


Bootstrapped Masked Autoencoders for Vision BERT Pretraining

Xiaoyi Dong^{1*}, Jianmin Bao², Ting Zhang², Dongdong Chen^{3†},
Weiming Zhang¹, Lu Yuan³, Dong Chen², Fang Wen², Nenghai Yu¹

¹University of Science and Technology of China

²Microsoft Research Asia ³Microsoft Cloud + AI

{dlight@mail., zhangwm@, ynh@}.ustc.edu.cn cddlyf@gmail.com

{jianbao, Ting.Zhang, luyuan, doch, fangwen}@microsoft.com

1 Cross attention in Regressor/Predictor

One core design of our BootMAE is decoupling the target-specific context from the encoder, *i.e.* provide low-level feature for the pixel regressor and high-level feature for the feature predictor. The feature injection procedure is conducted by a cross-attention operation. Formally, the cross-attention operator of regressor can be formulated as follows:

$$\begin{aligned}\text{CrossAttention}(Q, K, V) &= \text{softmax}\left(\frac{QK^T}{\sqrt{d_{\text{decoder}}}}\right)V \\ Q &= \mathbf{Z}_v W_Q \\ K &= \mathbf{Z}_v^{\text{shallow}} W_K \\ V &= \mathbf{Z}_v^{\text{shallow}} W_V\end{aligned}\tag{1}$$

$$\mathbf{Z}'_v = \mathbf{Z}_v + \text{CrossAttention}(Q, K, V)\tag{2}$$

here $W_Q \in \mathbb{R}^{d_{\text{decoder}} \times d_{\text{decoder}}}$ project the input feature \mathbf{Z}_v into the queue Q with dimension d_{decoder} , and $W_K \in \mathbb{R}^{d_{\text{decoder}} \times d_{\text{decoder}}}$ and $W_V \in \mathbb{R}^{d_{\text{decoder}} \times d_{\text{decoder}}}$ project the injection feature $\mathbf{Z}_v^{\text{shallow}}$ into the key K and value V . The $\mathbf{Z}_v^{\text{shallow}}$ can be replaced with $\mathbf{Z}_v^{\text{deep}}$ for the formulation of cross-attention in predictor. With such cross-attention, we provide context information to decoder and relieve the encoder from “memorizing” such context.

In our experiment, we use the output feature of the first encoder block as the low-level feature and the output feature after the last encoder block as the high-level feature. Such a simple strategy could be applied to models with different number of layers directly. Here we conduct some ablation to study how the injected feature affects the pretraining performance.

As shown in Table 1, for ViT-B with 12 blocks, we provide low-level feature (output of the first block), middle-level feature (output of the 6_{th} block), and high-level feature (output of the last block) to the pixel regressor and feature

* Work done during an internship at Microsoft Research Asia. † Dongdong Chen is the corresponding author.

Table 1: Illustrating the effect of different injection feature. We show fine-tune accuracy (%) on ImageNet-1K.

		Context for Feature Predict		
		Low-level	Middle-level	High-level
Context for Pixel Regress	High-level	83.2	83.3	83.4
	Middle-level	83.6	83.6	83.7
	Low-level	83.8	83.9	84.0

Table 2: Results of bootstrapped feature prediction. The performance is improved from 83.2% to 83.6% with the MAE features as prediction targets for 300 epochs, achieving the same performance with the vanilla MAE with pre-trained 1600 epochs. When the model is pre-trained for 800 epochs, it achieves 83.8%.

Model	Prediction Target	Pre-train Epoch	Fine-tuning
MAE	Pixel	300	83.2
MAE	Pixel	800	83.4
MAE	Pixel	1600	83.6
MAE	Feature	300	83.6
MAE	Feature	800	83.8

predictor respectively. We find that the pixel regression branch is sensitive to the level of injection feature. If we provide high-level or middle-level features to it, it performs poorly. Such a result proves our hypothesis that low-level context is crucial for the encoder to alleviate it from memorizing low-level context. If we provide high-level features to the pixel regressor, it is helpless and the encoder has to memorize the context anyway. On the contrary, we observe that the feature prediction is less sensitive to the provided context. One possible explanation is that the decoder input \mathbf{Z}_v and the prediction target are both high-level semantic features, so the context information may be not so crucial.

2 Simple Feature Prediction for MAE

As we mentioned in our introduction, we find simply replacing the pixel prediction task with feature prediction could help the MAE to get better performance. With a block-wise mask, MAE gets 83.8% accuracy with only 800 epochs, even better than the vanilla MAE pretrained for 1600 epochs. Note that here we provide the prediction target feature from a 800 epochs pretrained MAE (row 2 in Table 2).

3 Experiment Details

In this section, we provide more detailed experimental settings.

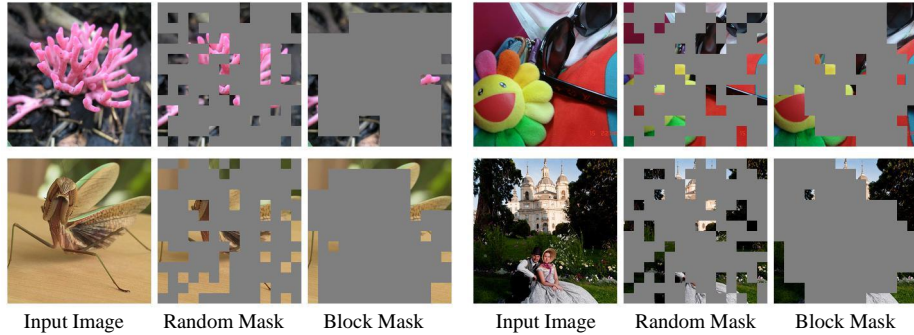


Fig. 1: Visualization of the two different masking strategies. The masked region is close to the visible region in random masking. For block-wise masking, a large continuous block is masked and most center patches are masked.

ImageNet Pretraining. We train our BootMAE with both ViT-B (12 transformer blocks with dimension 768) and ViT-L (24 transformer blocks with dimension 1024) for the encoder. The regressor and the predictor consist of 2 transformer blocks. The dimension of the regressor is set to 512 while the dimension of the predictor is set to the same as the encoder for feature prediction. The learn-able mask token for regressor and the predictor are both initialized by random noise. The input is partitioned 14×14 patches from the image of 224×224 , and each patch is of size 16×16 . Following the setting in MAE, we only use standard random cropping and horizontal flipping for data augmentation. The total masking ratio is 75%, same with that in MAE [4]. The block-wise mask is generated follow the BEiT and we set the minima number for each block is 16 and the maximum is 60. Both ViT-B and ViT-L model are trained for 800 epochs with batch size set to 4096 and the learning rate is set to $1.5e^{-4} * \text{batchsize} / 256$. We use Adam [6] and a cosine schedule [7] with a single cycle and we warm up the learning rate for 40 epochs. The learning rate is further annealed following the cosine schedule. For the momentum parameter, we increase it from 0.999 to 0.9999 linearly in the first 100 epochs. For ViT-B, we further increase it to 0.99999 in the first 400 epochs. We also use a weighted mask to assign larger loss weight to the center region of each block.

ADE20K Semantic segmentation. Here we use: UperNet [9] based on the implementation from mmsegmentaion [3]. For UperNet, we follow the settings in [1] and use AdamW [8] optimizer with initial learning rate $4e^{-4}$, weight decay of 0.05 and batch size of 16 (8 GPUs with 2 images per GPU) for 160K iterations. The learning rate warmps with 1500 iterations at the beginning and decays with a linear decay strategy. We use the layer decay [1] for the backbone and we set it as 0.65. As the ViT architecture outputs features with the same size, here we add four different scale FPNs to scale the feature map into different size. Specifically, we upsample the output feature of the 4th block $4 \times$, upsample the output feature of the 6th block $2 \times$, keep the output feature of the

8th block unchanged and downsample the output feature of the 12th block $2\times$. We use the default augmentation setting in mmsegmentation including random horizontal flipping, random re-scaling (ratio range $[0.5, 2.0]$) and random photometric distortion. All the models are trained with input size 512×512 . The stochastic depth is set to 0.1. When it comes to testing, we report single-scale test result.

COCO Object Detection and Instance Segmentation. We use the classical object detection framework Mask R-CNN [5] based on the implementation from mmdetection [2]. We train the framework with $1\times$ schedule and single-scale input (image is resized so that the shorter side is 800 pixels, while the longer side does not exceed 1333 pixels) for 12 epochs. We use AdamW [8] optimizer with a learning rate of $4e^{-4}$, weight decay of 0.05 and batch size of 16. We also use the layer decay [1] for the backbone and we set it to 0.75. The learning rate declines at the 8th and 11th epoch with decay rate being 0.1. The stochastic depth is set to 0.1. Similar to the implementation of semantic segmentation above, we also use four different scale FPNs to scale the feature map into different size.

References

1. Bao, H., Dong, L., Wei, F.: Beit: Bert pre-training of image transformers. arXiv preprint arXiv:2106.08254 (2021) 3, 4
2. Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C.C., Lin, D.: MMDetection: Open mmlab detection toolbox and benchmark. arXiv preprint arXiv:1906.07155 (2019) 4
3. Contributors, M.: Mmsegmentation, an open source semantic segmentation toolbox. <https://github.com/open-mmlab/msegmentation> (2020) 3
4. He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. arXiv preprint arXiv:2111.06377 (2021) 3
5. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 2961–2969 (2017) 4
6. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014) 3
7. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016) 3
8. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017) 3, 4
9. Xiao, T., Liu, Y., Zhou, B., Jiang, Y., Sun, J.: Unified perceptual parsing for scene understanding. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 418–434 (2018) 3