Object Detection Project: Compare STOA models to traditional algorithms in terms of mAP

DDA4220 Final Project

Xiangyi Li (119010153)
School of Data Science
Chinese University of Hong Kong, Shenzhen
`xiangyili@cuhk.edu.cn`

Yufei Ma (119010233)
School of Data Science
Chinese University of Hong Kong, Shenzhen
`yufeima@cuhk.edu.cn`

Abstract

In this project, we investigated the development of object detection algorithms in computer vision and in the context of the VOC2012 object detection competition, comparing methods before and after the R-CNN deep neural network proposal. We also tried reproducing STOA pre-trained models' results using PyTorch and then fine-tuning the models using the VOC2012's training data. Ultimately, we will compare the results between the traditional algorithms and STOA models.

1 Workload

- Xiangyi Li (50%): Responsible for implementing Faster R-CNN in PyTorch, the code framework used in other models. Responsible for setting up fine-tuning. Responsible for reading some of the paper writing.
- Yufei Ma (50%): Implemented YOLO and SSD models in PyTorch. Responsible for setting up GPUs. Responsible for writing the majority of the report. Responsible for running SSD models from `ssd.pytorch`

2 Introduction

Object detection is one of the most fundamental and well-established tasks in the computer vision area. It has been effectively solved once convolutional neural networks are re-introduced for image classification [1]. Early works like the Viola-Jones algorithm [2] and Histogram of Oriented Gradients feature descriptors [3] that mainly used handcrafted features have inspired methods like the Hybrid Coding for Selective Search [4] that had a relatively good performance on the VOC2012 object detection competition. According to [5], after AlexNet in 2012, deep neural networks have advanced the object detection task by a large margin in accuracy and speed.

In this project, we first examined previous works introduced and approached the image understanding problem. Then, we looked into how object detection originated from the task of face detection before deep neural networks became the standard approach to dealing with computer vision tasks. Then for the VOC2012 object detection competition, we first examined how traditional object detection algorithms have performed under this task; then, we researched and implemented STOA models and algorithms, including Faster R-CNN, YOLO, and SSD. Ultimately, we will compare the performance among the traditional methods, the pre-trained models implemented in PyTorch or HuggingFace, and the fine-tuned models in terms of mAP, known as mean Accuracy Precision.

3 Related Work

Early object detection methods could date back to 2001 [2], where the Viola-Jones algorithm was first applied to face recognition, which can be considered the predecessor to the object detection task.

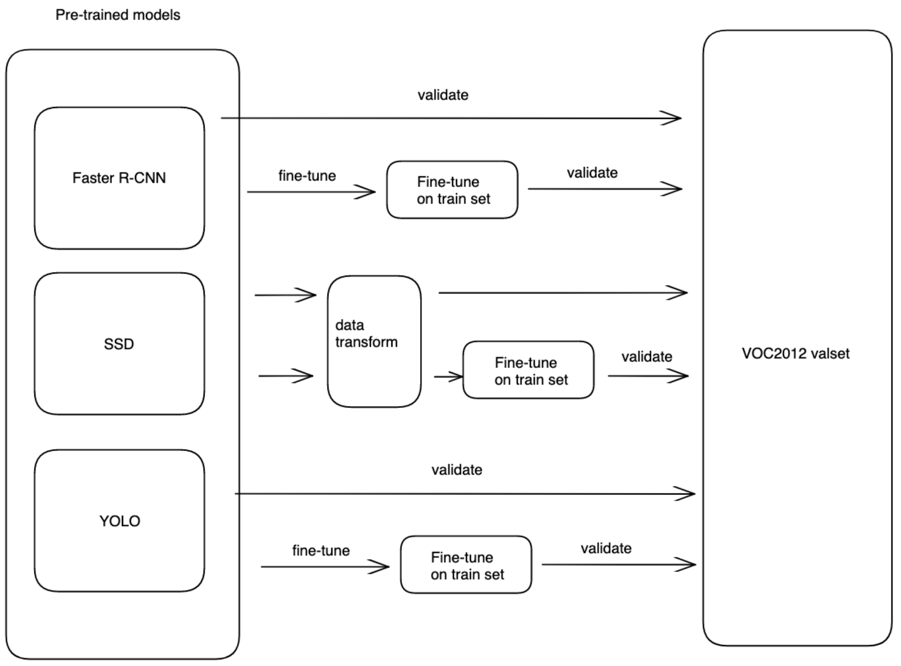

Figure 1: Designed experiment workflow for three STOA models.

It completes face recognition through a cascade of classifiers. Since its detection of side face images is unstable, different feature points representations, such as SIFT and HOG, have been explored in subsequent studies; these representations are robust to geometric and optical variations [3, 6].

Though traditional methods have seen advancements in detecting features, deep learning algorithms have gradually replaced traditional algorithms for manually labeling features with self-learning features. The regional CNN (R-CNN) proposed by [7] is the earliest two-stage algorithm that uses selective search methods to generate candidate regional networks and uses CNNs to extract features fed into classifiers. In 2014, SPPNet was proposed that uses different sizes of convolution kernels [8]. Fast R-CNN replaced the last layer with the pooling layer of interest [9]. Faster R-CNN proposed by [10] uses anchor boxes of different scales to extract features, reducing the generation time of candidate boxes.

Since a two-stage algorithm has high detection accuracy, numerous one-stage algorithms, such as YOLO and SSD [11, 12], have focused on making real-time object detection by boosting detection speed through techniques like the elimination of region proposal step and a single pass through the network. In other words, instead of first identifying areas in a picture where objects might exist and then looking closely at each of the identified areas in the two-stage algorithms, one-stage algorithms, for example, YOLO, splits the pictures into a grid where each cell in the grid predicts if it contains an object and what that object is. This is done in a single pass through the network, which makes it much faster.

According to a survey published in 2019, [13], other challenges, such as robustness to scales, orientations, occlusion, and clutter, are still met in the object detection task. Standard techniques that deal with this include data augmentation with generative adversarial networks, RNN, and Transformers have been introduced to facilitate the dealing with occlusion and clutter [14, 15].

4 Method

For the three models, we explored and investigated, we have implemented transforming, training loop, evaluation metric, and model fine-tuning from scratch in PyTorch and used Computer Vision toolkits like OpenMMLab and GluonCV. We also used standalone implementations for different models by forking their code to adjust our needs. We first implemented the Faster R-CNN from scratch in

PyTorch, got a score of the pre-trained model's performance on the VOC2012 validation image set, and then got the score of the fine-tuned model trained on the train image set. We referenced [7] to use results before R-CNN as our baseline, that is, mAP less than 33.3%. Also, according to [12], in our attempt to use GluonCV to implement SSD, we must perform data transformation for data augmentation before training.

5 Experiments

5.1 Evaluation method

We used mAP, mean average precision, to evaluate the models' performance in the object detection task. It is defined as the average of the precision values at different recall levels. Precision is the ratio of true positives to all predicted positives and recall is the ratio of true positives to all ground truth positives. According to [5], mAP was first introduced in the object detection task in 2007 by the PASCAL VOC challenge.

5.2 Experimental details

In planning to finish this project, we intended to go over the history of the object detection task. We tried to implement it from the first object detection algorithm to the most recent STOA models. The work before R-CNN was tough to replicate for two students with little computer vision knowledge, so instead, we went straight to deep learning methods, which eliminated the trouble of manual feature engineering for us.

Ultimately, we complete the VOC2012 competition with three STOA models [10, 12, 16], with different open-source pre-trained weights in different manners.

Faster R-CNN: We used PyTorch's built-in faster R-CNN pre-trained model and fine-tuned it on the training image set. The code file is `frcnn_first_try.ipynb`.

SSD with GluonCV: We explored the GluonCV toolkit and attempted first to run the validation on its built-in SSD model, but the val loop was taking too long. We surmised it was due to incompatible GPU or torch visions on instances on both the Colab and Vast.ai since it does not support PyTorch 2.0, and we had to downgrade many dependencies while encountering some cryptic error messages when doing it. Setting the batch size to be more than one also leads to failure. In the end, we chose not to pursue this approach. The code file is `SSD.ipynb`.

SSD with ssd.pytorch: This is a reimplementation of the original SSD model in 2016 developed by a group of developers as a side project. The implementation used a very early version of PyTorch, around 1.3.0, of which many functionalities and APIs have been deprecated, with many unresolved GitHub issues that led us to troubles when trying to run the model on Colab. Fortunately, there was productive discussion in the community, and we were able to fork the repository to incrementally modify the code so that it became compatible with PyTorch 2.0 in the end, which is also the version of PyTorch that Colab runs. We tested their pre-trained model on the VOC2012 for this approach and got a decent mAP score. Then we also tried to fine-tune a pre-trained model based on the COCO dataset for three epochs that got a not-so-good result.

YOLOv3 with GluonCV: similar scenario to SSD with GluonCV. YOLOv5 with Ultralytics: This was the most straightforward task since we have already figured out all the setbacks (see below) in this project and were able to smoothly train the YOLO model by setting up scripts on GitHub. For this model, we also got the most detailed analytics generated by scripts during training.

In the process, we also have had numerous failures for various reasons, which are listed below:

Google Colab: This was our go-to choice since we completed assignments on it, and it could access advanced computing power like Nvidia A100 GPUs. However, the relationship between Colab's virtual machines' disk spaces and Google Drive is very complicated, which eluded our knowledge before we spent countless hours on it. We were misguided by many previous tutorials, which often mount their execution space to Google Drive at the beginning of their file. This led to very encrypted bugs when performing intensive I/O operations due to the nature of this task where constant read & write from the annotation files and image files were necessary. It took us a few more setbacks on other cloud providers to realize that the error lies with using Google Drive.

Vast.ai: We rented bare-metal-like machines on this platform after seeing Colab produce too many error messages during training and validating loops. It turned out to be a worse choice for a few reasons. First, the VM instances are not as standardized as the Colab VM, leading to disk shortage and an incompatible CUDA version with the CV tools we were trying to install. Second, the virtual server's bandwidth is not stable, which caused significantly longer downloading datasets and pre-trained weights time. Finally, the `nohup` command does seem to work on the VM for unknown reasons. This led us to seek other cloud service providers before returning to Colab.

Vultr.com: The problem occurs even earlier. We managed to deposit around $70 in total to deploy a GPU compute instance but failed to. We contacted the support team, and no solutions have been working until this writing.

AWS, GCP, Azure: We encountered authentication with the storage buckets when instantiating EC2-g4dn instances on AWS. For GCP and Azure, we encountered problems with billing and could not successfully boot up any virtual machines.

In assignments, we were exposed to the OpenMMLab [17], which is a computer vision toolkit framework like GluenCV [18] that is integrated with datasets, evaluation methods, and pre-trained models; we tried to re-implement all three algorithms using MMDetection [17] so that we could enhance our understanding of the assignments. However, we encountered problems building mmdet on both the Colab and Vast.ai. However, after the experiments, we have found that directly reproducing standalone STOA models' implementations is much easier and more efficient than implementing them within a CV toolkit framework.

5.3 Results and analysis

Table 1: Object detection performance comparison

Method	mAP	Remark
Faster R-CNN	0.7412	PyTorch built-in pretrained model `fasterrcnn_resnet50_fpn_finetuned.pth`
Faster R-CNN fine-tuned	0.7253	Based on the above baseline model
SSD pretrained	0.7749	`github.com/amdegroot/ssd.pytorch`
SSD	N/A	Pre-trained model from GluonCV `ssd_512_resnet50_v1_voc`
YOLOv5	0.7010	Ultralytics official YOLOv5 implementation at `github.com/ultralytics/yolov5`
YOLOv5 with fine-tuning	0.7970	Ultralytics official YOLOv5 implementation at `github.com/ultralytics/yolov5`
YOLOv3 from GluonCV	N/A	Pre-trained model from GluonCV `yolo3_darknet53_voc`
MMDetection	N/A	Tried to use this to implement all three models. However, encountered numerous issues with dependencies and hardware on many cloud providers.

The result is shown in Table 1. Here is a summary of the results. In our experiment, a higher mAP indicates a better model performance.

Faster R-CNN: mAP of 0.7412 with PyTorch built-in pre-trained model and 0.7253 when fine-tuned on VOC2012 train image set.

SSD pretrained: mAP of 0.7749 using github.com/amdegroot/ssd.pytorch. We did not get results for the pre-trained model from GluonCV.

YOLOv5: mAP of 0.7010 with Ultralytics official YOLOv5 implementation and 0.7970 with fine-tuning. We did not get results for YOLOv3 from GluonCV.

Besides, all the models we explored achieved better results than the original. We found out the following from the test results:

Faster R-CNN: The PyTorch built-in pre-trained model achieved an mAP of 0.7412. However, after we fine-tuned the model on the VOC2012 training set, the mAP slightly decreased to 0.7253. This could suggest that the fine-tuning process might not have been appropriately optimized. For example, the number of epochs might not be enough, or the pre-trained model has already achieved the ceiling of Faster R-CNN on this dataset. Overfitting may also occur during fine-tuning, leading to decreased performance on the validation set.

SSD pre-trained: The SSD model, when trained and implemented using the code from github.com/amdegroot/ssd.pytorch, achieved an mAP of 0.7749, which is higher than both the pre-trained and fine-tuned Faster R-CNN models. This suggests the SSD model might be more effective for this particular task. Unfortunately, we could not get results for the pre-trained model from GluonCV. We also tried to train the SSD from scratch based on a COCO pre-trained model, but it might be that we only trained for three epochs. The mAP is only 0.2023.

YOLOv5: The Ultralytics official YOLOv5 implementation achieved an mAP of 0.7010. However, when fine-tuned, the mAP significantly increased to 0.7970. This is the highest mAP among all the models and configurations tested, suggesting that the fine-tuned YOLOv5 model was the most effective for this task. We did not get results for YOLOv3 from GluonCV, so a comparison could not be made. Also, note that we only fine-tuned the YOLOv5 for three epochs to achieve such performance. Because the COCO pre-trained SSD model only got 0.2023 on mAP after three epochs, it is obvious that selecting an appropriate pre-trained model is essential for object detection in terms of accuracy and training efficiency.

6 Conclusion

This project compared the performance of several implementations of STOA methods in the object detection task, including Faster R-CNN, SSD, and YOLOv5, with the mAP being the evaluation metric. The fine-tuned YOLOv5 model achieved the highest mean Average Precision (mAP) of 0.7970, outperforming the Faster R-CNN and SSD models. This suggests that YOLOv5 when properly fine-tuned, can be highly effective for object detection tasks. The main difficulty was dealing with the I/O system on Google Colab's virtual machine, namely unawareness of the unwieldiness of using Google Drive as the file system to perform object detection tasks.

In verifying that STOA methods in the object detection task outperform traditional methods and early detection models, we have learned how to evaluate the performance of a detection model, how to construct training and validating loops in PyTorch for the fine-tuning of pre-trained models, how to use CV toolkits like GluonCV to construct pipelines that could provide boilerplate codes and APIs that could simplify the workflow, and also how to run and modify standalone implementations of STOA models in PyTorch to train and predict, additionally, we also learned a great deal about the cloud computing platforms especially Colab, cloud file systems, and how to utilize the CUDA GPUs correctly and effectively (our initial training and validating of the Faster R-CNN model failed to utilize the GPU because we did not use the correct PyTorch API, the slowdown was immense). Learning the tools we are using is as important as learning object detection since they are equally transferable if we are to be working with other deep learning projects in the future.

There are many limitations to this project:

- We faced a significant constraint in terms of computational resources. We could not fully utilize GluonCV and MMDetection due to their strict dependency settings, which required more advanced computing devices than we had available. This limitation restricted our ability to experiment with these powerful tools and potentially impacted the performance of our models.
- We should have incorporated newer models like YOLOv7 into our project. This was because we were progressing from the earliest models. After encountering setbacks from YOLOv3 using the GluonCV toolkit, we were unsure about the stability and reproducibility of even newer models.
- Our project did not fully capture the breadth of novel ideas and approaches in computer vision, a highly active area of research. Many new methods and improvements have emerged since the publication of all three models we examine, such as Faster Faster R-CNN for Faster R-CNN, Improved SSD Network for SSD, and YOLOS (You Only Look One Sequence) for YOLO. Mathematically, the impact of different batch sizes and optimization algorithms could be huge, which we should have investigated better.

References

[1] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, and Brian Lee. A survey of modern deep learning based object detection models, 2021.

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

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[4] Koen van de Sande, Jasper Uijlings, Cees Snoek, and Arnold Smeulders. Hybrid coding for selective search. In *PASCAL VOC Workshop*. University of Amsterdam, 2012.

[5] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey, 2023.

[6] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Computer Vision – ECCV 2014*, pages 346–361. Springer International Publishing, 2014.

[9] Ross Girshick. Fast r-cnn, 2015.

[10] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[13] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey, 2019.

[14] Zitong Yu, Yuming Shen, Jingang Shi, Hengshuang Zhao, Yawen Cui, Jiehua Zhang, Philip Torr, and Guoying Zhao. Physformer++: Facial video-based physiological measurement with slowfast temporal difference transformer, 2023.

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[16] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[17] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Mmdetection: Open mmlab detection toolbox and benchmark, 2019.

[18] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, Aston Zhang, Hang Zhang, Zhi Zhang, Zhongyue Zhang, Shuai Zheng, and Yi Zhu. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing, 2020.