VanillaNet

深度学习中极简网络的威力

陈汉亭

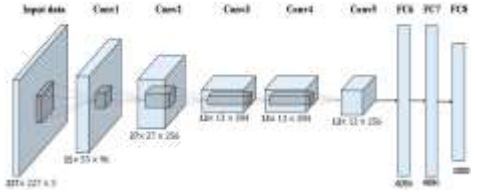
华为诺亚方舟实验室

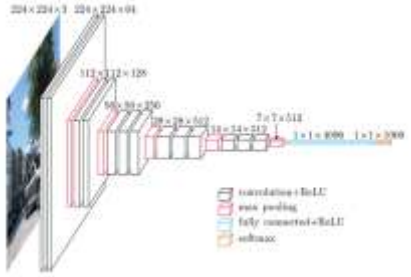
chenhanting@huawei.com

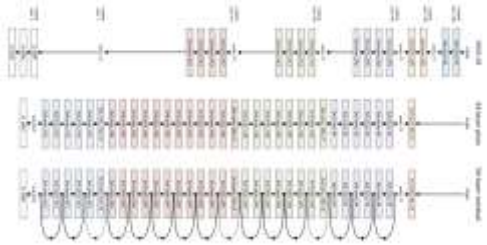
HUAWEI

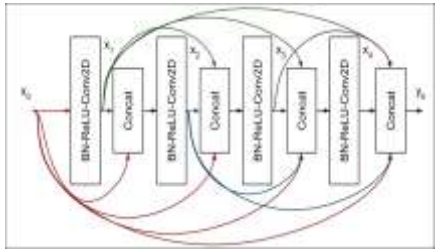
基础模型遵循“更多即是不同”的哲学，在视觉任务取得了惊人的效果

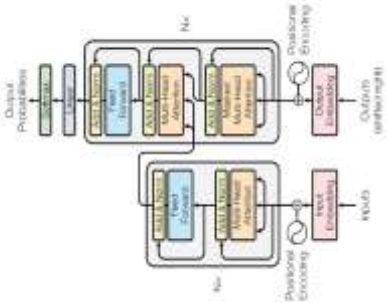
2012	2014	2015	2016	2020
12 layers	<50 layers	>1000 layers	20-100 layers	50-200 layers
AlexNet	**VGG/Inception/…**	**ResNet**	**Deep Nets Variants**	**ViT and its Variants**
深度学习的开创性工作。证明深度网络比传统方法获得更高的准确度。	深度卷积模型开始在计算机视觉领域占主导地位。	解决梯度消失问题，利用残差模块训练多达1000层的网络。	新兴结构逐渐涌现：DenseNet, ResNeXt, EfficientNet, ConvNext等。	将NLP中的Transformer架构引入CV中：DeiT, PVT, TNT, Swin等。
卷积、激活	卷积、激活，多路	卷积、激活、残差块	卷积、注意力、激活、跳连	自注意力、激活、残差块





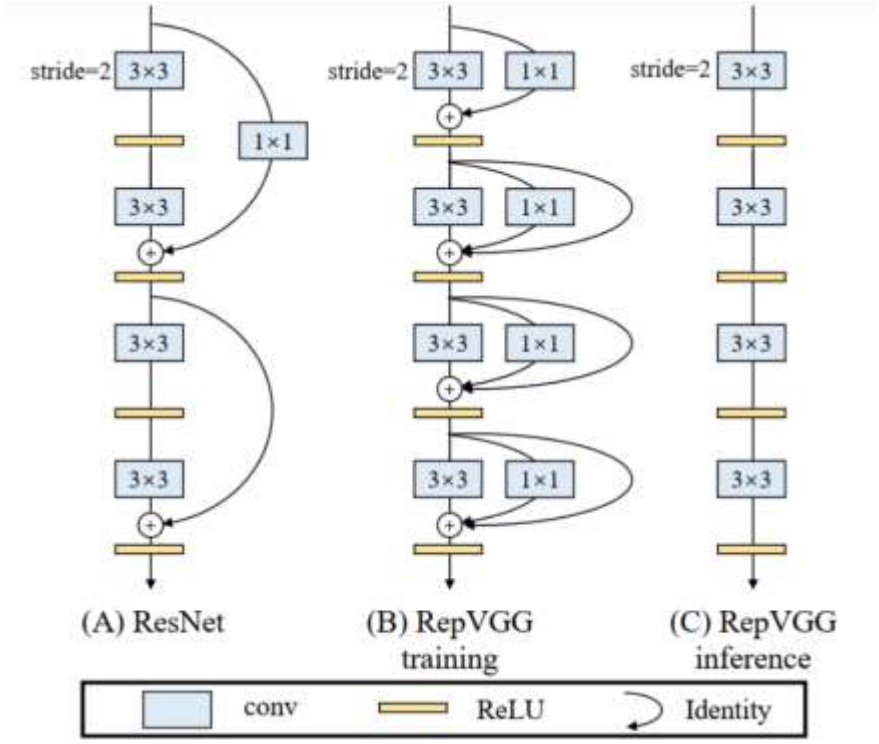






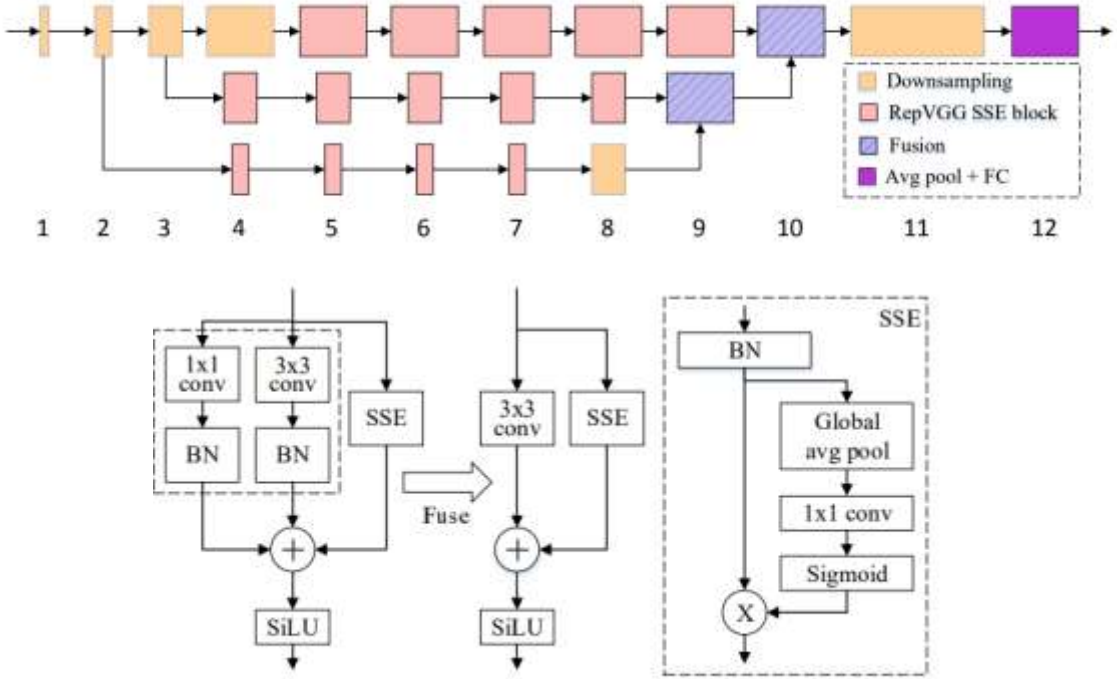
极简架构的探索：不同方案但仍存在改进空间

RepVGG：采用重参数化技巧



网络仍然有20+层

ParNet：使用并行架构减少层数

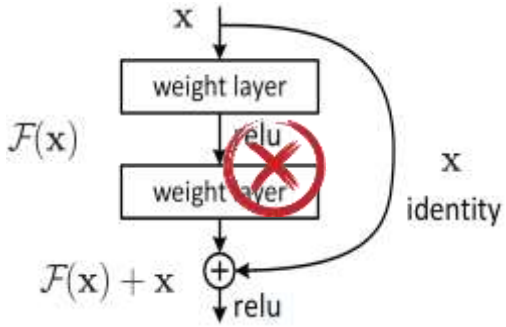


12层ImageNet80%+，但结构复杂

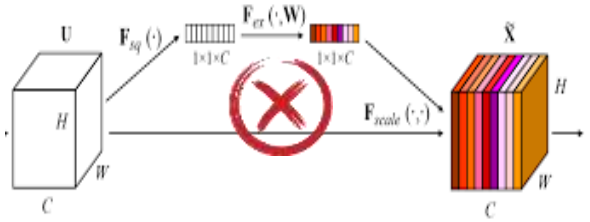
HUAWEI

在设计中拥抱优雅：反直觉的方式，构造简洁而强大的VanillaNet

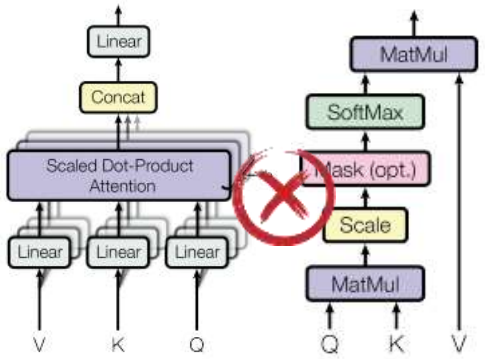
残差块



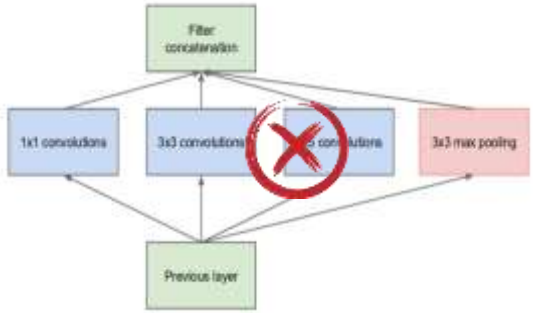
通道注意力



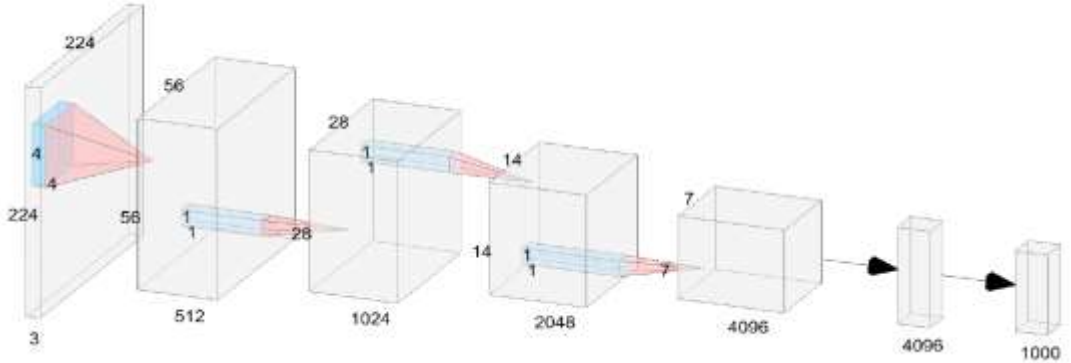
自注意力



多分支结构



构建只含卷积计算的**极简直筒结构**

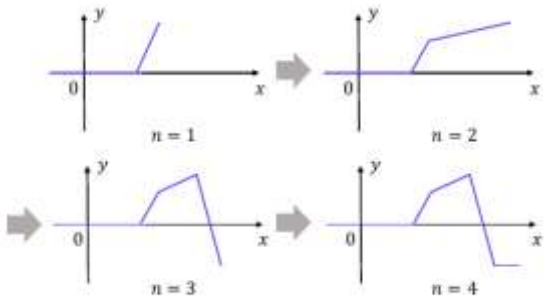


	Input	VanillaNet-5	VanillaNet-6	VanillaNet-7/8/9/10/11/12/13
stem	224×224	4×4, 512, stride 4		
stage1	56×56	[1×1, 1024]×1 MaxPool 2×2	[1×1, 1024]×1 MaxPool 2×2	[1×1, 1024]×2 MaxPool 2×2
stage2	28×28	[1×1, 2048]×1 MaxPool 2×2	[1×1, 2048]×1 MaxPool 2×2	[1×1, 2048]×1 MaxPool 2×2
stage3	14×14	[1×1, 4096]×1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

HUAWEI

解决非线性能力不足情况下的优化难题，大幅提升极简网络精度

解决方法：**并行**而非**串行**堆叠激活函数



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

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

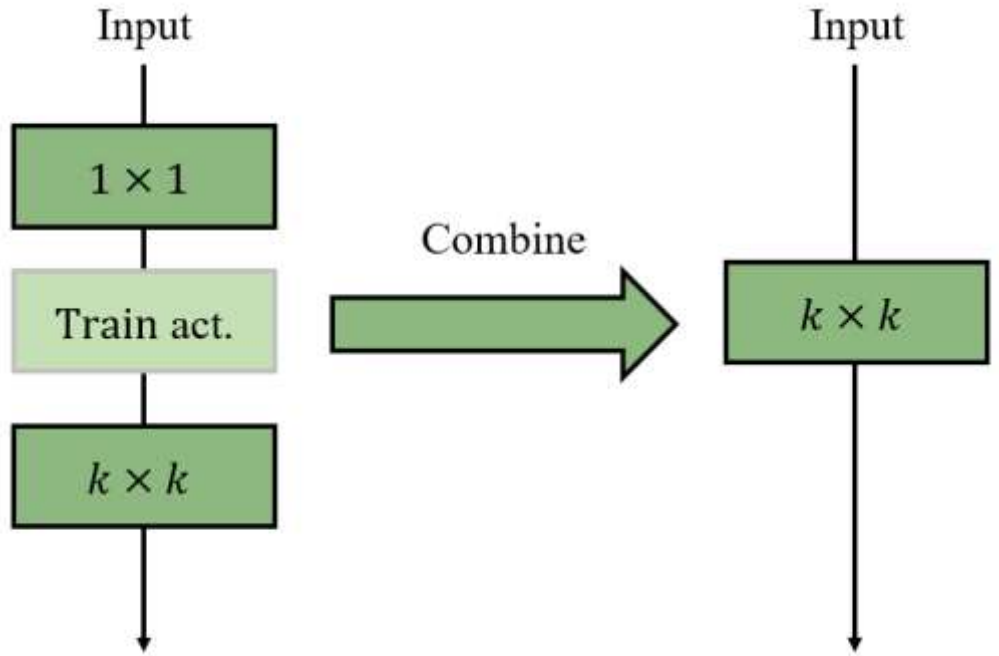
$$\mathcal{O}(\text{CONV}) = H \times W \times C_{in} \times C_{out} \times k^2$$

$$\mathcal{O}(\text{SA}) = H \times W \times C_{in} \times n^2$$

$$\frac{\mathcal{O}(\text{CONV})}{\mathcal{O}(\text{SA})} = \frac{H \times W \times C_{in} \times C_{out} \times K^2}{H \times W \times C_{in} \times n^2} = \frac{C_{out} \times k^2}{n^2}$$

解决方法：**深层训练**、**浅层部署**，训练时增加非线性

$$A'(x) = (1 - \lambda)A(x) + \lambda x$$



HUAWEI

深入分析极简网络的能力和解决方案的效果，证明极简主义的力量

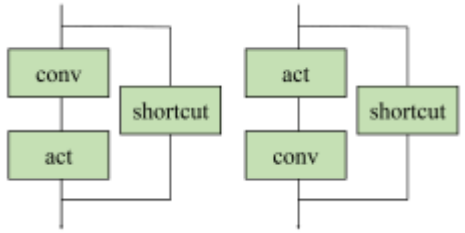
提出的**级数激活函数和深度训练**在不同网络的效果

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

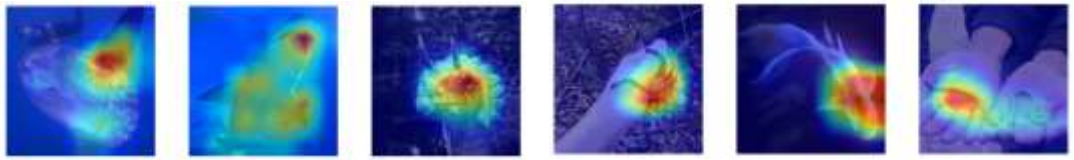
仅**对简单的浅网络有效**，对复杂网络无提升

残差模块对VanillaNet**并无帮助**

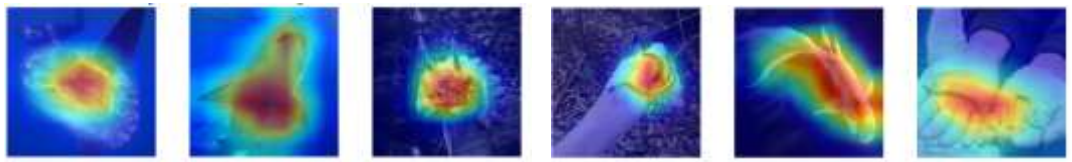
Type	Top-1 (%)
no shortcut	**76.36**
shortcut before act	75.92
shortcut after act	75.72



极简网络仍然具有**强大的特征提取能力**

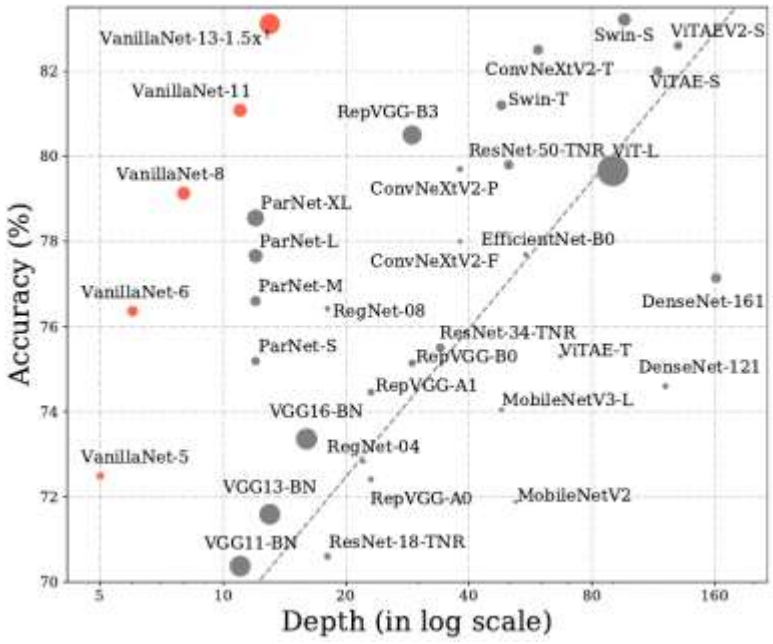


ResNet-50 特征可视化

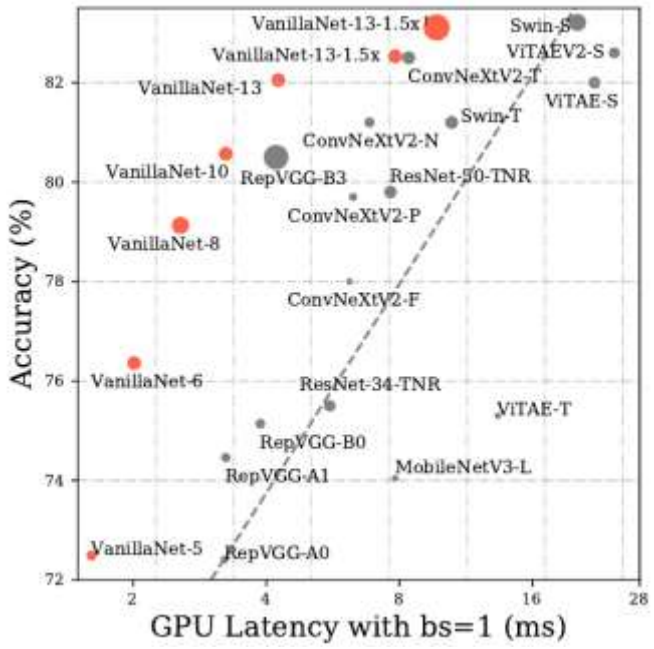


VanillaNet-9 特征可视化

无需残差等复杂模块，13层网络在ImageNet上达到83%精度



深度-精度对比



速度-精度对比

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

6层VanillaNet性能超越**34层**ResNet，速度提升**一倍以上**
13层 VanillaNet性能超越**近百层**Swin-S，速度提升**一倍以上**

HUAWEI

代码解读

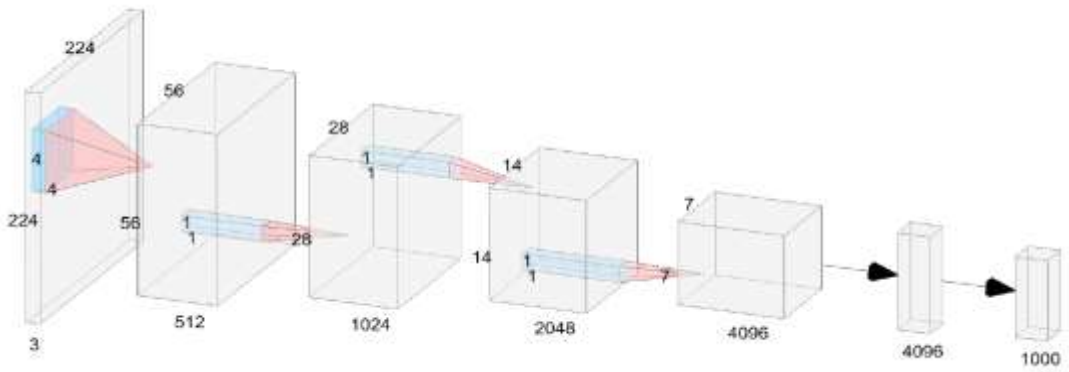
$$A_s(x_{h,w,c}) = \sum_{i,j \in \{-n,n\}} a_{i,j,c} A(x_{i+h,j+w,c}) + b_c$$

```
class activation(nn.ReLU):
    def __init__(self, dim, act_num=3):
        super(activation, self).__init__()
        self.weight = torch.nn.Parameter(torch.randn(dim, 1, act_num*2 + 1, act_num*2 + 1))
        self.bn = nn.BatchNorm2d(dim, eps=1e-6)
        self.dim = dim
        self.act_num = act_num

    def forward(self, x):
        return torch.nn.functional.conv2d(
            super(activation, self).forward(x),
            self.weight, self.bias, padding=(self.act_num*2 + 1)//2, groups=self.dim)
```

$$A'(x) = (1-\lambda)A(x) + \lambda x$$

```
class Block(nn.Module):
    def __init__(self, dim, dim_out, act_num=3, stride=2, deploy=False):
        super().__init__()
        self.act_learn = 1
        self.deploy = deploy
        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)
            )

        self.pool = nn.Identity() if stride == 1 else nn.MaxPool2d(stride)
        self.act = activation(dim_out, act_num)

    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
```

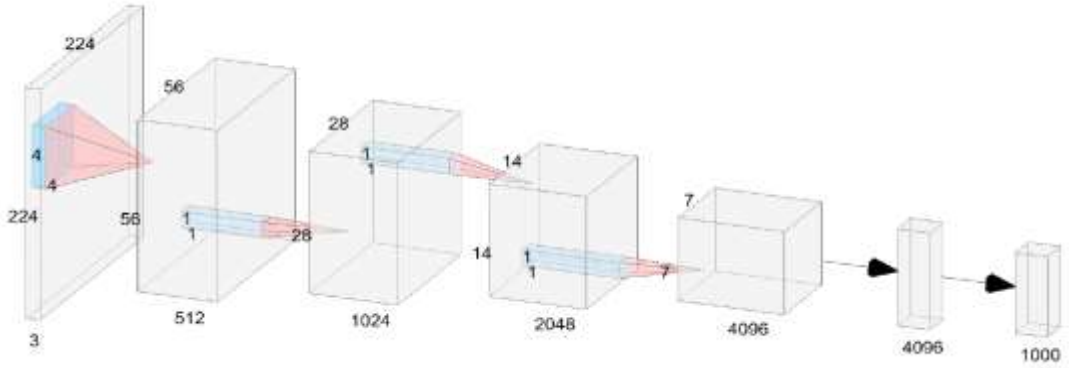
代码解读



	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		

```
class VanillaNet(nn.Module):
    def __init__(self, in_chans=3, num_classes=1000, dims=[96, 192, 384, 768],
                 drop_rate=0, act_num=3, strides=[2,2,2,1]):
        super().__init__()
        self.stem = nn.Sequential(
            nn.Conv2d(in_chans, dims[0], kernel_size=4, stride=4),
            activation(dims[0], act_num)
        )

        self.stages = nn.ModuleList()
        for i in range(len(strides)):
            stage = Block(dim=dims[i], dim_out=dims[i+1], act_num=act_num, stride=strides[i], deploy=deploy)
            self.stages.append(stage)
        self.depth = len(strides)

        self.cls = nn.Sequential(
            nn.AdaptiveAvgPool2d((1,1)),
            nn.Dropout(drop_rate),
            nn.Conv2d(dims[-1], num_classes, 1),
        )

    def forward(self, x):
        x = self.stem(x)

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

        x = self.cls(x)
        return x.view(x.size(0),-1)
```

代码解读



	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		

```
@register_model
def vanillanet_5(pretrained=False,in_22k=False, **kwargs):
    model = VanillaNet(dims=[128*4, 256*4, 512*4, 1024*4], strides=[2,2,2], **kwargs)
    return model

@register_model
def vanillanet_6(pretrained=False,in_22k=False, **kwargs):
    model = VanillaNet(dims=[128*4, 256*4, 512*4, 1024*4, 1024*4], strides=[2,2,2,1], **kwargs)
    return model

@register_model
def vanillanet_7(pretrained=False,in_22k=False, **kwargs):
    model = VanillaNet(dims=[128*4, 128*4, 256*4, 512*4, 1024*4, 1024*4], strides=[1,2,2,2,1], **kwargs)
    return model

@register_model
def vanillanet_8(pretrained=False, in_22k=False, **kwargs):
    model = VanillaNet(dims=[128*4, 128*4, 256*4, 512*4, 512*4, 1024*4, 1024*4], strides=[1,2,2,1,2,1], **kwargs)
    return model

@register_model
def vanillanet_9(pretrained=False, in_22k=False, **kwargs):
    model = VanillaNet(dims=[128*4, 128*4, 256*4, 512*4, 512*4, 512*4, 1024*4, 1024*4], strides=[1,2,2,1,1,2,1], **kwargs)
    return model

@register_model
def vanillanet_10(pretrained=False, in_22k=False, **kwargs):
    model = VanillaNet(
        dims=[128*4, 128*4, 256*4, 512*4, 512*4, 512*4, 512*4, 1024*4, 1024*4],
        strides=[1,2,2,1,1,1,2,1],
        **kwargs)
    return model

@register_model
def vanillanet_11(pretrained=False, in_22k=False, **kwargs):
    model = VanillaNet(
        dims=[128*4, 128*4, 256*4, 512*4, 512*4, 512*4, 512*4, 512*4, 1024*4, 1024*4],
        strides=[1,2,2,1,1,1,1,2,1],
        **kwargs)
    return model
```

谢谢！

请提问

论文地址：https://arxiv.org/abs/2305.12972

代码开源地址：https://github.com/huawei-noah/VanillaNet