DynaMixer: A Vision MLP Architecture with Dynamic Mixing

Ziyu Wang
Data Platform, Tencent

Wenhao Jiang[*]
Data Platform, Tencent

Yiming Zhu
Tsinghua university

Li Yuan
Peking University

Yibing Song
Tencent AI Lab

Wei Liu[†]
Data Platform, Tencent

February 17, 2022

Abstract

Recently, MLP-like vision models have achieved promising performances on mainstream visual recognition tasks. In contrast with vision transformers and CNNs, the success of MLP-like models shows that simple information fusion operations among tokens and channels can yield a good representation power for deep recognition models. However, existing MLP-like models fuse tokens through static fusion operations, lacking adaptability to the contents of the tokens to be mixed. Thus, customary information fusion procedures are not effective enough. To this end, this paper presents an efficient MLP-like network architecture, dubbed DynaMixer, resorting to dynamic information fusion. Critically, we propose a procedure, on which the DynaMixer model relies, to dynamically generate mixing matrices by leveraging the contents of all the tokens to be mixed. To reduce the time complexity and improve the robustness, a dimensionality reduction technique and a multi-segment fusion mechanism are adopted. Our proposed DynaMixer model (97M parameters) achieves 84.3% top-1 accuracy on the ImageNet-1K dataset without extra training data, performing favorably against the state-of-the-art vision MLP models. When the number of parameters is reduced to 26M, it still achieves 82.7% top-1 accuracy, surpassing the existing MLP-like models with a similar capacity. The implementation of DynaMixer will be made available to the public.

[*]corresponding author
[†]corresponding author

1 Introduction

Convolutional neural networks (CNNs) have been the dominating solution for a wide range of computer vision tasks for a long time, *e.g.,* visual recognition and segmentation. CNNs dedicate training the whole network end-to-end to remove hand-crafted visual features and inductive biases. CNNs heavily rely on convolution operators and pooling operators, introducing locality and spatial invariances into CNNs. Recently, Transformers [1], which have achieved overwhelming success in natural language processing, have also been introduced into the compute vision field. The Vision Transformers (ViTs) [2] have achieved state-of-the-art performances on visual recognition tasks. In ViT, images are first divided into a sequence of non-overlapping patches, which can be seen as word tokens in the natural language processing field. Those patches are then fed into a stack of blocks based on self-attention to obtain the contextualized output tokens. ViT removes the convolution and pooling operators from CNNs, and employs a self-attention mechanism to model the relationships among image patches. Inspired by ViTs, many efforts have been made to designing simpler models in the setting with a large amount of data available.

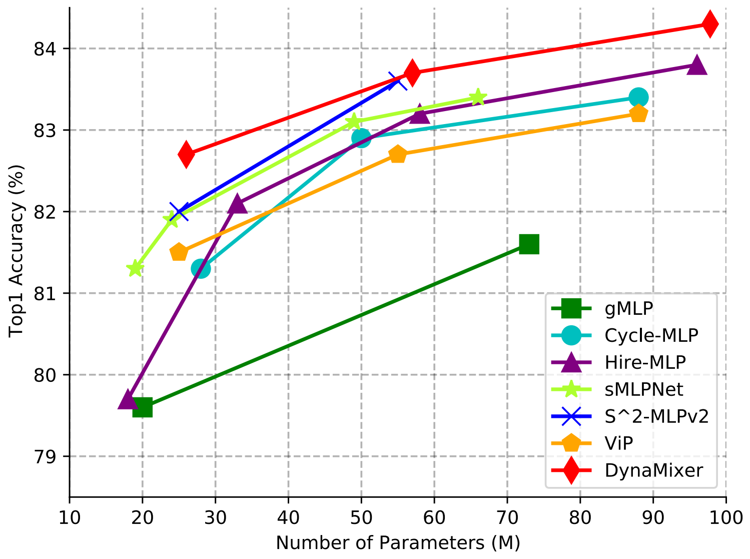

Figure 1: **ImageNet accuracy v.s. model capacity.** All models are trained on ImageNet-1K without extra data. DynaMixer outperforms all the existing MLP-like models, such as MLP-Mixer [3], Hire-MLP [4], S^2-MLP-v2 [5], ViP [6], and Cycle-MLP [7].

More recently, a pure multilayer perceptron (MLP) architecture [3], called MLP-Mixer, was proposed to reduce the inductive biases further and showed competitive performance with CNN-based models and vision transformers. MLP-Mixer is much simpler than transformer-based models because it utilizes MLP as a building block, removing the need of invoking self-attention. In MLP-Mixer, each layer mainly relies on two steps to perform information interaction: token mixing step and channel mixing step, both based on MLP blocks. The token mixing step performs fusion among the tokens, while the channel mixing step performs fusion on the channel dimension. Inspired by this, a number of MLP-like models [7, 6, 5, 8] have emerged for further improvements. However, these MLP-like methods rely on fixed static mixing matrices for patch communications, which may restrict the adaptability to the contents to be fused.

To solve the above limitation, we propose a vision MLP architecture with dynamic mixing, dubbed DynaMixer, which can generate mixing matrices dynamically for each set of tokens to be mixed by considering their contents. Note that mixing all the image tokens consumes a significant time cost. For a computational speedup, we mix tokens in a row-wise and column-wise way. During each mixing process, we reduce the feature dimensionality to generate the mixing matrices. We empirically find that we can reduce feature dimensionality significantly without undermining the performance much. *The feature dimensionality can even be reduced to 1 in our experiments.* Furthermore, we divide the feature channels into multiple segments and perform token mixing separately. This improves the mixing robustness as well as efficiency. As a result, our method achieves state-of-the-art performance among existing MLP-like vision models on the ImageNet-1K dataset [9], as summarized in Fig 1. Without any extra data, our DynaMixer accomplishes 84.3% top-1 accuracy with 97M parameters. When the model parameters are reduced to 26M, DynaMixer can still reach 82.7% top-1 accuracy, far exceeding other MLP-like models with a similar number of parameters.

2 Related Work

The deep neural networks for vision related tasks evolve from CNNs to ViTs and MLPs. In this section, we review these models and illustrate the relationships between our DynaMixer and these models.

2.1 Convolutional Neural Networks

The CNNs have drawn huge attention since AlexNet [10] improved image classification by a large margin. Convolution operations, nonlinear activation, and pooling operations have become prevalent for CNN architecture designs. Based on the previous works on CNN, inceptions [11, 12, 13] were proposed by employing multiple parallel branches. A breakthrough was made by ResNet [14], in which the network depth was significantly increased. To ease training, skip connections with identity mapping were introduced into ResNets. They achieved state-of-the-art visual recognition performances at that time and inspired further research, including ResNeXt [15] and DenseNet [16]. Afterward, attention mechanisms were gradually explored to benefit CNNs. Examples include SENet [17], non-local neural network [18], and local relationship network [19]. CNNs have been widely applied to computer vision related tasks, such as visual recognition, image/video generation, object detection, semantic segmentation, and so on.

2.2 Vision Transformers

Transformer [1] was firstly proposed in the machine translation area. It was then adopted for computer vision applications in ViT [2]. Input images in ViT are divided into a sequence of non-overlapping patches, which are regarded as tokens and are fed into transformers. Unlike CNNs aggregating information within local windows, ViTs model global information interactions among visual tokens via self-attention. Inspired by ViTs, many variants are proposed to remove the drawbacks of original ViTs. In DeepViT [20] and CaiT [21], the problem that the performance of ViTs saturates fast when scaled to be deeper was addressed. In [22, 23, 24, 25, 26], multi-granularity was introduced into ViTs to improve the generalization ability. A complete survey on vision transformers can be found in [27, 28]. All transformer-based models benefit from the self-attention mechanism for its flexibility and adaptiveness. Our model does not invoke the self-attention mechanism, while it is not lack of adaptability. Differences will be discussed in the following section.

2.3 Vision MLPs

Recently, MLP-based vision models [3, 29] were proposed to reduce the inductive bias further and showed competitive performances with CNN-based models and ViTs. Existing MLP-like models share a similar macro framework but have different micro block designs. MLP-like models usually divide one input image into patches, like in vision transformers, and then perform two main steps: token mixing step and channel mixing step. The specific details of these two steps, especially token-mixing steps, are different among the existing methods. ViP [6] was proposed to perform token mixing along the height and width dimensions with simple linear projections to encode spatial information, which is different from other MLP-like models that fuse token information among all tokens in one step. S^2-MLP [5] replaced the token mixing step with a spatial-shift step to enable information interaction among tokens. CycleMLP [7] samples local points along the channel dimension in a cyclical style, while AS-MLP [30] samples locations in a cross. Hire-MLP [4] mixes tokens within a local region and across local regions.

All these MLP-like methods rely on static mixing matrices for patch communications, which may limit the performance for lacking adaptiveness since it is natural that a different set of tokens should be mixed differently. Thus, we propose to generate the mixing matrices dynamically by considering the contents of tokens to be mixed.

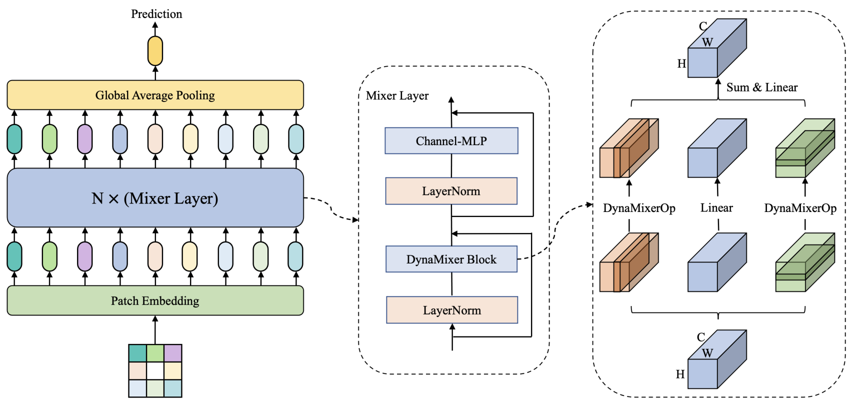

Figure 2: **The basic architecture of our proposed DynaMixer.** DynaMixer consists of a patch embedding layer, several mixer layers, a global average pooling layer, and a classifier head. The patch embedding layer transforms input non-overlapping patches into corresponding input tokens, which are fed into a sequence of mixer layers to generate the output tokens. All output tokens are averaged in the average pooling layer, and the final prediction is generated with a classifier head. The mixer layer (middle part) contains two layer-normalization layers, a DynaMixer block and a channel-MLP block. The DynaMixer block (right part) performs row mixing and column mixing via the DynaMixer operations (depicted in Fig. 3), and a simple channel-mixing via linear projection. The mixing results are summed and outputted with a linear transformation.

A recent work, Synthesizer [31], pointed out that either the attention matrix in ViTs or the mixing matrix in MLP can be seen as generated with a specially designed function. It is worth noting that although the Synthesizer (Dense) [31] adopts to generate the mixing matrix dynamically, the mixing weights for a specific position are determined only by the contents of the token at the same position. However, the weights for it in our model are determined by all the tokens. We will show how to design such a generating function and how to reduce the computational complexity in the following section.

3 The Architecture of DynaMixer

In this section, we present our model designs. The overall framework of our network is illustrated in Fig. 2. Like in other MLP-based models, the input image is divided into non-overlapping patches. All patches are projected onto the embedding space with a shared matrix, which are then fed into a sequence of mixer layers to generate the output tokens. All output tokens are averaged in the average pooling layer, and then sent into the classifier head to yield the final prediction. Except for the layer-normalization layers and skip connections, the mixer layer contains a DynaMixer block and a channel-MLP block, which are responsible for fusing token information and channel information, respectively. The channel-MLP block is just a feed-forward layer as in Transformer [1]. The DynaMixer block performs row mixing and column mixing via DynaMixer operations, and channel-mixing via simple linear projection, respectively. The mixing results are aggregated and outputted via a linear transformation. We first illustrate our DynaMixer operation, and then elaborate on the DynaMixer block. Finally, we analyze the differences from other MLP-like models.

3.1 The DynaMixer Operation

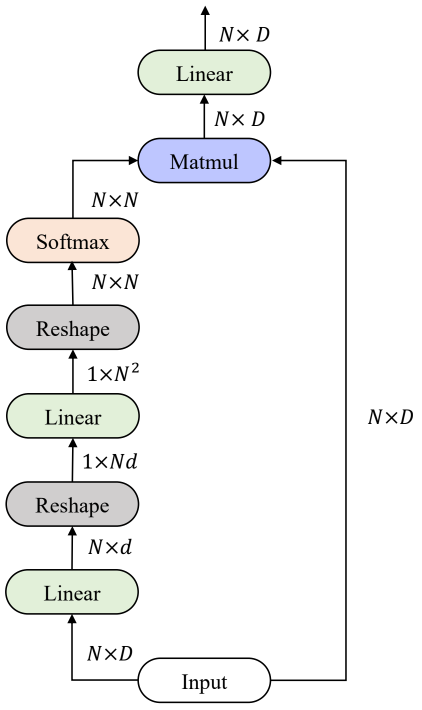

Figure 3: The procedure of our proposed DynaMixer operation for one segment.

Dynamic mixing matrix generation. The principle of our design is to generate a dynamic mixing matrix P given a set of input tokens $X \in R^{N \times D}$ by considering their contents, where N is the number of tokens, and D is the feature dimensionality. Once we obtain P, we can mix the tokens by $Y = PX$ to obtain the output tokens Y. A simple way to get P is utilizing a linear function of all input token features to estimate it. Thus, we can simply flatten X into a vector and generate the mixing matrix as

$$P_{i\cdot} = \texttt{softmax}\left(\texttt{flat}(X)^T W^{(i)}\right), \tag{1}$$

where $\texttt{flat}(X) \in R^{ND\times 1}$ is a vector by flattening X, $\texttt{softmax}(\cdot)$ is the softmax operator performing on a row vector, $W^{(i)} \in R^{ND\times N}$, and P is the mixing matrix. $P_{i\cdot}$ is the i-th row of P, and it contains the mixing weights for the i-th output token. However, the number of parameters of the above process is too large since ND is usually too big. Thus, we can perform dimensionality reduction first to reduce the number of parameters. Therefore, we have the following steps to generate the mixing matrix and perform mixing on tokens:

$$\hat{X} = XW_d, \tag{2}$$

$$P_{i\cdot} = \texttt{softmax}\left(\texttt{flat}(\hat{X})W^{(i)}\right), \tag{3}$$

$$Y = PX, \tag{4}$$

where $\hat{X} \in R^{N\times d}$, and $d \ll D$ is a quite small number, say 1 or 2. The whole procedure for generation and mixing is described pictorially in Fig 3, which can be implemented easily with PyTorch [32] or TensorFlow [33].

Multi-segment fusion mechanism. To improve the robustness of our model, we divide features into S segments, perform the mixing operation separately, and combine the mixed results to obtain the final results:

$$\hat{X}^{(s)} = XW_d^{(s)}, \tag{5}$$

$$P_{i\cdot}^{(s)} = \texttt{softmax}\left(\texttt{flat}(\hat{X}^{(s)})W^{(s,i)}\right), \tag{6}$$

$$Y = [P^{(0)}X^{(0)}, \cdots, P^{(S-1)}X^{(S-1)}]W_o, \tag{7}$$

where $\hat{X}^{(s)} \in R^{N\times d}, W^{(s,i)} \in R^{Nd\times N}$, $P_{i\cdot}^{(s)}$ is the i-th row of the token mixing matrix for the s-th segment, $X^{(s)}$ is the tokens in the s-th segment, $[\cdot,\cdot]$ is the concatenation operation, and $W_o \in R^{D\times D}$ is a feature fusion matrix for output. Note that, different dimensionality reduction matrices, *i.e.*, $W_d^{(s)}$, are used for different segments, which are beneficial for the expressive power of the model. Moreover, the mixing matrix for the s-th segment is also effected by the contents of other segments. Thus, the mixing behaviours for different segments are not independent.

Weights sharing among segments. To reduce the number of parameters, the matrices $W^{(s,i)}$ are shared among all segments. Thus, the number of parameters of the DynaMixer operation is $S \times D \times d + N^3 \times d + D^2$. In practice, we found that d can be quite small, *e.g.*, $d = 1$ or $d = 2$.

We use the following expression to denote the DynaMixer operation in the rest of this paper:

$$Y = \texttt{DynaMixerOp}(X). \tag{8}$$

3.2 The DynaMixer Block

The number of parameters of the above operation mainly depends on N, the number of tokens to be mixed, which is $H \times W$, where H and W are the numbers of patches in the height and width directions, respectively. Thus, to reduce the computational complexity, we adopt a strategy similar to ViP [6], which mixes tokens in a row or column at one step. As depicted in the right part of Fig 2, the DynaMixer block consists of three components, which are row mixing, column mixing, and channel mixing. In row mixing, the tokens that belong to the same row are mixed via DynaMixer operations. And the parameters of DynaMixer operations are shared among all rows. Similarly, the tokens that belong to the same column are mixed via DynaMixer operation in column mixing, in which the parameters of DynaMixer operations are also shared. The channel mixing is simply a linear transformation on features. We denote the outputs of the three components as Y_h, Y_w and Y_c, and the output of the DynaMixer block is

$$Y_{out} = (Y_h + Y_w + Y_c)W_o'. \tag{9}$$

The procedure for the DynaMixer block is summarized in Algorithm 1, in which we denote the input as $X \in R^{H \times W \times D}$ for convenience.

In Algorithm 1, the three mixing results are just summed to obtain the final result, as described in Eq. (9). To further improve the performance, importances of the three components can be estimated by a method similar to [34, 6]. In the following experiments, the weighted DynaMixer block is used by default.

3.3 Discussion

DynaMixer v.s. Synthesizer (Dense). The weights for the i-th output token generated by Synthesizer (Dense) are only determined by the contents of the i-th input token. Thus, *it cannot model the effects from the other tokens, leading to inaccurate weight estimation.* However, our model considers all the tokens to generate more accurate mixing weights. Moreover, our model divides the channel into non-overlapping segments and performs separate mixing operations on them. Such a design can improve the robustness and expressive power of our model.

DynaMixer v.s. ViP. Except that the mixing matrices in our model are generated dynamically, the mixing behaviors are also quite different. Suppose W tokens in the h-th row are to be mixed. Each token is divided into S segments along channel dimension, and the dimensionality of channel is denoted as D. In ViP, *$W \times S$ vectors of dimensionality $\frac{D}{S}$ will be mixed by multiplying one*

Algorithm 1 Pseudo-code for DynaMixer Block (PyTorch-like)

```
###### initializaiton #######
proj_c = nn.Linear(D, D)
proj_o = nn.Linear(D, D)

###### code in forward ######
def dyna_mixer_block(self, X):
  H, W, D = X.shape

  # row mixing
  for h = 1:H
    Y_h[h,:,:] = DynaMixerOp_h(X[h,:,:])

  # column mixing
  for w = 1:W
    Y_w[:,w,:] = DynaMixerOp_w(X[:,w,:])

  # channel mixing
  Y_c = proj_c(X)
  Y_out = Y_h + Y_w + Y_c

  return proj_o(Y_out)
```

fixed matrix. However, in DynaMixer, the mixing operation occurs S times. One time for one segment. And *each mixing operation is performed on W vectors of dimensionality $\frac{D}{S}$ with different mixing matrices.*

DynaMixer v.s. self-attention. The mixing matrix (attention map) in self-attention models the *pair-wise relationships between tokens*. Thus, it is a quadratic function of token features. However, in our method, the mixing matrix is estimated by computing the *relationships between tokens and a trainable matrix* ($W^{s,i}$ in Eq. (6)).

4 Experimental Results

In this section, we present the experimental results and analysis. First, we will give the configurations of our DynaMixer used in the experiments, and then the experimental settings and results on the ImageNet-1K dataset are provided. At last, the ablation studies are presented to provide a deep understanding of the designs in our model.

4.1 Configurations of DynaMixer

The configurations of our model used in the experiments are summarized in Table 1. There are three versions of DynaMixer, denoted as "DynaMixer-S", "DynaMixer-M", and "DynaMixer-L", according to the model sizes. In all our experiments, the input image size is 224×224, and the input patch size is 7×7. All our models have two stages, and each starts with a patch embedding layer. The patch size for the second stage is 2×2. Thus, the corresponding patch embedding layer can be seen as a downsampling layer.

Specification	DynaMixer-S	DynaMixer-M	DynaMixer-L
Patch size	7×7	7×7	7×7
Hidden size	192	256	256
#Tokens	32×32	32×32	32×32
#Mixer Layers	4	7	9
S	8	8	8
Patch size	2×2	2×2	2×2
Hidden size	384	512	512
#Tokens	16×16	16×16	16×16
#Mixer Layers	14	17	27
S	16	16	16
#Layers	18	24	36
MLP Ratio	3	3	3
Stoch. Dep.	0.1	0.1	0.3
d	2	2	8
#Parameters	26M	57M	97M

Table 1: **The configurations of DynaMixers used.** We design three DynaMixers (DynaMixer-S, DynaMixer-M, and DynaMixer-L) of different sizes (Small, Medium, and Large) for the experiments, according to the number of parameters.

4.2 Experimental Configurations

We train our proposed DynaMixer on the public image classification benchmark ImageNet-1K dataset [9], which covers $1K$ categories of natural images and contains $1.2M$ training images and $50K$ validation images. Because the test set for this benchmark is unlabeled, we follow the common practice by evaluating the performance on the validation set. The code implementation is based on PyTorch [32] and the TIMM [35] toolbox.

We train our model with one machine with 8 NVIDIA A100 GPUs with data parallelism. For model optimization, we adopt the AdamW optimizer [36]. The learning rates for DynaMixer-S, DynaMixer-M, and DynaMixer-L are 0.002, 0.002, and 0.001, respectively, and the corresponding batch sizes on one GPU are 256, 128, and 64, respectively. We set the weight decay rate to 0.05 and set the warmup learning rate to 10^{-6} to follow the settings in previous work [37, 38]. We use automatic mixed-precision of the PyTorch version for training acceleration. Stochastic Depth [39] is used, and the drop rate varies with the size of the model.

The model is trained for 300 epochs. For data augmentation methods, we use CutOut [40], RandAug [41], MixUp [42], and CutMix [43]. Detailed settings about S, d, and the path drop rates for different versions of DynaMixer can be found in Table 1.

4.3 Results on ImageNet-1K

In this subsection, we compare our model with MLP-like models, CNN-based models, and transformer-based models to show the advantages of our dynamic mixing mechanism.

Comparisons with MLP-like Models. First, we compare our DynaMixer with MLP-like models. The top-1 accuracies of different MLP-like models on ImageNet-1K [9] without external data are shown in Table 2. The “Param” and “FLOPs” columns denote the number of parameters and the number of floating point operations, respectively. The results are extracted from the corresponding papers, and the FLOPs are computed with the code of CycleMLP[1]. Firstly, it is obvious that our model consistently outperforms the other models of a similar number of parameters. In addition, we can see that our DynaMixer-S model with 26M parameters achieves top-1 accuracy of 82.7%, which has already surpassed most of the existing MLP-like models of all sizes and is even better than gMLP-B [44] with 73M parameters. Increasing the number of parameters to 57M, our DynamMixer-M obtains accuracy 83.7%, which is superior to all MLP-like models. Further expanding the network to 97M parameters, DynaMixer-L can achieve top-1 accuracy of 84.3%, which is a new state-of-the-art among the MLP-like models.

By comparing with ViP [6], which is the most similar model to ours, we find that the number of parameters and FLOPs of our model are only slightly larger than those of ViP. However, the improvements of performance are not caused by the increment on model size. The model sizes of our model and ViP are similar. For example, the number of parameters and FLOPs of DynaMixer-M are 57M and 17G. And those of ViP-Mediam/7 are 55M and 16.3G. Thus, the model size of ours is about 5% larger than ViP. But the performance is improved greatly from 82.7% to 83.7%. Therefore, the model structure should be attributed to such improvements. Specifically, the designs of dynamic mixing matrices and multi-segment fusion mechanism do help improve the performance. The mixing matrices generated dynamically could capture the contents of tokens to be mixed. Thus, different sets of tokens will be mixed with different mixing matrices. Therefore, token information could be fused more effectively. At last, the multi-segment fusion mechanism improves the robustness of the whole model. We also provide detailed ablation studies of these two designs in the following subsection for gaining a deeper understanding of our method.

[1]https://github.com/ShoufaChen/CycleMLP

Model	Date	Param	FLOPs	Top-1 (%)
Mixer-B/16 [3]	MAY, 2021	59M	12.7G	76.4
Mixer-B/16† [3]		59M	12.7G	77.3
ResMLP-S12 [29]		15M	3.0G	76.6
ResMLP-S24 [29]	MAY, 2021	30M	6.0G	79.4
ResMLP-B24 [29]		116M	23.0G	81.0
gMLP-Ti [44]		6M	1.4G	72.3
gMLP-S [44]	MAY, 2021	20M	4.5G	79.6
gMLP-B [44]		73M	15.8G	81.6
ViP-Small/14 [6]		30M	6.9G	80.7
ViP-Small/7 [6]	JUN, 2021	25M	6.9G	81.5
ViP-Medium/7 [6]		55M	16.3G	82.7
ViP-Large/7 [6]		88M	24.4G	83.2
AS-MLP-T [30]		28M	4.4G	81.3
AS-MLP-S [30]	JUL, 2021	50M	8.5G	83.1
AS-MLP-B [30]		88M	15.2G	83.3
S^2-MLPv2-Small/7 [8]	AUG, 2021	25M	6.9G	82.0
S^2-MLPv2-Medium/7 [8]		55M	16.3G	83.6
RaftMLP-36 [45]	AUG, 2021	44M	9.0G	76.9
RaftMLP-12 [45]		58M	12.0G	78.0
sMLPNet-T [46]	SEP, 2021	24M	5.0G	81.9
sMLPNet-S [46]		49M	10.3G	83.1
sMLPNet-B [46]		66M	14.0G	83.3
ConvMLP-S [47]		9M	2.4G	76.8
ConvMLP-M [47]	SEP, 2021	17M	3.9G	79.0
ConvMLP-L [47]		43M	9.9G	80.2
CycleMLP-T [7]		28M	4.4G	81.3
CycleMLP-S [7]	NOV, 2021	50M	8.5G	82.9
CycleMLP-B [7]		88M	15.2G	83.4
Hire-MLP-Ti [4]		18M	2.1G	79.7
Hire-MLP-S [4]	NOV, 2021	33M	4.2G	82.1
Hire-MLP-B [4]		58M	8.1G	83.2
Hire-MLP-L [4]		96M	13.4G	83.8
DynaMixer-S		26M	7.3G	82.7
DynaMixer-M		57M	17.0G	83.7
DynaMixer-L		97M	27.4G	84.3

Table 2: Image classification results of our DynamMixer and other MLP-like models on the ImageNet-1K benchmark without extra data. "Top-1" denotes Top-1 accuracy. "Param" and FLOPs denote the number of parameters and the number of floating point operations, respectively. Model with † was re-implemented by [44]. The date column means the corresponding initial release date on arXiv. The FLOPs are computed with the code of CycleMLP [7].

Comparison with SOTA Classification Models. We also compare our model with well-known state-of-the-art models, which include CNN-based models, transformer-based models, and hybrid models. The results are shown in Table 3. We can see that our proposed models still achieve the best performance among models with a similar number of parameters. Specifically, our models achieve better performance than Swin Transformer [24], which is the state-of-the-art transformer-based model. Our DynaMixer-S achieves accuracy 82.7% with slightly fewer parameters, while Swin-T achieved 81.3%. For medium-sized and large-sized models, our models are still better than Swin-S and Swin-B. Our model is also better than CrossFormer [25], which employs a multi-scale strategy. We believe that by introducing multi-granularity, the performance of our method could be further improved. From the above comparisons, we can see that DynaMixer is a promising architecture for visual recognition tasks.

Model	Family	Scale	Param	FLOPs	Top-1 (%)
ResNet18 [14]	CNN	224^2	12M	1.8G	69.8
EffNet-B3 [48]	CNN	300^2	12M	1.8G	81.6
PVT-T [49]	Trans	224^2	13M	1.9G	75.1
GFNet-H-Ti [50]	FFT	224^2	15M	2.0G	80.1
ResNet50 [14]	CNN	224^2	26M	4.1G	78.5
RegNetY-4G [51]	CNN	224^2	21M	4.0G	80.0
DeiT-S [37]	Trans	224^2	22M	4.6G	79.8
BoT-S1-50 [52]	Hybrid	224^2	21M	4.3G	79.1
PVT-S [49]	Trans	224^2	25M	3.8G	79.8
T2T-14 [53]	Trans	224^2	22M	4.8G	81.5
Swin-T [24]	Trans	224^2	29M	4.5G	81.3
GFNet-H-S [50]	FFT	224^2	32M	4.5G	81.5
CrossFormer-T [25]	Trans	224^2	27M	2.9G	81.5
CrossFormer-S [25]	Trans	224^2	30M	4.9G	82.5
DynaMixer-S	MLP	224^2	26M	7.3G	**82.7**
ResNet101 [14]	CNN	224^2	45M	7.9G	79.8
RegNetY-8G [51]	CNN	224^2	39M	8.0G	81.7
BoT-S1-59 [52]	Hybrid	224^2	34M	7.3G	81.7
PVT-M [49]	Trans	224^2	44M	6.7G	81.2
GFNet-H-B [50]	FFT	224^2	54M	8.4G	82.9
Swin-S [24]	Trans	224^2	50M	8.7G	83.0
PVT-L [49]	Trans	224^2	61M	9.8G	81.7
CrossFormer-B [25]	Trans	224^2	52M	9.2G	83.4
T2T-24 [53]	Trans	224^2	64M	13.8G	82.1
DynaMixer-M	MLP	224^2	57M	17.0G	**83.7**
CrossFormer-L [25]	Trans	224^2	92M	16.1G	84.0
Swin-B [24]	Trans	224^2	88M	15.4G	83.3
DeiT-B [37]	Trans	224^2	86M	17.5G	81.8
DeiT-B [37]	Trans	384^2	86M	55.4G	83.1
DynaMixer-L	MLP	224^2	97M	27.4G	**84.3**

Table 3: Top-1 accuracy comparisons with CNN-based models and transformer based models on the ImageNet-1K image classification benchmark. Note that all models are trained without external data. The "Scale" denotes the input size for training and testing stages. The "Param" and "FLOPs" columns denote the number of parameters and the number of floating point operations, respectively. The values in the column Family, *e.g.,* "CNN", "Trans", "Hybrid", and "FFT", mean CNN-based, transformer-based, CNN-and-transformer-based, and fast Fourier transform based models.

Limitations. The numbers of parameters and FLOPs of DynaMixer are slightly larger than those of other models, as the mixing matrix is generated dynamically. In practice, such an increment over model size will affect the training time. For example, the training speeds of DynaMixer-S, DynaMixer-M, and DynaMixer-L are 20%, 14%, 25% lower than those of ViP-Small/7, ViP-Mediam/7 and ViP-Large/7, respectively. Moreover, the input image size for our model should be fixed, which restricts its applications on some downstream tasks, *e.g.*, object detection, and segmentation. Removing such restrictions by adopting techniques like sliding window is our future work.

4.4 Ablation Study

In this subsection, we present ablation study results with DynaMixer-S to provide a deeper understanding of our methods.

Ablation on the number of segments S. To improve the robustness and generalization ability of our network, we introduce a multi-segment fusion mechanism in the DynaMixer operation. To show the effects of the number of segments, the performances with different values of segment dimensionalities, which is $\frac{D}{S}$, are reported in Table 4. Considering the computational complexity, we only test 4 values of segment dimensionalities, which are D, 96, 48, and 24. We can see that even without multiple segments, our model can still obtain satisfying accuracy (82.2%). As S increases ($\frac{D}{S}$ decreases), the performance becomes better. Moreover, our network is not sensitive to segment dimensionalities in a quite large range.

Model	D/S	# parameters	Top-1 (%)
DynaMixer-S	D	26M	82.2
DynaMixer-S	96	26M	82.4
DynaMixer-S	48	26M	82.5
DynaMixer-S	24	26M	82.7

Table 4: The top-1 accuracies with different segment dimensionalities.

Ablation on weight generating methods. We study the effects of different methods to generate mixing matrices in this paragraph. The methods include "Synthesizer (Random)" and "Synthesizer (Dense)", which are described in [31]. Once the mixing matrices are generated, they are applied to mixing tokens by following the procedure of DynaMixer, which means that tokens are mixed row by row and column by column with the multi-segment fusion mechanism. The mixing matrices generated by "Synthesizer (Random)" are actually trainable parameters. Thus it is similar to ViP [6], except that the information fusion among segments of channel is different. The mixing matrix generated by "Synthesizer (Dense)" does not consider the effects of other tokens. Thus, the mixing weight vector for a specific output token in "Synthesizer (Dense)" is generated merely with the information from the token at the same position. The performances of these methods are listed in Table 5. We can see that "Synthesize (Random)" achieved an accuracy of 81.5%, which is close to ViP's. It is natural since their mixing matrices are generated in the same way. The accuracy achieved by "Synthesizer (Dense)", which is 20% larger than DynaMixer-S, is only 81.4%, which demonstrates that leveraging the contents of all tokens is necessary, and the technique of reducing the computational complexity adopted in our model is effective.

Model	generating methods	# parameters	Top-1 (%)
DynaMixer-S	Synthesize(Random)	25M	81.5
DynaMixer-S	Synthesizer(Dense)	32M	81.4
DynaMixer-S	DynaMixer-S	26M	82.7

Table 5: The effects of different mixing matrix generation methods.

Ablation on the reduced dimensionality d. In our model, dimensionality is reduced to d by a liner projection to decrease the number of parameters. Ideally, d should be large to provide enough information. However, we found that d *can be extremely low in practice*. The top-1 accuracies with different values of d are shown in Table 6. We can see that our model with $d = 1$ can still achieve an accuracy of 82.4%, which is an absolute improvement of 0.9% compared with ViP. Thus, even dimensionality is reduced to 1, the reduced token feature could provide information of all tokens to some extent. The performances with values of d set to 2, 4, and 8 are similar. Thus, d is set to 2 for DynaMixer-S in our experiments.

Model	d	# parameters	Top-1 (%)
DynaMixer-S	1	26M	82.4
DynaMixer-S	2	26M	82.7
DynaMixer-S	4	27M	82.6
DynaMixer-S	8	29M	82.7

Table 6: The effects of reduced dimensionality.

Ablation on the designs of DynaMixer Block. In the DynaMixer block, the final results are from three components: row mixing, column mixing, and channel mixing. The final results are a weighted sum of the above three. In this paragraph, we study the effects of row mixing, column mixing, channel mixing, and reweighting by removing one of them in turn from our model while keeping the other components unchanged. The results are shown in Table 7. We can see that without row mixing or column mixing, the performance will decrease from 82.7% to about 79%. Moreover, if the channel mixing component is removed, the performance will drop 0.6% absolutely (from 82.7% to 82.1%). Without the reweighting step, the performance will also become worse. Thus, all the components in our DynaMixer block are necessary for our model. Finally, we evaluate the performance of our model when the parameters of DynaMixer operations for row mixing and column mixing are shared, which is a straightforward way to reduce the number of parameters. We can see that the accuracy only drops to 82.2%, which is still better than the other vision MLPs of similar size.

Model	# parameters	Top-1 (%)
DynaMixer-S	26M	82.7
- column mixing	23M	79.2
- row mixing	23M	79.4
- channel fusion	23M	82.1
- reweighting	24M	81.9
+ sharing DynaMixer op	23M	82.2

Table 7: The effects of different components in the DynaMixer block.

4.5 Transfer Learning

In this subsection, we show the advantages of our models on the transfer learning tasks. We transfer our pre-trained DynaMixer-S to downstream datasets such as CIFAR10 and CIFAR100. We finetune our Dynamixer-S with 60 epochs by using the SGD optimizer and cosine learning rate decay. The results are given in Table 8. We find that our DynaMixer can achieve higher accuracies than ViP and the original ViT with similar model sizes on the downstream datasets.

Model	dataset	# parameters	Top-1 (%)
ViT-S/16 [2]	CIFAR-10	49M	97.1
T2T-14 [53]	CIFAR-10	22M	97.5
ViP-S/7 [6]	CIFAR-10	25M	98.0
DynaMixer-S	CIFAR-10	26M	**98.2**
ViT-S/16 [2]	CIFAR-100	49M	87.1
T2T-14 [53]	CIFAR-100	22M	88.4
ViP-S/7 [6]	CIFAR-100	25M	88.4
DynaMixer-S	CIFAR-100	26M	**88.6**

Table 8: The results of fine-tuning the pre-trained DynaMixer on ImageNet to downstream datasets: CIFAR10 and CIFAR100.

5 Conclusion

MLP-based networks have recently been designed for vision recognition, targeting at introducing less inductive biases. The performances of MLP-based networks have been shown better than or comparable with transformer-based models.

Existing MLP-based vision models utilize static trainable mixing matrices to perform token information fusion. In this paper, we proposed a novel MLP-based vision architecture, called DynaMixer, which further improves the visual recognition performance. In DynaMixer, the mixing matrices are generated dynamically by leveraging the contents of all the tokens to be mixed. To reduce the time complexity and meanwhile improve the robustness, we exerted a dimensionality reduction technique and a multi-segment fusion mechanism. Our DynaMixer model has been demonstrated to achieve state-of-the-art performance on the ImageNet recognition task. We also provided an insightful analysis for gaining a deeper understanding of our model.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[3] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

[4] Jianyuan Guo, Yehui Tang, Kai Han, Xinghao Chen, Han Wu, Chao Xu, Chang Xu, and Yunhe Wang. Hire-mlp: Vision mlp via hierarchical rearrangement. *arXiv preprint arXiv:2108.13341*, 2021.

[5] Tan Yu, Xu Li, Yunfeng Cai, Mingming Sun, and Ping Li. S^2-mlp: Spatial-shift mlp architecture for vision. *arXiv preprint arXiv:2106.07477*, 2021.

[6] Qibin Hou, Zihang Jiang, Li Yuan, Ming-Ming Cheng, Shuicheng Yan, and Jiashi Feng. Vision permutator: A permutable mlp-like architecture for visual recognition. *arXiv preprint arXiv:2106.12368*, 2021.

[7] Shoufa Chen, Enze Xie, Chongjian Ge, Ding Liang, and Ping Luo. Cyclemlp: A mlp-like architecture for dense prediction. *arXiv preprint arXiv:2107.10224*, 2021.

[8] Tan Yu, Xu Li, Yunfeng Cai, Mingming Sun, and Ping Li. S^2-mlpv2: Improved spatial-shift mlp architecture for vision. *arXiv preprint arXiv:2108.01072*, 2021.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[13] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[16] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[17] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[18] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[19] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3464–3473, 2019.

[20] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. Deepvit: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*, 2021.

[21] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021.

[22] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. *arXiv preprint arXiv:2103.14899*, 2021.

[23] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *arXiv preprint arXiv:2103.00112*, 2021.

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[25] Wenxiao Wang, Lu Yao, Long Chen, Binbin Lin, Deng Cai, Xiaofei He, and Wei Liu. Crossformer: A versatile vision transformer hinging on cross-scale attention. *arXiv preprint arXiv:2108.00154*, 2021.

[26] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. *arXiv preprint arXiv:2104.11227*, 2021.

[27] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on visual transformer. *arXiv preprint arXiv:2012.12556*, 2020.

[28] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *arXiv preprint arXiv:2101.01169*, 2021.

[29] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, et al. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021.

[30] Dongze Lian, Zehao Yu, Xing Sun, and Shenghua Gao. AS-MLP: an axial shifted MLP architecture for vision. *CoRR*, abs/2107.08391, 2021.

[31] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention for transformer models. In *International Conference on Machine Learning*, pages 10183–10192. PMLR, 2021.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[34] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.

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

[36] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[37] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[38] Zihang Jiang, Qibin Hou, Li Yuan, Daquan Zhou, Xiaojie Jin, Anran Wang, and Jiashi Feng. Token labeling: Training a 85.5% top-1 accuracy vision transformer with 56m parameters on imagenet. *arXiv preprint arXiv:2104.10858*, 2021.

[39] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[40] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13001–13008, 2020.

[41] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[42] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[43] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.

[44] Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021.

[45] Yuki Tatsunami and Masato Taki. Raftmlp: Do mlp-based models dream of winning over computer vision? *CoRR*, abs/2108.04384, 2021.

[46] Chuanxin Tang, Yucheng Zhao, Guangting Wang, Chong Luo, Wenxuan Xie, and Wenjun Zeng. Sparse MLP for image recognition: Is self-attention really necessary? *CoRR*, abs/2109.05422, 2021.

[47] Jiachen Li, Ali Hassani, Steven Walton, and Humphrey Shi. Convmlp: Hierarchical convolutional mlps for vision. *arXiv preprint arXiv:2109.04454*, 2021.

[48] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[49] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.

[50] Yongming Rao, Wenliang Zhao, Zheng Zhu, Jiwen Lu, and Jie Zhou. Global filter networks for image classification. *arXiv preprint arXiv:2107.00645*, 2021.

[51] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

[52] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16519–16529, 2021.

[53] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis E.H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 558–567, October 2021.