FastSiam: Resource-Efficient Self-supervised Learning on a Single GPU

Daniel Pototzky[1,2(✉)], Azhar Sultan[1], and Lars Schmidt-Thieme[2]

[1] Robert Bosch GmbH, Hildesheim, Germany
daniel.pototzky@de.bosch.com
[2] Information Systems and Machine Learning Lab, University of Hildesheim, Hildesheim, Germany

Abstract. Self-supervised pretraining has shown impressive performance in recent years, matching or even outperforming ImageNet weights on a broad range of downstream tasks. Unfortunately, existing methods require massive amounts of computing power with large batch sizes and batch norm statistics synchronized across multiple GPUs. This effectively excludes substantial parts of the computer vision community from the benefits of self-supervised learning who do not have access to extensive computing resources.

To address that, we develop FastSiam with the aim of matching ImageNet weights given as little computing power as possible. We find that a core weakness of previous methods like SimSiam is that they compute the training target based on a single augmented crop (or "view"), leading to target instability. We show that by using multiple views per image instead of one, the training target can be stabilized, allowing for faster convergence and substantially reduced runtime. We evaluate FastSiam on multiple challenging downstream tasks including object detection, instance segmentation and keypoint detection and find that it matches ImageNet weights after 25 epochs of pretraining on a single GPU with a batch size of only 32.

Keywords: Resource-efficient · Self-supervised learning

1 Introduction

Self-supervised pretraining aims at learning transferable representations given large amounts of unlabeled images. The resulting weights can be used as initialization for finetuning on a small, labeled dataset.

In recent years, methods that compare views of images have increasingly attracted attention [3,6,8,17,18]. The underlying idea is that even if it is unknown what kind of object is depicted on an image, two augmented crops of

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-16788-1_4.

B. Andres et al. (Eds.): DAGM GCPR 2022, LNCS 13485, pp. 53–67, 2022.
https://doi.org/10.1007/978-3-031-16788-1_4

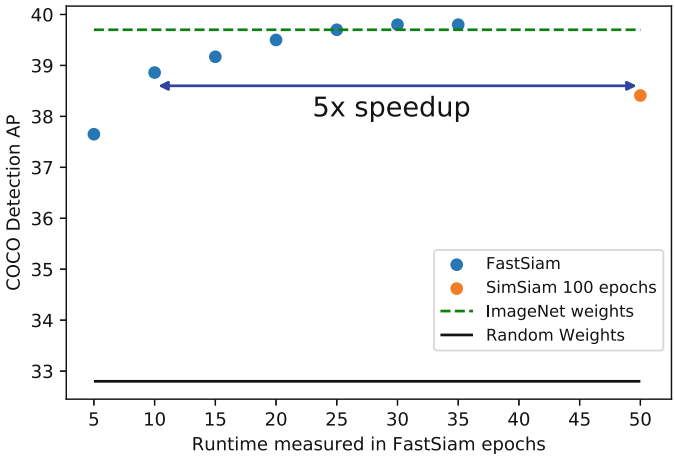


Fig. 1. Finetuning pretrained weights for object detection on MS COCO [22] following the 1x evaluation protocol with a Mask R-CNN and FPN-backbone as first described in MoCo [18]. FastSiam matches ImageNet weights if trained with a batch size of 32 on a single GPU for 25 epochs. FastSiam uses 4 views per image whereas SimSiam uses 2. Because of that FastSiam requires twice as long per epoch than SimSiam does but overall runtime to match SimSiam is still cut by a factor of more than 5.

it should still result in similar features. State-of-the-art methods based on this idea such as BYOL [17] and SwAV [3] have been shown to match or even outperform ImageNet weights. These approaches are typically trained with large batch sizes (i.e. up to 4096) and batch-norm statistics synchronized across multiple GPUs for up to 1000 epochs. Just reproducing them requires massive amounts of computing power. We argue that the dependence on large batch sizes and long training schedules effectively excludes large parts of the computer vision community from the benefits of self-supervised learning.

To understand why conventional self-supervised methods are very compute-intensive, compare them to supervised learning. The training targets in supervised learning are determined by the respective labels for each image that do not change throughout training. Conversely, the training targets of self-supervised pretraining methods are usually computed on the fly and change throughout training, depending on many factors including which crops are sampled for target computation. This property of ever-changing training targets slows down convergence.

In addition, self-supervised methods typically require large batch sizes to work well. One reason for this is that some methods contrast a positive view with another positive view from the same image and thousands of negative views from other images [6]. Only if many negative examples are included in a batch, the task of distinguishing views from the same image from those of other images gets sufficiently difficult. Even methods that do not use negative examples explicitly like BYOL [17] observe a decrease in performance if the batch size is small.

In contrast, SimSiam [8], which also does not use negative examples, is somewhat less dependent on large batches and works best with a batch size of 256. Our method FastSiam improves upon the shortcomings of SimSiam, in particular, by stabilizing the training target using multiple views per image instead of one. The suggested changes allow for training with smaller batch sizes, i.e. less unique images (32 vs. 256) and fewer views (128 vs. 512), and lead to much faster convergence and reduced computing requirements (see Fig. 1).

We show in experiments that FastSiam matches ImageNet weights on a variety of downstream tasks including object detection, instance segmentation and keypoint detection after 25 epochs of self-supervised pretraining on a single GPU with a batch size of only 32. We believe these findings to be very relevant for a large number of computer vision researchers who have only limited access to computing resources. FastSiam may allow them to benefit from recent progress in self-supervised pretraining. Overall, our contributions can be summarized as follows:

- We propose FastSiam, a self-supervised pretraining method that works best with small batch sizes and that converges much faster than competing methods.
- We identify target instability as a core reason why existing methods require very long training schedules and large batch sizes. We show that by stabilizing the training target, convergence can be reached much faster and a batch size of only 32 is found to be best.
- Weights trained by FastSiam for only 25 epochs on a single GPU match or improve upon ImageNet weights on a variety of downstream tasks including object detection, instance segmentation and keypoint detection.

2 Related Work

Self-supervised pretraining attempts to learn transferable image representations by only having access to a large number of unlabeled images. Early methods in the field focus on handcrafted pretext tasks including colorization [20], rotation prediction [16], clustering [2], solving jigsaw puzzles [24] and others [14].

In recent years, approaches that compare views of images have gained attention [3,6–8,10,15,17,18,33]. Some of these methods are called contrastive methods because they compare views of the same image with those of other images, either maximizing or minimizing feature similarity [6,7,10,15,18,33]. Among them, SimCLR [6] shows that contrastive methods benefit from strong data augmentation, large batch sizes and many epochs of training. Different from contrastive methods, BYOL [17] and SimSiam [8] find that using negative examples is not even necessary if an asymmetric architecture, as well as a momentum encoder or a stop-gradient operation, are applied. Moreover, SwAV [3] uses an online clustering of output features and enforces consistency between cluster assignments. Instead of contrasting per-image features, some methods compare class predictions [4].

One property of all these approaches is that they compute a global feature vector or a per-image class prediction. Conversely, other self-supervised methods are specifically designed for object detection or segmentation downstream tasks by comparing local patches [5,13,23,28,31] or even pixels [25] across views instead of entire images.

Recently, self-supervised methods that were originally developed for convolutional neural networks have also been applied to transformers [10,32] and vice versa [4]. Furthermore, some approaches have been developed which make use of transformer-specific properties [1,12,34].

While the vast majority of methods that compare views of images use two of them, some approaches include more than that [3,4]. The clustering-based approach SwAV [3], in particular, combines two views of the normal resolution with multiple smaller ones, resulting in an increase in performance. However, SwAV does not observe a speedup of convergence.

Typically, self-supervised methods are trained between 200 and 1000 epochs with batch sizes of up to 4096. While several methods report a strong decrease in performance given smaller batch sizes [17], others show only a moderate decline. For example, the performance of SwAV [3] only decreases somewhat if the batch size is reduced from 4096 to 256 unique images (with 6 to 8 views per image, some of which are of lower resolution). Different from competing methods, MoCov2 [7] is trained with a batch size of 256 by default, which is made possible by keeping a large queue of negative samples. By doing so, MoCov2 decouples the batch size from the number of negative instances used in the loss. Moreover, DINO [4] shows only a moderate decline in performance with a batch size of 128 when evaluating on a k-NN classification task. In addition, SimSiam [8], DenseCL [28] and DetCo [31] use a batch size of 256 by default.

Overall, some methods are moderately robust to small batch sizes, but they still observe a substantial decrease in performance for batch sizes lower than 128. No method matches ImageNet weights with a very short training schedule on a single GPU while only requiring a small batch size. This essentially means that there is no self-supervised method that is suited for computer vision researchers who have limited access to computing power. FastSiam addresses this issue and matches ImageNet weights after only 25 epochs of training on a single GPU with batch size as low as 32.

3 Method

In this section, we describe the core properties of FastSiam, which allow for fast convergence and training with small batch sizes on a single GPU. FastSiam extends upon SimSiam and addresses a core weakness of it, namely target instability.

3.1 Background

SimSiam inputs two augmented crops of an image, $x_1 \in \mathbb{R}^{H \times W \times C}$ and $x_2 \in \mathbb{R}^{H \times W \times C}$. The view x_1 is fed into the prediction branch which consists of an

encoder network f (e.g. a ResNet50 with an MLP on top) and a predictor head h (e.g. another MLP), resulting in prediction $p_1 \triangleq h(f(x_1))$. The target branch computes a vector that serves as a target by processing x_2 through the same encoder f that the prediction branch also uses, i.e. $z_2 \triangleq f(x_2)$. Gradients are not backpropagated on the target branch as a stop-gradient operation is applied. Overall, the network is trained to make the output features from the prediction branch similar to those of the target branch by minimizing their negative cosine similarity (see Eq. 1).

$$L(p_1, z_2) = -\frac{p_1}{\|p_1\|_2} \cdot \frac{z_2}{\|z_2\|_2} \tag{1}$$

3.2 Stabilizing the Training Target

SimSiam and other methods that compare views of images implicitly assume that the sampled views show at least parts of the same object. However, this is often not the case when applying the usual cropping strategy to object-centric images. Maximizing the feature similarity of views that show different objects is arguably not an ideal training strategy and may lead to training instability. Furthermore, even if both views show parts of the same object, target vectors may differ strongly depending on which views are sampled and how they get augmented.

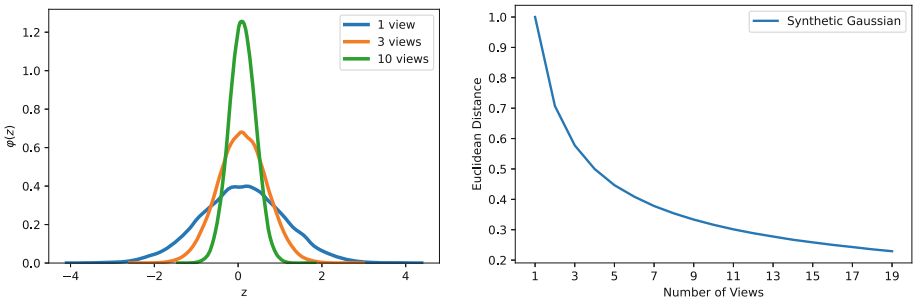
Computing the target vector for all possible views of an image results in a multivariate distribution. Assume, for simplicity, that this distribution has the mean μ and the variance Σ. We argue that SimSiam and other self-supervised methods suffer from target instability because they sample only one instance T_1 from this distribution (see Eq. 2). After all, a target vector generated from a single view may differ substantially from the overall properties of the distribution like the mean value, leading to target instability.

$$\mathbb{E}(T_1) = \mu, \quad Var(T_1) = \Sigma \tag{2}$$

To address this issue, we propose to use multiple views for target computation to get a more stable estimate of the mean value. This can be seen as a way of avoiding outliers to determine the training target. Sampling and averaging K independent and identically distributed target vectors is equivalent to sampling from a distribution with the same mean but reduced variance (see Eq. 3). Increasing the number of samples by a factor of four reduces the variance by a factor of four and the resulting standard error of the mean by a factor of two if samples are uncorrelated. More details on this connection are included in the supplementary material.

$$\mathbb{E}(\bar{T}_K) = \mu, \quad Var(\bar{T}_K) = \frac{\Sigma}{K} \tag{3}$$

Figure 2a depicts the distribution of the first dimension of a 2048-dimensional vector in the case of varying amounts of samples K that were drawn from $\mathcal{N}(0, I)$.



(a) (b)

Fig. 2. Target statistics for varying numbers of sampled views if the underlying distribution is Gaussian. The more views are used for target computation, the sharper the distribution of target vectors centers around its mean (a), and the smaller the average Euclidean distance to the target vector computed based on all possible views becomes (b). For visualization purposes, we only plot results for the first of 2048 dimensions of the target vector in (a).

The more views are used for target computation, the sharper the resulting distribution centers around its mean. Based on that, Fig. 2b shows the decreasing average Euclidean distance between the sampled target and the mean of using all possible views. If few views are provided, adding one more results in a strong decrease. Conversely, if already many views are available, including another one has only a small effect.

FastSiam makes use of the connection that averaging multiple samples reduces the standard error of the mean (see Fig. 3). By doing so, the training target becomes much more stable, allowing for faster convergence and training with smaller batch sizes both in terms of unique images and the total number of views. The resulting loss function in which multiple views are combined for target generation is shown in Eq. 4.

$$L(z_1, z_2, z_3, ..., z_K, p_{K+1}) = -\frac{p_{K+1}}{\|p_{K+1}\|_2} \cdot \frac{\frac{1}{K}\sum_{i=1}^{K} z_i}{\|\frac{1}{K}\sum_{i=1}^{K} z_i\|_2} = -\frac{p_{K+1}}{\|p_{K+1}\|_2} \cdot \frac{\bar{T}_K}{\|\bar{T}_K\|_2} \tag{4}$$

The total loss that is optimized is formulated in Eq. 5. Each view is passed through the prediction branch once and the remaining views are used for target computation. We find $K = 3$ to be best, meaning that the combination of three views is used for target computation.

$$L_{total} = \sum_{i=1}^{K+1} \frac{1}{K+1} L(\{z_j | j \neq i \wedge 1 \leq j \leq K+1\}, p_i) \tag{5}$$

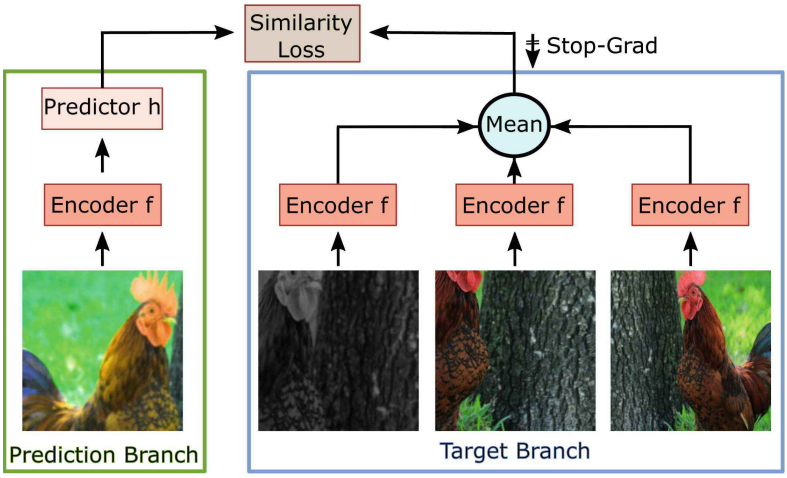


Fig. 3. Overall setup of FastSiam. On the target branch, FastSiam combines the information from multiple views to increase target stability. Gradients are not backpropagated, meaning that the target vector can be seen as a constant. On the prediction branch, only one view is used. The resulting prediction is trained to be similar to the output of the target branch. By increasing target stability, convergence can be reached much faster and with smaller batch sizes both in terms of unique images and the total number of views.

4 Experiments

The overall experimental setup is divided into two parts. First, FastSiam is pretrained without labels. Second, the resulting weights are evaluated by finetuning them on a diverse set of downstream tasks. These include object detection, instance segmentation and keypoint detection on MS COCO [22] and CityScapes [11]. We show that FastSiam matches or exceeds the performance of ImageNet weights and is competitive with other self-supervised methods while requiring much less computing power.

4.1 Experimental Setup

Pretraining Setup. For self-supervised pretraining we use ImageNet [26], which contains 1.2 million images from 1000 classes. We follow the training setup in SimSiam [8] including, for example, the augmentation pipeline. By default, we train for 25 epochs with a cosine learning rate decay schedule. The learning rate is set to 0.125 for a batch size of 32. In the case of larger batch sizes, the learning rate is scaled linearly. We use three views per image for target computation.

Evaluation Protocols. We evaluate the pretrained weights on a variety of downstream tasks using common protocols that are implemented in Detectron2 [29]. We adopt two different protocols for evaluating object detection and

instance segmentation on MS COCO. Following DenseCL [28] and InfoMin [27] we use a Mask R-CNN detector [19] with a FPN backbone [21] and the 1× schedule with 90,000 steps for evaluating object detection and instance segmentation on MS COCO. Furthermore, like in MoCo [18] and DetCo [31] we use a ResNet50-C4 backbone [19] in combination with a Mask R-CNN and the 1× schedule. Moreover, we use the evaluation protocol for instance segmentation on CityScapes [11], a dataset covering urban street scenes for autonomous driving, from MoCo [18] and DetCo [31]. In addition, we also evaluate keypoint detection on MS COCO again following MoCo [18] and DetCo [31]. SimSiam weights after 100 epochs were downloaded from the author's github website [9] and used in evaluations.

Table 1. Object Detection and Segmentation on MS COCO with a ResNet50-FPN Backbone. FastSiam matches ImageNet weights and performs comparably to other self-supervised methods.

Method	Epoch	Batch	Views	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Random	-	-	-	32.8	50.9	35.3	29.9	47.9	32.0
ImageNet	90	256	1	39.7	59.5	43.3	35.9	56.6	38.6
InsDis [30]	200	256	1	37.4	57.6	40.6	34.1	54.6	36.4
MoCo [18]	200	256	2	38.5	58.9	42.0	35.1	55.9	37.7
MoCov2 [7]	200	256	2	38.9	59.4	42.4	35.5	56.5	38.1
SwAV [3]	200	4096	8	38.5	60.4	41.4	35.4	57.0	37.7
DetCo [31]	200	256	20	40.1	61.0	43.9	36.4	58.0	38.9
SimCLR [6]	200	4096	2	38.5	58.0	42.0	34.8	55.2	37.2
DenseCL [28]	200	256	2	40.3	59.9	44.3	36.4	57.0	39.2
BYOL [17]	200	4096	2	38.4	57.9	41.9	34.9	55.3	37.5
SimSiam [8]	100	256	2	38.4	57.5	42.2	34.7	54.9	37.1
FastSiam	10	32	4	38.9	58.3	42.6	35.2	55.5	37.9
FastSiam	25	32	4	39.7	59.4	43.5	35.7	56.5	38.2

4.2 Main Results

Object Detection and Instance Segmentation on MS COCO. Table 1 shows the results on MS COCO for object detection and instance segmentation using a ResNet50-FPN backbone. FastSiam trained for 25 epochs with a batch size of 32 results in comparable performance to state-of-the-art self-supervised methods and also matches ImageNet weights. Even if FastSiam is only trained for 10 epochs, performance is comparable to many other self-supervised methods including SimSiam trained for 100 epochs while greatly outperforming a random initialization. Furthermore, in Table 2 we report results on MS COCO with a

ResNet-C4 backbone. If FastSiam is trained for 25 epochs, performance matches ImageNet weights and most other self-supervised methods. Notably, FastSiam clearly outperforms SimSiam trained for 100 epochs and almost matches the performance of it trained for 200 epochs. In the case of training FastSiam for only 10 epochs, performance remains competitive with other self-supervised methods while greatly improving upon a random initialization.

Table 2. Object Detection and Instance Segmentation on MS COCO with a ResNet50-C4 Backbone. FastSiam matches ImageNet weights and performs comparably to other self-supervised methods

Method	Epoch	Batch	Views	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Random	-	-	-	26.4	44.0	27.8	29.3	46.9	30.8
ImageNet	90	256	1	38.2	58.2	41.2	33.3	54.7	35.2
InsDis [30]	200	256	1	37.7	57.0	40.9	33.0	54.1	35.2
MoCo [18]	200	256	2	38.5	58.3	41.6	33.6	54.8	35.6
MoCov2 [7]	200	256	2	38.9	58.4	42.0	34.2	55.2	36.5
SwAV [3]	200	4096	8	32.9	54.3	34.5	29.5	50.4	30.4
DetCo [31]	200	256	20	39.8	59.7	43.0	34.7	56.3	36.7
SimSiam [8]	200	256	2	39.2	59.3	42.1	34.4	56.0	36.7
SimSiam [8]	100	256	2	35.8	54.5	38.7	31.7	51.5	33.7
FastSiam	10	32	4	37.3	56.9	40.2	32.8	53.8	34.7
FastSiam	25	32	4	38.5	58.0	41.6	33.7	54.8	35.8

Instance Segmentation on CityScapes. Table 3 shows results for instance segmentation on CityScapes. FastSiam trained for 25 epochs outperforms ImageNet weights and is competitive with other self-supervised methods while being trained for fewer epochs and with much smaller batch size. Even when training FastSiam for only 10 epochs, downstream performance almost matches ImageNet weights while being clearly superior to a random initialization.

Keypoint Detection on MS COCO. In Table 4 we report results for keypoint detection on MS COCO. FastSiam trained for only 10 epochs improves upon ImageNet weights and performs comparably to other self-supervised methods including SimSiam while requiring much less computing power. If FastSiam is trained for 25 epochs, detection performance is only slightly better than after 10 epochs.

Runtime Analysis. In this section, we compare the runtime of FastSiam with that of SimSiam. In particular, we investigate how much runtime by FastSiam is

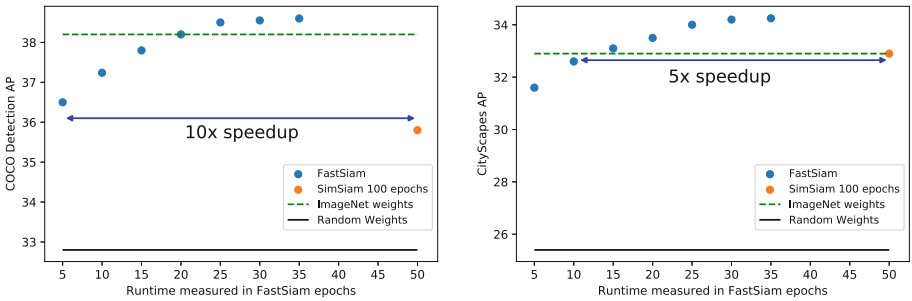
Table 3. Instance Segmentation on CityScapes. FastSiam matches competing methods and improves upon ImageNet weights.

Method	Epoch	Batch	Views	AP^{mk}	AP^{mk}_{50}
Random	-	-	-	25.4	51.1
ImageNet	90	256	1	32.9	59.6
InsDis [30]	200	256	1	33.0	60.1
MoCo [18]	200	256	2	32.3	59.3
MoCov2 [7]	200	256	2	33.9	60.8
SwAV [3]	200	4096	8	33.9	62.4
DetCo [31]	200	256	20	34.7	63.2
SimSiam [8]	100	256	2	32.9	60.9
FastSiam	10	32	4	32.6	59.9
FastSiam	25	32	4	34.0	60.7

Table 4. Keypoint Detection on MS COCO. FastSiam matches competing methods and outperforms ImageNet weights.

Method	Epoch	Batch	Views	AP^{kp}	AP^{kp}_{50}	AP^{kp}_{75}
Random	-	-	-	65.9	86.5	71.7
ImageNet	90	256	1	65.8	86.9	71.9
InsDis [30]	200	256	1	66.5	87.1	72.6
MoCo [18]	200	256	2	66.8	87.4	72.5
MoCov2 [7]	200	256	2	66.8	87.3	73.1
SwAV [3]	200	4096	8	66.0	86.9	71.5
DetCo [31]	200	256	20	67.2	87.5	73.4
SimSiam [8]	100	256	2	66.4	87.1	72.2
FastSiam	10	32	4	66.5	87.2	72.4
FastSiam	25	32	4	66.6	87.2	72.6

needed to match the performance of SimSiam and supervised ImageNet weights. Figure 4 shows results for AP^{bb} for object detection on MS COCO and AP^{mk} for instance segmentation on CityScapes. FastSiam requires almost exactly twice as much time per epoch as SimSiam because it uses four views per image instead of two. Therefore, four forward and backward paths are computed instead of two. Test are conducted on NVIDIA GeForce RTX 2080 Ti. Because time measurements in GPU hours depend on the specific hardware, we report the relative runtime measured in equivalents of FastSiam epochs. In terms of computing requirements, 100 epochs of SimSiam are equivalent to 50 FastSiam epochs. However, the total runtime of FastSiam is between 5 and 10 times lower because training converges significantly faster due to increased target stability.



(a) Object Detection on MS COCO with a ResNet50-C4 Backbone.

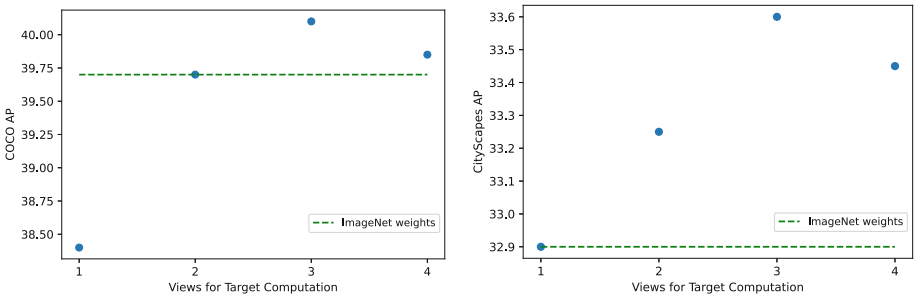
(b) Instance Segmentation on CityScapes with a Mask RCNN.

Fig. 4. Runtime comparison between FastSiam and SimSiam. Performance of pretrained weights is evaluated on object detection and instance segmentation. FastSiam reaches the same downstream performance as SimSiam with a speedup between factor 5 and 10.

4.3 Ablations

The Optimal Number of Views. We investigate what number of views is optimal for target computation. Figure 5 shows downstream performance for object detection on MS COCO and instance segmentation on CityScapes given a varying number of views for target computation, ranging from one to four. To ensure a fair comparison, we use a fixed amount of computing power (equivalent to 50 epochs in case of three views) and the same number of views per batch (512). If only one view is used like in SimSiam, the training target is unstable. Conversely, one could use as many views as fit in the batch taken from a single image. While this would maximize target stability, the diversity within each batch would be very low. We find that using three views for target computation offers the best trade-off and results in the highest transfer performance, hence we choose it as our default.

Effect of Batch Size on Performance. We investigate the effect that the chosen batch size in FastSiam has on the downstream performance. In Table 5 we compare AP^{bb} for object detection on MS COCO, AP^{mk} for instance segmentation on CityScapes and AP^{kp} for keypoint detection on MS COCO given different batch sizes in pretraining. We find that increasing batch size from the default of 32 to 64 or 128 does not lead to better downstream performance. Whereas other self-supervised methods [17] report that reducing the batch size to 256 or below decreases performance, FastSiam even works best with a batch size of 32. Reducing the batch size below 32 has no practical value because it already fits on a single GPU.



(a) Detection on MS COCO with FPN. (b) Instance Segmentation on CityScapes.

Fig. 5. The downstream performance for varying numbers of views used for target computation, three being the default in FastSiam. Total compute as well as the number of views per batch is the same for all settings.

Table 5. Effect of Batch Size on Downstream Performance. We compare AP^{bb} for object detection on MS COCO, AP^{mk} for instance segmentation on CityScapes and AP^{kp} for keypoint detection on MS COCO given different batch sizes in pretraining.

Method	Epoch	Batch	COCO FPN	COCO C4	CityScapes	Keypoint
Random	-	-	32.8	26.4	25.4	65.9
ImageNet	90	256	39.7	38.2	32.9	65.8
FastSiam	25	32	39.7	38.5	34.0	66.6
FastSiam	25	64	39.7	38.4	33.6	66.6
FastSiam	25	128	39.8	38.5	33.3	66.6

Activation Maps. We visualize activation maps for FastSiam trained for 25 epochs and SimSiam trained for 100 epochs (see Fig. 6). FastSiam activates more precise object regions than SimSiam does. This property is particularly important for downstream tasks that involve localization.

Qualitative Analysis. In this section, we qualitatively analyze which views of images result in target vectors that are either close to the mean vector or very far away from it. For this, we input 200 views for each randomly selected image in an untrained encoder network. We then determine the views corresponding to the three most similar and three most distant target vectors relative to the mean vector. The result is shown in Fig. 7. The first column depicts the original image. Columns 2–4 show the views corresponding to the three target vectors with the smallest difference to the mean. The remaining columns 5–7 show the three views which lead to the most distant target vectors. These outlier vectors result from views that either do not contain the object of interest (e.g. a dog) or are very heavily augmented (e.g. strong color jitter or conversion to greyscale).

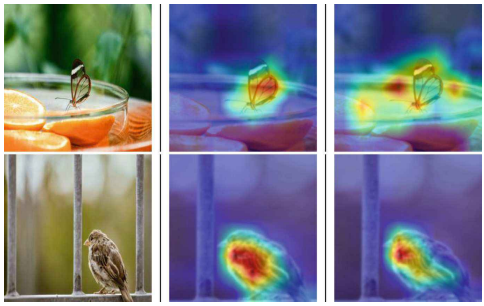


Fig. 6. Activation maps for FastSiam (middle column) and SimSiam (right column). FastSiam generates more precise object regions than SimSiam does.

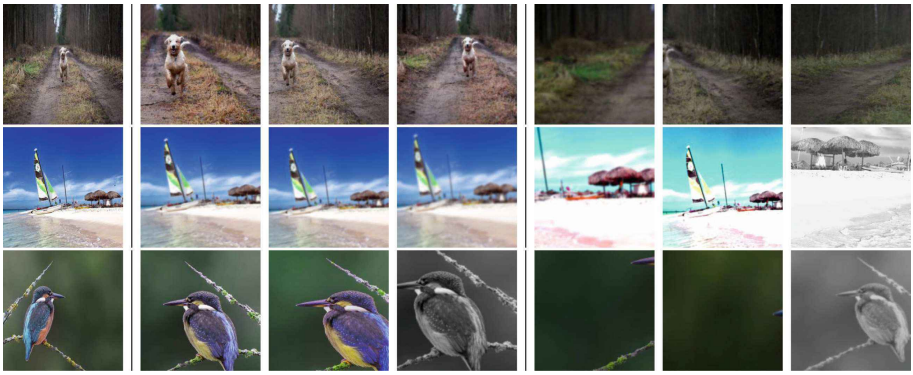


Fig. 7. Views of images that result in target vectors that are closest or most distant to the mean target. Column 1 shows the original image, columns 2–4 the three views which result in target vectors with the smallest distance to the mean, whereas columns 5–7 shows the three views that result in the most distant target vectors. Views corresponding to outlier vectors often do not contain the object of interest (e.g. a dog) or are heavily augmented.

5 Conclusion

In this work, we propose FastSiam, a self-supervised pretraining method that is optimized for fast convergence, small batch sizes and low computing requirements. On a diverse set of downstream tasks including object detection, instance segmentation and keypoint detection, FastSiam matches ImageNet weights and performs comparably to other self-supervised methods while requiring much less computing power. We believe that FastSiam is of particular interest for computer vision researchers who have only limited access to computing resources, finally allowing them to benefit from self-supervised pretraining.

References

1. Bar, A., et al.: Detreg: unsupervised pretraining with region priors for object detection (2021). arXiv:2106.04550
2. Caron, M., Bojanowski, P., Joulin, A., Douze, M.: Deep clustering for unsupervised learning of visual features. In: Proceedings of the European Conference on Computer Vision (ECCV) (2018)
3. Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., Joulin, A.: Unsupervised learning of visual features by contrasting cluster assignments. In: Advances in Neural Information Processing Systems, vol. 33, pp. 9912–9924 (2020)
4. Caron, M., et al.: Emerging properties in self-supervised vision transformers (2021). arXiv:2104.14294
5. Chen, K., Hong, L., Xu, H., Li, Z., Yeung, D.Y.: Multisiam: self-supervised multi-instance siamese representation learning for autonomous driving. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 7546–7554 (2021)
6. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 1597–1607 (2020)
7. Chen, X., Fan, H., Girshick, R., He, K.: Improved baselines with momentum contrastive learning (2020). arXiv:2003.04297
8. Chen, X., He, K.: Exploring simple siamese representation learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 15750–15758 (2021)
9. Chen, X., He, K.: Simsiam: exploring simple siamese representation learning (2021). https://github.com/facebookresearch/simsiam
10. Chen, X., Xie, S., He, K.: An empirical study of training self-supervised vision transformers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 9640–9649 (2021)
11. Cordts, M., et al.: The cityscapes dataset for semantic urban scene understanding (2016). arXiv:1604.01685
12. Dai, Z., Cai, B., Lin, Y., Chen, J.: UP-DETR: unsupervised pre-training for object detection with transformers. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1601–1610 (2021)
13. Ding, J., et al.: Unsupervised pretraining for object detection by patch reidentification (2021). arXiv:2103.04814
14. Doersch, C., Gupta, A., Efros, A.A.: Unsupervised visual representation learning by context prediction. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2015)
15. Gidaris, S., Bursuc, A., Puy, G., Komodakis, N., Cord, M., Perez, P.: Obow: online bag-of-visual-words generation for self-supervised learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6830–6840 (2021)
16. Gidaris, S., Singh, P., Komodakis, N.: Unsupervised representation learning by predicting image rotations. In: International Conference on Learning Representations ICLR (2018)
17. Grill, J.B., et al.: Bootstrap your own latent: a new approach to self-supervised learning (2020). arXiv:2006.07733

18. He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
19. He, K., Gkioxari, G., Dollar, P., Girshick, R.: Mask R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), October 2017
20. Larsson, G., Maire, M., Shakhnarovich, G.: Colorization as a proxy task for visual understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
21. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection (2017). arXiv:1612.03144
22. Lin, T.Y., et al.: Microsoft coco: common objects in context (2015). arXiv:1405.0312
23. Liu, S., Li, Z., Sun, J.: Self-EMD: self-supervised object detection without imagenet. CoRR (2020). arXiv:2011.13677
24. Noroozi, M., Favaro, P.: Unsupervised learning of visual representations by solving jigsaw puzzles. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9910, pp. 69–84. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46466-4_5
25. Pinheiro, P., Almahairi, A., Benmalek, R., Golemo, F., Courville, A.: Unsupervised learning of dense visual representations. In: Advances in Neural Information Processing Systems, vol. 33, pp. 4489–4500 (2020)
26. Russakovsky, O., et al.: Imagenet large scale visual recognition challenge (2015). arXiv:1409.0575
27. Tian, Y., Sun, C., Poole, B., Krishnan, D., Schmid, C., Isola, P.: What makes for good views for contrastive learning? In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, pp. 6827–6839. Curran Associates, Inc. (2020)
28. Wang, X., Zhang, R., Shen, C., Kong, T., Li, L.: Dense contrastive learning for self-supervised visual pre-training. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3024–3033 (2021)
29. Wu, Y., Kirillov, A., Massa, F., Lo, W.Y., Girshick, R.: Detectron2 (2019). https://github.com/facebookresearch/detectron2
30. Wu, Z., Xiong, Y., Yu, S.X., Lin, D.: Unsupervised feature learning via non-parametric instance discrimination. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018
31. Xie, E., et al.: DetCo: unsupervised contrastive learning for object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 8392–8401 (2021)
32. Xie, Z., et al.: Self-supervised learning with swin transformers (2021). arXiv:2105.04553
33. Zbontar, J., Jing, L., Misra, I., LeCun, Y., Deny, S.: Barlow twins: self-supervised learning via redundancy reduction. In: International Conference on Machine Learning (ICML) (2021)
34. Zhou, J., et al.: iBOT: image BERT pre-training with online tokenizer (2021). arXiv:2111.07832