Benchmarking Detection Transfer Learning with Vision Transformers

Yanghao Li Saining Xie Xinlei Chen Piotr Dollár Kaiming He Ross Girshick

Facebook AI Research (FAIR)

Abstract

*Object detection is a central downstream task used to test if pre-trained network parameters confer benefits, such as improved accuracy or training speed. The complexity of object detection methods can make this benchmarking non-trivial when new architectures, such as Vision Transformer (ViT) models, arrive. These difficulties (*e.g.*, architectural incompatibility, slow training, high memory consumption, unknown training formulae,* etc.*) have prevented recent studies from benchmarking detection transfer learning with standard ViT models. In this paper, we present training techniques that overcome these challenges, enabling the use of standard ViT models as the backbone of Mask R-CNN. These tools facilitate the primary goal of our study: we compare five ViT initializations, including recent state-of-the-art self-supervised learning methods, supervised initialization, and a strong random initialization baseline. Our results show that recent masking-based unsupervised learning methods may, for the first time, provide convincing transfer learning improvements on COCO, increasing AP^{box} up to 4% (absolute) over supervised and prior self-supervised pre-training methods. Moreover, these masking-based initializations scale better, with the improvement growing as model size increases.*

1. Introduction

Unsupervised/self-supervised deep learning is commonly used as a *pre-training* step that initializes model parameters before they are *transferred* to a downstream task, such as image classification or object detection, for *fine-tuning*. The utility of an unsupervised learning algorithm is judged by downstream task metrics (*e.g.* accuracy, convergence speed, *etc.*) in comparison to baselines, such as *supervised* pre-training or *no* pre-training at all, *i.e.*, random initialization (often called training "from scratch").

Unsupervised deep learning in computer vision typically uses standard convolutional network (CNN) models [25], such as ResNets [20]. Transferring these models is relatively straightforward because CNNs are in widespread use in most downstream tasks, and thus benchmarking protocols are easy to define and baselines are plentiful (*e.g.* [17]). In other words, unsupervised learning with CNNs produces a plug-and-play parameter initialization.

We are now witnessing the growth of unsupervised learning with *Vision Transformer* (ViT) models [10], and while the high-level transfer learning methodology remains the same, the low-level details and baselines for some important downstream tasks have not been established. Notably, object detection, which has played a central role in the study of transfer learning over the last decade (*e.g.*, [35, 14, 9, 17]), was not explored in the pioneering work on ViT training [10, 7, 5]—supervised or unsupervised—due to the challenges (described shortly) of integrating ViTs into common detection models, like Mask R-CNN [19].

To bridge this gap, this paper establishes a transfer learning protocol for evaluating ViT models on object detection and instance segmentation using the COCO dataset [28] and the Mask R-CNN framework. We focus on standard ViT models, with minimal modifications, as defined in the original ViT paper [10], because we expect this architecture will remain popular in unsupervised learning work over the next few years due to its simplicity and flexibility when exploring new techniques, *e.g.*, masking-based methods [1, 16].

Establishing object detection baselines for ViT is challenging due to technical obstacles that include mitigating ViT's large memory requirements when processing detection-sized inputs (*e.g.*, $\sim$20$\times$ more patches than in pre-training), architectural incompatibilities (*e.g.*, single-scale ViT *vs.* a multi-scale detector), and developing effective training formulae (*i.e.*, learning schedules, regularization and data augmentation methods, *etc.*) for numerous pre-trained initializations, as well as random initialization. We overcome these obstacles and present strong ViT-based Mask R-CNN baselines on COCO when initializing ViT from-scratch [18], with pre-trained ImageNet [8] supervision, and with unsupervised pre-training using recent methods like MoCo v3 [7], BEiT [1], and MAE [16].

Looking beyond ViT, we hope our practices and observations will serve as a blueprint for future work comparing pre-training methods for more advanced ViT derivatives, like Swin [29] and MViT [12]. To facilitate community development we will release code in Detectron2 [40].

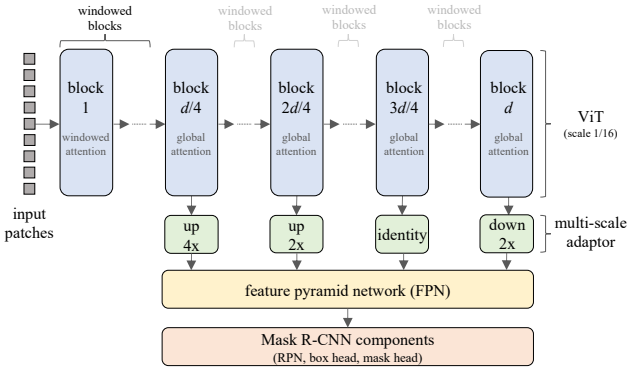

Figure 1. **ViT-based Mask R-CNN.** In §2 we describe how a standard ViT model can be used effectively as the backbone in Mask R-CNN. To save time and memory, we modify the ViT to use non-overlapping windowed attention in all but four of its Transformer blocks, spaced at an interval of $d/4$, where d is the total number of blocks (blue) [26]. To adapt the single-scale ViT to the multi-scale FPN (yellow), we make use of upsampling and downsampling modules (green) [11]. The rest of the system (light red) uses upgraded, but standard, Mask R-CNN components.

2. Approach

We select the Mask R-CNN [19] framework due to its ubiquitous presence in object detection and transfer learning research. Mask R-CNN is the foundation of higher complexity/higher performing systems, such as Cascade R-CNN [4] and HTC/HTC++ [6, 29], which may improve upon the results presented here at the cost of additional complexity that is orthogonal to the goal of benchmarking transfer learning. Our choice attempts to balance (relative) simplicity *vs*. complexity while providing compelling, even though not entirely state-of-the-art, results.

We configure Mask R-CNN with a number of upgraded modules (described in §2.2) and training procedures (described in §2.3) relative to the original publication. These upgrades, developed primarily in [39, 18, 13], allow the model to be trained effectively from random initialization, thus enabling a meaningful from-scratch baseline. Next, we will discuss how the backbone, which would typically be a ResNet, can be replaced with a Vision Transformer.

2.1. ViT Backbone

In this section we address two technical obstacles when using ViT as the backbone in Mask R-CNN: (1) how to adapt it to work with a feature pyramid network (FPN) [27] and (2) how to reduce its memory footprint and runtime to make benchmarking large ViT backbones tractable.

FPN Compatibility. Mask R-CNN can work with a backbone that either produces a single-scale feature map or feature maps at multiple scales that can be input into an FPN. Since FPN typically provides better detection results with minimal time and memory overhead, we adopt it.

However, using FPN presents a problem because ViT produces feature maps at a single scale (*e.g.*, $1/16$th), in contrast to the multi-scale feature maps produced by typical CNNs.[1] To address this discrepancy, we employ a simple technique from [11] (used for the single-scale XCiT backbone) to either upsample or downsample intermediate ViT feature maps by placing four resolution-modifying modules at equally spaced intervals of $d/4$ transformer blocks, where d is the total number of blocks. See Figure 1 (green blocks).

The first of these modules upsamples the feature map by a factor of 4 using a stride-two 2×2 transposed convolution, followed by group normalization [39] and GeLU [21], and finally another stride-two 2×2 transposed convolution. The next $d/4$th block's output is upsampled by $2\times$ using a single stride-two 2×2 transposed convolution (without normalization and non-linearity). The next $d/4$th block's output is taken as is and the final ViT block's output is downsampled by a factor of two using stride-two 2×2 max pooling. Each of these modules preserves the ViT's embedding/channel dimension. Assuming a patch size of 16, these modules produce feature maps with strides of 4, 8, 16, and 32 pixels, w.r.t. the input image, that are ready to input into an FPN.

We note that recent work, such as Swin [29] and MViT [12], address the single *vs*. multi-scale feature map problem by modifying the core ViT architecture (in pre-training) so it is inherently multi-scale. This is an important direction, but it also complicates the simple ViT design and may impede the exploration of new unsupervised learning directions, such as methods that sparsely process unmasked patches [16]. Therefore, we focus on *external* additions to ViTs that allow them to integrate into multi-scale detection systems. We also note that Beal *et al*. [2] integrate standard ViT models with Faster R-CNN [34], but report substantially lower AP^{box} compared to our results (>10 points lower), which suggests that our design is highly effective.

Reducing Memory and Time Complexity. Using ViT as a backbone in Mask R-CNN introduces memory and runtime challenges. Each self-attention operation in ViT takes $O(h^2w^2)$ space and time for an image tiled (or "patchified") into $h \times w$ non-overlapping patches [38].

During pre-training, this complexity is manageable as $h = w = 14$ is a typical setting (a 224×224 pixel image patchified into 16×16 pixel patches). In object detection, a standard image size is 1024×1024—*approximately* $21\times$ *more pixels and patches*. This higher resolution is needed in order to detect relatively small objects as well as larger ones. Due to the quadratic complexity of self-attention, even the "base" size ViT-B may consume ~20–30GB of GPU memory when used in Mask R-CNN with a *single*-image mini-batch and *half*-precision floating point numbers.

[1] We view the natural 2D spatial arrangement of intermediate ViT patch embeddings as a standard 2D feature map.

To reduce space and time complexity we use restricted (or "windowed") self-attention [38], which saves both space and time by replacing global computation with local computation. We partition the $h \times w$ patchified image into $r \times r$ patch *non-overlapping* windows and compute self-attention *independently* within each of these windows. This windowed self-attention has $O(r^2hw)$ space and time complexity (from $O(r^4)$ per-window complexity and $h/r \times w/r$ windows). We set r to the global self-attention size used in pre-training (*e.g.*, $r = 14$ is typical).

A drawback of windowed self-attention is that the backbone does not integrate information across windows. Therefore we adopt the hybrid approach from [26] that includes four global self-attention blocks placed evenly at each $d/4$th block (these coincide with the up-/downsampling locations used for FPN integration; see Figure 1).

2.2. Upgraded Modules

Relative to the original Mask R-CNN in [19], we modernize several of its modules. Concisely, the modifications include: (1) following the convolutions in FPN with batch normalization (BN) [23], (2) using two convolutional layers in the region proposal network (RPN) [33] instead of one, (3) using four convolutional layers with BN followed by one linear layer for the region-of-interest (RoI) classification and box regression head [39] instead of a two-layer MLP without normalization, (4) and following the convolutions in the standard mask head with BN. Wherever BN is applied, we use *synchronous* BN across all GPUs. These upgrades are implemented in the Detectron2 model zoo.[2]

2.3. Training Formula

We adopt an upgraded training formula compared to the original Mask R-CNN. This formula was developed in [18], which demonstrated good from-scratch performance when training with normalization layers and for long enough, and [13], which demonstrated that a simple data augmentation method called *large-scale jitter* (LSJ) is effective at preventing overfitting and improves results when models are trained for *very* long schedules (*e.g.*, 400 epochs).

We aim to keep the number of hyperparameters low and therefore resist adopting additional data augmentation and regularization techniques. However, we found that drop path regularization [24, 22] is highly effective for ViT backbones and therefore we include it (*e.g.*, it improves from-scratch training by up to 2 AP^{box}).

In summary, we train all models with the same simple formula: LSJ (1024×1024 resolution, scale range $[0.1, 2.0]$), AdamW [30] ($\beta_1, \beta_2 = 0.9, 0.999$) with half-period cosine learning rate decay, linear warmup [15] for 0.25 epochs, and drop path regularization. When using a pre-trained initialization, we fine-tune Mask R-CNN for up to 100 epochs. When training from scratch, we consider schedules of up to 400 epochs since convergence is slower than when using pre-training. We distribute training over 32 or 64 GPUs (NVIDIA V100-32GB) and always use a minibatch size of 64 images. We use PyTorch's automatic mixed precision. Additional hyperparameters are tuned by the consistent application of a protocol, describe next.

2.4. Hyperparameter Tuning Protocol

To adapt the training formula to each model, we tune three hyperparameters—learning rate (*lr*), weight decay (*wd*), and drop path rate (*dp*)—while keeping all others the same for all models. We conducted pilot experiments using ViT-B pre-trained with MoCo v3 to estimate reasonable hyperparameter ranges. Based on these estimates we established the following tuning protocol:

(1) For each initialization (from-scratch, supervised, *etc.*), we fix *dp* at 0.0 and perform a grid search over *lr* and *wd* using ViT-B and a 25 epoch schedule (or 100 epochs when initializing from scratch). We center a 3×3 grid at *lr*, *wd* $= 1.6\text{e}{-4}$, 0.1 and use doubled and halved values around the center. If a local optimum is not found (*i.e.* the best value is a boundary value), we expand the search.

(2) For ViT-B, we select *dp* from $\{0.0, 0.1, 0.2, 0.3\}$ using a 50 epoch schedule for pre-trained initializations. The shorter 25 epoch schedule was unreliable and 100 epochs was deemed impractical. For random initialization we're forced to use 100 epochs due to slow convergence. We found that *dp* $= 0.1$ is optimal for all initializations.

(3) For ViT-L, we adopt the optimal *lr* and *wd* from ViT-B (searching with ViT-L is impractical) and find *dp* $= 0.3$ is best using the same procedure as for ViT-B.

Limitations. The procedure above takes practical shortcuts to reduce the full hyperparameter tuning space. In particular, *lr* and *wd* are optimized separately from *dp*, thus the combination may be suboptimal. Further, we only tune *lr* and *wd* using ViT-B, therefore the choice may be suboptimal for ViT-L. We also tune *lr* and *wd* using a schedule that is $4\times$ shorter than the longest schedule we eventually train at, which again may be suboptimal. Given these limitations we aim to avoid biasing results by applying the *same* tuning protocol to all initializations.

Finally, we note that we tune *lr*, *wd*, and *dp* on the COCO 2017 `val` split and report results on the same split. While technically not an ML best-practice, a multitude of comparisons on COCO `val` *vs.* `test-dev` results over many years demonstrate that overfitting in not a concern for this kind of low-degree-of-freedom hyperparameter tuning.[3]

[2] https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md#new-baselines-using-large-scale-jitter-and-longer-training-schedule

[3] *E.g.*, Table 2 in [29] (version 1) shows that `text-dev` AP^{box} is systematically higher than `val` AP^{box} in seven system-level comparisons.

2.5. Additional Implementation Details

Images are padded during training and inference to form a 1024×1024 resolution input. During training, padding is necessary for batching. During (unbatched) inference, the input only needs to be a multiple of the ViT patch size on each side, which is possibly less than 1024 on one side. However, we found that such reduced padding performs worse (*e.g.*, decrease of ~0.5–1 AP^{box}) than padding to the same resolution used during training, likely due to ViT's use of positional information. Therefore, we use a 1024×1024 resolution input at inference time, even though the extra padding slows inference time by ~30% on average.

3. Initialization Methods

We compare five initialization methods, which we briefly summarize below.

Random: All network weights are randomly initialized and no pre-training is used. The ViT backbone initialization follows the code of [1] and the Mask R-CNN initialization uses the defaults in Detectron2 [40].

Supervised: The ViT backbone is pre-trained for supervised classification using ImageNet-1k images *and* labels. We use the DeiT released weights [36] for ViT-B and the ViT-L weights from [16], which uses an even stronger training formula than DeiT to avoid overfitting (moreover, the DeiT release does not include ViT-L). ViT-B and ViT-L were pre-trained for 300 and 200 epochs, respectively.

MoCo v3: We use the unsupervised ImageNet-1k pre-trained ViT-B and ViT-L weights from the authors of [7] (ViT-B is public; ViT-L was provided via private communication). These models were pre-trained for 300 epochs.

BEiT: Since ImageNet-1k pre-trained weights are not available, we use the official BEiT code release [1] to train ViT-B and ViT-L ourselves for 800 epochs (the default training length used in [1]) on unsupervised ImageNet-1k.

MAE: We use the ViT-B and ViT-L weights pre-trained on unsupervised ImageNet-1k from the authors of [16]. These models were pre-trained for 1600 epochs using *normalized* pixels as the target.

3.1. Nuisance Factors in Pre-training

We attempt to make comparisons as equally matched as possible, yet there are pre-training nuisance factors, listed below, that differ across methods.

(1) Different pre-training methods may use different numbers of epochs. We adopt the default number of pre-training epochs from the respective papers. While these values may not *appear* comparable, the reality is unclear: not all methods may benefit equally from longer training and not all methods have the same per-epoch training cost (*e.g.*, BEiT uses roughly 3× more flops than MAE).

initialization	pre-training data	AP^{box} ViT-B	AP^{box} ViT-L	AP^{mask} ViT-B	AP^{mask} ViT-L
supervised	IN1k w/ labels	47.9	49.3	42.9	43.9
random	*none*	48.9	50.7	43.6	44.9
MoCo v3	IN1k	47.9	49.3	42.7	44.0
BEiT	IN1k+DALL·E	49.8	**53.3**	44.4	47.1
MAE	IN1k	**50.3**	**53.3**	**44.9**	**47.2**

Table 1. **COCO object detection and instance segmentation** using our ViT-based Mask R-CNN baseline. Results are reported on COCO 2017 `val` using the best schedule length (see Figure 2). Random initialization does not use any pre-training data, supervised initialization uses IN1k *with* labels, and all other initializations use IN1k *without* labels. Additionally, BEiT uses a dVAE trained on the proprietary DALL·E dataset of ~250M images [32].

(2) BEiT uses learned relative position biases that are added to the self-attention logits [31] in each block, instead of the absolute position embeddings used by the other methods. To account for this, albeit imperfectly, we include *both* relative position biases and absolute position embeddings in all detection models regardless of their use in pre-training. For BEiT, we transfer the pre-trained biases and randomly initialize the absolute position embeddings. For all other methods, we zero-initialize the relative position biases and transfer the pre-trained absolute position embeddings. Relative position biases are shared across windowed attention blocks and (separately) shared across global attention blocks. When there is a spatial dimension mismatch between pre-training and fine-tuning, we resize the pre-trained parameters to the required fine-tuning resolution.

(3) BEiT makes use of layer scale [37] in pre-training, while the other methods do not. During fine-tuning, the BEiT-initialized model must also be parameterized to use layer scale with the pre-trained layer scaling parameters initialized from the pre-trained model. All other models do not use layer scale in pre-training or in fine-tuning.

(4) We try to standardize pre-training data to ImageNet-1k, however BEiT uses the DALL·E [32] discrete VAE (dVAE), which was trained on ~250 million proprietary and undisclosed images, as an image tokenizer. The impact of this additional training data is not fully understood.

4. Experiments and Analysis

4.1. Comparing Initializations

Results. In Table 1, we compare COCO fine-tuning results using the pre-trained initializations and random initialization described in §3. We show results after maximizing AP^{box} over the considered training lengths: 25, 50, or 100 epochs for pre-trained initializations, and 100, 200, or 400 epochs for random initialization. (We discuss convergence below.) Next, we make several observations.

(1) Our updated Mask R-CNN trains smoothly with ViT-B and ViT-L backbones regardless of the initialization

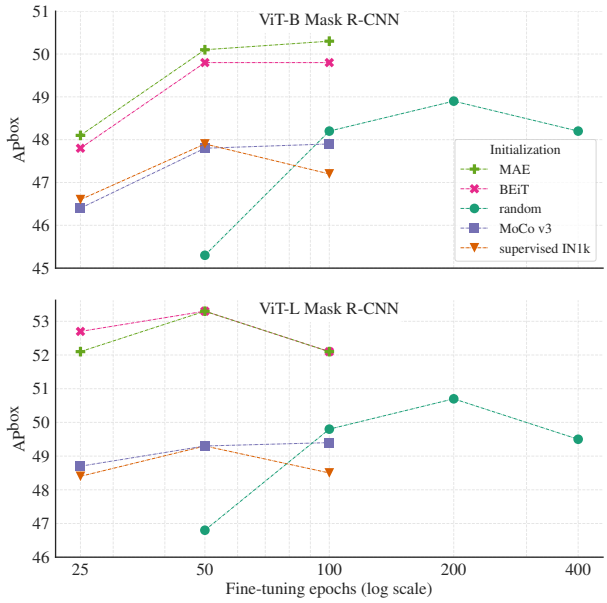

Figure 2. **Impact of fine-tuning epochs.** Convergence plots for fine-tuning from 25 and 400 epochs on COCO. All pre-trained initializations converge much faster (~4×) compared to random initialization, though they achieve varied peak AP^{box}. The performance gap between the masking-based methods (MAE and BEiT) and all others is visually evident. When increasing model scale from ViT-B (top) to ViT-L (bottom), this gap also increases, suggesting that these methods may have superior scaling properties.

method. It does not exhibit instabilities nor does it require stabilizing techniques like gradient clipping.

(2) Training from scratch yields up to 1.4 *higher* AP^{box} than fine-tuning from supervised IN1k pre-training (50.7 *vs*. 49.3). While the higher AP may sound surprising, the same trend is observed in [13]. Supervised pre-training is *not* always a stronger baseline than random initialization.

(3) The contrastive learning-based MoCo v3 underperforms random initialization's AP and has similar results compared to supervised initialization.

(4) For ViT-B, BEiT and MAE outperform both random initialization by up to 1.4 AP^{box} (50.3 *vs*. 48.9) and supervised initialization by up to 2.4 AP^{box} (50.3 *vs*. 47.9).

(5). For ViT-L, the AP^{box} gap *increases*, with BEiT and MAE *substantially* outperforming both random initialization by up to 2.6 AP^{box} (53.3 *vs*. 50.7) and supervised initialization by up to 4.0 AP^{box} (53.3 *vs*. 49.3).

Convergence. In Figure 2 we show how pre-training impacts fine-tuning convergence. Given the tuned hyperparameters for each initialization method, we train models for 2× and 4× longer (and also 0.5× for random initialization). Generally, we find that all pre-trained initializations significantly accelerate convergence compared to random initialization, as observed in [18]. Most methods show signs of overfitting when the training schedule is made sufficiently long, typically by 100 epochs for pre-trained initializations and 400 epochs for random initialization. Based on this data, pre-training tends to accelerate training on COCO by roughly 4× compared to random initialization.

We also note two caveats about these results: (i) The drop path rate should ideally be tuned for each training duration as we have observed that the optimal *dp* value may need to increase when models are trained for longer. (However, performing an exhaustive *dp* sweep for all initializations, model sizes, and training durations is likely computationally impractical.) (ii) Moreover, it may be possible to achieve better results in all cases by training for longer under a more complex training formula that employs heavier regularization and stronger data augmentation.

Discussion. The COCO dataset is a challenging setting for transfer learning. Due to the large training set (~118k images with ~0.9M annotated objects), it is possible to achieve strong results when training from random initialization. We find that existing methods, like supervised IN1k or unsupervised MoCo v3 pre-training, actually *underperform* the AP of the random initialization baseline (though they yield faster convergence). Prior works reporting unsupervised transfer learning improvements on COCO (*e.g*., [17]) tend to show modest gains over supervised pre-training (*e.g*., ~1 AP^{box}) and do not include a strong random initialization baseline as we do here (because strong training formulae based on large-scale jitter had not yet been developed). Moreover, they use weaker models and report results that are overall much lower (*e.g*., ~40 AP^{box}) making it unclear how well the findings translate to state-of-the-art practices.

We find that MAE and BEiT provide the first convincing results of substantial COCO AP improvements due to pre-training. Moreover, these masking-based methods show the potential to improve detection transfer learning as model size increases. We do not observe this important scaling trend with either supervised IN1k pre-training or unsupervised contrastive learning, as represented by MoCo v3.

4.2. Ablations and Analysis

We ablate several factors involved in the system comparison, analyze model complexity, and report tuned hyperparameter values. For these experiments, we use MAE and 50 epoch fine-tuning by default.

Single-scale *vs*. Multi-scale. In Table 2 we compare our default FPN-based multi-scale detector to a single-scale variant. The single-scale variant simply applies RPN and RoIAlign [19] to the final 1/16th resolution feature map generated by the ViT backbone. The RoI heads and all other choices are the same between the systems (in particular, note that both use the same hybrid windowed/global attention). We observe that the multi-scale FPN design in-

FPN	AP^{box} ViT-B	ViT-L
yes	**50.1**	**53.3**
no	48.4	52.0

Table 2. **Single-scale *vs*. multi-scale (FPN) ablation.** FPN yields consistent improvements. Our default setting is marked in gray.

self-attention	act checkpt	AP^{box}	train mem	train time	test time
(1) windowed	no	50.7	16GB	**0.67s**	**0.34s**
(2) windowed, 4 global	no	**53.3**	27GB	0.93s	0.40s
(3) global	yes	53.1	**14GB**	2.26s	0.65s
(4) global	no	-	OOM	-	-

Table 3. **Memory and time reduction strategies.** We compare methods for reducing memory and time when using ViT-L in Mask R-CNN. The strategies include: (1) replace *all* global self-attention with 14×14 non-overlapping windowed self-attention, (2) a hybrid that uses both windowed and global self-attention, or (3) all global attention with activation checkpointing. Without any of these strategies (row 4) an out-of-memory (OOM) error prevents training. We report AP^{box}, peak GPU training memory, average per-iteration training time, and average per-image inference time using NVIDIA V100-32GB GPUs. The per-GPU batch size is 1. Our defaults (row 2) achieves a good balance between memory, time, and AP^{box} metrics. In fact, our hybrid approach achieves comparable AP^{box} to full global attention, while being much faster.

initialization	pre-train (pt) abs	rel	fine-tuning abs	rel	AP^{box} ViT-B	ViT-L
(1) BEiT	no	yes	rand	pt	49.8	**53.3**
(2) BEiT	no	yes	rand	zero	49.5	53.2
(3) BEiT†	yes	no	pt	zero	-	53.1
(4) MAE	yes	no	pt	zero	**50.1**	**53.3**
(5) MAE	yes	no	pt	no	49.9	53.0

Table 4. **Positional information ablation.** In the BEiT code, the ViT is modified to use relative position biases (*rel*) instead of absolute position embeddings (*abs*). We study how these components impact results based on their use in pre-training (*pt*) and under various treatments in fine-tuning: (i) *pt*: initialized with pre-trained values; (ii) *rand*: random initialization; (iii) *zero*: initialized at zero; and (iv) *no*: this positional information is not used in the fine-tuned model. For BEiT† (row 3), we pre-train an additional model (ViT-L only) that, like MAE, uses absolute position embeddings instead of relative position biases. Our default settings are marked in gray. Comparing (1) and (2), we observe that pre-trained relative position bias initialization provides a slight benefit over zero initialization. Comparing (1,2) to (3), we see that BEiT pre-trained with absolute position embeddings performs similarly (perhaps slightly worse) to pre-training with relative position biases. Comparing (4) and (5), we see that including relative position biases in addition to absolute position embeddings provides a small improvement.

creases AP^{box} by ~1.3-1.7 (*e.g.*, 50.1 *vs*. 48.4), while increasing training and inference time by ~5 and ~10% relative, respectively. Multi-scale memory overhead is <1%.

Memory and Time Reduction. In Table 3 we compare several strategies for reducing memory and time complexity when using a standard ViT backbone in Mask R-CNN. Using a combination of 14×14 non-overlapping windowed self-attention together with four global attention blocks achieves a good balance between memory, training and inference time, and AP metrics. This finding motivates us to use this setting as our default. Somewhat surprisingly using only windowed attention is *not* catastrophic even though the backbone processes all windows entirely independently (AP^{box} decreases from 53.3 to 50.7). This is likely due to cross-window computation introduced by convolutions and RoIAlign in the rest of the Mask R-CNN model.

Positional Information. In the default BEiT code, the ViT is modified to use relative position biases [31] in each transformer block instead of adding absolute position embeddings to the patch embeddings. This choice is an orthogonal enhancement that is not used by the other pre-training methods (though it could be). In an attempt to make the comparison more equal, we include these biases (and absolute position embeddings) in all fine-tuning models by default, as discussed in §3.1.

In Table 4 we study the effect of relative position biases on fine-tuning performance. A detailed analysis is given in the caption. In summary, we observe that including relative position biases during fine-tuning may slightly improve AP^{box} by ~0.2–0.3 points (*e.g.*, 53.0 to 53.3) for a model that was pre-trained with only absolute position embeddings. We also observe that pre-training relative position biases, as done by BEiT, may also have a slight positive effect of ~0.1–0.3 points. Our practice of including both positional information types during fine-tuning appears to provide a reasonably fair comparison. We also note that using relative position biases introduces non-trivial overhead—it increases training and inference time by roughly 25% and 15% relative, respectively, increases memory by ~15% (even with shared biases), and perhaps should be avoided.

Pre-training Epochs. In Figure 3 we study the impact of MAE pre-training epochs on COCO AP^{box} by sweeping pre-training epochs from 100 to 1600 (the default). The results show that pre-training duration has a significant impact on transfer learning performance with large increases in AP^{box} continuing from 100 to 800 epochs. There is still a small improvement from 800 to 1600 epochs (+0.2 from 53.1 to 53.3), though the gradient has largely flattened.

TIDE Error Type Analysis. In Figure 4 we show the error type analysis generated by the TIDE toolbox [3]. A detailed description and analysis is given in the caption. The analysis reveals more granular information about *where* MAE and BEiT improve overall AP relative to the other initializations. In summary, we observe that all initializations lead

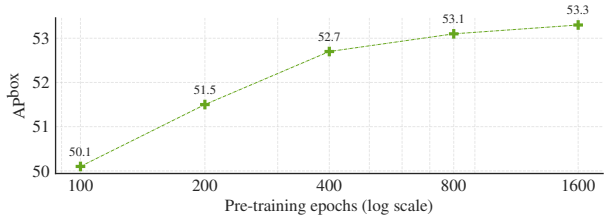

Figure 3. **Impact of pre-training epochs.** Increasing MAE pre-training from 100 to 800 epochs confers large transfer learning gains. The improvements start to plateau after 800 epochs.

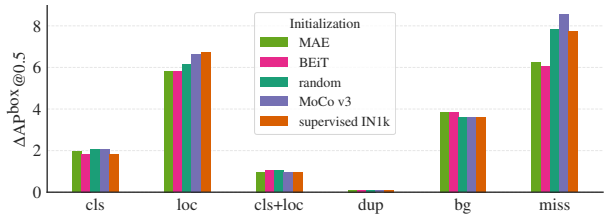

Figure 4. **TIDE analysis.** We plot the ΔAP$^{\text{box}}$ metric at an intersection-over-union (IoU) threshold of 0.5 as defined in [3]. Each bar shows how much AP can be added to the detector if an oracle fixes a certain error type. The error types are: *cls*: localized correctly (IoU $\geq$0.5), but classified incorrectly; *loc*: classified correctly, but localized incorrectly (IoU in $[0.1, 0.5)$); *cls+loc*: classified incorrectly and localized incorrectly; *dup*: detection would be correct if not for a higher scoring correct detection; *bg*: detection is in the background (IoU $<$0.1); *miss*: all undetected ground-truth objects not covered by other error types. (See [3] for more details and discussion.) We observe that the masking-based initializations (MAE and BEiT) make fewer localization errors than MoCo v3 and supervised initialization (random initialization is somewhere in-between) and, even more so, have fewer missed detections. The other error types are more similar across initializations.

to roughly the same classification performance for correctly localized objects, however the MAE and BEiT initializations improve localization compared to the other initializations. We observe an even stronger effect when looking at missed detections: the masking-based initializations yield notably higher recall than the other initializations and thus leave fewer undetected objects. This higher recall creates a small increase in background errors, thus leading to better overall AP.

Model Complexity. Table 5 compares various complexity and wall-clock time measures of our specific Mask R-CNN configuration. We also report these measures using a ResNet-101 backbone instead of ViT. When trained from scratch, both ResNet-101 and ViT-B backbones achieve 48.9 AP$^{\text{box}}$. At inference time, the ResNet-101 backbone is much faster; however, during training ViT-B reaches peak performance at 200 epochs compared to 400 for ResNet-101. ResNet-101 is not yet able to benefit from BEiT or MAE pre-training and therefore lags behind ViT-B in AP$^{\text{box}}$ ($\sim$1 point) when those methods are used for initialization.

backbone	params (M)	acts (M)	flops (G)	fps
ResNet-101	65	426 $\pm$ 43	422 $\pm$ 35	13.7
ViT-B	116	1532 $\pm$ 11	853 $\pm$ 13	5.1
ViT-L	339	2727 $\pm$ 10	1907 $\pm$ 12	2.5

Table 5. **Model complexity for inference** with the specific Mask R-CNN configuration used in this report. For ViT, the image resolution is 1024×1024 (padded as necessary). The flop and activation counts are measured at runtime and vary based on the number of detected objects. We report the mean $\pm$ one standard deviation from 100 validation images. Results change very slightly when using different initializations. For reference, we report results using the ResNet-101 backbone, which can (and does) use non-square inputs at inference time (longest side is 1024); otherwise inference settings are the same. The ResNet-101 based Mask R-CNN achieves 48.9 AP$^{\text{box}}$ when trained from scratch for 400 epochs. We also report wall-clock speed in frames-per-second (fps) on an NVIDIA V100-32GB GPU.

Hyperparameter Tuning. All pre-trained initializations preferred $wd = 0.1$ for fine-tuning. Random initialization benefitted from stronger regularization and selected a higher setting of 0.2. Most methods selected $lr = 8.0\text{e}{-5}$, except for random initialization and MoCo v3 initialization, which both preferred a higher setting of $1.6\text{e}{-4}$. As described previously, the drop path rate could not be reliably tuned using shorter schedules. As a result, we tuned *dp* with 50 epoch training for pre-trained initializations and 100 epoch training for random initialization. Based on this tuning, all initializations selected $dp = 0.1$ when using ViT-B and 0.3 when using ViT-L.

5. Conclusion

We have presented techniques that enable the practical use of standard ViT models as the backbone in Mask R-CNN. These methods yield acceptable training memory and time, while also achieving strong results on COCO without involving too many complex extensions. Using these techniques, we find effective training formulae that enable us to benchmark five different ViT initialization methods. We show that random initialization takes $\sim 4\times$ longer than any of the pre-trained initializations, but achieves a meaningfully higher AP than ImageNet-1k supervised pre-training. We find that MoCo v3, a representative of contrastive unsupervised learning, performs nearly the same as supervised pre-training (and thus worse than random initialization). Importantly, we witness an exciting new result: masking-based methods (BEiT and MAE) show considerable gains over both supervised and random initialization and these gains increase as model size increases. This scaling behavior is not observed with either supervised or MoCo v3-based initialization.

References

[1] Hangbo Bao, Li Dong, and Furu Wei. BEiT: Bert pre-training of image transformers. *arXiv:2106.08254*, 2021.

[2] Josh Beal, Eric Kim, Eric Tzeng, Dong Huk Park, Andrew Zhai, and Dmitry Kislyuk. Toward transformer-based object detection. *arXiv preprint arXiv:2012.09958*, 2020.

[3] Daniel Bolya, Sean Foley, James Hays, and Judy Hoffman. TIDE: A general toolbox for identifying object detection errors. In *ECCV*, 2020.

[4] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *CVPR*, 2018.

[5] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.

[6] 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. In *CVPR*, 2019.

[7] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised Vision Transformers. In *ICCV*, 2021.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[9] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[11] Alaaeldin El-Nouby, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. XCiT: Cross-covariance image transformers. *arXiv preprint arXiv:2106.09681*, 2021.

[12] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. *arXiv preprint arXiv:2104.11227*, 2021.

[13] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *CVPR*, 2021.

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

[15] 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.

[16] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

[17] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

[18] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking ImageNet pre-training. In *ICCV*, 2019.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017.

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

[21] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv:1606.08415*, 2016.

[22] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[24] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *ICLR*, 2016.

[25] 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.

[26] Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. Improved multiscale vision transformers for classification and detection. *In preparation*, 2021.

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

[28] 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.

[29] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[30] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[32] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv:2102.12092*, 2021.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *TPAMI*, 2017.

[35] Pierre Sermanet, Koray Kavukcuoglu, Sandhya Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, 2013.

[36] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv:2012.12877*, 2020.

[37] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[39] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

[40] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.