Combining Deep Learning and Verification for Precise Object Instance Detection

Siddharth Ancha[*], **Junyu Nan**[*], **David Held**
Carnegie Mellon University
United States
{sancha, jnan1, dheld}@andrew.cmu.edu

Abstract: Deep learning based object detectors often report false positives with very high confidence. Although they optimize generic detection performance, such as mean average precision (mAP), they are not designed for *robustness* or *verifiability*. We argue that, if a high confidence detection is made by a robot perception system, we would want high certainty that the object has indeed been detected. We present a detection system that can verify, with *high precision*, whether each detection of a machine-learning based object detector is correct or not. We present a set of verification checks based on a novel approach of using dense pixel correspondences between known images of objects and a scene, to verify whether the detections made in the scene are correct. We motivate this by developing a theoretical framework which proves that under certain assumptions, our proposed method will reject any false positives. We show that these tests can improve the overall accuracy of a base detector and that accepted examples are highly likely to be correct. This allows the detector to operate in a high precision regime, and can thus be used for robotic perception systems as a reliable instance detection method. Code is available at https://github.com/siddancha/FlowVerify.

Keywords: Robot Perception, Instance Detection, High Precision Recognition

1 Introduction

Instance detection is the task of detecting instances of a particular object in a scene. Here (as in previous work [1, 2, 3]), the term "instance" refers to a specific sub-type of object (e.g. "coke can" rather than just "can"). For example, a robot may be shown an image of a cereal box, and it may be required to detect it in a scene in order to fetch it. Unfortunately, current systems for object instance detection are not sufficiently reliable for use in real world applications; most methods will fail in real-world scenarios due to occlusions, lighting changes, viewpoint variation, and other difficulties.

Traditional non-parametric methods for object instance detection rely on keypoint-matching [4, 5, 6] or template matching [7, 8] to training images. However, these methods are not robust to large changes in object viewpoint (i.e. greater than 25 degrees [2]) and often fail for significant lighting changes. For a robust home perception system, we cannot always guarantee that the objects being observed will be viewed from the same conditions as in training. Thus we desire to have an object detection system that is robust to significant changes in the object viewpoint, as well as lighting, occlusions, and other variations. Recently, a number of machine learning approaches have been used for object instance detection [9, 10, 2]. However, machine-learning based approaches that learn parameters from data often produce a large number of false positive detections. These false detections can prevent deployment of robots in real-world applications.

We argue that a combination of parametric and non-parametric object detection methods is crucial for robust and precise object detection. While machine learning methods enable us to leverage large amounts of data to learn necessary invariances like lighting and viewpoint changes, non-parametric methods that match candidate test objects to training templates can ensure that detections are robust and verifiable. In sections 3 and 4.1 we propose verification tests that operate on top of existing

[*]Equal contribution

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

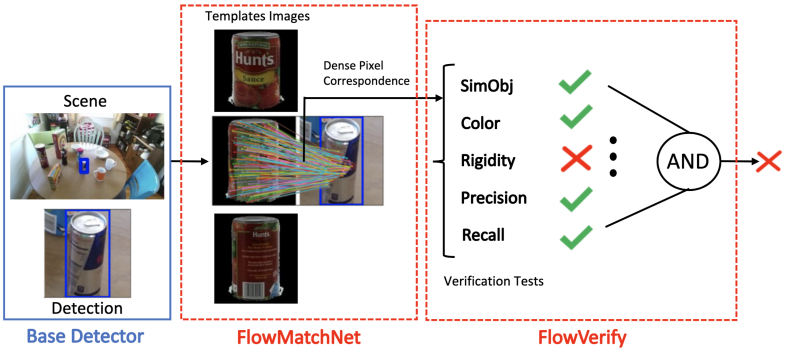


Figure 1: Pipeline of our instance detection verification system. *Base Detector*: generates object instance detections from real scenes; FLOWMATCHNET: computes dense pixelwise correspondence between template images of candidate object & detection; FLOWVERIFY: a suite of verification tests is applied to the detection using pixel correspondence. Detection is accepted only if all tests pass.

machine learning based detectors; these tests take detections as input and verify them by matching against training examples. Thus, by not throwing away training data (in contrast to the parametric machine learning approach) and using them to verify detections at test time, we show that we can increase the accuracy of the detector, especially in the high-precision regime. This makes our system more precise, reliable and interpretable, making it more suitable for robotics applications.

Furthermore, we advocate that a robust detection system should make use of both learning and non-learning approaches. We show that we can robustly compute dense pixel-wise correspondences between images using a machine-learning based matching approach, making the correspondences reliable across a range of viewpoints and lighting changes. We run our pixelwise-matching system, which we call "FLOWMATCHNET," between each proposed detection and a set of template images of an object. Then we use a series of non-learning based verification tests, which we call "FLOWVERIFY," to check if these correspondences are reasonable and ascertain whether the detection is similar to any of the templates. If any of the tests fail, FLOWVERIFY will re-rank the detection by reducing its initial confidence score. Our method can be combined with any object detector to improve its accuracy, especially in the high-precision regime.

We develop a theoretical framework for verified object detection. Specifically, We show that under certain simple conditions, a suite of verification tests will reject all false positive detections. Under these assumptions, any detection that is verified by our tests is guaranteed to be correct. Based on this theory, we implement FLOWVERIFY, an approximate but more practically suitable version of these tests and show that it can improve the performance of the detector in the high precision regime and perform no worse in the low-precision regime, thereby improving the overall accuracy of the detector. Figure 1 shows a diagram of our pipeline.

We demonstrate that our method can be used in a *one-shot* fashion, i.e. that our method can detect novel objects that have never been seen before during training. The system processes template images of novel objects at test time for detection. All components of our pipeline are trained on one set of objects but evaluated on a different set of objects; we also use a third validation set of objects for finetuning. Hence, we expect that our model can be downloaded and run off-the-shelf on custom objects without the need for any re-training or finetuning. Alternatively, our method can also be fine-tuned on the objects being detected for even better performance.

Furthermore, our system is flexible in the number of template images it can use for matching and verification. Hence, our system provides an accuracy-speed tradeoff; the user of our system can use more templates from various viewpoints to achieve higher accuracy, or choose to use fewer templates to increase speed. We summarize our contributions as:

- A theoretical framework for verified object detection, which guarantees no false positives.
- An approximate implementation of this framework which leads to improved detector performance, especially in the high-precision regime.
- A novel approach for object detection that combines parametric learned bounding box detection, parametric learned correspondence matching, and non-learned verification tests.

2 Related Work

Object Instance Detection. Traditional methods for object instance detection rely on keypoint-matching [4, 5, 6] or template matching [7, 8]. Recently, detectors based on deep neural networks have shown improved performance over these traditional approaches. For example, Target Driven Instance Detection (TDID) [11] is a state of the art instance detection method based on a Siamese style neural architecture [12]. We compare to TDID in our work and show improved performance over this state-of-the-art baseline.

Grocery product recognition. There has been significant effort in recognizing products on shelves of retail stores such as grocery product recognition [13, 14, 15, 16]. This problem is simpler than the general object instance recognition problem that we are aiming to solve due to the structured environment. For example, objects in such scenes are typically placed in front-facing canonical viewpoints with minimal occlusions and good lighting conditions. Furthermore, there is often a great deal of contextual information, such as the location of the product on the shelf. Our method is designed for the more general usage in the home or other unstructured locations.

Dense Pixel Correspondence. Our work builds on past work for computing dense pixel correspondences [17, 18, 19, 20]; unlike past work, we use these correspondences for improving the performance of object instance recognition.

Classification with reject option. We use a set of verification tests that try to verify a detection and an object class that is proposed by an initial object detector. This notion of verification is related to the literature on classification with rejection. Some of these previous approaches assume a cost function for rejection is provided and find the optimum rejection rule [21, 22, 23] or learn classification and rejection functions simultaneously [24, 25]. Some works [26, 27] treat the problem as conformal prediction, where the classification system can predict any subset of classes, including the null set which stands for rejection. In contrast, in our work, we verify whether the object class output by the detector is correct, and we re-score the confidence of detections based on the verification tests.

3 Theoretical Framework

In this section, we lay out a formal framework for high-precision instance detection. We specify a set of assumptions and prove that under these assumptions, the verification-based detector will have no false positives. We then discuss a modification of this method that we implement in practice which improves overall detection accuracy.

Problem Statement. We assume there are O categories of objects that we are trying to recognize. We also assume access to a dataset which consists of a variety of images per object, recorded from a set of different viewpoints and lighting conditions. We refer to these images as 'template images' (see Figure 1). At test time, we are given 'scene images' – these are real world images that contain (multiple) objects that need to be detected (see Figure 1). We also assume that we have an object instance detector trained to detect objects of interest; this detector will return detections of the target objects, along with a proposed object class for each detection. However, many of these proposed object classes will be incorrect, i.e. false positives. The goal of our framework is to filter these out.

Notation. We denote by O_i the ith object from the O categories of objects. For each object O_i, we assume access to a dataset of M_i images recorded from a variety of viewpoints and lighting conditions; we denote $I_{i,m}$ as the mth image for object O_i. We denote by $T : (u_1, v_1) \rightarrow (u_2, v_2)$ a 2D mapping from pixels of one image to the pixels of another image. We overload notation such that $T(I)$ is an application of the 2D mapping T to all pixels in the image I to produce a new image $T(I)$. Let $r(T)$ denote the "rigidity" of such a transformation, measured by the fraction of inlier pixel matches under the best approximating rigid transformation (see Appendix A). We denote by T^R the set of such 2D mappings which are perfectly rigid, such that $r(T) = 1$. Let D denote a detection in a scene image (e.g. a crop of a scene image). Note that, in the theoretical framerwork, we are implicitly assuming that all detections contain an image of some object in our dataset.

We assume access to a similarity classifier which returns a score $c(I_1, I_2)$ indicating the confidence that images I_1 and I_2 each contain the same object class. We also assume access to a distance metric $d(I_1, I_2)$ that measures the the distance between images I_1 and I_2; this could be any distance metric that satisfies certain properties specified in Appendix A such as pixelwise $L2$-distance or

normalized cross-correlation. We make the below assumptions under which we will prove that our algorithm will lead to no false positives. We will need to relax these assumptions for a practical implementation; nonetheless, this framework provides the theoretical basis for our approach.

Assumption 1. ***(Dense Dataset)***
For each detection D of class O_i, $\exists m \in M_i, T \in T^R$ such that $d(T(I_{i,m}), D) \leq \gamma$.

Assumption 2. ***(Similarity Classifier Smoothness)***
We have a similarity classifier c with a corresponding constant δ that satisfies the following property: for any two images I_1 and I_2, and for any detection D, if $\exists T \in T^R$ such that $d(T(I_1), I_2) \leq 2\gamma$, then $|c(I_1, D) - c(I_2, D)| < \delta$.

The first assumption states that our dataset is dense enough such that every detection can be constructed as a rigid transformation of some image in the dataset, with a bounded lighting change applied. The second assumption states that the similarity classifier c is a smooth function: if two images I_1 and I_2 are sufficiently similar such that I_2 can be created by a rigid transformation and a small lighting change applied to I_1, then the similarity classifier c will output a similar score when comparing I_1 and D as when comparing I_2 and D.

Our object detection pipeline proceeds as follows: we assume that an initial detector finds detections D in an image and proposes an object class O_i for each detection. We then pass the detection and its corresponding proposed class to our verification system TheoreticalFlowVerify(i, D) which returns True if it can verify that D contains an image of class O_i and False otherwise. Note that TheoreticalFlowVerify may return False either because O_i is the wrong object class or because the algorithm is simply unable to verify the class of the object. However, as we will show, the key to our approach is that TheoreticalFlowVerify will only return True when O_i is the true object class; hence when the system returns True, we can rely on the detection being accurate.

We present TheoreticalFlowVerify in detail in Appendix A; this method is a theoretical version of our practical algorithm FlowVerify described in Section 4.1. The method iterates over all images $I_{i,m}$ of the proposed object class O_i; it then estimates a set of dense pixel-wise correspondences $\hat{T}$ between the detection D and the image $I_{i,m}$. It then determines whether it can validate that D is of class O_i based on image $I_{i,m}$. These verification tests are:

1. Similar Object Comparison: Tests whether an image $I_{j,n}$ of another object class O_j looks similar to D according to similarity function c; formally: $\forall j \neq i, \forall n \in M_j$, if $c(I_{i,m}, D) < c(I_{j,n}, D) + \delta$, return False
2. Color Comparison: Tests whether the image distance between $I_{i,m}$ transformed using $\hat{T}$ and the detection D is sufficiently small: if $d(\hat{T}(I_{i,m}), D) > \gamma$, return False
3. Flow Rigidity: Tests whether the the correspondences $\hat{T}$ could have been derived from a rigid object transformation: if $r(\hat{T}) < 1$, return False

Theorem 1. ***(No False Positive Theorem)*** *[Proof: see Appendix A]*
Under assumptions 1 and 2, TheoreticalFlowVerify *does not produce any false positives. That is, the following statement always holds:* TheoreticalFlowVerify(i, D) *returns False whenever the ground-truth class for detection D is different from O_i.*

Note that TheoreticalFlowVerify(i, D) returns False whenever it cannot verify that the ground-truth class of D is O_i. This can sometimes occur even if O_i represents the ground-truth class of D, if the estimated correspondences $\hat{T}$ are not accurate. It will also return false whenever the ground-truth class of D is not O_i. Thus TheoreticalFlowVerify has 100% precision; every time it returns True, the proposed object class is the same as the ground-truth. On the other hand, the recall of the algorithm is not guaranteed; it may return False for an arbitrary proportion of examples, even if the proposed object class is correct. Note that the No False Positive Theorem does not make any assumptions on the quality of the set of predicted correspondences $\hat{T}$; if the correspondences are not accurate, then TheoreticalFlowVerify will also reject the detection. We will now describe an approximation of this framework that leads to a practical implementation of this algorithm.

4 Approach

We provide a brief overview of our pipeline for high-precision detection. It consists of three stages:

1. *Base Instance Detector*: We first run an instance detector trained on the objects of interest. This stage provides candidate bounding box detections for target objects, as well as a proposed object class O_i for each box. The focus of our work is verifying these detections, as described below.
2. *Dense Pixel-wise Correspondence* (FLOWMATCHNET): We next predict dense pixel-wise correspondences between template images and each proposed detection, cropped from the scene. See Appendix B for more details on the network architecture of FLOWMATCHNET.
3. *Verification tests* (FLOWVERIFY): Given the proposed detection and template images and estimated pixel-wise correspondences between the two, we conduct a set of *verification tests* to ascertain whether the proposed class of the detection is correct.

4.1 Verification Tests

We design verification tests that are modifications to our theoretical framework (Section 3 and Appendix A). These tests, which we call 'FLOWVERIFY' tests (since they make use of predicted flow correspondences), are intended to be stringent; any detection that does not pass these tests is deemed 'rejected' and will receive a lower detection score than the detections that pass the tests. We can be highly confident that the detections that pass our verification tests are likely to be true detections, boosting the performance of our detector in the high-precision regime, as our results demonstrate. The first three verification tests are derived from our theoretical framework of Section 3:

1. SIMOBJ: This test corresponds to the similar object comparison test in our theoretical framework. For each detection, there should be only one target object being matched to it with high confidence. The similar object comparison test is the following: for each detection D we compute a confidence score $c(D)$ using the initial detector (such as TDID [11]). We then search whether there is another detection D' of another object which has an IOU of at least 0.5 with D. If no such detection is found, $\text{SIMOBJ} = 1$. Otherwise, the similarity score for D is the minimum of the confidence difference across all other detections D': $\text{SIMOBJ} = \min_{D'} \max(0, c(D) - c(D'))$. We define the *similar object* test using a boolean variable given by $T_{\text{SIMOBJ}} = (\text{SIMOBJ} > \eta_{diff})$.

2. FCOLOR: This test corresponds to the color comparison test in our theoretical framework. We estimate the pixel-wise correspondences $\hat{T}$ between a template image I_i of the proposed object class O_i and the detection D. We then check the image similarity between D and the template image transformed using the predicted correspondences $\hat{T}(I_i)$. We use normalized cross correlation ("ncc") to measure this similarity, which lies in $[-1, 1]$. We define $\text{FCOLOR} = \frac{1}{2}(\text{ncc}(\hat{T}(I_i), D) + 1) \in [0, 1]$ and we define the *flow color* test using a boolean variable given by $T_{color} = (\text{FCOLOR} > \alpha_{color})$.

3. FRIGIDITY: This test corresponds to the rigidity test in our theoretical framework. Assuming objects are rigid, if the detection box contains the target object, then we would expect an ideal mapping between the detection and a corresponding template image from a similar viewpoint to describe a rigid body transformation. We use RANSAC [28] to find the best-fit fundamental matrix (using the 8-point algorithm [29]) that maximizes the number of corresponding inlier pairs. The proportion of inliers under the best-fit mapping is a measure of how rigid the flow is. We define flow rigidity as

$$\text{FRIGIDITY} = \frac{\text{\#inliers}}{\text{\#total correspondences}}$$

and we define a *flow rigidity* test using a boolean variable $T_{rig} = (\text{FRIGIDITY} > \alpha_{rig})$. Besides the verification tests motivated by our theoretical framework, we design two additional tests to evaluate the extent of each bounding box prediction:

4. FPRECISION: If we expect the detection box to contain the entire object (and not be too small), we would expect all pixels in the template image to be mapped to pixels *inside* the bounding box. We define *flow precision* as

$$\text{FPRECISION} = \frac{\text{\#target correspondences mapped to inside bbox}}{\text{\#total correspondences}}$$

and the *flow precision* test is defined using a boolean variable $T_{prec} = (\text{FPRECISION} > \alpha_{prec})$.

5. FRECALL: Complementary to precision, if we expect the detection box to contain only the detected object (and not be too large), we would expect the flow mapping to cover all pixels of the detection box. To measure this, we can look at the bounding box that tightly fits the extent of the mapped pixels in the scene. We define *flow recall* as the IoU between the box suggested by flow to the bounding box output by the detector, and compute a *flow recall* score as

$$\text{FRECALL} = \text{IoU}(\text{detector bbox}, \text{bbox suggested by flow}).$$

The *flow recall* test is defined using a boolean variable $T_{rec} = (\text{FRECALL} > \alpha_{rec})$. Our overall verification test is

$$\text{FLOWVERIFY} = T_{\text{SIMOBJ}} \wedge T_{color} \wedge T_{rig} \wedge T_{prec} \wedge T_{rec},$$

with given threshold values and parameters $\eta_{diff}, \alpha_{color}, \alpha_{rig}, \alpha_{prec}, \alpha_{rec} \in [0, 1]$, tuned with a validation set as described in Section 5. For a given object of interest, we could have multiple template images from different viewpoints and lighting conditions. In this case, we say that the set of template images for a given class passes FLOWVERIFY if any one of the template images for that class passes these tests.

While FLOWVERIFY tests are designed to improve performance in high precision regime, they might worsen performance in the low-precision regime as they could reject true positives. As there are cases where performance in low-precision regime is also important, instead of completely rejecting detections not passing FLOWVERIFY tests, we rerank all detections from our base detector based on 1) whether a detection passes FLOWVERIFY tests and 2) the score predicted by base detector. All detections that pass FLOWVERIFY are ranked higher than all detections that don't. The reranking procedure can be viewed as reducing the confidence of the detections that do not pass the FLOWVERIFY tests. As we will see, this improves performance in the high-precision regime while at the same time maintaining performance in the low-precision regime.

5 Experiments

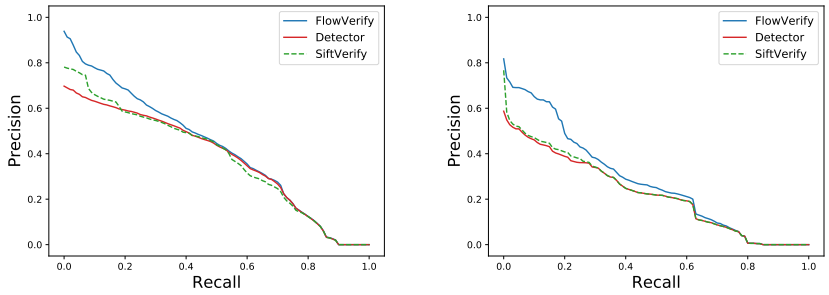
Datasets. We evaluate our framework on two tests sets – the GMU Kitchens test split [30] and W-RGBD scenes v1 Dataset [31]. Both datasets contain images of objects placed in indoor scenes such as kitchen surfaces, table tops, and living rooms. For FLOWMATCHNET and FLOWVERIFY, at test-time we use 15 equally spaced viewpoints per object taken from [32] or [33] to form our "template images". We also vary the number of template images to explore the speed-accuracy tradeoff of our system in section 5.1. For GMU, we evaluate on the 11 BigBIRD objects; for W-RGBD, we evaluate on 9 textured objects.

Base Instance Detector In this work, we focus on improving the precision of an existing instance detector. One could use any instance detector with our approach; we use *target driven instance detection* (TDID) [11], as this method produces state-of-the-art results for instance detection, including one-shot instance detection that we evaluate on in our work. In the one-shot scenario, the training and validation objects are separate from the objects evaluated on at test time; the detector must generalize to novel objects. TDID showed state-of-the-art performance for this task [11]. Similarly, we train other components of our system, such as FLOWMATCHNET, in a one-shot manner, such that our entire system can be used to detect novel objects.

The base detector TDID is trained on the Active Vision Dataset (AVD) [34], W-RGBD scenes Dataset [31], and synthetic images from Cut-Paste-Learn [35]. GMU objects common to the AVD dataset are removed from training so that we can evaluate this detector in a one-shot manner. In order to evaluate on W-RGBD Scenes dataset in a one-shot manner as well, for this evaluation we use the TDID model released by [11] that is trained only on AVD.

FLOWMATCHNET. Similar to previous optical flow models [19, 20], we train FLOWMATCHNET on synthetic data for which ground-truth correspondences can be computed. We train FLOWMATCHNET successively on synthetic datasets of increasing difficulty. First we train on objects from MS-COCO [36] with synthetic affine transformations; next we train on BigBIRD [32] images that are cropped out and blended into background via homographic transformations, randomized lighting conditions, and image blurs. More details are in Appendix C.

FLOWVERIFY. We use a validation set to tune the parameters for the tests in FLOWVERIFY. As our validation set, we use the YCB-Video training set, on which we tune the values



(a) (b)

Figure 2: Precision-Recall (PR) curves for the base instance detector, the detector improved by FLOWVERIFY, and the SIFTVERIFY baseline, evaluated on the GMU (a) and W-RGBD (b) datasets.

for $\alpha_{rig}, \alpha_{color}, \alpha_{prec}, \alpha_{rec}, \eta_{diff} \in [0, 1]$ by optimizing mAP. We find the optimal values to be $\alpha_{rig} = 0.9, \alpha_{color} = 0.5, \alpha_{prec} = 0.9, \alpha_{rec} = 0.3$ and $\eta_{diff} = 0.0$ (meaning that all other object classes should have strictly lesser confidence scores if detected).

5.1 Results

Figure 2 shows precision-recall curves of the base instance detector (TDID) and the improved detector using FLOWVERIFY, on GMU (a) and W-RGBD (b) datasets. On both datasets, FLOWVERIFY significantly improves detection performance in the high-precision regime to the left end of the PR curve. For high scoring detections, it makes significantly fewer errors than TDID as it is able to filter out many false positive detections using the verification tests. Table 1 summarizes mAP and maximum precision of baseline detectors and FLOWVERIFY on GMU and W-RGBD. It is worth noting that although FLOWVERIFY is designed to reject false positives and improve performance in the high-precision regime, the PR curves show that, with the re-ranking procedure, the performance does not degrade in the low-precision regime (right side of the curve). In the low-precision regime, FLOWVERIFY nearly matches the base detector in performance. This shows that verification tests can make an instance detector more reliable and precise while simultaneously improving overall performance.

	GMU-Test		RGBD		
Method	mAP	max Precision	mAP	max Precision	time(s)
Base Detector	0.379	0.697	0.222	0.587	**0.037**
SIFTVERIFY	0.382	0.781	0.228	0.767	0.940
FLOWVERIFY	**0.422**	**0.939**	**0.278**	**0.817**	0.872

Table 1: Overall performance (mAP) and max Precision results.

We note that our method presents a tradeoff between precision/accuracy and timing performance. As more template images are used, the precision/accuracy of our system increases, but it will take longer to run. Hence, the user of our system can adjust the number of template images based on their specific needs. We report the running times of our method for varying number of template viewpoints in Table 2.

	GMU-Test		RGBD		
Method	mAP	max Precision	mAP	max Precision	time(s)
FLOWVERIFY(15 vp)	**0.422**	0.939	**0.278**	0.817	0.872
FLOWVERIFY-12vp	0.413	0.946	**0.278**	0.838	0.696
FLOWVERIFY-9vp	0.411	**0.962**	0.275	**0.847**	0.540
FLOWVERIFY-6vp	0.408	0.898	0.263	0.766	0.379
FLOWVERIFY-3vp	0.405	0.932	0.252	0.783	**0.201**

Table 2: Tradeoff of performance vs speed as we vary the number of viewpoints.

(a) (b) (c) (d)

Figure 3: (a, b): False positives from the base detector filtered by FLOWVERIFY; (c, d): True positives from the base detector accepted by FLOWVERIFY.

Table 3 in the Appendix shows an ablation analysis of our method; we remove verification tests in FLOWVERIFY one at a time and report performance on the test set. Dropping FRIGIDITY leads to the largest drop in performance in terms of both mAP and maximum precision.

5.2 SIFTVERIFY Baseline

We implemented and evaluated another verification method using SIFT-based [37, 4] keypoint correspondences as a baseline for our learning-based dense-correspondences computed by FLOWMATCHNET. This baseline is designed to test our hypothesis that verification tests should make use of both machine learning and non-machine learning approaches; we use machine learning for computing the correspondences (FLOWMATCHNET) but we use the non-learning based verification tests of FLOWVERIFY to verify detections.

In contrast, SIFT [37, 4] is a non-learning based approach for computing correspondences, and we combine it with a non-learning based approach for verification. We implement verification tests on top of SIFT that are analogous to FLOWVERIFY, described in more detail in Appendix E. We call this baseline SIFTVERIFY. Figure 2 and Table 1 show the performance on the GMU and W-RGBD datasets. For both datasets, SIFTVERIFY slightly improves performance over the base detector, but performs consistently worse than FLOWVERIFY. This demonstrates the value of using machine learning for computing correspondences, even if the final verification tests are not learned.

5.3 Qualitative Analysis

Figure 3 (a, b) shows examples of false positive detections output by the base detector (TDID) but filtered out using FLOWVERIFY. Example (a) is a detection with high FRIGIDITY, FPRECISION, and FRECALL scores, but with a low FCOLOR score. In such cases, the detection is of an object with very different color and texture from the target object, resulting in low color similarity. Example (b) shows a detection with high FCOLOR and FPRECISION scores but with a low FRIGIDITY score. Here, the matched colors are fairly similar; however, the correspondences cannot be derived from a rigid body transformation, and hence the detection has a low FRIGIDITY score. Examples (c) and (d) in Figure 3 show true positive predictions output by the base detector which pass FLOWVERIFY. We can see that the flow quality is also quite good by the correct matching of features across the target and the bounding box area.

6 Conclusion

We have proposed a method to combine machine learning based detection and correspondence matching with non-learning based verification tests to increase the accuracy of an existing instance-detection system in the high-precision regime (without reducing overall detection performance). The verification tests are based on dense-pixel correspondences computed between the detection and template images; we reduce the confidence of any detection that does not pass these tests, thereby rejecting many false positives. Our system is grounded in a novel theoretical framework that we prove leads to no false positives, under certain assumptions. Furthermore, we use our method in a one-shot fashion, applying our approach to a novel set of objects at test time without finetuning. We hope that our system will be useful for robotic systems that need reliable performance for high confidence detections.

Acknowledgments

This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092 as well as by the NSF under Grant No. IIS-1849154.

References

[1] Z. Xie, A. Singh, J. Uang, K. S. Narayan, and P. Abbeel. Multimodal blending for high-accuracy instance recognition. In *IROS*, pages 2214–2221. IEEE, 2013.

[2] D. Held, S. Thrun, and S. Savarese. Robust single-view instance recognition. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 2152–2159. IEEE, 2016.

[3] G. Georgakis, M. A. Reza, A. Mousavian, P.-H. Le, and J. Kosecka. Multiview rgb-d dataset for object instance detection. *arXiv preprint arXiv:1609.07826*, 2016.

[4] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *ECCV*, pages 404–417. Springer, 2006.

[6] A. Quadros, J. P. Underwood, and B. Douillard. An occlusion-aware feature for range images. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4428–4435. IEEE, 2012.

[7] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *PAMI*, 15(9):850–863, 1993.

[8] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient response maps for real-time detection of textureless objects. *PAMI*, 34(5):876–888, 2012.

[9] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE, 2011.

[10] G. Georgakis, A. Mousavian, A. C. Berg, and J. Kosecka. Synthesizing training data for object detection in indoor scenes. *arXiv preprint arXiv:1702.07836*, 2017.

[11] P. Ammirato, C.-Y. Fu, M. Shvets, J. Kosecka, and A. C. Berg. Target driven instance detection. *arXiv preprint arXiv:1803.04610*, 2018.

[12] G. Koch. Siamese neural networks for one-shot image recognition. 2015.

[13] M. Merler, C. Galleguillos, and S. Belongie. Recognizing groceries in situ using in vitro training data. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[14] M. George and C. Floerkemeier. Recognizing products: A per-exemplar multi-label image classification approach. In *European Conference on Computer Vision*, pages 440–455. Springer, 2014.

[15] A. Franco, D. Maltoni, and S. Papi. Grocery product detection and recognition. *Expert Systems with Applications*, 81:163–176, 2017.

[16] W. Geng, F. Han, J. Lin, L. Zhu, J. Bai, S. Wang, L. He, Q. Xiao, and Z. Lai. Fine-grained grocery product recognition by one-shot learning. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 1706–1714. ACM, 2018.

[17] C. B. Choy, J. Gwak, S. Savarese, and M. Chandraker. Universal correspondence network. In *Advances in Neural Information Processing Systems*, pages 2414–2422, 2016.

[18] P. R. Florence, L. Manuelli, and R. Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *arXiv preprint arXiv:1806.08756*, 2018.

[19] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766, 2015.

[20] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, volume 2, page 6, 2017.

[21] C. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on information theory*, 16(1):41–46, 1970.

[22] P. L. Bartlett and M. H. Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(Aug):1823–1840, 2008.

[23] C. Cortes, G. DeSalvo, and M. Mohri. Boosting with abstention. In *Advances in Neural Information Processing Systems*, pages 1660–1668, 2016.

[24] Y. Geifman and R. El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. *arXiv preprint arXiv:1901.09192*, 2019.

[25] C. Cortes, G. DeSalvo, and M. Mohri. Learning with rejection. In *International Conference on Algorithmic Learning Theory*, pages 67–82. Springer, 2016.

[26] G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

[27] Y. Hechtlinger, B. Póczos, and L. Wasserman. Cautious deep learning. *arXiv preprint arXiv:1805.09460*, 2018.

[28] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24 (6):381–395, 1981.

[29] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133, 1981.

[30] G. Georgakis, M. A. Reza, A. Mousavian, P.-H. Le, and J. Kosecka. Multiview rgb-d dataset for object instance detection. *arXiv preprint arXiv:1609.07826*, 2016.

[31] K. Lai, L. Bo, and D. Fox. Unsupervised feature learning for 3d scene labeling. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3050–3057. IEEE, 2014.

[32] A. Singh, J. Sha, K. S. Narayan, T. Achim, and P. Abbeel. Bigbird: A large-scale 3d database of object instances. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 509–516. IEEE, 2014.

[33] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *2011 IEEE International Conference on Robotics and Automation*, pages 1817–1824, May 2011. doi:10.1109/ICRA.2011.5980382.

[34] P. Ammirato, P. Poirson, E. Park, J. Košecká, and A. C. Berg. A dataset for developing and benchmarking active vision. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1378–1385. IEEE, 2017.

[35] D. Dwibedi, I. Misra, and M. Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *The IEEE international conference on computer vision (ICCV)*, 2017.

[36] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[37] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, volume 2, pages 1150–1157. IEEE, 1999.

[38] R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems*, pages 9628–9639, 2018.

Appendix

A Theoretical Framework

In this section, we lay out a theoretical framework for instance detection with no false positives. We will specify a set of assumptions and prove that our detector will have no false positives if these assumptions are satisfied. Our practical system will make approximations to these assumptions; nonetheless the below theoretical framework will form the basis to our approach.

Notation. We denote by O_i the ith object from the O categories of objects. For each object O_i, we assume access to a dataset of M_i images recorded from a variety of viewpoints and lighting conditions; we denote $I_{i,m}$ as the mth image for object O_i.

We denote by $T : (u_1, v_1) \rightarrow (u_2, v_2)$ a 2D mapping from pixels of one image to the pixels of another image. We overload notation such that $T(I)$ is an application of the 2D mapping T to all pixels in the image I to produce a new image $T(I)$. We denote by T^R the set of such 2D mappings which are perfectly rigid, i.e. mappings that could be derived from some rigid object transformation. In other words, if $T \in T^R$, then there exists some Fundamental Matrix F such that, for all $x = (u, v)$, we have that ${x'}^\top F x = 0$, where $x' = T(u, v)$. Let $r(T)$ denote the "rigidity" of such a transformation, measured by the fraction of inlier pixel matches under the best approximating rigid transformation. In other words, for any given transformation T, we find

$$r(T) = \max_{\bar{T} \in T^R} \frac{\sum_{u,v} \mathbb{1}\{||T(u, v) - \bar{T}(u, v)|| \leq \epsilon\}}{\sum_{u,v} 1} \tag{1}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function and ϵ is a constant. We describe our practical implementation of this function in Section 4.1. Since T^R is the set of perfectly rigid transformations, for all $T \in T^R$, we have that $r(T) = 1$. Also, if $\epsilon = 0$, $r(T) = 1$ implies that $T \in T^R$. However for practical reasons, we relax ϵ to a non-zero but small value in our implementation.

Let D denote a detection in a scene image (e.g. D is a crop of a scene image). We also denote $gt(D)$ as the ground-truth object class for the object contained in the image D. Note that we are implicitly assuming that all detections contain an image of some object in our dataset.

We assume access to a similarity classifier which returns a score $c(I_1, I_2)$ that indicates the confidence that images I_1 and I_2 each contain the same object class. Practically, this corresponds to our base one-shot instance detector. We will specify below the assumptions that we make about the similarity classifier.

We also assume access to a distance metric $d(I_1, I_2)$ which measures the the distance between images I_1 and I_2; Any distance function $d : \mathcal{I}^2 \rightarrow \mathbb{R}^+$ that satisfies the following two properties can be used (the properties will be necessary for the proofs later):

1. Triangle Inequality: $d(I_1, I_2) \leq d(I_1, I_3) + d(I_2, I_3)$.
2. Permutational Invariance: $d(I_1, I_2) = d(\sigma(I_1), \sigma(I_2))$, where σ is any permutation of image pixels.

Note that many distance metrics satisfy the above properties, such as the maximum difference in pixel intensities $d(I_1, I_2) = ||I_1 - I_2||_\infty$, any L_p norm, or normalized cross-correlation that we implement in our system.

We denote $F(I_1, I_2)$ as a function that computes a 2D mapping between images I_1 and I_2. Typically, the objective of this function is to find a rigid 2D mapping that minimizes some distance metric d, i.e.

$$\arg \min_{T \in T^R} d(T(I_1), I_2) \tag{2}$$

In the below theorem, we make no assumptions about the output of $F(I_1, I_2)$; we will prove that our system will have no false positives, regardless of the output of $F(I_1, I_2)$. Nonetheless, if $F(I_1, I_2)$ does output a transformation that optimizes Equation 2, then this will maximize the recall of our algorithm.

We make the below assumptions; we will prove that, with these assumptions, our algorithm will lead to no false positives. In practice, we will need to relax these assumptions for a practical implementation; nonetheless, this theoretical framework provides the basis for our approach.

Assumption 1. (Dense Dataset)
For each detection D of ground-truth class O_i, $\exists m \in M_i, T \in T^R$ such that $d(T(I_{i,m}), D) \leq \gamma$.

This assumption states that our dataset is dense enough such that every detection can be constructed as a rigid transformation of some image in the dataset, with a bounded lighting change applied. Note that this assumption implicitly ignores occlusions or background variation, although our practical implementation handles occlusions and background variation for the detections.

Assumption 2. (Similarity Classifier Smoothness)
We have a similarity classifier c with a corresponding constant δ that satisfies the following property: for any two images I_1 and I_2, and for any detection D, if $\exists T \in T^R$ such that $d(T(I_1), I_2) \leq 2\gamma$, then $|c(I_1, D) - c(I_2, D)| < \delta$.

This assumption states that the similarity classifier c is a smooth function, in the following sense: if two images I_1 and I_2 are sufficiently similar such that I_2 can be created by a rigid transformation and a small lighting change applied to I_1, then the similarity classifier c will output a similar score when comparing I_1 and D as when comparing I_2 and D.

Note that we make no other assumptions on the output of the similarity classifier c, and hence this similarity classifier is not, by itself, capable of creating the strong "no false positive" results that we will provide below for our overall system; the similarity classifier must be combined with our other tests for verifying dense correspondences.

Our pipeline for object detection proceeds as follows: we assume that an initial detector finds detections in an image and proposes an object class O_i for each detection D. We then pass the detection and its corresponding proposed class to our verification system THEORETICALFLOWVERIFY(i, D) which returns True if it can verify that D contains an image of class O_i and False otherwise. Note that THEORETICALFLOWVERIFY may return False either because O_i is the wrong object class or because the algorithm is simply unable to verify the class of the object.

Our algorithm for THEORETICALFLOWVERIFY is described below. This algorithm is designed to return False for all false positive detections; in other words, THEORETICALFLOWVERIFY(i, D) will return False if $gt(D) \neq O_i$. The algorithm proceeds as follows: For any detection D and proposed object class O_i, THEORETICALFLOWVERIFY iterates over all images $m \in M_i$ in the dataset for object class O_i. It then compares each template image $I_{i,m}$ to the detection D using VERIFYMATCH, which either returns True if it can verify that O_i is the correct object class of D and False otherwise.

VERIFYMATCH proceeds as follows: First, for template image $I_{i,m}$, it performs "Similar Object Comparison": it checks whether there is another object $O_j \neq O_i$ in the dataset, and some template image n of object O_j, such that $I_{i,m}$ and $I_{j,n}$ both have a sufficiently similar appearance to D according to the similarity classifier c. In such a case, we cannot verify whether D is an object of class O_i or O_j, so the method returns False.

Next, VERIFYMATCH computes a set of dense pixel-wise correspondences $\hat{T}$ (also referred to as "Flow") between template image $I_{i,m}$ and the detection D. These correspondences $\hat{T}$ represent a 2D mapping from pixels of $I_{i,m}$ to the pixels of D, i.e. $\hat{T} : (u_1, v_1) \rightarrow (u_2, v_2)$. Note that we make no assumptions about $\hat{T}$; if our method for estimating correspondences is not very accurate, then this will reduce the recall of our system but we will still have no false positives, since our system will return False for such examples for all object classes.

Finally, VERIFYMATCH checks whether $\hat{T}$ corresponds to a rigid object transformation and returns False otherwise.

Here we state the main theorem of our approach:

Theorem 1. (No False Positive Theorem)
Under assumptions 1 and 2, THEORETICALFLOWVERIFY does not produce any false positives. That is, the following statement always holds: THEORETICALFLOWVERIFY(i, D) returns False

Algorithm 1 Theoretical Flow Verifier

1: **procedure** VERIFYMATCH($I_{i,m}, D$) ▷ returns True if verification succeeds
2: // Test 1: Similar Object Comparison
3: **for** $j \in \mathcal{I} \backslash \{i\}$ **do**
4: **for** $n \in M_j$ **do**
5: **if** $c(I_{i,m}, D) < c(I_{j,n}, D) + \delta$ **then**
6: **return** False
7: // Test 2: Color Comparision
8: $\hat{T} \leftarrow F(I_{i,m}, D)$ ▷ Estimated flow
9: **if** $d(\hat{T}(I_{i,m}), D) > \gamma$ **then**
10: **return** False
11: // Test 3: Flow Rigidity
12: **if** $r(\hat{T}) < 1$ **then**
13: **return** False
14: **return** True
15: **procedure** THEORETICALFLOWVERIFY(i, D)
16: **for** $m \in M_i$ **do**
17: **if** VERIFYMATCH($I_{i,m}, D$) **then**
18: **return True**
19: **return False**

whenever $gt(D) \neq O_i$, i.e. whenever the ground-truth class for detection D is different from class O_i

Note that THEORETICALFLOWVERIFY(i, D) may sometimes return False even if i is the correct ground-truth class of D if it cannot verify this proposal. In such cases, THEORETICALFLOWVERIFY will return false for all object classes, meaning that we cannot verify the category of detection D using our approach. Still, the benefit of our approach is that, when THEORETICALFLOWVERIFY(i, D)) returns true, we can be assured that detection D is an object of class O_i.

Proof. We need to show that THEORETICALFLOWVERIFY(i, D) returns False whenever the ground-truth class of object D is not O_i. Note that from line 17, we need to prove that for any object class O_i that is not equal to the ground truth class, every template image $I_{i,m}$, for all $m \in M_i$ of object O_i needs to be rejected by VERIFYMATCH. That is, VERIFYMATCH($I_{i,m}, D$) = False, $\forall m \in M_i$, if $gt(D) \neq O_i$.

Assume that the ground-truth object class of D is O_j. We can partition M_i into three classes:

1. $M_{i,1} := \{m : \exists T \in T^R$ such that $d(T(I_{i,m}), D) \leq \gamma\}$.
2. $M_{i,2} := \{m \in M_i \mid d(\hat{T}(I_{i,m}), D) > \gamma\} \backslash M_{i,1}$.
3. $M_{i,3} := M_i \backslash (M_{i,1} \cup M_{i,2})$.

Now we show VERIFYMATCH($I_{i,m}, D$) = False, for each of the three cases above.

1. Case 1: $m \in M_{i,1}$. VERIFYMATCH will return False at line 6.

   Since $m \in M_{i,1}$, we know that $\exists T_i \in T^R$ such that $d(T_i(I_{i,m}), D) \leq \gamma\}$. In other words, $M_{i,1}$ is a set of images of object class O_i that can be described by a rigid transformation and a small lighting change applied to D (which is an object of class O_j). Thus there are two object classes, O_i and O_j, that both appear similar to detection D, although detection D is an object of class O_j.

   From assumption 1, $\exists n \in M_j, T_j \in T^R$ such that $d(T_j(I_{j,n}), D) \leq \gamma$. Let us define a new operator $T_{ij} = (T_j)^{-1} \circ T_i$, i.e. first apply T_i and then apply inverse of T_j. First, note that, if $T_i \in T^R$ and $T_j \in T^R$, then $T_{ij} \in T^R$ since the composition of these rigid transforms is a rigid transform. Secondly,

$$
\begin{aligned}
d(T_{ij}(I_{i,m}), I_{j,n}) &= d(T_j \circ T_{ij}(I_{i,m}), T_j(I_{j,n})) \\
&= d(T_i(I_{i,m}), T_j(I_{j,n})) \\
&\leq d(T_i(I_{i,m}), D) + d(T_j(I_{j,n}), D) \\
&\leq 2\gamma
\end{aligned}
$$

The first line holds by the permutation invariance property of d, since T_j is a permutation of pixels. The second line holds by definition of T_{ij}. The third line holds by triangle inequality of d. The last line holds because of the definition of m and T_i that $d(T_i(I_{i,m}), D) \leq \gamma$, and by definition of n and T_j that $d(T_j(I_{j,n}), D) \leq \gamma$.

By assumption 2, since $d(T_{ij}(I_{i,m}), I_{j,n}) \leq 2\gamma$, then $|c(I_{i,m}, D) - c(I_{j,n}, D)| < \delta$. This then implies that

$$
\begin{aligned}
c(I_{i,m}, D) &< c(I_{j,n}, D) + \delta \\
c(I_{i,m}, D) &< \max_n c(I_{j,n}, D) + \delta
\end{aligned}
$$

Hence, the conditional at line 5 will be true and so VERIFYMATCH$(I_{i,m}, D)$ will return False at line 6.

2. Case 2: $m \in M_{i,2}$. VERIFYMATCH will return False at line 10.

   By definition of $M_{i,2}$, we know that $d(\hat{T}(I_{i,m}), D) > \gamma$. Hence the conditional at line 9 will be true, so VERIFYMATCH$(I_{i,m}, D)$ will return False at line 10.

3. Case 3: $m \in M_{i,3}$. VERIFYMATCH will return False at line 13.

   If $m \in M_{i,3}$, then by definition, $m \notin M_{i,1}$ and $m \notin M_{i,2}$. Since $m \notin M_{i,1}$, then $\nexists T \in T^R$ such that $d(T(I_{i,m}), D) \leq \gamma$. Since $m \notin M_{i,2}$, then $d(\hat{T}(I_{i,m}), D) \leq \gamma$. Hence we know that $\hat{T} \notin T^R$, so $r(\hat{T}) < 1$. Hence the conditional at line 12 will be true, so VERIFYMATCH$(I_{i,m}, D)$ will return False at line 13.

Hence we have shown that $\forall i \neq j, \forall m \in M_i$, VERIFYMATCH$(I_{i,m}, D) =$ False.
This implies that THEORETICALFLOWVERIFY$(i, D) =$ False whenever $i \neq j$. □

B Network Architecture Details

We call our network for predicting dense pixel-wise correspondences FLOWMATCHNET. For every proposed detection, we pad the detection to square, crop it (which we refer to as the 'cropped scene image'), and feed it to FLOWMATCHNET. FLOWMATCHNET also takes an image from the dataset and computes a mapping from every pixel in the template image to some pixel in the cropped scene image.

We compute pixel correspondence from each template image of the object to the cropped scene image. We train a deep neural network using a modification of the FlowNetC neural network architecture [19]. Specifically for FlowNetC, a series of convolutional layers are applied separately to input images to extract features from each image. Then, a *cross-correlation* layer is used to combine features coming from each image branch, to compare feature in every location in image1 with every location in image2. However, FlowNetC was designed for optical flow, which is typically applied to consecutive frames of a video with the assumption that pixels have small displacements between consecutive frames. Therefore, in the original FlowNetC, cross-correlation is approximated by considering only a 21×21 pixel neighborhood window [19, 20].

However, we cannot make such a 'small displacement' assumption in our case because the object in the scene crop might have undergone a fairly large rotation and translation relative to the template image. Hence, we perform a full cross correlation between all pixel pairs in the scene and template images. Furthermore, flow prediction is traditionally made in terms of displacement vectors. Since a full cross-correlations is implemented as unrolling the entire 2D feature map, we reparameterize

Figure 4: (a), (b): Target and scene images from MS-COCO synthetic dataset that applies affine transformations. (c), (d): Target and scene images from BigBIRD-RGBD synthetic dataset that applies homography transformations.

flow in terms of global coordinates of pixels being mapped to in the second image. Hence we concatenate (x, y) pixel coordinates as additional feature maps, following [38].

C Training details of FLOWMATCHNET

Optical flow models are trained using synthetic datasets that have ground truth flow [19, 20]. Flownet based models have been shown to generalize well when trained on such synthetic datasets.

For training FLOWMATCHNET, we train our model successively on synthetic datasets with increasing difficulty –

1. *MS-COCO with affine transformations*: we take images from the MS-COCO dataset [36], crop an area around the object using its annotated bounding box to simualate detections from a base detector, and apply a random rotation and translation to create a simulated scene image. These transforations help the flow model to learn simple affine tranformations. Although the transformations are relatively simple, MS-COCO has a huge variety of objects; training on such a diverse set of objects can help the network to generalize to different objects types. Examples of transformations are in Figure 4.

2. *BigBIRD + RGBD scenes with homographies*: We take objects from the BigBIRD dataset [32], crop them out using segmentation masks, and paste them onto background scenes in the W-RGBD scenes dataset [31]. Since we want FLOWMATCHNET to generalize to novel objects, we exclude training on the 11 objects that are part of GMU Kitchens [30], our test set. We apply random homographies to the cropped images and blend them into scenes using Cut-Paste-Learn [35], while simulating randomized lighting conditions and image blurs. We limit the random homography to ensure that it is not too unrealistic and does not distort the object too much; specifically we ensure that the top-left corner of the bounding box is still the top-left corner after applying the homograph, the bottom-right corner is still at the bottom-right, and so on. Importantly, since these are controlled synthetic transformations, ground truth flow can be computed. We train on L2 loss for optical flow similar to [19, 20].

We first trained on the MS-COCO synthetic flow dataset for 1.6M iterations using Adam optimizer with a learning rate of 10^{-4} with a batch size of 2. We then finetuned on the BigBIRD-RGBD synthetic flow dataset for 50,500 iterations using Adam optimizer with a learning rate of 10^{-6} and batch size 2.

D Ablation Details

This section lists detailed results of our ablation analysis in terms of mAP and maximum precision for FLOWVERIFY, versus dropping each verification test one at a time. As shown in Table 3, FLOWVERIFY has the best performance in terms of maximum precision and mAP on GMU-Test, as well the second highest maximum precision on W-RGBD. Dropping FRIGIDITY results in the largest drop in maximum precision and mAP on both GMU-Test and W-RGBD. This confirms the importance of *flow rigidity* test as suggested by our theoretical framework.

	GMU-Test		W-RGBD	
Method	mAP	max Precision	mAP	max Precision
FLOWVERIFY	**0.422**	**0.939**	0.278	0.817
FLOWVERIFY-SIMOBJ	0.412	0.906	**0.316**	0.727
FLOWVERIFY-FRIGIDITY	0.394	0.808	0.235	0.700
FLOWVERIFY-FCOLOR	**0.422**	0.933	0.278	**0.819**
FLOWVERIFY-FPRECISION	0.415	0.913	0.280	0.811
FLOWVERIFY-FRECALL	0.413	0.887	0.277	0.753

Table 3: Ablation analysis.

E SIFTVERIFY

Here we describe the details of the verification tests used for SIFTVERIFY. For FPRECISION, we computes the proportion of matched pixels that lie inside the predicted bounding box. This test is often imprecise because SIFT matches are sparse, often as few as 2-3 matches. However, to estimate rigidity using the fundamental matrix, we need at least 8 matches. Hence, instead we simply compute the raw number of SIFT keypoint matches. Since SIFT matches are few in number, it is also not possible to estimate recall the way we do for FRECALL, so we omit that test as well. We find that these settings give the best performance for this baseline. We run grid-search to find the best thresholds for SIFTVERIFYon YCB following the same procedure as for FLOWVERIFY.

F Further Qualitative Analysis

In this supplement, we perform a comparative and qualitative analysis of our method FLOWVERIFY with the SIFTVERIFY baseline, to complement the quantitative analysis in the main paper. We will first show examples where SIFTVERIFY succeeds in filtering out false positives from a base detector and where it is successful in retaining true positives. Then we will analyze some failure cases of SIFTVERIFY, where our model is able to filter false positives and retain correct detections whereas SIFTVERIFYis not.

It is important to note that since we are interested in high-precision detection, we will analyze detections which were given a very high confidence score (> 0.99) by the original detector on the GMU Kitchens [30] test set. In order to improve detector performance in the high precision regime (the leftmost parts of the PR-curve), the system must be able to filter out high-confidence false positives whilst not rejecting too many true detections with high confidence.

As a reminder, FLOWVERIFY uses five tests – SIMOBJ, FCOLOR, FRIGIDITY, FPRECISION and FRECALL . In order to pass a test, there must exist atleast one viewpoint of the target object whose SIMOBJ, FCOLOR, FRIGIDITY, FPRECISION and FRECALL scores are all above their respective thresholds. The thresholds that were tuned on YCB-Video train for FCOLOR, FRIGIDITY, FPRECISION and FRECALL turn out to be 0.5, 0.9, 0.9, and 0.3 respectively, with parameters for SIMOBJ being $\eta_{iou} = 0.5$, $\eta_{ratio} = 0.0$.

Similarly, SIFT filters detections by computing the number of keypoint matches between the target image and cropped scene image. If the SMATCHES and SPRECISION are above their corresponding thresholds, the detection is deemed to be true by the SIFT baseline. When tuned on YCB-Video Train, the thresholds for SMATCHES and SPRECISION turns out to be 30 and 0.9 respectively.

F.1 Visualization Format

The visualization format is consistent across all Figure 5, Figure 6, Figure 7, and Figure 8 – the top image shows a visualization of matching by FLOWMATCHNET and filtering by FLOWVERIFY, whereas the bottom image shows matching using SIFTVERIFY. Each visualization consists of two images. On the left is a 'target image' – it is an image of the object being detected. If the detection passes the FLOWVERIFY or SIFT verification tests, the displayed viewpoint is the one that passes all tests and has the largest product of scores, which denotes the "best" viewpoint of the object that

is able to be matched by each of the methods. Otherwise, the canonical viewpoint of the object is displayed in the visualization.

At the top of each visualization are scores for tests associated with that method, as well as thresholds, in a *score/threshold* format. 20 randomly-chosen pixel-wise mappings are depicted. In the case of SIFT, sometimes there are less than 20 keypoint matches found. In such cases, all keypoint matches are depicted.

F.2 SIFT Successful Cases

In this section, we analyze cases where SIFT matching successfully filters false positives while retaining true detections.

F.2.1 False Positives rejected by SIFT and FLOWVERIFY

Figure 5 contains four examples that denote false detections that are successfully filtered out by both methods. In these examples , the homography based on flow computed by FLOWMATCHNET produces many outliers, and hence are rejected by the FRIGIDITY test. This happens because when the object are distinct and do not match, in which case the predicted flow can be arbitrary. SIFT also successfully rejects all these examples, as it produces very few (< 30) keypoint matches.

F.2.2 True Positives accepted by SIFT and FLOWVERIFY

Figure 6 contains fours examples that represent true detections successfully retained by both methods. In these examples, FLOWMATCHNETand SIFT matches seem nearly perfect. This section demonstrates that SIFT features can be effective in object matching, and our implementation of SIFT is a reasonably strong baseline. However, we show below that SIFT can also fail on many cases.

F.3 SIFT Unsuccessful Cases

In this section, we analyze cases where SIFT matching fails to filter false positives or retain high-confidence true detections. We illustrate how FLOWVERIFY successfully handles these cases, to highlight the strengths of our method over the SIFT baseline.

F.3.1 False Positives accepted by SIFT but rejected by FLOWVERIFY

In Figure 7, the four examples represent false detections that SIFT fails to filter out, but are successfully filtered out our method. In Figure 7(a), the target object is different from the object in the bounding box, but they have the same text in their logos. SIFT fails in this case since it can find enough keypoint matches just in the logo area. FLOWVERIFY handles this case successfully since it relies on dense pixelwise correspondence. As the object in the cropped scene is different from that in the target image, FLOWMATCHNET cannot find a rigid transformation that matches color for all pixels, thereby not passing the FRIGIDITY test.

Figure 7(b, c, d) illustrate another difference between dense and sparse pixel-wise matching. In these examples, the bounding boxes are incomplete and the cropped scene only contains a part of the target object. As SIFT only needs to match a few keypoints, even if small parts of two objects seem to match, it is still enough for SIFT to accept it as a correct detection. FLOWVERIFY combined with FLOWMATCHNET successfully rejects these as false detections since FLOWMATCHNET computes dense matches across the entire object, resulting in flow fields that have a low FRIGIDITY score.

F.3.2 True Positives rejected by SIFT but retained by FLOWVERIFY

Figure 8 contains four examples that denote true positives, which are successfully retained by FLOWVERIFY but are incorrectly *rejected* by SIFTVERIFY. In all examples (a, b, c, d) in Figure 8, FLOWMATCHNET produces good-quality pixel-wise correspondence, leading to high FCOLOR, FRIGIDITY, FPRECISION and FRECALL scores. However, for most examples, SIFT can produce very few keypoint matches.

Our experiments show that SIFT struggles to perform reliable keypoint matching in our setting where there can be drastic changes in viewpoints, lighting conditions and even occlusions. We

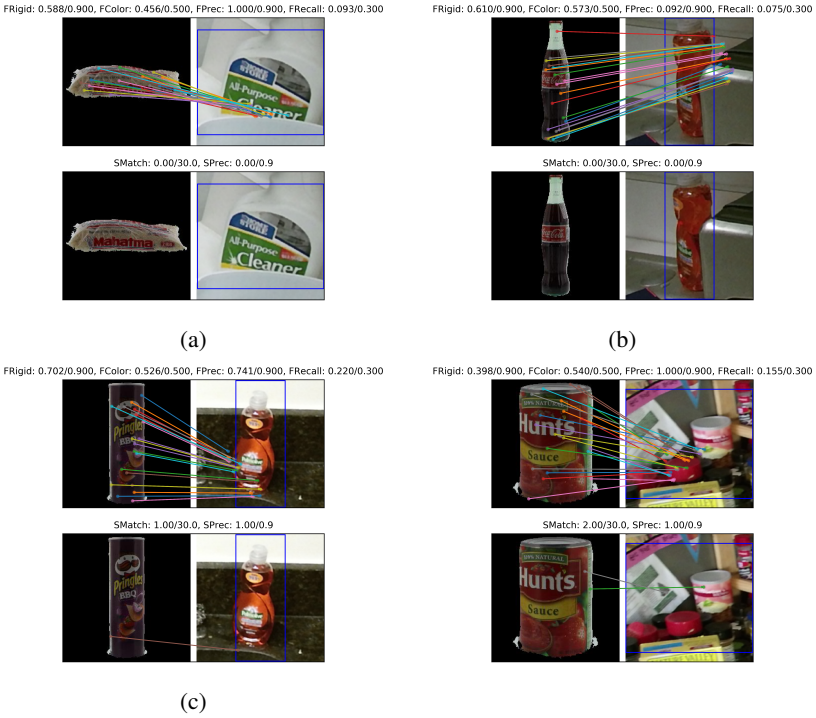


Figure 5: False detections which are successfully filtered out by both FLOWVERIFY and SIFTVERIFY. In each image, top: FLOWVERIFY, bottom: SIFTVERIFY.

show that in such scenarios, predicting a dense pixel-wise correspondence and designing subsequent verification tests can improve instance detectors towards high-precision and verifiable recognition.

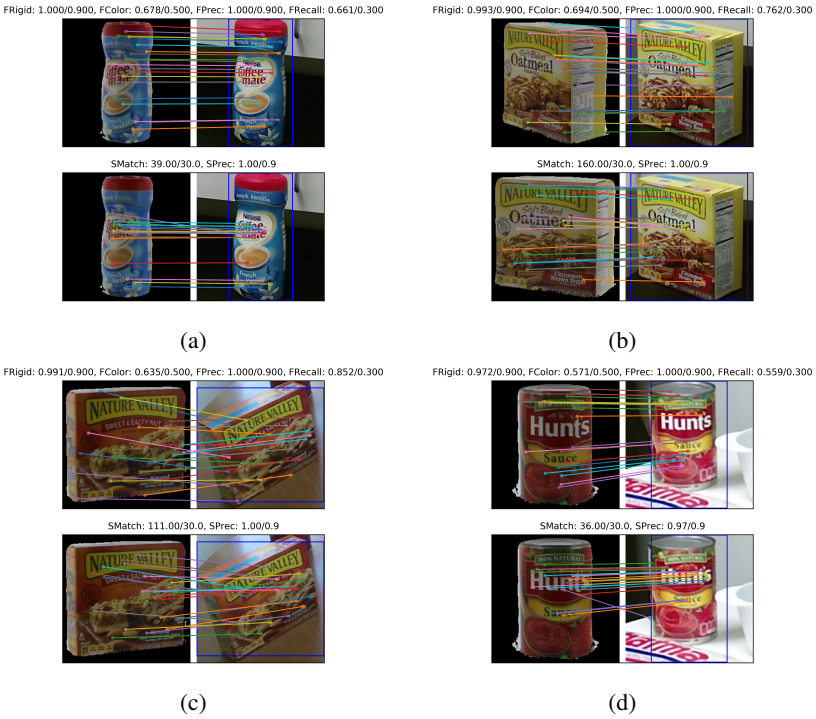


Figure 6: True detections that are successfully retained out by both FLOWVERIFY and SIFTVERIFY. In each image, top: FLOWVERIFY, bottom: SIFTVERIFY.

Figure 7: False detections which are incorrectly retained by SIFTVERIFY. In each image, top: FLOWVERIFY, bottom: SIFTVERIFY.

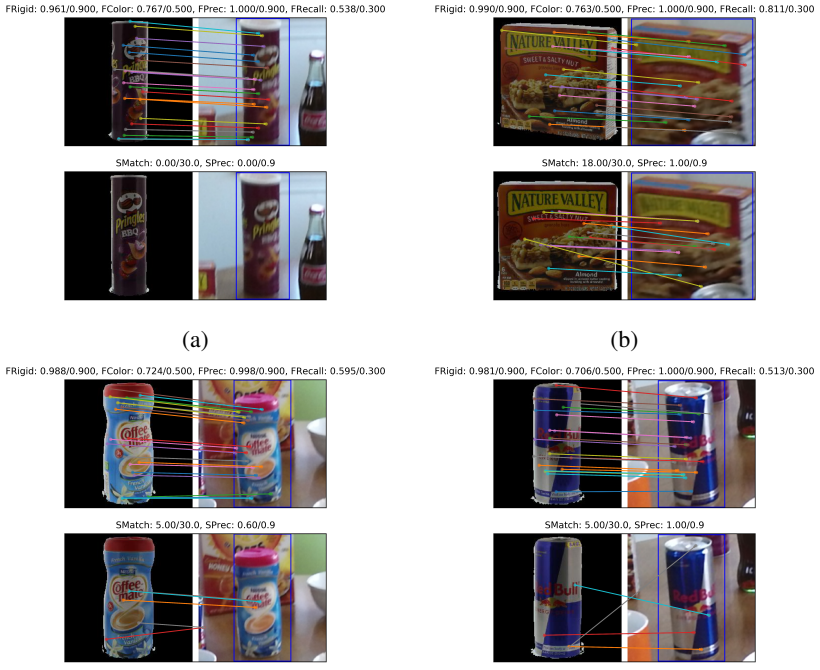


Figure 8: True detections that are incorrectly filtered by SIFTVERIFY. In each image, top: FLOWVERIFY, bottom: SIFTVERIFY.