TIDE: A General Toolbox for Identifying Object Detection Errors

Daniel Bolya, Sean Foley, James Hays, and Judy Hoffman

Georgia Institute of Technology

Abstract. We introduce TIDE, a framework and associated toolbox[1] for analyzing the sources of error in object detection and instance segmentation algorithms. Importantly, our framework is applicable across datasets and can be applied directly to output prediction files without required knowledge of the underlying prediction system. Thus, our framework can be used as a drop-in replacement for the standard mAP computation while providing a comprehensive analysis of each model's strengths and weaknesses. We segment errors into six types and, crucially, are the first to introduce a technique for measuring the contribution of each error in a way that isolates its effect on overall performance. We show that such a representation is critical for drawing accurate, comprehensive conclusions through in-depth analysis across 4 datasets and 7 recognition models.

Keywords: Error Diagnosis, Object Detection, Instance Segmentation

1 Introduction

Object detection and instance segmentation are fundamental tasks in computer vision, with applications ranging from self-driving cars [6] to tumor detection [9]. Recently, the field of object detection has rapidly progressed, thanks in part to competition on challenging benchmarks, such as CalTech Pedestrians [8], Pascal [10], COCO [20], Cityscapes [6], and LVIS [12]. Typically, performance on these benchmarks is summarized by one number: mean Average Precision (mAP).

However, mAP suffers from several shortcomings, not the least of which is its complexity. It is defined as the area under the precision-recall curve for detections at a specific intersection-over-union (IoU) threshold with a correctly classified ground truth (GT), averaged over all classes. Starting with COCO [20], it became standard to average mAP over 10 IoU thresholds (interval of 0.05) to get a final $mAP^{0.5:0.95}$. The complexity of this metric poses a particular challenge when we wish to analyze errors in our detectors, as error types become intertwined, making it difficult to gauge how much each error type affects mAP.

Moreover, by optimizing for mAP alone, we may be inadvertently leaving out the relative importance of error types that can vary between applications. For instance, in tumor detection, correct classification arguably matters more than box localization; the existence of the tumor is essential, but the precise

[1] `https://dbolya.github.io/tide/`

Table 1: **Comparison to Other Toolkits.** We compare our desired features between existing toolkits and ours. ✔ indicates a toolkit has the feature, ✱ indicates that it partially does, and ✗ indicates that it doesn't.

Feature	Hoiem [14]	COCO [1]	UAP [4]	TIDE (Ours)
Compact Summary of Error Types	✱	✗	✔	✔
Isolates Error Contribution	✱	✗	✗	✔
Dataset Agnostic	✗	✗	✔	✔
Uses All Detections	✗	✔	✔	✔
Allows for deeper analysis	✔	✔	✔	✔

location may be manually corrected. In contrast, precise localization may be critical for robotic grasping where even slight mislocalizations can lead to faulty manipulation. Understanding how these sources of error relate to overall mAP is crucial to designing new models and choosing the proper model for a given task.

Thus we introduce TIDE, a general Toolkit for Identifying Detection and segmentation Errors, in order to address these concerns. We argue that a complete toolkit should: 1.) compactly summarize error types, so comparisons can be made at a glance; 2.) fully isolate the contribution of each error type, such that there are no confounding variables that can affect conclusions; 3.) not require dataset-specific annotations, to allow for comparisons across datasets; 4.) incorporate all the predictions of a model, since considering only a subset hides information; 5.) allow for finer analysis as desired, so that the sources of errors can be isolated.

Why we need a new analysis toolkit. Many works exist to analyze the errors in object detection and instance segmentation [15, 24, 7, 17, 22], but only a few provide a useful summary of all the errors in a model [14, 1, 4], and none have all the desirable properties listed above.

Hoiem et al. introduced the foundational work for summarizing errors in object detection [14], however their summary only explains false positives (with false negatives requiring separate analysis), and it depends on a hyperparameter N to control how many errors to consider, thus not fulfilling (4). Moreover, to use this summary effectively, this N needs to be swept over which creates 2d plots that are difficult to interpret (see error analysis in [11, 21]), and thus in practice only partially addresses (1). Their approach also doesn't fulfill (3) because their error types require manually defined superclasses which are not only subjective, but difficult to meaningfully define for datasets like LVIS [12] with over 1200 classes. Finally, it only partially fulfills (2) since the classification errors are defined such that if the detection is both mislocalized and misclassified it will be considered as misclassified, limiting the effectiveness of conclusions drawn from classification and localization error.

The COCO evaluation toolkit [1] attempts to update Hoiem et al.'s work by representing errors in terms of their effect on the precision-recall curve (thus tying them closer to mAP). This allows them to use all detections at once (4), since the precision recall curve implicitly weights each error based on its confidence.

However, the COCO toolkit generates 372 2d plots, each with 7 precision-recall curves, which requires a significant amount of time to digest and thus makes it difficult to compactly compare models (1). Yet, perhaps the most critical issue is that the COCO eval toolkit computes errors progressively which we show drastically misrepresents the contribution of each error (2), potentially leading to incorrect conclusions (see Sec. 2.3). Finally, the toolkit requires manual annotations that exist for COCO but not necessarily for other datasets (3).

As concurrent work, [4] attempts to find an upper bound for AP on these datasets and in the process addresses certain issues with the COCO toolkit. However, this work still bases their error reporting on the same progressive scheme that the COCO toolkit uses, which leads them to the dubious conclusion that background error is significantly more important all other types (see Fig. 2). As will be described in detail later, to draw reliable conclusions, it is essential that our toolkit work towards isolating the contribution of each error type (2).

Contributions In our work, we address all 5 goals and provide a compact, yet detailed *summary* of the errors in object detection and instance segmentation. Each error type can be represented as a single meaningful number (1), making it compact enough to fit in ablation tables (see Tab. 2), incorporates all detections (4), and doesn't require any extra annotations (3). We also weight our errors based on their effect on overall performance while carefully avoiding the confounding factors present in mAP (2). And while we prioritize ease of interpretation, our approach is modular enough that the same set of errors can be used for more fine-grained analysis (5). The end result is a compact, meaningful, and expressive set of errors that is applicable across models, datasets, and even tasks.

We demonstrate the value of our approach by comparing several recent CNN-based object detectors and instance segmenters across several datasets. We explain how to incorporate the summary into ablation studies to quantitatively justify design choices. We also provide an example of how to use the summary of errors to guide more fine-grained analysis in order to identify specific strengths or weaknesses of a model.

We hope that this toolkit can form the basis of analysis for future work, lead model designers to better understand weaknesses in their current approach, and allow future authors to quantitatively and compactly justify their design choices. To this end, full toolkit code is released at `https://dbolya.github.io/tide/` and opened to the community for future development.

2 The Tools

Object detection and instance segmentation primarily use one metric to judge performance: mean Average Precision (mAP). While mAP succinctly summarizes the performance of a model in one number, disentangling errors in object detection and instance segmentation from mAP is difficult: a false positive can be a duplicate detection, misclassification, mislocalization, confusion with background, or even both a misclassification and mislocalization. Likewise, a false negative could be a completely missed ground truth, or the potentially correct prediction

could have just been misclassified or mislocalized. These error types can have hugely varying effects on mAP, making it tricky to diagnose problems with a model off of mAP alone.

We could categorize all these types of errors, but it's not entirely clear how to weight their relative importance. Hoiem et al. [14] weight false positives by their prevalence in the top N most confident errors and consider false negatives separately. However, this ignores the effect many low scoring detections could have (so effective use of it requires a sweep over N), and it doesn't allow comparison between false positives and false negatives.

There is one easy way to determine the importance of a given error to overall mAP, however: simply fix that error and observe the resulting change in mAP. Hoiem et al. briefly explored this method for certain false positives but didn't base their analysis off of it. This is also similar to how the COCO eval toolkit [1] plots errors, with one key difference: the COCO implementation computes the errors *progressively*. That is, it observes the change in mAP after fixing one error, but keep those errors fixed for the next error. This is nice because at the end result is trivially 100 mAP, but we find that fixing errors progressively in this manner is misleading and may lead to false conclusions (see Sec. 2.3).

So instead, we define errors in such a way that fixing all errors will still result in 100 mAP, but we weight each error individually starting from the original model's performance. This retains the nice property of including confidence and false negatives in the calculation, while keeping the magnitudes of each error type comparable.

2.1 Computing mAP

Before defining error types, we focus our attention on the definition of mAP to understand what may cause it to degrade. To compute mAP, we are first given a list of predictions for each image by the detector. Each ground truth in the image is then matched to at most one detection. To qualify as a positive match, the detection must have the same class as the ground truth and an IoU overlap greater than some threshold, t_f, which we will consider as 0.5 unless otherwise specified. If multiple detections are eligible, the one with the highest overlap is chosen to be true positive while all remaining are considered false positives.

Once each detection has matched with a ground truth (true positive) or not (false positive), all detections are collected from every image in the dataset and are sorted by descending confidence. Then the cumulative precision and recall over all detections is computed as:

$$P_c = \frac{TP_c}{TP_c + FP_c} \qquad R_c = \frac{TP_c}{N_{GT}} \tag{1}$$

where for all detections with confidence $\geq c$, P_c denotes the precision, R_c recall, TP_c the number of true positives, and FP_c the number of false positives. N_{GT} denotes the number of GT examples in the current class.

Then, precision is interpolated such that P_c decreases monotonically, and AP is computed as a integral under the precision recall curve (approximated by

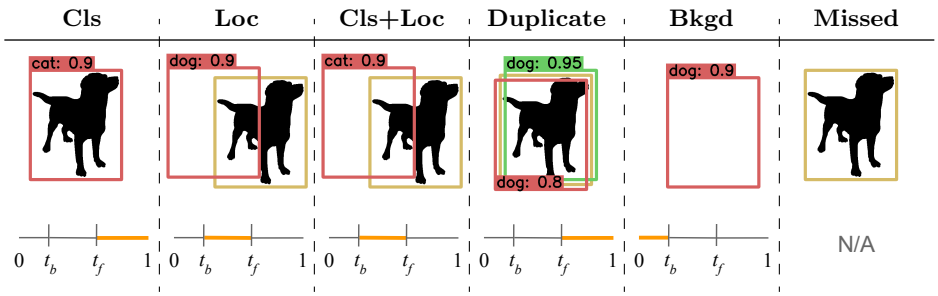


Fig. 1: **Error Type Definitions.** We define 6 error types, illustrated in the *top row*, where box colors are defined as: ■ = false positive detection; ■ = ground truth; ■ = true positive detection. The IoU with ground truth for each error type is indicated by an orange highlight and shown in the *bottom row*.

a fixed-length Riemann sum). Finally, mAP is defined as the average AP over all classes. In the case of COCO [20], mAP is averaged over all IoU thresholds between 0.50 and 0.95 with a step size of 0.05 to obtain $mAP^{0.5:0.95}$.

2.2 Defining Error Types

Examining this computation, there are 3 places our detector can affect mAP: outputting false positives during the matching step, not outputting true positives (i.e., false negatives) for computing recall, and having incorrect calibration (i.e., outputting a higher confidence for a false positive then a true positive).

Main Error Types In order to create a meaningful distribution of errors that captures the components of mAP, we bin all false positives and false negatives in the model into one of 6 types (see Fig. 1). Note that for some error types (classification and localization), a false positive can be paired with a false negative. We will use $IoU_{\max}$ to denote a false positive's maximum IoU overlap with a ground truth of the given category. The foreground IoU threshold is denoted as t_f and the background threshold is denoted as t_b, which are set to 0.5 and 0.1 (as in [14]) unless otherwise noted.

1. **Classification Error**: $IoU_{\max} \geq t_{\mathrm{f}}$ for GT of the *incorrect* class (i.e., localized correctly but classified incorrectly).
2. **Localization Error**: $t_{\mathrm{b}} \leq IoU_{\max} \leq t_{\mathrm{f}}$ for GT of the *correct* class (i.e., classified correctly but localized incorrectly).
3. **Both Cls and Loc Error**: $t_{\mathrm{b}} \leq IoU_{\max} \leq t_{\mathrm{f}}$ for GT of the *incorrect* class (i.e., classified incorrectly *and* localized incorrectly).
4. **Duplicate Detection Error**: $IoU_{\max} \geq t_{\mathrm{f}}$ for GT of the *correct* class but another higher-scoring detection already matched that GT (i.e., would be correct if not for a higher scoring detection).

5. **Background Error**: $IoU_{\max} \leq t_{\text{b}}$ for all GT (i.e., detected background as foreground).
6. **Missed GT Error**: All undetected ground truth (false negatives) not already covered by classification or localization error.

This differs from [14] in a few important ways. First, we combine both `sim` and `other` errors into one classification error, since Hoiem et al.'s `sim` and `other` require manual annotations that not all datasets have and analysis of the distinction can be done separately. Then, both classification errors in [14] are defined for all detections with $IoU_{\max} \geq t_b$, even if $IoU_{\max} < t_f$. This confounds localization and classification errors, since using that definition, detections that are both mislocalized and misclassified are considered class errors. Thus, we separate these detections into their own category.

Weighting the Errors Just counting the number of errors in each bin is not enough to be able to make direct comparisons between error types, since a false positive with a lower score has less effect on overall performance than one with a higher score. Hoiem et al. [14] attempt to address this by considering the top N highest scoring errors, but in practice N needed to be swept over to get the full picture, creating 2d plots that are hard to interpret (see the analysis in [11, 21]).

Ideally, we'd like one comprehensive number that represents how each error type affects overall performance of the model. In other words, for each error type we'd like to ask the question, how much is this category of errors holding back the performance of my model? In order to answer that question, we can consider what performance of the model would be if it didn't make that error and use how that changed mAP.

To do this, for each error we need to define a corresponding "oracle" that fixes that error. For instance, if an oracle $o \in \mathcal{O}$ described how to change some false positives into true positives, we could call the AP computed after applying the oracle as AP_o and then compare that to the vanilla AP to obtain that oracle's (and corresponding error's) effect on performance:

$$\Delta AP_o = AP_o - AP \tag{2}$$

We know that we've covered all errors in the model if applying all the oracles together results in 100 mAP. In other words, given oracles $\mathcal{O} = \{o_1, \ldots, o_n\}$:

$$AP_{o_1,\ldots,o_n} = 100 \qquad AP + \Delta AP_{o_1,\ldots,o_n} = 100 \tag{3}$$

Referring back to the definition of AP in Sec. 2.1, to satisfy Eq. 3 the oracles used together must fix all false positives and false negatives.

Considering this, we define the following oracles for each of the main error types described above:

1. **Classification Oracle**: Correct the class of the detection (thereby making it a true positive). If a duplicate detection would be made this way, suppress the lower scoring detection.

2. **Localization Oracle**: Set the localization of the detection to the GT's localization (thereby making it a true positive). Again, if a duplicated detection would be made this way, suppress the lower scoring detection.
3. **Both Cls and Loc Oracle**: Since we cannot be sure of which GT the detector was attempting to match to, just suppress the false positive detection.
4. **Duplicate Detection Oracle**: Suppress the duplicate detection.
5. **Background Oracle**: Suppress the hallucinated background detection.
6. **Missed GT Oracle**: Reduce the number of GT (N_{GT}) in the mAP calculation by the number of missed ground truth. This has the effect of stretching the precision-recall curve over a higher recall, essentially acting as if the detector was equally as precise on the missing GT. The alternative to this would be to add new detections, but it's not clear what the score should be for that new detection such that it doesn't introduce confounding variables. We discuss this choice further in the Appendix.

Other Error Types While the previously defined types fully account for all error in the model, how the errors are defined doesn't clearly delineate false positive and negative errors (since cls, loc, and missed errors can all capture false negatives). There are cases where a clear split would be useful, so for those cases we define two separate error types by the oracle that would address each:

1. **False Positive Oracle**: Suppress all false positive detections.
2. **False Negative Oracle**: Set N_{GT} to the number of true positive detections.

Both of these oracles together account for 100 mAP like the previous 6 oracles do, but they bin the errors in a different way.

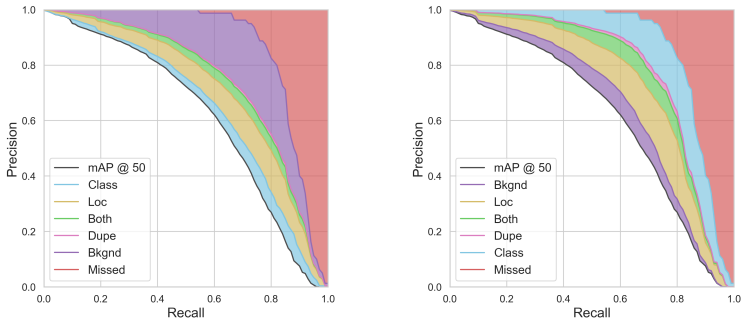
2.3 Limitations of Computing Errors Progressively

Note that we are careful to compute errors *individually* (i.e., each ΔAP starts from the vanilla AP with no errors fixed). Other approaches [1, 4] compute their errors *progressively* (i.e., each ΔAP starts with the last error fixed, such that fixing the last error results in 100 AP). While we ensure that applying all oracles together also results in 100 AP, we find that a progressive ΔAP misrepresents the weight of each error type and is strongly biased toward error types *fixed last*.

To make this concrete, we can define progressive error $\Delta AP_{a|b}$ to be the change in AP from applying oracle a given that you've already applied oracle b:

$$\Delta AP_{a|b} = AP_{a,b} - AP_b \tag{4}$$

Then, computing errors progressively amounts to setting the importance of error i to $\Delta AP_{o_i|o_1,\ldots,o_{i-1}}$. This is problematic for two reasons: the definition of precision includes false positives in the *denominator*, meaning that if you start with fewer false positives (as would be the case when having fixed most false positives already), the change in precision will be *much higher*. Furthermore, any changes in recall (e.g., by fixing localization or classification errors) amplifies the effect of precision on mAP, since the integral now has more area.



(a) Default COCO eval style error curves.

(b) Swapping the order of errors changes magnitudes drastically.

Fig. 2: **The problem with computing errors progressively.** The COCO eval `analyze` function [1] computes errors progressively, which we show for Mask R-CNN [13] detections on mAP_{50}. On the right, we swap the order of applying the classification and background oracles. The quantity of each error remains the same, but the perceived contribution from background error (purple region) significantly decreases, while it increases for all other errors. Because COCO computes background error second to last, this instills a belief that it's more important than other errors, which does not reflect reality (see Sec. 2.3).

We show this empirically in Fig. 2, where Fig. 2a displays the original COCO eval style PR curves, while Fig. 2b simply swaps the order that background and classification error are computed. Just computing background first leads to an incredible decrease in the prevalence of its contribution (given by the area of the shaded region), meaning that the true weight of background error is likely much less than COCO eval reports. This makes it difficult to draw factual conclusions from analysis done this way.

Moreover, computing errors progressively doesn't make intuitive sense. When using these errors, you'd be attempting to address them individually, one at a time. There will never be an opportunity to correct *all* localization errors, and then start addressing the classification errors—there will always be some amount of error in each category left over after improving the method, so observing $AP_{a|b}$ isn't useful, because there is no state where you're starting with AP_b.

For these reasons, we entirely avoid computing errors progressively.

3 Analysis

In this section we demonstrate the generality and usefulness of our analysis toolbox by providing detailed analysis across various object detection and instance segmentation models and across different data and annotation sets. We also compare errors based on general qualities of the ground truth, such as object size, and find a number of useful insights. To further explain complicated error cases, we provide more granular analysis into certain error types. All modes of analysis used in this paper are available in our toolkit.

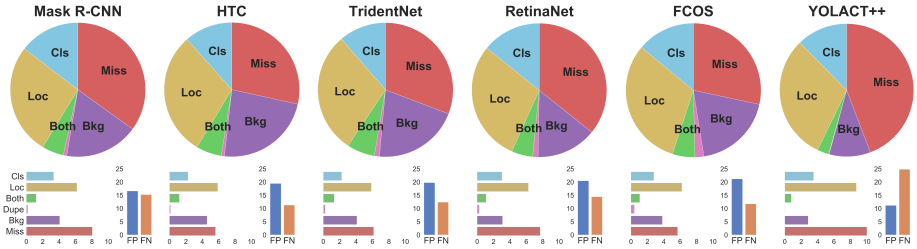


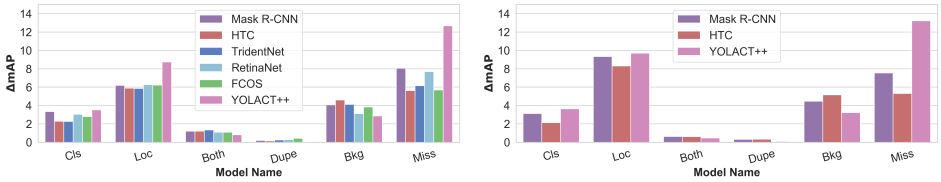
Fig. 3: **Summary of errors on COCO Detection.** Our model specific error analysis applied to various object detectors on COCO. The pie chart shows the relative contribution of each error, while the bar plots show their absolute contribution. For instance segmentation results, see the Appendix.

Models We choose various object detectors and instance segmenters based on their ubiquity and/or unique qualities which allows us to study the performance trade-offs between different approaches and draw several insights. We use Mask R-CNN [13] as our baseline, as many other approaches build on top of the standard R-CNN framework. We additionally include three such models: Hybrid Task Cascades (HTC) [5], TridentNet [18], and Mask Scoring R-CNN (MS-RCNN) [13]. We include HTC due to its strong performance, being the 2018 COCO challenge winner. We include TridentNet [18] as it specifically focuses on increasing scale-invariance. Finally, we include MS R-CNN as a method which specifically focuses on fixing calibration based error. Distinct from the two-stage R-CNN style approaches, we also include three single-stage approaches, YOLACT/YOLACT++ [3, 2] to represent real-time models, RetinaNet [19] as a strong anchor-based model, Fully Convolutional One-Stage Object Detection (FCOS) [23] as a non anchor-based approach. Where available, we use the ResNet101 versions of each model. Exact models are indicated in the Appendix.

Datasets We present our core cross-model analysis on MS-COCO [20], a widely used and active benchmark. In addition, we seek to showcase the power of our toolbox to perform cross-dataset analysis by including three additional datasets: Pascal VOC [10] as a relatively simple object detection dataset, Cityscapes [6] providing high-res, densely annotation images with many small objects, and LVIS [12] using the same images at COCO but with a massive diversity of annotated objects with 1200+ mostly-rare class.

3.1 Validating Design Choices

The authors of each new object detector or instance segmenter make design choices they claim to affect their model's performance in different ways. While the goal is almost always to increase overall mAP, there remains the question: does the intuitive justification for a design choice hold up? In Fig. 3 we present the distribution of errors for all object detectors and instance segmenters we



(a) Detection Performance. (b) Instance Segmentation Performance.

Fig. 4: **Comparison across models on COCO.** Weight of each error type compared across models. This has the same data as Fig. 3.

consider on COCO [20], and in this section we'll analyze the distribution of errors for each detector to see whether our errors line up with the intuitive justifications.

R-CNN Based Methods First, HTC [5] makes two main improvements over Mask R-CNN: 1.) it iteratively refines predictions (i.e., a cascade) and passes information between all parts of the model each time, and 2.) it introduces a module specifically for improved detection of foreground examples that look like background. Intuitively, (1) would improve classification and localization significantly, as the prediction and the features used for the prediction are being refined 3 times. And indeed, the classification and localization errors for HTC are the lowest of the models we consider in Fig. 4 for both instance segmentation and detection. Then, (2) should have the effect of eliciting higher recall while potentially adding false positives where something in the background was misclassified as an object. And this is exactly what our errors reveal: HTC has the lowest missed GT error while having the highest background error (not counting YOLACT++, whose distribution of errors is quite unique).

Next, TridentNet [18] attempts to create scale-invariant features by having a separate pipeline for small, medium, and large objects that all share weights. Ideally this would improve classification and localization performance for objects of different scales. Both HTC and TridentNet end up having the same classification and localization performance, so we test this hypothesis further in Sec. 3.2. Because HTC and TridentNet make mostly orthogonal design choices, they would likely compliment each other well.

One-Stage Methods RetinaNet [19] introduces focal loss that down-weights confident examples in order to be able to train on all background anchor boxes (rather than the standard 3 negative to 1 positive ratio). Training on all negatives by itself should cause the model to output fewer background false positives, but at the cost of significantly lower recall (since the detector would be biased toward predicting background). The goal of focal loss then is to train on all negatives without causing extra missed detections. We observe this is successful as RetinaNet has one of the lowest background errors across models in Fig. 4a, while retaining slightly less missed GT error than Mask R-CNN.

Then FCOS [23] serves as a departure from traditional anchor-based models, predicting a newly defined box at each location in the image instead of regressing

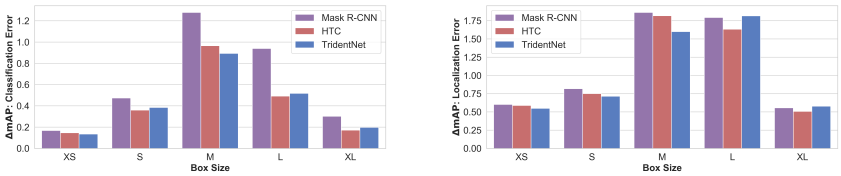
Table 2: **Mask Rescoring.** An ablation of MS-RCNN [13] and YOLACT++ [2] mask performance using the errors defined in this paper. ΔmAP_{50} is denoted as E for brevity, and only errors that changed are included. Mask scoring better calibrates localization, leading to decrease in localization error. However, by scoring based on localization, the calibration of other error types suffer. Note that this information is impossible to glean from the change in AP_{50} alone.

Method	$AP_{50}\uparrow$	$E_{cls}\downarrow$	$E_{loc}\downarrow$	$E_{bkg}\downarrow$	$E_{miss}\downarrow$	$E_{FP}\downarrow$	$E_{FN}\downarrow$
Mask R-CNN (R-101-FPN)	58.1	3.1	9.3	4.5	7.5	15.9	17.8
+ Mask Scoring	58.3	3.6	7.8	5.1	7.8	15.9	18.1
Improvement	+0.2	+0.4	**−1.5**	+0.7	+0.3	+0.0	+0.3
YOLACT++ (R-50-FPN)	51.8	3.3	10.4	3.2	13.0	10.7	27.7
+ Mask Scoring	52.3	3.6	9.7	3.2	13.2	10.1	28.2
Improvement	+0.5	+0.3	**−0.7**	+0.0	+0.2	−0.5	+0.6

an existing prior. While the primary motivation for this design choice was simplicity, getting rid of anchor boxes has other tangible benefits. For instance, an anchor-based detector is at the mercy of its priors: if there is no applicable prior for a given object, then the detector is likely to completely miss it. FCOS on the other hand doesn't impose any prior-based restriction on its detections, leading to it having one of the lowest missed detection errors of all the models we consider (Fig. 4a). Note that it also has the highest duplication error because it uses an NMS threshold of 0.6 instead of the usual 0.5.

Real-Time Methods YOLACT [3] is a real-time instance segmentation method that uses a modified version of RetinaNet as its backbone detector without focal loss. YOLACT++ [2] iterates on the former and additionally includes mask scoring (discussed in Tab. 2). Observing the distribution of errors in Fig. 3, it appears that design choices employed to speed up the model result in a completely different distribution of errors w.r.t. RetinaNet. Observing the raw magnitudes in Fig. 4a, this is largely due to YOLACT having much higher localization and missed detection error. However, the story changes when we look at instance segmentation, where it localizes almost as well as Mask R-CNN despite the bad performance of its detector (see Appendix). This substantiates their claim that YOLACT is more conducive to high quality masks and that its performance is likely limited by a poor detector.

A Note on Ablations To demonstrate the potential usefulness of this toolkit for isolating error contribution and debugging, we showcase how an ablation over error types instead of only over mAP provides meaningful insights while still being compact. As an example, consider the trend of rescoring a mask's confidence based on its predicted IoU with a ground truth, as in Mask Scoring R-CNN [16] and YOLACT++ [2]. This modification is intended to increase the score of good quality masks and decrease the score of poor quality masks, which intuitively should result in better localization. In order to justify their claims,



(a) Classification Error (b) Localization Error

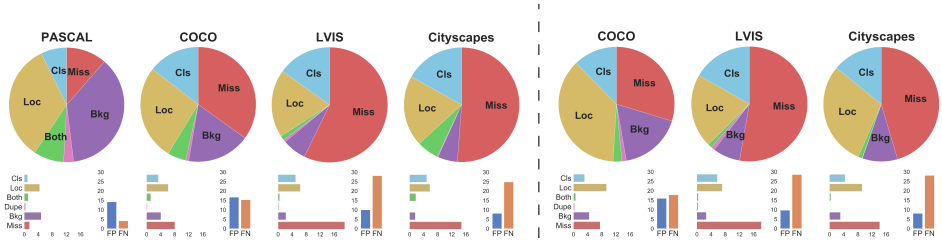
Fig. 5: **Comparison of Scales between HTC and TridentNet.** Both HTC and TridentNet have the same classification and localization error on COCO detection. Using fine analysis, we can isolate the cause of these errors further.

the authors of both papers provide qualitative examples where this is the case, but limit quantitative support to the change to an observed increase in mAP. Unfortunately, a change in mAP alone does not illuminate the cause of that change, and some ablations may show little change in mAP despite the method working. By adding the error types that were affected by the change to ablation tables (e.g., see Tab. 2) we not only provide quantitative evidence for the design choice, but also reveal side effects (such that classification calibration error went up), which were previously hidden by the raw increase in mAP.

3.2 Comparing Object Attributes for Fine Analysis

In order to compare performance across object attributes such as scale or aspect ratio, the typical approach is to compute mAP on a subset of the detections and ground truth that have the specified attributes (with effective comparison requiring normalized mAP [14]). While we offer this mode of analysis in our toolkit, this doesn't describe the effect of that attribute on overall performance, just how well a model performs on that attribute. Thus, we propose an additional approach based on the tools we defined earlier for summarizing error's affect on overall performance: simply fix errors and observe ΔmAP as before, but only those whose associated prediction or ground truth have the desired attribute.

Comparing Across Scale As an example of using this approach across different scales of objects, we return to the case of TridentNet vs. HTC discussed in Sec. 3.1. Both models have the same classification and localization error and we would like to understand where the difference, if any, lies. Since TridentNet focuses specifically on scale-invariance, we turn our attention to performance across scales. We define objects with pixel areas of between 0 and 16^2 as extra small (XS), 16^2 to 32^2 as small (S), 32^2 to 96^2 as medium (M), 96^2 to 288^2 as (L), and 288^2 and above as extra large (XL). In Fig. 5 we apply our approach across HTC and TridentNet (with Mask R-CNN detections included for reference). This comparison reveals that TridentNet localizes and classifies medium sized objects better than HTC, while HTC is better at large objects. This could potentially be why the authors of TridentNet find that they can achieve nearly the same performance by only evaluating their branch for medium sized objects [18]. Other



(a) Object Detection (b) Instance Segmentation

Fig. 6: **Performance of Mask R-CNN Across Datasets.** Because our toolkit is dataset agnostic, we can fix a detection architecture and compare performance across datasets to gain valuable insights into properties of each dataset.

comparisons between subsets of detections such as across aspect ratios, anchor boxes, FPN layers, etc. are possible with the same approach.

3.3 Comparing Performance Across Datasets

Our toolkit is dataset agnostic, allowing us to compare the same model across several datasets, as in Fig. 6, where we compare Mask R-CNN (Faster R-CNN for Pascal) across Pascal VOC [10], COCO [20], Cityscapes [6], and LVIS [12].

In this comparison, the first immediately clear pattern is that Background error decreases both in overall prevalence (pie charts) and absolute magnitude (bar charts) with increasing density of annotations. Faster R-CNN on Pascal is dominated by background error, but of reduced concern on COCO. Both LVIS and Cityscapes, which are very densely annotated, have almost no background error at all. This potentially indicates that much of the background error in Pascal and COCO are simply due to unannotated objects (see Sec. 3.4).

As expected, missed ground truths are a large issue for densely annotated datasets like LVIS and Cityscapes. The core challenge on Cityscapes is the presence of many small objects, which are well known to be difficult to detect with modern algorithms. On the other hand, LVIS's challenge is to deal with the vast number of possible objects that the detector has to recognize. We can see from the relatively normal classification error on LVIS that the model isn't particularly suffering directly from misclassifying rare objects, but instead completely failing to detect them when they appear. This is also reflected in the false positive and false negative error distributions (vertical bars). Overall, Pascal is heavily biased toward false positives, COCO is mixed, and LVIS and Cityscapes are both biased toward false negatives.

On COCO, Mask R-CNN has a harder time localizing masks (Fig. 6b) than boxes (Fig. 6a), but the opposite is true for LVIS, possibly because of its higher quality masks, which are verified with expert studies [12]. Again, this potentially indicates that a lot of the error in instance segmentation may be derived by mis-annotations.

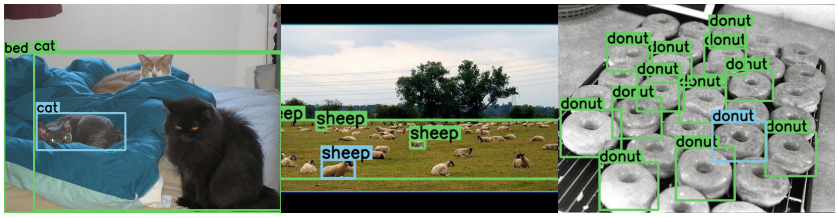


Fig. 7: **Examples of Poor Annotations.** In modern detectors, highly confident detections classified as both mislocalized and misclassified or background errors are likely to be mislabeled examples on COCO. In the first two images, the ground truth should have been labeled as crowds. In the third, some of the donuts simply weren't labeled. ■ = ground truth, ■ = predictions.

3.4 Unavoidable Errors

We find in Sec. 3.3 that a lot of the background and localization error may simply be due to mis- or unannotated ground truth. Examining the top errors more closely, we find that indeed (at least in COCO), many of the most confident errors are actually misannotated or ambiguously misannotated ground truth (see Fig. 7). For instance, 30 of the top 100 most confident localization errors in Mask R-CNN detections are due to bad annotations, while the number soars to 50 out of 100 for background error. These misannotations are simple mistakes like making the box too big or forgetting to mark a box as a crowd annotation. More examples are ambiguous: should a mannequin or action figure be annotated as a person? Should a sculpture of a cat be annotated as a cat? Should a reflection of an object be annotated as that object? Highly confident mistakes result in large changes in overall mAP, so misannotated ground truth considerably lower the maximum mAP a reasonable model can achieve.

This begs the question, what is the upper bound for mAP on these datasets? Existing analyses into the potential upper bound in object detection such as [4] don't seem to account for the rampant number of mislabeled examples. The state-of-the-art on the COCO challenge are slowly stagnating, so perhaps we are nearing the "reasonable" upper bound for these detectors. We leave this for future work to analyze.

4 Conclusion

In this work, we define meaningful error types and a way of tying these error types to overall performance such that it minimizes any confounding variables. We then apply the resulting framework to evaluate design decisions, compare performance on object attributes, and reveal properties of several datasets, including the prevalence of misannotated ground truth in COCO. We hope that our toolkit can not only serve as method to isolate and improve on errors in detection, but also lead to more interpretability in design decisions and clearer descriptions of the strengths and weaknesses of a model.

References

1. COCO Analysis Toolkit: `http://cocodataset.org/#detection-eval`, accessed: 2020-03-01
2. Bolya, D., Zhou, C., Xiao, F., Lee, Y.J.: Yolact++: Better real-time instance segmentation. arXiv:1912.06218 (2019)
3. Bolya, D., Zhou, C., Xiao, F., Lee, Y.J.: Yolact: real-time instance segmentation. In: ICCV (2019)
4. Borji, A., Iranmanesh, S.M.: Empirical upper-bound in object detection and more. arXiv:1911.12451 (2019)
5. Chen, K., Pang, J., Wang, J., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Shi, J., Ouyang, W., et al.: Hybrid task cascade for instance segmentation. In: CVPR (2019)
6. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: CVPR (2016)
7. Divvala, S.K., Hoiem, D., Hays, J.H., Efros, A.A., Hebert, M.: An empirical study of context in object detection. In: CVPR (2009)
8. Dollár, P., Wojek, C., Schiele, B., Perona, P.: Pedestrian detection: A benchmark. In: CVPR (2009)
9. Dong, H., Yang, G., Liu, F., Mo, Y., Guo, Y.: Automatic brain tumor detection and segmentation using u-net based fully convolutional networks. In: MIUA (2017)
10. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. IJCV (2010)
11. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR (2014)
12. Gupta, A., Dollar, P., Girshick, R.: Lvis: A dataset for large vocabulary instance segmentation. In: CVPR (2019)
13. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: ICCV (2017)
14. Hoiem, D., Chodpathumwan, Y., Dai, Q.: Diagnosing error in object detectors. In: ECCV (2012)
15. Hosang, J., Benenson, R., Schiele, B.: How good are detection proposals, really? In: BMVC (2014)
16. Huang, Z., Huang, L., Gong, Y., Huang, C., Wang, X.: Mask scoring r-cnn. In: CVPR (2019)
17. Kabra, M., Robie, A., Branson, K.: Understanding classifier errors by examining influential neighbors. In: CVPR (2015)
18. Li, Y., Chen, Y., Wang, N., Zhang, Z.: Scale-aware trident networks for object detection. In: ICCV (2019)
19. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: CVPR (2017)
20. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: ECCV (2014)
21. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.: Ssd: Single shot multibox detector. In: ECCV (2016)
22. Pepik, B., Benenson, R., Ritschel, T., Schiele, B.: What is holding back convnets for detection? In: GCPR (2015)
23. Tian, Z., Shen, C., Chen, H., He, T.: Fcos: Fully convolutional one-stage object detection. In: ICCV (2019)
24. Zhu, H., Lu, S., Cai, J., Lee, Q.: Diagnosing state-of-the-art object proposal methods. arXiv:1507.04512 (2015)