RepMAF: When Re-parameterization Meets Multi-scale Attention

Haoming Liu
NYU Shanghai
hl3797@nyu.edu

Chen Song Zhang
NYU Courant
cz2119@nyu.edu

Jiayao Jin
NYU Shanghai
jj2915@nyu.edu

Abstract

This project proposes a novel RepMAF block, which can be used as the building block of feature extraction networks. The RepMAF block explores a paradigm of fusing features through channel-wise attention while its multi-branch design ensures the robustness of the representations of different scales. Moreover, the re-parameterizable property allows friendly deployment on mobile devices. Extensive experiments have been performed on the CIFAR-10 dataset, and the code is available at: `https://github.com/hmdliu/RepMAF`.

1. Introduction

Convolutional Neural Network (CNN) has achieved great success in visual understanding with various applications in wearable devices, IoT devices, and mobile phones. However, such devices require adequate visual recognition performance but lack in computational power to support it. With limited budget in computational resources, it is impractical to employ a model with an extensive amount of trainable parameters or intricate connections. Hence, it's meaningful to improve the performance of CNN without extra computations during model inference.

With more model architectures being proposed, we have an increasing number of powerful baseline models for feature extraction. However, the search of new architectures inevitably require a huge amount of human work and GPU hours. Accordingly, some plug-and-use modules (*e.g.*, SE block, SimAM block, *etc.*) seem to be a good solution, which can be directly combined with various up-to-date architectures to improve the performance [8, 17].

In this project, we aim to discover a lightweight yet effective building block for the models deployed on mobile and IoT devices. We propose a RepMAF block that can fully utilize multi-scale features through attentional fusion, and is also friendly to devices with limited computational power given the re-parameterizable property. Extensive experiments have been performed on the CIFAR-10 dataset to show the effectiveness of the proposed architecture [10].

2. Related Works

2.1. From Single-path to Multi-branch

After VGG achieved above 70% top-1 classification accuracy on the ImageNet dataset, there have been many research models dedicated to learning complementary features through multi-branch designs which inevitably complicates the model architecture while gaining better performace [1, 13]. The Inception network includes convolution branches with different filter sizes as well as a pooling branch to gain various receptive fields. ResNet proposed a simplified two-branch architecture which boosted the performance of deeper networks. DenseNet made the network topology even more complicated by connecting low-level and high-level layers [7, 9, 14].

Essentially, multi-branch designs enable different network components to have distinctive receptive fields, which increases the variety and robustness of the representations. However, the complicated branching strategy may reduce the degree of parallelism, hence slowing down the inference-time.

2.2. Structural Re-parameterization

Recently, the notion of structural re-parameterization is proposed, which aims to use different mathematically equivalent architectures for training and inference [5]. With properly designed building blocks, it is possible to apply multi-branch architectures for training and re-parameterize the fine-tuned model through merging some of its components. In this way, we are able to achieve an inference-time model with fewer parameters and faster inference speed.

There have been a few works that explored the effectiveness of structural re-parameterization. ACNet decomposes a $K \times K$ convolution layer into three branches with filters of size $K \times K$, $K \times 1$, and $1 \times K$ while training. ResRep is a lossless CNN pruning method based on the trick of re-parameterization, which can slim down a standard ResNet-50 with 76.15% accuracy on ImageNet to a narrower architecture with only 45% FLOPs and no accuracy drop. RepVGG designs a re-parameterizable building block that can be converted to a plain VGG-like network through mathematically equivalent transformations. DBB enhances the representational capacity of a single convolution by combining diverse branches of different scales and complexities to enrich the feature space, which introduces more choices for re-parameterizable branches [2, 3, 4, 5].

2.3. Attention Mechanism

Attention mechanisms have been widely used in computer vision tasks, serving as a tool to emphasize salient features as well as to suppress insignificant features. Non-local neural network introduced a non-local operation which captures long-range spatial dependencies. SENet proposed an SE block to model inter-dependencies between different channels and to recalibrate channel-wise features. SKNet adopted a channel attention module to adaptively adjust the receptive field size of the network based on multiple scales of input information [8, 11, 16].

3. Methods

3.1. Re-parameterizable Module

ResNet is one of the most prominent backbone architecture so far, whose information flow can be formulated as $y = g(x) + f(x)$, where f is learned by the residual branch, and g denotes the shortcut branch (*i.e.* $g(x) = x$ when the input channels and output channels are identical; otherwise, there is a mapping to the output channels through a 1×1 conv). A possible explanation for ResNet's success is that such a multi-branch architecture makes the model an implicit ensemble of numerous shallow models [15]. With two distinctive paths in each block, a ResNet with n blocks can be interpreted as an ensemble of 2^n models.

However, such a multi-branch design has its drawback at inference-time. It requires twice as much memory to store intermediate results, and incurs extra run-time to sum the results up. The recently proposed RepVGG block alleviates this problem to some extent [5]. It constructs a multi-branch structure that is *training-time-only*. When the input channels and output channels are identical, the information flow of RepVGG block can be denoted as $y = x + f(x) + g(x)$, where f is the residual branch with 3×3 conv, and g is an extra branch with 1×1 conv. In this way, we can get an ensemble of 3^n models with n RepVGG blocks.

If the network is simply a stack of RepVGG blocks, the inference-time model will be a VGG-like plain network, which is achieved by applying some mathematically equivalent transformations on the fine-tuned weights of a training-time network [5, 13]. From a high level perspective, the re-parameterization can be formulated as:

$$\begin{aligned} conv_{3\times3}(x, W_{rep}) = &BN(conv_{3\times3}(x, W_{3\times3})) \\ &+ BN(conv_{1\times1}(x, W_{1\times1})) \\ &+ BN(x) \end{aligned} \quad (1)$$

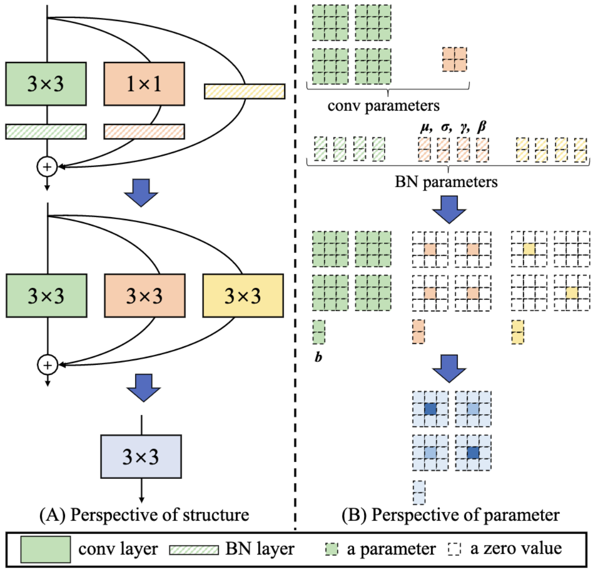

Figure 1. Structural re-parameterization of a RepVGG block, where $C_{in} = C_{out} = 2$. The weight of a 3×3 conv is of size $(2, 2, 3, 3)$, and weight of a 1×1 conv is of size $(2, 2, 1, 1)$. The figure is drawn from the original paper.

Notably, both identity mapping and 1×1 conv can be treated as a special case of a 3×3 conv [5]. Therefore, we can convert all of them to a mathematically equivalent 3×3 conv. For the identity branch, we can construct a identity matrix as the kernel; whereas for the 1×1 conv branch, we can pad the kernel to 3×3 with zeros. This way, we can achieve equivalent 3×3 conv at all three branches.

Additionally, as shown in Figure 1, the conv layer followed with a BN layer can be converted to conv layer with bias [5]. Here, we denote the kernel of a 3×3 conv layer as $W^{(3)} \in \mathbb{R}^{C_{out} \times C_{in} \times 3 \times 3}$, where C_{in} and C_{out} are the number of input and output channels respectively. We also denote the kernel of a 1×1 conv layer as $W^{(1)} \in \mathbb{R}^{C_{out} \times C_{in}}$, and denote the accumulated mean, standard deviation and learned scaling factor and bias of the BN layers as $\mu^{(i)}, \sigma^{(i)}, \gamma^{(i)}, \beta^{(i)}$, where $i = 0, 1, 3$ and corresponds to the BN layer at the identity, 1×1 conv, and 3×3 conv branch respectively. In addition, we denote the input and output as $X \in \mathbb{R}^{N \times C_{in} \times H_1 \times W_1}$ and $Y \in \mathbb{R}^{N \times C_{out} \times H_2 \times W_2}$ respectively, and denote the convolution operator as $*$. When $H_1 = H_2, W_1 = W_2$, we have:

$$\begin{aligned} Y = &BN(X * W^{(3)}, \mu^{(3)}, \sigma^{(3)}, \gamma^{(3)}, \beta^{(3)}) \\ &+ BN(X * W^{(1)}, \mu^{(1)}, \sigma^{(1)}, \gamma^{(1)}, \beta^{(1)}) \\ &+ BN(X, \mu^{(0)}, \sigma^{(0)}, \gamma^{(0)}, \beta^{(0)}) \end{aligned} \quad (2)$$

The last term can be omitted if $C_{in} \neq C_{out}$. Then, by the definition of a BN layer, for $1 \leq i \leq C_{out}$:

$$BN(X, \mu, \sigma, \gamma, \beta)_{:,i,:,:} = (X_{:,i,:,:} - \mu_i) \cdot \frac{\gamma_i}{\sigma_i} + \beta_i \quad (3)$$

Let $\{W', b'\}$ be the kernel and bias converted from $\{W, \mu, \sigma, \gamma, \beta\}$, we have:

$$W'_{i,:,:,:} = \frac{\gamma_i}{\sigma_i} \cdot W_{i,:,:,:}, \quad b'_i = -\frac{\mu_i \gamma_i}{\sigma_i} + \beta_i \tag{4}$$

Accordingly, it's easy to verify that $\forall\, 1 \leq i \leq C_{out}$:

$$(X * W')_{:,i,:,:} + b'_i = BN(X * W, \mu, \sigma, \gamma, \beta)_{:,i,:,:} \tag{5}$$

Then we can derive that:

$$\begin{aligned} Y &= X * W'^{(3)}_{3\times3} + X * W'^{(3)}_{1\times1} + X * W'^{(3)}_{idt} \\ &= X * W^{(3)}_{rep} \end{aligned} \tag{6}$$

Hence, we've shown the mathematical equivalence between the training-time and inference-time architecture of the RepVGG block. In this project, we use the RepVGG block to replace all the 3×3 convolution in the network, which can learn a more robust representation through each block.

3.2. Channel-wise Attention

Attention mechanisms have been widely used as a tool to emphasize salient features and to suppress insignificant features. Among the proposed attention blocks, the *Squeeze-and-Excitation* (SE) block is a simple yet effective one, which can adaptively recalibrate channel-wise feature responses by explicitly modeling interdependencies between channels [8]. As a plug-and-use module, the *Squeeze-and-Excitation* block can bring significant improvements in performance on most existing tasks with the cost of additional computations.

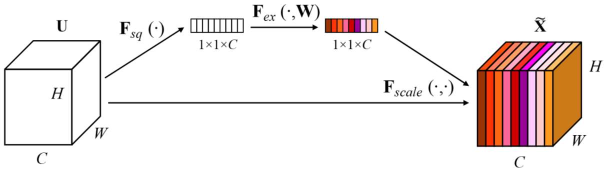
U
$\mathbf{F}_{sq}(\cdot)$
1×1×C
$\mathbf{F}_{ex}(\cdot,\mathbf{W})$
1×1×C
$\widetilde{\mathbf{X}}$
H
W
C
$\mathbf{F}_{scale}(\cdot,\cdot)$
H
W
C

Figure 2. Squeeze-and-Excitation Block. Fig. from original paper.

The SE building block includes two operators: *Squeeze* and *Excitation*. The whole *Squeeze* operator generates channel descriptors through global average pooling to aggregate channel-wise statistics. Formally, a channel embedding vector $z \in \mathbb{R}^C$ is generated by shrinking an input $U \in \mathbb{R}^{C\times H\times W}$ through its spatial dimensions, and then the c-th dimension of the embedding vector z is given by:

$$z_c = F_{sq}(U_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} U_c(i,j) \tag{7}$$

To make use of the information aggregated by the *Squeeze* operator, the *Excitation* operator aims to fully capture channel-wise dependencies through a gating mechanism with a sigmoid activation:

$$s = F_{ex}(z, \boldsymbol{W}) = \sigma(\boldsymbol{W_2}\delta(\boldsymbol{W_1}z)) \tag{8}$$

where δ denotes the ReLU fuction, σ denotes the Sigmoid function, $\boldsymbol{W_1} \in \mathbb{R}^{\frac{C}{r}\times C}$ and $\boldsymbol{W_2} \in \mathbb{R}^{C\times\frac{C}{r}}$ denote the weights of the two fully connected (FC) layers with a reduction ratio r[1], which significantly reduces the number of trainable parameters. The final output of a SE block is given by rescaling U with s through scalar multiplication:

$$\widetilde{X_c} = F_{scale}(U_c, s_c) = s_c U_c \tag{9}$$

Hence, we can rescale the channel dimension of the input feature map to refine the feature representation. The RepMAF module proposed in this project adopts this channel-wise rescaling idiom and applies it on feature fusion which we will discuss in the following section.

3.3. RepMAF Module

Originally, this project aims to find a re-parameterizable block that can learn more robust representations (compared to the RepVGG block), and we attempted to aggregate other re-parameterizable transformations in the multi-branch design, such as sequential convolutions without non-linearity and average pooling. Such designs were able to maintain the plain VGG-like structure after re-parameterization, and results in a faster inference speed. However, our experiments have demonstrated that adding new branches (to a structure that is re-parameterizable to plain network at inference-time) only provides a slight improvement on the classification accuracy, so we presented these experiments in the appendix.

In comparison, we found that maintaining a non-plain structure at inference-time usually secures a higher classification accuracy, but inevitably takes longer to inference. Specifically, we proposed a RepMAF block based on multi-scale features and channel-wise attention. With the re-parameterizable property preserved to some extent, it can be used as a building block in lightweight backbones deployed on mobile devices.

As shown in figure 3, a RepMAF module consists of two stages: feature extraction and feature fusion, where the former is done by two parallel RepVGG blocks at different scales, and the latter achieved by the Multi-scale Attention Fusion (MAF) block proposed in this project. It needs to be emphasized that the MAF block has two versions with distinctive guided features, and both methods have been evaluated in the Experiment section.

Inspired from the *Squeeze-and-Excitation* block, the Multi-scale Attention Fusion block is implemented via two operators: *Squeeze*, and *Select*. The former captures the

[1] A usual choice of the reduction ratio is $r = 16$.

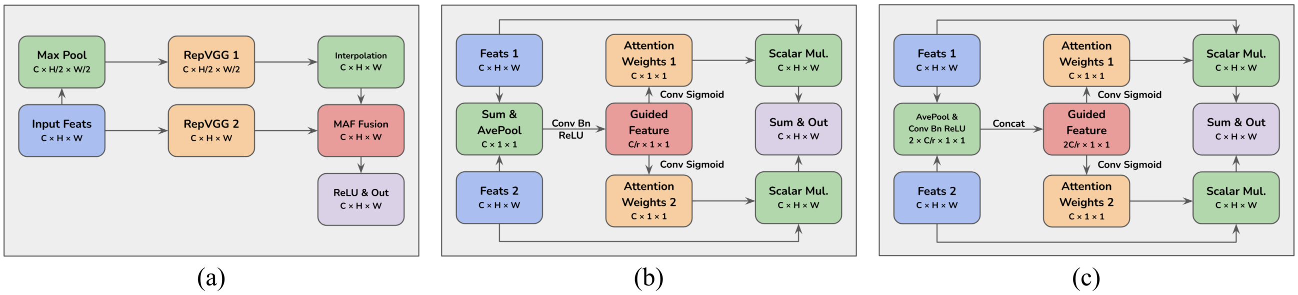

Figure 3. (a) RepMAF Block Architecture; (b) Multi-scale Attention Fusion Block V1; (c) Multi-scale Attention Fusion Block V2.

channel-wise interdependencies in the two feature maps and generates an embedding vector as the guided feature for fusion; whereas the latter models the importance of each input channel through a gating mechanism, and adaptively select and fuse useful feature from the input feature maps.

Squeeze. Let $X, Y \in \mathbb{R}^{C\times H\times W}$ be the input feature maps, where one of them is obtained by the nearest neighbor interpolation. Our goal is to find a mapping $f_{sq}(X, Y) = g$, where g is the guided feature used for the ensuing fusion stage. Notably, the *Squeeze* operator in the MAF block includes the functionality of both the *Squeeze* operator and part of the *Excitation* operator in SE block [8]. The implementation of the two versions are as follow:

Version 1 This version aims to generate the guided feature $g \in \mathbb{R}^{\frac{C}{r}}$ through a raw fusion, where r is a constant reduction ratio. Formally, we first apply a raw fusion via element-wise summation:

$$U = X + Y \tag{10}$$

Here, we also use a global average pooling to aggregate the channel embedding vector $z \in \mathbb{R}^C$, similar to equation 7. Then, we compress the channel-wise statistics to obtain the guided feature $g \in \mathbb{R}^{\frac{C}{r}}$:

$$g = F_{sq}(z, \boldsymbol{W_{sq}}) = \delta(BN(\boldsymbol{W_{sq}}z)) \tag{11}$$

where δ denotes the ReLU function, BN is a batch norm layer, and $\boldsymbol{W_{sq}} \in \mathbb{R}^{\frac{C}{r}\times C}$ is a fully connected (FC) layer.

Version 2 Different from the previous version, this version aims to preserve the channel-wise statistics from the input feature maps. Hence, we apply global average pooling on X and Y respectively to obtain the channel embedding vectors $z_1, z_2 \in \mathbb{R}^C$. Then, we use a shared guided feature generator F_{sq} (as we did in equation 11) to compute the guided feature $g_1, g_2 \in \mathbb{R}^{\frac{C}{r}}$ respectively:

$$g_1 = F_{sq}(z_1, \boldsymbol{W_{sq}}) = \delta(BN(\boldsymbol{W_{sq}}z_1)) \tag{12}$$

$$g_2 = F_{sq}(z_2, \boldsymbol{W_{sq}}) = \delta(BN(\boldsymbol{W_{sq}}z_2)) \tag{13}$$

where δ denotes the ReLU function, BN is a batch norm layer, and $\boldsymbol{W_{sq}} \in \mathbb{R}^{\frac{C}{r}\times C}$ is a fully connected (FC) layer. Finally, we can get the guide feature $g = [g_1 \;\; g_2] \in \mathbb{R}^{\frac{2C}{r}}$ through concatenation.

Select. Let $g \in \mathbb{R}^{C_g}$ be the guided feature we obtained through the *Squeeze* operator. Our goal is to generate the channel attention weights $s_1, s_2 \in \mathbb{R}^C$ to scale the input feature maps by a coefficient between 0 and 1, and sum the scaled feature maps as the fusion output.

Here, we adopt two Sigmoid gates to generate channel-wise attention weights:

$$s_1 = \sigma(\boldsymbol{W_1}g), \quad s_2 = \sigma(\boldsymbol{W_2}g) \tag{14}$$

where σ demotes the Sigmoid function, and $\boldsymbol{W_1}, \boldsymbol{W_2} \in \mathbb{R}^{C\times C_g}$ are two fully connected (FC) layers corresponding to the two input feature maps. Then, the fusion output is given by:

$$\widetilde{X_c} = F_{select}(X_c, Y_c, s_{1_c}, s_{2_c}) = s_{1_c}X_c + s_{2_c}Y_c \tag{15}$$

In this way, we can get a refined feature map based on adaptively selected features from both inputs.

3.4. Network Architecture

For a faster iteration of block architectures, this project uses a lightweight setting to evaluate the effectiveness of the building blocks. Apart from the input block that uses 5 × 5 convolution (padding = 2), the rest of the feature extraction network adopts the building blocks to be evaluated by default (*e.g.*, 3 × 3 conv layer, RepVGG block, and RepMAF block, *etc.*). The channels for each stages are $[64, 128, 256, 512]$, and the number of blocks at each stage are $[1, 4, 3, 1]$ respectively. A simple linear classifier is used for classification, which consists of a global average pooling layer, a dropout layer, and a FC layer mapping to the number of classes.

4. Experiments

4.1. Dataset and Implementation Details

We evaluate the performance of our models on the CIFAR-10 dataset [10]. It consists of colored natural images with 32×32 pixels drawn from 10 classes. The training

Building Block (& Attention)	Top-1 Acc.(%)	Train Params (M)	Inference Params (M)	Train Fwd Speed (img/s)	Inf. Fwd Speed (img/s)
VGG	93.02	3.66	3.66	3484	3484
VGG + SE	93.41	3.74	3.74	3263	3263
RepVGG	93.26	4.08	3.66	2889	3451
RepVGG + SE	93.68	4.15	3.74	2745	3275
BiRepVGG	93.22	8.14	7.31	1632	2010
BiRepVGG + SE	93.66	8.29	7.46	1542	1878
RepMAF-V1	94.26	8.25	7.43	2170	2617
RepMAF-V2	94.36	8.33	7.50	2184	2640

Table 1. Results on CIFAR-10 trained for 150 epochs with standard data augmentation.

and test sets contain 50,000 and 10,000 images respectively. We adopt a standard data augmentation scheme (*i.e.* mirroring and shifting) that is widely used on this dataset [7, 9]. We trained for 150 epochs with a batch size of 64 and used a SGD optimizer with a momentum of 0.9. The initial learning rate is 0.1 and is adjusted according to a polynomial scheduler [2]. We use a weight decay of 0.0005 and set the dropout in the classifier as 0.5. Though the training setting might be better tuned for each building block, we still use the same setting for all the experiments to ensure a fair comparison and analysis.

The experiments are performed on the Greene cluster of NYU HPC with a single RTX8000 GPU. The forward speed is estimated by the average speed for inferring the CIFAR-10 validation set 20 times. All the classification accuracy and forward speed reported in this project are the median of at least three experiments with identical model settings.

4.2. Network Performance

Table 1 compares RepMAF with the classic designs and their combinations, including VGG, RepVGG, and SE block [5, 8, 13]. We evaluate the top-1 accuracy for classification as well as the number of trainable parameters at both training-time and inference-time. As the RepVGG paper suggested, theoretical FLOPs may not be a good measure for the density of computation due to the complexity of branching [5]. Hence, we evaluate the speed of inference directly on a RTX8000 GPU with a batch size of 64. Considering the fact that a RepMAF block has two parallel RepVGG blocks, which contains roughly twice as much parameters as the traditional RepVGG blocks, we construct a BiRepVGG block[3] that has a similar parallel structure to maintain our motif of fair comparison. By default, we insert the SE blocks between the convolution unit and fusion/activation unit.

In terms of classification accuracy, we can observe that the SE block indeed improves the classification accuracy with slight additional computation costs. Comparing the results from RepVGG and BiRepVGG, we can assert that simply stacking branches may not secure a better performance; whereas the RepMAF blocks, achieves the highest accuracy under the same training setting while taking advantage of a properly-designed feature fusion module. Clearly, the trick of structural re-parameterization reduces the number of trainable parameters at inference-time and boosts the speed of inference.

4.3. Comparison with variants

Building Block (& Attention)	Data Aug.	Attention Layers	Params (M)	Top-1 Acc.(%)
RepVGG		-	4.08	91.35
RepVGG	✓	-	4.08	93.31
RepVGG + SE		1	4.65	91.33
RepVGG + SE	✓	1	4.65	93.59
RepVGG + SE		2	4.15	91.39
RepVGG + SE	✓	2	4.15	93.68

Table 2. Comparison on data augmentation and channel attention design. The 2-layer SE block sets reduction ratio $r = 16$.

Building Block (& Attention)	Branch 1 Feats Size	Branch 2 Feats Size	Top-1 Acc.(%)
BiRepVGG + SE	16×16	16×16	93.66
BiRepVGG + SE	16×16	8×8	91.52
BiRepVGG + SE	8×8	8×8	91.12
RepMAF-V1	16×16	16×16	94.05
RepMAF-V1	16×16	8×8	94.26
RepMAF-V1	8×8	8×8	91.57
RepMAF-V2	16×16	16×16	94.09
RepMAF-V2	16×16	8×8	94.36
RepMAF-V2	8×8	8×8	91.23

Table 3. Comparison on the size of feature maps.

[2] $lr = base\_lr \times 0.9^{1-\frac{curr\_iter}{total\_iter}}$

[3] The two branches of the BiRepVGG block use feature maps of size 16×16, which achieves the highest accuracy among all the settings.

Table 2 evaluates the SE block in terms of data augmentation and the number of FC layers [8]. Interestingly, the SE block only achieves a significant improvement on the accuracy when we apply the standard data augmentation. Apart from data augmentation, the results have demonstrated that the bottleneck design in the SE block can boost the performance without introducing many parameters.

Table 3 explores how the size of the feature maps influence the performance of the BiRepVGG block and the RepMAF block. The results have shown that the larger the feature maps are, the higher accuracy that the BiRepVGG block can achieve. On the contrary, for the RepMAF block, the multi-scale combination seems to be the preeminent choice. We believe that the feature fusion method implemented in the RepMAF block is effective in integrating complementary features from the given input.

5. Discussion

Overall, the proposed RepMAF block has achieved a satisfying classification accuracy with a reasonable inference speed. In summary, we accredit the success of the RepMAF block to three specific designs: 1) The MAF fusion module is able to adaptively integrate complementary features from the inputs through channel-wise attention, which is a novel attempt according to our knowledge; 2) The RepVGG block learns more robust representations during training (without a cost of inference speed); 3) The RepMAF block has receptive fields at different scales, which is achieved by the parallel structure.

A potential improvement on the implementation of the RepMAF block is to merge the convolution branches through group convolution, which may further speed up the inference. Also, we would like to emphasize that the potential of structural re-parameterization has only been exploited to a limited extent in this project. Based on our experiments, we believe that using structural re-parameterization is the future direction in multi-branch models since a plain VGG-like structure usually causes gradient explosion or disappearance when the models become deeper.

Meanwhile, we are glad to see some recent works, which aims to implicitly integrate identity mapping through structural re-parameterization in a plain network, or constructs non-deep networks with re-parameterizable building blocks [6, 12]. Inspired from their works, we may further explore the potential of structural re-parameterization in the future.

Acknowledgements. Special thanks to Professor Rob Fergus (CSCI-GA 2271: Computer Vision, Fall 2021).

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[2] X. Ding, Y. Guo, G. Ding, and J. Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks, 2019.

[3] X. Ding, T. Hao, J. Tan, J. Liu, J. Han, Y. Guo, and G. Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting, 2021.

[4] X. Ding, X. Zhang, J. Han, and G. Ding. Diverse branch block: Building a convolution as an inception-like unit, 2021.

[5] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13733–13742, 2021.

[6] A. Goyal, A. Bochkovskiy, J. Deng, and V. Koltun. Non-deep networks, 2021.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[8] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu. Squeeze-and-excitation networks, 2019.

[9] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks, 2018.

[10] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *https://www.cs.toronto.edu/ kriz/cifar.html*, 55(5), 2014.

[11] X. Li, W. Wang, X. Hu, and J. Yang. Selective kernel networks, 2019.

[12] F. Meng, H. Cheng, J. Zhuang, K. Li, and X. Sun. Rmnet: Equivalently removing residual connection from networks, 2021.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014.

[15] A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks, 2016.

[16] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks, 2018.

[17] L. Yang, R.-Y. Zhang, L. Li, and X. Xie. Simam: A simple, parameter-free attention module for convolutional neural networks. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11863–11874. PMLR, 18–24 Jul 2021.

6. Appendix

This section discusses some abandoned ideas that has limited or no improvements on the classification accuracy. The implementations can be found in the full codebase, which is less compact compared with the one in the abstract, but contains most of our attempts. The sheet for experiment results is also available here.

6.1. More Re-parameterizable Branches

We introduce two extra branches into the RepVGG block: sequential convolution without non-linearity ($1 \times 1 - 3 \times 3$), and average pooling (kernel_size=3, padding=1). The mathematical equivalence of re-parameterizing such branches can be found in the Diverse Branch Block paper [4]. The code corresponding to this part is the `DBB_Module` class under *basic.py*.

Our results have demonstrated that the classification accuracy provided by the 5-branch design is approximately the same as that of the original 3-branch design. One possible explanation is that when all the re-parameterizable branches are passed through various layers, they are essentially still linear transformations so the representational power may never be as good as a simple non-linear activation.

6.2. Multi-head & Unified Branch Dropout

We include multiple RepVGG blocks (without activation) that works in parallel for each building block to follow a multi-head design. The output feature maps are fused through element-summation or an extra 1×1 conv layer (with the weights initialized to mimic a element-wise summation or an element-wise average). To ensure each branch are learning various features, we proposed a unified branch dropout strategy to randomly disable some of the parallel branches during training. For example, assume we have three branches $[1, 2, 3]$ at each building block, and for a batch of training samples, branch 1 is randomly picked to be disabled. Then, branch 1 at all the building blocks will be set to 0 during this iteration. Notably, the parallel branches are also re-parameterizable since non-linear activations are only applied after branch fusion. The code corresponding to this part is the `RepTree_Module` class under *basic.py*.

We were hoping that such a design can achieve self-ensembling at training-time, but it did not perform to our expectations. The version without branch dropout performs similar to the baseline RepVGG model, which again reflects the limitation of parallel linear transformations. On the other hand, the version with unified branch dropout accuracy performs less than the baseline accuracy. One possible explanation is that the branch-wise correspondence did not work as anticipated, and the distribution of the feature maps vary a lot in different iterations, which inevitably made it harder to learn stable representations.

6.3. Multi-scale Shuffle

This idea is a prototype of the proposed RepMAF block where we attempted to integrate multi-scale features in two parallel RepVGG blocks. Different from the current version, two feature maps of size 16×16 and 8×8 are forwarded through the network. In particular, both of them incorporate features from the other after the convolution unit, where the 8×8 features are up-sampled via nearest neighbor interpolation and are added to the 16×16 feature map. The 16×16 features are downsampled via max pooling and fused with the 8×8 feature map. The code corresponding to this part is the `RepMSS_Module` class under *basic.py*.

This design increases the accuracy to some extent and sets up the cornerstone for the RepMAF block.

6.4. Super Augmentation

This attempt expands the standard data augmentation we used in the experiments (*i.e.* mirroring and shifting). The super augmentation approach stacks five individually-applied transformations, including crop and flip, color jitter, random affine, random perspective, and Gaussian blur. Accordingly, the augmented dataset is five times larger and thus requires much longer time to train. Corresponding code can be found under *dataset.py*.

The super augmentation undoubtedly improves the classification accuracy, but we still adopted the standard augmentation in the experiments due to the reason of excessive training time for the super augmentations.