Supplementary Material (Appendix) OS2D: One-Stage One-Shot Object Detection by Matching Anchor Features

A Data and evaluation

A.1 Retail products

Data. We used the GroZi-3.2k dataset [7] for retail product detection (680 images collected from 5 different stores) as development data. We could not use the original annotation of this dataset because it was often grouping multiple similar objects into a single bounding box and had no strict policy about what objects are of the same class and what are of different classes. Thus, we created new annotation where each object has an individual bounding box, and only identical objects belong to the same class. For each class, we selected a template class image either from the original annotation (if available) or from the internet by querying the product names. Importantly, each object was assigned the "difficult" flag when the human annotator could not assign a class without guessing. We ended up having 8921 objects of 1063 classes annotated.

Data splits. We created the splits of the dataset by selecting 185 classes and keeping the images with those classes away from training: 622 objects in 84 images. We will refer to this set as `val-new-cl`. The selected images also contained 518 objects of classes that appeared at training. We will refer to this set as `val-old-cl`. Throughout our experiments we used `val-new-cl` as our main validation set, and `val-old-cl` as a secondary validation showing performance on the training classes.

Extra test sets. To assess the generalization ability of our method, we collected extra test sets containing objects of new classes in the images taken in different conditions. The `dairy` set consists of 12 images with 786 objects of 166 classes of dairy products. The `paste-f` set consists of 91 images with 4861 objects of 259 classes of toothpaste and accompanying products. However, the `paste-f` set contains objects of all orientations, which is different from the training conditions. We also selected a subset `paste-v`, where all the objects with incorrect orientation are masked out with the "difficult" flag.

Table 3 of the main paper contains results of the OS2D methods and baselines on the test and validation subsets.

A.2 Additional dataset: INSTRE

In addition to the domain of retail products, we applied our methods to the INSTRE dataset [41], which was originally collected for large scale instance retrieval and has bounding box annotation for all objects. The dataset is considered hard due to occlusions and large variations in scales and rotations [14]. The dataset has

28,543 images and 200 object classes: 100 classes representing physical objects in the lab of the dataset creators, 100 classes collected on-line representing buildings, logos and common objects. We refer to the classes of the first and second types as INSTRE-S1 and INSTRE-S2, respectively.

The INSTRE dataset is used for evaluating retrieval systems, so does not have splits into train and test. We modify the evaluation protocol of Iscen et al. [14] who selected 5 images of each class as queries (correspond to class images in our terminology) by splitting the classes (with the corresponding images) of both INSTRE-S1 and INSTRE-S2 in the following proportion: 75% for training, 5% for validation and 20% for testing. In this setting, we can train and evaluate our method and the baselines. Table 4 of the main paper provides results on the two versions on the dataset. We report the results on the test sets of INSTRE-S1 and INSTRE-S2 after training of the training subsets of INSTRE-S1 and INSTRE-S2, respectively.

A.3 Evaluation metric.

In our experiments, we've used the standard Pascal VOC metric [5], which is the mean average precision at the intersection-over-union (IoU) threshold of 0.5 (we will refer to it as mAP). Importantly, this metric also supports the "difficult" flag of the annotation: it is used to exclude ground-truth objects and their detections when computing both recall and precision, which means that the method is neither penalized nor rewarded for detecting objects with this flag.

B Implementation details

B.1 TransformNet architecture

We follow Rocco et al. [35–37] and use the same architecture of TransformNet: ReLU, channelwise L2-normalization, conv2d with the kernel $7 \times 7 \times 225 \times 128$, batch norm, ReLU, conv2d with the kernel $5 \times 5 \times 128 \times 64$, batch norm, ReLU, conv2d with the kernel $5 \times 5 \times 64 \times P$. Here P is the number of parameters of the transformation, which equals 6 for the affine transformation. This network was designed for aligning two feature maps of size 15×15, i.e., $h^{\mathcal{T}} = w^{\mathcal{T}} = 15$ (corresponds to the image size 240×240 if using the features after the fourth block of ResNet).

Note that the network starts with ReLU, which corresponds to taking only positive correlations when building transformations (Rocco et al. [35] did not include this layer into TransformNet but applied it right after computing the correlations).

B.2 OS2D details

Implementation and hardware. We implemented the OS2D model based on the PyTorch library [28]. The models were both trained and tested on

GPUs. The hyperparameters for training were selected to fit the process on Nvidia GTX 1080 Ti. However evaluation of retail test sets required more device memory because of higher resolution, small objects and a large quantity of classes. We used Nvidia V100 devices for such runs.

Training. The OS2D models were trained with the SGD optimizer for 200k steps with the learning rate of 10^{-4}, weight decay of 10^{-4} and momentum of 0.9. We decreased the learning rate by a factor of 10 after 100k and 150k training iterations. We used the input image batch size of 4, cropped patches of size 600×600 and used at most 15 different labels per batch. Note that cropping patches of the correct size is effectively a version of random crop/scale data augmentation. We tried using more types of data augmentation, but none of them was effective.

When training all the models, we converted the switched layers of the feature extractor to the evaluation mode, i.e., did not estimate batch mean and variance. Keeping batchnorm in the training mode significantly degraded the performance. When training the V1 and V2 models we kept batchnorm of the transformation network in the training and evaluation modes, respectively.

We followed Rocco et al. [35–37] and trained TransformNet on only positive pairs. Technically we achieved this by computing two versions of the transformations at training – one with the full computational graph, another with the TransformNet parameters detached from the graph. The first version was used to train on positives, the second one – to train on negatives. We used this approach because when training transformations on negatives the networks started to ruin the transformation model by moving the transformation in random directions. On top of that, we often have very similar classes, and we still want them to be aligned properly to better compare the matched features.

When training all V1 models we initialized TransformNet to always output identity transformation by setting the weights of the last convolutional layer of TransformNet to 0 and biases to 0 or 1.

For the objective function, we use the margins $m_{\text{pos}} = 0.6$ and $m_{\text{neg}} = 0.5$ when the recognition scores were normalized to the segment $[-1, 1]$. To train the V1 models, we used the weight of the localization loss of 0.2. To fine-tune the V2 models, we turned the localization loss completely, i.e., set its weight to zero.

Detection. Before computing the final results, for all the methods we used the standard non-maximum suppression (NMS) with the IoU threshold of 0.3. Differently from the maskrcnn-benchmark [24], we did not do joint NMS for all the classes – it always degraded the performance.

At evaluation, we resized the class images with preserving their aspect ratio to have their product of dimensions equal to 240^2. For the input images, we used the image pyramid to deal with objects of different scales. We always use the pyramid of 7 levels: 0.5, 0.625, 0.8, 1, 1.2, 1.4, 1.6 times the dataset scale. For each dataset, we estimated its scale by computing and rounding the average object size. The GroZi-3.2k dataset was of scale 1280, the `dairy` dataset was of scale 3500, the `paste-v` dataset were also of scale 3500. However, the `paste-v` dataset had too many labels, so the largest image size did not fit into the GPU

memory, thus we reduced its scale to 2000 for all experiments with OS2D (the baselines were still run on the initial scale). Evaluation of an image at a particular scale, e.g., 0.5 * 3500 = 1750, means that we resize the input image such that its largest size equals the scale, e.g., 1750, before feeding it into the feature extractor (or objects detector for the baselines).

C Details of the baselines

In this section, we describe the details of the baselines that were important to improve their performance. Note that implementations of both baselines use open-source code, and we provide all the changes and launching scripts together with the OS2D code.[6]

C.1 Main baseline: detector + retrieval

For the detector, we used the maskrcnn-benchmark system [24]. We used the Faster R-CNN detectors [34] with the feature pyramid backbone [18] based on ResNet-50 and ResNet-101. We used the standard hyperparameters, but added multi-scale training and testing (supported by the library), which were improving results. The scales of images for both training and testing were set the same to the OS2D training regime.

For the retrieval system used on top of the detections, we used the open-source library by Radenović et al. [30, 31]. We used the trainable Generalized-Mean (GeM) pooling and end-to-end trainable whitening layers. For the training dataset, we used the class images as queries, annotated detections of the correct/incorrect classes as positives and negatives, respectively. We also randomly sampled 10 bounding boxes per training image and automatically labeled them as positive/negatives based on their IoU with annotated objects. In the training process, we used the standard setting with the contrastive loss, hard negative mining, Adam optimizer and learning rate schedule with an exponential decay. We resized all images (queries, positives and negatives) to have the maximal side equal to 240 (with preserving the aspect ratio).

At the testing stage, we used the same image pyramid as in OS2D for the detector and the multi-scale descriptor (3 scales) for retrieval.

C.2 CoAE one-shot detector

We compared our methods with the official implementation of the recent CoAE method of Hsieh et al. [12]. Their released models (trained on ImageNet) did not generalize well to our settings, so we reported only the results of the retrained models. For fair comparison with OS2D, we added multi-scale training and testing to the original code. Multi-scale training helped significantly, while multi-scale testing did not help at all. In training, we used the same number of iterations as for OS2D (the process converged well) and the same learning rate schedule (but different initial value).

[6] `https://github.com/aosokin/os2d`

D Evaluation in the ImageNet-based setup

In this section, we evaluate our methods on the setting proposed by Karlinsky et al. [17]. The test set is based on the images from 214 categories of the ImageNet-LOC dataset [39] and is organized in 500 episodes each containing 5 classes (5-way). Each class of an episode is represented by 10 random images with the class instances. In the 1-shot setting, each class additionally has one representative: an image with a bounding box around the class instance. The quality on each episode is measured by the average precision computed on the jointly sorted list of detections of different classes (positive/negative labels are assigned based on the IoU threshold 0.5). The overall quality is computed as the average AP over episodes. To distinguish this metric from mAP used in the rest of this paper (where AP is computed for each class independently and the mean is taken over classes), we refer to it as AP. Karlinsky et al. [17] released the exact episodic data and reported the AP of 56.9 without finetuning on each episode and 59.2 with finetuning. To train our methods without looking at the test classes, we retrained the ResNet101 backbone on the remaining 786 ImageNet classes using the standard PyTorch training script.[7] For the main baseline, we initialized both the detector and the retrieval system from this network and finetuned them on the images of ImageNet-LOC (the same training classes) for detection of all classes and image retrieval, respectively. We used exactly the same code and hyperparameters as selected for the Grozi32k dataset. The resulting method delivered the AP of **60.3** (better than the 1-shot methods of [17]), which confirms the strength of our baseline. The matching based methods were not competitive at all: the sliding window baseline and OS2D V2-init gave 15.8 and 21.8 AP, respectively. Training OS2D did not succeed and did not lead to any improvements. We interpret such a huge difference of results as a confirmation that the settings based on the standard detection datasets (e.g., ImageNet, PASCAL VOC, COCO) are very different from Grozi2k an INSTRE showcasing the difference between instance-level vs. semantic recognition.

E Additional qualitative results

In Figures 4, 5, 6, we present extra detection results, provided by an OS2D model. For the purposes of visualization, we've run these results through NMS over all classes. The detection threshold was set a bit lower, so one can also see highest scoring wrong detections.

[7] `https://github.com/pytorch/examples/tree/master/imagenet`

Fig. 4. Detection results on the `val-new-cl` subset of the GroZi-3.2k dataset

Fig. 5. Detection results on the `dairy` test set

Fig. 6. Detection results on the `paste-f` test set