Localizing Grouped Instances for Efficient Detection in Low-Resource Scenarios

Amélie Royer  
IST Austria  
aroyer@ist.ac.at

Christoph H. Lampert  
IST Austria  
chl@ist.ac.at

Abstract

State-of-the-art detection systems are generally evaluated on their ability to exhaustively retrieve objects densely distributed in the image, across a wide variety of appearances and semantic categories. Orthogonal to this, many real-life object detection applications, for example in remote sensing, instead require dealing with large images that contain only a few small objects of a single class, scattered heterogeneously across the space. In addition, they are often subject to strict computational constraints, such as limited battery capacity and computing power.

To tackle these more practical scenarios, we propose a novel flexible detection scheme that efficiently adapts to variable object sizes and densities: We rely on a sequence of detection stages, each of which has the ability to predict groups of objects as well as individuals. Similar to a detection cascade, this multi-stage architecture spares computational effort by discarding large irrelevant regions of the image early during the detection process. The ability to group objects provides further computational and memory savings, as it allows working with lower image resolutions in early stages, where groups are more easily detected than individuals, as they are more salient. We report experimental results on two aerial image datasets, and show that the proposed method is as accurate yet computationally more efficient than standard single-shot detectors, consistently across three different backbone architectures.

1. Introduction

As a core component of natural scene understanding, object detection in natural images has made remarkable progress in recent years through the adoption of deep convolutional networks. A driving force in this growth was the rise of large public benchmarks, such as PASCAL VOC [5] and MS COCO [15], which provide extensive bounding box annotations for objects in natural images across a large diversity of semantic categories and appearances.

However, many real-life detection problems exhibit drastically different data distributions and computational requirements, for which state-of-the-art detection systems are not well suited, as summarized in Figure 1. For example, object detection in aerial or satellite imagery often requires localizing objects of a single class, e.g., cars [37], houses [19] or swimming pools [31]. Similarly, in biomedical applications, only some specific objects are relevant, e.g. certain types of cells [35]. Moreover, input images in practical detection tasks are often of much higher resolution, yet contain small and sparsely distributed objects of interest, such that only a very limited fraction of pixels is actually relevant, while most academic benchmarks often contain more salient objects and cluttered scenes. Last but not least, detection speed is often at least as important as detection accuracy for practical applications. This is particularly apparent when models are meant to run on embedded devices, such as autonomous drones, which have limited computational resources and battery capacity.

In this work, we propose ODGI (Object Detection with Grouped Instances), a top-down detection scheme specifically designed for efficiently handling inhomogeneous object distributions, while preserving detection performance. Its key benefits and components are summarized as follows:

(i) a multi-stage pipeline, in which each stage selects only a few promising regions to be analyzed by the next stage, while discarding irrelevant image regions.

| Dataset | VEDAI [21] | SDD [29] | MS COCO [15] |
|---------|------------|----------|--------------|
| Avg. object size (fraction of image pixels) | small (0.113%) | small (0.159%) | large (14.96%) |
| Avg. empty cell ratio (on a 16x16 uniform grid) | 95.1% | 97.1% | 49.4% |
| Object distribution | Few, specific, classes. Sparse and heterogeneous object distribution | high object density across many and varied categories | robotics, artificial intelligence, HCO |
| Typical applications | remote sensing, traffic monitoring, agriculture, medical imaging | | |
| Resource constraints | imposed by hardware, e.g. limited battery, no GPU | generally none | |

Figure 1. Recent benchmarks and challenges highlight the task of detecting small objects in aerial views, in particular for real-life low-resource scenarios [21, 26, 38, 34, 26, 1]. The data distribution and computational constraints for such tasks often vastly differ from state-of-the-art benchmarks, for instance MS COCO [15].
(ii) Fast single-shot detectors augmented with the ability to identify groups of objects rather than just individual objects, thereby substantially reducing the number of regions that have to be considered.

(iii) ODGI reaches similar accuracies than ordinary single-shot detectors while operating at lower resolution because groups of objects are generally larger and easier to detect than individual objects. This allows for a further reduction of computational requirements.

We present the proposed method, ODGI, and its training procedure in Section 3. We then report main quantitative results as well as several ablation experiments in Section 4.

2. Related work

Cascaded object detection. A popular approach to object detection consists in extracting numerous region proposals and then classifying them as one of the object categories of interest. This includes models such as RFCN [3], RCNN and variants [7, 8, 2], or SPPNet [9]. Proposal-based methods are very effective and can handle inhomogeneously distributed objects, but are usually too slow for real-time usage, due to the large amount of proposals generated. Furthermore, with the exception of [25], the proposals are generally class-independent, which makes these methods more suitable for general scene understanding tasks, where one is interested in a wide variety of classes. When targeting a specific object category, class-independent proposals are wasteful, as most proposal regions are irrelevant to the task.

Single-shot object detection and Multi-scale pyramids. In contrast, single-shot detectors, such as SSD [17], or YOLO [22, 23, 24], split the image into a regular grid of regions and predict object bounding boxes in each grid cell. These single-shot detectors are efficient and can be made fast enough for real-time operation, but only provide a good speed-versus-accuracy trade-off when the objects of interest are distributed homogeneously on the grid. In fact, the grid size has to be chosen with worst case scenarios in mind: in order to identify all objects, the grid resolution has to be fine enough to capture all objects even in image regions with high object density, which might rarely occur, leading to numerous empty cells. Furthermore, the number of operations scales quadratically with the grid size, hence precise detection of individual small objects in dense clusters is often mutually exclusive with fast operation. Recent work [16, 30, 20, 17, 18, 36] proposes to additionally exploit multi-scale feature pyramids to better detect objects across varying scales. This helps mitigate the aforementioned problem but does not suppress it, and, in fact, these models are still better tailored for dense object detection.

Orthogonal to this, ODGI focuses on making the best of the given input resolution and resources and instead resort to grouping objects when individual small instances are too hard to detect, following the paradigms that “coarse predictions are better than none”. These groups are then refined in subsequent stages if necessary for the task at hand.

Speed versus accuracy trade-off. Both designs involve intrinsic speed-versus-accuracy trade-offs, see for instance [10] for a deeper discussion, that make neither of them entirely satisfactory for real-world challenges, such as controlling an autonomous drone [38], localizing all objects of a certain type in aerial imagery [1] or efficiently detecting spatial arrangements of many small objects [32].

Our proposed method, ODGI, falls into neither of these two designs, but rather combines the strength of both in a flexible multi-stage pipeline: It identifies a small number of specific regions of interest, which can also be interpreted as a form of proposals, thereby concentrating most of its computations on important regions. Despite the sequential nature of the pipeline, each individual prediction stage is based on a coarse, low resolution, grid, and thus very efficient. ODGI’s design resembles classical detection cascades [14, 27, 33], but differs from them in that it does not sequentially refine classification decisions for individual boxes but rather refines the actual region coordinates. As such, it is conceptually similar to techniques based on branch-and-bound [12, 13], or on region selection by reinforcement learning [6]. Nonetheless, it strongly differs from these on a technical level as it only requires minor modifications of existing object detectors and can be trained with standard backpropagation instead of discrete optimization or reinforcement learning. Additionally, ODGI generates meaningful groups of objects as intermediate representations, which can potentially be useful for other visual tasks. For example, it was argued in [28] that recurring group structures can facilitate the detection of individual objects in complex scenes. Currently, however, we only make use of the fact that groups are visually more salient and easier to detect than individuals, especially at low image resolution.

3. ODGI: Detection with Grouped Instances

In Section 3.1 we introduce the proposed multi-stage architecture and the notion of group of objects. We then detail the training and evaluation procedures in Section 3.2.

3.1. Proposed architecture

We design ODGI as a multi-stage detection architecture $S \circ \cdots \circ \phi_1$, $S > 1$. Each stage $\phi_s$ is a detection network, whose outputs can either be individual objects or groups of objects. In the latter case, the predicted bounding box defines a relevant image subregion, for which detections can be refined by feeding it as input to the next stage. To compare the model with standard detection systems, we also constrain the last stage to only output individual objects.
**Grouped instances for detection.** We design each stage as a lightweight neural network that performs fast object detection. In our experiments, we build on standard single-shot detectors such as YOLO [22] or SSD [17]. More precisely, $\phi_s$ consists of a fully-convolutional network with output map $[I, J]$ directly proportional to the input image resolution. For each of the $I \times J$ cells in this uniform grid, the model predicts bounding boxes characterized by four coordinates – the box center $(x, y)$, its width $w$ and height $h$, and a predicted confidence score $c \in [0, 1]$. Following common practice [22, 23, 17], we express the width and height as a fraction of the total image width and height, while the coordinates of the center are parameterized relatively to the cell it is linked to. The confidence score $c$ is used for ranking the bounding boxes at inference time.

For intermediate stages $s \leq S - 1$, we further incorporate the two following characteristics: First, we augment each predicted box with a binary group flag, $g$, as well as two real-valued offset values $(o_w, o_h)$: The flag indicates whether the detector considers the prediction to be a single object, $g = 0$, or a group of objects, $g = 1$. The offset values are used to appropriately rescale the stage outputs which are then passed on to subsequent stages. Second, we design the intermediate stages to predict one bounding box per cell. This choice provides us with an intuitive definition of groups, which automatically adapts itself to the input image resolution without introducing additional hyperparameters: If the model resolution $[I, J]$ is fine enough, there is at most one individual object per cell, in which case the problem reduces to standard object detection. Otherwise, if a cell is densely occupied, then the model resorts to predicting one group enclosing the relevant objects. We provide further details on the group training process in Section 3.2.

**Multi-stage pipeline.** An overview of ODGI’s multi-stage prediction pipeline is given in Figure 2.

Each intermediate stage takes as inputs the outputs of the previous stage, which are processed to produce image regions in the following way: Let $B$ be a bounding box predicted at stage $\phi_s$, with confidence $c$ and binary group flag $g$. We distinguish three possibilities: (i) the box can be discarded, (ii) it can be accepted as an individual object prediction, or (iii) it can be passed on to the next stage for further refinement. This decision is made based on two confidence thresholds, $\tau_{\text{low}}$ and $\tau_{\text{high}}$, leading to one of the three following actions:

1. If $c \leq \tau_{\text{low}}$: The box $B$ is discarded.
2. If $c > \tau_{\text{high}}$ and $g = 0$: The box $B$ is considered a strong individual object candidate: we make it “exit” the pipeline and directly propagate it to the last stage’s output as it is. We denote the set of such boxes as $B_s$. If $c > \tau_{\text{low}}$ and $g = 1$: The box $B$ is either a group or an individual with medium confidence and is a candidate for refinement.

After this filtering step, we apply non-maximum suppression (NMS) with threshold $\gamma_{\text{nms}}$ to the set of refinement candidates, in order to obtain (at most) $\gamma_s$ boxes with high confidence and little overlap. The resulting $\gamma_s$ bounding boxes are then processed to build the image regions that will be passed on to the next stage by multiplying each box’s width and height by $1/o_w$ and $1/o_h$, respectively, where $o_w$ and $o_h$ are the offset values learned by the detector. This rescaling step ensures that the extracted patches cover the relevant region well enough, and compensates for the fact that the detectors are trained to exactly predict ground-truth coordinates, rather than fully enclose them, hence sometimes underestimate the extent of the relevant region. The resulting rescaled rectangular regions are extracted from the input image and passed on as inputs to the next stage. The final output of ODGI is the combination of object boxes predicted in the last stage, $\phi_S$, as well as the kept-back outputs from previous stages: $B_1 \ldots B_{S-1}$.

The above patch extraction procedure can be tuned via four hyperparameters: $\tau_{\text{low}}, \tau_{\text{high}}, \gamma_{\text{nms}}, \gamma_s$. At training time, we allow as many boxes to pass as the memory budget allows. For our experiments, this was $\gamma_s^{\text{train}} = 10$. We also do not use any of the aforementioned filtering during training, nor thresholding ($\tau_{\text{low}} = 0, \tau_{\text{high}} = 1$) nor NMS...
(\gamma_{\text{train}} = 1), because both negative and positive patches can
be useful for training subsequent stages. For test-time pred-
iction we use a held-out validation set to determine their
optimal values, as described in Section 4.2. Moreover, these
hyperparameters can be easily changed on the fly, without
retraining. This allows the model to easily adapt to changes
of the input data characteristics, or to make better use of an
increased or reduced computational budget for instance.

**Number of stages.** Appending an additional refinement
stage benefits the speed-vs-accuracy trade-off fit when the
following two criteria are met: First, a low number of
non empty cells; This correlates to the number of extracted
crops, thus to the number of feed-forward passes of subse-
quently stages. Second, a small average group size: Smaller
extracted regions lead to increased resolution once rescaled
to the input size of the next stage, making the detection task
which is fed to subsequent stages effectively easier.

From the statistics reported in Table 1, we observe that
for classical benchmarks such as MS-COCO, using only
one stage suffices as groups are often dense and cover large
portions of the image: In that case, ODGI collapses to using
a single-shot detector, such as [22, 17]. In contrast, datasets
of aerial views such as VEDAI [21] or SDD [26] contain
small-sized group structures in large sparse areas. This is
a typical scenario where the proposed refinement stages on
groups improve the speed-accuracy trade-off. We find that
for the datasets used in our experiments \( S = 2 \) is suffi-
cient, as regions extracted by the first stage typically exhibit
a dense distribution of large objects. We expect the case
\( S > 2 \) to be beneficial for very large, e.g. gigapixel images,
but leave its study for future work. Nonetheless, extending
the model to this case should be straightforward: This
would introduce additional hyperparameters as we have to
tune the number of boxes \( \gamma_s \) for each stage; However, as
we will see in the next section, these parameters have little
impact on training and can be easily tuned at test time.

**3.2. Training the model**

We train each ODGI stage independently, using a com-
ination of three loss terms that we optimize with standard
backpropagation (note that in the last stage of the pipeline,
we will see in the next section, these parameters have little
impact on training and can be easily tuned at test time).

\[
\mathcal{L}_{\text{ODGI}} = \mathcal{L}_{\text{groups}} + \mathcal{L}_{\text{coords}} + \mathcal{L}_{\text{offsets}} \tag{1}
\]

\( \mathcal{L}_{\text{coords}} \) is a standard mean squares regression loss on the
predicted coordinates and confidence scores, as described
for instance in [22, 17]. The additional two terms are part of
our contribution: The group loss, \( \mathcal{L}_{\text{groups}} \), drives the model
to classify outputs as individuals or groups, and the offsets
loss, \( \mathcal{L}_{\text{offsets}} \), encourages better coverage of the extracted
regions. The rest of this section is dedicated to formally
defining each loss term as well as explaining how we obtain
ground-truth coordinates for group bounding boxes.

**Group loss.** Let \( b = b_n = 1 \ldots N \) be the original ground-truth
individual bounding boxes. We define \( A^{ij}(n) \) as an indica-
tor which takes value 1 if \( b_i \) ground-truth box \( b_n \) is assigned
to output cell \((i, j)\) and 0 otherwise:

\[
A^{ij}(n) = \| b_n \cap cell_{ij} > 0 \|, \quad \text{with } \| x \| = 1 \text{ if } x, \text{ else } 0 \tag{2}
\]

For the model to predict groups of objects, we should in
principle consider all the unions of subsets of \( b \) as potential
targets. However, we defined our intermediate detectors to
predict only one bounding box per cell by design, which
allows us to avoid this combinatorial problem. Formally, let
\( B^{ij} \) be the predictor associated to cell \((i, j)\). We define its
target ground-truth coordinates \( \bar{B}^{ij} \) and group flag \( \bar{g}^{ij} \) as:

\[
\bar{B}^{ij} = \bigcup_{n | A^{ij}(n) = 1} b_n \tag{3}
\]

\[
\bar{g}^{ij} = \| \# \{n | A^{ij}(n) = 1 \} > 1 \|, \tag{4}
\]

with \( \bigcup \) denoting the minimum enclosing bounding box of a
set. We define \( \mathcal{L}_{\text{groups}} \) as a binary classification objective:

\[
\mathcal{L}_{\text{groups}} = -\sum_{i,j} A^{ij}(\bar{g}^{ij} \log(g^{ij})
+ (1 - \bar{g}^{ij}) \log(1 - g^{ij})) \tag{5}
\]

where \( A^{ij} = \| \sum_n A^{ij}(n) > 0 \| \) denotes whether cell
\((i, j)\) is empty or not. In summary, we build ground-truth
\( \bar{B}^{ij} \) and \( \bar{g}^{ij} \) as follows: For each cell \((i, j)\), we build the
set \( G^{ij} \) which ground-truth boxes \( b_n \) of ground-truth boxes
it intersects with. If the set is non empty and only a sin-
gle object box, \( b \), falls into this cell, we set \( \bar{B}^{ij} = b \) and
\( \bar{g}^{ij} = 0 \). Otherwise, \( |G^{ij}| > 1 \) and we define \( B^{ij} \) as
the union of bounding boxes in \( G^{ij} \) and set \( \bar{g}^{ij} = 1 \). In par-
cular, this procedure automatically adapts to the resolution
\([I, J]\) in a data-driven way, and can be implemented as a
pre-processing step, thus does not produce any overhead at
training time.

**Coordinates loss.** Following the definition of target bound-
ing boxes \( B^{ij} \) in (3), we define the coordinates loss as a
standard regression objective on the box coordinates and
confidences, similarly to existing detectors [8, 7, 17, 4, 22].

\[
\mathcal{L}_{\text{coords}} = \sum_{i,j} A^{ij}(\| B^{ij} - \bar{B}^{ij} \|^2 + \omega_{\text{conf}} \| c^{ij} - \bar{c}^{ij} \|^2
+ \omega_{\text{obj}} \sum_{i,j} (1 - A^{ij}) (c^{ij})^2) \tag{6}
\]

\[
\bar{c}^{ij} = \setdiff_{\bigcup_{i,j} B^{ij}} (B^{ij}) = \frac{|B^{ij} \cap \bar{B}^{ij}|}{|B^{ij} \cup \bar{B}^{ij}|} \tag{7}
\]

The first two terms are ordinary least squares regression ob-
jectives between the predicted coordinates and confidence
scores and their respective assigned ground-truth. The
ground-truth for the confidence score is defined as the inter-
section over union (IoU) between the corresponding pre-
diction and its assigned target. Finally, the last term in the
sum is a weighted penalty term to push confidence scores
for empty cells towards zero. In practice, we use the same
weights as in [22], i.e. $\omega_{\text{conf}} = 5$ and $\omega_{\text{no-obj}} = 1$.

Offsets loss. In intermediate stages, ODGI predicts offset
values for each box, $o_w$ and $o_h$, that are used to rescale the
region of interest when it is passed as input to the next stage,
as described in Section 3.1. The corresponding predictors
are trained using the following offsets loss:

$$
L_{\text{offsets}} = \sum_{i,j} A_{ij} \left[ (o_w - \tilde{o}_w(B_{ij}, \hat{B}_{ij}))^2 + (o_h - \tilde{o}_h(B_{ij}, \hat{B}_{ij}))^2 \right].
$$

(8)

The target values, $\tilde{o}_w(B_{ij}, \hat{B}_{ij})$ and $\tilde{o}_h(B_{ij}, \hat{B}_{ij})$, for vertical and horizontal offsets, are determined as follows: First, let $\alpha$ denote the center y-coordinate and $h$ the height. Ideally, the vertical offset should cause the rescaled version of $\hat{B}_{ij}$ to encompass both the original $B_{ij}$ and its assigned ground-truth box $B^{\text{ij}}$ with a certain margin $\delta$, which we set to half the average object size ($\delta = 0.0025$). Formally:

$$
h_{\text{scaled}}(B, \hat{B}) = \max(\lfloor (\alpha(B) + h(\hat{B})/2 + \delta - \alpha(B)) \rfloor, |(\alpha(B) - h(\hat{B})/2 - \delta - \alpha(B))|)
$$

$$
\tilde{o}_h(B_{ij}, \hat{B}_{ij}) = \max(1, h(B_{ij})/h_{\text{scaled}}(B^{\text{ij}}, \hat{B}_{ij}))
$$

(9)

For the horizontal offset, we do the analogous construction using the $B^{\text{ij}}$’s center x-coordinate and its width instead.

Evaluation metrics. We quantitatively evaluate the ODGI
pipeline as a standard object detector. Following the com-
mon protocol from PASCAL VOC 2010 and later chal-
enges [5], we sort the list of predicted boxes in decreasing
order of confidence score and compute the average pre-
cision (mAP) respectively to the ground-truth, at the IoU
cut-offs of 0.5 (standard) and 0.75 (more precise). In line
with our target scenario of single-class object detection, we
ignore class information in experiments and focus on raw
detection. Class labels could easily be added, either on the
level of individual box detections, or as a post-processing
classification operation, which we leave for future work.

Multi-stage training. By design, the inputs of stage $s$ are
obtained from the outputs of stage $s - 1$. However it is
cumbersome to wait for each stage to be fully trained before
starting to train the next one. In practice we notice that even
after only a few epochs, the top-scoring predictions of in-
termediate detectors often detect image regions that can be
useful for the subsequent stages, thus we propose the follow-
ing training procedure: After $n_e = 3$ epochs of training
the first stage, we start training the second, querying new in-
puts from a queue fed by the outputs of the first stage. This
allows us to jointly and efficiently train the two stages, and
this delayed training scheme works well in practice.

4. Experiments

We report experiments on two aerial views datasets: VEDAI [21] contains 1268 aerial views of countryside and



city roads for vehicle detection. Images are 1024x1024 pix-
els and contain on average 2.96 objects of interest. We per-
form 10-fold cross validation, as in [21]. For each run, we
use 8 folds for training, one for validation and one for test-
ing. All reported metrics are averaged over the 10 runs. Our
second benchmark, SDD [26], contains drone videos taken
at different locations with bounding box annotations of road
users. To reduce redundancy, we extract still images ev-
evry 40 frames, which we then pad and resize to 1024x1024
pixels to compensate for different aspect ratios. For each
location, we perform a random train/val/test split with ra-
tios 70%/5%/25%, resulting in total in 9163, 651 and 3281
images respectively. On average, the training set contains
12.07 annotated objects per image. SDD is overall much
more challenging than VEDAI: at full resolution, objects are
small and hard to detect, even to the human eye.

We consider three common backbone networks for
ODGI and baselines: tiny, a simple 7-layer fully con-
volutional network based on the tiny-YOLO architecture,
yolo, a VGG-like network similar to the one used in
YOLOv2 [22] and finally MobileNet [29], which is for
instance used in SSD Lite [17]. More specifically, on the
VEDAI dataset, we train a standard tiny-yolov2 detec-
ctor as baseline and compare it to ODGI-teeny-tiny (ODGI-
tt), which refers to two-stage ODGI with tiny backbones.
For SDD, objects are much harder to detect, thus we use a
stronger YOLO V2 model as baseline. We compare this to
ODGI-teeny-tiny as above as well as a stronger variant, ODGI-
yolo-tiny (ODGI-yt), in which $\phi_1$ is based on the yolo
backbones and $\phi_2$ on tiny. Finally we also experiment
with the lightweight MobileNet architecture as baseline
backbones, with depth multipliers 1 and 0.35. The cor-
responding ODGI models are denoted as ODGI-100-35 and
ODGI-35-35. All models are trained and evaluated at var-
ious resolutions to investigate different grouping scenarios.
In all cases, the detector grid size scales linearly with the
image resolution, because of the fully convolutional net-
work structures, ranging from a $32 \times 32$ grid for 1024px
inputs to $2 \times 2$ for 64px.

We implement all models in Tensorflow and train with the
Adam optimizer [11] and learning rate 1e-3. To facilitate
reproducibility, we make our code publicly available.

\footnote{Github repository, \url{https://github.com/ameroyer/ODGI}}
4.1. Main results

To benchmark detection accuracy, we evaluate the average precision (MAP) for the proposed ODGI and baselines. As is often done, we also apply non-maximum suppression to the final predictions, with IoU threshold of 0.5 and no limit on the number of outputs, to remove near duplicates for all methods. Besides retrieval performance, we assess the computational and memory resource requirements of the different methods: We record the number of boxes predicted by each model, and measure the average runtime of our implementation for one forward pass on a single image. As reference hardware, we use a server with 2.2 GHz Intel Xeon processor (short: CPU) in single-threaded mode. Additional timing experiments on weaker and stronger hardware, as well as a description of how we pick ODGI’s test-time hyperparameters can be found in Section 4.2.

We report experiment results in Figure 3 (see Table 1 for exact numbers). We find that the proposed method improves over standard single-shot detectors in two ways: First, when comparing models with similar accuracies, ODGI generally requires fewer evaluated boxes and shorter runtimes, and often lower input image resolution. In fact, only a few relevant regions are passed to the second stage, at a smaller input resolution, hence they incur a small computational cost, yet the ability to selectively refine the boxes can substantially improve detection. Second, for any given input resolution, ODGI’s refinement cascade generally improves detection retrieval, in particular at lower resolutions, e.g. 256px: In fact, ODGI’s first stage can be kept efficient and operate at low resolution, because the regions it extracts do not have to be very precise. Nonetheless, the regions selected in the first stage form an easy-to-solve detection task for the second stage (see for instance Figure 4 (d)), which leads to more precise detections after refinement. This also motivates our choice of mixing backbones, e.g. using ODGI-yolo-tiny, as detection in stage 2 is usually much easier.

![Figure 3](image.png)

---

**Table 1.** MAP and timing results on the VEDAI and SDD datasets for the model described in Section 4. The results for ODGI models are reported with $\gamma_1$ chosen as described in Section 4.2.
Table 1. We also report the total number of pixels processed per feed-forward pass for the same models and baselines as in Hyperparameters.

Table 2. Additional timing results. Time is indicated in seconds for a Raspberry Pi (Raspi), and in milliseconds for an Nvidia GTX 1080Ti graphics card (GPU). \( \# \text{pixels} \) is the total number of pixels processed and \( \# \text{parameters} \) the number of model parameters.

4.2. Additional Experiments

Runtime. Absolute runtime values always depend on several factors, in particular the software implementation and hardware. In our case, software-related differences are not an issue, as all models rely on the same core backbone implementations. To analyze the effect of hardware, we performed additional experiments on weaker hardware, a Raspberry Pi 3 Model B with 1.2 GHz ARMv7 CPU (Raspi), as well as stronger hardware, an Nvidia GTX 1080Ti graphics card (GPU). Table 2 shows the resulting runtimes of one feed-forward pass for the same models and baselines as in Table 1. We also report the total number of pixels processed by each method, \textit{i.e.} that have to be stored in memory during one feed-forward pass, as well as the number of parameters.

The main observations of the previous section again hold: On the Raspberry Pi, timing ratios are roughly the same as on the Intel CPU, only the absolute scale changes. The differences are smaller on GPU, but ODGI is still faster than the baselines in most cases at similar accuracy levels. Note that for the application scenario we target, the GPU timings are the least representative, as systems operating under resource constraints typically cannot afford the usage of a 250W graphics card (for comparison, the Raspberry Pi has a power consumption of approximately 1.2W).

Hyperparameters. As can be seen in Figure 3, a higher number of crops, \( \gamma_1 \text{test} \), improves detection, but comes at a higher computational cost. Nonetheless, ODGI appears to have a better accuracy-speed ratio for most values of \( \gamma_1 \text{test} \). For practical purposes, we suggest to choose \( \gamma_1 \text{test} \) based on how many patches are effectively used for detection. We define the occupancy rate of a crop as the sum of the intersection ratios of ground-truth boxes that appear in this crop. We then say a crop is relevant if it has a non-zero occupancy rate, \textit{i.e.} it contains objects of interest: For instance, at input resolution 512px on VEDAI's validation set, we obtain an average of 2.33 relevant crops, hence we set \( \gamma_1 \text{test} = 3 \). The same analysis on SDD yields \( \gamma_1 \text{test} = 6 \).

| SDD | \( \gamma_1 = 1 \) | \( \gamma_1 = 3 \) | \( \gamma_1 = 5 \) | \( \gamma_1 = 10 \) |
| --- | --- | --- | --- | --- |
| ODGI-n 512-256 | 0.245 | 0.361 | 0.415 | 0.457 |
| no groups | 0.225 | 0.321 | 0.380 | 0.438 |
| fixed offsets | 0.199 | 0.136 | 0.246 | 0.244 |
| no offsets | 0.127 | 0.127 | 0.125 | 0.122 |
| ODGI-n 256-128 | 0.128 | 0.243 | 0.293 | 0.331 |
| no groups | 0.122 | 0.229 | 0.282 | 0.326 |
| fixed offsets | 0.088 | 0.136 | 0.150 | 0.154 |
| no offsets | 0.030 | 0.040 | 0.040 | 0.040 |

Three additional hyperparameters influence ODGI's behavior: \( \tau_{\text{test}} \), \( \tau_{\text{high}} \), and \( \tau_{\text{nms}} \), all of which appear in the patch extraction pipeline. For a range of \( \gamma_1 \in [1, 10] \), and for each input resolution, we perform a parameter sweep on the held-out validation set over the ranges \( \tau_{\text{test}} \in \{0, 0.1, 0.2, 0.3, 0.4\} \), \( \tau_{\text{high}} \in \{0.6, 0.7, 0.8, 0.9, 1.0\} \), and \( \tau_{\text{nms}} \in \{0.25, 0.5, 0.75\} \). Note that network training is independent from these parameters as discussed in Section 3.1. Therefore the sweep can be done efficiently using pretrained \( \phi_1 \) and \( \phi_2 \), changing only the patch extraction process. We report full results of this validation process in the supplemental material. The main observations are as follows:

(i) \( \tau_{\text{test}} \) is usually in \([0, 0.1]\). This indicates that the low confidence patches are generally true negatives that need not be filtered out. (ii) \( \tau_{\text{high}} \in [0.8, 0.9] \) for VEDAI and \( \tau_{\text{high}} \in [0.6, 0.7] \) for SDD. This reflects intrinsic properties of each dataset: VEDAI images contain only few objects which are easily covered by the extracted crops. It is always beneficial to refine these predictions, even when they are individuals with high confidence, hence a high value of \( \tau_{\text{high}} \). In contrast, on the more challenging SDD, ODGI more often uses the shortcut for confident individuals in stage 1, in order to focus the refinement stage on groups and lower-confidence individuals which can benefit more. (iii) \( \tau_{\text{nms}} \) is usually equal to 0.25, which encourages non-overlapping patches and reduces the number of redundant predictions.
Figure 4. Qualitative results for ODGI. No filtering step was applied here, but for readability we only display boxes predicted with confidence at least 0.5. Best seen on PDF with zoom. Additional figures are provided in the supplemental material.

4.3. Ablation study

In this section we briefly report on ablation experiments that highlight the influence of the proposed contributions. Detailed results are provided in the supplemental material.

**Memory requirements.** ODGI stages are applied consequently, hence only one network needs to live in memory at a time. However having independent networks for each stage can still be prohibitory when working with very large backbones, hence we also study a variant of ODGI where weights are shared across stages. While this reduces the number of model parameters, we find that it can significantly hurt detection accuracy in our settings. A likely explanation is that the data distribution in stage 1 and stage 2 are drastically different in terms of object resolution and distribution, effectively causing a domain shift.

**Groups.** We compare ODGI with a variant without group information: we drop the loss term $L_{\text{groups}}$ in (1) and ignore group flags in the transition between stages. Table 3 (row no groups) shows that this variant is never as good as ODGI, even for larger number of crops, confirming that the idea of grouped detections provides a consistent advantage.

**Offsets.** We perform two ablation experiments to analyze the influence of the region rescaling step introduced in Section 3.2. First, instead of using learned offsets we test the model with offset values fixed to $\frac{2}{3}$, i.e. 50% expansion of the bounding boxes, which corresponds to the value of the target offsets margin $\delta$ we chose for standard ODGI. Our experiments in Table 3 show that this variant is inferior to ODGI, confirming that the model benefits from learning offsets tailored to its predictions. Second, we entirely ignore the rescaling step during the patch extraction step (row no offsets). This affects the $\text{MAP}$ even more negatively: extracted crops are generally localized close to the relevant objects, but do not fully enclose them. Consequently, the second stage retrieves partial objects, but with very high confidence, resulting in strong false positives predictions. In this case, most correct detections emerge from stage 1’s early-exit predictions, hence increasing $\gamma_1$, i.e. passing forward more crops, does not improve the $\text{MAP}$ in this scenario.

5. Conclusions

We introduce ODGI, a novel cascaded scheme for object detection that identifies groups of objects in early stages, and refines them in later stages as needed: Consequently, (i) empty image regions are discarded, thus saving computations especially in situations with heterogeneous object density, such as aerial imagery, and (ii) groups are typically larger structures than individuals and easier to detect at lower resolutions. Furthermore, ODGI can be easily added to off-the-shelf backbone networks commonly used for single-shot object detection: In extensive experiments, we show that the proposed method offers substantial computational savings without sacrificing accuracy. The effect is particularly striking on devices with limited computational or energy resources, such as embedded platforms.
[33] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision (IJCV)*, pages 137–154, 2004. 2

[34] G.-S. Xia, X. Bai, L. Zhang, S. Belongie, J. Luo, M. Datcu, and M. Pelilo. Dota: A large-scale dataset for object detection in aerial images. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1

[35] W. Xie, J. A. Noble, and A. Zisserman. Microscopy cell counting and detection with fully convolutional regression networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 6(3):283–292, 2018. 1

[36] F. Yang, H. Fan, P. Chu, E. Blasch, and H. Ling. Clustered object detection in aerial images. In *International Conference on Computer Vision (ICCV)*, 2019. 2

[37] T. Zhao and R. Nevatia. Car detection in low resolution aerial images. In *International Conference on Computer Vision (ICCV)*, 2001. 1

[38] P. Zhu, L. Wen, X. Bian, H. Ling, and Q. Hu. Vision meets drones: A challenge, a ECCV 2018 Workshop, 2018. 1, 2