VanillaNet: the Power of Minimalism in Deep Learning

Hanting Chen[1], **Yunhe Wang**[1*], **Jianyuan Guo**[1], **Dacheng Tao**[2]
[1] Huawei Noah's Ark Lab. [2] School of Computer Science, University of Sydney.

Content

- Background
- Motivation
- Method
- Result
- Extension

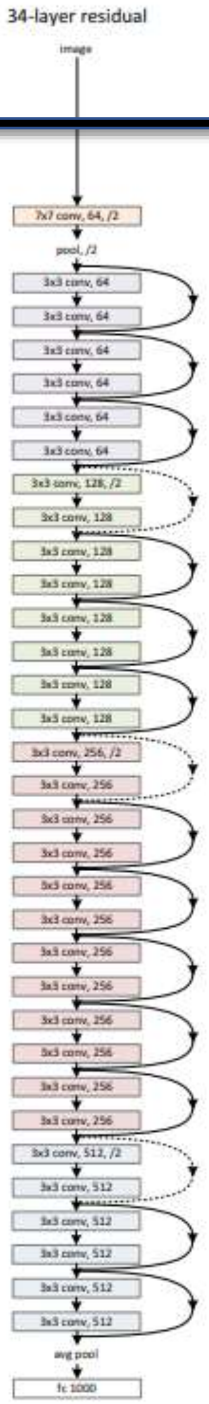
Background

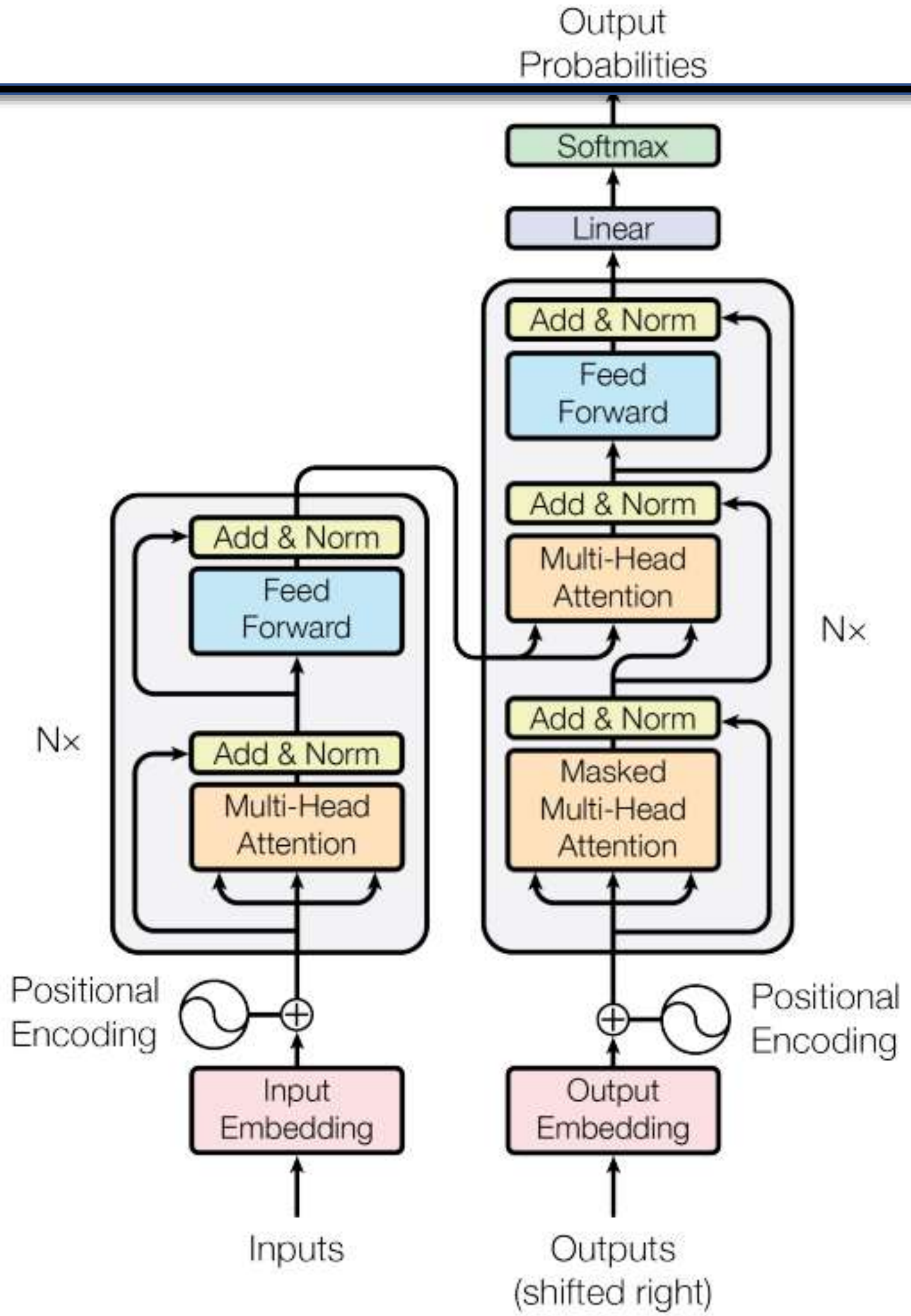
Mainstream network architectures

- Residual-based
- Transformer-based

Both: numerous layers with **a large number of** neurons or transformer blocks

Is their status unassailable?





Motivation

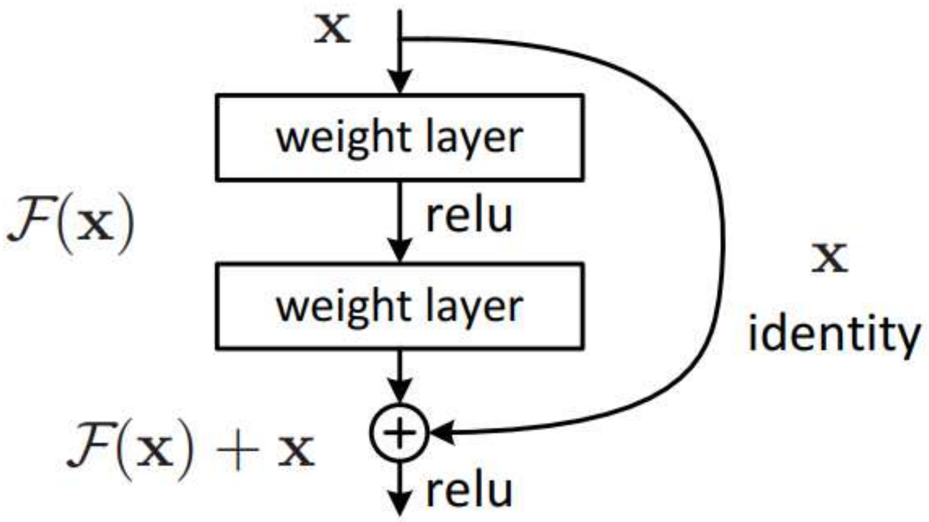
Previous problems

- Inherent **complexity**(high depth, shortcuts, self-attention...)
- Hard for **deployment**

Can we eschew all of this?

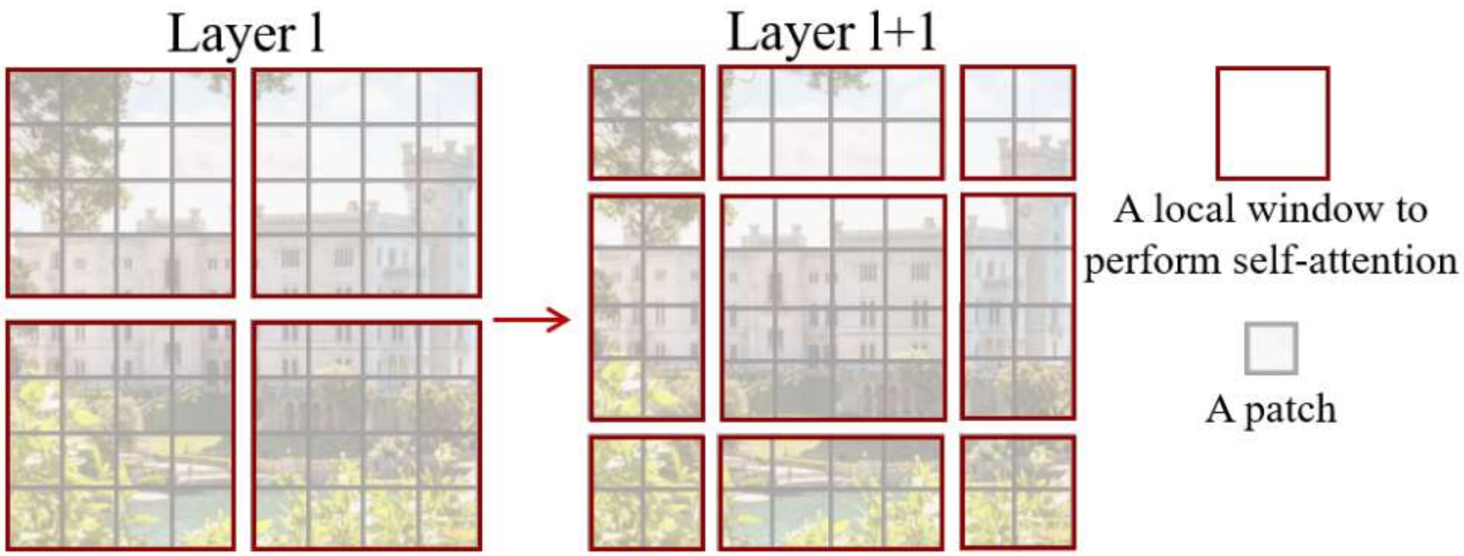
Yes. VanillaNet

Residual block



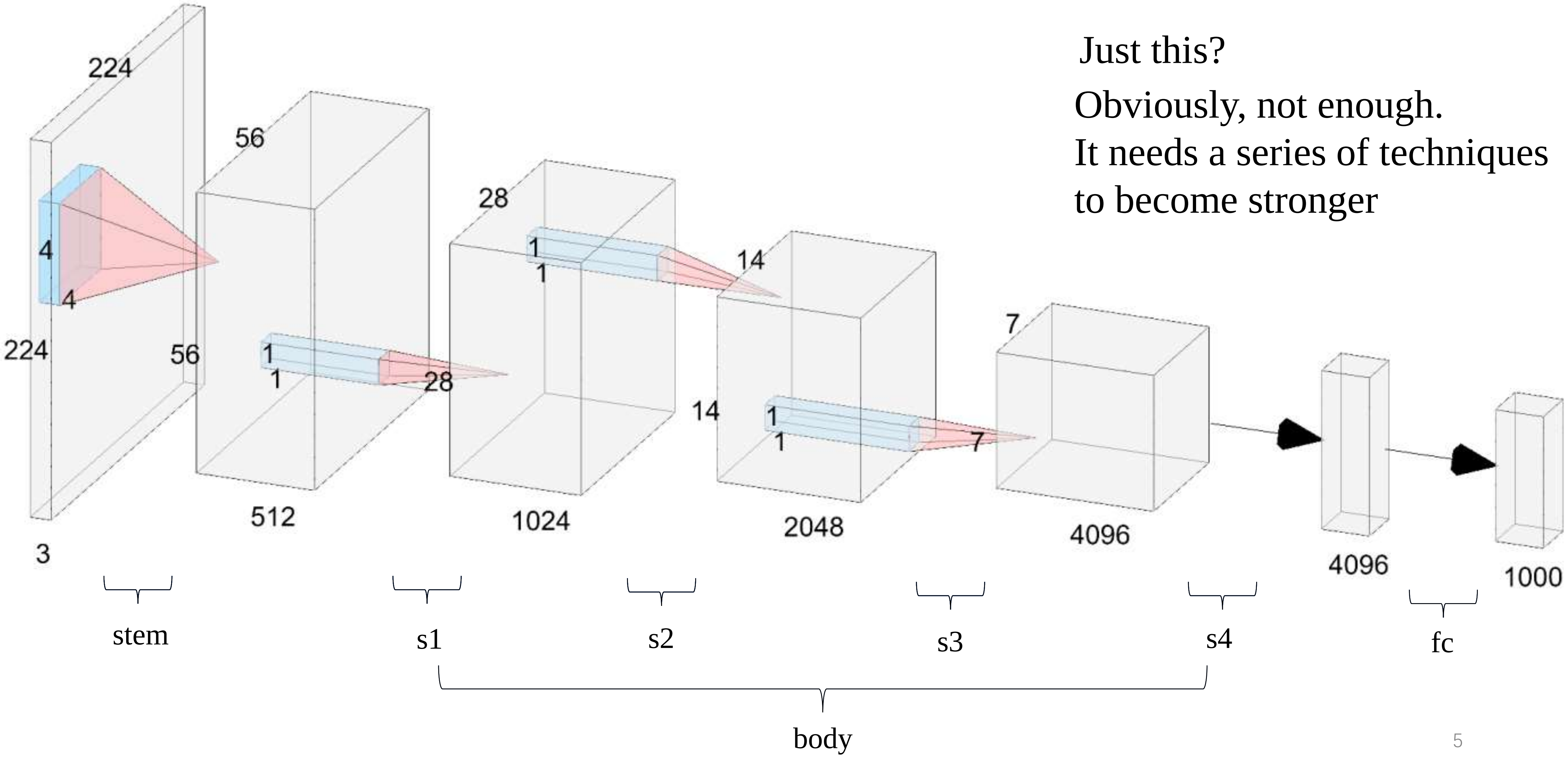
Significant off-chip **memory** traffic

Sliding window self-attention in Swin Transformer



sophisticated engineering **implementation**, e.g. Rewriting CUDA code

Method-overview



Just this?

Obviously, not enough.
It needs a series of techniques to become stronger

Method-techniques

Let’s consider why the current network is weak

- **Weak non-linearity**

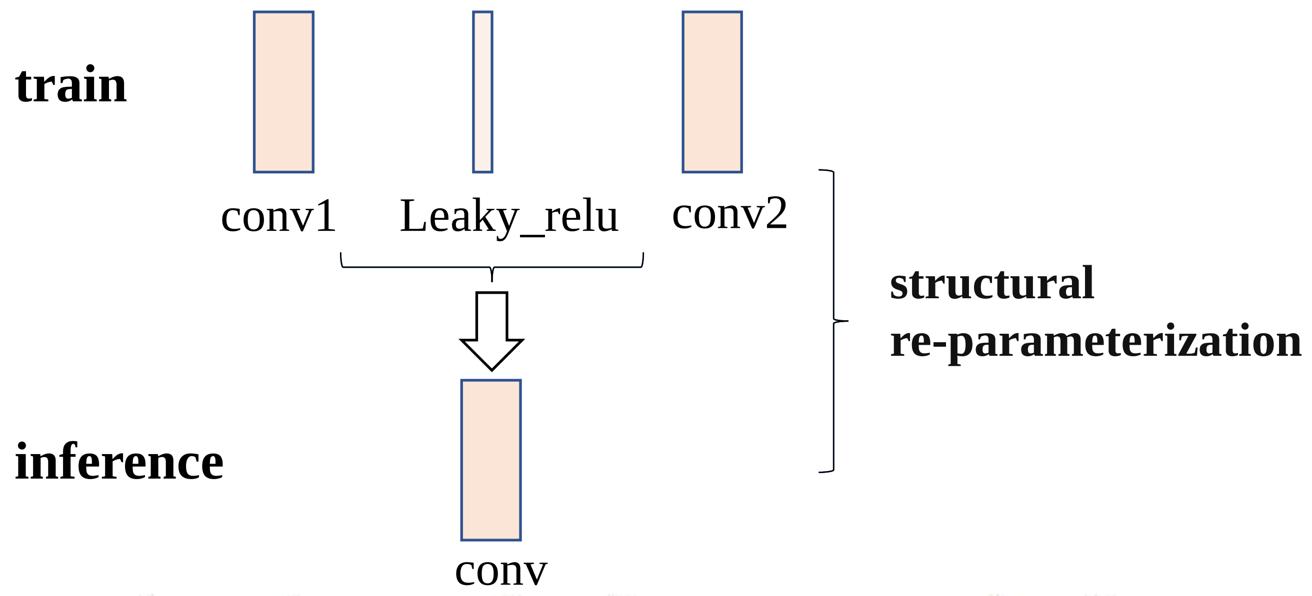
How?

Deep Training Strategy

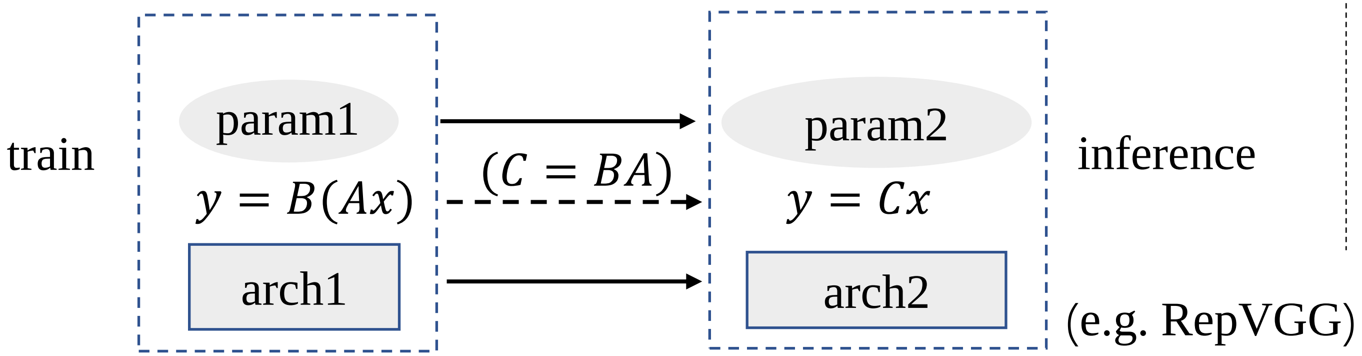
Series Informed Activation Function

Deep training

- Deep Training Strategy



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



Conv + BN -> new conv

$$W_i' = \frac{\gamma_i}{\sigma_i} W_i, B_i' = \frac{(B_i - \mu_i)\gamma_i}{\sigma_i} + \beta_i$$

Proof:

$$Conv(X) = WX + B$$

$$BN(X) = \gamma * \frac{X - \mu}{\sigma} + \beta$$

$$newconv(X) = \mathrm{BN}(\mathrm{Conv(X)})$$

$$= \gamma * \frac{WX + B - \mu}{\sigma} + \beta$$

$$= \underbrace{\boxed{\gamma * \frac{W}{\sigma}}}_{W'} X + \underbrace{\boxed{\gamma * \frac{B - \mu}{\sigma} + \beta}}_{B'}$$

Series Informed Activation Function

improve non-linearity

1. **serially stacking** of activation function
2. **increase the non-linearity** of activation layer

Choice2: **concurrently** stacking activation layer

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

learn the global information by **varying the inputs from their neighbors**

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

Similar to
Batch Normalization with Enhanced Linear Transformation

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

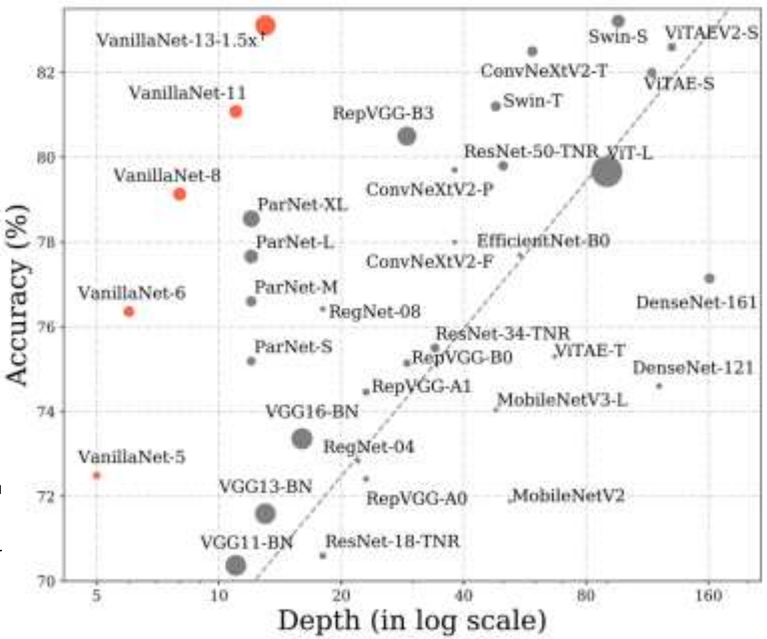
HantingChen commented last week — Collaborator

Thanks for the attention. We use the depth conv as an efficient implementation of our activation function, which is same as Eq. (6) in our paper. Each element of the output of this activation is related to various non-linear inputs, which can be regarded as concurrently stacking.

Result

- **comparable** performance

- **Much smaller** depth and latency



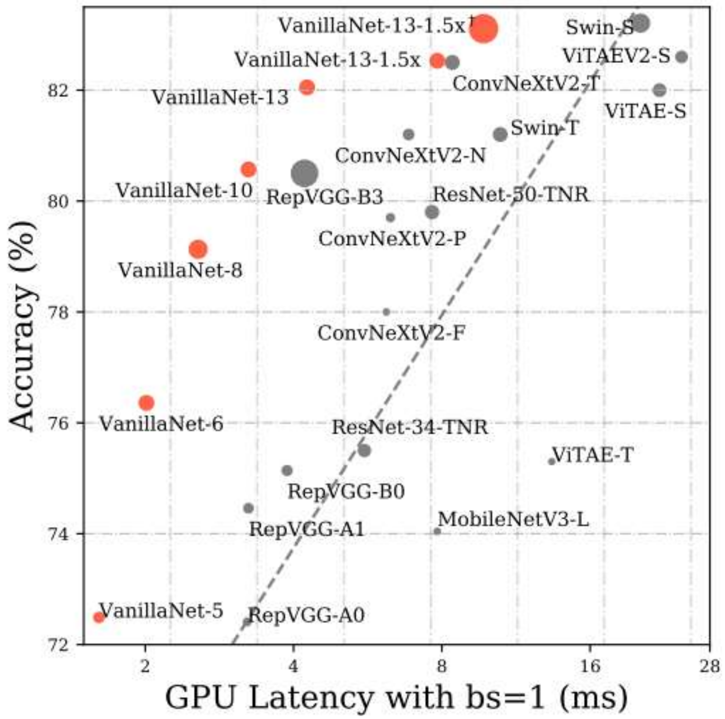


Table 4: Comparison on ImageNet. Latency is tested on Nvidia A100 GPU with batch size of 1.

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 [45]	11.7	1.8	18	3.12	70.6	79.4
ResNet-34-TNR [45]	21.8	3.7	34	5.57	75.5	83.4
ResNet-50-TNR [45]	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

Ablation Study

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

Table 3: Ablation on adding shortcuts.

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

The shortcut is **useless** for bringing up the non-linearity and may decrease non-linearity

deep training technique is **useful for the shallow** network

Extension

1. What is most important for the performance improvement of a deep neural network?

- Depth? Receptive field? Attention? Params?...

2. Could we replace the complex backbones of current big visual models with **simple, shallow yet effective** backbones?