# HyperMAE: Modulating Implicit Neural Representations for MAE Training

**Varun Belagali   Lei Zhou   Xiang Li   Dimitris Samaras**
Department of Computer Science
Stony Brook University
{vbelagali,lezzhou,xiangli8,samaras}@cs.stonybrook.edu

## Abstract

Implicit Neural Representations (INRs) have been applied successfully for reconstruction tasks in computer vision. However, to the best of our knowledge, using INRs for self-supervised visual recognition has not been explored. In this work, we propose HyperMAE, an INR version of the masked autoencoder (MAE). HyperMAE combines a transformer and a coordinate-MLP to form an efficient decoder architecture that maps the patch coordinates to all pixels in the patch, conditioned on the encoder outputs. The conditioning is implemented as the weight modulation of the coordinate-MLP, which is an INR of the image. Compared with the standard MAE, HyperMAE achieves comparable ImageNet-1k finetuning accuracy with only 72.9% pretraining time using 56.5% GPU memory and 46.5% pretraining time using 88.6% GPU memory. We hope our work could inspire further investigation on INRs for self-supervised learning. The code is available at `https://github.com/cvlab-stonybrook/HyperMAE`.

## 1 Introduction

Recent progress in natural language processing [1, 2, 3, 4] shows large-scale self-supervised learning (SSL) with Transformers is a basis of many powerful foundation models. Similarly in computer vision, Masked image modeling (MIM) [5, 6, 7, 8, 9, 10], a self-supervised learning paradigm, is very effective for pretraining Vision Transformers (ViT). The Masked Autoencoder (MAE) [5] is a simple and efficient MIM framework. MAE has an asymmetric ViT encoder/decoder architecture. The encoder which is usually a deep ViT, processes only a small portion of the image patches. The encoder output is then padded with a learnable mask token to match the size of the input image. Then a ViT decoder is applied on top of it to reconstruct the input image. Positional embedding (a function of the patch coordinates) is added to mask tokens to guide the decoder to transform the masked token into a reconstructed image patch at a specific location. Although avoiding computing on mask tokens in the encoder reduces computation, the MAE decoder is still a Transformer that processes the full length of tokens. This results in a decoder with quadratic complexity with respect to the number of patches.

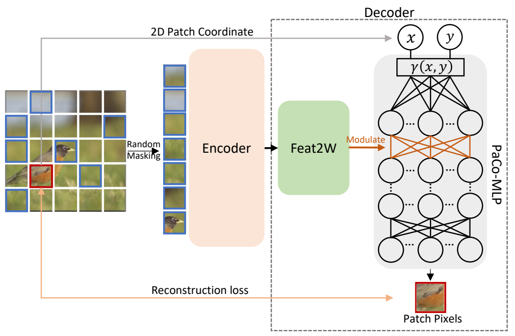

Figure 1: Overview of HyperMAE

In HyperMAE, we propose a new decoder that has two components, 1) the **Pa**tch **Co**ordinate Multi-layer Perceptron (PaCo-MLP) which maps patch coordinates to patch pixels for image reconstruction and 2) Feat2W, a light-weight transformer that transforms the encoder output to weight vectors, which modulate the weights in PaCo-MLP. In contrast to the MAE decoder which processes both encoder output tokens and mask tokens, the input of Feat2W is a combination of encoder output tokens and weight tokens. Note that the number of weight tokens can be much smaller than the number of masked tokens. Inspired by recent work [11], we only modulate the weights of a single PaCo-MLP layer. Thus, we can restrict the number of weight tokens to only 10% of the mask tokens. Figure 1 indicates the high-level design of HyperMAE.

We pretrain our HyperMAE on ImageNet-1k [12] and finetune the pretrained encoder to perform a downstream classification task. Our HyperMAE achieves comparable ImageNet-1k finetuning accuracy with the same number of pretraining epochs while using only 72.9% pretraining time using 56.5% GPU memory and 46.5% pretraining time using 88.6% GPU memory. When finetuning on downstream tasks, for the same number of training epochs, HyperMAE performs comparably, actually slightly better, on Places 365 classification [13] than the MAE baseline, whereas we observe inferior performance on iNaturalist2021 [14] classification.

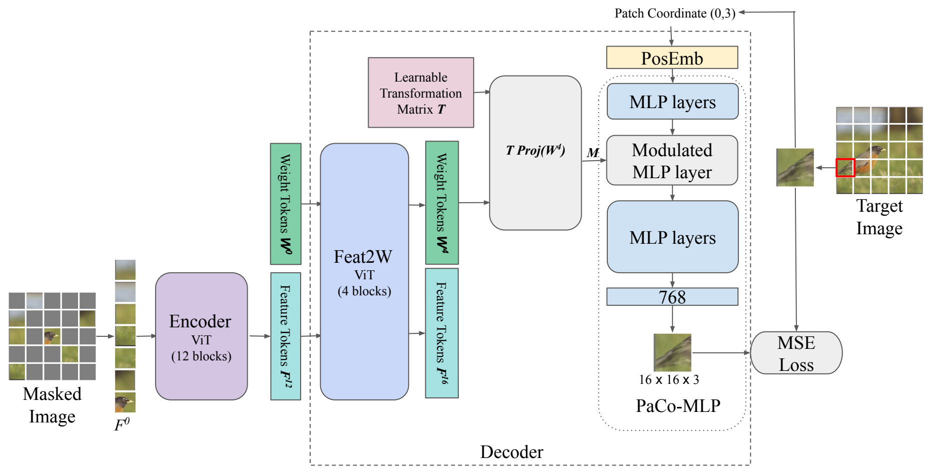

Figure 2: **Block diagram of HyperMAE**. The encoder converts the masked input to Feature tokens ($F^{12}$). Feat2W extracts MLP Weight tokens ($W^4$) from ($F^{12}$). Next, the Transformation matrix: $T$ maps the Weight tokens ($W^4$) to the PaCo-MLP weight matrix $M$. PaCo-MLP is used for patch-level image reconstruction using patch coordinates. An MSE loss is computed between reconstructed and target patches.

## 2 HyperMAE

We propose HyperMAE, an MAE design that takes advantage of a novel decoder module for efficient training. The key intuition of HyperMAE is that instead of directly utilizing a transformer decoder like in MAE [5] to predict the whole image, we use a **Pa**tch **Co**ordinate **M**ulti-**L**ayer **P**erceptron (PaCo-MLP) for patch-level reconstruction from the patch coordinates. In contrast to a conventional transformer in MAE, PaCo-MLP has no attention operations, eliminating the need for a sequence of trainable mask tokens as input. This distinctive characteristic makes PaCo-MLP compute and memory efficient in the process of image reconstruction.

HyperMAE consists of encoder and decoder modules. The encoder acts as the feature extractor as proposed in MAE. The decoder can be further split into the Feat2W and PaCo-MLP modules. Feat2W converts the visual features from the encoder into the weights of a single layer in PaCo-MLP. Feat2W is a transformer that operates on feature tokens from the encoder and a small number of weight tokens (10% of the number of mask tokens used in MAE). Subsequently, PaCo-MLP maps the coordinates of the masked patches to their corresponding patch pixels, thereby reconstructing the image. We train the encoder, Feat2W, and PaCo-MLP end-to-end by applying mean squared error loss between the reconstructed patches and the input image patches. Figure 2 illustrates the overall framework.

## 2.1 Encoder

The goal of the encoder is to extract feature tokens from input patches. These feature tokens will later be passed through Feat2W to generate the weight matrix for PaCo-MLP. The encoder follows the same design as the original MAE encoder [5]. First, the image $I$ is divided into patches $x_p$. Random masking is applied to split patches into two groups: visible $x_{vis}$ and masked $x_{mask}$. The encoder takes only the visible patches as input and first converts them to patch embedding. Then, these patch embeddings are passed through 12 ViT blocks to get feature tokens $F^{12} = \mathit{Encoder}(x_{vis})$. For all tokens, we use superscript to note the number of ViT blocks the token has passed through.

## 2.2 Decoder

Different from the MAE decoder, our decoder consists of 2 modules: 1) Feat2W: a shallow transformer network that maps feature tokens to PaCo-MLP weight matrix. 2) Paco-MLP: a multiple layer perceptron used as implicit neural representation to reconstruct the image at patch level.

**Feat2W** The purpose of the Feat2W module is to extract weights from the encoder's feature tokens in order to modulate PaCo-MLP for image reconstruction. We specifically modulate a single layer of PaCo-MLP (without bias), which means that the Feat2W module outputs a matrix $M$ with dimension $h \times h$, where $h$ represents the hidden layer dimension of PaCo-MLP.
To extract $M$, we first append the $m$ feature tokens $F^{12}$ with $n$ $(< m)$ learnable weight tokens $W^0$ of same dimension $d$. The weight tokens can be regarded as similar to the classification token, which is generally used to extract the global features of the image in ViT. The weight tokens extract the information needed to derive the MLP weights. The $m$ feature tokens $F^{12}$ and $n$ weight tokens $W^0$ are passed through 4 ViT blocks as a list of $m + n$ tokens to get $F^{16}$ and $W^4$ respectively. Next, the weight tokens $W^4$ are projected to the hidden dimension of PaCo-MLP $h$ using a linear layer $Proj$. This results in a tensor with shape $n \times h$. But, the target weight matrix $M$ of the PaCo-MLP is of the shape $h \times h$. We use a learnable transformation matrix $T$ of size $h \times n$ to transform the weight tokens to $M$. $M$ is derived by a matrix multiplication between $T$ and $Proj(W^4)$, resulting in a $h \times h$ tensor $M$.

**Patch Coordinate-MLP** We use PaCo-MLP as an implicit neural representation of the target image. The PaCo-MLP is a Multi-Layer Perceptron with $l$ layers and $h$ hidden dimension. It takes the 2-D patch coordinates as inputs and outputs the patch pixels.
Among the $l$ layers, the $l-1$ layers are shared across all images in the dataset. The weights of a single layer $mod$ $(0 \leq mod \leq l-1)$ are predicted by Feat2W to provide instance-specific information. The layers before layer $mod$ act as positional embedding transformations. After the instance-specific modulation by layer $mod$, the next layers transform the modulated output to patch pixels. We use ReLU as activation for the hidden layers.
Different from usual implicit neural representations, for each forward pass, we predict all pixels in one patch instead of an individual pixel because the number of patches ($N = HW/P^2$) is much less compared to the number of pixels in the image ($HW$). This reduces the number of passes through PaCo-MLP and results in a more efficient way to reconstruct the image. The output dimension of the PaCo-MLP is a flattened patch of size $P^2C$. $H$, $W$, $P$, and $C$ are the height, width, patch size, and channels of the image respectively.
During pretraining phase, the mean squared error between the input image patches and the reconstructed image patches is used as the loss function. After the pretraining, the encoder of HyperMAE is used as ViT Backbone for downstream task fine-tuning. Feat2W and PaCo-MLP are discarded after pretraining similar to the transformer decoder in MAE. The pseudo-code for HyperMAE loss computation is presented in Appendix (Algorithm 1).

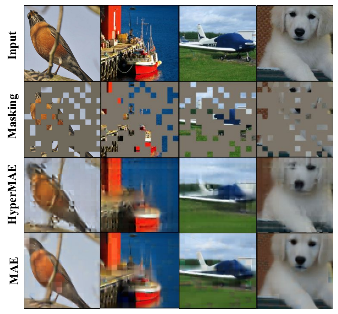

Figure 3: Sample image reconstructions by HyperMAE and MAE. First row: input image, second row: masked image, third row: reconstruction by HyperMAE, and fourth row: reconstruction by MAE.

## 3 Experiments

We benchmark our method and its variants along with baseline MAE [5] using the image classification task on multiple datasets.

**Main results** We follow the widely adopted pretrain and finetune training scheme in SSL. The pretraining is performed on ImageNet-1k [12] for 400 epochs, utilizing ViT Base (ViT-B) architecture. After pretraining, we finetune all models for 50 epochs with a linear classification head on top of the global average pooling of all tokens.
Additionally, we investigate the transfer learning capabilities of HyperMAE on two downstream classification datasets: Places365 [13] and iNaturalist2021 [14]. We start with the model pretrained on ImageNet-1k and perform fine-tuning for 50 epochs, matching the ImageNet setting. The complete set of hyperparameters and training details are provided in Table 3 and Appendix 5.3. We report two versions of HyperMAE: HyperMAE-wo/norm and HyperMAE-w/norm, where the only difference is that HyperMAE-w/norm is trained with normalized patches (the pixels in target patch are normalized based on statistics of all pixels belonging to the patch), while HyperMAE-wo/norm is trained on the original patch pixels as target which is not normalized at the patch level. MAE [5] is trained with target patch normalization too as it has been shown that normalization improves the performance.

Table 1: **Top-1 accuracy**. The models are pretrained for 400 epochs and then finetuned for 50 epochs. Along with accuracy, the pretraining metrics: Time - Walk clock time, Mem: GPU memory, MAC: Multiply-Accumulate operations are also indicated.

| **Method** | **Time**(hrs) | **Mem**(GB) | **MAC**(G) | **Epochs** | **ImageNet-1k** | **Places365** | **iNaturalist2021** |
|---|---|---|---|---|---|---|---|
| HyperMAE-wo/norm | 68.27/107.1 | 12.59/8.03 | 6.42 | 400 | 81.92 | 58.83 | 70.99 |
| HyperMAE-w/norm | 68.27/107.1 | 12.59/8.03 | 6.42 | 400 | 82.35 | 58.89 | 71.53 |
| MAE | 146.93 | 14.21 | 9.78 | 400 | 82.40 | 58.72 | 73.81 |

We report the top-1 classification accuracy in Table 1, along with the wall clock time, GPU memory usage, and Multiply-Accumulate operations (MAC) computed during pretraining. The wall clock time was calculated using 4 NVIDIA TITAN RTX 24GB GPUs. On ImageNet-1k, the finetuning accuracy of HyperMAE-w/norm is closely comparable with MAE. However, HyperMAE uses only 72.86% pretraining time with 56.5% GPU memory as MAE and HyperMAE is also efficient in terms of MAC. Further, if we increase the memory consumption by doubling the number of images per GPU, HyperMAE takes only 46.5% of training time when using 88.6% of GPU RAM. HyperMAE outperforms MAE by a small margin on Places365 and underperforms on iNaturalist2021. HyperMAE presents an efficient alternative design for image reconstruction which can be used to reduce the computational cost of pretraining.

We also show qualitative results of image reconstruction in Figure 3.

**Ablations** We also conduct detailed ablations on many design choices of HyperMAE. Table 2 presents the ablation study on the number of weight tokens and Feat2W depth, 16 weight tokens and 4 blocks of Feat2W achieve the best result. More ablations regarding network architecture, masking ratio and other designed choices are presented in Appendix 5.2.

Table 2: Top-1 accuracy: HyperMAE ablation using ViT-Tiny on ImageNet100 under different number of weight tokens and depth of Feat2W. More ablations are presented in the Appendix.

| **# of weight tokens** $n$ | 4 | 8 | 16 | 32 |
|---|---|---|---|---|
| **Acc1 (%)** | 77.16 | 78.02 | **79.26** | 78.38 |

| **Feat2W depth** $f$ | 1 | 4 | 8 |
|---|---|---|---|
| **Acc1 (%)** | 78.46 | **79.26** | 78.78 |

## 4 Conclusion

We present HyperMAE, a new design of Masked Autoencoder that takes advantage of implicit neural representation and patch-level pixel prediction. In comparison with MAE, HyperMAE achieves comparable ImageNet-1k finetuning accuracy with less GPU memory and pretraining time. We hope that HyperMAE provides motivation to further explore INRs in self-supervised visual recognition.

**Acknowledgement:** The reported research was partly supported by NSF awards IIS-2212046 and IIS-2123920, and a Stony Brook OVPR seed grant.

## References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[2] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[5] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16 000–16 009.

[6] Z. Xie, Z. Zhang, Y. Cao, Y. Lin, J. Bao, Z. Yao, Q. Dai, and H. Hu, "Simmim: A simple framework for masked image modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 9653–9663.

[7] C. Wei, H. Fan, S. Xie, C.-Y. Wu, A. Yuille, and C. Feichtenhofer, "Masked feature prediction for self-supervised visual pre-training," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 14 668–14 678.

[8] H. Bao, L. Dong, S. Piao, and F. Wei, "Beit: Bert pre-training of image transformers," *arXiv preprint arXiv:2106.08254*, 2021.

[9] Z. Tong, Y. Song, J. Wang, and L. Wang, "Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training," *arXiv preprint arXiv:2203.12602*, 2022.

[10] R. Girdhar, A. El-Nouby, M. Singh, K. V. Alwala, A. Joulin, and I. Misra, "Omnimae: Single model masked pretraining on images and videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 10 406–10 417.

[11] C. Kim, D. Lee, S. Kim, M. Cho, and W.-S. Han, "Generalizable implicit neural representations via instance pattern composers," *arXiv preprint arXiv:2211.13223*, 2022.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[13] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," *Advances in neural information processing systems*, vol. 27, 2014.

[14] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie, "The inaturalist species classification and detection dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8769–8778.

[15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[16] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[17] K. Ranasinghe, M. Naseer, S. Khan, F. S. Khan, and M. S. Ryoo, "Self-supervised video transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 2874–2884.

[18] X. Li, J. Shang, S. Das, and M. Ryoo, "Does self-supervised learning really improve reinforcement learning from pixels?" *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 865–30 881, 2022.

[19] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[20] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 9650–9660.

[21] S. Becker and G. E. Hinton, "Self-organizing neural network that discovers surfaces in random-dot stereograms," *Nature*, vol. 355, no. 6356, pp. 161–163, 1992.

[22] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 15 750–15 758.

[23] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in neural information processing systems*, vol. 33, pp. 21 271–21 284, 2020.

[24] X. Chen, S. Xie, and K. He, "An empirical study of training self-supervised vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9640–9649.

[25] L. Zhou, H. Liu, J. Bae, J. He, D. Samaras, and P. Prasanna, "Self pre-training with masked autoencoders for medical image analysis," *arXiv preprint arXiv:2203.05573*, 2022.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[27] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[28] L. Huang, S. You, M. Zheng, F. Wang, C. Qian, and T. Yamasaki, "Green hierarchical vision transformer for masked image modeling," *Advances in Neural Information Processing Systems*, vol. 35, pp. 19 997–20 010, 2022.

[29] A. Baevski, W.-N. Hsu, Q. Xu, A. Babu, J. Gu, and M. Auli, "Data2vec: A general framework for self-supervised learning in speech, vision and language," in *International Conference on Machine Learning*. PMLR, 2022, pp. 1298–1312.

[30] M. Assran, Q. Duval, I. Misra, P. Bojanowski, P. Vincent, M. Rabbat, Y. LeCun, and N. Ballas, "Self-supervised learning from images with a joint-embedding predictive architecture," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 15 619–15 629.

[31] S. Li, D. Wu, F. Wu, Z. Zang, B. Sun, H. Li, X. Xie, S. Li *et al.*, "Architecture-agnostic masked image modeling–from vit back to cnn," *arXiv preprint arXiv:2205.13943*, 2022.

[32] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene representation networks: Continuous 3d-structure-aware neural scene representations," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[33] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 165–174.

[34] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4460–4470.

[35] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, and R. Ramamoorthi, "and ren ng. nerf: Representing scenes as neural radiance fields for view synthesis," 2020.

[36] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, "Nerf in the wild: Neural radiance fields for unconstrained photo collections," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7210–7219.

[37] C. Reiser, S. Peng, Y. Liao, and A. Geiger, "Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 14 335–14 345.

[38] I. Anokhin, K. Demochkin, T. Khakhulin, G. Sterkin, V. Lempitsky, and D. Korzhenkov, "Image generators with conditionally-independent pixel synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 278–14 287.

[39] I. Skorokhodov, S. Ignatyev, and M. Elhoseiny, "Adversarial generation of continuous images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 753–10 764.

[40] H. Chen, B. He, H. Wang, Y. Ren, S. N. Lim, and A. Shrivastava, "Nerv: Neural representations for videos," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 557–21 568, 2021.

[41] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7462–7473, 2020.

[42] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7537–7547, 2020.

[43] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[44] A. Nichol, J. Achiam, and J. Schulman, “On first-order meta-learning algorithms,” *arXiv preprint arXiv:1803.02999*, 2018.

[45] D. Ha, A. Dai, and Q. V. Le, “Hypernetworks,” *arXiv preprint arXiv:1609.09106*, 2016.

[46] M. Tancik, B. Mildenhall, T. Wang, D. Schmidt, P. P. Srinivasan, J. T. Barron, and R. Ng, “Learned initializations for optimizing coordinate-based neural representations,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2846–2855.

[47] Y. Chen and X. Wang, “Transformers as meta-learners for implicit neural representations,” in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVII.* Springer, 2022, pp. 170–187.

[48] Y. Tian, D. Krishnan, and P. Isola, “Contrastive multiview coding,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16.* Springer, 2020, pp. 776–794.

## 5 Appendix

### 5.1 Preliminaries

**Vision Transformer** Vision Transformer proposed in [15] is a transformer-based architecture for images. The key idea involves converting the input image into discrete tokens so that transformer blocks can process them. This is done by dividing the image $I \in \mathbb{R}^{(H\times W\times C)}$ into non-overlapping patches using a grid size of $H/P \times W/P$ where the size of each patch is $P \times P \times C$. Each patch is further flattened to the size $P^2C$, representing a token. Further, the grid is flattened to form image representation $x_p \in \mathbb{R}^{(N\times P^2C)}$, where $N$ (number of patches) = $HW/P^2$. Next, these patches are passed through a series of transformer blocks termed as ViT Blocks. Each ViT block consists of a Multi-head Self Attention Layer and MLP layers.

**Masked Autoencoder** Masked Autoencoder (MAE) [5] is a transformer-based self-supervised learning method, that actively masks random patches in an image and predicts the masked patches given the other visible patches. It contains a transformer encoder to process the tokens visible to the model and a transformer decoder to reconstruct the masked patches given the output of the visual encoder and an array of learnable mask tokens. As in standard ViT, first the image $I \in \mathbb{R}^{(H\times W\times C)}$ is divided into non-overlapping patches $x_p$. Next, $x_p$ is divided into 2 sets of patches, 1) visible patches, and 2) masked patches. The number of masked patches is $mr$ * $N$ where $mr$ is a masking ratio. Only the visible patches are passed through the encoder. The output of the encoder is appended with the learnable mask token along with the positional embedding of masked patches. Positional embedding guides the reconstruction of different patches using the same mask token. These tokens are passed through a transformer decoder to reconstruct the image at the patch level.

**Self-supervised Learning** Self-supervised learning (SSL) [16] aims at training neural networks on unlabelled data using a pretext task. Leveraging large-scale unlabelled data, SSL is able to learn stronger representation accelerating and improving downstream task adaptation [17, 18]. One popular direction in recent SSL works is contrastive learning [19, 20, 21, 22, 23, 24], where the supervision signal is based on the semantic invariance between dual-stream encoder representations. Another direction is based on self-prediction [5, 7, 25], which actively masks some information of the data sample, and predicts the masked parts given the left information [26, 5, 7, 25].

**Masked Image Modeling** In addition to MAE, there are many directions for using masked image modeling for self-supervised learning. Other works include BEiT [8], SimMIM [6], and MaskFeat [7]. BEiT takes motivation from BERT [26] to build a masking-based SSL technique for vision. It uses dVAE tokenizer to treat image patches as discrete tokens. The masked and visible tokens are passed through transformer layers and then the masked tokens are predicted using softmax output. SimMIM explores both ViT and Swin [27] architectures. The encoder processes both visible and masked tokens and the decoder is a lightweight MLP used to predict raw pixels. The encoder is more computationally expensive when compared to MAE. Further, GreenMIM [28] is a technique proposed for training Hierarchical Vision Transformers in a masked image modeling setting. They propose to use group window attention coupled with dynamic programming to obtain optimal grouping and sizes. MaskFeat is a concurrent work of MAE, where instead of predicting the pixel value as the target for reconstruction, features like HOG were used. data2Vec [29] proposed a generalized self-supervised learning for vision, speech, and language. The key idea is to have a student-teacher network setup. The masked input is passed through the student network and the unmasked input is passed through the teacher network. The teacher is updated as an exponential moving average of student and the loss is applied between student and teacher output to learn the student. I-JEPA [30] divides the image into context block and target blocks, which are processed by context and target encoders. The context encoder’s output is used to predict the target block features. The target encoder is updated as the exponential moving average of the context encoder. Further, A recent work [31], proposes architecture-agnostic masked image modeling training that fits both ViT and CNN.

**Implicit Neural Representation** Implicit neural representations (INRs) are compact and continuous data representations. INRs are essentially coordinate-MLPs that map coordinates to the target values with an MLP. INRs originate from 3D object and scene representations [32, 33, 34, 35]. Specifically, DeepSDF [33] represents 3D shapes as a signed distance function; Occupancy Networks [34] represent 3D shapes as binary classification networks predicting each 3D coordinate as an inside point or an outside one. NeRFs [35] and more recent works [36, 37] apply INRs to novel view synthesis. They represent a scene as a neural radiance field mapping every 3D position to a density and a view-dependent RGB value. With differentiable volumetric rendering, the whole pipeline can be optimized end-to-end from 2D images shot in multiple views. Beyond 3D vision, INRs have also been applied to represent 2D images [38, 39] and videos [40]. As the ReLU activation function may lack the capacity for representing fine details, the sine activation function [41] and Fourier features [42] of coordinates have been proposed to preserve more data details. To generalize an INR to different targets, meta-learning [43, 44] and hyper networks [45] are adopted to learn a shared initialization [46] or modulate the INR weights directly [32, 47, 11].

## 5.2 Design disucssion

In this section, we discuss the effect of various components of HyperMAE. We experiment using ViT-Tiny by pretraining for 400 epochs and finetuning for 50 epochs on ImageNet100 [48]. ImageNet100 is a subset of ImageNet-1k containing 100 classes. We report the top-1 accuracy on the validation dataset. The default configuration of HyperMAE is described in Table 3.

Table 3: Default setting of HyperMAE.

| **Property** | **Value** |
|---|---|
| Patch size ($P$) | $16 \times 16$ |
| Masking Ratio | 75% |
| Encoder depth ($e$) | 12 |
| Feat2W depth ($f$) | 4 |
| Number of weight tokens ($n$) | 16 |
| Depth of PaCo-MLP ($l$) | 6 |
| Hidden dim of PaCo-MLP ($h$) | 256 |
| Modulation layer ($mod$) | 1 |
| Target Patch | Masked patch only |
| Normalize Target Patch? | No |

### 5.2.1 Number of weight tokens

The Feat2W takes two sets of tokens: $m$ feature tokens and $n$ weight tokens. Table 4 shows the finetuning accuracy achieved with $n = \{4, 8, 16, 32\}$. $n = 16$ achieves the best accuracy. Under 75% masking ratio ($m = 50$), the Feat2W module processes a total of 66 tokens ($m + n = 50 + 16$). These 66 tokens represent approximately 33% of the total tokens ($N = 196$). In contrast, the transformer decoder used in MAE [5] operates on the entire set of 196 tokens. Therefore, despite the inclusion of an additional Feat2W module in HyperMAE, our approach is inherently more efficient in terms of token usage.

Table 4: Finetuning accuracy on ImageNet100 using different number ($n$) of weight tokens in Feat2W module.

| **# of weight tokens** $n$ | 4 | 8 | 16 | 32 |
|---|---|---|---|---|
| **Acc1 (%)** | 77.16 | 78.02 | **79.26** | 78.38 |

### 5.2.2 Depth of Feat2W Block

The finetuning accuracy achieved by different Feat2W depths $f$ is presented in Tables 5. The depth $f$ is the number of ViT blocks. The results indicate that a depth of $f = 4$ yields the best accuracy, suggesting that Feat2W functions effectively with a shallow transformer architecture.

### 5.2.3 Sharing ViT Blocks

Within the HyperMAE framework, both the encoder and Feat2W employ ViT blocks of the same type. This similarity enables us to explore the sharing of ViT layers between the encoder and Feat2W components. Since

Table 5: Finetuning accuracy on ImageNet100 using different number $f$ of ViT Blocks in Feat2W module.

| **Feat2W depth $f$** | 1 | 4 | 8 |
|---|---|---|---|
| **Acc1 (%)** | 78.46 | **79.26** | 78.78 |

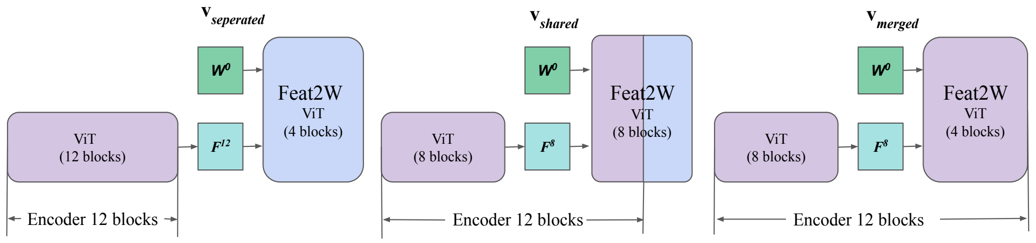

Figure 4: **Different versions of HyperMAE**. In $v_{separated}$, the encoder and Feat2W do not share ViT Blocks. In $v_{shared}$, Feat2W shares half of the ViT Blocks with the encoder. In $v_{merged}$, Feat2W shares all blocks with encoder. Purple color indicates the ViT Blocks belonging to the encoder. Blue color represents the blocks that are exclusively part of Feat2W.

the encoder is used as the backbone for downstream tasks, the ViT blocks belonging only to Feat2W are dropped after pretraining. Figure 4 illustrates the different ways of ViT block sharing between the encoder and Feat2W. In $v_{separated}$, no blocks are shared between the encoder and Feat2W. In $v_{shared}$, Feat2W consists of 8 blocks, and 4 blocks are shared with the encoder. In $v_{merged}$, Feat2W has 4 blocks and shares all 4 blocks with the encoder. In $v_{merged}$, all transformer blocks will be used for downstream task finetuning as part of the encoder. In $v_{shared}$ and $v_{merged}$, the weight tokens are inputted at the first shared ViT block.

Table 6 shows the finetuning accuracy achieved by the various versions. Our default version $v_{separated}$ outperforms other versions. An intuition behind this observation is that, under a shared scenario, some layers of encoder observe weight tokens during pretraining and these weight tokens are not used in finetuning. This creates a discrepancy between the type of the tokens seen by the encoder during pretraining and finetuning, thus can affect negatively finetuning accuracy.

Table 6: Finetuning accuracy on ImageNet100 using a different number of shared ViT blocks between encoder and Feat2W modules. $e$: Encoder depth, $f$: Feat2W depth, $s$: number of shared ViT blocks between encoder and Feat2W.

| **Version** | $e$ | $s$ | $f$ | **Acc1 (%)** |
|---|---|---|---|---|
| $v_{separated}$ (default) | 12 | 0 | 4 | **79.26** |
| $v_{shared}$ | 12 | 4 | 8 | 78.44 |
| $v_{merged}$ | 12 | 4 | 4 | 77.18 |

### 5.2.4 Modulation Layer

In our default configuration, we utilize $mod = 1$. The performance results for different values of $mod$ are presented in Table 7. When $mod = 0$, the performance is notably low. This can be attributed to the absence of a layer before $mod$ to modify the positional encoding of the coordinates. On the other hand, similar performance is observed for $mod = 1$ and $mod = 2$. However, higher values of $mod$ yield poorer results, likely due to insufficient layers after modulation to accurately predict patch pixels. This finding aligns with the findings in [11].

Table 7: Finetuning accuracy on ImageNet100 by modulating layers at different indices $mod$ in PaCo-MLP. The results indicate that $mod = 1$ and $mod = 2$ produce comparable results and outperform modulation at other layer indices.

| $mod$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **Acc1(%)** | 76.64 | 79.26 | **79.36** | 78.54 | 76.20 |

#### 5.2.5 Depth of PaCo-MLP

Next, we experiment with the depth $l$ of PaCo-MLP. Table 8 shows the finetuning accuracy with various values for $l$. $l = 6$ achieves the best finetuning accuracy. $l = 3$ achieves very low accuracy compared to $l > 3$, as there is only one layer after modulation, and this might not be sufficient to represent the image.

Table 8: Finetuning accuracy on ImageNet100 using different number of layers ($l$) in PaCo-MLP. Accuracy increases with a number of layers up to 6 layers.

| **PaCo-MLP Depth $l$** | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|
| **Acc1 (%)** | 76.88 | 78.16 | 78.72 | **79.26** | 78.66 |

#### 5.2.6 Hidden dim of PaCo-MLP

It has been a standard practice to use MLP with hidden dimension $h = 256$ in implicit neural representations [47]. Table 9 presents the results for $h = \{128, 256, 512\}$. Our analysis reveals that the best performance is obtained when $h$ is set to 256. On the other hand, using $h = 128$ may be insufficient to adequately represent the image, while employing $h = 512$ could result in a stronger decoder but weaker representation learning by the encoder.

Table 9: Finetuning accuracy on ImageNet100 using different hidden dimesions $h$ for PaCo-MLP layers.

| $h$ | 128 | 256 | 512 |
|---|---|---|---|
| **Acc1 (%)** | 78.50 | **79.26** | 78.64 |

#### 5.2.7 Masking Ratio

The masking ratio plays a crucial role in determining the amount of information available to the MAE for image reconstruction. A higher masking ratio increases efficiency by allowing the encoder to exclusively process visible patches. We explore the impact of different masking ratios on HyperMAE and report the corresponding accuracies in Table 10. Consistent with the findings of MAE [5], we observe that a masking ratio of 75% yields the best result. This implies that the optimal masking ratio is dependent on the encoder and remains unaffected by the specific decoding approach employed for image reconstruction.

Table 10: Finetuning accuracy on ImageNet100 under different masking ratios. Inline with MAE [5], HyperMAE achieves the best result with 75% masking.

| **Masking Ratio** | 0.45 | 0.60 | 0.75 | 0.90 |
|---|---|---|---|---|
| **Acc1 (%)** | 77.96 | 78.50 | **79.26** | 77.38 |

#### 5.2.8 Target Patch

In the default setting, we only predict masked patches. Table 11 indicates that predicting masked patches only results in better accuracy. This observation is in line with MAE [5]. Predicting visible patches can make image reconstruction easier, and thus the encoder might not learn strong representation during pretraining.

Table 11: Finetuning accuracy on ImageNet100 under different configurations of target patches. As in MAE, applying loss to only masked patches achieves the best accuracy.

| **Target Patches** | **Acc1 (%)** |
|---|---|
| All Patches | 78.04 |
| Masked Patches Only | 79.26 |

### 5.3 Architecture details

**MAE** We use the ViT-Base version of MAE presented in [5]. Encoder has 12 ViT blocks with 768 dimension and 12 heads. The decoder has 8 ViT blocks with 512 dimension and 16 heads.

**HyperMAE** HyperMAE has 3 networks: Encoder, Feat2W, and PaCo-MLP. Encoder has 12 ViT blocks with 12 heads and a dimension of 768. Feat2W has 4 ViT Blocks with 12 heads and a dimension of 768. The number of weight tokens in Feat2W is 16. PaCo-MLP is an MLP with 256 hidden dimension and 768 output dimension. For the ViT-Tiny version used in section 5.2, ViT blocks have 3 heads and 192 dimension. Similar to MAE, HyperMAE also uses class token as additional input to the encoder along with image patches.

## 5.4 Training details

We built our model using the MAE [5] codebase. Table 12 indicates the configuration of pretraining on ImageNet-1k. Table 13 presents the details for finetuning on ImageNet-1k, Places365, and iNaturalist2021. For ImageNet-100 pretraining and finetuning, we use the same settings as ImageNet-1k but reduce the batch size to 512.

Table 12: Pretraining configuration on ImageNet-1k.

| **config** | **value** |
|---|---|
| optimizer | AdamW |
| base learning rate | 1.5e-4 |
| weight decay | 0.05 |
| optimizer momentum | $\beta_1$=0.9, $\beta_2$=0.95 |
| batch size | 4096 |
| learning rate schedule | cosine decay |
| warmup epochs | 40 |
| augmentation | RandomResizedCrop |
| training epochs | 400 |

**Algorithm 1:** PyTorch-style pseudocode for HyperMAE loss computation

```
# Learnable modules:  Encoder:  12-layer ViT; Feat2W: 4-layer ViT; PaCo-MLP: l-1
 linear layers; W_0:  Weight tokens; T: Transformation matrix; Proj:  single
 linear layer;
# Input image:  I
def compute_loss(I):
   # Encoder
   x_vis, x_masked = RandomMask(Patchify(I))
   F_0 = x_vis
   F_12 = Encoder(F_0)

   # Decoder-Feat2W
   W_4, F_16 = Feat2W(concat([W_0, F_12]))
   M = torch.matmul(T, Proj(W_4))

   # Decoder-PaCo-MLP
   pco = PatchCoordinates(x_masked)
   z = PositionalEmbedding(pco, h)
   # MLP layers before modulation
   for i in range(mod):
      z = PaCo-MLP[i](z)
   # Modulation using multiplication
   z = torch.matmul(z, M)
   # MLP layers after modulation
   for i in range(mod, l-1):
      z = PaCo-MLP[i](z)
   # Patch & Channel size P=16, C=3
   z = z.reshape(..., P, P, C)

   # Compute loss
   loss = MSELoss(z, x_masked)
   return loss
```

Table 13: Finetuning details for classification on Imagenet-1k, Places365, iNaturalist2021.

| **config** | **value** |
|---|---|
| optimizer | AdamW |
| base learning rate | 1e-3 |
| weight decay | 0.05 |
| optimizer momentum | $\beta_1$=0.9, $\beta_2$=0.999 |
| layer-wise lr decay | 0.75 |
| batch size | 1024 |
| learning rate schedule | cosine decay |
| warmup epochs | 5 |
| training epochs | 50 |
| augmentation | RandAug (9, 0.5) |
| label smoothing | 0.1 |
| mixup | 0.8 |
| cutmix | 1.0 |
| drop path | 0.1 |