Ensembling Large-Scale Object Detectors

Kunj Mehta
Rutgers University
kcm161@scarletmail.rutgers.edu

Abstract

Large-scale object detection models like YOLO, SSD and R-CNN have become the state-of-the-art in terms of results and the go-to solution for their fast and real-time inference times. These models owe their impressive performance in detection to their depth, among other things. Taking inspiration from the seminal classical Viola-Jones face detector that employed cascading to achieve the same objectives of high performance and speed, I explore the effect of ensembling YOLO on average precision for the problem of generic logo detection. I setup bagging and boosting pipelines and report results on the benchmark Flickr-32 dataset. I observe that both bagging and boosting improve detection confidences and average precision with boosting leading to a marked increase of 2% in AP as calculated by one measure.

1. Related Work

Object detection is one of the fundamental and mature use cases in machine learning and computer vision. With the rise of neural networks and deep learning, the prowess of object detection systems has also increased. This has broadly led to two types of detection frameworks developing: one-stage and two-stage. In the one-stage detector framework like YOLO [6] and SSD, the detection is output after only a single pass over the image. In the two-stage framework like R-CNN, the model first generates region proposals and then the detections. Nonetheless, both of these frameworks have seen various versions that keep improving on detection performance and inference speed.
Logo detection and classification as a subset of object detection and can be categorized into two: open set and closed set. Open set detection is when the dataset contains logos captured naturally in the real-world while closed set detection datasets have images which contain logos as the only thing in the image. [4] Closed set detection is traditionally easier to do because of less variability in the data. For open set detection, two major problems arise: the size of the logo RoIs in the image which can be very small in comparison to image size, and the issue of logos being different for the same brand across different points in time.
Typically, for open set logo classification, the best performing systems breakdown the problem into two steps; in the first, a generic logo detector detects *where* the logos are while in the second, a classifier is trained to output *what* the class (name) of the logo is. [2]
Logo detection, and object detection as well has benefited from models like YOLO. However, before all the deep learning, one of the traditional detectors used cascading. Viola-Jones face detector [8] successfully detected faces using cascading classifiers. Specifically, Viola-Jones employed different features that could recognize faces in a grid like fashion on each image, where if one feature could not find a face in a cell in the grid, that grid will be rejected and never seen again. This proved the effectiveness of ensembling in detecting objects, especially in low-compute environments.
Use of cascading resurfaced again succesfully in Cascade-RCNN where the detector is cascaded by increasing IoU thresholds under the assumption that the model will be able to incrementally learn the data patterns and detect positives and negatives across all IoUs. [3] This also solves overfitting problems that occur when individual detectors are trained for higher IoU thresholds.
Taking inspiration from cascaded detectors such as Viola-Jones and Cascade-RCNN, I decide to implement ensembling pipelines for bagging and boosting on other large scale detectors; specifically YOLOv5 [5] due to its accessibility. As for the benchmark dataset, many of the open set logo datasets like QMULOpenLogo and LogosintheWild are inaccessible due to being taken down or being behind firewalls. Additionally, there is no universally recognized benchmark for open set logo detection. Due to this, I simulate the two-stage approach and build a generic logo detector on the closed set Flickr-32 dataset. [7]
Section 2 details the architecture and modifications to the codebase that I implemented in order to get the ensembling system to work. Section 3 focuses on the results while Section 4 is a discussion on the same.

2. Design and Implementation

2.1. Dataset

The dataset used is Flickr-32 which contains 8,240 images of background and 32 classes of logos partitioned into train, val and test sets. As recommended by the publishers of the dataset, the actual training set is constructed by combining the train and val sets. [7] As a result, the size of the training set in use becomes 4,280 images out of which 3,000 are background. The test set has 960 annotated images and 3,000 background images. Following the open set detection approach, these bounding box annotations are changed to represent the presence or absence of logos in the images and bounding boxes. That is, instead of representing the brand of the logo as a class, *we now only have two classes: 1 for logo and 0 for background.* In addition, I change the annotation format of the dataset to be compatible with the YOLO format.

2.2. Ensembling

In Viola-Jones, cascading was used as an ensembling technique. Cascading is using all information collected from the output from a given instance as additional information for the next instance in the cascade (ensemble). I build bagging and boosting pipelines. Bagging is a homogeneous weak learners' model where all instances learn from each other independently in parallel and inference combines them for determining the model average. On the other hand, boosting is where learners learn sequentially and adaptively to improve model predictions.
YOLOv5 is a packaged framework that takes care of everything from training, validation, testing, inference and evaluation using just the command line. If we want to train and deploy a single model, this framework is very easy to use. However, to setup a bagging and boosting pipeline, I need to separate the training and, inference and evaluation phases since I will be using multiple trained models for inference and evaluation. This is explained below.
After preparing the dataset, for bagging, I implement a bootstrap function that keeps generating a dataset of samples from the actual Flickr-32 dataset distribution. Each of these bootstrap samples are fed to a new instance of the ImageNet pre-trained YOLOv5-s model that have different initialization values due to a change in the seed value.
For boosting, I train the first YOLO model in the ensemble on the whole dataset and keep a track of the detections on the training set itself that it got wrong. These training examples are then given a higher weight when passed to the next model in the pipeline. Note here that for boosting, all the models are initalized with the same seed. For the purposes of this system, this is how I define wrong detections:

- False negatives (images that contain an object, but are not detected)
- False positives (images that do not contain an object but have a detection made my the model)
- In the case of multiple detections in a single image, if there are some detections that are either false positives or false negatives
- In the case where a multi-class detector (not a generic one) is to be built, if more than half of the detected classes are incorrect

The *training* bagging and boosting pipelines then work as follows (Figure 1):

- *Bagging*: Different instances of the same model are initialized with different random states and trained on different bootstraps of the same dataset. The different instances can be trained concurrently or sequentially
- *Boosting*: All the training examples that are "incorrect" detections for the previous instance of the model are given a higher sample weight in the training of the next instance. This higher sample weight is the inverse of the confidence value of the previous model during inference.

The *inference* bagging and boosting pipelines work as follows (Figure 1):

- *Bagging*: Perform inference for the image on all instances of the model in the bagging ensemble and perform Non-Maximum Suppression with a predetermined IoU threshold on the detections output from all the instances. The final detections are the ones that are above a pre-specified confidence threshold.
- *Boosting*: Assign weights to each model based on its detection performance (inverse of Average Precision) and perform inference for the image on all instances of the model in the boosting ensemble. Post-processing is done on these detections to obtain overlapping detections from different model instances. The overlap is determined again based on a pre-specified IoU threshold. Finally, the weighted confidence score of these overlapping detections is calculated and the detection is output if that weighted confidence score is above a confidence threshold.

In both of these pipelines, I train model instances using Adam optimizer with early stopping using the YOLOv5 implementation functionality that offers saving the weights of the best model so far. [5] The decision on how many model instances are required to be trained for the ensemble is decided on the fraction of examples out of the total that the current model got wrong. For the Flickr-32 dataset, I set

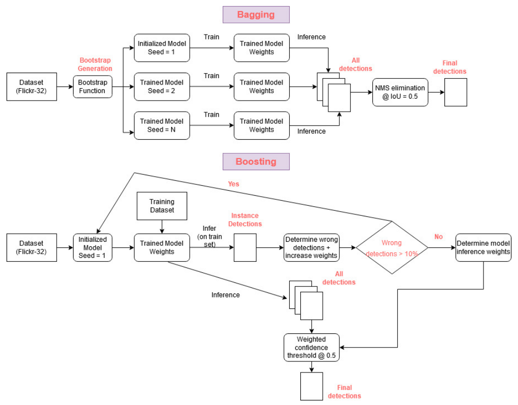

Figure 1. Architecture of bagging and boosting pipeline

it to at least 10% and that threshold is reached on the second instance. So, for this dataset, the ensemble is only two instances large. For larger datasets, I assume we will need more instances.

3. Results

3.1. Quantitative Evaluation

The state-of-the-art Average Precision @ 0.5 IoU as per PapersWithCode for generic logo detection on the Flickr-32 dataset is 87.1. Now, the Average Precision calculation used by the YOLOv5 framework is different than the usual calculation. The YOLOv5 framework reports results for a confidence threshold of 0.25 [5] while I use a confidence threshold of 0.5. I report both in Table 1. We can see that for both the ways of metric calculation, the boosting ensemble slightly outperforms the bagging ensemble, as it should.

Method	Model	AP@0.5
YOLOv5 [5]	Bagging Ensemble	0.853
YOLOv5	Boosting Ensemble	0.875
YOLOv5	One Boostrapped Instance	0.837
YOLOv5	One Instance	0.86
Normal [1]	Bagging Ensemble	0.7916
Normal	Boosting Ensemble	0.7947
Normal	One Boostrapped Instance	0.7107
Normal	One Instance	0.7918

Table 1. YOLOv5 ensembling results

3.1.1 Ablation Studies

I also perform small ablation studies to further interpret and analyze the results. Apart from performing inference and calculating the Average Precision for the ensembles, I also do the same for a model instance that is trained on a randomly generated bootstrap, that is one instance of the bagging ensemble ("One Bootstrapped Instance" in 1). We can see that this has a worse performance than the ensembles, and rightly so because it only has access to part of the data which is in all probability, also biased.
Additionally, I also use just the one instance of YOLOv5-s that I train and test on the whole dataset ("One Instance" in 1). We see that the boosting ensemble outperform a single model while the bagging ensemble has a similar performance to a single instance.

3.1.2 Qualitative Evaluation

In the report, I also include a detection sample from the test set of the Flickr-32 dataset. The same image is used for inference for all three cases: a single instance of YOLOv5-s, the bagging pipeline and the boosting pipeline. See Figures 2, 3, 4.

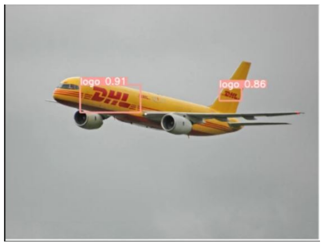

Figure 2. Single instance detection example

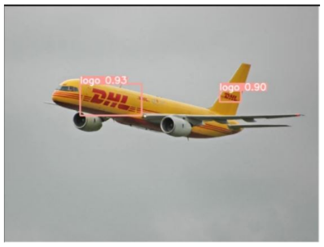

Figure 3. Bagging pipeline detection example

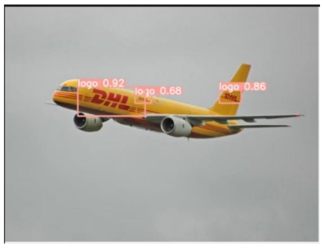

Figure 4. Boosting pipeline detection example

We can see that there are *three* DHL logos in the image. The single YOLO model is able to detect two of them with good confidence. The bagging ensemble improves on the confidence scores: 0.93 and 0.90 instead of 0.91 and 0.86 but can still only detect two logos. The best detection is made by the boosting ensemble which is able to detect all three logos.

3.1.3 Discussion

Through the numbers and the detection sample, we can clearly see that both the pipelines work and show improvement over a single instance. Boosting works better than bagging, as it should. However, I theorize that the improvement we see for this dataset is less in absolute numbers than what is possible if we use larger datasets. I believe that due to the depth of the base model, a single instance itself is powerful enough to be able to learn to detect objects in a dataset which is smaller. This does not leave much more performance to be eked out by ensembling for such small datasets.
As such, for future work, I would like to test this system on larger datasets, which would also have the additional complexity of not having just two instances of the base model. I would also like to extend this pipeline to that of a multi-class detector, instead of the generic detector it is now.

4. Conclusion

In this project, I designed two ensembling pipelines viz. bagging and boosting for large-scale object detectors like YOLO and SSD. Taking inspiration from cascading detectors, I used the very accessible YOLOv5 codebase to test the generic detection ensemble pipelines on the benchmark logo detection dataset Flickr-32 and observed successfully that both the pipelines achieve greater confidence in detections and / or average precision than when using just an individual model. I believe this opens up an avenue into further research on how ensembling can improve performance on much larger datasets

References

[1] Timothy C. Arlen. 3

[2] Muhammet Bastan, Hao-Yu Wu, Tian Cao, Bhargava Kota, and Mehmet Tek. Large scale open-set deep logo detection. *CoRR*, abs/1911.07440, 2019. 1

[3] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: delving into high quality object detection. *CoRR*, abs/1712.00726, 2017. 1

[4] Istvan Fehervari and Srikar Appalaraju. Scalable logo recognition using proxies. *CoRR*, abs/1811.08009, 2018. 1

[5] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, (Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, Nov. 2022. 1, 2, 3

[6] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 1

[7] Stefan Romberg, Lluis Garcia Pueyo, Rainer Lienhart, and Roelof van Zwol. Scalable logo recognition in real-world images. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, ICMR '11, New York, NY, USA, 2011. Association for Computing Machinery. 1, 2

[8] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001. 1