Lazier Detection?

Abstract

Modern imagery resolutions grow extremely fast for better quality. Inference on such large inputs in real-time is proved to be hardware-demanding. This project, rather than studying on common neural networks like CNN or transformer, is more like a competitive programming contest that emphasizes on time/space complexity. Exploring some pre-processing techniques, and worked on some implementation problems. We put great effort on implementing algorithms and benchmarked its speed against common computation libraries like numpy.

1 Introduction

The problem begins with the resolution of images. Through past 10 to 20 years, we see a non-stop trend of growing resolution sizes. From 640 x 480, 1024 x 768, 1920 x 1080, to the top-of-the-line 4K or even 8K standards. More pixels of course, means more details. Meanwhile, this also costs more resources to compute on them. There are some common pre-processing approaches to reduce the computation time needed.

The most common one might be simply down-scaling the image using different algorithms like Lanczos sampling [1] since it is straight-forward. However this is a lossy transformation that some information is inevitably lost. Suppose the region of interest (ROI) is relatively very small in a image. It would be unrecognizable even on human eyes after down-scaling. Some may also use a compressed representation like voronoi diagram[2], but the neural network for such representation of image often needed to be tailor-made to train on these images.

Cordonnier Et. al.[4] proposed a smaller-bigger network approach to "skip" some of the unrelated part of the input image by using a very shallow network close to Regional Proposal Network (RPN) of Faster R-CNN[3] to guess likeliness of a part of the image to intersecting with the ROI. Differentiable top-K process was used then to select the most possible crops of the image to inference in a bigger network for better accuracy. This approach is somewhat universal as it operate on any "smaller" or "bigger" network. We chose this as our baseline model to improve on its accuracy and extensively studied techniques on implementing highly efficient image-processing scripts.

2 Problem of the selection process

Cordonnier Et al., used differentiable top-K to select k number of most possible regions. Given the fact that this k and grid sizes they divided the image for scorer network and differentiable top-K, are all hyper-parameters.[1]

We observed the following when reproducing their experiments: Many highly overlapping regions are selected and each of them could not cover the entire object. Not only this would

[1]They mainly used k=10, 32x32 pixel patches for scorer network, and 50x50 pixels for selection module. We also use these settings in our experiments.

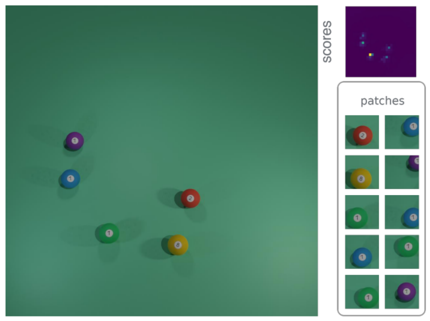

Figure 1: Demo from Cordonnier Et al., showing that green, blue purple have 2 3 overlapping selection boxes with. Some of them are cropped badly and only small part of the ball is being captured.

waste time on processing highly similar sub-images, we also suspected that this would lower the accuracy. Intuitively, Imagine a relatively big ground-truth object, and only a small part of it is captured by the bounding box.

We could try to explain this phenomenon by arguing on the k number and patches sizes. To begin with, k should be sufficiently bigger than the expected numbers of objects in the input image, but since this is a hyper-parameter, we should set a more aggressive value to avoid missing out some useful regions.

Meanwhile, Denote the following ground truth number of objects be n, and size of some arbitrary relevant objects be a matrix $A = \begin{bmatrix} — & a_1 & — \\ — & a_2 & — \\ & \vdots & \\ — & \vdots & — \\ — & a_n & — \end{bmatrix}$. For some $s_i \in S$, s_i is greater than the patch size. There would be a lot of overlapping-but-imperfect selection boxes produced and we are wasting time on computing highly similar badly bounded image input which would give similarly bad results. For instance, a 100 x 100 pixel ball against some 25 x 25 selection boxes. At minimum 4 of these boxes are needed to completely bound the object wanted.

We could formulate the minimum number of rectangular selection boxes N needed by:

$$N \geq \lceil \sum_{i=1}^{n} a_{i,x}/d_x \rceil \cdot \lceil \sum_{i=1}^{n} a_{i,y}/d_y \rceil$$

, where $a_{i,x}, a_{i,y}$ and d_x, d_y are horizontal size of the object (in case of being overlap with each other, count them as single object here) and selection boxes.

3 On alleviating the problem

We did some experiments trying to alleviate these issues on both speed and accuracy. Mainly by:

1. clustering the proposed ROIs

, and will be discussed in the following section. But before that, we would like to address on the previous finding in presentation. We have made several important implementation improvements which would greatly change our experimental results and conclusion since then. They would be discussed in the following sections.

3.1 Clustering the proposed ROIs

It is easy to understand the idea. As there are multiple regions selected on certain objects, then regrouping these boxes would give a better bounding box for that specific object. Our

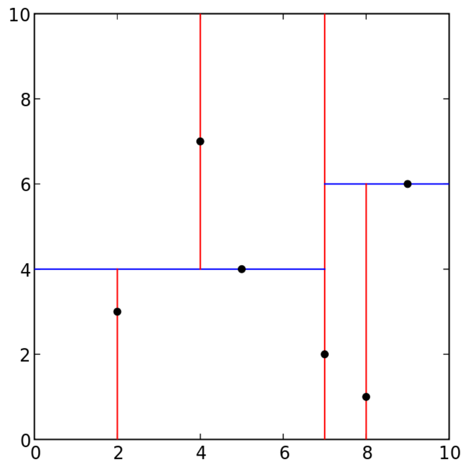

Figure 2: A k-d tree visualization for the point set (2,3), (5,4), (9,6), (4,7), (8,1), (7,2). From `https://en.wikipedia.org/wiki/K-d_tree` under CC BY-SA 3.0

goal is to make some one-to-one mapping between object and selection boxes. It is done by after regrouping the boxes, resultant areas is then rescaled to the hyper-parameter size of selection boxes.

One major challenge of clustering is that we could assume no fact on the ground truth grouping since we do not have sufficient information on the input, more specifically on the number of groups (a.k.a. number of objects!). So K-means is not a choice.

Density-based clustering like DBSCAN may also help but may be badly affected by the selection for hyper-parameters ϵ and $MinPts$. A possibly better set of $\{\epsilon, MinPts\}$ is chosen after primitive selection is done, by:

$$\epsilon = median(d(x,y)) \forall x \neq y \in S$$

The d above is the pairwise distance. We would have obtain a least one median-distanced pair, denote them as (z_1, z_2).... to obtain the average number of ϵ-neighbors of these pairs.

$$MinPts = \frac{1}{|Z|} \sum d(z_i, x) \quad \forall z_i \in Z, x \notin Z, d(z_i, x) \leq \epsilon$$

Besides that, we also experimented with Tree Preserving Embeddings (TPE), in which they uses hierarchical clustering as the information to embed into its clustering. This algorithm arouse our interest since it takes NO hyper-parameters on input and might be useful on tackling images without any additional information and might solve the problem of overcrowding in many clustering algorithms.

However, the original paper in ICML gives a $O(N^3)$ time algorithm, that they are basically finding n times pairwise distances of all nodes, where n is the number of nodes. Anything above $O(N^2)$ time is not practical in reality. In our first implementation using python (mainly scipy and numpy), clustering 1000 pseudo-random points in $\mathbb{R}^2$ takes around 40 seconds vs. DBSCAN for only 0.4 seconds. This is incredibly slow and unacceptable.

The major bottleneck in hierarchical clustering is finding pairwise distance between data points, which is in $O(N^2)$ time. We accelerated this process to near $O(N)$ time by the use of k-d tree. The explanation is as of follows: Given a array of location data of some points, the brute force method to find the distance matrix would be doing $O(N^2)$ scan. However, a k-d tree is basically a higher-dimension binary search tree, but in every node it branches out to all different axis. For instance, if a data point is in format (x, y), then a k-d tree would contain 2 interconnected search tree in both x and y directions. One could get its closest neighbor in $O(N)$ time and especially fast if manhattan distance is used as we only need to traverse each search tree of different axis once only. The major drawback of k-d tree is that the speed gain in query time diminishes very quickly with the number of axis, but is not a concern in our problem setting of x-y axis only.

On the implementation side, We are making a lot of custom data structures and loops by ourselves. Given the fact that cpython's slow interpreter and numpy/scipy have major part of their computing source codes written in C/C++. We also implement major part of this improved TPE using C++ (which is very tedious since we do not have handy libraries like numpy for matrix processing available in C++!). The compiled script is then used by making external calls in python main script, which is done by piping their I/O streams behind the scenes. This lowers the time complexity of TPE to be about $O(N^2)$.

4 Experimental Results

4.1 Accuracy

We use the billiard balls experiment in the original paper as our baseline. Whereas the "deeper" model after selection is ResNet18. The dataset is generated using Kubric, a google's in-house software to artificially generate datasets of billiard balls as our source of 20k billiard balls image. The result is as of follows:

Table 1: Billiard balls experiment w/ ResNet18 "deeper" network

Approach	Description	Test Acc. [%]
Baseline		81.6 ± 0.35
DBSCAN		77.8 ± 0.51
TPE	python (bruteforce)	(81.0 ± 0.76) (halted after 500 image due to time)
TPE	C++ (k-d tree)	85.8 ± 0.37

The results shows that TPE could effectively regroup selection boxes while DBSCAN is even worse than the baseline. This might be due to a over-aggressive hyper-parameters setting which may be broken upon cases that pairwise distances between selection boxes have a low variance, that DBSCAN regonizes them as a single ball. ResNet only outputs YES/NO answers to object existence against our regrouped centroid centered box.

Table 2: TPE speed benchmarked on 1k datapoints

Approach	Description	Running time (s)
Baseline $O(N^3)$	python(w/ numpy/scipy)	$\approx 40s$
C++ $O(N^3)$		$\approx 1.4s$
C++ $O(N^2)$	k-d tree	$\approx 0.38s$
DBSCAN	numpy	$\approx 0.4s$

Meanwhile, we managed to beat DBSCAN's in numpy in terms of running time by making better multi-threading and OS-level control (blocking, concurrency, etc). However, given that there are no handy computational libraries in C++, one should consider the cost for implementing using C++. Moreover, it is hard to beat a well-written python library that everyone uses. In our implementation, we spent a lot of time on grinding os-level controls and matrix computation from scratch, which may not worth the cost.

5 Discussion and future work

We improved the Cordonnier Et al.'s work by accuracy on the billiard ball experiment, and implemented a algorithm that could beat numpy implementation in similar time complexity (Not exactly same algorithm though) There are, of course more room for improvement. For example, we could try on more different experiments. In the original paper they also worked with traffic signs and birds dataset.

This approach is of course, not perfect. That hyper-parameters of k, size of patches, and size of selection boxes are still required. We initially are thinking on exploiting some convex polytopes[5] properties by mapping a voronoi diagram on it but this is left for later study due to time limit.

References

[1] Turkowski, K. (1990). Filters for common resampling tasks. Graphics Gems, 147–165. https://doi.org/10.1016/b978-0-08-050753-8.50042-5

[2] Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Computing Surveys, 23(3), 345–405. https://doi.org/10.1145/116873.116880

[3] Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137–1149. https://doi.org/10.1109/tpami.2016.2577031

[4] Cordonnier, J.-B., Mahendran, A., Dosovitskiy, A., Weissenborn, D., Uszkoreit, J., & Unterthiner, T. (2021). Differentiable patch selection for image recognition. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr46437.2021.00238

[5] Transactions on large-scale data- and knowledge-centered systems I. (2009). Lecture Notes in Computer Science, 357. https://doi.org/10.1007/978-3-642-03722-1