RepVGG: Making VGG-style ConvNets Great Again

Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, Jian Sun

Conference on Computer Vision and Pattern Recognition 2021

Last revised on arXiv on: 29 March 2021

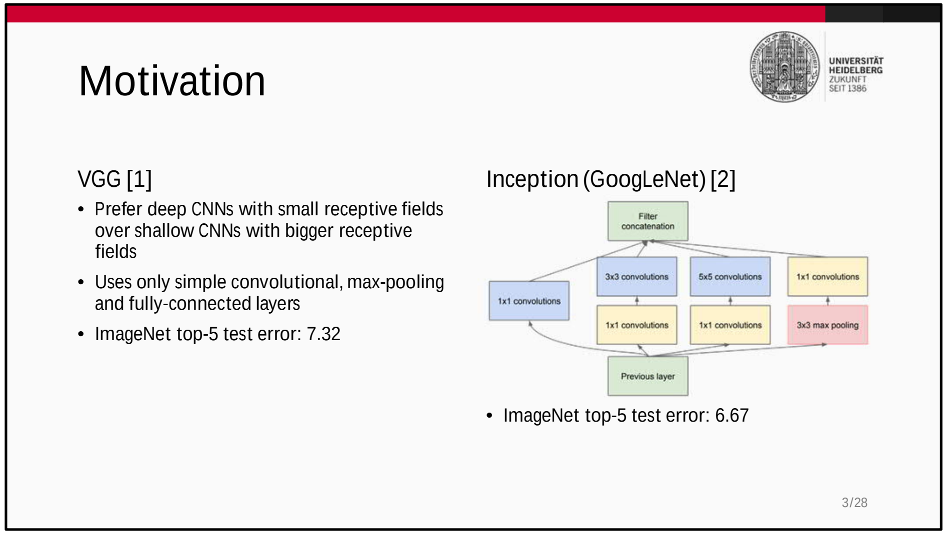
Repository: https://github.com/DingXiaoH/RepVGG

- Visual Geometry Group, Department of Engineering Science, University of Oxford
- This paper originates from Beijing, Hong Kong, UK

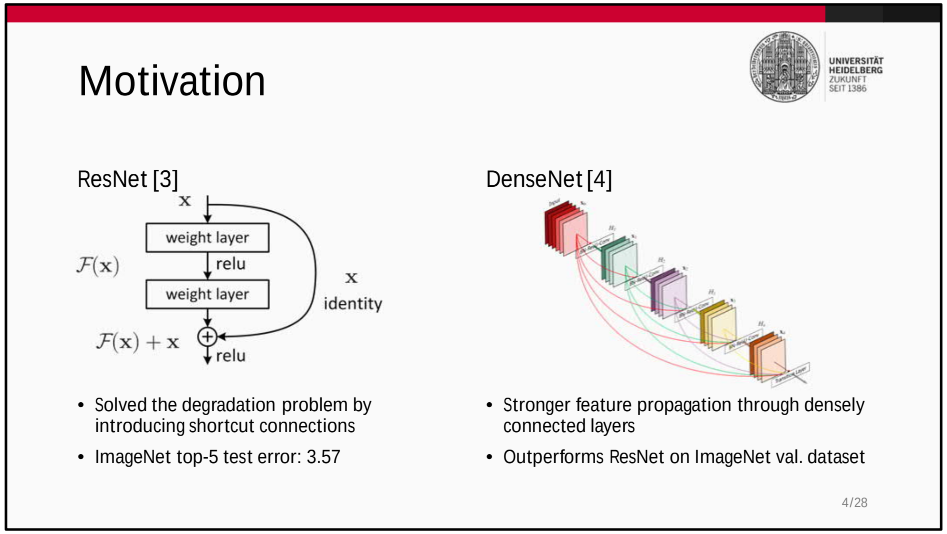
Agenda

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

1. Motivation
2. Fundamentals
3. Approach
4. Experiments
5. Highlights and Weaknesses
6. Conclusion



- VGG (2014):
  - Strenghten the discriminative character of the network as the non-linear activation function ReLU is applied more often
  - Number of parameters to train is lower
- Inception(2014):
  - Inception modules:
    - 1x1 convolutions for dimension reduction over the channel size
    - Detect cross-channel correlations
    - Better detect spatial correlations and objects at various scales, apply kernels of various scales simultaniously
    - Branch concatenation
    - Split-Transform-Merge

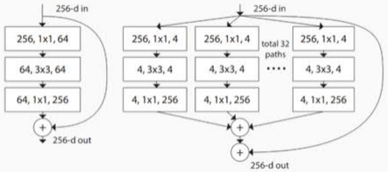


- ResNet (2015):
  - Degradation problem: Ideal mapping was learned up until a certain depth by shallow layers, remaining layers struggle to learn implicit identity mapping through several non-linearity steps =>Loss curve during training ascents again
  - Set weights of residual components to zero and realize an identity function
  - Early layers receive much more training signal (solves vanishing gradient problem)
  - Implicit ensemble of multiple smaller networks
  - Many paths through the network, but only 0.45% are valid ones (contribute much to the gradient), predominantly short paths
  - Still keeps complexity eight times lower than VGG for a single net
- DenseNet (2016)

Motivation

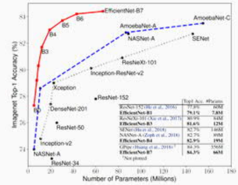
UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

ResNeXt [5]



- Cardinality dimension: size of the set of transformations
- Increase cardinality instead of depth or width

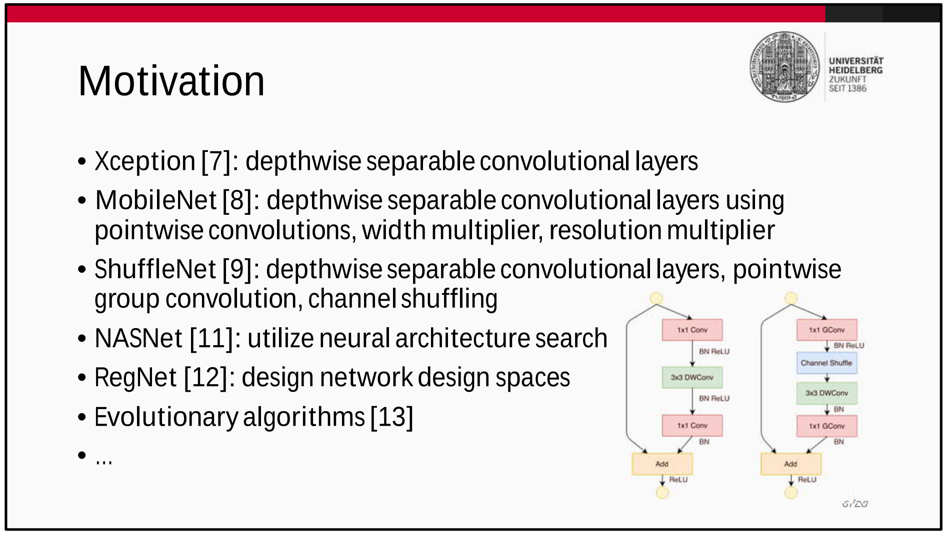
EfficientNet [6]



- Compound scaling method: uniform scaling in depth, width and resolution
- ImageNet top-1 accuracy: 84.3 (B7)

5/28

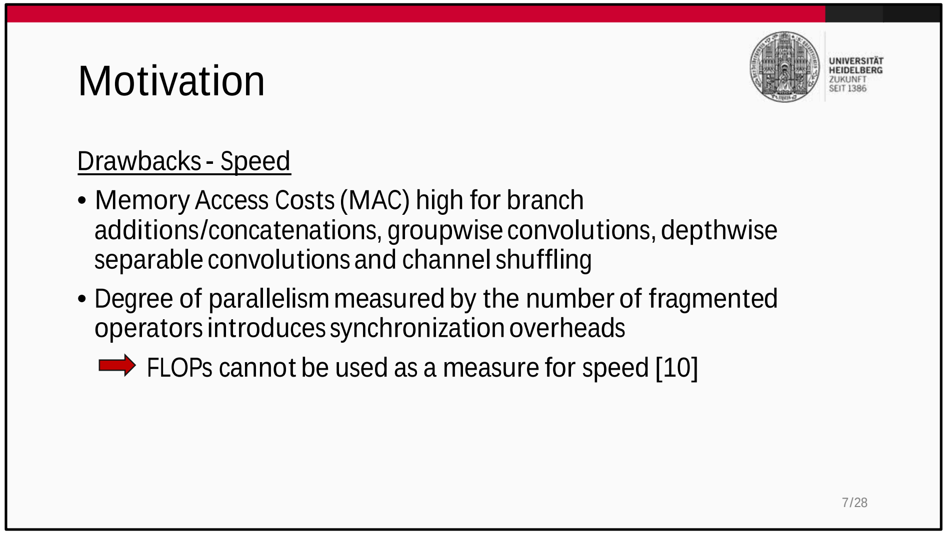
- ResNeXt (2016):
    - Much more efficient
- EfficientNet (2019):
    - Again fewer parameters to train



- Xception (2016):
  - Similar to inception modules, but first convolutes spatially and afterwards the channels using poitwise 1x1 convolution without using non-linearity in between
  - Maps spatial and cross-channel correlations completely separately
  - Stack several depthwise separable convolutional layers with residual connections
  - Only negligible improvements compared to Inception v3
- MobileNet (2017):
  - Use depthwise separable convolutional layers in their nature of data reduction (1x1 conv) to make CNNs accessible for mobile devices and embedded systems
  - Width (channels), resolution multiplier: achieve better accuracy-speed tradeoff (latency vs. size)
  - In terms of accuracy MobileNet can be compared to VGG16 while being 32 times smaller and 27 times less computationally expensive (measured by the Mult Adds)
- ShuffleNet (2017):
  - Possible information bottlenecks by the pointwise group convolution are tackled by using subsequent channel

shuffeling to keep the information flow entropy of channels the same

- NASNet (2017):
    - Run NAS for single convolutional cell on CIFAR-10, then stacked to whole CNN
    - NASNet is actually the design space
- RegNet (2020):
    - Search for suitable design spaces in order to derive a common understanding about important design principles



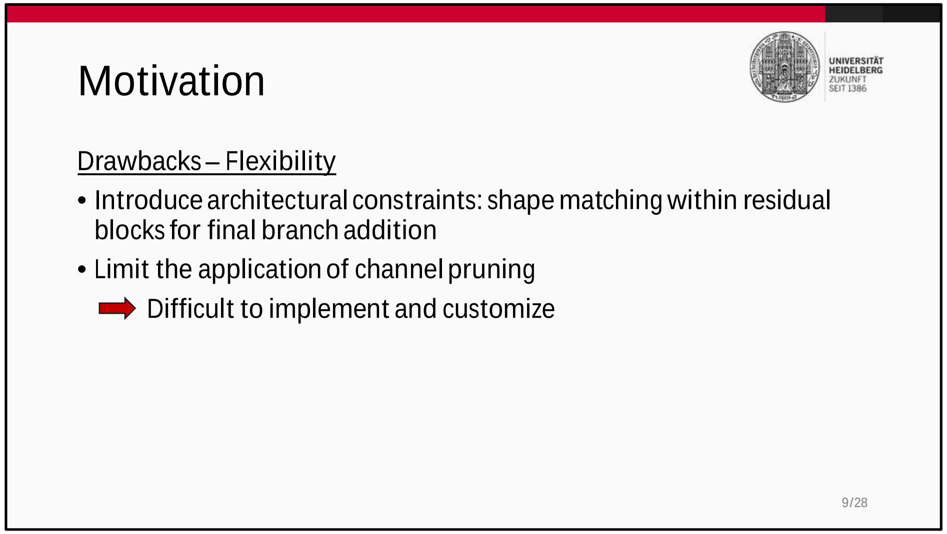
- Fragmented operators: number of individual convolution or pooling operations in one building block
- FLOPs cannot be used as a measure for speed: also because of I/O operations, optimized runtime target platforms and elementwise operations not considered
- NAS, designing design spaces: high performance networks, but very high computing costs and slow down degree of parallelism, not trainable on normal GPUs

Motivation

<u>Drawbacks – Memory Efficiency</u>

- Multi-branch topology like residual architectures keep results of every branch until addition/concatenation

  ➡ Less computing units to be integrated onto the chip



- Channel pruning: removal of unimportant channels by sparse parameter tensors, then drop unnecessary filters for better performance-efficiency tradeoff
- Automatic discovery of appropriate layer width cannot be applied

Motivation

Idea: RepVGG

- Multi-branch topology during training
- VGG-like plain inference-time architecture
- Transformation by structural re-parameterization
- Advantages:
  - VGG-like plain feed-forward topology using only 3x3 conv and ReLU
  - No automatic search, manual refinement, compound scaling or other heavy design methods
  - Fewer types of operators enable more computing units integrated onto the chip
  - Good accuracy-speed trade-off

10/28

- Training a plain CNN without shortcut connections is possible (references in paper), but certainly difficult (vanishing gradient problem)
- Training-time RepVGG: indentity, 1x1 conv and 3x3 conv branches (inspired by ResNet)
- Identity = degraded 1x1 conv, 1x1 conv = degraded 3x3 conv
- 3x3 conv constructed from trained parameters form 3x3 conv, 1x1 conv, identity and batch normalization => used for test and deployment
- Chips specialized for RepVGG can have an enormous number of 3x3-ReLU units and fewer memory units (as being memory economical) => higher speed

Fundamentals

Winograd's minimal filtering algorithm [14]:

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}$$

$$m_1 = (d_0 - d_2) g_0 \qquad m_2 = (d_1 + d_2) \frac{g_0 + g_1 + g_2}{2}$$

$$m_4 = (d_1 - d_3) g_2 \qquad m_3 = (d_2 - d_1) \frac{g_0 - g_1 + g_2}{2}$$

11/28

- 4 multiplications instead of using 6 multiplications as originally
- 4 additions involving the data
- 3 additions and 2 multiplications by a constant involving the filter => can be done in preprocessing
- 4 additions to reduce the products to the final result

Fundamentals

Winograd's minimal filtering algorithm [14]: $\boxed{Y = A^T\left[(Gg) \odot (B^T d)\right]}$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix} \qquad G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$g = \begin{bmatrix} g_0 & g_1 & g_2 \end{bmatrix}^T$$

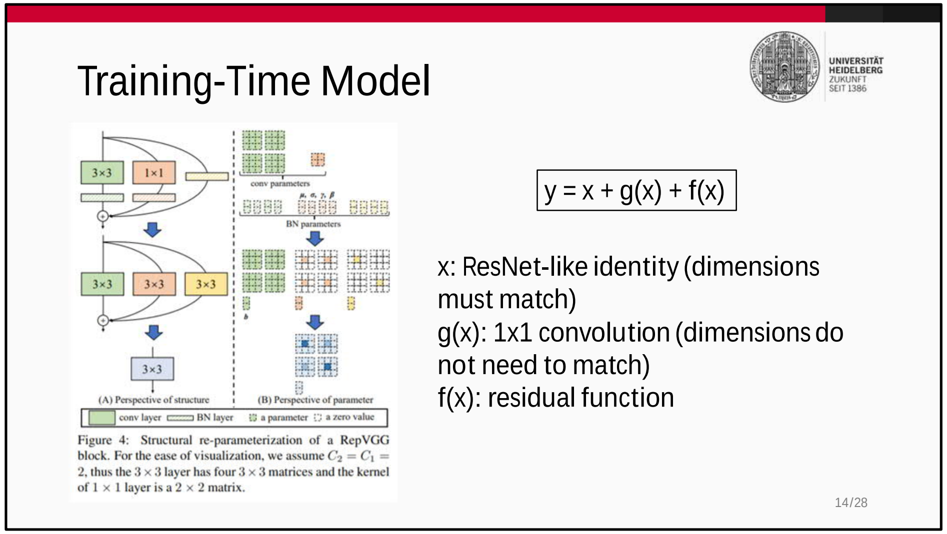
$$d = \begin{bmatrix} d_0 & d_1 & d_2 & d_3 \end{bmatrix}^T$$

Fundamentals

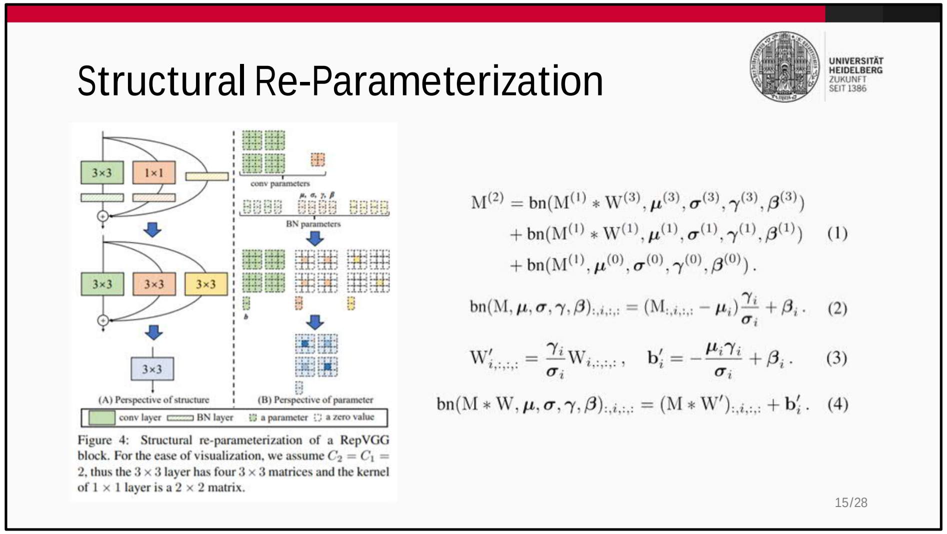
<u>Winograd's minimal filtering algorithm [14]:</u>

$$Y = A^T \left[[GgG^T] \odot [B^T dB] \right] A$$

- F(2x2,3x3) uses 16 mults, whereas normal convolution takes 36 mults
- Speedup by factor 2.25
- 3x3 concolutions highly optimized by modern computing libraries like cuDNN (by using Winograd's algorithm)



- Here: Would have the mentioned drawbacks during inference, therefore only training time model
- Inspired by ResNet: multi-branch architecture makes the model an implicit ensemble of numerous shallower models (n block => 2^n models)
- Stack several x + g(x) + f(x) blocks to an ensemble (3^n models)
- Apply batch normalization before the addition (output channel wise)
- Omit identity branch in case the channel dimensions do not match



- Convert every BN and its preceding convolutional layer into a convolutional layer with a bias vector
- W‘, B‘: new kernel and bias vector
- Identity: 1x1 conv with identity matrix as kernel
- Add up three bias vectors, zero-pad the 1x1 conv to 3x3 and add all three 3x3 together
- But: all layers need to have the same stride, padding configuration of 1x1 one pixel less than 3x3

Architectures

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	$1 \times 512b$	$1 \times 512b$

- Interleave groupwise 3x3 convolution layer every two layers
  - trade accuracy for efficiency but still maintain inter-channel information exchange
  - Group factor: 1,2 or 4

16/28

- VGG-style: adopts plain topology and heavily uses 3x3 conv, no max pooling like VGG as body should only have one type of operator

- 1st layer of each stage downsamples with stride 2
- First stage has only one layer: lower latency as operates with high resolutions
- Avoid large scale conv in first layer
- Last stage should have more channels => use only one layer to save the parameters
- Most layers in stage 4 following ResNet and its variants
- Groupwise conv every two layers to ensure inter-channel exchange: otherwise channel output only derived from fraction of input channels
- Task specific heads

- RepVGG-A vs lightweight/middleweight models (ResNet-18/34/50)
- RepVGG-B vs high performance models

- Width setting from VGG and ResNet (uniform scaling)
- Sacling factors b>a (richer features for the classification or other downstream tasks)
- As last stage has only one layer: large b does not introduce much latency or far more amount of parameters

Experiments

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

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

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	-

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.

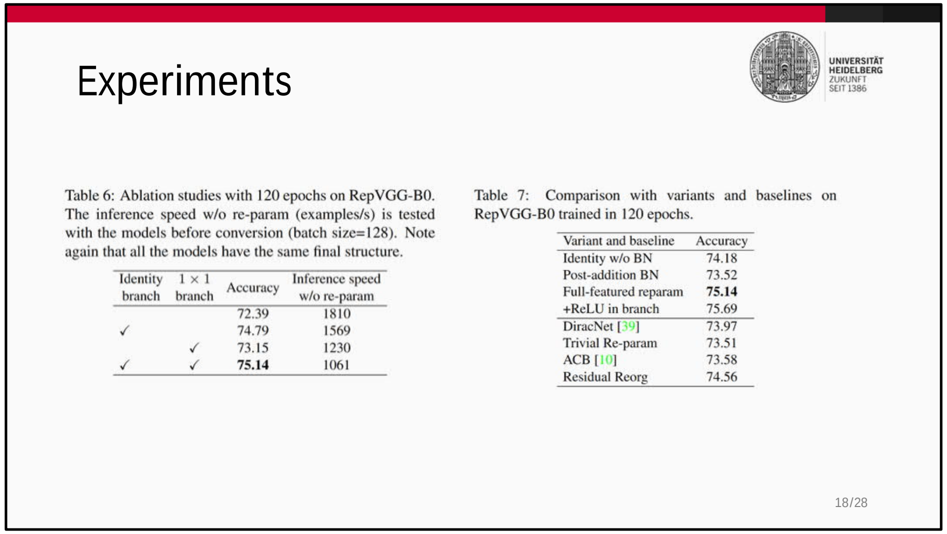
Model	Top-1 acc	Speed	Params (M)	Theo FLOPs (B)	Wino MULs (B)
RepVGG-A0	72.41	3256	8.30	1.4	0.7
ResNet-18	71.16	2442	11.68	1.8	1.0
RepVGG-A1	74.46	2339	12.78	2.4	1.3
RepVGG-B0	75.14	1817	14.33	3.1	1.6
ResNet-34	74.17	1419	21.78	3.7	1.8
RepVGG-A2	76.48	1322	25.49	5.1	2.7
RepVGG-B1g4	77.58	868	36.12	7.3	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

17/28

- RepVGG models with interleaved groupwise layers: postfix g2/g4
- Training the leight/middleweight models: random cropping, left-right flipping, B=256, 8 GPUs, lr=0.1, cosine annealing=120, SGD, momemtum=0.9, weight decay^=10^-4
- Training the heavyweight models: 5-epoch warmup, cosine-annealing: 200, label smoothing, mixup, random cropping, flipping, data augmentation pipeline
- All models tested on the same GPU
- All conv-BN sequences of the baselines are also converted into a conv with bias
- Comparison against EfficientNet-B0/B3 (middleweight) and RegNet-3.2GF/12GF (heavyweight)

- RepVGG-A models are more accurate and faster than their ResNet competitors
- Interleaved groupwise layer models: reasonable accuracy decrease
- Impressive improvements in speed (RepVGG-B1g4 is 101% faster than ResNet-101, RepVGG-B1g2 is 2.66 times faster than ResNet-152 (while having the same accuracy))
- More parameter efficient: RepVGG-B2 has only 58% parameters compared to VGG-16, 10% faster, 6.57% higher accuracy

- Can even hold up with RegNetX-12GF and even runs 31% faster! (without designing network design spaces, architectural hyper-parameters are set casually)
- Over 80% accuracy with plain models reached (for the first time)
- FLOPS vs Winograd MULTs: VGG-16 vs ResNet-152 -> Proofs thesis!



Experiments

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

Table 7: Comparison with variants and baselines on RepVGG-B0 trained in 120 epochs.

Variant and baseline	Accuracy
Identity w/o BN	74.18
Post-addition BN	73.52
Full-featured reparam	**75.14**
+ReLU in branch	75.69
DiracNet [39]	73.97
Trivial Re-param	73.51
ACB [10]	73.58
Residual Reorg	74.56

18/28

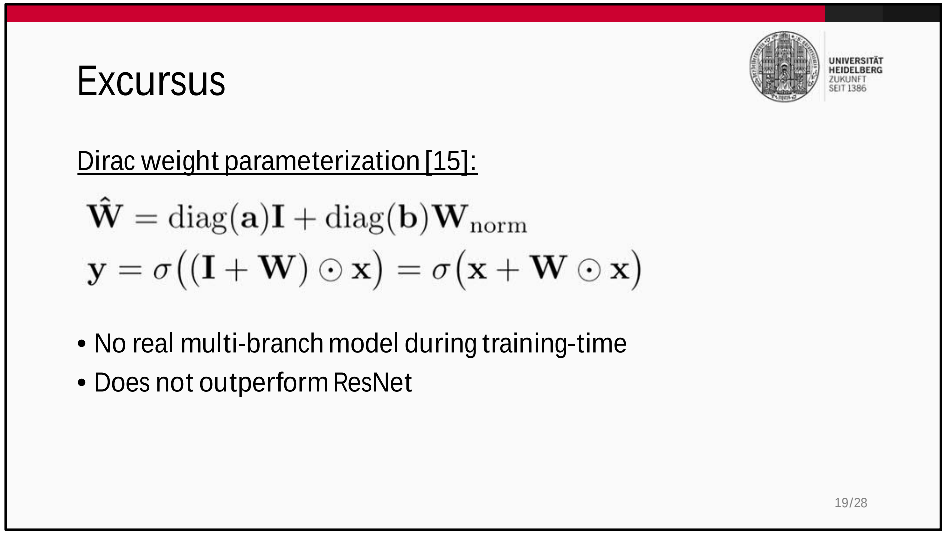
Strutural Re-parameterization is key!:

- Full featured RepVGG-B0 is 75.14%, 2.75% higher than ordinary plain model (without 1x1 conv and identity)

- ReLU after BN and before addition: Such a block cannot be converted into a single block anymore!
- Trivial Re-param: Trivial DiracNet: W' = I + W
- ACB: Do improvements come from component-level over-parameterization? 3x3,3x1,1x3 kernels added back together

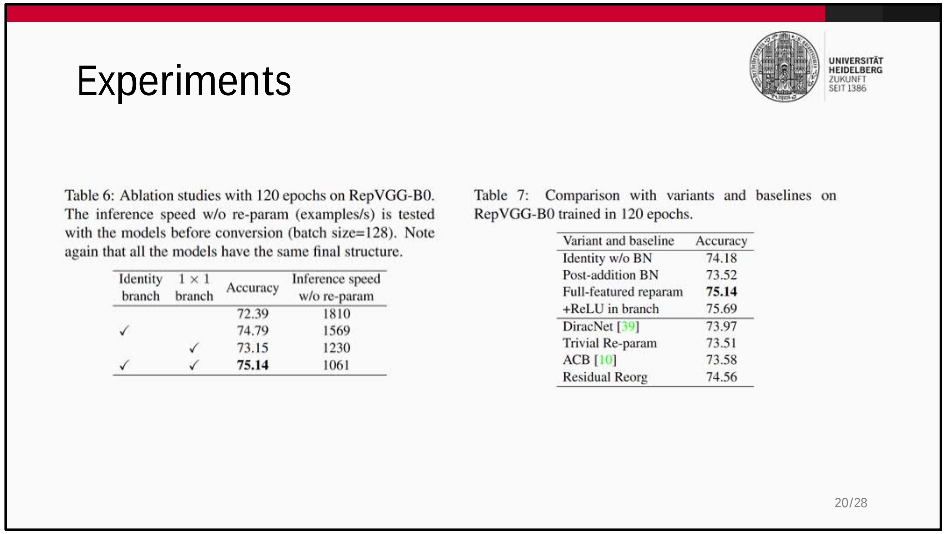
- RepVGG: concrete structure with nonlinear behavior, DiracNet uses another mathematical expression of conv kernels ("using the params of a structure to parameterize another structure" vs. "computing the params first with another set of params, then using them for other computations")
- RepVGG does not perform well only because of over parameterization (even better than ACB, which has even more parameters, ResNet-50 gives same performance with RepVGG-blocks) => methodology critical to train plain networks
- Residual Reorg: same amount of 3x3 conv + additional shortcuts: RegVGG still outperforms because of more branches (bigger ensemble)

- Similar results in semantic segmentation on Cityscapes: minor accuracy increases, but much faster (Exchange ResNet-50/101 backbone with RepVGG-B1g2/B2)



- Previous work:
  - Make plain models converge but do not outperform multi-branch models
  - New theoretical initialization method, LReLU, max-norm and careful initialization
  - This paper: simple model with reasonable depth and favorable accuracy-speed trade-off, which can be simply implemented

- Model Re-parameterization:
  - DiracNet:
    - a and b are scaling vectors learned during training, W_norm is the normalized weight matrix
    - achieve deep network performances close to residual networks without actual skip-connections
    - DiracNet was able to closely match 1001-layer ResNet with only 28 layers on CIFAR-10 as well as ResNet-18 and ResNet-34 on ImageNet wile having the same amount of parameters (27.79% vs. 27.17% top-1 error with 34-layer depth configuration)
    - Similar to actual ResNet layer: only differ in the order of non-linearities



Experiments

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

Table 7: Comparison with variants and baselines on RepVGG-B0 trained in 120 epochs.

Variant and baseline	Accuracy
Identity w/o BN	74.18
Post-addition BN	73.52
Full-featured reparam	**75.14**
+ReLU in branch	75.69
DiracNet [39]	73.97
Trivial Re-param	73.51
ACB [10]	73.58
Residual Reorg	74.56

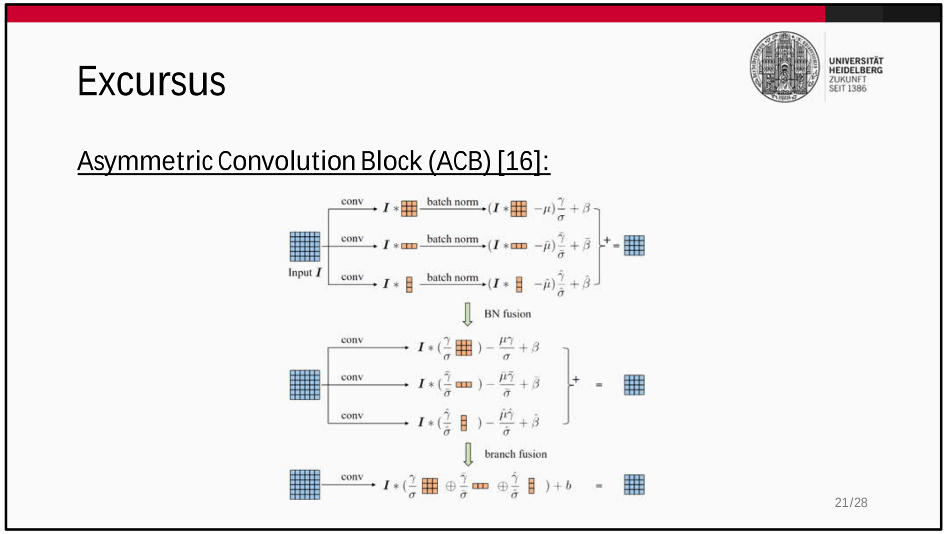
Strutural Re-parameterization is key!:

- Full featured RepVGG-B0 is 75.14%, 2.75% higher than ordinary plain model (without 1x1 conv and identity)

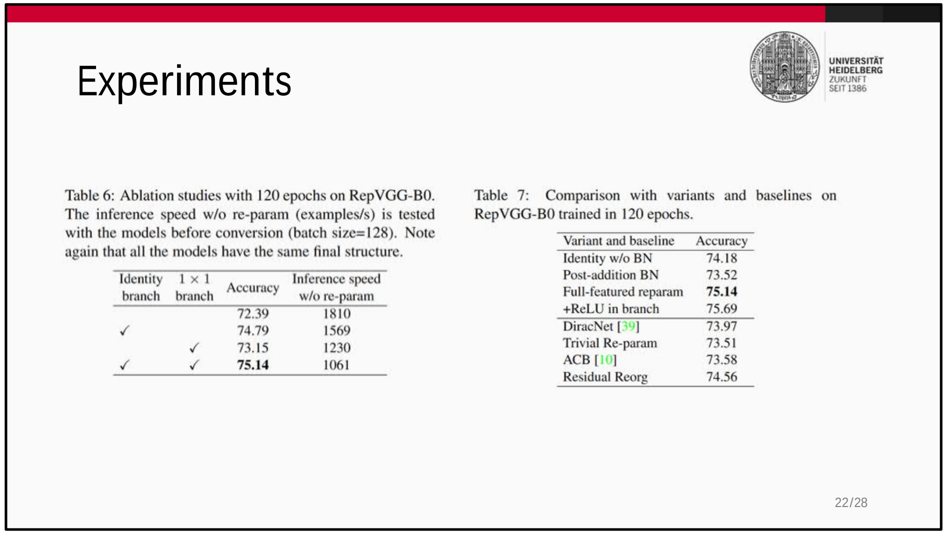
- ReLU after BN and before addition: Such a block cannot be converted into a single block anymore!
- Trivial Re-param: Trivial DiracNet: W' = I + W
- ACB: Do improvements come from component-level over-parameterization? 3x3,3x1,1x3 kernels added back together

- RepVGG: concrete structure with nonlinear behavior, DiracNet uses another mathematical expression of conv kernels (“using the params of a structure to parameterize another structure” vs. “computing the params first with another set of params, then using them for other computations”)
- RepVGG does not perform well only because of over parameterization (even better than ACB, which has even more parameters, ResNet-50 gives same performance with RepVGG-blocks) => methodology critical to train plain networks
- Residual Reorg: same amount of 3x3 conv + additional shortcuts: RegVGG still outperforms because of more branches (bigger ensemble)

- Similar results in semantic segmentation on Cityscapes: minor accuracy increases, but much faster (Exchange ResNet-50/101 backbone with RepVGG-B1g2/B2)



- Model Re-parameterization:
  - ACNet:
    - During training time a normal squared 3x3 convolutional kernel is replaced by multi-branch 3x3, 3x1 and 1x3 kernels that are added back together after batch normalization
    - ACBs are architecture-neutral meaning they can replace normal 3x3 conv layers without having additional hyperparameters to tune, without further assumptions to take about the model and without additional computational complexity induced
    - Strengthens the skeletons of squared convolutional kernels, but in practice only leads to few but consistent performance improvements
  - DO-Conv (depthwise over-parameterized) layers
  - ExpandNet: additional consecutive linear layers without further non-linearity in between
  - All architectures can also be folded back into the same structure as the original for the inference time (can be used as drop-in replacement, component level improvement)



Experiments

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

Table 7: Comparison with variants and baselines on RepVGG-B0 trained in 120 epochs.

Variant and baseline	Accuracy
Identity w/o BN	74.18
Post-addition BN	73.52
Full-featured reparam	**75.14**
+ReLU in branch	75.69
DiracNet [39]	73.97
Trivial Re-param	73.51
ACB [10]	73.58
Residual Reorg	74.56

22/28

Strutural Re-parameterization is key!:

- Full featured RepVGG-B0 is 75.14%, 2.75% higher than ordinary plain model (without 1x1 conv and identity)

- ReLU after BN and before addition: Such a block cannot be converted into a single block anymore!
- Trivial Re-param: Trivial DiracNet: W’ = I + W
- ACB: Do improvements come from component-level over-parameterization? 3x3,3x1,1x3 kernels added back together

- RepVGG: concrete structure with nonlinear behavior, DiracNet uses another mathematical expression of conv kernels (“using the params of a structure to parameterize another structure” vs. “computing the params first with another set of params, then using them for other computations”)
- RepVGG does not perform well only because of over parameterization (even better than ACB, which has even more parameters, ResNet-50 gives same performance with RepVGG-blocks) => methodology critical to train plain networks
- Residual Reorg: same amount of 3x3 conv + additional shortcuts: RegVGG still outperforms because of more branches (bigger ensemble)

- Similar results in semantic segmentation on Cityscapes: minor accuracy increases, but much faster (Exchange ResNet-50/101 backbone with RepVGG-B1g2/B2)

Highlights and Weaknesses

<u>Highlights:</u>

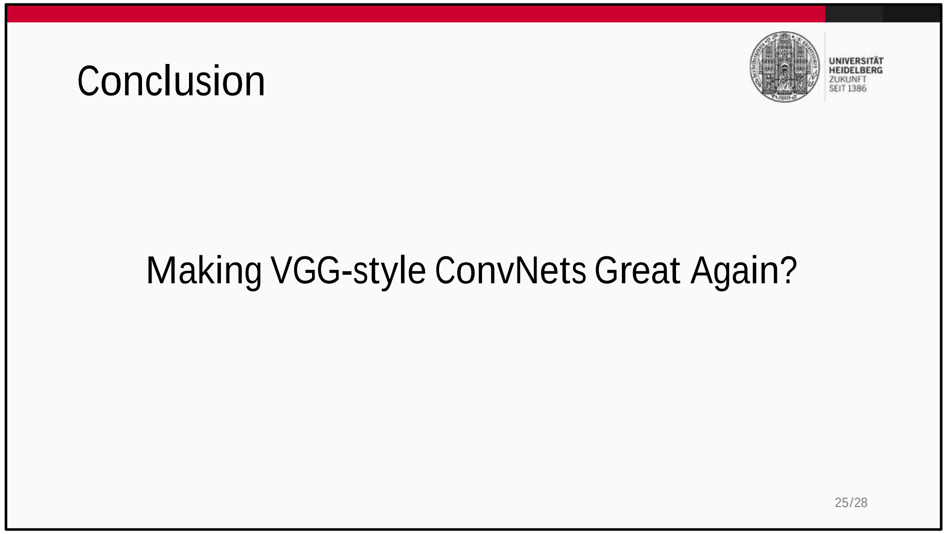
- Proof that plain ConvNets can outperform ResNet-like architectures
- Novel re-parameterization technique based on parameters of a structure to parameterize another structure
- Accuracy-speed trade-off
- Optimized for GPUs and other specialized hardware
- Simple to implement

Highlights and Weaknesses

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Weaknesses:

- Parameter inefficient compared to „modern" architectures like EfficientNet or RegNetX (less favored than mobile-regime MobileNets and ShuffleNets for low-power devices)
- Models they compare with are biased for better positioning
- Additional restrictions: 3x3 convolutions, equal stride, padding



- Own thoughts:
  - In terms of introducing a simple to implement model with a good accuracy-speed tradeoff very well done
  - Gives another point of view into image classification networks (hardware support, re-parameterization techniques, training-time/inference-time separation)
  - As it is clearly positioned and did not have the intention to provide another state-of-the-art, it is a very important contribution into this specific field of research
  - „Great again“: In the end it convinces with its speed gain instead of its accuarcy gain!

Bibliography

[1] Andrew Zisserman, Karen Simonyan. „Very Deep Convolutional Networks for Large-Scale Image Recognition", 2014. https://arxiv.org/pdf/1409.1556.pdf.

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. „Going Deeper with Convolutions", 2014. https://arxiv.org/pdf/1409.4842.pdf.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. „Deep Residual Learning for Image Recognition", 2015. https://arxiv.org/pdf/1512.03385.pdf.

[4] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. „Densely Connected Convolutional Networks", 2016. https://arxiv.org/pdf/1608.06993.pdf.

[5] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He. „Aggregated Residual Transformations for Deep Neural Networks", 2016. https://arxiv.org/pdf/1611.05431.pdf.

[6] Quoc V. Le Mingxing Tan. „EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", 2019. https://arxiv.org/pdf/1905.11946.pdf.

Bibliography

[7] Franc¸ois Chollet. „Xception: Deep Learning with Depthwise Separable Convolutions", 2016. https://arxiv.org/pdf/1610.02357.pdf.

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017. https://arxiv.org/pdf/1704.04861.pdf.

[9] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, Jian Sun. „ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices", 2017. https://arxiv.org/pdf/1707.01083.pdf.

[10] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, Jian Sun. „ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design", 2018. https://arxiv.org/pdf/1807.11164.pdf.

[11] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le. „Learning transferable architectures for scalable image recognition", 2017. https://arxiv.org/pdf/1707.07012.pdf.

[12] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, Piotr Dollar. „Designing network design spaces", 2020. https://arxiv.org/pdf/2003.13678.pdf.

Bibliography

[13] Esteban Real, Alok Aggarwal, Yanping Huang, Quoc V Le. „Regularized evolution for image classifier architecture search", 2018. https://arxiv.org/pdf/1802.01548.pdf.

[14] Andrew Lavin, Scott Gray. „Fast Algorithms for Convolutional Neural Networks", 2015. https://arxiv.org/pdf/1509.09308.pdf.

[15] Nikos Komodakis Sergey Zagoruyko. „Diracnets: Training very deep neural networks without skip-connections", 2017. https://arxiv.org/pdf/1706.00388.pdf.

[16] Xiaohan Ding, Yuchen Guo, Guiguang Ding, Jungong Han. „Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks", 2019. https://arxiv.org/pdf/1908.03930.pdf.