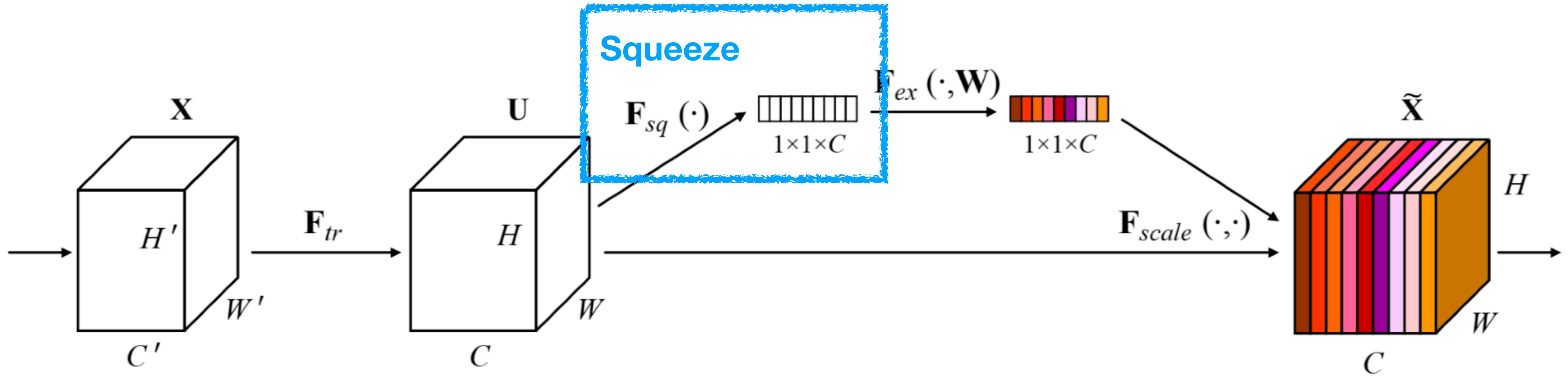
SENet

Cho Sung Man

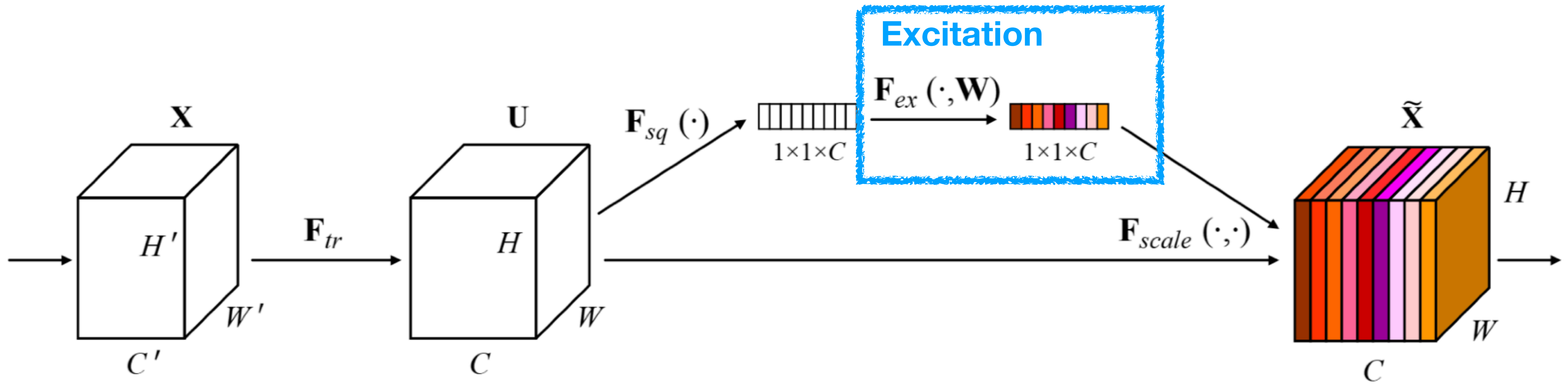
INTRO

SQUEEZE



Embedding of the global distribution of channel-wise feature responses, allowing information from the global receptive field of the network to be used by all its layers

EXCITATION



Simple self-gating mechanism that takes the embeddings as input and produces a collection of per-channel modulation weights.

Related Works

Deeper Architecture

VGG, Inception

increasing the depth of a network could significantly increase the quality of representations that it was capable of learning

Deeper Architecture

VGG, Inception

increasing the depth of a network could significantly increase the quality of representations that it was capable of learning

Batch Normalization

added stability to the learning process and produced smoother optimization surfaces

Deeper Architecture

VGG, Inception

increasing the depth of a network could significantly increase the quality of representations that it was capable of learning

Batch Normalization

added stability to the learning process and produced smoother optimization surfaces

ResNet

learn considerably deeper and stronger networks through the use of identity-based skip connections

Deeper Architecture

VGG, Inception

increasing the depth of a network could significantly increase the quality of representations that it was capable of learning

Batch Normalization

added stability to the learning process and produced smoother optimization surfaces

ResNet

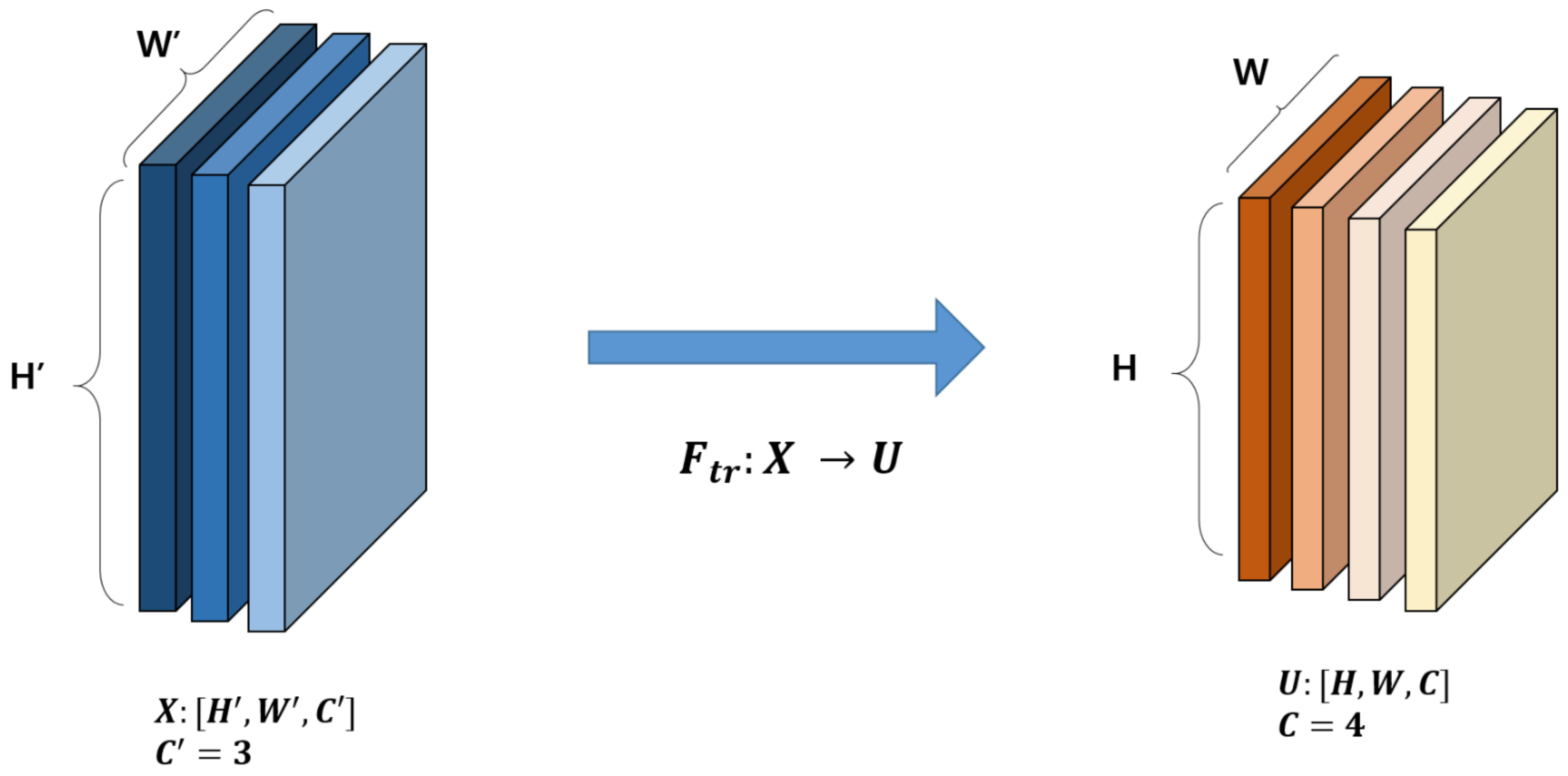
learn considerably deeper and stronger networks through the use of identity-based skip connections

Highway networks

introduced a gating mechanism to regulate the flow of information along shortcut connections.

SQUEEZE AND EXCITATION

transformation



$$U = Ftr(X),\ X \in R^{H' \times W' \times C'},\ U \in R^{H \times W \times C}$$

Conv Layer

convolutional operator. Let $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_C]$ denote the learned set of filter kernels, where $\mathbf{v}_c$ refers to the parameters of the c-th filter. We can then write the outputs of $\mathbf{F}_{tr}$ as $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_C]$, where

$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s. \tag{1}$$

Conv Layer

convolutional operator. Let $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_C]$ denote the learned set of filter kernels, where $\mathbf{v}_c$ refers to the parameters of the c-th filter. We can then write the outputs of $\mathbf{F}_{tr}$ as $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_C]$, where

$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \boxed{\sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s .} \qquad (1)$$

the channel relationships modeled by convolution are inherently local.

To be Global !

How !?

Squeeze & Excitation

Squeeze (Global Information Embedding)

squeeze **global spatial information into a channel descriptor.**

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j).$$

Channel Descriptor

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j).$$

In this paper, author use GAP (Global Average Pooling).

Simple ! Do not increase training-parameters !

The transformation output U can be interpreted as a collection of the local descriptors whose statistics are expressive for the whole image. we can use other channel descriptor function.

Before Excitation,

To make use of information aggregated in the *squeeze* operation, we follow it with a second operation which aims to fully capture channel-wise dependencies.

To fulfil this objective, the function must meet two criteria:

1. **It must be flexible.**

   must be capable of learning a nonlinear interaction between channels

2. **It must learn a non-mutually-exclusive relationship**

   since we would like to ensure that multiple channels are allowed to be emphasized (rather than enforcing a one-hot activation).

Excitation

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \overset{\text{Sigmoid}}{\sigma}(\mathbf{W}_2 \underset{\text{ReLU}}{\delta}(\mathbf{W}_1 \overset{\text{Channel Descriptor}}{\mathbf{z}})),$$

$$\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C} \qquad \mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}.$$

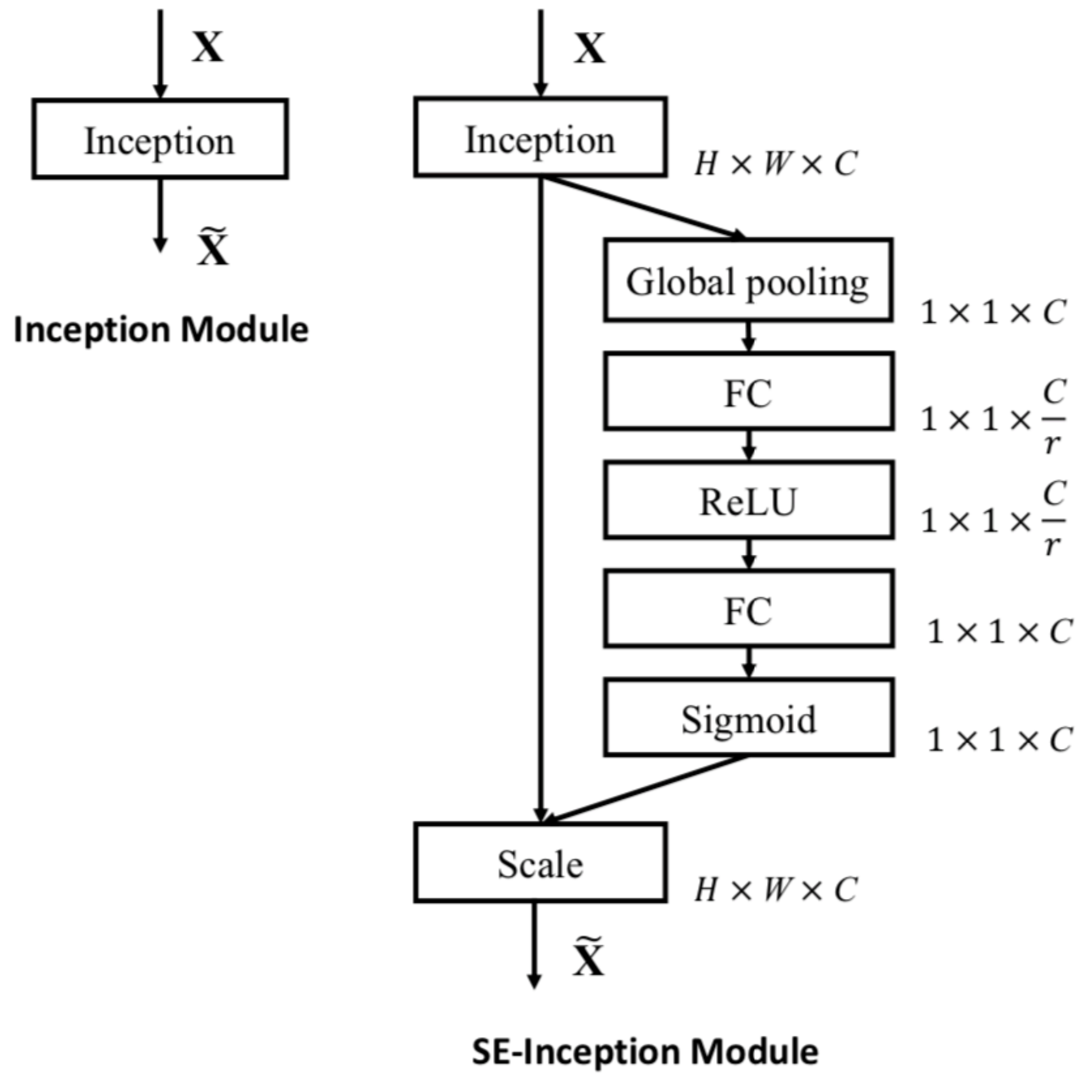
$\underline{r \text{ : reduction ratio}}$

To reduce FC layer parameters

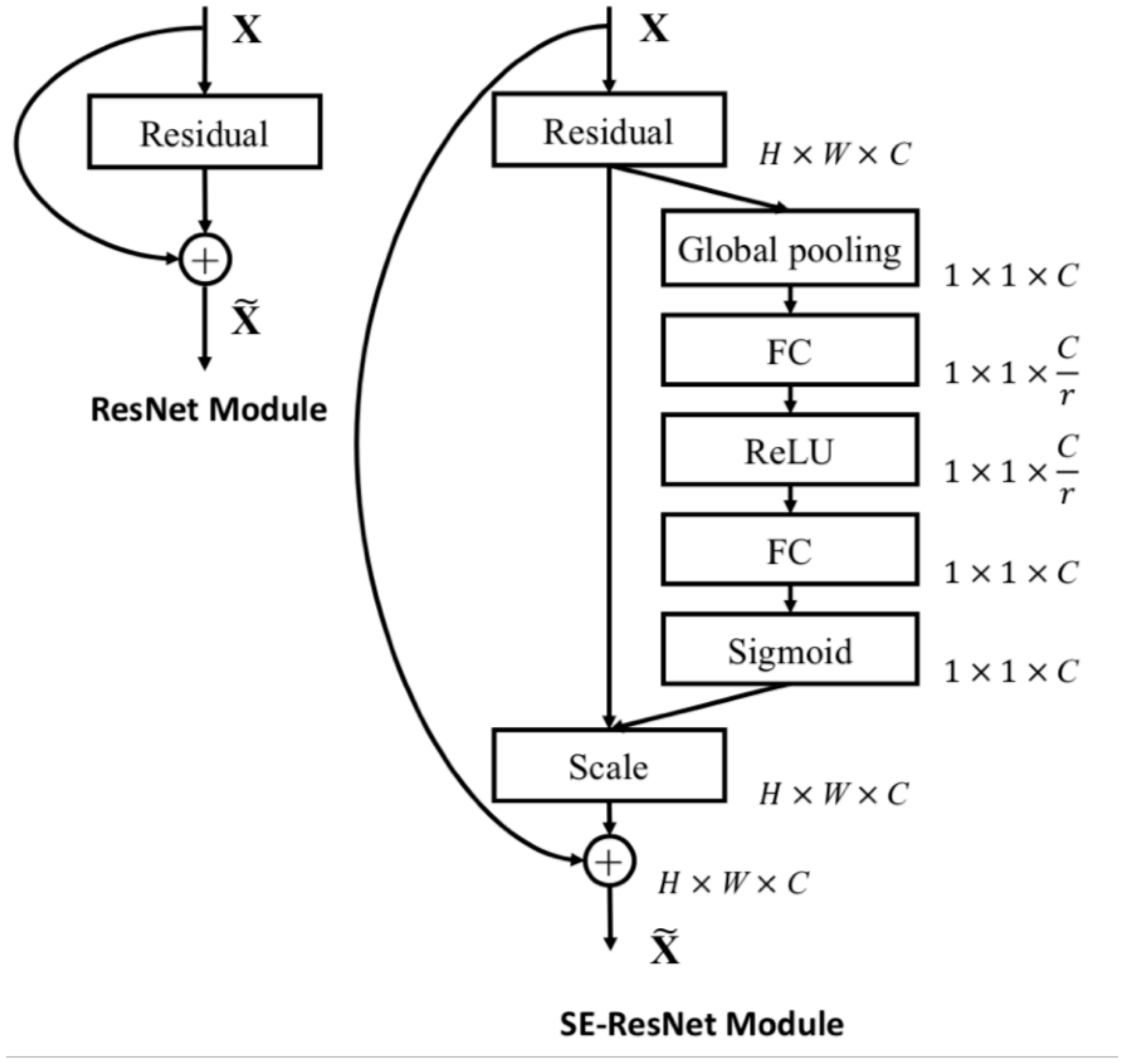
$$\widetilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \cdot \mathbf{u}_c,$$

ADVATAGES

Flexible !



X
Inception
X̃
Inception Module
X
Inception
$H \times W \times C$
Global pooling
$1 \times 1 \times C$
FC
$1 \times 1 \times \frac{C}{r}$
ReLU
$1 \times 1 \times \frac{C}{r}$
FC
$1 \times 1 \times C$
Sigmoid
$1 \times 1 \times C$
Scale
$H \times W \times C$
X̃
SE-Inception Module

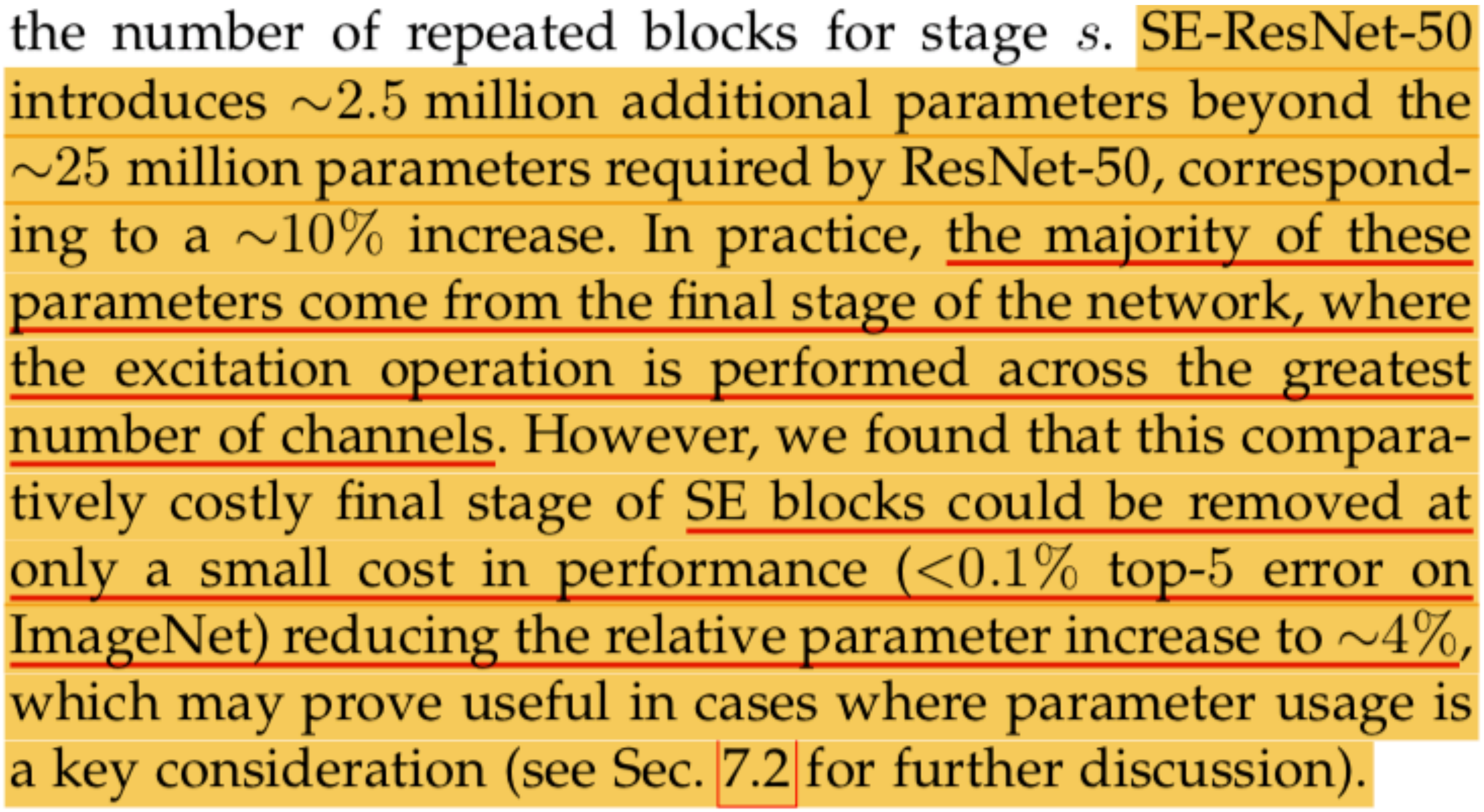


X
Residual
X̃
ResNet Module
X
Residual
$H \times W \times C$
Global pooling
$1 \times 1 \times C$
FC
$1 \times 1 \times \frac{C}{r}$
ReLU
$1 \times 1 \times \frac{C}{r}$
FC
$1 \times 1 \times C$
Sigmoid
$1 \times 1 \times C$
Scale
$H \times W \times C$
$H \times W \times C$
X̃
SE-ResNet Module

Additional Parameters

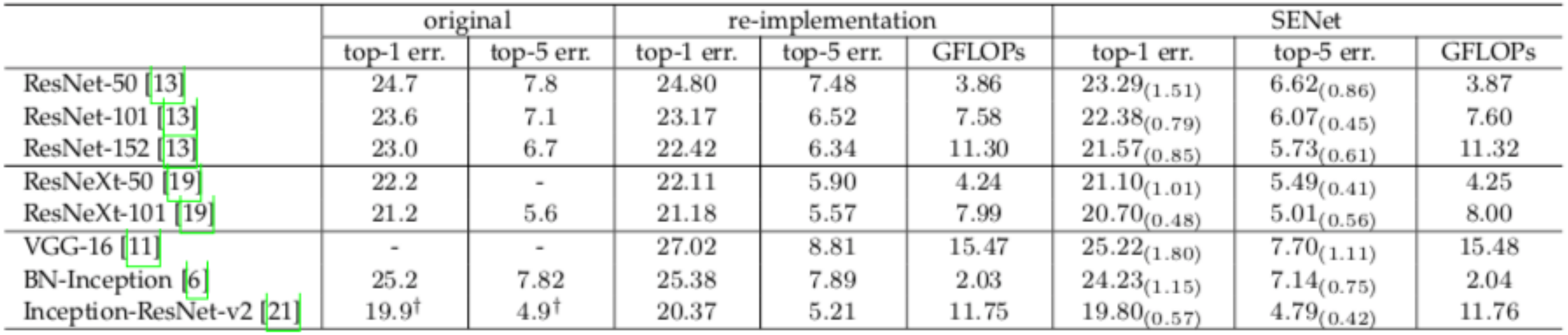
$$\frac{2}{r}\sum_{s=1}^{S} N_s \cdot {C_s}^2,$$

S : the number of stages
C: dimension of the output channels
N : the number of repeated blocks for stage s



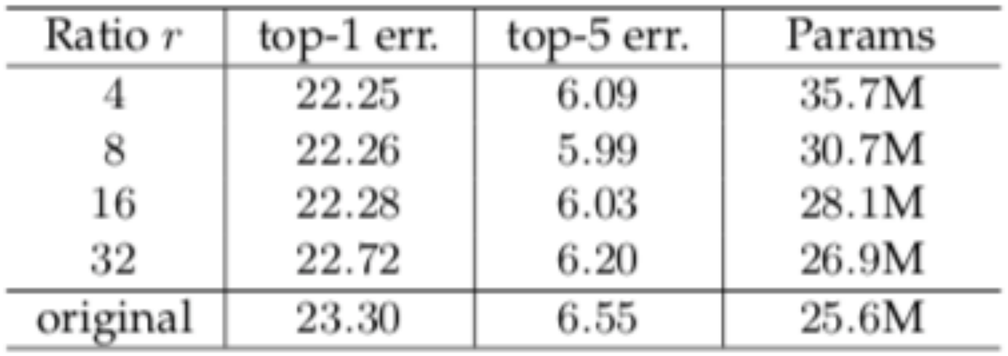
the number of repeated blocks for stage s. SE-ResNet-50 introduces ∼2.5 million additional parameters beyond the ∼25 million parameters required by ResNet-50, corresponding to a ∼10% increase. In practice, the majority of these parameters come from the final stage of the network, where the excitation operation is performed across the greatest number of channels. However, we found that this comparatively costly final stage of SE blocks could be removed at only a small cost in performance (<0.1% top-5 error on ImageNet) reducing the relative parameter increase to ∼4%, which may prove useful in cases where parameter usage is a key consideration (see Sec. 7.2 for further discussion).

Computational Complexity



	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [13]	23.0	6.7	22.42	6.34	11.30	$21.57_{(0.85)}$	$5.73_{(0.61)}$	11.32
ResNeXt-50 [19]	22.2	-	22.11	5.90	4.24	$21.10_{(1.01)}$	$5.49_{(0.41)}$	4.25
ResNeXt-101 [19]	21.2	5.6	21.18	5.57	7.99	$20.70_{(0.48)}$	$5.01_{(0.56)}$	8.00
VGG-16 [11]	-	-	27.02	8.81	15.47	$25.22_{(1.80)}$	$7.70_{(1.11)}$	15.48
BN-Inception [6]	25.2	7.82	25.38	7.89	2.03	$24.23_{(1.15)}$	$7.14_{(0.75)}$	2.04
Inception-ResNet-v2 [21]	$19.9^{\dagger}$	$4.9^{\dagger}$	20.37	5.21	11.75	$19.80_{(0.57)}$	$4.79_{(0.42)}$	11.76

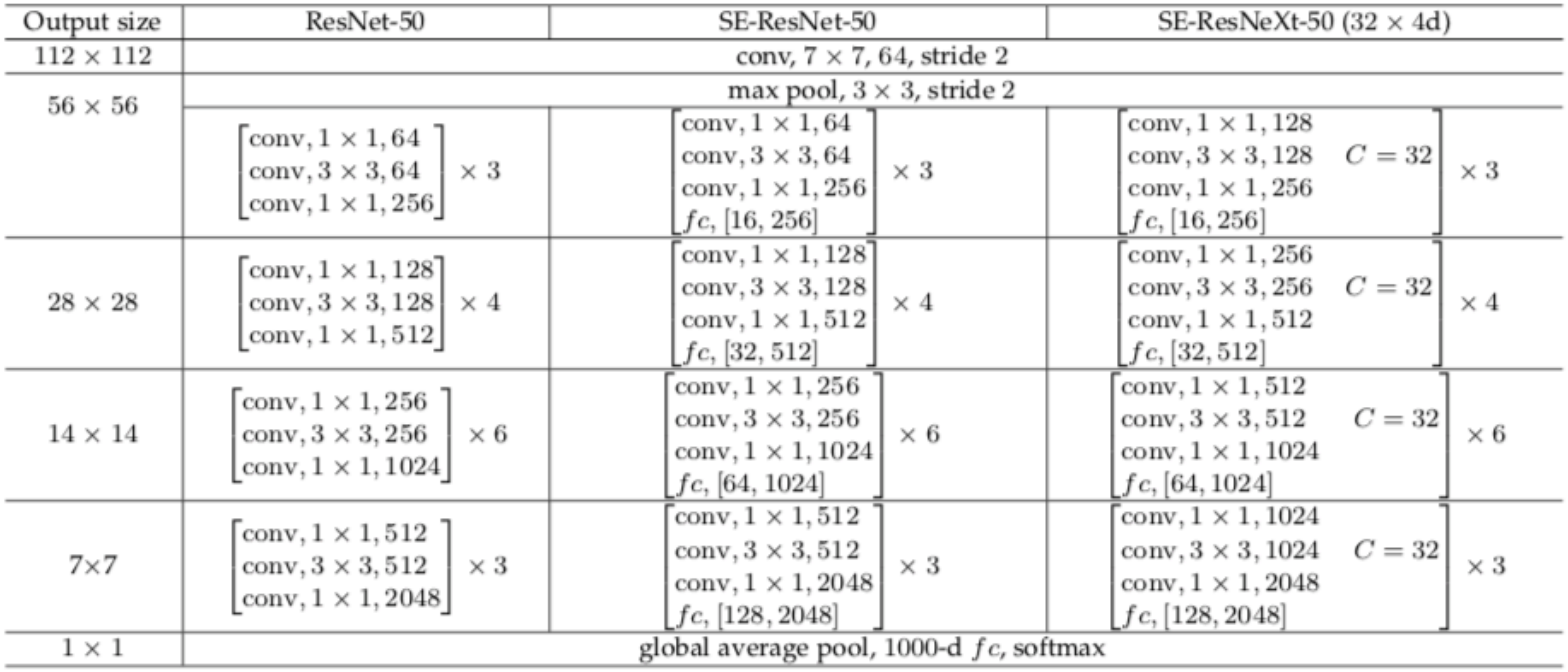
TABLE 10
Single-crop error rates (%) on ImageNet validation set and parameter sizes for SE-ResNet-50 at different reduction ratios r. Here, *original* refers to ResNet-50.



Ratio r	top-1 err.	top-5 err.	Params
4	22.25	6.09	35.7M
8	22.26	5.99	30.7M
16	22.28	6.03	28.1M
32	22.72	6.20	26.9M
original	23.30	6.55	25.6M

Comparisons

TABLE 1
(Left) ResNet-50. (Middle) SE-ResNet-50. (Right) SE-ResNeXt-50 with a 32×4d template. The shapes and operations with specific parameter settings of a residual building block are listed inside the brackets and the number of stacked blocks in a stage is presented outside. The inner brackets following by *fc* indicates the output dimension of the two fully connected layers in an SE module.



Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
112×112	conv, 7×7, 64, stride 2		
56×56	max pool, 3×3, stride 2		
	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \\ \text{conv}, 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \\ \text{conv}, 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 & \\ \text{conv}, 3 \times 3, 128 & C = 32 \\ \text{conv}, 1 \times 1, 256 & \\ fc, [16, 256] & \end{bmatrix} \times 3$
28×28	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 \\ \text{conv}, 3 \times 3, 128 \\ \text{conv}, 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 \\ \text{conv}, 3 \times 3, 128 \\ \text{conv}, 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 & \\ \text{conv}, 3 \times 3, 256 & C = 32 \\ \text{conv}, 1 \times 1, 512 & \\ fc, [32, 512] & \end{bmatrix} \times 4$
14×14	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 \\ \text{conv}, 3 \times 3, 256 \\ \text{conv}, 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 \\ \text{conv}, 3 \times 3, 256 \\ \text{conv}, 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 & \\ \text{conv}, 3 \times 3, 512 & C = 32 \\ \text{conv}, 1 \times 1, 1024 & \\ fc, [64, 1024] & \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 \\ \text{conv}, 3 \times 3, 512 \\ \text{conv}, 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 \\ \text{conv}, 3 \times 3, 512 \\ \text{conv}, 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 1024 & \\ \text{conv}, 3 \times 3, 1024 & C = 32 \\ \text{conv}, 1 \times 1, 2048 & \\ fc, [128, 2048] & \end{bmatrix} \times 3$
1×1	global average pool, 1000-d fc, softmax		

Comparisons

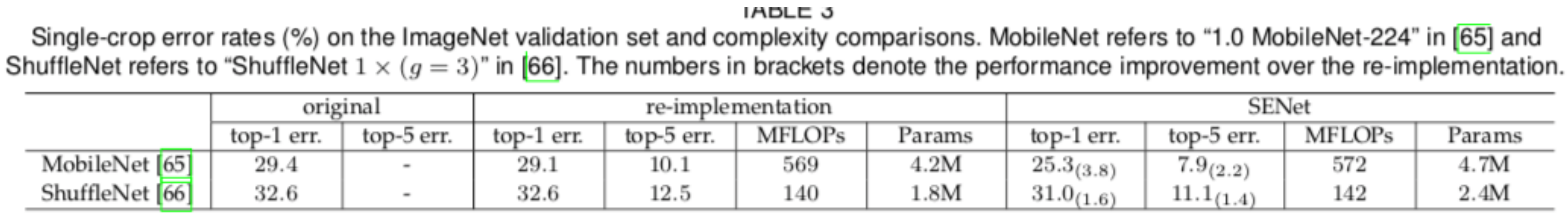


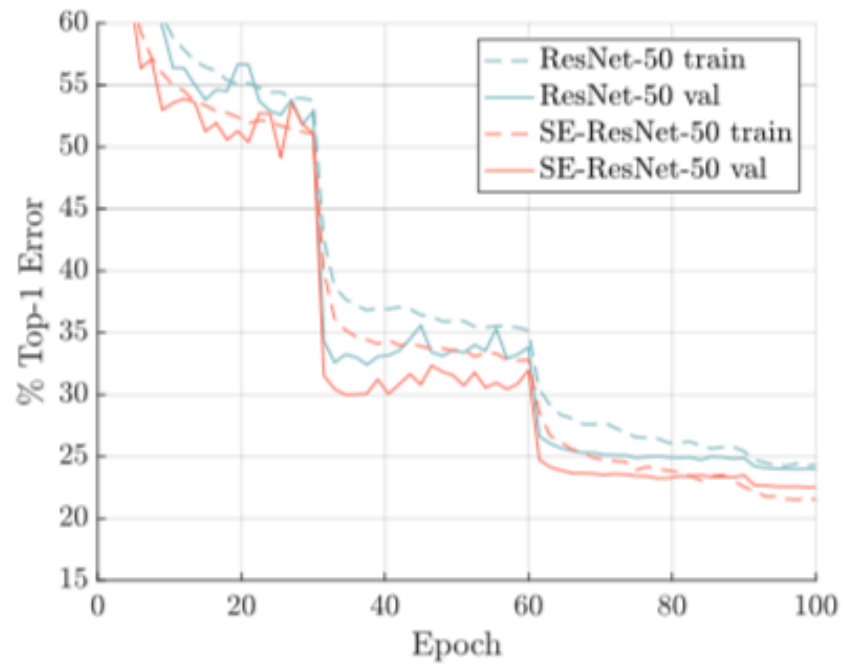
TABLE 3

Single-crop error rates (%) on the ImageNet validation set and complexity comparisons. MobileNet refers to “1.0 MobileNet-224” in [65] and ShuffleNet refers to “ShuffleNet $1 \times (g = 3)$” in [66]. The numbers in brackets denote the performance improvement over the re-implementation.

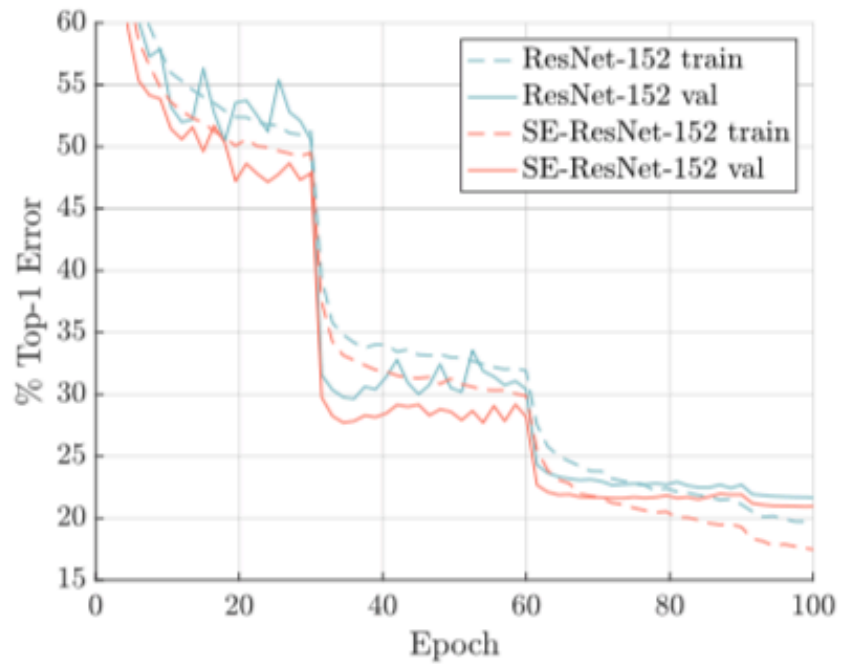
	original		re-implementation				SENet			
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	MFLOPs	Params	top-1 err.	top-5 err.	MFLOPs	Params
MobileNet [65]	29.4	-	29.1	10.1	569	4.2M	$25.3_{(3.8)}$	$7.9_{(2.2)}$	572	4.7M
ShuffleNet [66]	32.6	-	32.6	12.5	140	1.8M	$31.0_{(1.6)}$	$11.1_{(1.4)}$	142	2.4M

Experiments & Results

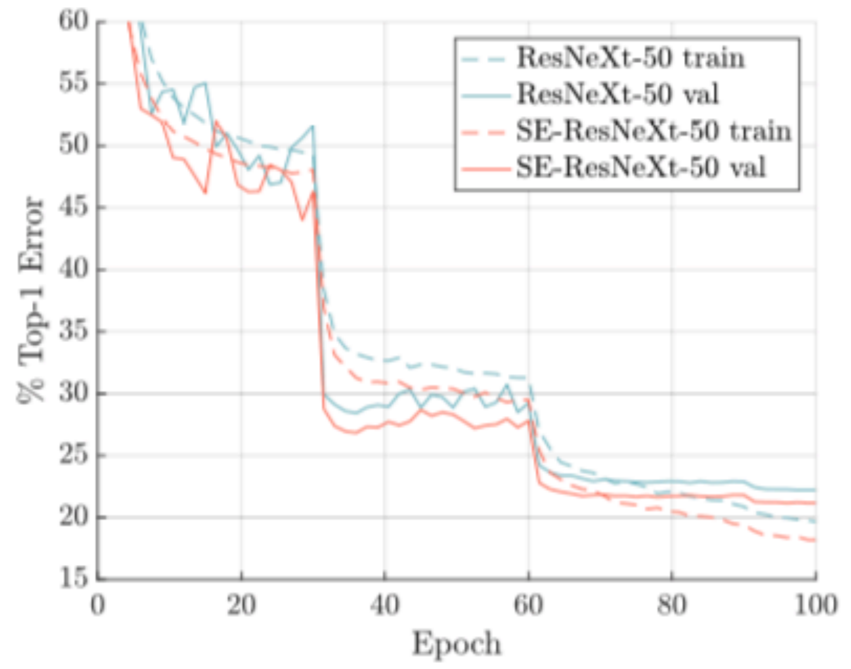
Experiments & Results



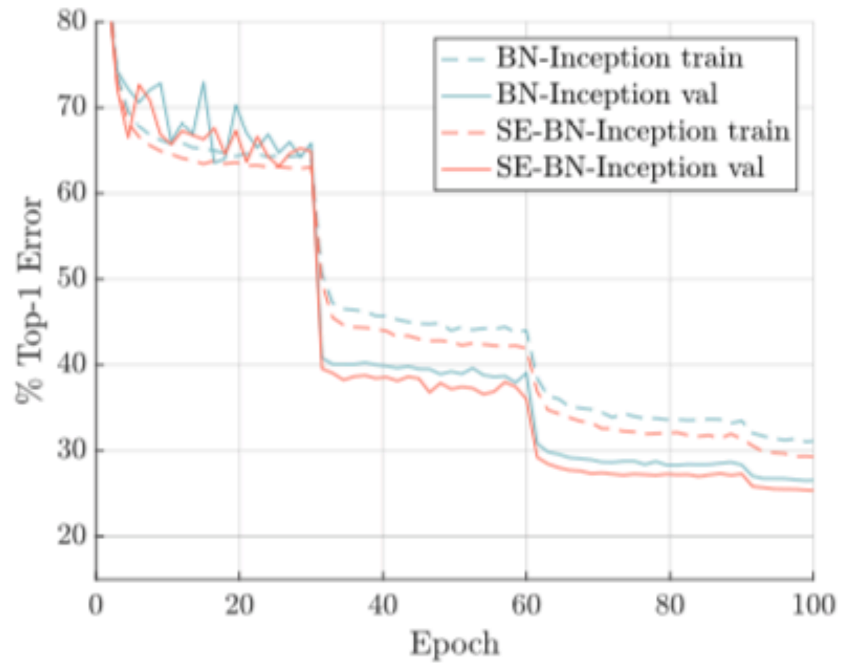
(a) ResNet-50 and SE-ResNet-50



(b) ResNet-152 and SE-ResNet-152



(c) ResNeXt-50 and SE-ResNeXt-50



(d) BN-Inception and SE-BN-Inception

Experiments & Results

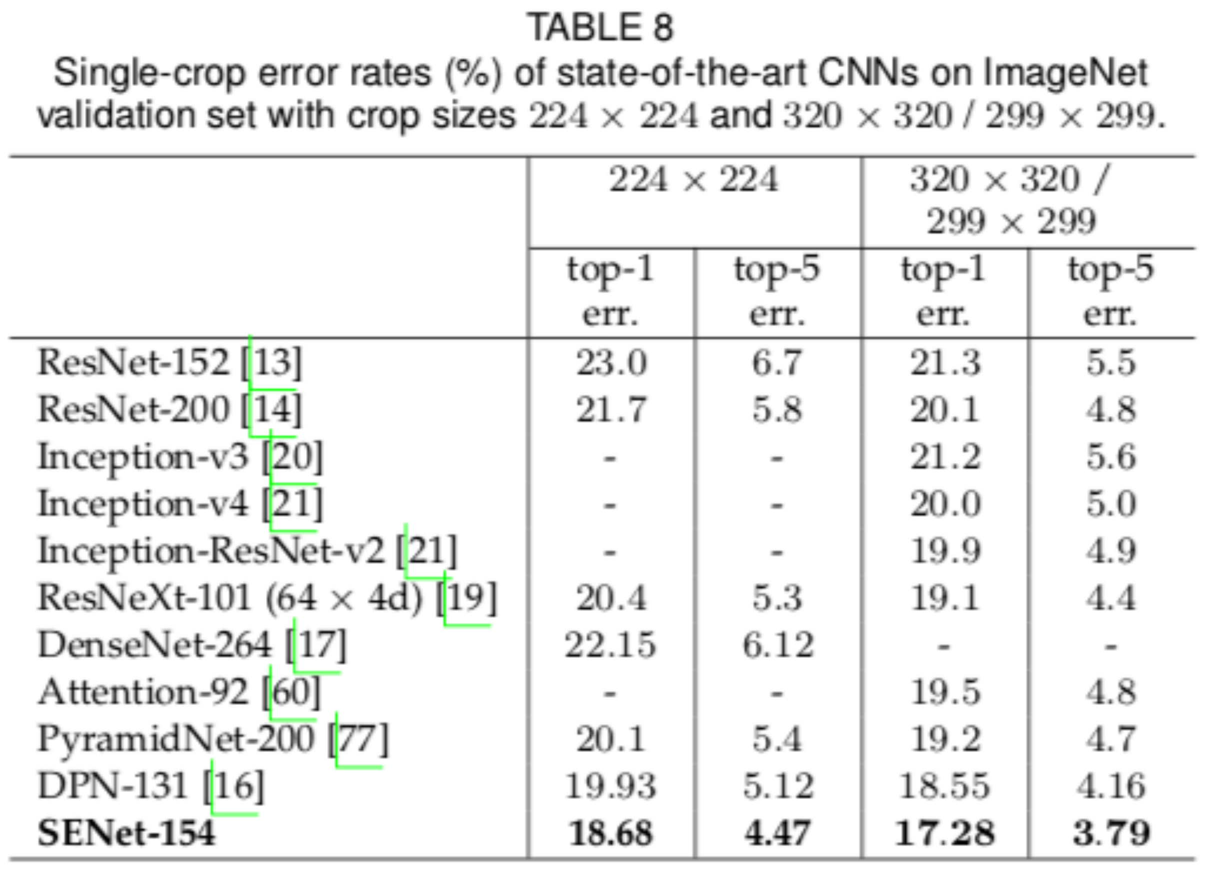


TABLE 8
Single-crop error rates (%) of state-of-the-art CNNs on ImageNet validation set with crop sizes 224×224 and 320×320 / 299×299.

	224×224		320×320 / 299×299	
	top-1 err.	top-5 err.	top-1 err.	top-5 err.
ResNet-152 [13]	23.0	6.7	21.3	5.5
ResNet-200 [14]	21.7	5.8	20.1	4.8
Inception-v3 [20]	-	-	21.2	5.6
Inception-v4 [21]	-	-	20.0	5.0
Inception-ResNet-v2 [21]	-	-	19.9	4.9
ResNeXt-101 ($64 \times 4d$) [19]	20.4	5.3	19.1	4.4
DenseNet-264 [17]	22.15	6.12	-	-
Attention-92 [60]	-	-	19.5	4.8
PyramidNet-200 [77]	20.1	5.4	19.2	4.7
DPN-131 [16]	19.93	5.12	18.55	4.16
SENet-154	**18.68**	**4.47**	**17.28**	**3.79**

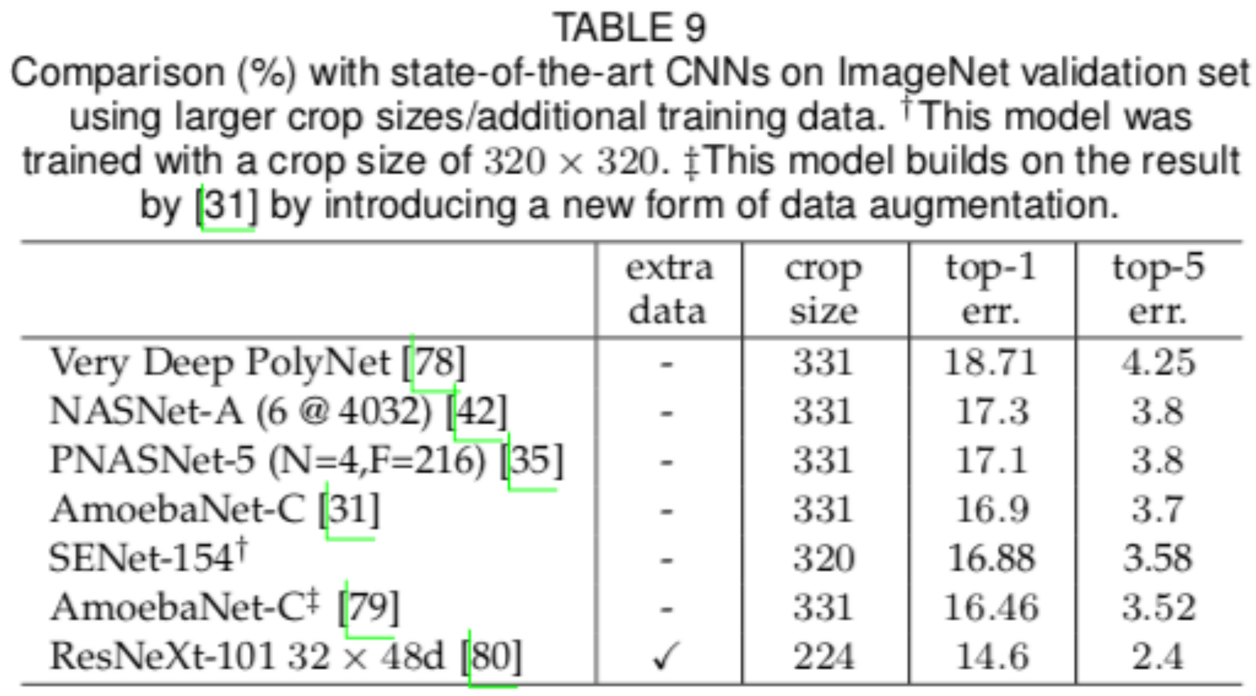


TABLE 9
Comparison (%) with state-of-the-art CNNs on ImageNet validation set using larger crop sizes/additional training data. †This model was trained with a crop size of 320×320. ‡This model builds on the result by [31] by introducing a new form of data augmentation.

	extra data	crop size	top-1 err.	top-5 err.
Very Deep PolyNet [78]	-	331	18.71	4.25
NASNet-A (6 @ 4032) [42]	-	331	17.3	3.8
PNASNet-5 (N=4,F=216) [35]	-	331	17.1	3.8
AmoebaNet-C [31]	-	331	16.9	3.7
SENet-154†	-	320	16.88	3.58
AmoebaNet-C‡ [79]	-	331	16.46	3.52
ResNeXt-101 $32 \times 48d$ [80]	✓	224	14.6	2.4

Experiments & Results

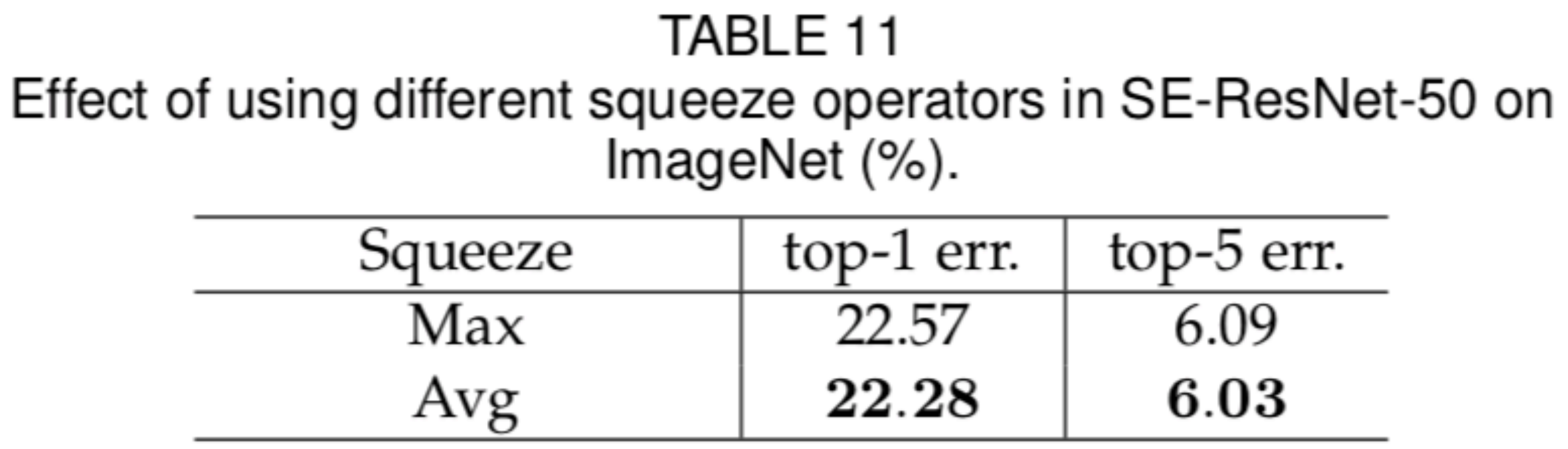


TABLE 11
Effect of using different squeeze operators in SE-ResNet-50 on ImageNet (%).

Squeeze	top-1 err.	top-5 err.
Max	22.57	6.09
Avg	**22.28**	**6.03**

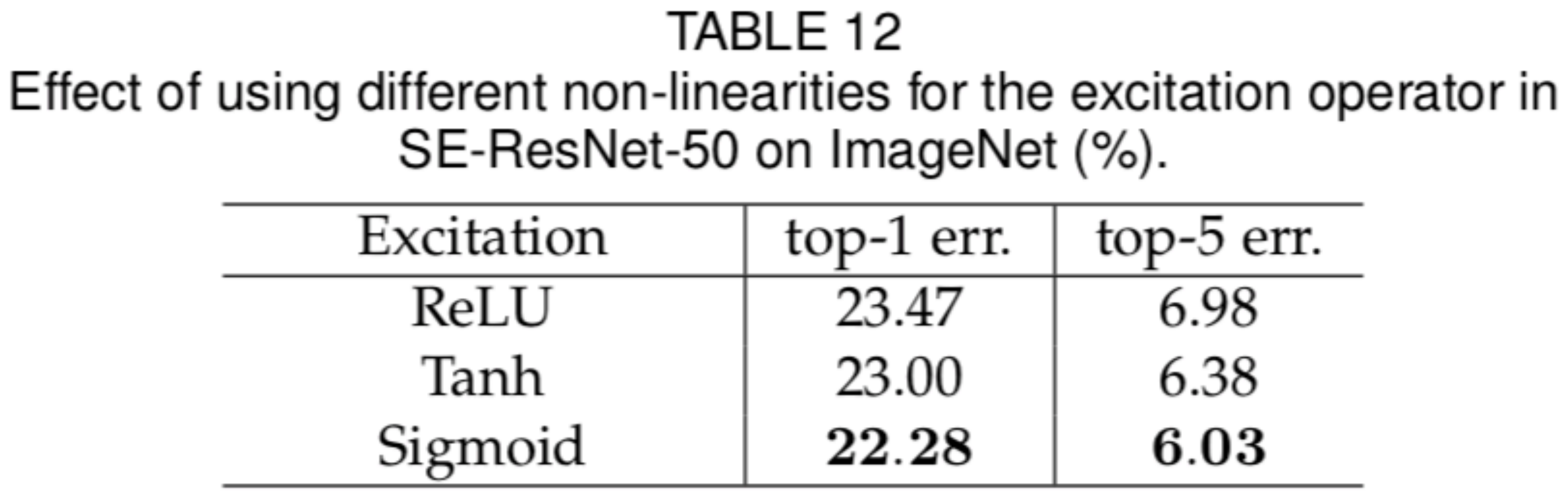
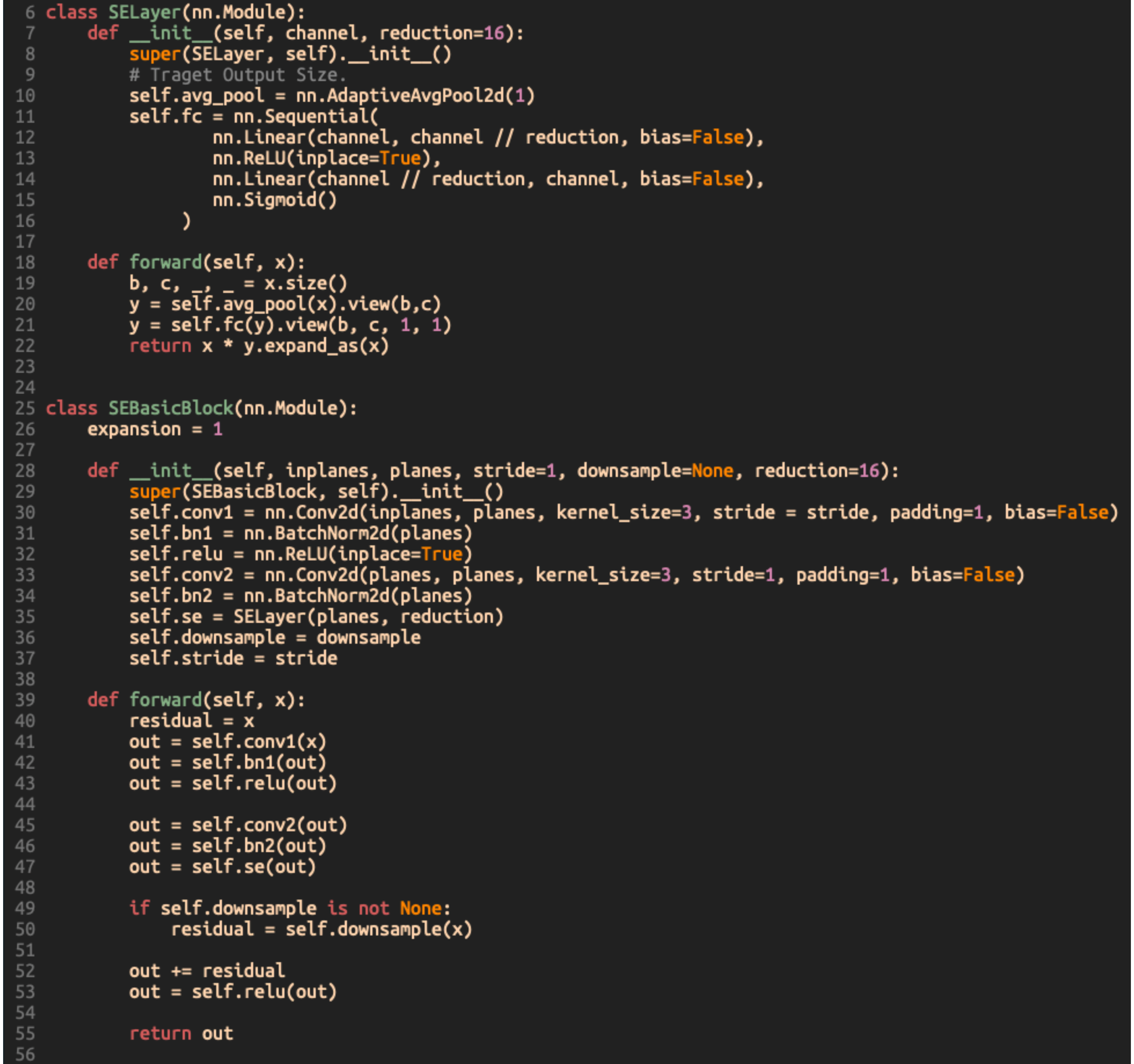


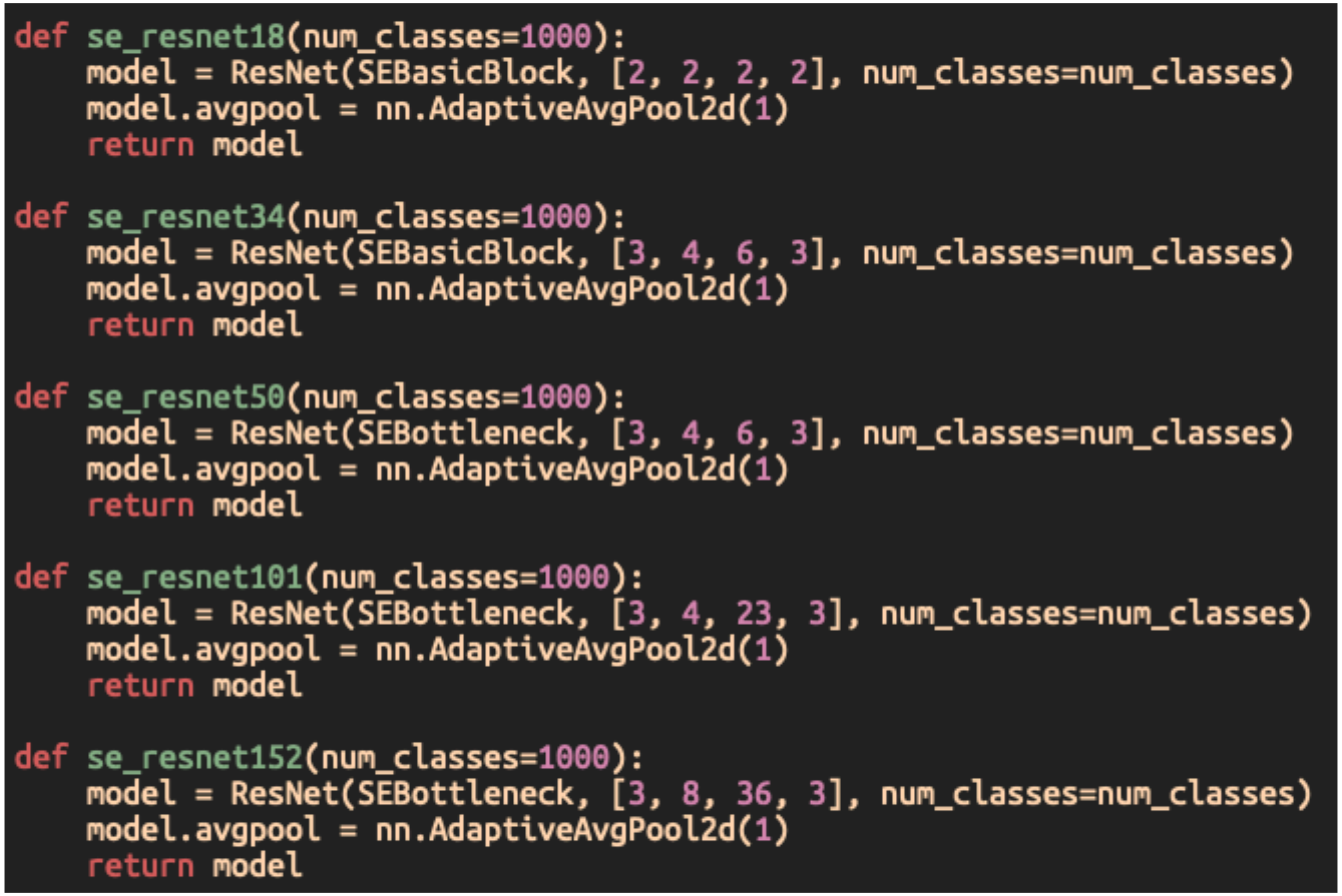
TABLE 12
Effect of using different non-linearities for the excitation operator in SE-ResNet-50 on ImageNet (%).

Excitation	top-1 err.	top-5 err.
ReLU	23.47	6.98
Tanh	23.00	6.38
Sigmoid	**22.28**	**6.03**

Let’s see with Code



```python
class SELayer(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SELayer, self).__init__()
        # Traget Output Size.
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
                nn.Linear(channel, channel // reduction, bias=False),
                nn.ReLU(inplace=True),
                nn.Linear(channel // reduction, channel, bias=False),
                nn.Sigmoid()
             )

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b,c)
        y = self.fc(y).view(b, c, 1, 1)
        return x * y.expand_as(x)


class SEBasicBlock(nn.Module):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, downsample=None, reduction=16):
        super(SEBasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(inplanes, planes, kernel_size=3, stride = stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.se = SELayer(planes, reduction)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.se(out)

        if self.downsample is not None:
            residual = self.downsample(x)

        out += residual
        out = self.relu(out)

        return out

```



```python
def se_resnet18(num_classes=1000):
    model = ResNet(SEBasicBlock, [2, 2, 2, 2], num_classes=num_classes)
    model.avgpool = nn.AdaptiveAvgPool2d(1)
    return model

def se_resnet34(num_classes=1000):
    model = ResNet(SEBasicBlock, [3, 4, 6, 3], num_classes=num_classes)
    model.avgpool = nn.AdaptiveAvgPool2d(1)
    return model

def se_resnet50(num_classes=1000):
    model = ResNet(SEBottleneck, [3, 4, 6, 3], num_classes=num_classes)
    model.avgpool = nn.AdaptiveAvgPool2d(1)
    return model

def se_resnet101(num_classes=1000):
    model = ResNet(SEBottleneck, [3, 4, 23, 3], num_classes=num_classes)
    model.avgpool = nn.AdaptiveAvgPool2d(1)
    return model

def se_resnet152(num_classes=1000):
    model = ResNet(SEBottleneck, [3, 8, 36, 3], num_classes=num_classes)
    model.avgpool = nn.AdaptiveAvgPool2d(1)
    return model
```