Compressing Transformers: Features Are Low-Rank, but Weights Are Not!

Hao Yu, Jianxin Wu*

State Key Laboratory for Novel Software Technology, Nanjing University, China
yuh@lamda.nju.edu.cn, wujx2001@nju.edu.cn

Abstract

Transformer and its variants achieve excellent results in various computer vision and natural language processing tasks, but high computational costs and reliance on large training datasets restrict their deployment in resource-constrained settings. Low-rank approximation of model weights has been effective in compressing CNN models, but its application to transformers has been less explored and is less effective. Existing methods require the complete dataset to fine-tune compressed models, which are both time-consuming and data-hungry. This paper reveals that the features (i.e., activations) are low-rank, but model weights are surprisingly not low-rank. Hence, AAFM is proposed, which adaptively determines the compressed model structure and locally compresses each linear layer's output features rather than the model weights. A second stage, GFM, optimizes the entire compressed network holistically. Both AAFM and GFM only use few training samples without labels, that is, they are few-shot, unsupervised, fast and effective. For example, with only 2K images without labels, 33% of the parameters are removed in DeiT-B with 18.8% relative throughput increase, but only a 0.23% accuracy loss for ImageNet recognition. The proposed methods are successfully applied to the language modeling task in NLP, too. Besides, the few-shot compressed models generalize well in downstream tasks.

Introduction

The transformer architecture (Vaswani et al. 2017) has been widely used in the natural language processing (NLP) area over the past years. Inspired by its excellent performance in NLP, transformer-based models have established numerous new records in various computer vision (CV) tasks, such as image classification (Dosovitskiy et al. 2021) and object detection (Liu et al. 2021). Despite these progresses, most of these transformer-based structures suffer from large model sizes, huge run-time memory consumption and high computational costs, which prohibit the model deployment to resource-constrained platforms. Therefore, compressing transformer-based models has attracted immense interests in recent years.

Low-rank approximation is a useful technique to strike a balance between model accuracy and model size. Some previous NLP research efforts (Noach and Goldberg 2020; Hsu et al. 2022) focus on exploring the factorization of model weights. These compression methods have been very successful in reducing model size and to speedup inference. But, the accuracy drops dramatically, hence fine-tuning for many epochs using the entire training set is needed to partly recover from the accuracy loss. Nevertheless, to protect data privacy and/or achieve rapid deployment, in many scenarios there may be only the original model and *a small number of samples available* under a tight compression time budget. Fine-tuning a deep learning model with limited data will easily lead to overfitting, which invalidates these existing methods. Furthermore, these methods simply compress the large transformer model itself, lacking the exploration of how the compressed models perform on downstream tasks. Lastly, automatically determining the compression ratio of each layer is one of the main difficulties in low-rank decomposition, but efforts in this aspect remain scarce.

Hence, we believe that in order to successfully perform low-rank approximation on transformers, we need an *effective* approach which can be easily applied in *both* NLP and CV, which only requires *few* samples and *short* compression time, *adaptively* determines the compression model structure and *generalizes* well to downstream tasks.

To fulfill these goals, our key finding is that although in linear layers of transformers *the weight matrices are almost full rank* (i.e., not suitable for decomposition), *the features (i.e., activations) are generally low-rank*. Therefore, we propose a novel Atomic Feature Mimicking (AFM) approach to replace traditional weight approximation methods locally. Besides, since different layers have different compression sensitivity, we design an adaptive search method to determine the compressed model architecture, namely Adaptive AFM (AAFM), for low-rank decomposition. The approximation will accumulate errors along with the depth increment, so we propose Global Feature Mimicking (GFM) to fine-tune the compressed models with only a small number of unlabeled samples. Our contributions are:

- We propose a novel and effective framework for low-rank approximation of transformers, with the key finding and novelty being mimicking the features rather than the weights. Extensive experiments demonstrate that our methods can compress both vanilla transformers and its variants in CV *and* NLP. With fewer parameters and

*J. Wu is the corresponding author.

higher throughput, our compressed models achieve comparable accuracy to the original models.

- Our framework is few-shot, unsupervised and swift. Given a pre-set compression target and a small number of unlabeled samples, our methods can *quickly* determine the sub-model architecture and fine-tune the compressed model. *No extra hyper-parameters* are introduced.
- The compressed model can be generalized into various downstream tasks. Experimental results show that even under the few-shot settings, our methods achieve excellent generalization of the compressed model.

Related Works

We first briefly review some closely related works.

Transformers

The transformer (Vaswani et al. 2017) utilizes the multi-head self-attention (MHSA) mechanism to handle long-range dependencies between pairs of input tokens in NLP tasks. Language models constructed with transformers have achieved widespread success in NLP tasks such as translation (Vaswani et al. 2017), language modeling (Baevski and Auli 2018) and question answering (Kenton and Toutanova 2019). Transformers have recently been introduced into the CV field, too. ViT (Dosovitskiy et al. 2021) first demonstrated that with sufficient training data (e.g., JFT-300M), standard transformers can achieve state-of-the-art accuracy in image classification tasks. DeiT (Touvron et al. 2021) further explores existing data augmentation and regularization strategies. With the same architecture as ViT, DeiT is also effective when using the smaller ImageNet-1K dataset. Swin Transformer (Liu et al. 2021) applies a shifted windowing scheme. In this paper, we will show that our feature-mimicking approach can handle standard transformers and their variants in both CV and NLP.

Low-Rank Approximation for Transformers

Transformer models tend to have a large number of parameters and are computationally intensive. To reduce model size and speed up inference, a natural idea is to factorize one weight matrix into two or more smaller matrices. A common technique for low-rank factorization is Singular Value Decomposition (SVD) (Golub and Van Loan 2013), which can be applied to any linear layer. Noach and Goldberg (2020) first decomposed each weight matrix by SVD. Then, they used knowledge distillation to refine the weights. Drone (Chen et al. 2021) minimizes the approximation error of the input representations layer-by-layer by exploiting the full dataset. Later we show that it is a better idea to approximate the output vector at low rank. FWSVD (Hsu et al. 2022) introduces Fisher information to measure the importance of parameters. But it requires a large amount of labeled data to fine-tune the compressed model, and the weight matrices are not necessarily low rank—in fact they are often full rank instead. We perform low-rank decomposition on the output feature maps rather than the weights, which requires only a small number of unlabeled samples. To the best of our knowledge, although weight matrix decomposition is widespread, this is the first attempt to decompose transformers' features under the few-shot unsupervised settings.

Feature Mimicking

Knowledge Distillation (KD) (Hinton, Vinyals, and Dean 2015) is a popular method to train student networks with the help of high-capability teacher networks. The typical KD approach attempts to refine the softmax outputs, and the feature mimicking distillation method was first proposed by FitNets (Romero et al. 2014). Wang, Ge, and Wu (2021) demonstrate that it is more beneficial to make students only mimic the teacher's features in the penultimate layer. Feature mimicking is also often used for model compression. FSKD (Li et al. 2020) adds a 1×1 convolutional layer after each layer and mimics the one-layer feature maps by solving a least-square problem. CD (Bai et al. 2020) optimizes model layer-by-layer by cross distillation between the compressed and the original networks. MiR (Wang et al. 2022) is a global distillation method that only mimics one layer's feature map. Our framework combines their ideas, i.e., first, we mimic the single-layer feature maps and then distill the global features. Numerous experiments demonstrate the effectiveness of our approach.

The Proposed Methods

We describe our framework in this section, starting from the preliminaries. Then, we present our low-rank decomposition method, Atomic Feature Mimicking (AFM). To automatically determine the structure of the compressed model, we propose Adaptive Atomic Feature Mimicking (AAFM), a simple but effective low-rank approximation framework. Finally, we apply Global Feature Mimicking (GFM) to minimize the output features' difference. Throughout the compression process, our framework requires only a small number of unlabeled samples.

Preliminaries

We first introduce the classical low-rank decomposition method. Suppose we have a fully-connected (FC) layer, whose input is a matrix $x \in \mathbb{R}^{n\times c}$ and output is another matrix $y \in \mathbb{R}^{m\times c}$. The relationship between them is simple:

$$y = Wx + b\,, \tag{1}$$

where $W \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. A standard way to accelerate this computation is to perform low-rank approximation of W, i.e.,

$$y \approx W_2(W_1x + b_1) + b_2\,, \tag{2}$$

where $W_1 \in \mathbb{R}^{k\times n}$, $b_1 \in \mathbb{R}^k$ and $W_2 \in \mathbb{R}^{m\times k}$, $b_2 \in \mathbb{R}^m$. The solution of Eq. 2 can be obtained by applying SVD on W. We can decompose W into

$$W = USV^T\,, \tag{3}$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthonormal matrices. $S \in \mathbb{R}^{m\times n}$ is a diagonal rectangular matrix containing singular values in the decreasing order. If we only use the largest k terms of the singular values, the resulting matrix is an optimal approximation of W with a lower rank $k < n$:

$$W \approx U'V'^T\,, \tag{4}$$

where $U' \in \mathbb{R}^{m\times k}$ and $V' \in \mathbb{R}^{k\times n}$ are the rank-k approximation matrices by taking $U' = US_k^{\frac{1}{2}}$ and $V' = VS_k^{\frac{1}{2}}$, and $S_k^{\frac{1}{2}}$ is a diagonal matrix formed by the square-roots of the corresponding top k singular values in S. Then, the original Eq. 1 can be approximated as:

$$y = Wx + b \approx U'V'^T x + b. \tag{5}$$

After this low-rank approximation, the number of parameters in this linear layer decreases from $O(mn)$ to $O((m+n)k)$. However, in many models, particularly transformer and its variants, the weight W is nearly full rank (cf. Figure 1). In other words, if we separate one FC into two by SVD, we have to either choose a small k but endure *large accuracy drop*, or use a large k but *increase* the model size.

Atomic Feature Mimicking (AFM)

Now we propose Atomic Feature Mimicking (AFM). Instead of decomposing the model weights, our AFM aims to factorize the output features. Since we only mimic the feature within one single layer and do not involve any other layers, it is *atomic*.

Specifically, we follow the notation described in Wu (2020). Let us treat the output feature in $\mathbb{R}^{m\times c}$ as c instantiations of the random feature vector y (each in $\mathbb{R}^m$), and compute the covariance matrix:

$$\mathrm{Cov}(y) = \mathbb{E}\left[yy^T\right] - \mathbb{E}[y]\mathbb{E}[y]^T, \tag{6}$$

where $\mathbb{E}[\cdot]$ is the expectation operator. Since $\mathrm{Cov}(y)$ is positive semi-definite, its eigendecomposition (i.e., the principal component analysis or PCA) is

$$\mathrm{Cov}(y) = USU^T. \tag{7}$$

We only keep the top k eigenvalues and extract the first k columns of $U \in \mathbb{R}^{m\times m}$ into $U_k \in \mathbb{R}^{m\times k}$ and $U_kU_k^T \approx I$, hence

$$y - \mathbb{E}[y] \approx U_kU_k^T(y - \mathbb{E}[y]), \text{ or,} \tag{8}$$

$$y \approx U_kU_k^T y + \mathbb{E}[y] - U_kU_k^T\mathbb{E}[y]. \tag{9}$$

That is, one linear layer can be transformed into two:

$$y \approx U_kU_k^T(Wx + b) + \mathbb{E}[y] - U_kU_k^T\mathbb{E}[y], \tag{10}$$

$$= U_k(U_k^TWx + U_k^Tb) + \mathbb{E}[y] - U_kU_k^T\mathbb{E}[y], \tag{11}$$

where the first FC layer has weights $U_k^TW \in \mathbb{R}^{k\times n}$ and bias $U_k^Tb \in \mathbb{R}^k$, and the second one has weights $U_k \in \mathbb{R}^{m\times k}$ and bias $\mathbb{E}[y] - U_kU_k^T\mathbb{E}[y] \in \mathbb{R}^m$.

Algorithm 1 presents the details of AFM. In order to calculate the covariance matrix, we randomly select a small number of samples from the training dataset to establish a proxy dataset $\mathcal{D}$. In only one forward execution, we can gather the output features for computing the covariance matrices in all FC layers and decompose all FCs. Note that we adaptively update $\mathbb{E}[yy^T]$ and $\mathbb{E}[y]$ in a streaming fashion instead of storing all output features. As our experiments will show later, with only few samples we can compute weights with good generalization and will not significantly decrease the model accuracy.

Algorithm 1 Atomic Feature Mimicking

Input: The original model $\mathcal{M}$ with weights W and bias b in the i-th layer, the proxy dataset $\mathcal{D}$ and a pre-set rank k.
Output: Two compressed FC layers with weights W_1 and W_2, and biases b_1 and b_2.

1: **for** each sample x in $\mathcal{D}$ **do**
2: Forward propagate $\mathcal{M}(x)$ to obtain the output feature y in the i-th layer and update $\mathbb{E}[yy^T]$ and $\mathbb{E}[y]$.
3: **end for**
4: Calculate the eigenvectors U based on Eq. 6 and Eq. 7.
5: Extract the first k columns of U into U_k, and obtain $W_1 = U_k^TW$, $b_1 = U_k^Tb$, $W_2 = U_k$, and $b_2 = \mathbb{E}[y] - U_kU_k^T\mathbb{E}[y]$.
6: **return** (W_1, b_1), (W_2, b_2)

Figure 1 explains why we should approximate the features rather than the weights. We send the entire ImageNet-1K (Deng et al. 2009) validation set into DeiT-B. DeiT models contain two components, the attention layer and the Feed-Forward Network (FFN). Each component contains two FC layers, which we refer to as QKV and PROJ in the attention layer, plus FC1 and FC2 in the FFN. In particular, we collect the input and output features of the QKV and FC1 layers in every block. Then we compute the covariance matrix of these features separately and calculate the eigenvalues. We also decompose the weights of QKV and FC1 by SVD. Figure 1 shows the percentages of eigen or singular vectors we need to keep in order to reach 90% of the energy for them. As we can see, when 90% of the energy is retained, the dimensionality required for the output features is less than the input features and model weights, which shows that output features are more likely low-rank (i.e., decomposition friendly) but the model weights are not.

Adaptive Atomic Feature Mimicking (AAFM)

As aforementioned, one great challenge in low-rank decomposition is to accurately determine the ranks k retained by different layers. We propose Adaptive AFM (AAFM) to overcome this difficulty. The basic idea behind AAFM is to keep higher rank or even not compress those more sensitive layers, while adopting a more aggressive compression strategy for those less sensitive ones.

To measure a layer's sensitivity score, we apply the proxy dataset $\mathcal{D}$ and extract the output logits of the original model $\mathcal{M}$. Then we evaluate the performance change before/after applying AFM in a single layer. To maximize the GPU utilization and to reduce the search overhead, we fix rank k to be a multiple of 32. Then, we compute the sensitivity score for each layer separately, as the KL-Divergence between two models with and without AFM, i.e.,

$$S_i(k) = \sum_{x\in\mathcal{D}} D_{\mathrm{KL}}(\mathcal{M}(x, w) \| \mathcal{M}(x, w_i(k))), \tag{12}$$

where $S_i(k)$ measures how sensitive the i-th layer is when the rank is k, and $w_i(k)$ refers to compress model weights in the i-th layer with rank k. The larger the score $S_i(k)$, the more sensitive this layer is.

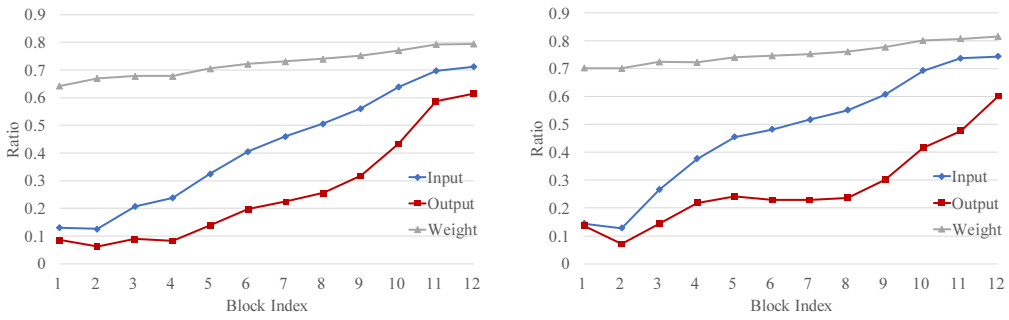

Figure 1: Ratio of dimensions kept in each QKV (left) and FC1 (right) layer in DeiT-B when 90% energy is retained. The output features (shown in red) have the lowest proportion compared to the input features (shown in blue) and the weights (shown in gray). The x-axis is the block index. DeiT-B has 12 blocks, and every block contains a QKV and an FC1 layer.

After obtaining sensitivity scores with different rank k, given a target model size P_{tar}, we minimize the sum of sensitivity scores of all layers, i.e.,

$$\min_{\{k_i\}_{i=1}^{l}} \quad S = \sum_{i=1}^{l} S_i(k_i) \tag{13}$$

$$\text{s.t.} \quad \sum_{i=1}^{l} P_i(k_i) \leq P_{tar}\,, \tag{14}$$

where $P_i(k_i)$ is the number of parameters for the i-th layer with rank k_i, and l is the total number of linear layers. Our AAFM can generate multiple sub-networks from a well-trained large model when a set of different targets P_{tar} are given, which is versatile.

As k_i are integers, this is an integer programming problem and we approximately solve it by proposing a simple greedy algorithm. It is worth noting that we have made a simplifying assumption: the sensitivity of a layer is independent of the ranks chosen for other layers. Our greedy algorithm is not guaranteed to find the best possible configuration. However, finding the precise global minimum from all potential rank configurations is very time-consuming, and our simplified greedy approximation can greatly speedup the search process. Later, we will demonstrate that the adaptive configuration we find via the greedy algorithm achieves excellent results in experiments for different model architecture and in diverse tasks.

Global Feature Mimicking (GFM)

Although the AAFM reconstruction error is small in one layer, they accumulate as more linear layers are approximated. Hence, following MiR (Wang et al. 2022), our final step is to use Global Feature Mimicking (GFM) to correct them after applying AAFM.

GFM is very simple: with the few-shot examples (i.e., the proxy dataset $\mathcal{D}$), we mimic the output feature in the penultimate layer (before the GAP layer). We use the mean squared error to measure the distance and the optimization is

$$\mathcal{L}_{\text{MSE}}\left(f_c^L, f_o^L\right)\,, \tag{15}$$

where f_c and f_o are the features of the compressed network and the original network, respectively, and L is the index of the layer whose features are mimicked. In ViTs, f^L is the output feature map after the final LayerNorm layer; in the language modeling task, it is the feature map after the final block and before the adaptive softmax layer. Note that GFM does *not* involve the classification FC layer. We will empirically show that even in the few-shot setting, fine-tuning the compressed network by GFM will not lead to overfitting, and can be very helpful in boosting the accuracy.

In summary, our framework first uses AAFM to determine the sub-model structure, and applies Algorithm 1 (AFM) to decompose it, and finally uses GFM to fine-tune the compressed model. Our framework only requires the unlabeled proxy dataset $\mathcal{D}$, which is unsupervised. Because the proxy dataset $\mathcal{D}$ contains only a small number of samples, our approach is few-shot and fast. We only need a pre-defined compression target to determine the model architecture and parameters adaptively, so no additional hyper-parameters are introduced.

Experiments

We now evaluate our methods. We first compress DeiT and Swin Transformers on ImageNet-1K classification. In addition, more results on downstream small-scale classification and object detection datasets will be presented. We also evaluate our framework on a language modeling task. We will demonstrate that our approaches attain equivalent accuracies with significantly fewer parameters on these tasks. Finally, we end this section with several analyses. All the experiments were conducted with PyTorch.

Datasets and Metrics

Classification. The ImageNet-1K (Deng et al. 2009) dataset consists of 1.28 million training and 50K validation images. Those images have various spatial resolutions and come from 1K different categories. ImageNet-1K is usually used as the benchmark for model compression.

Besides ImageNet-1K, we also evaluate our compressed models on several small-scale datasets.

Objection Detection & Segmentation. We evaluate object detection & segmentation performance on the MS COCO2017 (Lin et al. 2014) dataset. MS COCO2017 con-

tains 80 categories with 118K training and 5K validation images, respectively. We use mean Average Precision (mAP) to measure the accuracy.

Language Modeling. We also evaluate our approach in the WikiText-103 (Merity et al. 2017) dataset. WikiText-103 is composed of shuffled Wikipedia articles where the context carries across sentences. The training data of WikiText-103 comprises about 100M tokens with 28K articles and a vocabulary of around 260K. The test data contains 245K tokens with 4358 sentences. We use perplexity to measure the performance of models. A lower perplexity indicates the probability distribution is good at predicting the sample.

Compressing DeiT & Swin

As mentioned before, our method needs a proxy dataset $\mathcal{D}$ to calculate the compressed weights, so we first sample 2K images from the ImageNet-1k training dataset to form it. Then, since larger models tend to have more parameter redundancy, we test the performances of AAFM and GFM on DeiT-B and Swin-B & L.

Implementation details. First, we set three different compression levels when applying AAFM. In particular, 1/5, 1/4, or 1/3 of the parameters of Swin-B & Swin-L were to be removed, while DeiT-B's model size was reduced by 1/5, 1/3, or 2/5, respectively. We only compressed the four FC layers in the blocks and used eight NVIDIA 3090 GPUs to calculate the sensitivity scores. The whole AAFM process took 0.6 hours when compressing DeiT-B. As a baseline or comparison method for AAFM, we also performed SVD on the original transformer model, i.e., we retain half of the singular values for each FC layer in the transformer's blocks.

Then we fine-tuned the sub-models with GFM on the proxy dataset. When fine-tuning DeiT-B, we initialized the learning rate as 8e-5 and used a mini-batch size of 512. When we fine-tuned Swin-B & Swin-L, we set the learning rate and mini-batch size as 3e-5 and 256, respectively. In the above experiments, we used the AdamW (Loshchilov and Hutter 2018) optimizer and the cosine decay schedule (Loshchilov and Hutter 2017). The sub-models were fine-tuned with 1000 epochs and the weight decay was 0.01. Since the proxy dataset $\mathcal{D}$ is small, fine-tuning 1000 epochs is still very fast. Random horizontal flipping, color jittering, Mixup (Zhang et al. 2018) and CutMix (Yun et al. 2019) were applied as data augmentations. Particularly, Yu et al. (2021) found that strong regularization has a negative influence on model performance in the later training period, and we also noticed a similar observation during the GFM process. Hence, in all of our experiments, we removed random erasing (Zhong et al. 2020), Rand-Augment (Cubuk et al. 2020) and layer dropout (Huang et al. 2016). When fine-tuning the DeiT-B with 40% parameters removed, the entire GFM process consumed 0.6 hours. Therefore, our AAFM and GFM feature mimicking framework can finish fairly quickly.

Results. Table 1 shows the results of compressing DeiT-B and Swin-B & Swin-L. We tested model accuracy on the ImageNet-1K validation dataset. During testing, the shorter side was resized as 256 by bilinear interpolation and then we cropped the 224×224 image patch in the center. The accuracy of the last epoch is reported. We also list the throughput in a 3090 GPU with a fixed 512 mini-batch size.

Model	Throughput	#Param.(M)	Acc. (%)
DeiT-B	619.46	86.57	81.85
+SVD	741.07 (+19.6%)	58.27 (-33%)	77.21
+GFM			80.36
+AAFM	682.23 (+10.1%)	69.25 (-20%)	81.76
+GFM			81.83
+AAFM	735.97 (+18.8%)	58.26 (-33%)	81.21
+GFM			81.62
+AAFM	**771.07** (+24.5%)	**51.95** (-40%)	**80.33**
+GFM			**81.28**
Swin-B	458.86	88.10	83.47
+SVD	489.95 (+6.8%)	60.20 (-33%)	74.30
+GFM			81.13
+AAFM	471.64 (+2.8%)	70.50 (-20%)	82.89
+GFM			83.19
+AAFM	477.71 (+4.1%)	66.09 (-25%)	82.41
+GFM			83.00
+AAFM	**489.46** (+6.7%)	**60.20** (-33%)	**81.15**
+GFM			**82.68**
Swin-L	257.40	196.87	86.25
+SVD	288.82 (+12.2%)	134.09 (-33%)	82.02
+GFM			84.52
+AAFM	275.14 (+6.9%)	157.52 (-20%)	85.94
+GFM			86.01
+AAFM	282.58 (+9.8%)	147.67 (-25%)	85.73
+GFM			85.83
+AAFM	**292.04** (+13.5%)	**134.09** (-33%)	**85.04**
+GFM			**85.44**

Table 1: Top-1 accuracy (%) of performing low-rank approximation on DeiT-B and Swin-B & Swin-L.

Comparison between the original models and our compressed models proves the effectiveness of our framework. We obtained a 24.5% throughput speedup and only a 0.57% accuracy reduction when we removed 40% of the parameters in DeiT-B, while we lost only 0.02% accuracy when we removed 20% of the parameters. When compressing Swin-B, we gained a 6.7% increase in throughput and a 33% decrease in parameters with dropping 0.79% accuracy. It is worth noting that our AAFM performance far exceeds that of traditional SVD methods, for example, when compressing DeiT-B and Swin-B by removing 33% of the parameters, our AAFM is 4 percentage points (81.21% vs. 77.21%) and 6.85 percentage points (81.15% vs. 74.30%) higher than that of SVD, respectively. Furthermore, Swin-L is pre-trained on the large-scale ImageNet-21K dataset, and our method has been demonstrated to be effective in it, too.

Transferring Ability

To further validate the effectiveness of our methods, we investigated the compressed models' transferring ability in several downstream tasks. We first investigated Swin-B's mAP on MS COCO2017 detection and segmentation. Then we tested the accuracy of the compressed DeiT-B in small-scale classification datasets.

Implementation details. We used both the original Swin-

Backbone	Tasks	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Swin-B	Detection	**52.0**	**70.8**	**56.4**	35.0	55.6	**67.4**
Ours		51.9	70.5	**56.4**	**35.5**	**55.8**	67.0
Swin-B	Segmentation	**45.0**	**68.3**	**48.8**	28.5	**48.6**	**60.6**
Ours		44.7	67.9	48.5	**28.8**	48.4	59.7

Table 2: mAP of Swin-B and our compressed model on the MS COCO2017 validation dataset.

Models	DeiT-B	Ours		
#Param.(M)	86.57	69.25	58.26	51.95
CIFAR-100	**90.99**	90.67	90.37	90.17
CUB-200	**85.88**	85.07	85.38	84.85
Cars	90.45	**91.18**	90.66	90.72
Aircraft	79.87	80.92	**81.19**	80.80
Pets	**94.74**	94.22	93.98	93.95
Flowers	**97.77**	97.45	97.30	97.02
iNaturalist-2019	77.39	**77.56**	76.70	77.13

Table 3: Accuracy (%) of DeiT-B and our compressed models on different small-scale classification datasets.

Dataset	Throughput	#Param.(M)	Perplexity
WikiText-103	3137.3	246.9	18.66
+SVD	3303.3 (+5.3%)	196.6 (-20%)	29.76
+GFM			20.24
+AAFM	3252.3 (+3.7%)	209.9 (-15%)	20.23
+GFM			19.07
+AAFM	3293.2 (+5.0%)	196.6 (-20%)	22.34
+GFM			19.46
+AAFM	**3356.4** (+7.0%)	**185.2** (-25%)	**26.20**
+GFM			**20.05**

Table 4: Results of compressing the standard transformer with adaptive inputs in language modeling.

B and our compressed Swin-B with 33% parameters removed as backbones of Cascade Mask R-CNN (Cai and Vasconcelos 2018). We followed the training settings of the original Swin Transformer paper and set the training schedule as 3x (36 epochs).

When fine-tuning the compressed DeiT-B, we adopted mini-batch size 1024 and learning rate 1e-4. We applied the AdamW optimizer, cosine learning rate schedule and the CutMix augmentation strategy. CutMix's ratio was set to 0.5. We trained models with 100 epochs.

Results. Table 2 shows the detection and segmentation results on MS-COCO2017. The compressed Swin-B model achieved similar mAPs compared to the original model. For example, after removing 33% of the backbone parameters, our model achieved 51.9 and 44.7 mAPs on object detection and segmentation tasks, which is on par with that of the original Swin-B.

Table 3 shows the classification results. The compressed models *always* achieved similar accuracy to DeiT-B on all 7 datasets, indicating that our method maintains the generalization ability of the original model. On the Cars, Aircraft and iNaturalist-2019 datasets, our compressed model with fewer parameters even performed better than the original DeiT-B model.

Compressing Transformer for NLP Tasks

We also compressed the standard transformer with adaptive input representations (Baevski and Auli 2018) on the WikiText-103 dataset. Our methods achieved comparable perplexity with the original model.

Implementation details. We implemented our methods based on fairseq (Ott et al. 2019). The original transformer model follows the architectural choice described in Baevski and Auli (2018), which includes 16 decoder blocks and sinusoidal position embeddings in the input layer. Each MHSA module has 8 heads and adaptive input representations have three bands of size 20K, 40K and 200K. The embedding layer and FFN's hidden-state have dimensions of 1024 and 4096, respectively. We sampled 4K sentences to form the proxy dataset $\mathcal{D}$. Then, we reduced the parameters by 15%, 20% and 25%.

During the GFM process, we removed layer dropout and trained on 8 GPUs. We limited the number of tokens per GPU to a maximum threshold 1536, which means each GPU processes 1536 tokens using the same model parameters. We accumulated gradient updates over 8 batches before committing a parameter update following Ott et al. (2018). We set the batch size as 32 and trained 1000 updates. The AdamW optimizer was used and the weight decay was 0.05. The learning rate was linearly warmed up from 1e-7 to 3e-5 for 30 steps and then annealed using a cosine learning rate schedule. We renormalized gradients if their norm exceeds 0.1. In particular, we fixed the adaptive input and softmax layer during fine-tuning.

Results. Table 4 shows the results on WikiText-103. During testing, we denoted the size of context window as 2560. Similar to previous experiments, our algorithms obtained results that are comparable to those of the original model. Especially, AAFM decreased the perplexity by 7.42 compared to SVD when removing 20% parameters.

Analyses

To explore the impact of different modules of our method, we performed three analyses in this section.

The influence of AAFM. We first explore the influence of our adaptive structure searching approach. In particular, we take Swin-B and the transformer trained in WikiText-103 as examples, and we compress these two models by 33% and 20% of parameters, respectively. We design two experiments. The first employs AFM while retaining half of the dimensions for each FC layer in model blocks, while the sec-

Model	Adaptive	Top-1 Acc. (%)	Perplexity
Original Model		83.47	18.66
+AFM	✗	78.16	22.55
+GFM	✗	81.61	**20.21**
+Adaptive SVD	✓	80.24	122.41
+GFM	✓	**82.49**	20.23

Table 5: Top-1 accuracy (%) and perplexity of exploring the impact of adaptive structure searching method on Swin-B and transformer.

ond one exploits the model structure searched by AAFM but using SVD to initialize sub-model. We refer to the second method as adaptive SVD. For a fair comparison, we adopt the same training strategies as above.

Table 5 summarizes the results. We discovered that our AFM outperformed SVD but was inferior to AAFM. For example, on Swin-B AFM achieved 78.16% accuracy, which is higher than SVD's 74.30% accuracy but lower than AAFM's 81.15% accuracy (cf. Table 1). Furthermore, the model structures we searched have good generalization. For example, our adaptive SVD obtained 80.24% accuracy on the ImageNet validation dataset, which is better than the typical SVD low-rank approximation method (77.21%, cf. Table 1).

Knowledge distillation. We then compare several different distillation strategies when fine-tuning sub-models. We continue to use the Swin-B and transformer sub-models with 33 and 25 percent parameter removal, respectively. Besides GFM, we consider the following distillation approaches:

- Soft distillation. q is the teacher's output after softmax. p is the output of the compressed sub-model and y is the true label. The soft distillation objective is:

$$\mathcal{L}_{\mathrm{CE}}(p, y) + \alpha \mathcal{L}_{KL}(p, q). \tag{16}$$

- Soft distillation without labels. The loss function is:

$$\mathcal{L}_{KL}(p, q). \tag{17}$$

- Hard distillation. Let $y_t = \arg\max(q)$ be the hard decision of the teacher. The loss function of hard-label distillation is:

$$\mathcal{L}_{\mathrm{CE}}(p, y) + \alpha \mathcal{L}_{\mathrm{CE}}(p, y_t). \tag{18}$$

- Hard distillation without labels. The loss function is:

$$\mathcal{L}_{\mathrm{CE}}(p, y_t). \tag{19}$$

- GFM with labels. We add label information into GFM, i.e.,

$$\mathcal{L}_{\mathrm{CE}}(p, y) + \alpha \mathcal{L}_{\mathrm{MSE}}\left(f_c^L, f_o^L\right). \tag{20}$$

We set the α as 1.0 and illustrate the results in Table 6. We can conclude that under the few-shot settings, including label information when training the compressed model can easily lead to overfitting, while our GFM still obtains the best results when fine-tuning the compressed model.

The proxy dataset size. We further research on the influence of the number of training samples in the proxy dataset $\mathcal{D}$. Let us take Swin-B with 33% parameters removed as an example. We set the size of dataset $\mathcal{D}$ to 1K, 2K, 5K, 10K, 100K and the whole training dataset respectively. We applied AAFM and GFM sequentially on different proxy datasets. In particular, we trained 10 and 100 epochs when $\mathcal{D}$ contains the full training samples and 100K samples, respectively, and trained 1000 epochs in other cases.

Distillation		Top 1 Acc. (%)	Perplexity
Soft	w/ label	82.34	24.81
	w/o label	82.38	20.53
Hard	w/ label	81.05	323.95
	w/o label	82.08	1.2×10^5
GFM	w/ label	78.81	20.10
	w/o label	**82.68**	**20.05**

Table 6: Results on the ImageNet-1K and WikiText-103 validation datasets with different distillation strategies.

Sizes	1K	2K	5K	10K	100K	1.28M
AAFM	81.21	81.15	81.20	**81.31**	81.30	81.22
GFM	82.38	82.68	82.75	82.88	82.97	**82.99**

Table 7: Top-1 accuracy (%) on the ImageNet-1K validation set with different number of training samples in the proxy dataset $\mathcal{D}$.

The results are showed in Table 7. The 1.28M in the table refers to the entire dataset. We can conclude the final accuracy increases as the sample size grows, but the benefit is very limited. When applying the full training dataset, the AAFM and final accuracy is only 0.07% (81.22% vs. 81.15%) and 0.31% (82.99% vs. 82.68%) higher than using 2K samples each, respectively. This indicates that few unlabeled samples are sufficient for our feature-mimicking algorithms. Therefore, we sample 2K images as the proxy dataset for faster speed and better simplicity.

Discussions and Conclusions

In this paper, we presented a novel framework for low-rank approximation of transformers. We built our framework after revealing that the features are low-rank but model weights are not, which worked well in both CV and MLP. Extensive experiments confirmed the efficacy of our framework. We can quickly reduce the model size with small drop in model accuracy. In addition, our approach requires only a small number of unlabeled samples and effectively preserves the original model's generalization capability.

We discover that because low-rank decomposition divides a linear layer into two layers, it does not improve throughput significantly. Therefore, applying low-rank decomposition in a reasonable way to speed up inference is an intriguing future direction. Besides this, we randomly select samples to form the proxy dataset, so we will continue to explore the effects of data distribution in the proxy dataset. Furthermore, our method can theoretically be applied to various deep learning models, such as CNNs, so we will continue to extend our method to these models in the future.

Acknowledgments

This research was partly supported by the National Natural Science Foundation of China under Grant 62276123 and Grant 61921006.

References

Baevski, A.; and Auli, M. 2018. Adaptive Input Representations for Neural Language Modeling. In *International Conference on Learning Representations (ICLR)*.

Bai, H.; Wu, J.; King, I.; and Lyu, M. 2020. Few shot network compression via cross distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 3203–3210.

Cai, Z.; and Vasconcelos, N. 2018. Cascade R-cnn: Delving into high quality object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6154–6162.

Chen, P.; Yu, H.-F.; Dhillon, I.; and Hsieh, C.-J. 2021. Drone: Data-aware low-rank compression for large nlp models. In *Advances in Neural Information Processing Systems*, volume 34, 29321–29334.

Cubuk, E. D.; Zoph, B.; Shlens, J.; and Le, Q. V. 2020. RandAugment: Practical automated data augmentation with a reduced search space. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 702–703.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A large-scale hierarchical image database. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 248–255.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations (ICLR)*.

Golub, G. H.; and Van Loan, C. F. 2013. *Matrix computations*. Johns Hopkins University Press.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531.

Hsu, Y.-C.; Hua, T.; Chang, S.; Lou, Q.; Shen, Y.; and Jin, H. 2022. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representations (ICLR)*.

Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; and Weinberger, K. Q. 2016. Deep Networks with Stochastic Depth. In *The European Conference on Computer Vision (ECCV)*, volume 9908 of *LNCS*, 646–661. Springer.

Kenton, J. D. M.-W. C.; and Toutanova, L. K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*, 4171–4186.

Li, T.; Li, J.; Liu, Z.; and Zhang, C. 2020. Few sample knowledge distillation for efficient network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 14639–14647.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft COCO: Common Objects in Context. In *The European Conference on Computer Vision (ECCV)*, volume 8693 of *LNCS*, 740–755. Springer.

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 10012–10022.

Loshchilov, I.; and Hutter, F. 2017. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*.

Loshchilov, I.; and Hutter, F. 2018. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations (ICLR)*.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2017. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations (ICLR)*.

Noach, M. B.; and Goldberg, Y. 2020. Compressing pre-trained language models by matrix decomposition. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, 884–889.

Ott, M.; Auli, M.; Grangier, D.; and Ranzato, M. A. 2018. Analyzing uncertainty in neural machine translation. In *International Conference on Machine Learning (ICML)*, 3956–3965.

Ott, M.; Edunov, S.; Baevski, A.; Fan, A.; Gross, S.; Ng, N.; Grangier, D.; and Auli, M. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Romero, A.; Ballas, N.; Kahou, S. E.; Chassang, A.; Gatta, C.; and Bengio, Y. 2014. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*.

Touvron, H.; Cord, M.; Douze, M.; Massa, F.; Sablayrolles, A.; and Jegou, H. 2021. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning (ICML)*, 10347–10357.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30, 5998–6008.

Wang, G.-H.; Ge, Y.; and Wu, J. 2021. Distilling knowledge by mimicking features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Wang, H.; Liu, J.; Ma, X.; Yong, Y.; Chai, Z.; and Wu, J. 2022. Compressing Models With Few Samples: Mimicking Then Replacing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 701–710.

Wu, J. 2020. *Essentials of Pattern Recognition: An Accessible Approach*. Cambridge University Press.

Yu, H.; Wang, H.; and Wu, J. 2021. Mixup without hesitation. In *International Conference on Image and Graphics*, 143–154. Springer.

Yun, S.; Han, D.; Oh, S. J.; Chun, S.; Choe, J.; and Yoo, Y. 2019. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 6023–6032.

Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2018. Mixup: Beyond Empirical Risk Minimization. In *International Conference on Learning Representations (ICLR)*.

Zhong, Z.; Zheng, L.; Kang, G.; Li, S.; and Yang, Y. 2020. Random Erasing Data Augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 13001–13008.