# Appendix

This appendix provides implementation details, including model configurations, pre-training and fine-tuning recipes, and sparse and dense encoding methods for FCMAE pre-training (see §A). In §B, we present complete fine-tuning accuracy comparisons between ConvNeXt V1 and V2 on ImageNet 1K and 22K. In §C, we perform analyses on the efficiency of sparse encoding and general feature analysis using the class selectivity index. Finally, in §D, we conduct further ablation studies and analysis.

## A. Implementation Details

### A.1. ConvNeXt V2 model configurations

The basic models, *i.e.*, Tiny (28M), Base (89M) and Large (198M), follow the same configurations of the stage, block (B), and channel (C) settings of the ConvNeXt V1 [25].

- ConvNeXt V2-T: $C$=96, $B$=(3, 3, 9, 3)
- ConvNeXt V2-B: $C$=128, $B$=(3, 3, 27, 3)
- ConvNeXt V2-L: $C$=192, $B$=(3, 3, 27, 3)

Given the same definitions above, we scale the model to provide a broad model size spectrum, targeting versatile scenarios. First, to obtain efficient models, we scale down as follows:

- ConvNeXt V2-A: $C$=40, $B$=(2, 2, 6, 2)
- ConvNeXt V2-F: $C$=48, $B$=(2, 2, 6, 2)
- ConvNeXt V2-P: $C$=64, $B$=(2, 2, 6, 2)
- ConvNeXt V2-N: $C$=80, $B$=(2, 2, 8, 2)

A, F, P, N denote Atto (3.7M), Femto (5.2M), Pico (9.1M), and Nano (15.6M) models designed originally in [31]. Next, to introduce the large-capacity variant, we scale up as follows:

- ConvNeXt V2-H: $C$=352, $B$=(3, 3, 27, 3)

H denotes Huge (659M) model, which is newly presented in this work.

### A.2. ImageNet Experiments

**Pre-training** All models share the same pre-training setup, as noted in Table 1. We use the linear *lr* scaling rule [18]: *lr* = *base_lr*×batchsize / 256.

**ImageNet-1K fine-tuning** As the learning capacity varies by model size, we adopt different fine-tuning recipes for each model. We summarize them in Table 2, 3 and 4. We see longer fine-tuning epochs help small models. We adopt two different learning-rate layer decay strategies in this work: group-wise [25], where we treat three sequential layers as a single "layer" and use the same decaying value for them, and the layer-wise [2], where we assign a distinct value for each layer, both following the standard decaying rule. The default is a layer-wise strategy, but we apply the group-wise decaying strategy to Base and Large models.

**ImageNet-22K intermediate fine-tuning** We conduct ImageNet-22K intermediate fine-tuning with the FCMAE-pretrained ConvNeXt models. We use nano, tiny, base, large, and huge models. The setups are summarized in Table 5 and 6. Similarly, using larger layer-wise learning rate decay values for small models is helpful.

**Sparse encoding implementations.** We propose two possible implementations to enable FCMAE pre-training: 1) sparse encoding using sparse convolution [12, 19, 20] supported by external libraries [12, 15], and 2) simulating sparse encoding with the masked dense convolution, which can be easily implemented by applying binary masks *before and after* the standard convolution operation. As they produce numerically identical outputs, both can be adopted depending on different use cases. In this work, we adopt sparse encoding on the GPU environment, where we use MinkowskiEngine library [12] and PyTorch framework [29]; we use dense masked conv based encoding on TPU accelerators using Jax [4]. The experiments in the main paper are all conducted on TPU (v3-256) pods and we release a PyTorch reproduction.

### A.3. Object detection and segmentation on COCO

For COCO experiments, we use the MMDetection [7] toolbox and the final model weights from ImageNet-1K pre-training as network initializations. All models are trained with a 3x schedule (36 epochs) and a batch size of 32. We utilize an AdamW optimizer [27] with a learning rate of 1e-4, a weight decay of 0.05 and sweep layer-wise learning rate decay in {0.9, 0.95}, stochastic depth rate in {0.2, 0.3, 0.4, 0.5}. We employ a large-scale jittering augmentation [17] (1024×1024 resolution, scale range [0.1, 2.0]). We use single-scale testing with soft-NMS [3] during inference.

### A.4. Semantic segmentation in ADE20K

For ADE20K experiments, we use the MMSegmentation [14] toolbox. We use an AdamW optimizer [27] with the following hyperparameters: a weight decay of 0.05, a batch size of 16 and sweep layer-wise decay rate {0.8, 0.9}, learning rate {1e-4, 2e-4, 3e-4}, stochastic depth rate {0.1, 0.2, 0.3, 0.4}. All models are trained for 160K iterations with an input resolution of 512×512. In inference, a multi-scale test using resolutions that are [0.75,0.875,1.0,1.125,1.25] of 512×2048 is employed.

Similar to [32], we initialized the segmentation models using model weights after supervised fine-tuning on

| config | value |
|---|---|
| optimizer | AdamW [27] |
| base learning rate | 1.5e-4 |
| weight decay | 0.05 |
| optimizer momentum | $\beta_1, \beta_2$=0.9, 0.95 [8] |
| batch size | 4096 |
| learning rate schedule | cosine decay [26] |
| warmup epochs [18] | 40 |
| training epochs | 800 or 1600 |
| augmentation | RandomResizedCrop |

Table 1. **Pre-training setting.**

| config | value |
|---|---|
| optimizer | AdamW |
| base learning rate | 2e-4 |
| weight decay | 0.05 (F), 0.3 (A/P/N) |
| optimizer momentum | $\beta_1, \beta_2$=0.9, 0.999 |
| layer-wise lr decay [2, 13] | 0.9 |
| batch size | 1024 |
| learning rate schedule | cosine decay |
| warmup epochs | 0 |
| training epochs | 600 |
| augmentation | RandAug (9, 0.5) [16] |
| label smoothing [30] | 0.2 |
| mixup [34] | 0.0 (A), 0.3 (F/P), 0.5 (N) |
| cutmix [33] | 0.0 (A), 0.3 (F/P), 0.5 (N) |
| drop path [24] | 0.1 (A/N), 0.0 (F/P), |
| head init [25] | 0.001 |
| ema | 0.9999 |

Table 2. **End-to-end IN-1K fine-tuning setting for Atto (A), Femto (F), Pico (P) and Nano (N) models.**

| config | value |
|---|---|
| optimizer | AdamW |
| base learning rate | 8e-4 |
| weight decay | 0.05 |
| optimizer momentum | $\beta_1, \beta_2$=0.9, 0.999 |
| layer-wise lr decay [2, 13] | 0.9 |
| batch size | 1024 |
| learning rate schedule | cosine decay |
| warmup epochs | 40 |
| training epochs | 300 |
| augmentation | RandAug (9, 0.5) [16] |
| label smoothing [30] | 0.1 |
| mixup [34] | 0.8 |
| cutmix [33] | 1.0 |
| drop path [24] | 0.2 |
| head init [25] | 0.001 |
| ema | 0.9999 |

Table 3. **End-to-end IN-1K fine-tuning setting for Tiny model.**

ImageNet-1K, as we found its performance superior to using the self-supervised pre-trained weights directly.

## B. Complete comparisons with V1

In Tables 7 and 8, we present detailed experiment-level comparisons between ConvNeXt V1 [25, 31] and V2. In particular, Table 7 shows ImageNet-1K fine-tuning results using eight models: Atto, Femto, Nano, Pico, Tiny, Base, Large, and Huge, which range from low-compute (Atto, 3.7M) to large-capacity models (Huge, 660M). We see a consistent and significant improvement across all models. The best performance is achieved when the architecture is upgraded from V1 to V2 and the self-supervised learning framework FCMAE is used, demonstrating the effectiveness of the co-design. In Table 8, we present ImageNet-22K intermediate fine-tuning results. The pre-training and fine-tuning process consists of three steps: 1) FCMAE pre-training, 2) ImageNet-22K fine-tuning, and 3) ImageNet-1K fine-tuning. Here, we focus on five V2 models: Nano, Tiny, Base, Large and Huge. We see consistent improvement over the V1 counterparts. In particular, the V2 Base (86.8%/87.7%) and Large (87.3%/88.2%) models outperform the next-level model sizes of V1, which are the Large (86.6%/87.5%) and XLarge (87.0%/87.8%) models. The

| config | value |
|---|---|
| optimizer | AdamW |
| base learning rate | 6.25e-3 (B/L), 1.25e-3 (H) |
| weight decay | 0.05 |
| optimizer momentum | $\beta_1, \beta_2$=0.9, 0.999 |
| layer-wise lr decay [2, 13] | 0.6 (B/L), 0.75 (H) |
| batch size | 1024 |
| learning rate schedule | cosine decay |
| warmup epochs | 20 (B/L), 10 (H) |
| training epochs | 100 (B/L), 50 (H) |
| augmentation | RandAug (9, 0.5) [16] |
| label smoothing [30] | 0.1 |
| mixup [34] | 0.8 |
| cutmix [33] | 1.0 |
| drop path [24] | 0.1 (B), 0.2 (L), 0.3 (H) |
| head init [25] | 0.001 |
| ema | 0.9999 |

Table 4. **End-to-end IN-1K fine-tuning setting for Base (B), Large (L), and Huge (H) models.**

| config | value |
|---|---|
| optimizer | AdamW |
| base learning rate | 2.5e-4 |
| weight decay | 0.05 |
| optimizer momentum | $\beta_1, \beta_2$=0.9, 0.999 |
| layer-wise lr decay [2, 13] | 0.8 (B/L/H), 0.9 (N/T) |
| batch size | 4096 |
| learning rate schedule | cosine decay |
| warmup epochs | 5 |
| training epochs | 90 |
| augmentation | RandAug (9, 0.5) [16] |
| label smoothing [30] | 0.1 |
| mixup [34] | 0.8 |
| cutmix [33] | 1.0 |
| drop path [24] | 0.(N/T), 0.1 (B/L), 0.3 (H) |
| head init [25] | 0.001 |
| ema | None |

Table 5. **End-to-end IN-22K intermediate fine-tuning settings.**

| config | value |
|---|---|
| optimizer | AdamW |
| base learning rate | 2.5e-5 |
| weight decay | 1e-8 |
| optimizer momentum | $\beta_1, \beta_2$=0.9, 0.999 |
| layer-wise lr decay [2, 13] | 0.8 (B/L), 0.85 (H), 0.9 (N/T) |
| batch size | 512 |
| learning rate schedule | cosine decay |
| warmup epochs | None |
| training epochs | 30 (B/L/H), 90 (N/T) |
| augmentation | RandAug (9, 0.5) [16] |
| label smoothing [30] | 0.1 |
| mixup [34] | None |
| cutmix [33] | None |
| drop path [24] | 0.1(N/T), 0.2 (B), 0.3 (L), 0.5(H) |
| head init [25] | 0.001 |
| ema | 0.9999 (N/T/B/L), None (H) |

Table 6. **End-to-end IN-1K fine-tuning settings (after IN-22K intermediate fine-tuning).**

| Backbone | Method | #param | FLOPs | Val acc. |
|---|---|---|---|---|
| ConvNeXt V1-A | Supervised | 3.7M | 0.55G | 75.7 |
| ConvNeXt V2-A | Supervised | 3.7M | 0.55G | 76.2 (+0.5) |
| ConvNeXt V2-A | FCMAE | 3.7M | 0.55G | **76.7** (+1.0) |
| ConvNeXt V1-F | Supervised | 5.2M | 0.78G | 77.5 |
| ConvNeXt V2-F | Supervised | 5.2M | 0.78G | 78.0 (+0.5) |
| ConvNeXt V2-F | FCMAE | 5.2M | 0.78G | **78.5** (+1.0) |
| ConvNeXt V1-P | Supervised | 9.1M | 1.37G | 79.5 |
| ConvNeXt V2-P | Supervised | 9.1M | 1.37G | 79.7 (+0.2) |
| ConvNeXt V2-P | FCMAE | 9.1M | 1.37G | **80.3** (+0.8) |
| ConvNeXt V1-N | Supervised | 15.6M | 2.45G | 80.8 |
| ConvNeXt V2-N | Supervised | 15.6M | 2.45G | 81.2 (+0.4) |
| ConvNeXt V2-N | FCMAE | 15.6M | 2.45G | **81.9** (+1.1) |
| ConvNeXt V1-T | Supervised | 28.6M | 4.47G | 82.1 |
| ConvNeXt V2-T | Supervised | 28.6M | 4.47G | 82.5 (+0.4) |
| ConvNeXt V2-T | FCMAE | 28.6M | 4.47G | **83.0** (+0.9) |
| ConvNeXt V1-B | Supervised | 89M | 15.4G | 83.8 |
| ConvNeXt V1-B | FCMAE | 89M | 15.4G | 83.7 |
| ConvNeXt V2-B | Supervised | 89M | 15.4G | 84.3 (+0.5) |
| ConvNeXt V2-B | FCMAE | 89M | 15.4G | **84.9** (+1.1) |
| ConvNeXt V1-L | Supervised | 198M | 34.4G | 84.3 |
| ConvNeXt V1-L | FCMAE | 198M | 34.4G | 84.4 |
| ConvNeXt V2-L | Supervised | 198M | 34.4G | 84.5 (+0.2) |
| ConvNeXt V2-L | FCMAE | 198M | 34.4G | **85.8** (+1.5) |
| ConvNeXt V2-H | FCMAE | 660M | 115G | **86.3** |

Table 7. **ImageNet-1K fine-tuning results** with a single 224×224 crop. The improvement over the V1 supervised model is shown in parentheses.

| Backbone | image size | #param | FLOPs | Val acc. |
|---|---|---|---|---|
| ConvNeXt V2-N | $224^2$ | 15.6M | 2.45G | **82.1** |
| ConvNeXt V2-N | $384^2$ | 15.6M | 7.21G | **83.4** |
| ConvNeXt V1-T | $224^2$ | 28.6M | 4.47G | 82.9 |
| ConvNeXt V2-T | $224^2$ | 28.6M | 4.47G | **83.9**(+1.0) |
| ConvNeXt V1-T | $384^2$ | 28.6M | 13.1G | 84.1 |
| ConvNeXt V2-T | $384^2$ | 28.6M | 13.1G | **85.1**(+1.0) |
| ConvNeXt V1-B | $224^2$ | 89M | 15.4G | 85.8 |
| ConvNeXt V2-B | $224^2$ | 89M | 15.4G | **86.8**(+1.0) |
| ConvNeXt V1-B | $384^2$ | 89M | 45.2G | 86.8 |
| ConvNeXt V2-B | $384^2$ | 89M | 45.2G | **87.7**(+0.9) |
| ConvNeXt V1-L | $224^2$ | 198M | 34.4G | 86.6 |
| ConvNeXt V2-L | $224^2$ | 198M | 34.4G | **87.3**(+0.7) |
| ConvNeXt V1-L | $384^2$ | 198M | 101.1G | 87.5 |
| ConvNeXt V2-L | $384^2$ | 198M | 101.1G | **88.2**(+0.7) |
| ConvNeXt V1-XL | $224^2$ | 350M | 60.9G | 87.0 |
| ConvNeXt V1-XL | $384^2$ | 350M | 179.0G | 87.8 |
| ConvNeXt V2-H | $384^2$ | 660M | 337.9G | **88.7** |
| ConvNeXt V2-H | $512^2$ | 660M | 600.8G | **88.9** |

Table 8. **ImageNet-22K intermediate fine-tuning results** with a single 224×224 crop. The improvement over the V1 supervised model is shown in parentheses.

V2 Huge model also achieves a new state-of-the-art with a performance of 88.9%. Our proposal demonstrates that pure convolutional models can also be strong and scalable vision learners with mask-based pre-training.

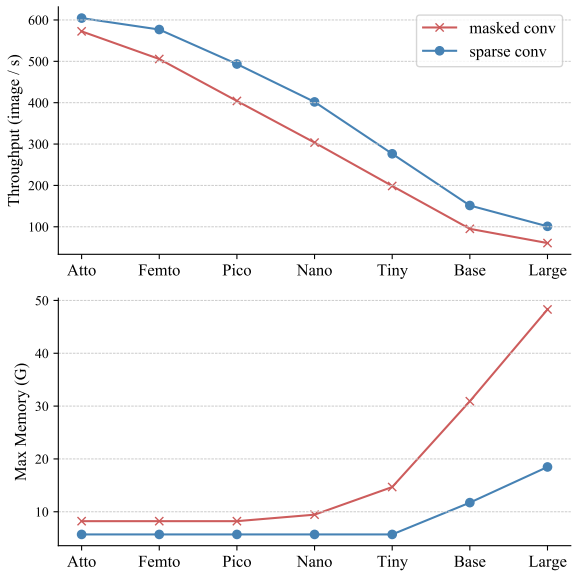


Figure 1. **Sparse encoding efficiency.** Under the pre-training setup, we measure the training throughput (image/s) and max GPU memory usage (G). The per GPU batch size is 64, and the throughput values are measured using 20 forward and backward steps. Our results show that the sparse convolution-based encoder allows for improved pre-training efficiency compared to the dense masked convolution-based counterpart.

## C. Further Analyses

**Sparse encoding efficiency.** One of the key design choices in our FCMAE framework is the use of sparse convolution [12, 19, 20] during pre-training. The primary purpose is to block the flow of information from the masked region and facilitate masked autoencoder pre-training. As a byproduct, it also offers improved computational and memory efficiency during pre-training, as the kernels only apply to the visible pixels. However, we note that the sparse convolution libraries [12, 15] are not highly optimized for modern hardware, and the efficiency achieved usually depends on the frameworks [1, 4, 29] used in practice.

To better understand the actual pre-training efficiency achieved using sparse convolution, we conducted benchmark experiments using a controlled setup with Minkowski Engine v0.5.4 [12] and PyTorch [29]. We simulated the pre-training masked input (image size 224×224, masking ratio 0.6, mask size 32×32) and compared the training throughput (image/s) and max GPU memory usage (G) between the sparse convolution-based and dense masked convolution-based encoders. While the results may vary depending on the experimental environment (we used PyTorch V1.8.0,

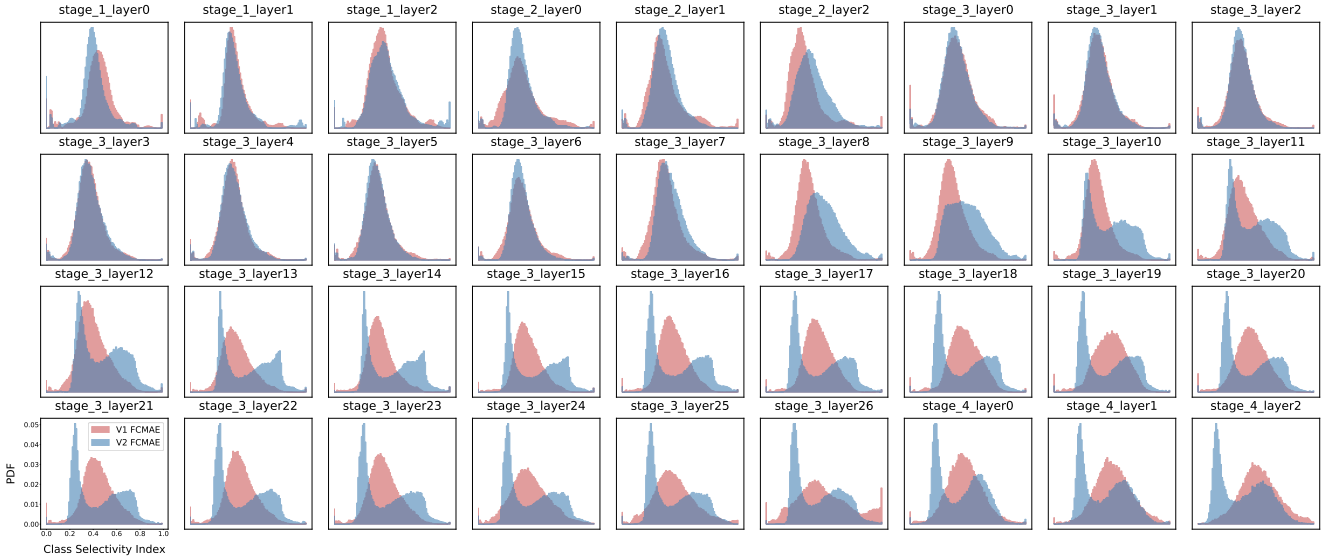


Figure 2. **Class selectivity index distribution.** The x-axis and y-axis show the class selectivity index and its density (PDF), respectively. Using the ImageNet-1K validation dataset, we calculated the class selectivity index distribution of both FCMAE pre-trained ConvNeXt V1 (red) and V2 (blue). While they tend to match closely in the early stages, the distribution becomes different in the deep layers. V2 tends to include more class-generic features in the later stages.

CUDA 11.1, CuDNN 8.2, and NVIDIA RTX A6000 GPU), we observed a moderate increase in pre-training efficiency, with an average of 1.3× increase in throughput and a 2× decrease in max memory usage across the models. The gap becomes more salient as the model size increases.

**Class Selectivity Index.** FCMAE pre-trained ConvNeXt V2 has a distinctive feature characteristic compared to V1. We conducted a class selectivity index analysis on the FCMAE pre-trained weights for ConvNeXt V1 and V2 to understand this. The class selectivity index is a metric that measures the difference between the highest class-conditional mean activity and all other class-conditional mean activities. The final normalized value lies between 0 and 1, with 1 indicating that a filter activates only for a single class and 0 indicating that the filter activates uniformly for all classes. In Figure 2, we plot the class selectivity index distribution for all intermediate layers in the model, using the output of every residual block. The distribution is closely matched between V1 and V2 in the early stages, but they begin to diverge in the deep layers, such as stage 3 layer 12. As the layer becomes deeper, the plot shows that V2 (bimodal) tends to include more class-generic features than V1 (unimodal). Since class-agnostic features are more transferrable [28], this leads to better fine-tuning performance in downstream tasks. We leave more explorations as a future study.

| case | GRN functions: aggregation | GRN functions: normalization | Val acc. |
|---|---|---|---|
| base | - | - | 83.7 |
| **(a)** | X | - | 83.9 |
| **(b)** | $X * \mathcal{G}(X)$ | ✓ | 83.9 |
| **(c)** | $X * \mathcal{N}(X)$ | - | ✓ unstable |
| **(d)** | $X * \mathcal{N}(\mathcal{G}(X))$ | ✓ | ✓ 84.6 |

Table 9. **GRN component analysis**. We report the fine-tuning performance after the 800 epoch FCMAE pre-training. Here, affine parameters and residual connection are omitted for clarity. The base denotes the ConvNeXt V1 fine-tuning performance. The **aggregation** and **normalization** are spatial L2-norm pooling and channel-wise divisive normalization, respectively. Case-(a) indicates a simple baseline of channel-wise scaling and shifting (with affine parameters) without explicit feature normalization.

## D. Additional Experiments

**GRN component analysis.** The proposed Global Relation Network (GRN) consists of three steps: global feature aggregation, feature normalization, and feature calibration. The main paper demonstrates that the combination of L2-norm based aggregation and divisive normalization works well in practice. Table 9 verifies the individual contribution of these components using ConvNeXt V2-Base as the encoder. When either component is dropped, performance significantly decreases, and the training becomes unstable if feature normalization is not preceded by global aggregation. This supports the idea that both operations work together to make GRN effective.

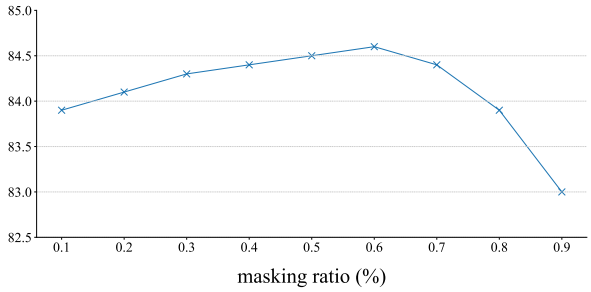


Figure 3. **Masking ratio.** We observe that a masking ratio of 0.6 provides the best result. The y-axis is ImageNet-1K accuracy (%).

**Masking ratios.** We conduct a hyper-parameter analysis on the masking ratio for a mask size of 32 × 32. The results, shown in Figure 3, suggest that a masking ratio in the range of 0.5 to 0.7 produces the best results, with a masking ratio of 0.6 providing the highest performance. The model's performance declines at the two extremes of either removing or leaving 90% of the input information, although it is more robust when more information is retained.

**Comparison with contrastive SSL.** In this work, we compare the performance of the two dominant self-supervised learning (SSL) approaches: contrastive learning [5, 6, 9–11, 21, 23] and masked image modeling [2, 22, 32]. Specifically, we compare the end-to-end fine-tuning performance of MoCoV3 [11], the current state-of-the-art contrastive learning method, with our proposed FCMAE framework using the same ConvNeXt V2-Base as the encoder. We follow the default pre-training and fine-tuning recipes for each approach and present the results below.

| Sup, 300ep. | MoCo V3 | FCMAE |
|---|---|---|
| 84.3 | 83.7 | 84.9 |

We use the 300-epoch supervised learning baseline as a reference. The above table shows that FCMAE leads to better representation quality than MoCo V3 and also outperforms the supervised baseline. This is consistent with the recent observations that masked image modeling offers superior results over contrastive learning-based SSL for end-to-end fine-tuning. In this work, this success was also made possible with pure ConvNets.

**Understanding feature collapse.** Given the same masked image modeling task, the fundamental difference between convolution (*sliding window* with local kernels) and self-attention (*set operator*) leads to different learning behaviors. If the Conv kernel size is too big (*e.g.* larger than the masked region), instead of learning useful features, convolution can "cheat" the MAE task by simply copying/interpolating neighbor patterns, giving the network no incentive to learn diverse features across channels. Due to spatial downsampling, this degradation happens more in later stages of the network, which also explains the Fig 4 in the main paper: feature diversity decreases along the layers.

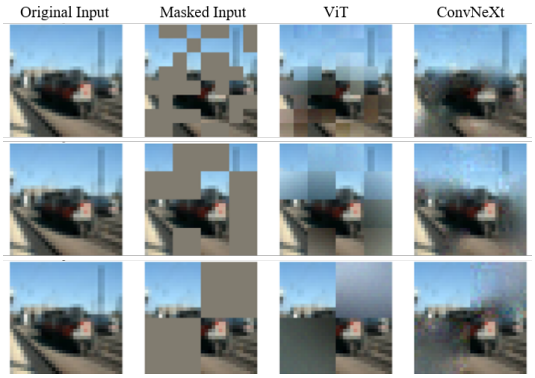


Figure 4. **MAE reconstruction** with different mask sizes: 4×4 (top), 8×8 (middle), and 16×16 (bottom)

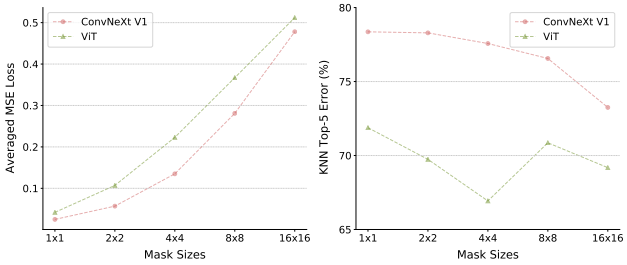


Figure 5. **Analysis on reconstruction error and representation quality**.

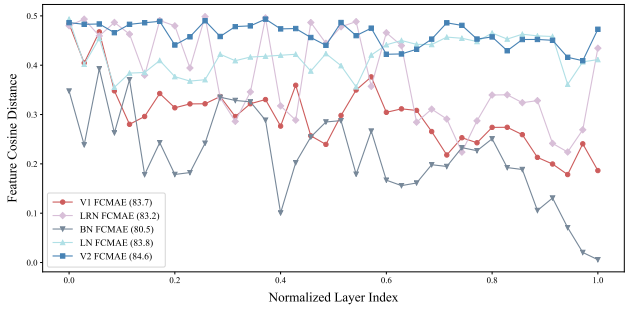


Figure 6. **More feature cosine distance analysis.**

To better understand this phenomenon and provide more evidence, we conduct further experiments and analyses. For this toy experiments, we use CIFAR 100 (32×32 images); We test ConvNeXt encoder (6 blocks, with 7×7 kernels) and Transformer encoder (4 blocks), both of which have 0.6M params; We vary the mask size ([1,2,4,8,16]) and use a masking ratio of 0.5; We train the models for 50 epochs. The representation quality is evaluated by computing the top-5 error from a KNN classifier (K=20).

In Figure 4, we present the reconstruction results obtained by ViT and ConvNeXt on inputs with different mask sizes. ViT has been known to produce block-structure artifacts in its reconstructions, whereas the reconstructions generated by ConvNeXt are significantly smoother. In terms of quantitative evaluation, the mean squared errors (MSE) for ConvNeXt are generally smaller than those for ViT (as shown in Figure 5-left).

Although ConvNeXt produces smaller mean squared errors (MSE) than ViT in the reconstruction task (as shown in Figure 5-left), it is important to note that this outcome has a potential downside. As demonstrated in Figure 5-right, there is a negative correlation between the reconstruction loss and the quality of representation learning (top-5 err) when evaluating the learned representation quality using K-NN (K=20). This suggests that the ConvNeXt may prioritize ease of reconstruction over learning more semantically-clustered, meaningful features, which could lead to the learning of relatively trivial features.

While increasing the mask size can make the task more challenging for ConvNeXt and help prevent trivial solutions, it can also lead to information loss, eventually resulting in no visible region at all. As a consequence, the quality of representation learning may ultimately decrease. In contrast, our GRN design explicitly promotes diverse high-dimensional features during the MAE pre-training and does not rely on a specific mask design.

**Additional feature cosinedistance analysis.** We conduct additional feature cosine distance analysis with different normalization layers, including Layer Normalization (LN), Batch Normalization (BN), and Local Response Normalization (LRN) in Fig 6. The overall tendencies are consistent with the final fine-tuning performances: BN exhibits a worst feature-collapse phenomenon, LRN generates an irregular diversity pattern, and LN mitigates the issue to some extent but with generally lower diversity than GRN.

## References


[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Operating Systems Design and Implementation*, 2016.

[2] Hangbo Bao, Li Dong, and Furu Wei. BEiT: BERT pre-training of image transformers. In *ICLR*, 2022.

[3] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *ICCV*, 2017.

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*, 2020.

[6] 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.

[7] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv:1906.07155*, 2019.

[8] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pre-training from pixels. In *ICML*, 2020.

[9] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[10] Xinlei Chen and Kaiming He. Exploring simple Siamese representation learning. In *CVPR*, 2021.

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

[12] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019.

[13] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*, 2020.

[14] MMSegmentation contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020.

[15] Spconv Contributors. Spconv: Spatially sparse convolution library. https://github.com/traveller59/spconv, 2022.

[16] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPRW*, 2020.

[17] 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.

[18] 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.

[19] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *CVPR*, 2018.

[20] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

[21] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap your own latent - a new approach to self-supervised learning. In *NeurIPS*, 2020.

[22] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

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

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

[25] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, 2022.

[26] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

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

[28] Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *ICLR*, 2018.

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NeurIPS*, 2017.

[30] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[31] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[32] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: A simple framework for masked image modeling. In *CVPR*, 2022.

[33] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

[34] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.