv3 network using PyTorch, bug fixing and testing as we went. The final implementation consists primarily of a network definition, detection code, and some helper functions.

The basic operation of the system will now be outlined. To start, a configuration file defines all of the pertinent network details, including input dimensions and channels, batch size, and details for each layer, defined sequentially for the entire network. The network implementation takes this configuration as input and generates a neural network model to match it, entirely in PyTorch. Finally, the detection code loads pre-trained weights into the network, conforms images according to the network's input dimensions, passes images through the network, and then interprets and processes its output to generate copies of the inputted images with bounding boxes and classifications overlayed onto the detected objects.

A separate set of detection code was also re-implemented to input and process videos, outputting in real-time the inputted frames with bounding boxes and object classifications overlayed on top of them.

Figure 1 below is a visual explanation of the structure of the image-based implementation.

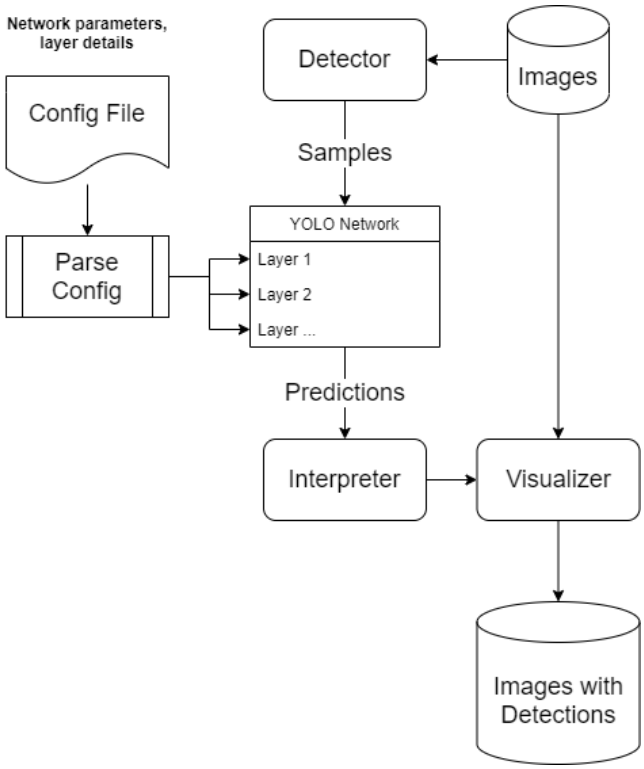

Figure 1. Overview of custom PyTorch implementation structure

For the "Darknet" portion of our implementation, we downloaded, installed, and evaluated the popular "Darknet" framework (Redmon, 2021) by testing it on the same images as our own PyTorch implementation.

2.2. Evaluation

The two implementations were then evaluated on several samples images, including those canonically used to test object detection as well as some randomly selected images from the internet. Additionally, the video detection version of the PyTorch implementation was tested on the author's own dash cam footage from driving around a local college campus.

To clarify further, the same network configuration files and pre-trained weights for both implementations. The differences seen are purely a result of the differences in the network and detection code.

To evaluate overall performance, observations were made about bounding box size, scaling, and accuracy, as well as classification accuracy and coverage. Some example comparisons between the custom PyTorch implementation and "Darknet" are shown in figures 2 and 3.

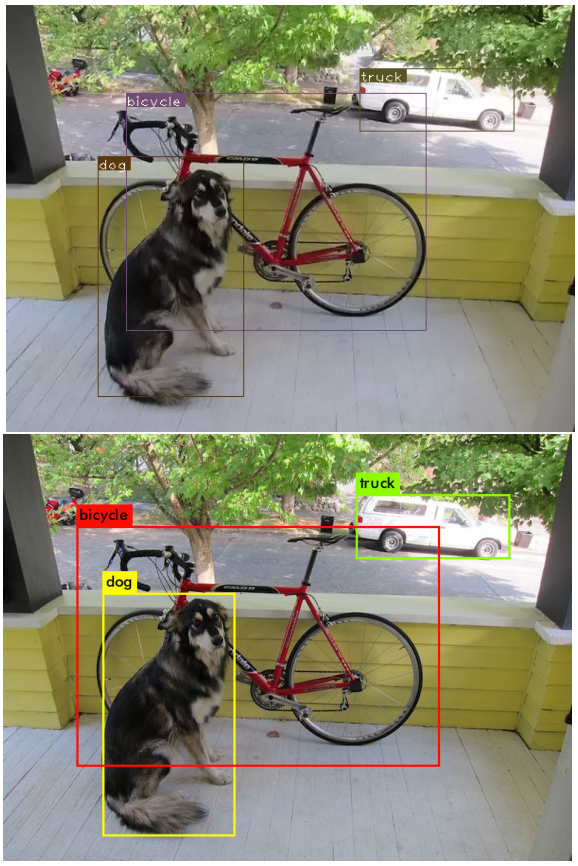

Figure 2. "Dog Cycle Car" object detection comparison between custom PyTorch (top) and Darknet (bottom) implementations

Early on in the PyTorch implementation development, bounding boxes were often misplaced or improperly sized. After some bug fixing in the helper functions, they were properly scaled and transformed to fit the outputted image

Figure 3. "Donut" object detection comparison between custom PyTorch (top) and Darknet (bottom) implementations

dimensions, and overall the PyTorch implementation exhibits strong bounding box performance.

2.3. Discussion

Overall, our PyTorch implementation performs strongly when compared to the well-known and well-accepted "Darknet" implementation. Bounding boxes and object detection are comparable, and in some cases, our PyTorch implementation arguably outperforms its counterpart.

In the "Dog Cycle Car" comparison in figure 2, we can see that both implementations correctly identify the dog, bicycle, and truck in the image, with appropriately sized, scaled, and placed bounding boxes. The "Darknet" implementation appears to place a bit more padding around the bicycle and dog objects, while the PyTorch implementation is draws the boxes a little tighter around this objects in this case. This may be due to the exact tuning and scaling of the bounding boxes in the Darknet implementation being a bit more generous, although this isn't the case in every situation.

In some cases, our PyTorch implementation can outperform the "Darknet" implementation. This can be easily observed

in the ”Donut” comparison in figure 3. In this case, we can see that ”Darknet” fails to detect and draw bounding boxes around many of the donuts that our PyTorch implementation successfully identified. This is likely due to the overall confidence threshold in ”Darknet” being slightly higher than our PyTorch implementation. By default, ”Darknet” prefers to be more confident about its object detections, leading to less detections of higher average confidence than our PyTorch implementation. We can observe in this scenario that the potentially lower confidence threshold in our PyTorch implementation is advantageous, as more donuts are detected.

In other ways, the two implementations have similar flaws. Again in the ”Donut” comparison in figure 3, one or both of the two green donuts identified is mislabeled as an ”apple”, likely due to its green color and round shape. Some additional training on images like this might help teach the model to discern between green donuts and apples, but as-is with the pre-trained weights, this is not to be unexpected.

Through all of this, the performance and speed of the network was on full display. On average, each input image took about 0.05 seconds to completely predict when using a 6-core Intel i7 processor and NVIDIA GTX 1080 graphics card. This is due to the simplicity and straightforward approach of YOLO. Unlike previous object detectors, which were simply classifiers re-purposed for computer vision, YOLO is a truly end-to-end regression pipeline, detecting and classifying objects in a single pass (hence its name ”You Only Look Once”). With performance like this, real-time video detection is feasible, as will be explored in the next subsection.

2.4. Real-time Video Detection

The results for the video version of our PyTorch implementation were just as promising and exciting as those for its image counterpart. Figure 4 below shows a screenshot from a video detection run.

Though it may be difficult to see on the small image, the truck passing on the left has an appropriately sized bounding box and is properly labeled ”truck”. The bus further ahead is also identified as a ”truck”, likely due to the fact that it is farther away. The traffic light is also properly identified, as well as two more cars in the distance to the right. During video playback, the bounding boxes jump around a bit but overall stay locked fairly consistently to their respective objects.

Given that there is no continuity from frame to frame (each frame is processed as a unique image from the others), bounding boxes sometimes ”pop in” and ”pop out”, and classifications can jitter and oscillate. However in general, the bounding boxes follow their objects well and classifications are relatively stable, especially for closer objects (which the network can evaluate in greater detail than farther objects).

Figure 4. Screenshot from video detection test (PyTorch implementation)

Below in figure 5 is a screenshot from a slightly different video detection run.

Figure 5. Screenshot from video detection test (PyTorch implementation)

Again, though it may be difficult to see, this frame displays the ability of our PyTorch YOLO implementation to recognize many objects, partially overlapping, in real time in a video frame. Numerous people are correctly identified and classified, and even smaller items such as a backpack are properly detected. Persons that are not identified are either partially obscured or sitting at such an angle that the classifier cannot recognize them. Regardless, this is a strong result for YOLO, especially considering that again, this detection is being done in real-time on a personal computer.

3. Conclusion

In conclusion, our PyTorch re-implementation, developed with the help of a detailed guide (Kathuria, 2018), in con-

juction with pre-trained weights, performs very well, even in comparison to the highly-developed C-based "Darknet" implementation. On a variety of images, it exhibits good detection and classification of objects, and its lower confidence threshold allows it to identify more objects, which can be both a blessing and a curse depending on the scenario. A lot was learned in the process of re-implementing YOLOv3 step-by-step. Since we were essentially implementing the network and detector from scratch, each part of the process was developed and debugged. This forced us to really strive to understand the lifecycle of an image from getting sampling from the dataset to going through the network and finally being outputted with bounding boxes overlayed.

The video demonstration also showed the true power of YOLO in its computation efficiency. With such efficiency, real-time detection becomes not just possible but powerful. Because of this, YOLO opens up a whole new world of practical applications. For example, a small surveillance camera could have object detection embedded within it to be able to detect when a person is at your door rather than just a squirrel, which would otherwise trigger more naive presence detection solutions.

Finally, the fact that the authors of the original YOLO paper have continued to iterate upon their original network speaks to the legs that this object detection approach has. For such a relatively simple system, YOLO's potential continues to grow, and its widespread experimentation and adoption has left a large imprint on the entire computer vision space.

References

Chen, W., Gao, J., Ling, H., Smith, E., Lehtinen, J., Jacobson, A., and Fidler, S. Learning to predict 3d objects with an interpolation-based differentiable renderer. *33rd Conference on Neural Information Processing Systems*, pp. 1–11, 2019.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–10, 2017.

Kathuria, A. Yolov3 tutorial from scratch, 2018. URL https://github.com/ayooshkathuria/YOLO_v3_tutorial_from_scratch.

Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2017.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameter sharing. pp. 1–11, 2018.

Redmon, J. Darknet: Convolutional neural networks, 2021. URL https://github.com/pjreddie/darknet.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–10, 2016. URL https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf.