TensorMask: A Foundation for Dense Object Segmentation

Xinlei Chen Ross Girshick Kaiming He Piotr Dollár
Facebook AI Research (FAIR)

Abstract

Sliding-window object detectors that generate bounding-box object predictions over a dense, regular grid have advanced rapidly and proven popular. In contrast, modern instance segmentation approaches are dominated by methods that first detect object bounding boxes, and then crop and segment these regions, as popularized by Mask R-CNN. In this work, we investigate the paradigm of dense sliding-window instance segmentation, which is surprisingly under-explored. Our core observation is that this task is fundamentally different than other dense prediction tasks such as semantic segmentation or bounding-box object detection, as the output at every spatial location is itself a geometric structure with its own spatial dimensions. To formalize this, we treat dense instance segmentation as a prediction task over 4D tensors and present a general framework called TensorMask *that explicitly captures this geometry and enables novel operators on 4D tensors. We demonstrate that the tensor view leads to large gains over baselines that ignore this structure, and leads to results comparable to Mask R-CNN. These promising results suggest that TensorMask can serve as a foundation for novel advances in dense mask prediction and a more complete understanding of the task. Code will be made available.*

1. Introduction

The sliding-window paradigm—*finding objects by looking in each window placed over a dense set of image locations*—is one of the earliest and most successful concepts in computer vision [36, 38, 9, 10] and is naturally connected to convolutional networks [20]. However, while today's top-performing object detectors rely on sliding window prediction to generate initial candidate regions, a *refinement* stage is applied to these candidate regions to obtain more accurate predictions, as pioneered by Faster R-CNN [34] and Mask R-CNN [17] for bounding-box object detection and instance segmentation, respectively. This class of methods has dominated the COCO detection challenges [24].

Recently, bounding-box object detectors which eschew the refinement step and focus on direct sliding-window prediction, as exemplified by SSD [27] and RetinaNet [23], have witnessed a resurgence and shown promising results. In contrast, the field has *not* witnessed equivalent progress in dense sliding-window instance segmentation; there are no direct, dense approaches analogous to SSD / RetinaNet for mask prediction. Why is the dense approach thriving for box detection, yet entirely missing for instance segmentation? This is a question of fundamental scientific interest. *The goal of this work is to bridge this gap and provide a foundation for exploring dense instance segmentation.*

Figure 1. Selected output of *TensorMask*, our proposed framework for performing *dense sliding-window instance segmentation*. We treat dense instance segmentation as a prediction task over *structured* 4D tensors. In addition to obtaining competitive quantitative results, TensorMask achieves results that are *qualitatively* reasonable. Observe that both small and large objects are well delineated and more critically *overlapping objects are properly handled*.

Our main insight is that the core concepts for defining dense mask representations, as well as effective realizations of these concepts in neural networks, are both lacking. Unlike bounding boxes, which have a fixed, low-dimensional representation regardless of scale, segmentation masks can benefit from richer, more structured representations. For example, each mask is itself a 2D spatial map, and masks for larger objects can benefit from the use of larger spatial maps. Developing effective representations for dense masks is a key step toward enabling dense instance segmentation.

To address this, we define a set of core concepts for representing masks with high-dimensional tensors that allows for the exploration of novel network architectures for dense mask prediction. We present and experiment with several such networks in order to demonstrate the merits of the pro-

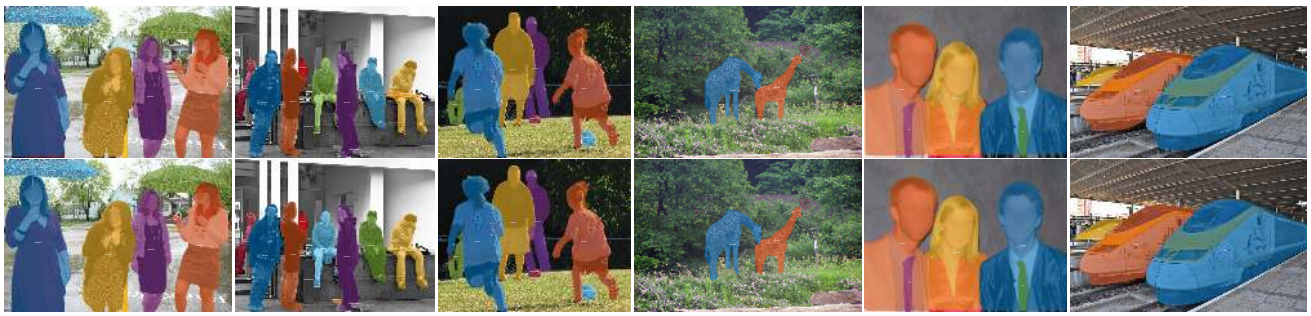


Figure 2. Example results of TensorMask and Mask R-CNN [17] with a ResNet-101-FPN backbone (on the same images as used in Fig. 6 of Mask R-CNN [17]). The results are *quantitatively* and *qualitatively* similar, demonstrating that the dense sliding window paradigm can indeed be effective for the instance segmentation task. We challenge the reader to identify which results were generated by TensorMask.[1]

posed representations. Our framework, called *TensorMask*, establishes the *first* dense sliding-window instance segmentation system that achieves results near to Mask R-CNN.

The central idea of the TensorMask representation is to use *structured* 4D tensors to represent masks over a spatial domain. This perspective stands in contrast to prior work on the related task of segmenting class-agnostic object proposals such as DeepMask [31] and InstanceFCN [7] that used *unstructured* 3D tensors, in which the mask is packed into the third 'channel' axis. The channel axis, unlike the axes representing object position, does not have a clear geometric meaning and is therefore difficult to manipulate. By using a basic channel representation, one misses an opportunity to benefit from using structural arrays to represent masks as 2D entities—analogous to the difference between MLPs and ConvNets [20] for representing 2D images.

Unlike these channel-oriented approaches, we propose to leverage 4D tensors of shape (V, U, H, W), in which both (H, W)—*representing object position*—and (V, U)—*representing relative mask position*—are geometric sub-tensors, *i.e.*, they have axes with well-defined units and geometric meaning w.r.t. the image. This shift in perspective from encoding masks in an unstructured channel axis to using structured geometric sub-tensors enables the definition of novel operations and network architectures. These networks can operate directly on the (V, U) sub-tensor in geometrically meaningful ways, including coordinate transformation, up-/downscaling, and use of scale pyramids.

Enabled by the TensorMask framework, we develop a pyramid structure over a scale-indexed list of 4D tensors, which we call a *tensor bipyramid*. Analogous to a feature pyramid, which is a list of feature maps at multiple scales, a tensor bipyramid contains a list of 4D tensors with shapes $(2^k V, 2^k U, \frac{1}{2^k} H, \frac{1}{2^k} W)$, where $k \geq 0$ indexes scale. This structure has a pyramidal shape in *both* (H, W) and (V, U) geometric sub-tensors, but growing in *opposite* directions. This natural design captures the desirable property that large objects have high-resolution masks with coarse spatial localization (large k) and small objects have low-resolution masks with fine spatial localization (small k).

We combine these components into a network backbone and training procedure closely following RetinaNet [23] in which our dense mask predictor extends the original dense bounding box predictor. With detailed ablation experiments, we evaluate the effectiveness of the TensorMask framework and show the importance of explicitly capturing the geometric structure of this task. Finally, we show TensorMask yields similar results to its Mask R-CNN counterpart (see Figs. 1 and 2). These promising results suggest the proposed framework can help pave the way for future research on dense sliding-window instance segmentation.

2. Related Work

Classify mask proposals. The modern instance segmentation task was introduced by Hariharan *et al*. [15] (before being popularized by COCO [24]). In their work, the method proposed for this task involved first generating object *mask proposals* [37, 1], then classifying these proposals [15]. In earlier work, the *classify-mask-proposals* methodology was used for other tasks. For example, Selective Search [37] and the original R-CNN [12] classified mask proposals to obtain box detections and semantic segmentation results; these methods could easily be applied to instance segmentation. These early methods relied on bottom-up mask proposals computed by pre-deep-learning era methods [37, 1]; our work is more closely related to dense sliding-window methods for mask object proposals as pioneered by DeepMask [31]. We discuss this connection shortly.

Detect then segment. The now dominant paradigm for instance segmentation involves first detecting objects with a box and then segmenting each object using the box as a guide [8, 39, 21, 17]. Perhaps the most successful instantiation of the *detect-then-segment* methodology is Mask R-CNN [17], which extended the Faster R-CNN [34] detector with a simple mask predictor. Approaches that build on Mask R-CNN [26, 30, 4] have dominated leaderboards of recent challenges [24, 29, 6]. Unlike in bounding-box

[1] In Fig. 2, Mask R-CNN results on top; TensorMask results on bottom.

detection, where sliding-window [27, 33, 23] and region-based [11, 34] methods have both thrived, in the area of instance segmentation, research on dense sliding-window methods has been missing. Our work aims to close this gap.

Label pixels then cluster. A third class of approaches to instance segmentation (*e.g.*, [3, 19, 2, 25]) builds on models developed for semantic segmentation [28, 5]. These approaches label each image pixel with a category and some auxiliary information that a clustering algorithm can use to group pixels into object instances. These approaches benefit from improvements on semantic segmentation and natively predict higher-resolution masks for larger objects. Compared to detect-then-segment methods, *label-pixels-then-cluster* methods lag behind in accuracy on popular benchmarks [24, 29, 6]. Instead of employing fully convolutional models for *dense pixel labeling*, TensorMask explores the framework of building fully convolutional (*i.e.*, dense sliding window) models for *dense mask prediction*, where the output at each spatial location is itself a 2D spatial map.

Dense sliding window methods. To the best of our knowledge, *no prior methods exist for dense sliding-window instance segmentation*. The proposed TensorMask framework is the *first* such approach. The closest methods are for the related task of class-agnostic mask *proposal* generation, specifically models such as DeepMask [31, 32] and InstanceFCN [7] which apply convolutional neural networks to generate mask proposals in a *dense sliding-window* manner. Like these approaches, TensorMask is a dense sliding-window model, but it spans a more expressive design space. DeepMask and InstanceFCN can be expressed naturally as class-agnostic TensorMask models, but TensorMask enables novel architectures that perform better. Also, unlike these class-agnostic methods, TensorMask performs multi-class classification in parallel to mask prediction, and thus can be applied to the task of instance segmentation.

3. Tensor Representations for Masks

The central idea of the TensorMask framework is to use *structured high-dimensional tensors* to represent image content (*e.g.*, masks) in a set of densely sliding windows.

Consider a $V \times U$ window sliding on a feature map of width W and height H. It is possible to represent all masks in all sliding window locations by a tensor of a shape (C, H, W), where each mask is parameterized by $C = V \cdot U$ pixels. This is the representation used in DeepMask [31].

The underlying spirit of this representation, however, is in fact a higher dimensional (4D) tensor with shape (V, U, H, W). The sub-tensor (V, U) represents a mask as a 2D spatial entity. Instead of viewing the channel dimension C as a black box into which a $V \times U$ mask is arranged, the tensor perspective enables several important concepts for representing dense masks, discussed next.

3.1. Unit of Length

The *unit of length* (or simply *unit*) of each spatial axis is a necessary concept for understanding 4D tensors in our framework. Intuitively, the unit of an axis defines the length of one pixel along it. Different axes can have different units.

The unit of the H and W axes, denoted as σ_{HW}, can be set as the *stride* w.r.t. the input image (*e.g.*, res_4 of ResNet-50 [18] has $\sigma_{\text{HW}}=16$ image pixels). Analogously, the V and U axes define another 2D spatial domain and have their own unit, denoted as σ_{VU}. Shifting one pixel along the V or U axis corresponds to shifting σ_{VU} pixels on the input image. The unit σ_{VU} need not be equal to the unit σ_{HW}, a property that our models will benefit from.

Defining units is necessary because the interpretation of the tensor shape (V, U, H, W) is ambiguous if units are not specified. For example, (V, U) represents a $V \times U$ window in image pixels if $\sigma_{\text{VU}}=1$ image pixel, but a $2V \times 2U$ window in image pixels if $\sigma_{\text{VU}}=2$ image pixels. The units and how they change due to up/down-scaling operations are central to multi-scale representations (more in §3.6).

3.2. Natural Representation

With the definition of units, we can formally describe the representational meaning of a (V, U, H, W) tensor. In our simplest definition, this tensor represents the windows sliding over (H, W). We call this the *natural representation*. Denoting $\alpha = \sigma_{\text{VU}} / \sigma_{\text{HW}}$ as the ratio of units, formally we have:

Natural Representation: *For a 4D tensor of shape (V, U, H, W), its value at coordinates (v, u, y, x) represents the mask value at $(y + \alpha v, x + \alpha u)$ in the $\alpha V \times \alpha U$ window centered at (y, x).*[2]

Here $(v, u, y, x) \in [-\frac{V}{2}, \frac{V}{2}) \times [-\frac{U}{2}, \frac{U}{2}) \times [0, H) \times [0, W)$, where '$\times$' denotes cartesian product. Conceptually, the tensor can be thought of as a continuous function in this domain. For implementation, we must instead rasterize the 4D tensor as a discrete function defined on sampled locations. We assume a sampling rate of one sample per unit, with samples located at integer coordinates (*e.g.*, if $U=3$, then $u \in \{-1, 0, 1\}$). This assumption allows the same value U to represent both the length of the axis in terms of units (*e.g.*, $3\sigma_{\text{VU}}$) and also the number of discrete samples stored for the axis. This is convenient for working with tensors produced by neural networks that are discrete *and* have lengths.

Fig. 3 (left) illustrates an example when $V=U=3$ and α is 1. The natural representation is intuitive and easy to parse as the output of a network, but it is not the only possible representation in a deep network, as discussed next.

[2] Derivation: on the input image pixels, the center of a sliding window is $(y \cdot \sigma_{\text{HW}}, x \cdot \sigma_{\text{HW}})$, and a pixel located w.r.t. this window is at $(y \cdot \sigma_{\text{HW}} + v \cdot \sigma_{\text{VU}}, x \cdot \sigma_{\text{HW}} + u \cdot \sigma_{\text{VU}})$. Projecting to the HW domain (*i.e.*, normalizing by the unit σ_{HW}) gives us (y, x) and $(y + \alpha v, x + \alpha u)$.

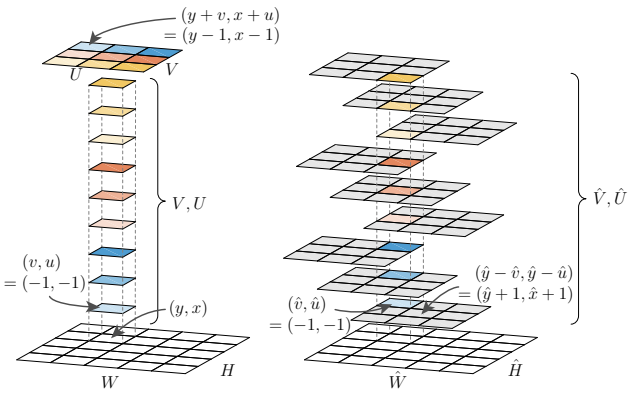


Figure 3. Left: **Natural representation**. The (V, U) sub-tensor at a pixel represents a window centered at this pixel. Right: **Aligned representation**. The $(\hat{V}, \hat{U})$ sub-tensor at a pixel represents the values at this pixel in each of the windows overlapping it.

3.3. Aligned Representation

In the natural representation, a sub-tensor (V, U) located at (y, x) represents values at *offset* pixels $(y+\alpha v, x+\alpha u)$ instead of directly at (y, x). When using convolutions to compute features, preserving *pixel-to-pixel alignment* between input pixels and predicted output pixels can lead to improvements (this is similar to the motivation for RoIAlign [17]). Next we describe a pixel-aligned representation for dense masks under the tensor perspective.

Formally, we define the *aligned representation* as:

Aligned Representation: *For a 4D tensor $(\hat{V}, \hat{U}, \hat{H}, \hat{W})$, its value at coordinates $(\hat{v}, \hat{u}, \hat{y}, \hat{x})$ represents the mask value at $(\hat{y}, \hat{x})$ in the $\hat{\alpha}\hat{V}\times\hat{\alpha}\hat{U}$ window centered at $(\hat{y}-\hat{\alpha}\hat{v}, \hat{x}-\hat{\alpha}\hat{u})$.*

$\hat{\alpha}=\hat{\sigma}_{\text{VU}}/\hat{\sigma}_{\text{HW}}$ is the ratio of units in the aligned representation.

Here, the sub-tensor $(\hat{V}, \hat{U})$ at pixel $(\hat{y}, \hat{x})$ always describes the values taken at this pixel, *i.e.* it is *aligned*. The subspace $(\hat{V}, \hat{U})$ does *not* represent a single mask, but instead enumerates mask values in all $\hat{V}\cdot\hat{U}$ windows that overlap pixel $(\hat{y}, \hat{x})$. Fig. 3 (right) illustrates an example when $\hat{V}=\hat{U}=3$ (nine overlapping windows) and $\hat{\alpha}$ is 1.

Note that we denote tensors in the aligned representation as $(\hat{V}, \hat{U}, \hat{H}, \hat{W})$ (and likewise for coordinates/units). This is in the spirit of '*named tensors*' [35] and proves useful.

Our aligned representation is related to the *instance-sensitive score maps* proposed in InstanceFCN [7]. We prove (in §A.2) that those score maps behave like our aligned representation but with nearest-neighbor interpolation on $(\hat{V}, \hat{U})$, which makes them *unaligned*. We test this experimentally and show it degrades results severely.

3.4. Coordinate Transformation

We introduce a coordinate transformation between natural and aligned representations, so they can be used interchangeably in a single network. This gives us additional flexibility in the design of novel network architectures.

For simplicity, we assume units in both representations are the same: *i.e.*, $\sigma_{\text{HW}}=\hat{\sigma}_{\text{HW}}$ and $\sigma_{\text{VU}}=\hat{\sigma}_{\text{VU}}$, and thus $\alpha=\hat{\alpha}$ (for the more general case see §A.1). Comparing the definitions of natural *vs.* aligned representations, we have the following two relations for x, u: $x+\alpha u=\hat{x}$ and $x=\hat{x}-\hat{\alpha}\hat{u}$. With $\alpha=\hat{\alpha}$, solving this equation for $\hat{x}$ and $\hat{u}$ gives: $\hat{x}=x+\alpha u$ and $\hat{u}=u$. A similar results hold for y, v. So the transformation from the aligned representation ($\hat{\mathcal{F}}$) to the natural representation ($\mathcal{F}$) is:

$$\mathcal{F}(v, u, y, x) = \hat{\mathcal{F}}(v, u, y+\alpha v, x+\alpha u). \quad (1)$$

We call this transform `align2nat`. Likewise, solving this set of two relations for x and u gives the reverse transform of `nat2align`: $\hat{\mathcal{F}}(\hat{v}, \hat{u}, \hat{y}, \hat{x})=\mathcal{F}(\hat{v}, \hat{u}, \hat{y}-\alpha\hat{v}, \hat{x}-\alpha\hat{u})$. While all the models presented in this work only use `align2nat`, we present both cases for completeness.

Without restrictions on α, these transformations may involve indexing a tensor at a non-integer coordinate, *e.g.* if $x+\alpha u$ is not an integer. Since we only permit integer coordinates in our implementation, we adopt a simple strategy: when the op `align2nat` is called, we ensure that α is a positive integer. We can satisfy this constraint on α by changing units with up/down-scaling ops, as described next.

3.5. Upscaling Transformation

The aligned representation enables the use of a *coarse* $(\hat{V}, \hat{U})$ sub-tensors to create finer (V, U) sub-tensors, which proves quite useful. Fig. 4 illustrates this transformation, which we call `up_align2nat` and describe next.

The `up_align2nat` op accepts a $(\hat{V}, \hat{U}, \hat{H}, \hat{W})$ tensor as input. The $(\hat{V}, \hat{U})$ sub-tensor is $\lambda\times$ coarser than the desired output (so its unit is $\lambda\times$ bigger). It performs bilinear upsampling, `up_bilinear`, in the $(\hat{V}, \hat{U})$ domain by λ, reducing the underlying unit by $\lambda\times$. Next, the `align2nat` op converts the output into the natural representation. The full `up_align2nat` op is shown in Fig. 4.

As our experiments demonstrate, the `up_align2nat` op is effective for generating high-resolution masks without inflating channel counts in preceding feature maps. This in turn enables novel architectures, as described next.

3.6. Tensor Bipyramid

In multi-scale box detection it is common practice to use a *lower*-resolution feature map to extract *larger*-scale objects [10, 22]—this is because a sliding window of a *fixed size* on a lower-resolution map corresponds to a larger region in the input image. This also holds for multi-scale mask detection. However, unlike a box that is always represented by four numbers regardless of its scale, a mask's pixel size must scale with object size in order to maintain constant resolution density. Thus, instead of always using $V\times U$ units to present masks of different scales, we propose to adapt the number of mask pixels based on the scale.

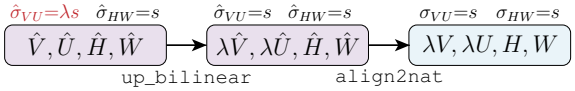


Figure 4. The **`up_align2nat` op** is defined as a sequence of two ops. It takes an input tensor that has a coarse, $\lambda\times$ lower resolution on $\hat{V}\hat{U}$ (so the unit $\hat{\sigma}_{\text{VU}}$ is $\lambda\times$ larger). The op performs upsampling on $\hat{V}\hat{U}$ by λ followed by `align2nat`, resulting in an output where $\sigma_{\text{VU}}{=}\sigma_{\text{HW}}{=}s$ (where s is the stride).

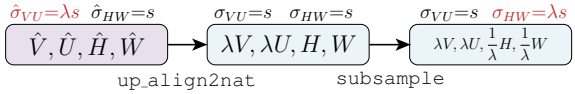


Figure 5. The **`swap_align2nat` op** is defined by two ops. It upscales the input by `up_align2nat` (Fig. 4), then performs `subsample` on the HW domain. Note how the op *swaps* the units between the VU and HW domains. In practice, we implement this op in place so the complexity is independent of λ.

Consider the natural representation (V,U,H,W) on a feature map of the *finest* level. Here, the (H,W) domain has the highest resolution (smallest unit). We expect this level to handle the *smallest* objects, so the (V,U) domain should have the lowest resolution. With reference to this, we build a pyramid that gradually *reduces* (H,W) and *increases* (V,U). Formally, we define a *tensor bipyramid* as:

Tensor bipyramid: *A tensor bipyramid is a list of tensors of shapes:* $(2^kV, 2^kU, \frac{1}{2^k}H, \frac{1}{2^k}W)$*, for* $k{=}0,1,2,\ldots$*, with units* $\sigma^{k+1}_{VU} = \sigma^k_{VU}$ *and* $\sigma^{k+1}_{HW} = 2\sigma^k_{HW}, \forall k$.

Because the units σ^k_{VU} are the same across all levels, a $2^kV\times 2^kU$ mask has $4^k\times$ more pixels in the input image. In the (H,W) domain, because the units σ^k_{HW} increase with k, the number of predicted masks decreases for larger masks, as desired. Note that the total size of each level is the same (it is $V{\cdot}U{\cdot}H{\cdot}W$). A tensor bipyramid can be constructed using the `swap_align2nat` operation, described next.

This `swap_align2nat` op is composed of two steps: first, an input tensor with fine $(\hat{H},\hat{W})$ and coarse $(\hat{V},\hat{U})$ is upscaled to $(2^kV, 2^kU, H, W)$ using `up_align2nat`. Then (H,W) is subsampled to obtain the final shape. The combination of `up_align2nat` and `subsample`, shown in Fig. 5, is called `swap_align2nat`: the units before and after this op are *swapped*. For efficiency, it is not necessary to compute the intermediate tensor of shape $(2^kV, 2^kU, H, W)$ from `up_align2nat`, which would be prohibitive. This is because only a small subset of values in this intermediate tensor appear in the final output after subsampling. So although Fig. 5 shows the conceptual computation, in practice we implement `swap_align2nat` as a single op that only performs the necessary computation and has complexity $O(V{\cdot}U{\cdot}H{\cdot}W)$ regardless of k.

4. TensorMask Architecture

We now present models enabled by TensorMask representations. These models have a mask prediction head that generates masks in sliding windows and a classification head to predict object categories, analogous to the box regression and classification heads in sliding-window object detectors [27, 23]. Box prediction is not necessary for TensorMask models, but can easily be included.

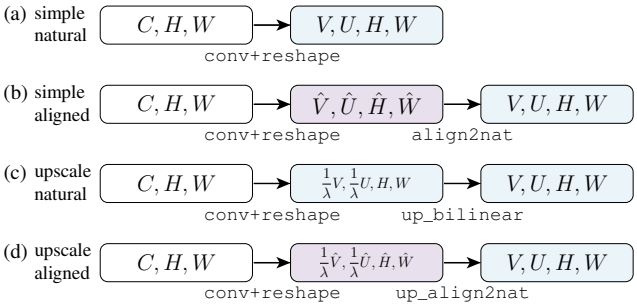


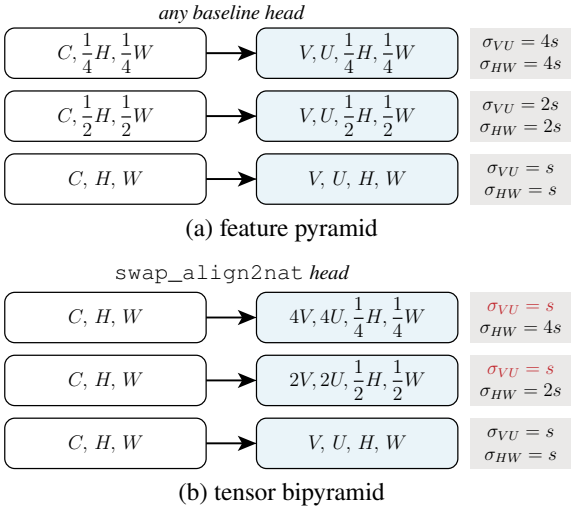
Figure 6. **Baseline mask prediction heads**: Each of the four heads shown starts from a feature map (*e.g.*, from a level of an FPN [22]) with an arbitrary channel number C. Then a 1×1 `conv` layer projects the features into an appropriate number of channels, which form the specified 4D tensor by `reshape`. The output units of these four heads are the same, and $\sigma_{\text{VU}}{=}\sigma_{\text{HW}}$.

4.1. Mask Prediction Heads

Our mask prediction branch attaches to a convolutional backbone. We use FPN [22], which generates a pyramid of feature maps with sizes $(C, \frac{1}{2^k}H, \frac{1}{2^k}W)$ with a fixed number of channels C per level k. These maps are used as input for each prediction head: mask, class, and box. Weights for the heads are shared across levels, but not between tasks.

Output representation. We always use the *natural* representation (§3.2) as the *output* format of the network. Any representation (natural, aligned, *etc.*) can be used in the intermediate layers, but it will be transformed into the natural representation for the output. This standardization decouples the loss definition from network design, making use of different representations simpler. Also, our mask output is *class-agnostic*, *i.e.*, the window always predicts a single mask regardless of class; the class of the mask is predicted by the classification head. Class-agnostic mask prediction avoids multiplying the output size by the number of classes.

Baseline heads. We consider a set of four baseline heads, illustrated in Fig. 6. Each head accepts an input feature map of shape (C,H,W) for any (H,W). It then applies a 1×1 convolutional layer (with ReLU) with the appropriate number of output channels such that reshaping it into a 4D tensor produces the desired shape for the next layer, denoted as '`conv+reshape`'. Fig. 6a and 6b are ***simple heads*** that use natural and aligned representations, re-



(a) feature pyramid

(b) tensor bipyramid

Figure 7. Conceptual comparison between: (a) a **feature pyramid** with any one of the baseline heads (Fig. 6) attached, and (b) a **tensor bipyramid** that uses `swap_align2nat` (Fig. 5). A baseline head on the feature pyramid has $\sigma_{\text{VU}}{=}\sigma_{\text{HW}}$ for each level, which implies that masks for large objects and small objects are predicted using the same number of pixels. On the other hand, the `swap_align2nat` head can keep the mask resolution high (*i.e.*, σ_{VU} is the same across levels) despite the HW resolution changes.

spectively. In both cases, we use $V{\cdot}U$ output channels for the $1{\times}1$ `conv`, followed by `align2nat` in the latter case. Fig. 6c and 6d are ***upscaling heads*** that use the natural and aligned representations, respectively. Their $1{\times}1$ `conv` has $\lambda^2{\times}$ *fewer* output channels than in the simple heads.

In a baseline TensorMask model, one of these four heads is selected and attached to all FPN levels. The output forms a pyramid of $(V, U, \frac{1}{2^k}H, \frac{1}{2^k}W)$, see Fig. 7a. For each head, the output sliding window always has the same unit as the feature map on which it slides: $\sigma_{\text{VU}}{=}\sigma_{\text{HW}}$ for all levels.

Tensor bipyramid head. Unlike the baseline heads, the tensor bipyramid head (§3.6) accepts a feature map of fine resolution (H, W) at all levels. Fig. 8 shows a minor modification of FPN to obtain these maps. For each of the resulting levels, now all (C, H, W), we first use `conv+reshape` to produce the appropriate 4D tensor, then run a mask prediction head with `swap_align2nat`, see Fig. 7b. The tensor bipyramid model is the most effective TensorMask variant explored in this work.

4.2. Training

Label assignment. We use a version of the DeepMask assignment rule [31] to label each window. A window satisfying three conditions w.r.t. a ground-truth mask m is positive:

(i) Containment: the window fully contains m and the longer side of m, in image pixels, is at least 1/2 of the longer

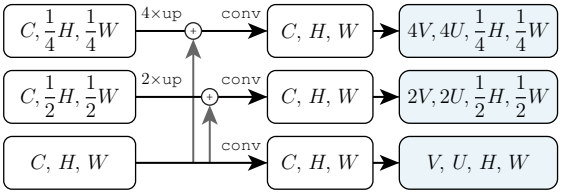


Figure 8. **Conversion of FPN** feature maps from $(C, \frac{1}{2^k}H, \frac{1}{2^k}W)$ to (C, H, W) for use with tensor bipyramid (see Fig. 7b). For an FPN level $(C, \frac{1}{2^k}H, \frac{1}{2^k}W)$, we apply bilinear interpolation to upsample the feature map by a factor of 2^k. As the upscaling can be large, we add the finest level feature map to all levels (including the finest level itself), followed by one $3{\times}3$ `conv` with ReLU.

side of the window, that is, $\max(U{\cdot}\sigma_{\text{VU}}, V{\cdot}\sigma_{\text{VU}})$.[3]

(ii) Centrality: the center of m's bounding box is within one unit (σ_{VU}) of the window center in ℓ_2 distance.

(iii) Uniqueness: there is no other mask $m'{\neq}m$ that satisfies the other two conditions.

If m satisfies these three conditions, then the window is labeled as a *positive example* whose ground-truth mask, object category, and box are given by m. Otherwise, the window is labeled as a *negative example*.

In contrast to the IoU-based assignment rules for boxes in sliding-window detectors (*e.g.*, RPN [34], SSD [27], RetinaNet [23]), our rules are *mask-driven*. Experiments show that our rules work well even when using only 1 or 2 window sizes with a single aspect ratio of 1:1, versus, *e.g.*, RetinaNet's 9 anchors of multiple scales and aspect ratios.

Loss. For the mask prediction head, we adopt a per-pixel binary classification loss. In our setting, the ground-truth mask inside a sliding window often has a wide margin, resulting in an imbalance between foreground *vs*. background pixels. To address this imbalance, we set the weights for foreground pixels to 1.5 in the binary cross-entropy loss. The mask loss of a window is averaged over all pixels in the window (note that in a tensor bipyramid the window size varies across levels), and the total mask loss is averaged over all positive windows (negative windows do not contribute to the mask loss).

For the classification head, we again adopt FL* with $\gamma{=}3$ and $\alpha{=}0.3$. For box regression, we use a parameter-free ℓ_1 loss. The total loss is a weighted sum of all task losses.

Implementation details. Our FPN implementation closely follows [23]; each FPN level is output by four $3{\times}3$ `conv` layers of C channels with ReLU (instead of one `conv` in the original FPN [22]). As with the heads, weights are shared across levels, but not between tasks. In addition, we found that averaging (instead of summing [22]) the top-down and lateral connections in FPN improved training stability. We

[3] A fallback is used to increase small object recall: masks smaller than the minimum assignable size are assigned to windows of the smallest size.

Figure 9. **Baseline upscaling heads** (λ=5). *Top:* the natural upscaling head (a) produces coarse masks, and is ineffective for large λ. *Left:* for simple scenes, the unaligned head (b) and aligned head (c) (which use nearest-neighbor and bilinear interpolation, respectively), behave similarly. *Right:* for overlapping objects the difference is striking: *the unaligned head creates severe artifacts*.

use FPN levels 2 through 7 (k=0, . . . , 5) with C=128 channels for the four `conv` layers in the mask and box branches, and C=256 (the same as RetinaNet [23]) for the classification branch. Unless noted, we use ResNet-50 [18].

For training, all models are initialized from ImageNet pre-trained weights. We use scale jitter where the shorter image side is randomly sampled from [640, 800] pixels [16]. Following SSD [27] and YOLO [33], which train models longer (∼65 and 160 epochs) than [23, 17], we adopt the '6×' schedule [16] (∼72 epochs), which improves results. The minibatch size is 16 images in 8 GPUs. The base learning rate is 0.02, with linear warm-up [14] of 1k iterations. Other hyper-parameters are kept the same as [13].

4.3. Inference

Inference is similar to dense sliding-window object detectors. We use a single scale of 800 pixels for the shorter image side. Our model outputs a mask prediction, a class score, and a predicted box for each sliding window. Non-maximum suppression (NMS) is applied to the top-scoring predictions using box IoU on the regressed boxes, following the settings in [22]. To convert predicted soft masks to binary masks at the original image resolution, we use the same method and hyper-parameters as Mask R-CNN [17].

5. Experiments

We report results on COCO instance segmentation [24]. All models are trained on the ∼118k `train2017` images and tested on the 5k `val2017` images. Final results are on `test-dev`. We use COCO *mask* average precision (denoted by AP). When reporting *box* AP, we denote it as AP^{bb}.

head	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
natural	28.5	52.2	28.6	14.4	30.2	40.1
aligned	28.9	52.5	29.3	14.6	30.8	40.7

Table 1. **Simple heads**: natural *vs*. aligned (Fig. 6a *vs*. 6b) with $V \times U$=15×15 perform comparably if upscaling is not used.

5.1. TensorMask Representations

First we explore various tensor representations for masks using V=U=15 and a ResNet-50-FPN backbone. We report quantitative results in Tab. 2 and show qualitative comparisons in Figs. 2 and 9.

Simple heads. Tab. 1 compares natural *vs*. aligned representations with simple heads (Fig. 6a *vs*. 6b). Both representations perform similarly, with a marginal gap of 0.4 AP. The simple natural head can be thought of as a *class-specific* variant of DeepMask [31] with an FPN backbone [22] and focal loss [23]. As we aim to use lower-resolution intermediate representations, we explore upscaling heads next.

Upscaling heads. Tab. 2a compares natural *vs*. aligned representations with upscaling heads (Fig. 6c *vs*. 6d). The output size is fixed at $V \times U$=15×15. Given an upscaling factor λ, the `conv` in Fig. 6 has $\frac{1}{\lambda^2}VU$ channels, *e.g*., 9 channels with λ=5 (*vs*. 225 channels if no upscaling). The difference in accuracy is big for large λ: the aligned variant improves AP *+9.2* over the natural head (48% relative) when λ=5.

The visual difference is clear in Fig. 9a (natural) *vs*. 9c (aligned). The upscale aligned head still produces sharp masks with large λ. *This is critical for the tensor bipyramid, where we have an output of* $2^kV \times 2^kU$*, which is achieved with a large upscaling factor of* λ=2^k (e.g., 32); see Fig. 5.

Interpolation. The tensor view reveals the $(\hat{V}, \hat{U})$ subtensor as a 2D spatial entity that can be manipulated. Tab. 2b compares the upscale aligned head with bilinear (default) *vs*. *nearest-neighbor* interpolation on $(\hat{V}, \hat{U})$. We refer to this latter variant as *unaligned* since quantization breaks pixel-to-pixel alignment. The unaligned variant is related to InstanceFCN [7] (see §A.2).

We observe in Tab. 2b that bilinear interpolation yields solid improvements over nearest-neighbor interpolation, especially if λ is large (ΔAP=*3.1*). These interpolation methods lead to striking visual differences when objects overlap: see Fig. 9b (unaligned) *vs*. 9c (aligned).

Tensor bipyramid. Replacing the best feature pyramid model with a tensor bipyramid yields a large 5.1 AP improvement (Tab. 2c). Here, the mask size is $V \times U$=15×15 on level k=0, and is $32V \times 32U$=480×480 for k=5; see Fig. 7b. The higher resolution masks predicted for large objects (*e.g*., at k=5) have clear benefit: AP_L jumps by

head	λ	AP	AP_{50}	AP_{75}	Δ aligned - natural		
natural	1.5	28.0	51.7	27.8	*+0.9*	*+0.7*	*+1.5*
aligned		**28.9**	**52.4**	**29.3**			
natural	3	24.7	48.4	22.7	*+4.1*	*+3.9*	*+6.4*
aligned		**28.8**	**52.3**	**29.1**			
natural	5	19.2	42.1	15.6	*+9.2*	*+9.7*	*+13.0*
aligned		**28.4**	**51.8**	**28.6**			

(a) **Upscaling heads**: natural *vs.* aligned heads (Fig. 6c *vs.* 6d). The $V\times U$=15×15 output is upscaled by $\lambda\times$: `conv+reshape` uses $\frac{1}{\lambda^2}VU$ output channels as input. *The aligned representation has a large gain over its natural counterpart when λ is large.*

head	λ	AP	AP_{50}	AP_{75}	Δ bilinear - nearest		
nearest	1.5	28.6	52.1	29.0	*+0.3*	*+0.3*	*+0.3*
bilinear		**28.9**	**52.4**	**29.3**			
nearest	3	27.8	51.0	28.0	*+1.0*	*+1.3*	*+1.1*
bilinear		**28.8**	**52.3**	**29.1**			
nearest	5	25.3	47.6	25.0	*+3.1*	*+4.2*	*+3.6*
bilinear		**28.4**	**51.8**	**28.6**			

(b) **Upscaling: bilinear *vs.* nearest-neighbor** interpolation for the aligned head (Fig. 6d). The output has $V\times U$=15×15. With nearest-neighbor interpolation, the aligned upscaling head is similar to the InstanceFCN [7] head. *Bilinear interpolation shows a large gain when λ is large.*

head	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
feature pyramid, best	28.9	52.5	29.3	14.6	30.8	40.7
tensor bipyramid	**34.0**	**55.2**	**35.8**	**15.3**	**36.3**	**48.4**
Δ	*+5.1*	*+2.7*	*+6.5*	*+0.7*	*+5.5*	*+7.7*

(c) The **tensor bipyramid** substantially improves results compared to the best baseline head (Tab. 2a, row 2) on a *feature pyramid* (Fig. 7a).

$V\times U$	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
15×15	34.0	55.2	35.8	15.3	36.3	48.4
15×15, 11×11	**35.2**	**56.4**	**37.0**	**17.4**	**37.4**	**49.7**
Δ	*+1.2*	*+1.2*	*+1.2*	*+2.1*	*+1.1*	*+1.3*

(d) **Window sizes**: extending from one $V\times U$ window size (per level) to two increases all AP metrics. Both rows use the tensor bipyramid.

Table 2. **Ablations on TensorMask representations** on COCO `val2017`. All variants use ResNet-50-FPN and a 72 epoch schedule.

method	backbone	aug	epochs	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN [13]	R-50-FPN		24	34.9	57.2	36.9	15.4	36.6	50.8
Mask R-CNN, *ours*	R-50-FPN		24	34.9	56.8	36.8	15.1	36.7	50.6
Mask R-CNN, *ours*	R-50-FPN	✓	72	**36.8**	**59.2**	**39.3**	**17.1**	**38.7**	**52.1**
TensorMask	R-50-FPN	✓	72	35.4	57.2	37.3	16.3	36.8	49.3
Mask R-CNN, *ours*	R-101-FPN	✓	72	**38.3**	**61.2**	**40.8**	**18.2**	**40.6**	**54.1**
TensorMask	R-101-FPN	✓	72	37.1	59.3	39.4	17.4	39.1	51.6

Table 3. **Comparison with Mask R-CNN** for instance segmentation on COCO `test-dev`.

7.7 points. This improvement does *not* come at the cost of denser windows as the k=5 output is at $(\frac{H}{32}, \frac{W}{32})$ resolution.

Again, we note that it is intractable to have, *e.g.*, a 480^2-channel `conv`. The upscaling aligned head with bilinear interpolation is key to making tensor bipyramid possible.

Multiple window sizes. Thus far we have used a single window size (per-level) for all models, that is, $V\times U$=15×15. Analogous to the concept of *anchors* in RPN [34] that are also used in current detectors [33, 27, 23], we extend our method to multiple window sizes. We set $V\times U\in\{15\times15, 11\times11\}$, leading to two heads per level. Tab. 2d shows the benefit of having two window sizes: it increases AP by 1.2 points. More window sizes and aspect ratios are possible, suggesting room for improvement.

5.2. Comparison with Mask R-CNN

Tab. 3 summarizes the best TensorMask model on `test-dev` and compares it to the current dominant approach for COCO instance segmentation: Mask R-CNN [17]. We use the `Detectron` [13] code to reflect improvements since [17] was published. We modify it to match our implementation details (FPN average fusion, 1k warm-up, and ℓ_1 box loss). Tab. 3 row 1 & 2 verify that these subtleties have a negligible effect. Then we use training-time scale augmentation and a longer schedule [16], which yields an ~2 AP increase (Tab. 3 row 3) and establishes a fair and solid baseline for comparison.

The best TensorMask in Tab. 2d achieves 35.4 mask AP on `test-dev` (Tab. 3 row 4), close to Mask R-CNN counterpart's 36.8. With ResNet-101, TensorMask achieves 37.1 mask AP with a 1.2 AP gap behind Mask R-CNN. These results demonstrate that dense sliding-window methods *can* close the gap to 'detect-then-segment' systems (§2). Qualitative results are shown in Figs. 2, 10, and 11.

We report box AP of TensorMask in §A.3. Moreover, compared to Mask R-CNN, one intriguing property of TensorMask is that masks are *independent* from boxes. In fact, we find joint training of box and mask only gives marginal gain over mask-only training, see §A.4.

Speed-wise, the best R-101-FPN TensorMask runs at 0.38s/im on a V100 GPU (all post-processing included), *vs.* Mask R-CNN's 0.09s/im. Predicting masks in dense sliding windows (>100k) results in a computation overhead, *vs.* Mask R-CNN's sparse prediction on ≤100 final boxes. Accelerations are possible but outside the scope of this work.

Conclusion. TensorMask is a dense sliding-window instance segmentation framework that, for the *first* time, achieves results close to the well-developed Mask R-CNN framework—both qualitatively and quantitatively. It establishes a conceptually complementary direction for instance segmentation research. We hope our work will create new opportunities and make both directions thrive.

References

[1] Pablo Arbeláez, Jordi Pont-Tuset, Jonathan T Barron, Ferran Marques, and Jitendra Malik. Multiscale combinatorial grouping. In *CVPR*, 2014. 2

[2] Anurag Arnab and Philip HS Torr. Pixelwise instance segmentation with a dynamically instantiated network. In *CVPR*, 2017. 3

[3] Min Bai and Raquel Urtasun. Deep watershed transform for instance segmentation. In *CVPR*, 2017. 3

[4] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. *arXiv:1901.07518*, 2019. 2

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015. 3

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 2, 3

[7] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *ECCV*, 2016. 2, 3, 4, 7, 8, 9

[8] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, 2016. 2

[9] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. In *BMVC*, 2009. 1

[10] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 2010. 1, 4

[11] Ross Girshick. Fast R-CNN. In *ICCV*, 2015. 3

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 2

[13] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018. 7, 8

[14] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv:1706.02677*, 2017. 7

[15] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *ECCV*, 2014. 2

[16] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *arXiv:1811.08883*, 2018. 7, 8

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 1, 2, 4, 7, 8, 10, 11

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3, 7

[19] Alexander Kirillov, Evgeny Levinkov, Bjoern Andres, Bogdan Savchynskyy, and Carsten Rother. Instancecut: from edges to instances with multicut. In *CVPR*, 2017. 3

[20] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989. 1, 2

[21] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *CVPR*, 2017. 2

[22] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 4, 5, 6, 7

[23] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 1, 2, 3, 5, 6, 7, 8, 9

[24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 1, 2, 3, 7

[25] Shu Liu, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. SGN: Sequential grouping networks for instance segmentation. In *ICCV*, 2017. 3

[26] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018. 2

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *ECCV*, 2016. 1, 3, 5, 6, 7, 8

[28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 3

[29] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *ICCV*, 2017. 2, 3

[30] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. MegDet: A large mini-batch object detector. In *CVPR*, 2018. 2

[31] Pedro Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *NIPS*, 2015. 2, 3, 6, 7

[32] Pedro Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In *ECCV*, 2016. 3

[33] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *CVPR*, 2017. 3, 7, 8

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 2, 3, 6, 8

[35] Alexander Rush. Tensor considered harmful. 2019. 4

[36] R. Vaillant, C. Monrocq, and Y. LeCun. Original approach for the localisation of objects in images. *IEE Proc. on Vision, Image, and Signal Processing*, 1994. 1

[37] Koen EA van de Sande, Jasper RR Uijlings, Theo Gevers, and Arnold WM Smeulders. Segmentation as selective search for object recognition. In *ICCV*, 2011. 2

[38] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001. 1

[39] Sergey Zagoruyko, Adam Lerer, Tsung-Yi Lin, Pedro Pinheiro, Sam Gross, Soumith Chintala, and Piotr Dollár. A multipath network for object detection. In *BMVC*, 2016. 2