RepVGG:
Making VGG-style ConvNets Great Again

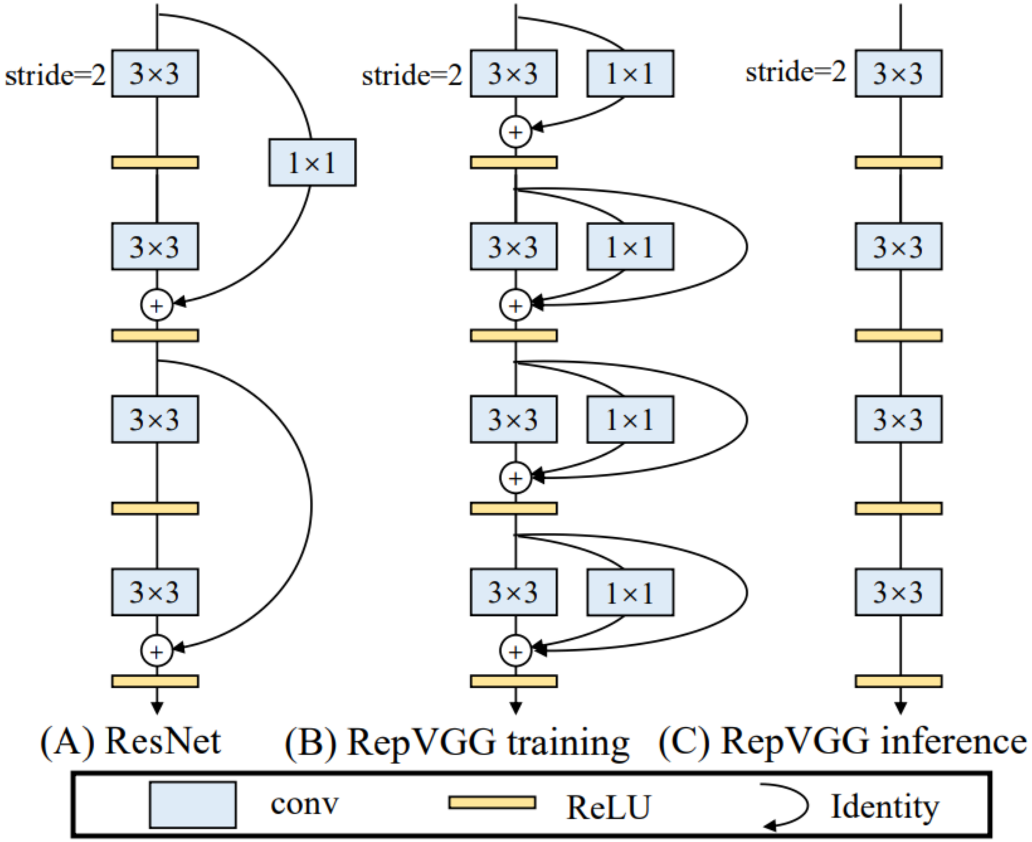
(CVPR, 2021)

Jaehoon Jang
Korea University

Introduction

RepVGG: Making VGG-style ConvNets Great Again 

- CNN architecture
- Proposes a "structural re-parameterization" technique
- Blends the simplicity of VGG-style networks with the performance benefits of multi-branch models



Background

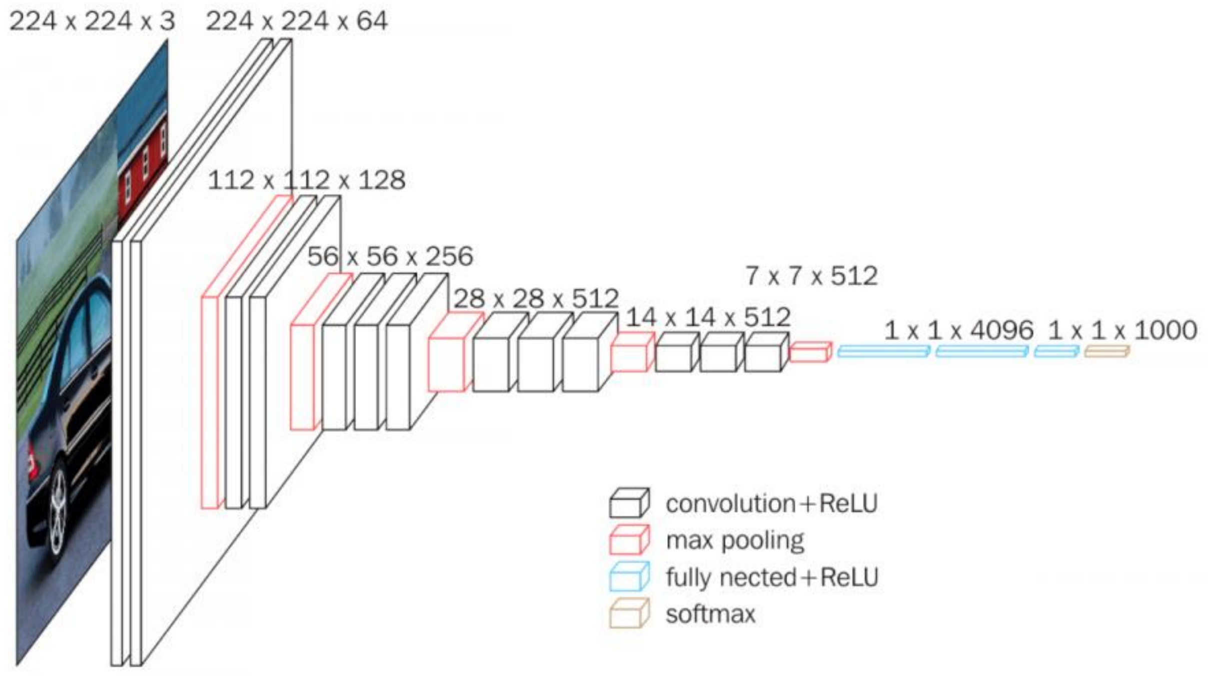
What is VGG-style ConvNet?

- **Simple, Sequential, Deep, Uniform Design**

  The network depth is consistent, with each layer (convolution, ReLU, pooling) added in a uniform fashion, making it intuitive to scale by adding more layers

- **Convolutional Layers Only**

  Primarily use 3x3 convolutional layers stacked with increasing depth, removing complex modules like multi-branch structures or residual connections



VGG Architecture, 92.7% on Top-5, ImageNet Challenge

Background

What is Multi-branch Model?

- **Definition**

Multi-branch models are convolutional neural network (CNN) architectures where multiple branches or parallel paths process the input simultaneously.

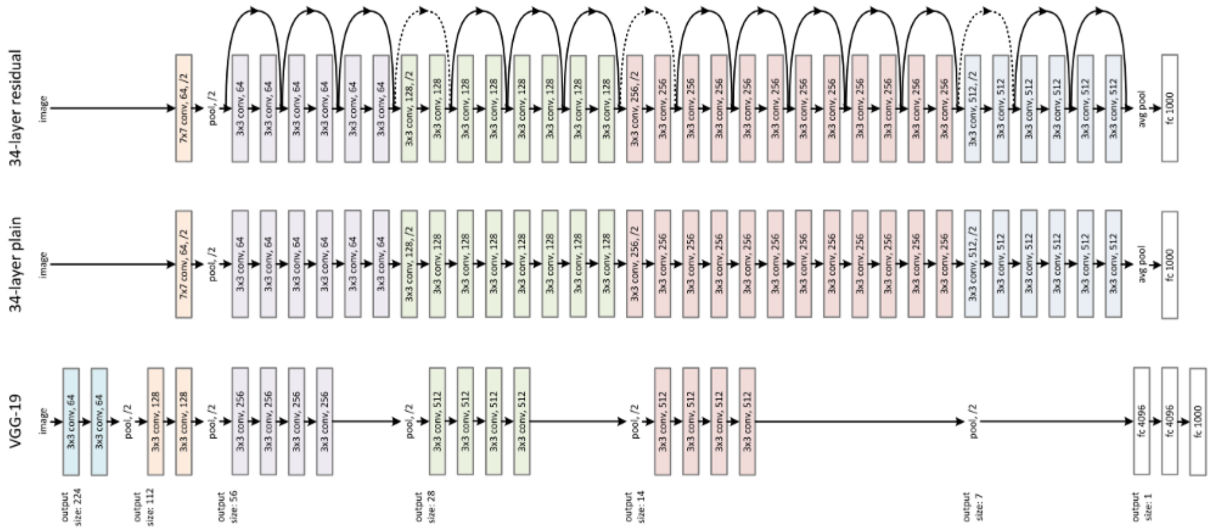
- **Advantages**

The multi-branch setup can be seen as an ensemble of smaller networks within one model, helping it learn a richer variety of features and address vanishing gradient problems, or helping gradient flow

- **Trade-Offs**

While multi-branch models enhance accuracy and robustness, they are often computationally intensive, slower in inference, and more complex to deploy efficiently

Multiple branches, as direct paths in ResNet

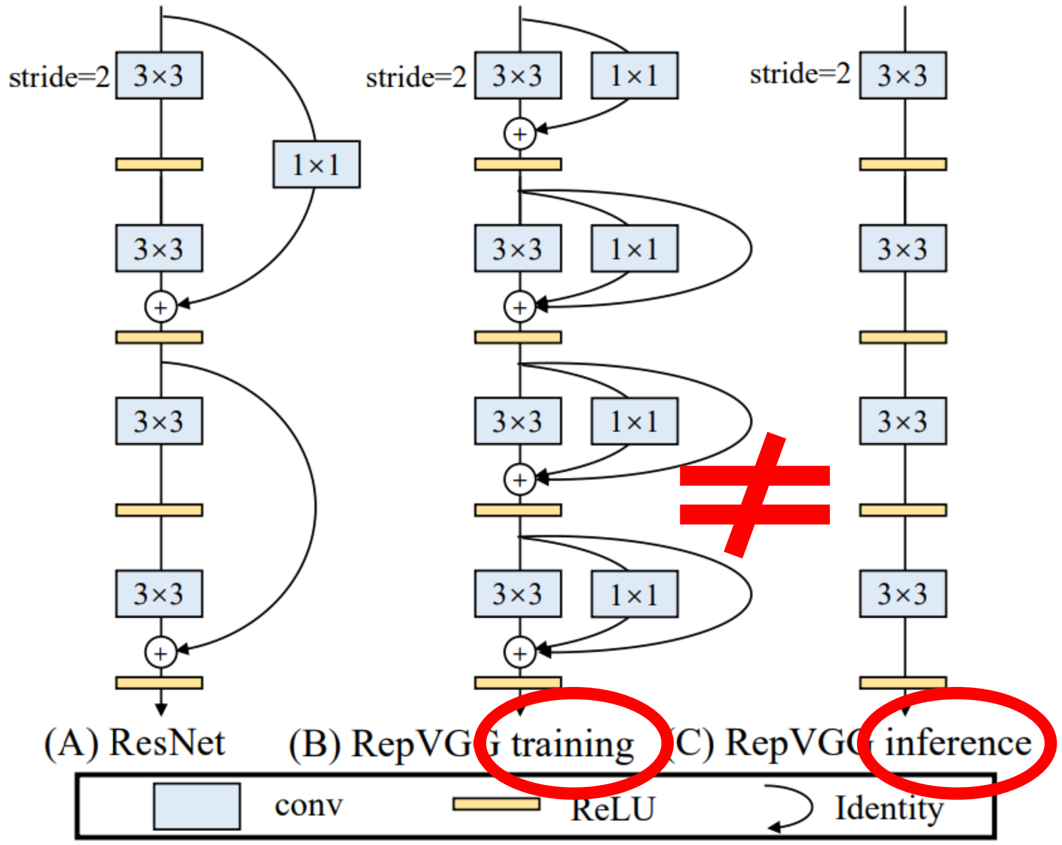


ResNet architecture, one of Multi-branch Models

Method

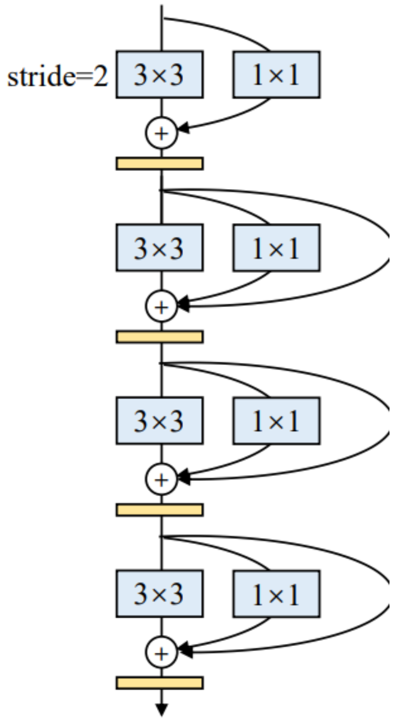
General Overview of Method

- VGG-style ConvNet
- During training, uses a multi-branch topology but transforms into a single-branch structure with only 3x3 convolutions and ReLU at inference time by using a re-parameterization technique.
- Achieves higher accuracy and faster speed compared to ResNet variants, while also performing on par with state-of-the-art models like EfficientNet and RegNet.



RepVGG: Taking advantage of both sides, Training vs Inference

Training

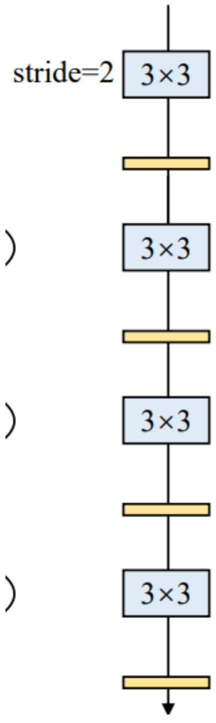


Allows the model to learn more effectively.

Each branch can act as a separate "pathway" or sub-model, allowing the network to learn different features simultaneously.

This setup can be seen as an ensemble of smaller networks within the larger network, which enhances gradient flow, mitigates gradient vanishing, and ultimately improves model accuracy and convergence

Inference



Computationally complex and memory-intensive during inference.

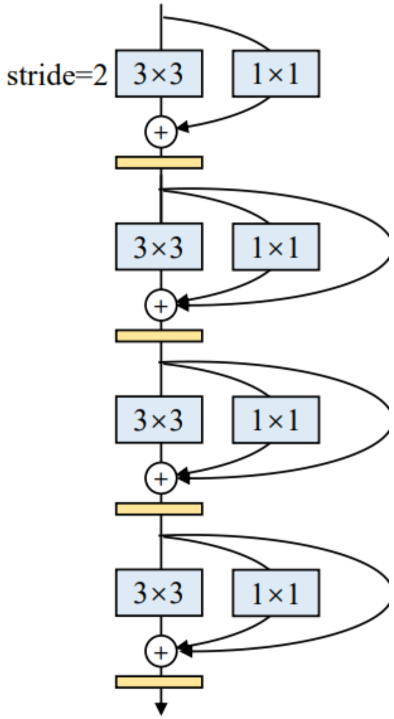
Each branch requires separate memory allocation, which increases memory access costs and can slow down inference speed due to parallelism limitations on hardware.

This complexity can make multi-branch models inefficient and challenging to deploy in real-time or resource-constrained environments

Method

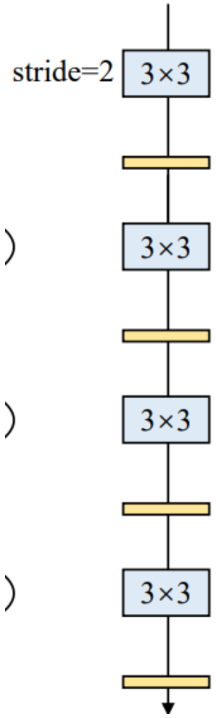
Solution: Re-parameterization Trick

Training



Re-parameterize!

Inference



Method

How re-parameterization works

- Transform the training-time multi-branch structure into a single 3x3 convolution layer for inference
- Use structural re-parameterization to fuse the 3x3, 1x1, and identity branches into a single, efficient convolutional layer
- During training, each RepVGG block contains:
  - A 3x3 Conv branch
  - A 1x1 Conv branch
  - An identity (skip) connection branch
- These are fused into a single 3x3 Conv
- Parameter-wise,
  - Conv parameters (kernel weights, biases)
  - BatchNorm parameters (mean μ, variance σ, scaling factor γ, and shift β

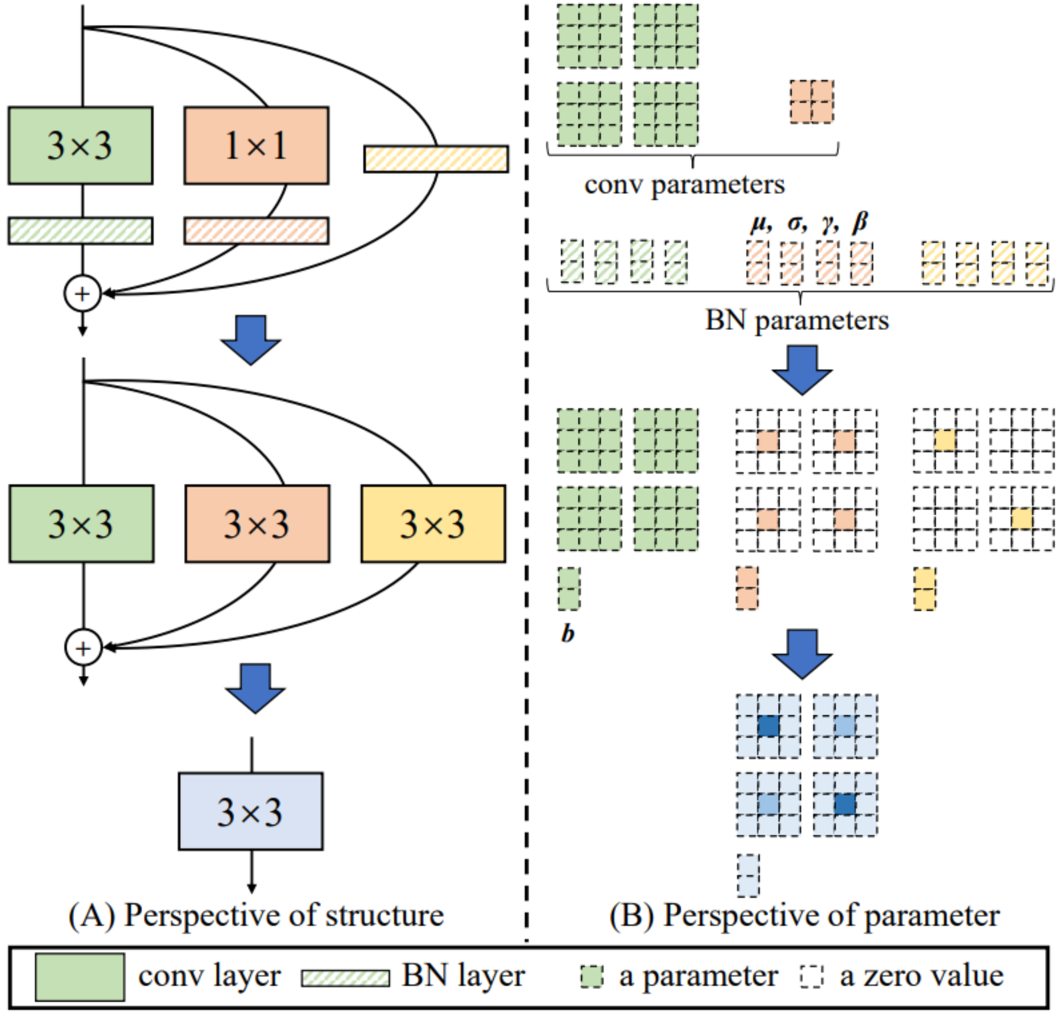


Figure 4: Structural re-parameterization of a RepVGG block. For the ease of visualization, we assume $C_2 = C_1 = 2$, thus the 3×3 layer has four 3×3 matrices and the kernel of 1×1 layer is a 2×2 matrix.

Method

Step-by-Step re-parameterization

- Step 1. Convert Convolutions with BN to a Single Conv Layer with Bias
  - Each convolution (3x3 and 1x1) is followed by a Batch Normalization (BN) layer, which has parameters
  - μ (mean), σ (variance), γ (scale), and β (shift) for normalization
- To merge BatchNorm with Conv
  - Adjust the kernel weights and biases to incorporate the BN effect, transforming them into a single convolution with bias

$$W' = \frac{\gamma}{\sigma} W, \quad b' = \beta - \frac{\mu\gamma}{\sigma}$$

$$W^{'}_{3x3}, b^{'}_{3x3}, W^{'}_{1x1}, b^{'}_{1x1}$$

- Applies to both the 3x3 and 1x1 branches, producing modified weights and biases

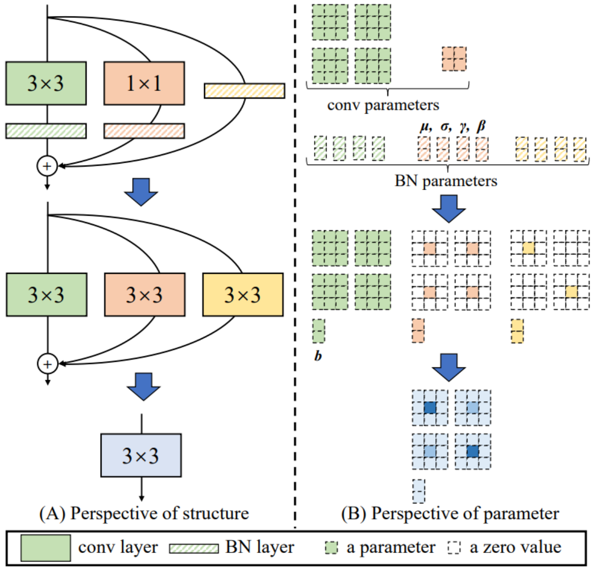


Figure 4: Structural re-parameterization of a RepVGG block. For the ease of visualization, we assume $C_2 = C_1 = 2$, thus the 3×3 layer has four 3×3 matrices and the kernel of 1×1 layer is a 2×2 matrix.

Method

Step-by-Step re-parameterization

- Step 2. Padding 1x1 and Identity Branches to Match 3x3
  - 1 x 1 Kernel Expansion: Zero-Pad 1 x 1 kernel to match the 3 x 3 dimensions
  - Identity Branch: Also padded to match the 3 x 3 dimensions

$$W_{1x1}^{\text{padded}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & W_{1x1} & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad I^{\text{padded}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

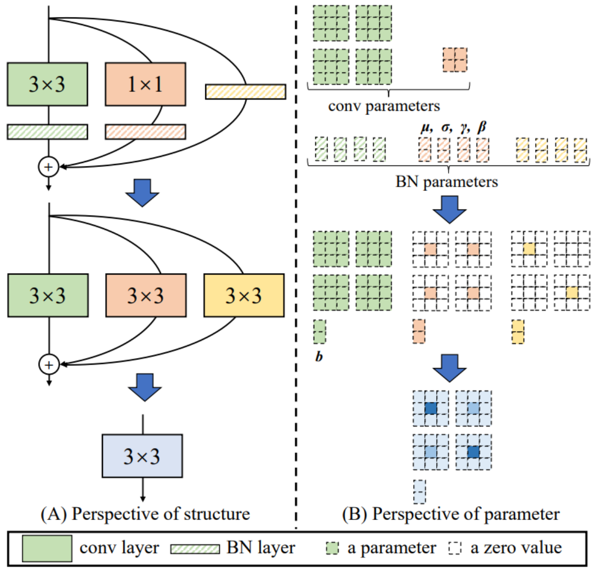


Figure 4: Structural re-parameterization of a RepVGG block. For the ease of visualization, we assume $C_2 = C_1 = 2$, thus the 3×3 layer has four 3×3 matrices and the kernel of 1×1 layer is a 2×2 matrix.

- Step 3. Fuse All Kernels and Biases
  - Kernel Fusion: Sum the kernels of the 3x3, padded 1x1, and padded identity branches
  - Bias Fusion: Sum the biases from each branch to form the final bias for the fused 3x3 convolution

$$W_{\text{fused}} = W_{3x3} + W_{1x1}^{\text{padded}} + I^{\text{padded}}$$

$$b_{\text{fused}} = b_{3x3} + b_{1x1} + b_{\text{identity}}$$

Experiment

Table 4: Results trained on ImageNet with simple data augmentation in 120 epochs. The speed is tested on 1080Ti with a batch size of 128, full precision (fp32), and measured in examples/second. We count the theoretical FLOPs and Wino MULs as described in Sect. 2.4. The baselines are our implementations with the same training settings.

Model	Top-1 acc	Speed	Params (M)	Theo FLOPs (B)	Wino MULs (B)
RepVGG-A0	72.41	3256	8.30	1.4	0.7
ResNet-18	71.16	2442	11.68	1.8	1.0
RepVGG-A1	74.46	2339	12.78	2.4	1.3
RepVGG-B0	75.14	1817	14.33	3.1	1.6
ResNet-34	74.17	1419	21.78	3.7	1.8
RepVGG-A2	76.48	1322	25.49	5.1	2.7
RepVGG-B1g4	77.58	868	36.12	7.3	3.9
EfficientNet-B0	75.11	829	5.26	0.4	-
RepVGG-B1g2	77.78	792	41.36	8.8	4.6
ResNet-50	76.31	719	25.53	3.9	2.8
RepVGG-B1	78.37	685	51.82	11.8	5.9
RegNetX-3.2GF	77.98	671	15.26	3.2	2.9
RepVGG-B2g4	78.50	581	55.77	11.3	6.0
ResNeXt-50	77.46	484	24.99	4.2	4.1
RepVGG-B2	78.78	460	80.31	18.4	9.1
ResNet-101	77.21	430	44.49	7.6	5.5
VGG-16	72.21	415	138.35	15.5	6.9
ResNet-152	77.78	297	60.11	11.3	8.1
ResNeXt-101	78.42	295	44.10	8.0	7.9

Table 2: Architectural specification of RepVGG. Here $2 \times 64a$ means stage2 has 2 layers each with $64a$ channels.

Stage	Output size	RepVGG-A	RepVGG-B
1	112×112	$1 \times \min(64, 64a)$	$1 \times \min(64, 64a)$
2	56×56	$2 \times 64a$	$4 \times 64a$
3	28×28	$4 \times 128a$	$6 \times 128a$
4	14×14	$14 \times 256a$	$16 \times 256a$
5	7×7	$1 \times 512b$	$1 \times 512b$

Table 3: RepVGG models defined by multipliers a and b.

Name	Layers of each stage	a	b
RepVGG-A0	1, 2, 4, 14, 1	0.75	2.5
RepVGG-A1	1, 2, 4, 14, 1	1	2.5
RepVGG-A2	1, 2, 4, 14, 1	1.5	2.75
RepVGG-B0	1, 4, 6, 16, 1	1	2.5
RepVGG-B1	1, 4, 6, 16, 1	2	4
RepVGG-B2	1, 4, 6, 16, 1	2.5	5
RepVGG-B3	1, 4, 6, 16, 1	3	5

- RepVGG models achieve high top-1 accuracy, comparable to or surpassing ResNet and EfficientNet variants
- Inference Speed: RepVGG models demonstrate significantly faster inference speeds than ResNet counterparts, benefiting from the single-branch structure
- RepVGG-A0 outperforms ResNet-18 in both accuracy (72.41% vs. 71.16%) and speed
- RepVGG-B3 reaches 80.52% accuracy, close to RegNetX-12GF, while being 31% faster

Table 5: Results on ImageNet trained in 200 epochs with Autoaugment [5], label smoothing and mixup.

Model	Acc	Speed	Params	FLOPs	MULs
RepVGG-B2g4	79.38	581	55.77	11.3	6.0
RepVGG-B3g4	80.21	464	75.62	16.1	8.4
RepVGG-B3	80.52	363	110.96	26.2	12.9
RegNetX-12GF	80.55	277	46.05	12.1	10.9
EfficientNet-B3	79.31	224	12.19	1.8	-

Experiment

- Significance of Re-parameterization
  - Re-parameterization enables the multi-branch model to be transformed into a single-branch model for efficient inference
  - Allows RepVGG to achieve both high accuracy and fast inference speed, without the complexity of multiple branches
- Performance Boost
  - Ablation studies show that removing the identity and 1x1 branches lowers model accuracy significantly
- Comparison with Alternatives
  - Structural re-parameterization outperforms simpler re-parameterization methods like DiracNet, which doesn't use real multi-branch data flow
  - RepVGG's structure enables superior accuracy-speed trade-offs compared to standard multi-branch models like ResNet

Table 6: Ablation studies with 120 epochs on RepVGG-B0. The inference speed w/o re-param (examples/s) is tested with the models before conversion (batch size=128). Note again that all the models have the same final structure.

Identity branch	1×1 branch	Accuracy	Inference speed w/o re-param
		72.39	1810
✓		74.79	1569
	✓	73.15	1230
✓	✓	**75.14**	1061

Thank You

Jaehoon Jang
Korea University