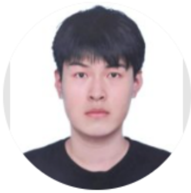
VanillaNet: the Power of Minimalism in Deep Learning

Paper Reading by Zhiying Lu

2023.06.05

智能多媒体内容计算实验室
Intelligent Multimedia Content Computing Lab

- 作者介绍
- 研究背景
- 方法
- 实验效果
- 总结



Hanting Chen

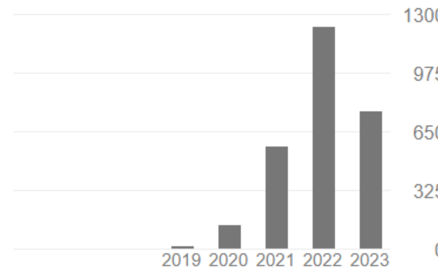
FOLLOW

Huawei Noah's Ark Lab
Verified email at pku.edu.cn
deep learning machine learning computer vision

TITLE	CITED BY	YEAR
Pre-trained image processing transformer H Chen, Y Wang, T Guo, C Xu, Y Deng, Z Liu, S Ma, C Xu, C Xu, W Gao Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern …	903	2021
A survey on vision transformer K Han, Y Wang, H Chen, X Chen, J Guo, Z Liu, Y Tang, A Xiao, C Xu, … IEEE transactions on pattern analysis and machine intelligence 45 (1), 87-110	804 *	2022
Data-Free Learning of Student Networks H Chen, Y Wang, C Xu, Z Yang, C Liu, B Shi, C Xu, C Xu, Q Tian Proceedings of the IEEE International Conference on Computer Vision, 3514-3522	262	2019
AdderNet: Do we really need multiplications in deep learning? H Chen, Y Proceedin	173	2020

Cited by

	All	Since 2018
Citations	2715	2712
h-index	14	14
i10-index	18	18



Public access VIEW ALL

12 articles

Yunhe Wang (王云鹤)

FOLLOW

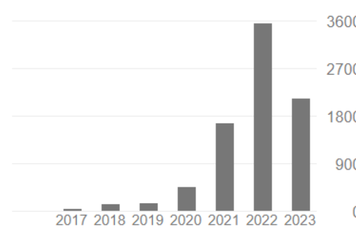
Other names ▸

Senior Researcher, Huawei Noah's Ark Lab, Huawei Technologies
Verified email at huawei.com - Homepage
Deep learning Computer vision Machine learning

TITLE	CITED BY	YEAR
GhostNet: More features from cheap operations K Han, Y Wang, Q Tian, J Guo, C Xu, C Xu CVPR, 1580-1589	1429	2020
Pre-trained image processing transformer H Chen, Y Wang, T Guo, C Xu, Y Deng, Z Liu, S Ma, C Xu, C Xu, W Gao CVPR, 12299-12310	903	2021
A survey on vision transformer K Han, Y Wang, H Chen, X Chen, J Guo, Z Liu, Y Tang, A Xiao, C Xu, … IEEE transactions on pattern analysis and machine intelligence 45 (1), 87-110	804 *	2022
Transformer in transformer K Han, A Xiao, E Wu, J Guo, C Xu, Y Wang Advances in Neural Information Processing Systems 34, 15908-15919	717	2021
Data-Free Learning of Student Networks H Chen, Y Wang, C Xu, Z Yang, C Liu, B Shi, C Xu, C Xu, Q Tian ICCV, 3514-3522	262	2019

Cited by

	All	Since 2018
Citations	8103	8052
h-index	42	42
i10-index	83	83



Public access VIEW ALL

0 articles 65 articles

not available available

Based on funding mandates

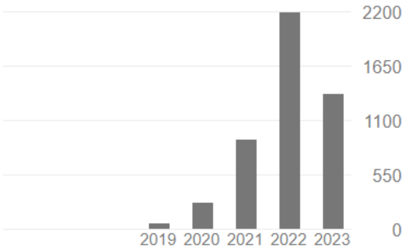
Jianyuan Guo

University of Sydney
Verified email at uni.sydney.edu.au - Homepage

FOLLOW

Cited by

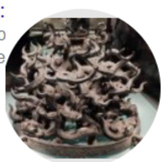
	All	Since 2018
Citations	4810	4808
h-index	18	18
i10-index	21	21



Public access VIEW ALL

1 article — 15 articles
not available — available
Based on funding mandates

TITLE	CITED BY	YEAR
Ghostnet: More features from cheap operations K Han, Y Wang, Q Tian, J Guo, C Xu, C Xu Proceedings of the IEEE/CVF conference on computer vision and pattern ...	1429	2020
Transformer in transformer K Han, A Xiao, E Wu, J Guo, C Xu, Y Wang Advances in Neural Information Processing Systems 34, 15908-15919	717	2021
Ocnet: Object context network for scene parsing Y Yuan, L Huang, J Guo, C Zhang, X Chen, J Wang arXiv preprint arXiv:1809.00916	538	2018
A survey on vision transformer K Han, Y Wang, H Chen, X Chen, J Guo, Z Liu, Y Tang, A Xiao, C Xu, ... IEEE transactions on pattern analysis and machine intelligence 45 (1), 87-110	515	2022
A survey on visual transformer K Han, Y Wang, H Chen, X Chen, J Guo, Z Liu, Y Tang, A Xiao, C Xu, ... arXiv preprint arXiv:2012.12556 2 (4)	306	2020
Cmt: J Guo Proce		



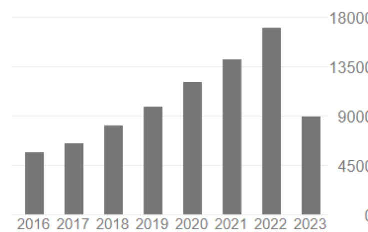
Dacheng Tao

The University of Sydney
在 sydney.edu.au 的电子邮件经过验证 - 首页
artificial intelligence machine learning computer vision image processing data mining

关注

引用次数 查看全部

	总计	2018 年至今
引用	98026	70230
h 指数	152	128
i10 指数	940	814



开放获取的出版物数量 查看全部

182 篇文章 — 483 篇文章
无法查看的文章 — 可查看的文章

标题	引用次数	年份
Dehazenet: An end-to-end system for single image haze removal B Cai, X Xu, K Jia, C Qing, D Tao IEEE Transactions on Image Processing 25 (11), 5187-5198	2152	2016
The visual object tracking vot2015 challenge results M Kristan, J Matas, A Leonardis, M Felsberg, L Cehovin, G Fernandez, ... Proceedings of the IEEE international conference on computer vision ...	2030	2015
Deep ordinal regression network for monocular depth estimation H Fu, M Gong, C Wang, K Batmanghelich, D Tao Proceedings of the IEEE conference on computer vision and pattern ...	1435	2018
Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge S Bakas, M Reyes, A Jakab, S Bauer, M Rempfler, A Crimi, RT Shinohara, ... arXiv preprint arXiv:1811.02629	1373	2018

- 作者介绍
- 研究背景
- 方法
- 实验效果
- 总结

研究背景

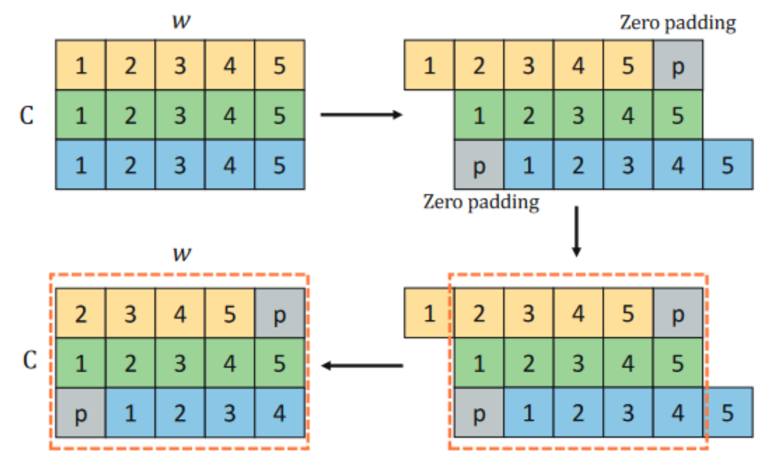
- CNN和ViT作为模型的backbone部分，承担了基础视觉特征提取任务
- 同时，嵌入式AI芯片逐渐成为主流
- AlexNet、ResNet和ViT，是视觉网络设计的里程碑，提供了网络设计的范式
- 后续提出的网络包含了大量人工设计的模块，在增加网络复杂度的同时，使网络具有更强的表征能力

研究背景

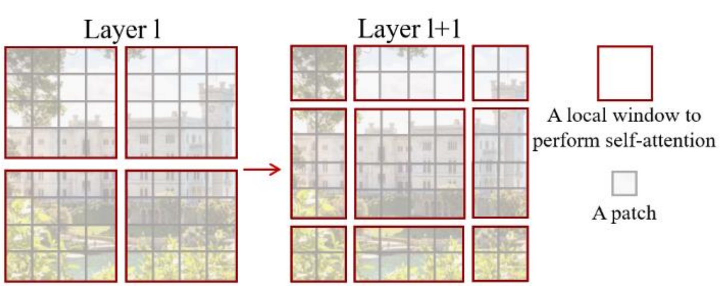
- 网络的复杂度虽然能提升表征能力，但也造成了实际部署的困难
- 例如ResNet中的shortcut就会消耗大量的off-chip memory traffic，因为它进行了多层特征的融合
- SwinTrans中的window shift和AS-MLP中的axial shift操作需要大量工程实现，包括重写CUDA等

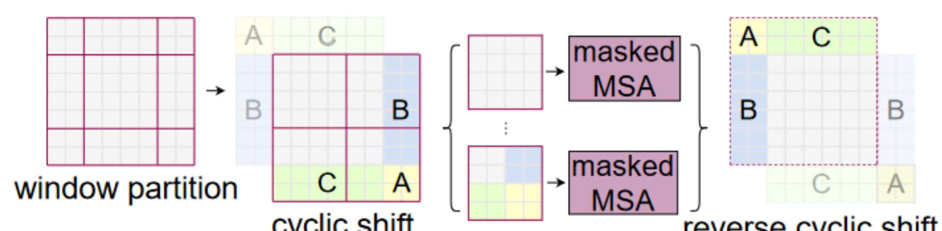
Table 1: Max memory (KB) on the mobile NPU

Input size	256	384	512
ResNet50	438737	832733	OOM
plain-CNN 50	356657	669397	1106989
Reduction	18.7%	19.6%	-



Horizontal shift

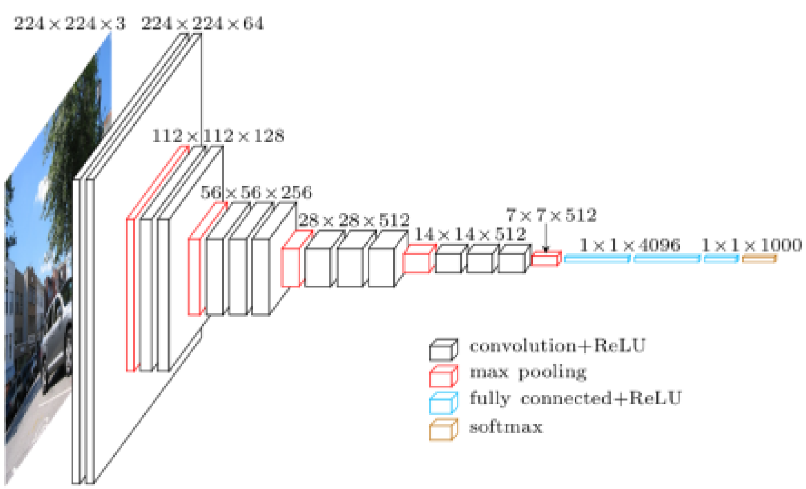


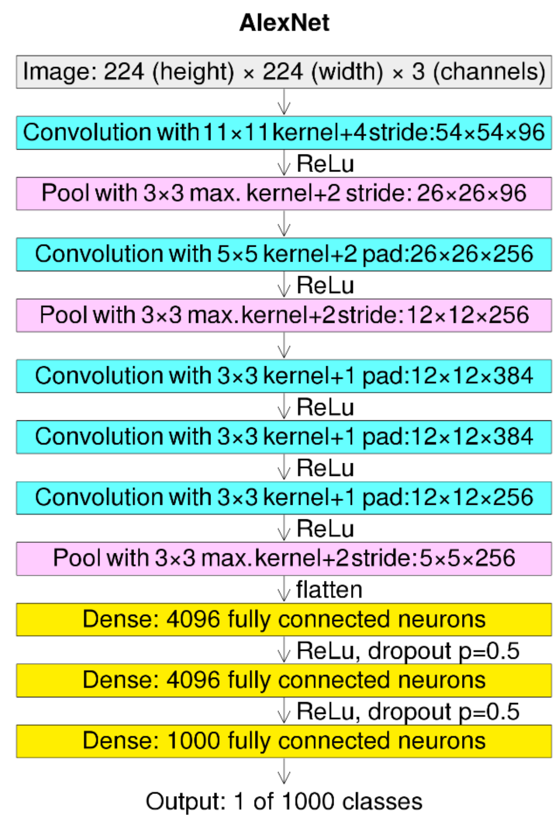


研究背景

- 因此将网络范式设计精简化很重要
- 类似ResNet网络的设计就偏离了精简化，但是不加shortcut会导致梯度消失问题，单纯增加卷积层的深度提升不如预期
- 简单网络如AlexNet和VGG的设计和优化受到的关注不多，因此该研究点具有较大的价值

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**



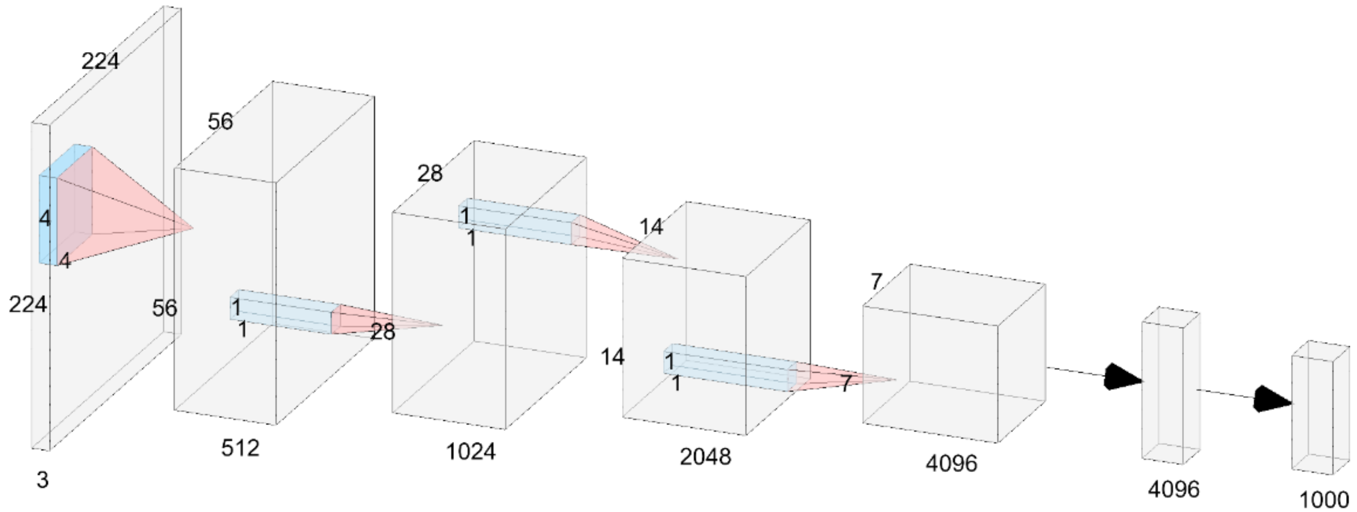


研究背景

- 随着AI芯片的发展，网络推理速度的瓶颈不再来自于FLOPs和参数量，因为GPU可以进行并行计算
- 复杂的结构和网络层数的深度在更大程度上限制了网络的推理速度
- 因此本文在设计时，采用极简的网络层数设计，抛弃复杂的操作如shortcut和attention，以极少的卷积层数和更快地推理速度，达到SOTA水平

- 作者介绍
- 研究背景
- 方法
- 实验效果
- 总结

VanillaNet



- 网络遵循一贯的四个stage设计，但是每个stage只包含一层卷积
- 跨stage采用2x2的maxpooling
- 每个卷积层不含batchnorm，不含shortcut
- 每个卷积层都是1x1卷积，激活函数放在卷积后
- 如何训练这样的网络使其达到SOTA效果？

Deep Training Strategy

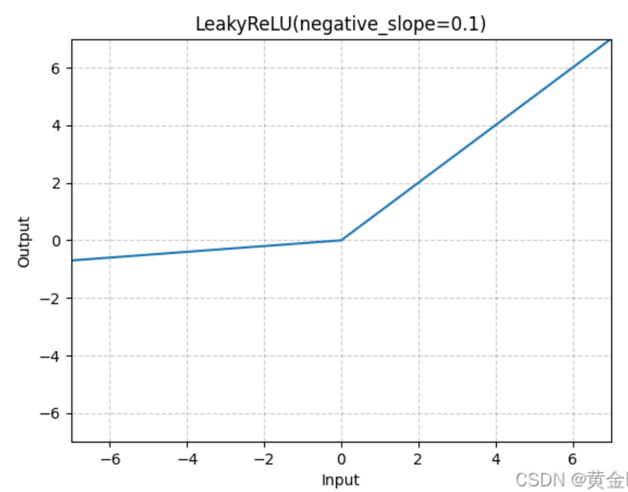
- 在网络训练的初期，通过训练两个卷积层，中间带激活函数
- 训练完成后，将二者参数融合成为单个卷积层，增加推理速度
- 激活函数会随着网络训练过程，逐步退化为identity mapping

$$A'(x) = (1-\lambda)A(x) + \lambda x, \qquad \lambda = \frac{e}{E}.$$

```python
def forward(self, x):
    if self.deploy:
        x = self.conv(x)
    else:
        x = self.conv1(x)
        x = torch.nn.functional.leaky_relu(x,self.act_learn)
        x = self.conv2(x)

    x = self.pool(x)
    x = self.act(x)
    return x
```

$$\text{LeakyRELU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$



Deep Training Strategy

- 网络在训练时，结构为：conv1-bn1-relu-conv2-bn2
- 首先将relu消除得到：conv1-bn1-conv2-bn2 $W_i' = \frac{\gamma_i}{\sigma_i} W_i, B_i' = \frac{(B_i - \mu_i)\gamma_i}{\sigma_i} + \beta_i,$
- 再通过重参数化，将BN参数融入conv1中得到：conv1-conv2
- 最后融合两个1x1conv的参数，得到最终的单层conv

$$y = W^1 * (W^2 * x) = W^1 \cdot W^2 \cdot \mathrm{im2col}(x) = (W^1 \cdot W^2) * X,$$

- 卷积层与BN层合并的操作如下：

卷积层公式为

$$Conv(x) = W(x) + b$$

而BN层公式为

$$BN(x) = \gamma * \frac{(x - mean)}{\sqrt{var}} + \beta$$

然后我们将卷积层结果带入到BN公式中

$$BN(Conv(x)) = \gamma * \frac{W(x) + b - mean}{\sqrt{var}} + \beta$$

进一步化简为

$$BN(Conv(x)) = \frac{\gamma * W(x)}{\sqrt{var}} + (\frac{\gamma * (b - mean)}{\sqrt{var}} + \beta)$$

这其实就是一个卷积层，只不过权重考虑了BN的参数 我们令：

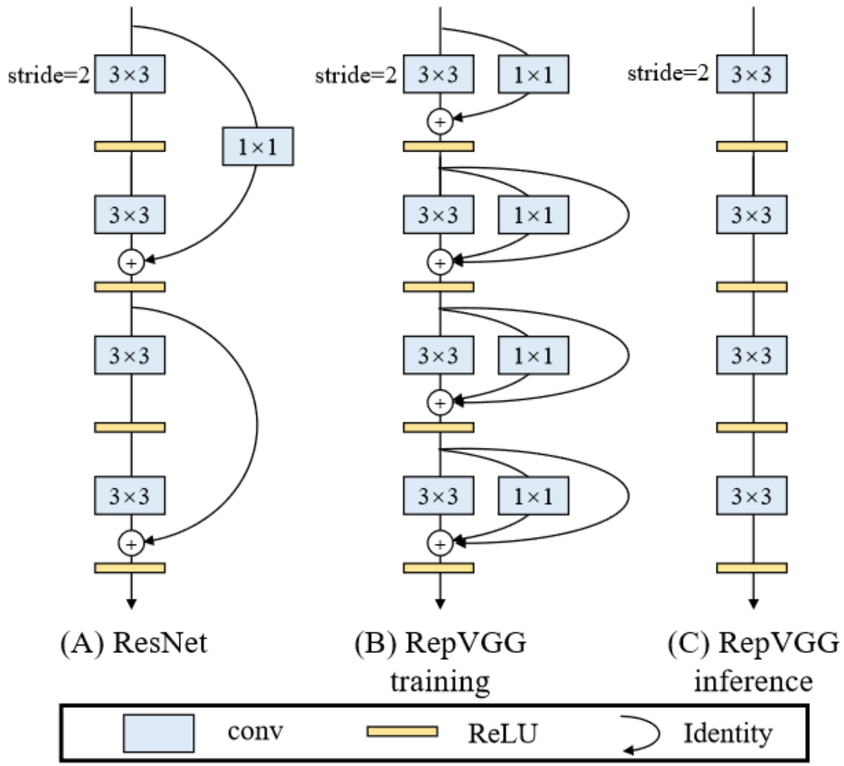
$$W_{fused} = \frac{\gamma * W}{\sqrt{var}}$$

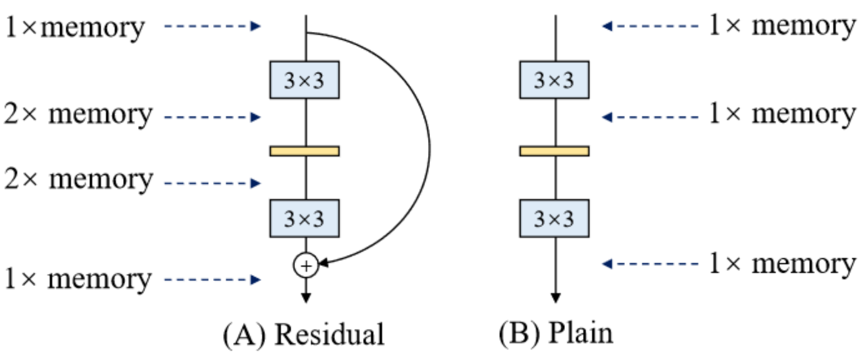
$$B_{fused} = \frac{\gamma * (b - mean)}{\sqrt{var}} + \beta$$

重参数化：RepVGG

RepVGG: Making VGG-style ConvNets Great Again

Xiaohan Ding [1*] Xiangyu Zhang [2] Ningning Ma [3]
Jungong Han [4] Guiguang Ding [1†] Jian Sun [2]
[1] Beijing National Research Center for Information Science and Technology (BNRist); School of Software, Tsinghua University, Beijing, China
[2] MEGVII Technology
[3] Hong Kong University of Science and Technology
[4] Computer Science Department, Aberystwyth University, SY23 3FL, UK





Deep Training Strategy

- 实际融合BN的操作，以及1x1卷积参数的融合：

```
if self.deploy:
    self.conv = nn.Conv2d(dim, dim_out, kernel_size=1)
else:
    self.conv1 = nn.Sequential(
        nn.Conv2d(dim, dim, kernel_size=1),
        nn.BatchNorm2d(dim, eps=1e-6),
    )
    self.conv2 = nn.Sequential(
        nn.Conv2d(dim, dim_out, kernel_size=1),
        nn.BatchNorm2d(dim_out, eps=1e-6)
    )
```

```
def _fuse_bn_tensor(self, conv, bn):
    kernel = conv.weight
    bias = conv.bias
    running_mean = bn.running_mean
    running_var = bn.running_var
    gamma = bn.weight
    beta = bn.bias
    eps = bn.eps
    std = (running_var + eps).sqrt()
    t = (gamma / std).reshape(-1, 1, 1, 1)
    return kernel * t, beta + (bias - running_mean) * gamma / std
```

```
def switch_to_deploy(self):
    kernel, bias = self._fuse_bn_tensor(self.conv1[0], self.conv1[1])
    self.conv1[0].weight.data = kernel
    self.conv1[0].bias.data = bias
    # kernel, bias = self.conv2[0].weight.data, self.conv2[0].bias.data
    kernel, bias = self._fuse_bn_tensor(self.conv2[0], self.conv2[1])
    self.conv = self.conv2[0]
    self.conv.weight.data = torch.matmul(kernel.transpose(1,3), self.conv1[0].weight.data.squeeze(3).squeeze(2)).transpose(1,3)
    self.conv.bias.data = bias + (self.conv1[0].bias.data.view(1,-1,1,1)*kernel).sum(3).sum(2).sum(1)
    self.__delattr__('conv1')
    self.__delattr__('conv2')
    self.act.switch_to_deploy()
    self.deploy = True
```

Series Informed Activation

- 一些工作发现，结构简单且层数较浅的网络主要受限于较差的非线性能力
- 有两种路线增强非线性能力：堆叠非线性层/提升单层的非线性能力
- 本文采取后者，且采用并行的非线性加权

$$A_s(x) = \sum_{i=1}^{n} a_i A(x + b_i),$$

- 为增强全局视野（卷积只是1x1大小），进一步扩展到邻域加权

$$A_s(x_{h,w,c}) = \sum_{i,j \in \{-n,n\}} a_{i,j,c} A(x_{i+h,j+w,c} + b_c),$$

Series Informed Activation

- 实际代码如下：

```
def forward(self, x):
    if self.deploy:
        x = self.conv(x)
    else:
        x = self.conv1(x)
        x = torch.nn.functional.leaky_relu(x,self.act_learn)
        x = self.conv2(x)

    x = self.pool(x)
    x = self.act(x)
    return x
```

```
class activation(nn.ReLU):
    def __init__(self, dim, act_num=3, deploy=False):
        super(activation, self).__init__()
        self.act_num = act_num
        self.deploy = deploy
        self.dim = dim
        self.weight = torch.nn.Parameter(torch.randn(dim, 1, act_num*2 + 1, act_num*2 + 1))
        if deploy:
            self.bias = torch.nn.Parameter(torch.zeros(dim))
        else:
            self.bias = None
            self.bn = nn.BatchNorm2d(dim, eps=1e-6)
        weight_init.trunc_normal_(self.weight, std=.02)

    def forward(self, x):
        if self.deploy:
            return torch.nn.functional.conv2d(
                super(activation, self).forward(x),
                self.weight, self.bias, padding=self.act_num, groups=self.dim)
        else:
            return self.bn(torch.nn.functional.conv2d(
                super(activation, self).forward(x),
                self.weight, padding=self.act_num, groups=self.dim))
```

其他细节

- 网络所有的卷积层，包括stem和head，均采用deep training方式训练：
- Stem将图像划分成4x4的不重叠区域，即non-overlap patch embedding

```
if self.deploy:
    self.stem = nn.Sequential(
        nn.Conv2d(in_chans, dims[0], kernel_size=4, stride=stride, padding=padding),
        activation(dims[0], act_num, deploy=self.deploy)
    )
else:
    self.stem1 = nn.Sequential(
        nn.Conv2d(in_chans, dims[0], kernel_size=4, stride=stride, padding=padding),
        nn.BatchNorm2d(dims[0], eps=1e-6),
    )
    self.stem2 = nn.Sequential(
        nn.Conv2d(dims[0], dims[0], kernel_size=1, stride=1),
        nn.BatchNorm2d(dims[0], eps=1e-6),
        activation(dims[0], act_num)
    )
```

```
if self.deploy:
    self.cls = nn.Sequential(
        nn.AdaptiveAvgPool2d((1,1)),
        nn.Dropout(drop_rate),
        nn.Conv2d(dims[-1], num_classes, 1),
    )
else:
    self.cls1 = nn.Sequential(
        nn.AdaptiveAvgPool2d((1,1)),
        nn.Dropout(drop_rate),
        nn.Conv2d(dims[-1], num_classes, 1),
        nn.BatchNorm2d(num_classes, eps=1e-6),
    )
    self.cls2 = nn.Sequential(
        nn.Conv2d(num_classes, num_classes, 1)
    )
```

```
def forward(self, x):
    if self.deploy:
        x = self.stem(x)
    else:
        x = self.stem1(x)
        x = torch.nn.functional.leaky_relu(x,self.act_learn)
        x = self.stem2(x)

    for i in range(self.depth):
        x = self.stages[i](x)

    if self.deploy:
        x = self.cls(x)
    else:
        x = self.cls1(x)
        x = torch.nn.functional.leaky_relu(x,self.act_learn)
        x = self.cls2(x)
    return x.view(x.size(0),-1)
```

其他细节

- 网络各种variant，scale-up主要在第三个stage：

	Input	VanillaNet-5	VanillaNet-6	VanillaNet-7/8/9/10/11/12/13
stem	224×224	4×4, 512, stride 4		
stage1	56×56	[1×1, 1024]×1 MaxPool 2×2	[1×1, 1024]×1 MaxPool 2×2	[1×1, 1024]×2 MaxPool 2×2
stage2	28×28	[1×1, 2048]×1 MaxPool 2×2	[1×1, 2048]×1 MaxPool 2×2	[1×1, 2048]×1 MaxPool 2×2
stage3	14×14	[1×1, 4096]×1 MaxPool 2×2	[1×1, 4096]×1 MaxPool 2×2	[1×1, 4096]×1/2/3/4/5/6/7 MaxPool 2×2
stage4	7×7	-	[1×1, 4096]×1	[1×1, 4096]×1
classifier	7× 7	AvgPool 7×7 1×1, 1000		

Table 6: Detailed architecture specifications.

其他细节

- 使用了较为复杂的数据增强操作（左），对比以往方法（右）：

Training Config	VanillaNet-{5/6/7/8/9/10/11/12/13}
weight init	trunc. normal (0.2)
optimizer	LAMB [51]
loss function	BCE loss
base learning rate	3.5e-3 {5,8-13} /4.8e-3 {6-7}
weight decay	0.35/0.35/0.35/0.3/0.3/0.25/0.3/0.3/0.3
optimizer momentum	β_1, β_2=0.9, 0.999
batch size	1024
training epochs	300
learning rate schedule	cosine decay
warmup epochs	5
warmup schedule	linear
dropout	0.05
layer-wise lr decay [5, 1]	0 {5,8-12} /0.8 {6-7,13}
randaugment [6]	(7, 0.5)
mixup [54]	0.1/0.15/0.4/0.4/0.4/0.4/0.8/0.8/0.8
cutmix [52]	1.0
color jitter	0.4
label smoothing [41]	0.1
exp. mov. avg. (EMA) [36]	0.999996 {5-10} /0.99992 {11-13}
test crop ratio	0.875 {5-11} /0.95 {12-13}

Table 7: ImageNet-1K training settings.

	PoolFormer				
	S12	S24	S36	M36	M48
Peak drop rate of stoch. depth d_r	0.1	0.1	0.2	0.3	0.4
LayerScale initialization ϵ	10^{-5}	10^{-5}	10^{-6}	10^{-6}	10^{-6}
Data augmentation	AutoAugment				
Repeated Augmentation	off				
Input resolution	224				
Epochs	300				
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	4096				
Peak learning rate used in the paper	4×10^{-4}				
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 7. **Hyper-parameters for image classification on ImageNet-1K**

- 作者介绍
- 研究背景
- 方法
- 实验效果
- 总结

实验效果—消融实验

Table 1: Ablation study on the number of series.

n	FLOPs (B)	Latency (ms)	Top-1 (%)
0	5.83	1.96	60.53
1	5.86	1.97	74.53
2	5.91	1.99	75.62
3	5.99	2.01	76.36
4	6.10	2.18	76.43

Table 3: Ablation on adding shortcuts.

Type	Top-1 (%)
no shortcut	**76.36**
shortcut before act	75.92
shortcut after act	75.72

Table 2: Ablation study on different networks.

Network	Deep train.	Series act.	Top-1 (%)
VanillaNet-6			59.58
	✓		60.53
		✓	75.23
	✓	✓	76.36
AlexNet			57.52
	✓		59.09
		✓	61.12
	✓	✓	63.59
ResNet-50			76.13
	✓		76.16
		✓	76.30
	✓	✓	76.27

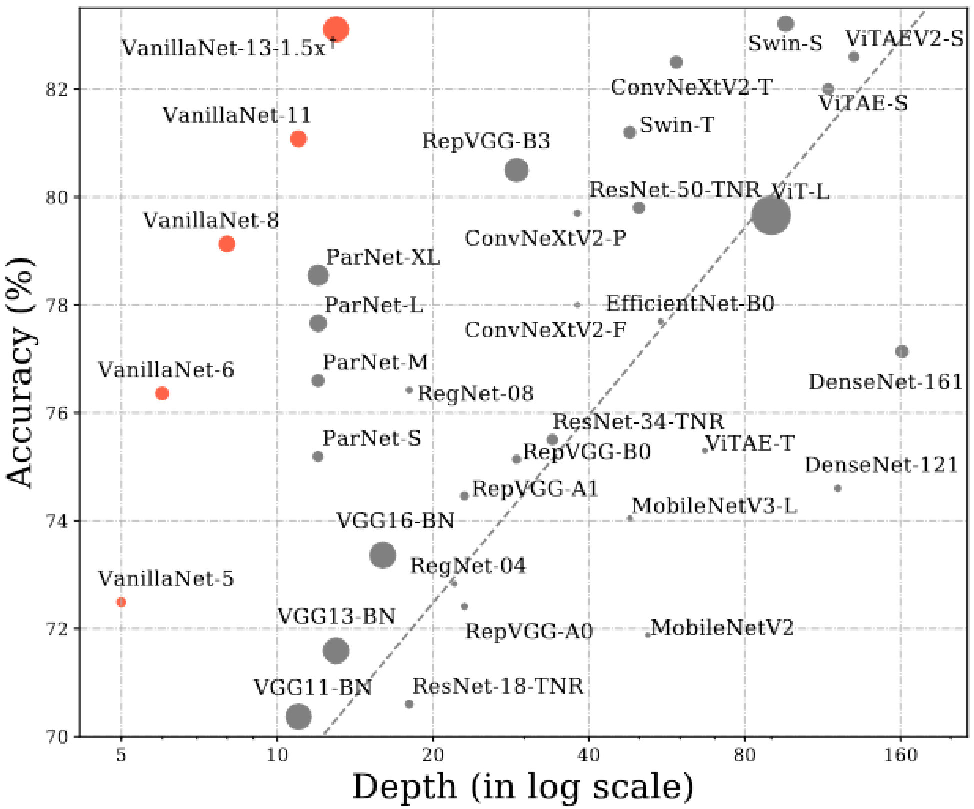
实验

Table 4: Comparison on ImageNet. Latency is tested on Nvidia A100 GPU with batch size of 1.

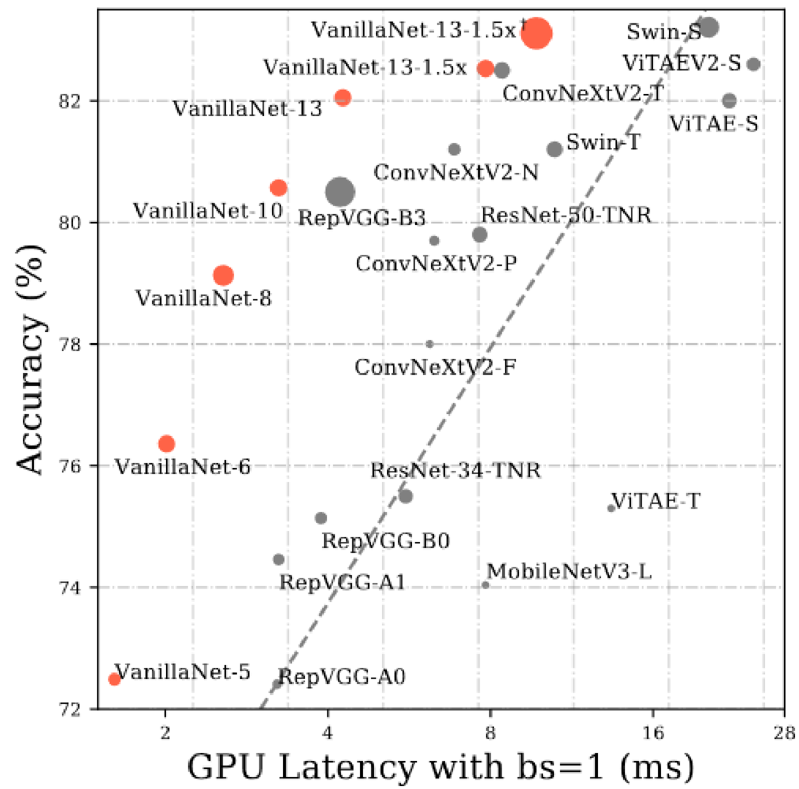
Model	Params (M)	FLOPs (B)	Depth	Latency (ms)	Acc (%)	Real Acc (%)
MobileNetV3-Small [21]	2.5	0.06	48	6.65	67.67	74.33
MobileNetV3-Large [21]	5.5	0.22	48	7.83	74.04	80.01
ShuffleNetV2x1.5 [39]	3.5	0.30	51	7.23	73.00	80.19
ShuffleNetV2x2 [21]	7.4	0.58	51	7.84	76.23	82.72
RepVGG-A0 [12]	8.1	1.36	23	3.22	72.41	79.33
RepVGG-A1 [12]	12.8	2.37	23	3.24	74.46	81.02
RepVGG-B0 [12]	14.3	3.1	29	3.88	75.14	81.74
RepVGG-B3 [12]	110.9	26.2	29	4.21	80.50	86.44
ViTAE-T [48]	4.8	1.5	67	13.37	75.3	82.9
ViTAE-S [48]	23.6	5.6	116	22.13	82.0	87.0
ViTAEV2-S [55]	19.2	5.7	130	24.53	82.6	87.6
ConvNextV2-A [46]	3.7	0.55	41	6.07	76.2	82.79
ConvNextV2-F [46]	5.2	0.78	41	6.17	78.0	84.08
ConvNextV2-P [46]	9.1	1.37	41	6.29	79.7	85.60
ConvNextV2-N [46]	15.6	2.45	47	6.85	81.2	-
ConvNextV2-T [46]	28.6	4.47	59	8.40	82.5	-
ConvNextV2-B [46]	88.7	15.4	113	15.41	84.3	-
Swin-T [31]	28.3	4.5	48	10.51	81.18	86.64
Swin-S [31]	49.6	8.7	96	20.25	83.21	87.60
ResNet-18-TNR [43]	11.7	1.8	18	3.12	70.6	79.4
ResNet-34-TNR [43]	21.8	3.7	34	5.57	75.5	83.4
ResNet-50-TNR [43]	25.6	4.1	50	7.64	79.8	85.7
VanillaNet-5	15.5	5.2	5	1.61	72.49	79.66
VanillaNet-6	32.5	6.0	6	2.01	76.36	82.86
VanillaNet-7	32.8	6.9	7	2.27	77.98	84.16
VanillaNet-8	37.1	7.7	8	2.56	79.13	85.14
VanillaNet-9	41.4	8.6	9	2.91	79.87	85.66
VanillaNet-10	45.7	9.4	10	3.24	80.57	86.25
VanillaNet-11	50.0	10.3	11	3.59	81.08	86.54
VanillaNet-12	54.3	11.1	12	3.82	81.55	86.81
VanillaNet-13	58.6	11.9	13	4.26	82.05	87.15
VanillaNet-13-1.5×	127.8	26.5	13	7.83	82.53	87.48
VanillaNet-13-1.5×†	127.8	48.9	13	9.72	83.11	87.85

容计算实验室
Content Computing Lab

实验效果—效果对比



(a) Accuracy vs. depth



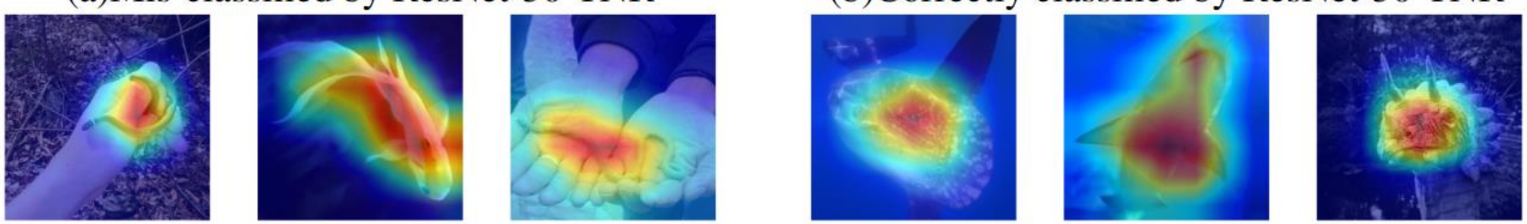
(b) Accuracy v.s. inference speed

实验效果—效果对比

Table 5: Performance on COCO detection and segmentation. FLOPs are calculated with image size (1280, 800)on Nvidia A100 GPU.

Framework	Backbone	FLOPs	Params	FPS	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^b_{75}
RetinaNet [29]	Swin-T [31]	245G	38.5M	27.5	41.5	62.1	44.2	-	-	-
	VanillaNet-13	397G	74.6M	29.8	41.8	62.8	44.3	-	-	-
Mask RCNN [16]	Swin-T [31]	267G	47.8M	28.2	42.7	65.2	46.8	39.3	62.2	42.2
	VanillaNet-13	421G	76.3M	32.6	42.9	65.5	46.9	39.6	62.5	42.2

(a)Mis-classified by ResNet-50-TNR (b)Correctly classified by ResNet-50-TNR



(c)Mis-classified by VanillaNet-9 (d)Correctly classified by VanillaNet-9

Figure 2: Visualization of attention maps of the classified samples by ResNet-50 and VanillaNet-9. We show the attention maps of their mis-classified samples and correctly classified samples for comparison.

- 作者介绍
- 研究背景
- 方法
- 实验效果
- 总结

总结反思

- 在实际网络部署中，目前GPU对纯卷积网络适应性最好，因此本文设计了纯卷积网络，并且极致地压缩了网络层数，并摈弃了shortcut操作，进一步简化
- 通过重参数化方法融合BN和多层卷积，将训练时的网络参数重组，得到更简化的推理网络结构，是当今网络轻量化的常见套路
- 通过在网络训练前期引入非线性激活，以及对邻域进行加权，补足了网络的非线性能力

谢谢！