RIFormer: Keep Your Vision Backbone Effective But Removing Token Mixer

Jiahao Wang[1,2] Songyang Zhang[1*] Yong Liu[3] Taiqiang Wu[3] Yujiu Yang[3]
Xihui Liu[2] Kai Chen[1*] Ping Luo[2] Dahua Lin[1]
[1]Shanghai AI Laboratory [2]The University of HongKong
[3]Tsinghua Shenzhen International Graduate School
wang-jh19@tsinghua.org.cn {zhangsongyang,chenkai,lindahua}@pjlab.org.cn
yang.yujiu@sz.tsinghua.edu.cn xihuiliu@eee.hku.hk pluo@cs.hku.hk

1. Detailed hyper-parameters of Sec.4

We provide some experimental settings of the exploration roadmap of Sec.4 in the main paper. Generally, we use a RIFormer-S12 model in this section, which is trained and evaluated on ImageNet-1K for 120 epochs. We take AdamW [3, 4] optimizer with batch size of 512 in all circumstances. For distillation experiments in Sec.4.2 and Sec.4.3, GFNet-H-B [5] serves as the teacher with a logit distillation following [8].

2. Proof of Eq.4

Given $\mathrm{T}^{(a)} \in \mathbb{R}^{N\times C\times H\times W}$, $\mathrm{T}'^{(a)} \in \mathbb{R}^{N\times C\times H\times W}$ respectively as the input and output of an affine residual sub-block in Fig.2-(a) in the main paper. During the training time, we have:

$$\mathrm{T}'^{(a)} = \mathrm{Affine}(\mathrm{LN}(\mathrm{T}^{(a)}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta}), \mathbf{s}, \mathbf{t}) - \mathrm{T}^{(a)} \tag{1}$$

where LN is the LN layer, $\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta}$ as the mean, standard deviation and learned scaling factor and bias of the LN layer, Affine is the affine transformation, $\mathbf{s} \in \mathbb{R}^C$ and $\mathbf{t} \in \mathbb{R}^C$ are its learnable scaling and shift parameters. During the training time, we have:

$$\mathrm{T}'^{(a)} = \mathrm{LN}(\mathrm{T}^{(a)}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}', \boldsymbol{\beta}') \tag{2}$$

According to the equivalence of the structural re-parameterization of the affine residual sub-block during training (Eq. 1) and inference (Eq. 2), for $\forall 1 \le n \le N$, $\forall 1 \le i \le C$, $\forall 1 \le h \le H$, $\forall 1 \le w \le W$, we have:

$$\begin{aligned} &((\mathrm{T}^{(a)}_{n,i,h,w} - \boldsymbol{\mu}_{n,h,w})\frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_{n,h,w}} + \boldsymbol{\beta}_i)\mathbf{s}_i + \mathbf{t}_i \\ &- (\mathrm{T}^{(a)}_{n,i,h,w} - \boldsymbol{\mu}_{n,h,w})\frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_{n,h,w}} - \boldsymbol{\beta}_i \\ &= (\mathrm{T}^{(a)}_{n,i,h,w} - \boldsymbol{\mu}_{n,h,w})\frac{\boldsymbol{\gamma}'_i}{\boldsymbol{\sigma}_{n,h,w}} + \boldsymbol{\beta}'_i \end{aligned} \tag{3}$$

Algorithm 1 Affine Transformation, PyTorch-like Code

```
import torch.nn as nn

class Affine(nn.Module):
    def __init__(self, in_features):
        super().__init__()
        self.affine = nn.Conv2d(
            in_features, in_features, kernel_size=1,
                stride=1, padding=0, groups=in_features
                , bias=True)

    def forward(self, x):
        """
        [B, C, H, W] = x.shape
        Subtraction of the input itself is added
        since the block already has a
        residual connection.
        """
        return self.affine(x) - x
```

Eq. 3 can be reformulated as:

$$\boldsymbol{\gamma}'_i = \boldsymbol{\gamma}_i(\mathbf{s}_i - 1), \quad \boldsymbol{\beta}'_i = \boldsymbol{\beta}_i(\mathbf{s}_i - 1) + \mathbf{t}_i, \tag{4}$$

Then Eq.4 in the main paper follows.

3. Code in PyTorch

3.1. PyTorch-like code of our affine operator

We provide the PyTorch-like code of the affine transformation in Alg. 1 affiliated with the training-time model in the RIFormer block. The affine transformation can be implemented as a depth-wise convolution by specifying the kernel size as 1 and the group number as the input channels. We follow [11] to add a subtraction of the input during implementation due to the residual connection.

3.2. PyTorch-like code of the RIFormer block

We provide the PyTorch-like code of the RIFormer block with the strctural re-parameterization in Alg. 2.

4. Detailed hyper-parameters on ablations

We provide the experimental settings of ablation studies of Sec.5 in the main paper.

For the 74.90% and 75.36% top-1 accuracy experiments in Tab.7 in the main paper, GFNet-H-B [5] serves as the teacher with a logit distillation, and PoolFormer-S12 [11] serves as the teacher for adopting the proposed module imitation strategy. For $(\mathcal{L}'_{in}, \mathcal{L}_{out})$ and $\mathcal{L}_{rel}$ in Eq.9 in the main paper, the number of epochs of using them are 80 and 20, respectively. The differences are as follows. In the 74.90% experiment, we set $\lambda_2 = \lambda_3 = 0$ in Eq.9 to obtain a hidden state feature distillation. In the 75.36% experiment, we choose λ_2 and λ_3 as in Tab. 1, and initialize the weights of RIFormer (except the affine operator) with the corresponding teacher network, as presented in Sec.4.4 of the main paper.

For Tab.9 in the main paper, we adopt the architectural modifications in [12] to our RIFormer, and construct two models with 12 and 18 layers as our student model, respectively. For distillation experiments in Tab.9, GFNet-H-B [5] serves as the teacher with a logit distillation. RandFormer-S12 [12], PoolFormer V2-S12 [12], ConvFormer-S18 [12], CAFormer-S18 [12] serve as teacher networks for module imitation. We train and evaluate on ImageNet-1K for 130 epochs (with 10 patient epochs).

5. Detailed hyper-parameters of ImageNet-1K

We provide the experimental settings of ImageNet-1K classification of Sec.5 in the main paper in Tab. 1. The hyper-parameters generally follow [11]. We use AdamW [4] optimizer with a batch size of 1024 and weight decay of 0.05, and learning rate $\mathrm{lr} = 1e^{-3} \cdot \mathrm{batch\ size}/1024$. Stochastic Depth [2], Label Smoothing [7] and Layer Scale [9] are also adopted to regularize the networks. For the RIFormer result with 224^2 input resolution in Tab.6 in the main paper, GFNet-H-B [5] serves as the teacher with a logit distillation, and a PoolFormer [11] with same parameter size serves as the teacher for adopting the proposed module imitation strategy. For the 384^2 RIFormer finetuning results, we use ConvFormer [12] as the teacher with a logit distillation and do not perform module imitation in the step. For the throughput measurement, we take a batch size of 2048 at 224^2 resolution (1024 for 384^2 resolution) with one 80GB A100 GPU and calculate the average throughput over 30 runs to inference that batch. The process is repeated for three times and the medium are treated as the statistical throughput.

6. Visualization of the learned coefficients

To further evaluate the effect of the proposed module imitation algorithm, we visualize the learned coefficients of the weights (denoted as $\mathbf{s}$) of the affine operator with (above the black dotted line) or without (below the black dotted line) the module imitation technique. Specifically, we provide the learned affine weights of a shallow block (Stage 1, Block 1), an intermediate block (Stage 3, Block 6), and a deep block (Stage 4, Block 1). As shown in Fig. 1, the affine weights trained using module imitation show differences with those of trained without such technique. Take Fig. 1-(c) as an instance. The affine weights without using module imitation are relatively more consistent and appear to show more positive values. As a comparison, module imitation help the affine operator learn more diverse and negative weights, which may useful for the expressiveness of our RIFormer. Similarly in Fig. 1-(b), the affine weights using module imitation have more moderate amplitude, compared to a higher amplitude of without the method.

7. Visualization of the activation parts

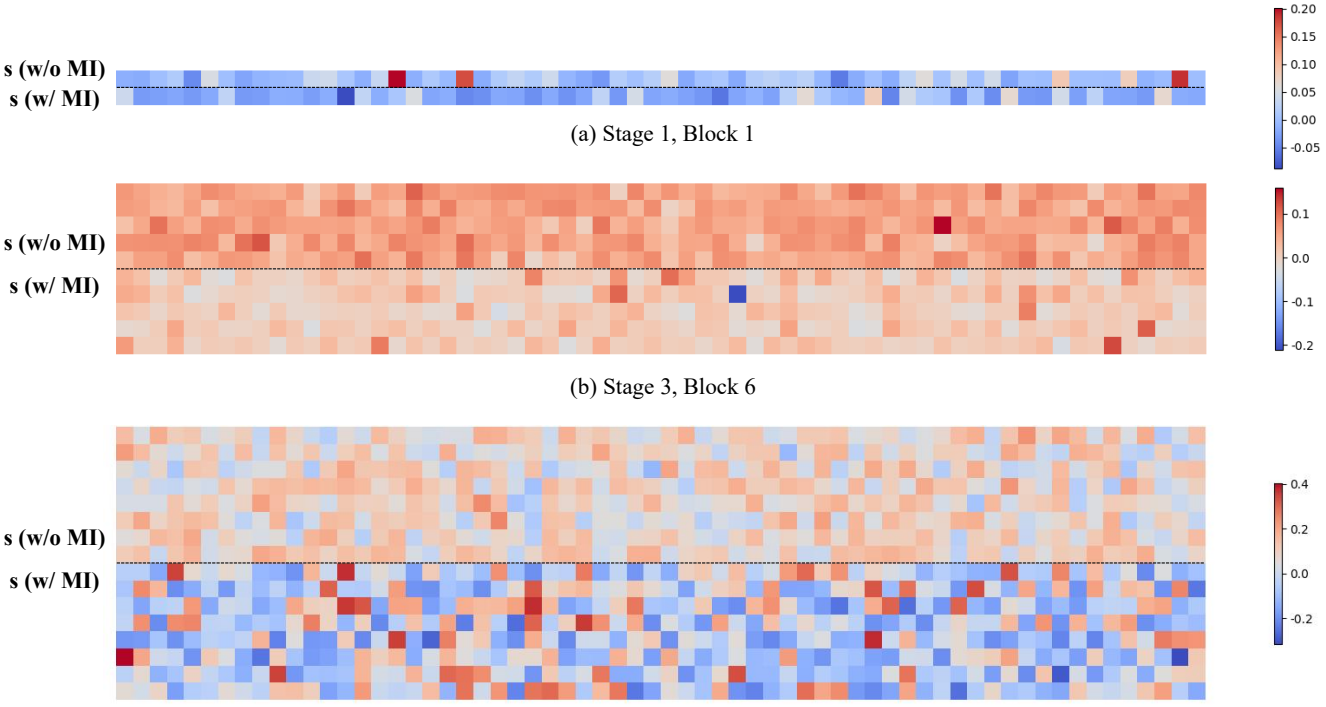
Following [11], we provide qualitative results of four different pre-trained backbones obtained by Grad-CAM [6], respectively RSB-ResNet50 [1, 10], DeiT-S [8], PoolFormer-S24 [11] and our RIFormer-S24. As observed in [11], the activation parts in the map of a transformer model are scattered, while that of a convnet are more aggregated. Interestingly, two additional observation can be made. Firstly, it seems that RIFormer trained with the proposed module imitation algorithm combines the characteristics of both convnet and transformer. We deem the reason might be that RIFormer has the same general architecture as transformer, but without any attention (*i.e.*, token mixer), and thus it is essentially a convnet. Secondly, the activation parts in the RIFormer map of show similar characteristics of that in PoolFormer, which may be due to the inductive bias implicitly incorporated from the teacher model via the knowledge distillation process.

	RIFormer				
	S12	S24	S36	M36	M48
Peak drop rate of stoch. depth d_r	0.1	0.1	0.1	0.1	0.1
LayerScale initialization ϵ	10^{-5}	10^{-5}	10^{-6}	10^{-6}	10^{-6}
$\lambda_1 \times \text{batch\_size}$ in Eq.9	0.0001	0.0003	0.0001	0.0001	0.0001
$\lambda_2 \times \text{batch\_size}$ in Eq.9	0.001	0.005	0.001	0.001	0.001
$\lambda_3 \times \text{batch\_size}$ in Eq.9	1.0	4.0	1.0	1.0	1.0
Data augmentation	AutoAugment				
Repeated Augmentation	off				
Input resolution	224				
Epochs	600				
Number of epochs of using $(\mathcal{L}'_{in}, \mathcal{L}_{out})$	400				
Number of epochs of using $\mathcal{L}_{rel}$	100				
Warmup epochs	5				
Hidden dropout	0				
GeLU dropout	0				
Classification dropout	0				
Random erasing prob	0.25				
EMA decay	0				
Cutmix α	1.0				
Mixup α	0.8				
Cutmix-Mixup switch prob	0.5				
Label smoothing	0.1				
Relation between peak learning rate and batch size	$\text{lr} = \frac{\text{batch\_size}}{1024} \times 10^{-3}$				
Batch size used in the paper	1024	1024	1024	1024	512
Peak learning rate used in the paper	1×10^{-3}				
Learning rate decay	cosine				
Optimizer	AdamW				
Adam ϵ	1e-8				
Adam (β_1, β_2)	(0.9, 0.999)				
Weight decay	0.05				
Gradient clipping	None				

Table 1. **Hyper-parameters for image classification on ImageNet-1K**

Algorithm 2 RIFormer Block (dubbed as AffineFormerBlock), PyTorch-like Code

```
import torch.nn as nn

class AffineFormerBlock(nn.Module):

   def __init__(self, dim, mlp_ratio=4., act_layer=nn.GELU, norm_layer=GroupNorm,
             drop=0., drop_path=0.,
             use_layer_scale=True, layer_scale_init_value=1e-5, deploy=False):
      super().__init__()
      if deploy:
         self.norm_reparam = norm_layer(dim)
      else:
         self.norm1 = norm_layer(dim)
         self.token_mixer = Affine(in_features=dim)
      self.norm2 = norm_layer(dim)
      mlp_hidden_dim = int(dim * mlp_ratio)
      self.mlp = Mlp(in_features=dim, hidden_features=mlp_hidden_dim,
                act_layer=act_layer, drop=drop)

      # The following two techniques are useful to train deep AffineFormers.
      self.drop_path = DropPath(drop_path) if drop_path > 0. \
         else nn.Identity()
      self.use_layer_scale = use_layer_scale
      if use_layer_scale:
         self.layer_scale_1 = nn.Parameter(
            layer_scale_init_value * torch.ones((dim)), requires_grad=True)
         self.layer_scale_2 = nn.Parameter(
            layer_scale_init_value * torch.ones((dim)), requires_grad=True)
      self.norm_layer = norm_layer
      self.dim = dim
      self.deploy = deploy

   def forward(self, x):
      if hasattr(self, 'norm_reparam'):
         if self.use_layer_scale:
            x = x + self.drop_path(
               self.layer_scale_1.unsqueeze(-1).unsqueeze(-1)
               * self.norm_reparam(x))
            x = x + self.drop_path(
               self.layer_scale_2.unsqueeze(-1).unsqueeze(-1)
               * self.mlp(self.norm2(x)))
         else:
            x = x + self.drop_path(self.norm_reparam(x))
            x = x + self.drop_path(self.mlp(self.norm2(x)))
      else:
         if self.use_layer_scale:
            x = x + self.drop_path(
               self.layer_scale_1.unsqueeze(-1).unsqueeze(-1)
               * self.token_mixer(self.norm1(x)))
            x = x + self.drop_path(
               self.layer_scale_2.unsqueeze(-1).unsqueeze(-1)
               * self.mlp(self.norm2(x)))
         else:
            x = x + self.drop_path(self.token_mixer(self.norm1(x)))
            x = x + self.drop_path(self.mlp(self.norm2(x)))
      return x

   def fuse_affine(self, norm, token_mixer):
      gamma_affn = token_mixer.affine.weight.reshape(-1)
      gamma_affn = gamma_affn - torch.ones_like(gamma_affn)
      beta_affn = token_mixer.affine.bias
      gamma_ln = norm.weight
      beta_ln = norm.bias
      print('gamma_affn:', gamma_affn.shape)
      print('beta_affn:', beta_affn.shape)
      print('gamma_ln:', gamma_ln.shape)
      print('beta_ln:', beta_ln.shape)
      return (gamma_ln * gamma_affn), (beta_ln * gamma_affn + beta_affn)

   def get_equivalent_scale_bias(self):
      eq_s, eq_b = self.fuse_affine(self.norm1, self.token_mixer)
      return eq_s, eq_b

   def switch_to_deploy(self):
      if self.deploy:
         return
      eq_s, eq_b = self.get_equivalent_scale_bias()
      self.norm_reparam = self.norm_layer(self.dim)
      self.norm_reparam.weight.data = eq_s
      self.norm_reparam.bias.data = eq_b
      self.__delattr__('norm1')
      if hasattr(self, 'token_mixer'):
         self.__delattr__('token_mixer')
      self.deploy = True
```

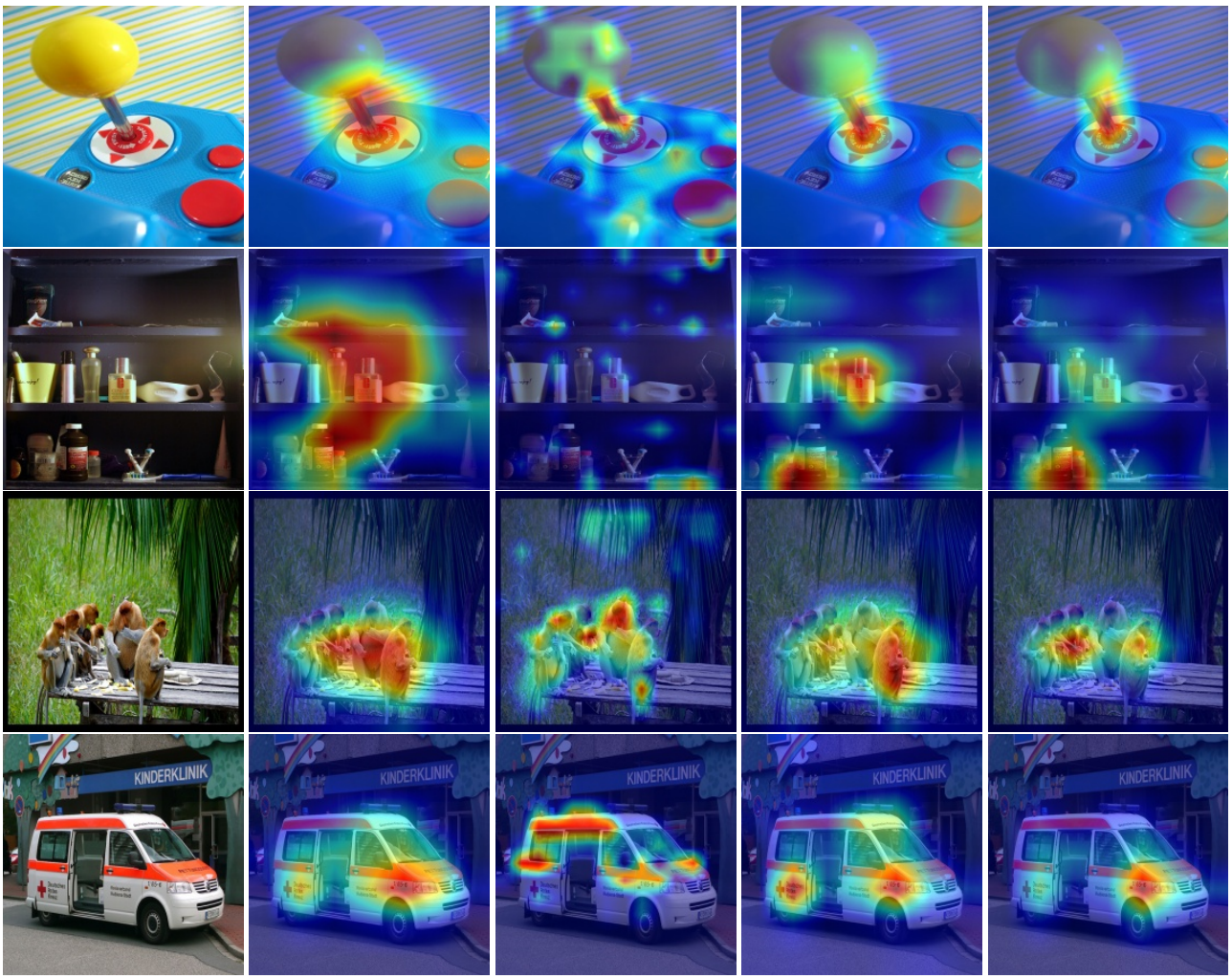


(a) Stage 1, Block 1

(b) Stage 3, Block 6

(c) Stage 4, Block 1

Figure 1. The heatmap of the learned coefficients of the affine transformation in (a) Stage 1, Block 1, (b) Stage 3, Block 6, (c) Stage 4, Block 1, respectively. The values of the learned coefficients are given different colors for positive and negative number.



Input RSB-ResNet-50 [10] DeiT-small [8] PoolFormer-S24 [11] RIFormer-S24

Figure 2. Grad-CAM [6] activation maps of four different pre-trained backbones on ImageNet-1K. We sample 4 images to visualize from the validation set.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2

[2] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 2

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 1

[4] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *ICLR*, 2019. 1, 2

[5] Yongming Rao, Wenliang Zhao, Zheng Zhu, Jiwen Lu, and Jie Zhou. Global filter networks for image classification. In *NeurIPS*, 2021. 1, 2

[6] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017. 2, 6

[7] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 2

[8] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 1, 2, 6

[9] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021. 2

[10] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476*, 2021. 2, 6

[11] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *CVPR*, 2022. 1, 2, 6

[12] Weihao Yu, Chenyang Si, Pan Zhou, Mi Luo, Yichen Zhou, Jiashi Feng, Shuicheng Yan, and Xinchao Wang. Metaformer baselines for vision. *arXiv preprint arXiv:2210.13452*, 2022. 2