Advanced CNN

2021.10.25

LG

KAIST AI
Graduate School of AI

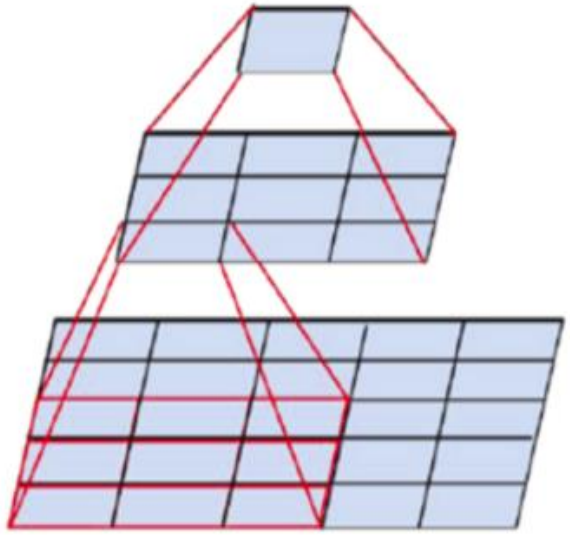
Contents

- VGGNet
  - VGGNet 구현
- Residual Network
  - ResNet 구현
- MobileNet
  - MobileNet 구현
- Recent CNN

VGGNet

- Very Deep Convolutional Networks for Large-Scale Image Recognition

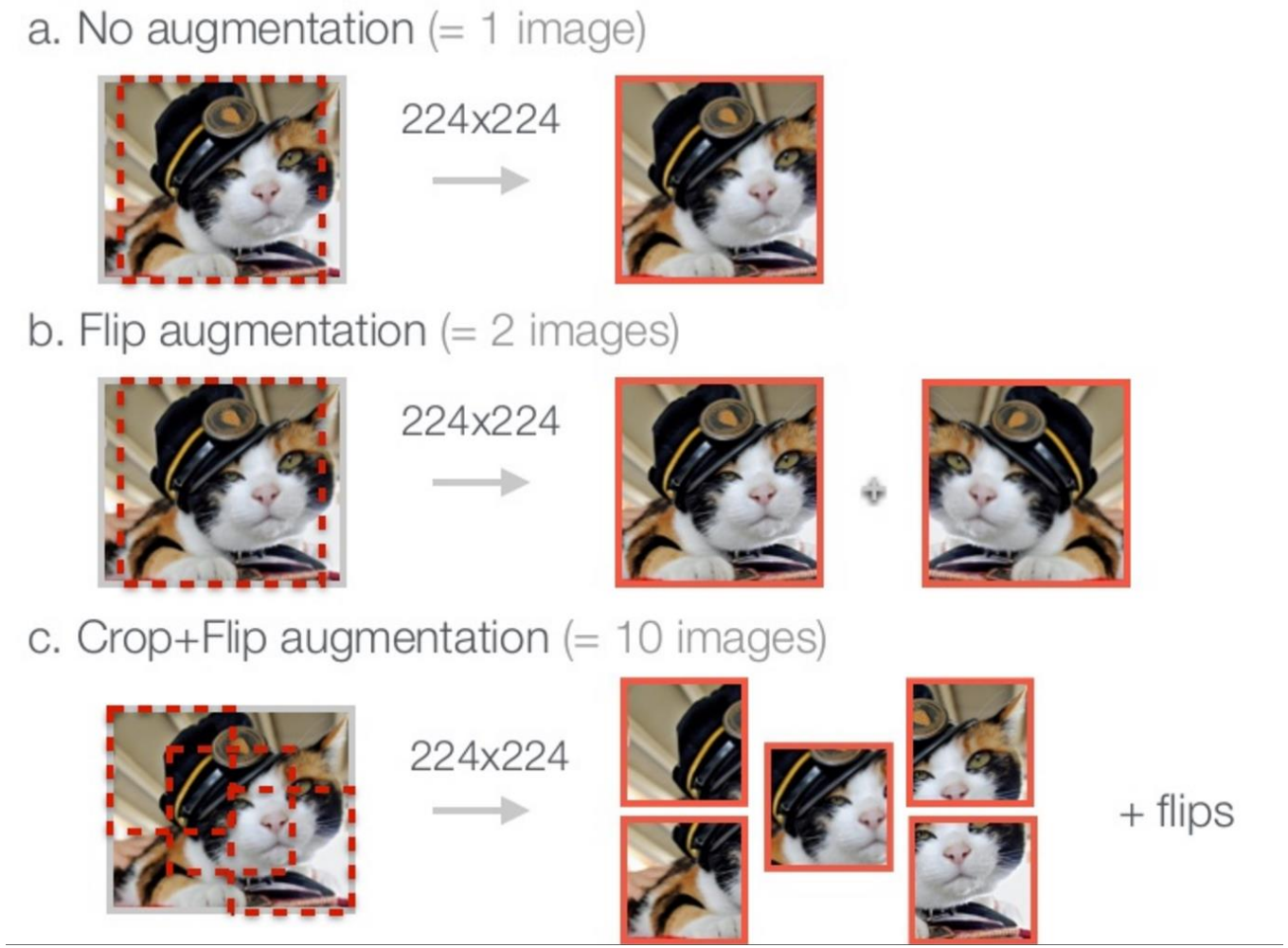
- 1. 깊이를 증가하면 정확도가 좋아진다.
- 2. 3×3 filter를 여러 겹 사용하여 3×3, 3×3 filter를 분해하면 추가적인 비선형성을 부여하고 parameter 수를 감소시킨다.



- 3. pre-initialization을 이용하면 모델이 빠르게 수렴한다.
- 4. data augmentation (resize, crop, flip)을 적용하면 다양한 scale로 feature를 포착할 수 있다.

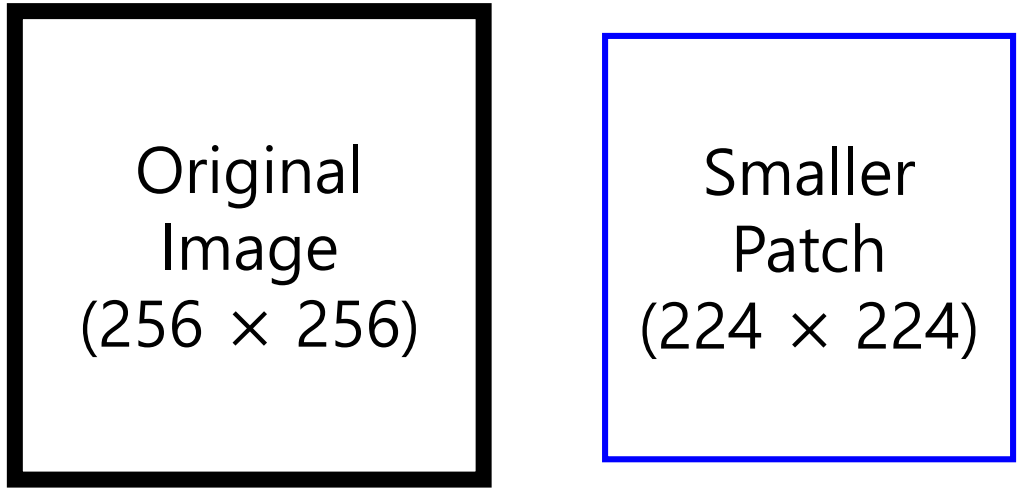
VGGNet

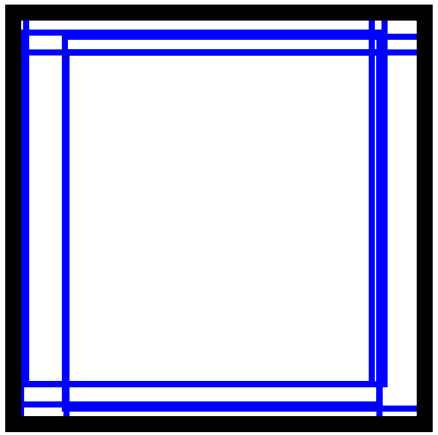
- Data augmentation



VGGNet

- Data augmentation





This increases the size of the training set by a **factor of 2048** $(32 * 32 * 2)$.
Two comes from horizontal reflections.

VGGNet

- 모든 파라미터를 0으로 세팅하게 된다면?
- 가중치가 0이기 때문에 모든 뉴런은 다 같은 연산을 할 것입니다.
- 결국 gradient가 같을 것이고 모든 뉴런이 똑같은 업데이트 될 것입니다.

- 이를 해결하기 위해 적절한 값으로 초기화를 시켜줘야 한다.
- Xavier Initialization

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

- He Initialization

$$W \sim N(0, Var(W))$$

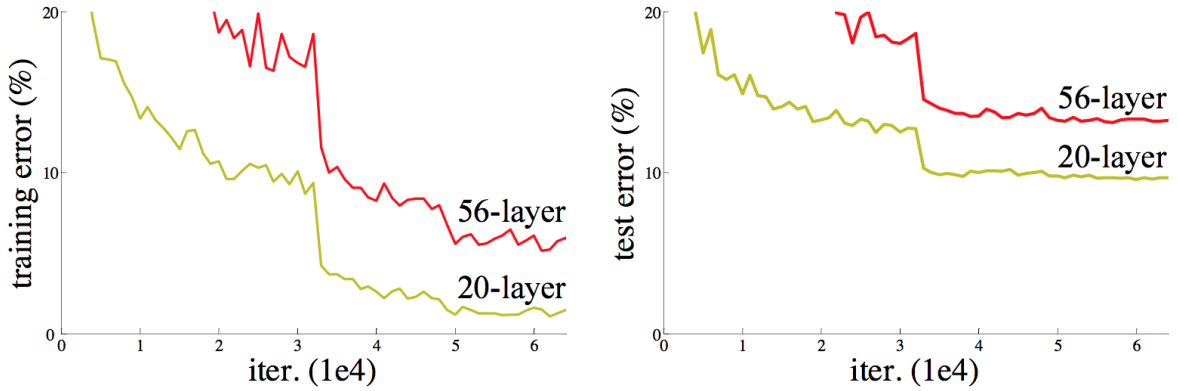
$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

```python
# 가중치 초기화 함수
def _initialize_weights(self):
    # .modules로 sequential에 있는 layer를 하나하나 불러올 수 있습니다.
    for m in self.modules():
        # isinstance 함수로 m이 nn.Conv2d인지 확인합니다.
        if isinstance(m, nn.Conv2d):
            # He 가중치 초기화(relu에 최적화되어있는 가중치 초기화 기법)
            nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
            if m.bias is not None:
                nn.init.constant_(m.bias, 0)
        elif isinstance(m, nn.BatchNorm2d):
            nn.init.constant_(m.weight, 1)
            nn.init.constant_(m.bias, 0)
        elif isinstance(m, nn.Linear):
            nn.init.normal_(m.weight, 0, 0.01)
            nn.init.constant_(m.bias, 0)
```

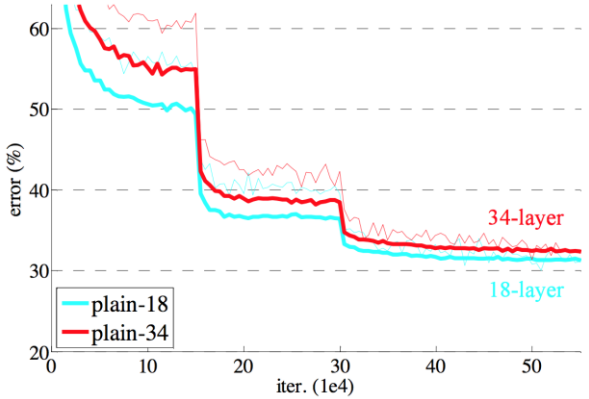
Residual Network

- Is deeper network always better?
- **Degradation problem**

- CIFAR 100 Dataset

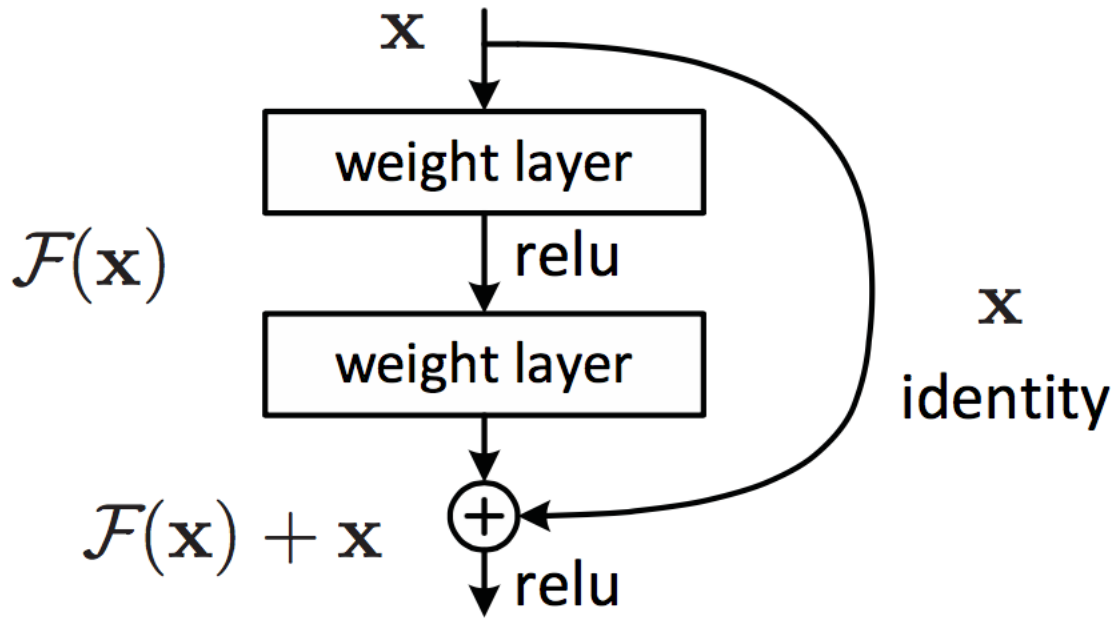


- ImageNet Dataset



Residual Network

- Residual learning building block
- Authors **hypothesize** that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping
- **Shortcut connections** are used.



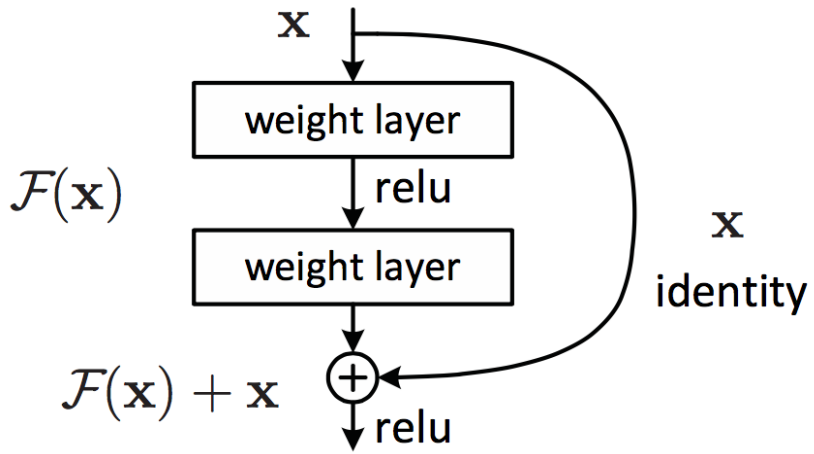
Residual Network

- “The extremely deep residual nets are easy to optimize.”
- “The deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks.”

Residual Networks

- Residual mapping
- Basic residual mapping (same dim.)

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

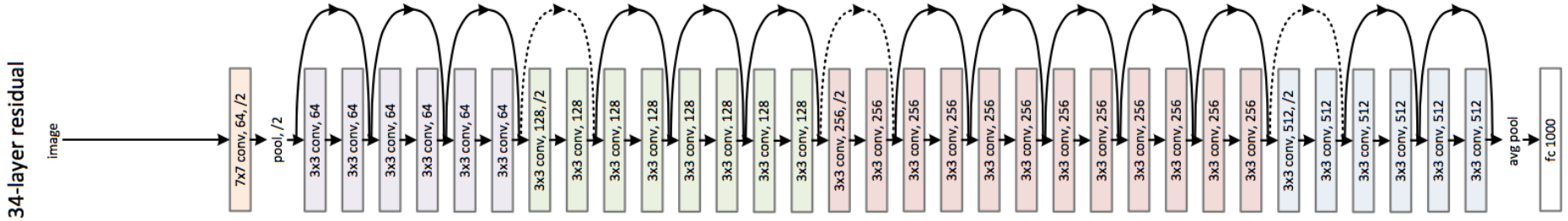


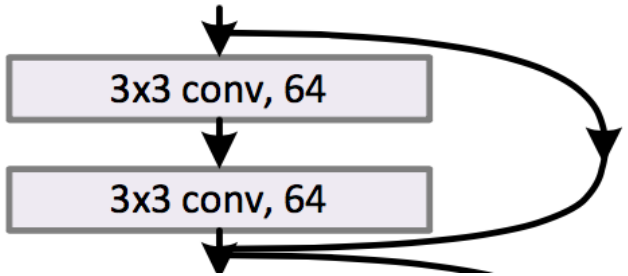
$$\mathcal{F} = W_2\sigma(W_1\mathbf{x})$$

- Basic residual mapping (different dim.)

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

Residual Networks

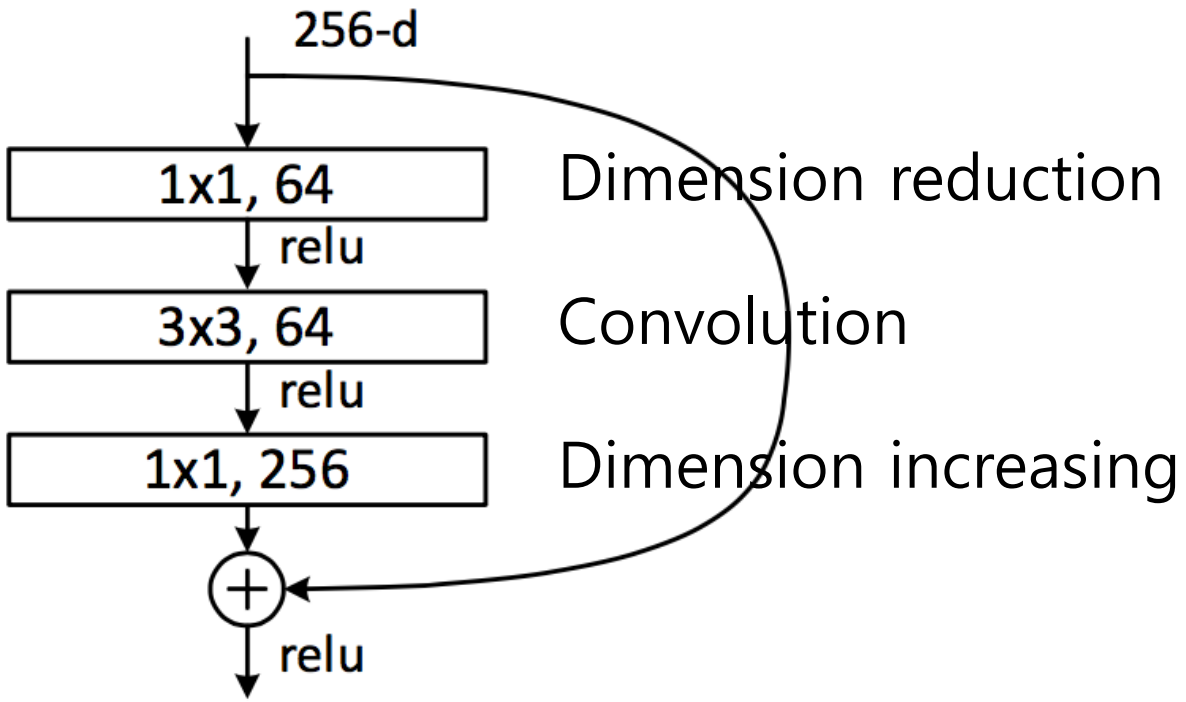





```python
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

Residual Networks

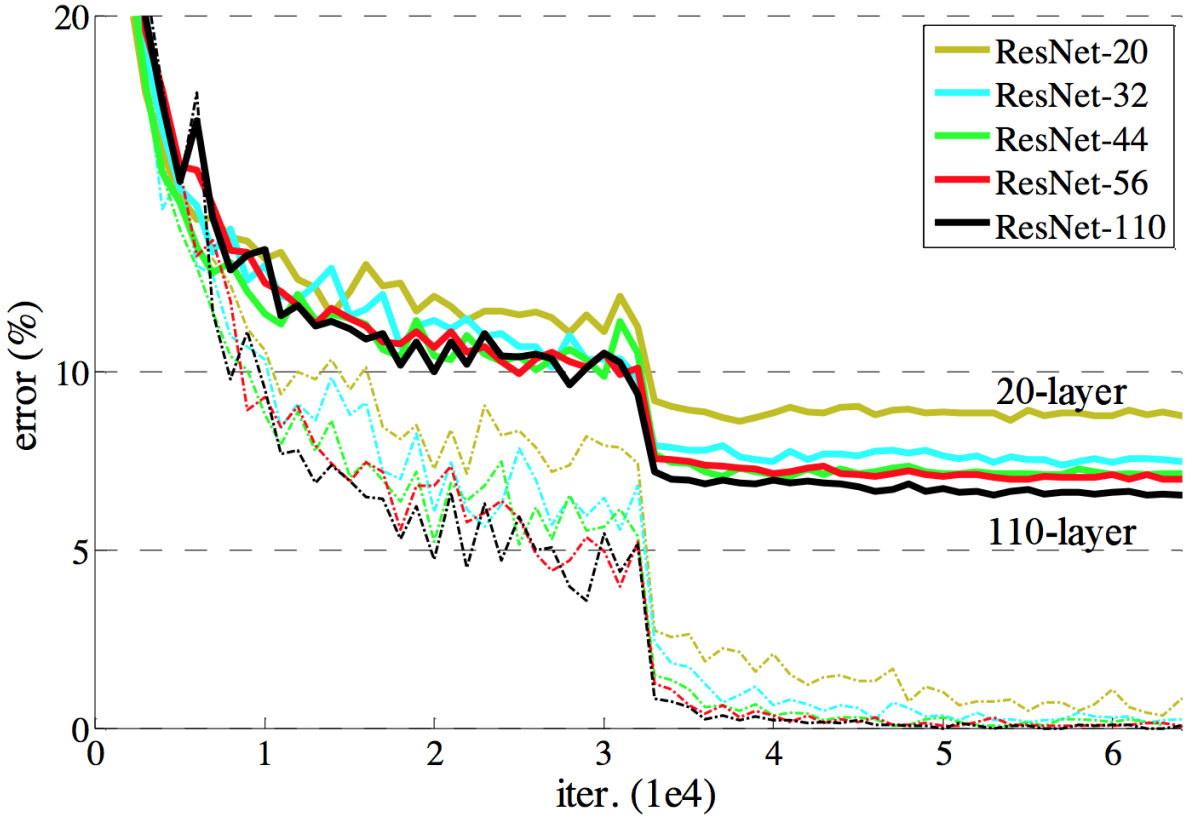
- Deeper bottlenet architecture




```python
class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_planes, planes, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion *
                               planes, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion*planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

Residual Networks

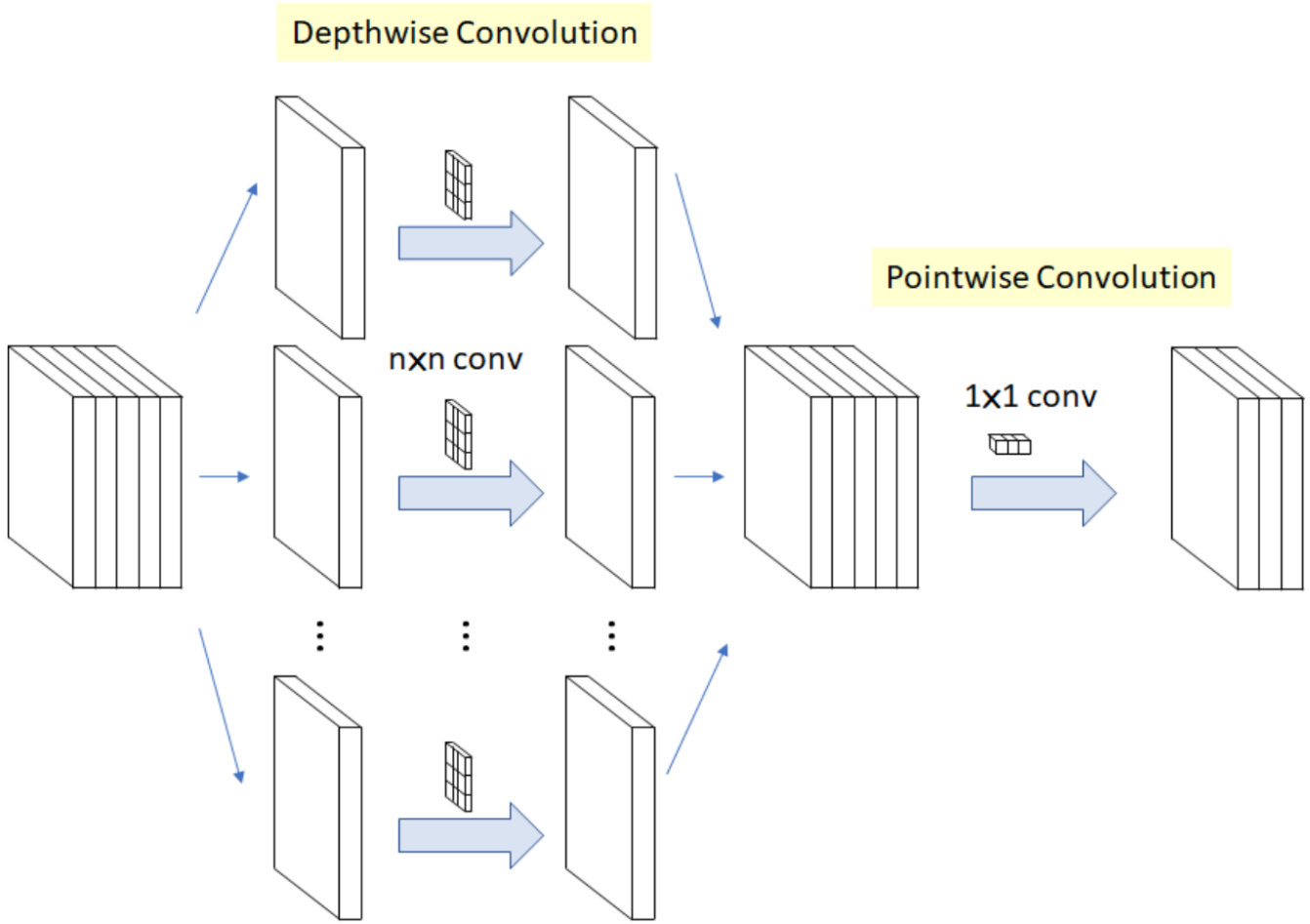
- Experimental results



MobileNet

- 단순히 채널 수를 증가시키며 convolution-pooling의 연속으로 구성.
- 가장 직관적인 구조로 이해도 쉽고 구현하기 쉬우나, 모바일 환경에서 구동 시키기에는 convolution 구조가 다소 무겁다.
- 파라미터의 수를 획기적으로 줄이기 위해서 MobileNet이라는 새로운 구조를 발표.

MobileNet



Original Depthwise Separable Convolution

MobileNet

- **Depth-wise convolution**
- 채널별로 분리하여 각 채널을 각각의 커널로 convolution 연산을 진행함.
- 이 때, 입력의 채널과 출력의 채널은 항상 동일함.

- **Point-wise convolution**
- 출력의 채널을 바꿀 수 있으며, 1×1 convolution filter를 이용하여 연산을 진행함.

- 기존 convolution의 경우
- $3 \times 3 \times 3$의 kernel이 3개이므로 파라미터 수는 $3 \times 3 \times 3 \times 3 = 81$.
- Depth-wise separable convolution의 경우
- $3 \times 3 \times 1$의 kernel이 3개 (depth-wise), $1 \times 1 \times 3$의 kernel이 3개이므로
- $3 \times 3 \times 1 \times 3 + 1 \times 1 \times 3 \times 3 = 27 + 9 = 36$

MobileNet

- 획기적으로 파라미터의 수가 줄었음에도 불구하고,
- VGG16과 비교하였을 때 성능은 크게 떨어지지 않는 실험 결과를 보임.

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Recent CNN (1) - Wide ResNet

- **Width vs. Depth**
- 신경망의 깊이 문제는 오래전부터 연구 되어오던 주제.
- ResNet을 설계한 연구자들은 최대한 weight를 적게 사용하면서도 깊은 망을 사용할 수 있도록 연구.
- 하지만 Identity mapping을 허용하는 residual block은 학습 시 약점이 되기도 함.
  - Gradient flow 과정 중에 residual block으로 gradient를 반드시 전달하지 않아도 되는 구조라서 실제 training 과정 중에 학습이 잘 안될 수 있음.
  - 따라서 **일부 block만이 유용한 정보들을 학습**.
  - 결국 대부분의 block이 정보를 가지고 있지 못하거나 많은 block들에 아주 적은 정보만 담긴 채 공유.
  - 저자들은 이 문제를 해결하기 위해 **residual block을 무작위로 비 활성화하는 방법을 제안**.
  - 이 방법은 dropout의 특별한 예로 볼 수 있으며 dropout이 적용되는 영역의 residual block에 identity scalar weight가 적용.
- 이 논문에서는 망의 깊이를 증가시키는 것보다 residual block을 개선하여 성능을 향상시킴.
  - 더 **넓은 residual block을 사용함**으로써 성능이 향상되는 것을 확인하였다.

Recent CNN (1) - Wide ResNet

- Wide Residual Networks

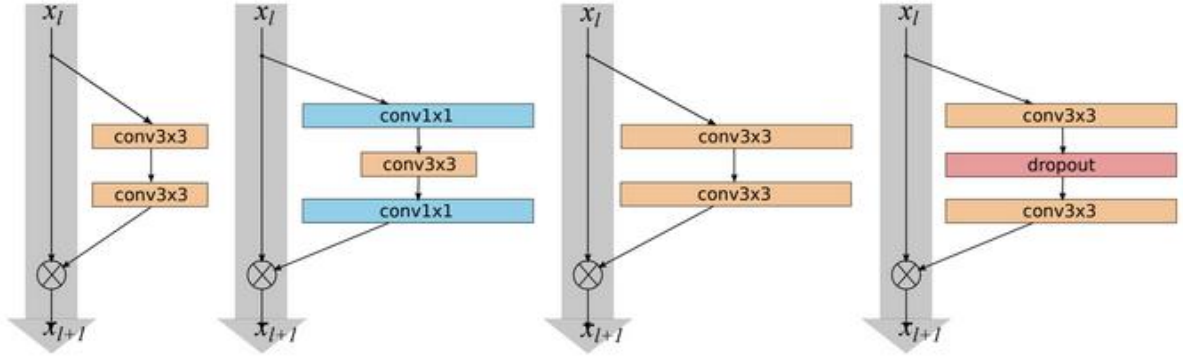


Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)

- Original ResNet과는 다르게 BN, ReLU 적용 순서를 다르게 하였다.
  - ResNet: conv - BN - ReLU
  - Wide ResNet: BN - ReLU - Conv
  - 이를 통해 학습이 더 빠르게 되고 더 좋은 결과를 얻는 것을 확인함.

Recent CNN (1) - Wide ResNet

- Dropout 적용하기
  - Dropout은 한 때 인기있는 기법이었으나 Batch Norm (BN) 등장 이후 사용 빈도가 줄어듦.
  - Dropout을 사용하면 regularization 효과가 생겨 성능이 증가한다는 사실은 이미 알려짐.
  - Wide ResNet의 경우 더 넓은 residual block을 사용하게 되므로 parameter의 수가 증가.
    - Dropout이 overfitting을 막아주게 되므로 이를 적용.
    - 이전 연구에 따르면 dropout을 identity 영역에 적용하면 성능이 더 하락한다는 결과가 있음.
    - 저자들은 대신 이를 convolution layer에 적용.
    - 실험결과 더 좋은 성능을 달성.

depth	k	dropout	CIFAR-10	CIFAR-100	SVHN
16	4		5.02	24.03	1.85
16	4	✓	5.24	23.91	1.64
28	10		4.00	19.25	-
28	10	✓	**3.89**	**18.85**	-
52	1		6.43	29.89	2.08
52	1	✓	6.28	29.78	1.70

Table 6: Effect of dropout in residual block. (mean/std preprocessing, CIFAR numbers are based on median of 5 runs)

Recent CNN (1) - Wide ResNet

- 다양한 폭 (width), 깊이 (depth), 파라미터 수 (# of params) 실험
- 동일한 depth에서는 k가 클수록 우수
- 동일한 k에서는 depth가 클수록 우수

depth	k	# params	CIFAR-10	CIFAR-100
40	1	0.6M	6.85	30.89
40	2	2.2M	5.33	26.04
40	4	8.9M	4.97	22.89
40	8	35.7M	4.66	-
28	10	36.5M	**4.17**	20.50
28	12	52.5M	4.33	**20.43**
22	8	17.2M	4.38	21.22
22	10	26.8M	4.44	20.75
16	8	11.0M	4.81	22.07
16	10	17.1M	4.56	21.59

Table 4: Test error (%) of various wide networks on CIFAR-10 and CIFAR-100 (ZCA pre-processing).

- WRN-28-10 모델이 PreAct ResNet-1001보다 0.92%만큼 성능이 좋다.

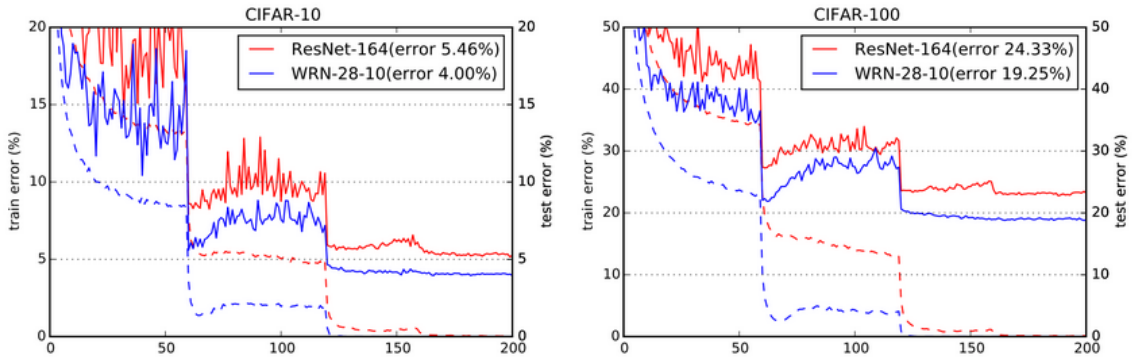


Figure 2: Training curves for thin and wide residual networks on CIFAR-10 and CIFAR-100. Solid lines denote test error (y-axis on the right), dashed lines denote training loss (y-axis on the left).

Recent CNN (2) - SENet

- SENet (Squeeze and Excitation Networks)
- Our goal is to improve the representational power of a network by explicitly modeling the interdependencies between the channels of its convolutional features.

- 1. 네트워크 어떤 곳이라도 바로 붙일 수 있음.
- 2. 파라미터 증가량에 비해 모델 성능 향상도가 매우 큼. 모델 복잡도 (Model complexity)와 계산 복잡도 (computational burden)이 크게 증가하지 않다는 장점이 있음.

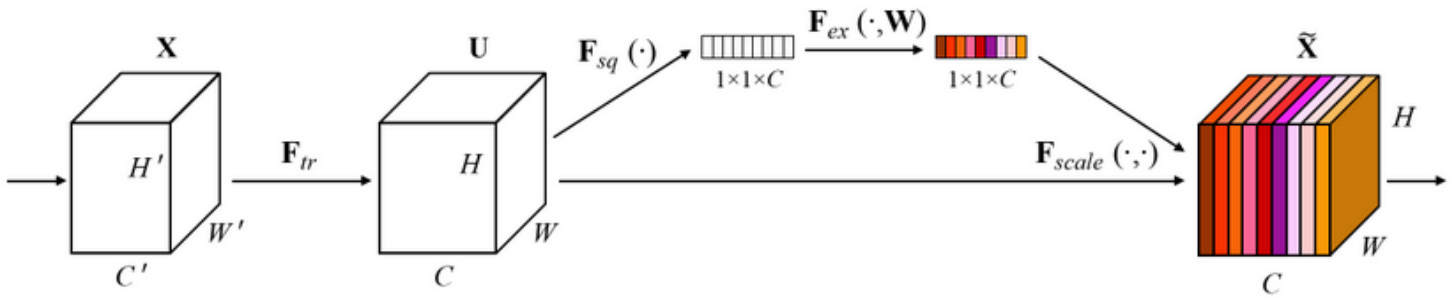


Figure 1: A Squeeze-and-Excitation block.

Recent CNN (2) - SENet

- **Squeeze: Global Information Embedding**
- Squeeze operation은 말 그대로 짜내는 연산을 담당.
- 각 채널들의 중요한 정보만 추출해서 가져가는 것과 동일.

- Local receptive field가 매우 작은 네트워크 하위 부분에서는 정보 추출이라는 개념이 중요.
- 중요 정보를 추출하는 가장 일반적인 방법 중 하나인 GAP (Global Average Pooling)을 사용.
- GAP를 사용하면 global spatial information을 channel descriptor로 압축.

$$z_c = F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j)$$

- H, W, C 크기의 feature map을 $1, 1, C$ 크기로 만든 것.
- GAP 대신 다른 정보 압축 방법론을 사용해도 된다고 언급.

Recent CNN (2) - SENet

- **Excitation: Adaptive Recalibration**
- Excitation operation은 채널 간 의존성 (channel-wise dependencies)를 계산하는 과정.
- Fully connected layer와 비선형 함수를 조절하는 것으로 간단하게 계산.

$$s = F_{ez}(z, W) = \sigma\big(W_2\delta(W_1 z)\big)$$

- 주목해야 할 부분은 reduction ratio r을 통해서 W_1의 노드 수를 줄이고 W_2에서 다시 feature map의 수 C만큼 증가시킴.

$$\tilde{x}_c = F_{scale}(u_c, s_c) = s_c \cdot u_c$$

- 결국 excitation operation을 거친 스케일 값 s들이 모두 0과 1 사이의 값을 가지기 때문에 채널들의 중요도에 따라 스케일 됨.

Recent CNN (2) - SENet

- VGGNet, GoogLeNet, ResNet에 SE block을 적용

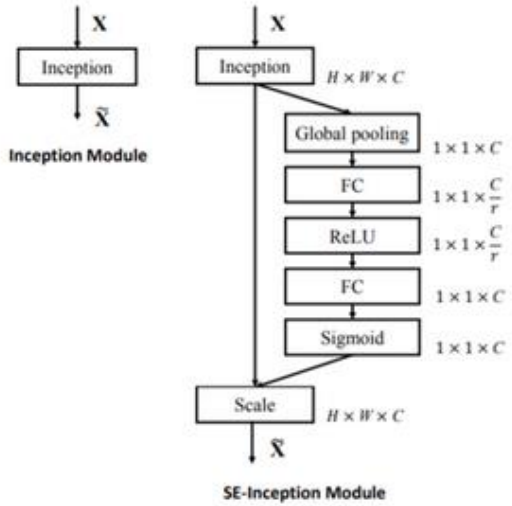


Figure 2: The schema of the original Inception module (left) and the SE-Inception module (right).

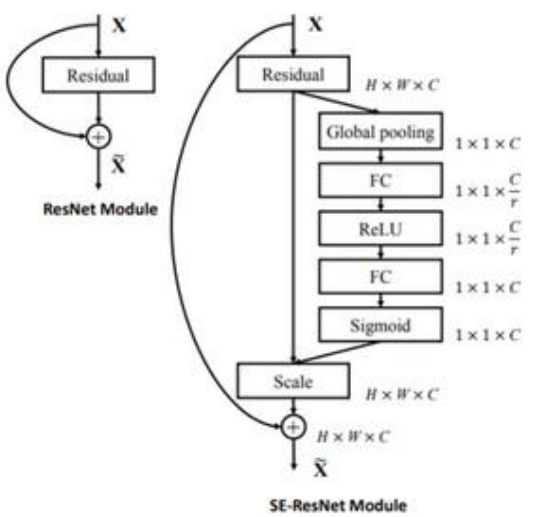


Figure 3: The schema of the original Residual module (left) and the SE-ResNet module (right).

Recent CNN (2) - SENet

- ResNet 결과

	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [10]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [10]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [10]	23.0	6.7	22.42	6.34	11.30	$21.57_{(0.85)}$	$5.73_{(0.61)}$	11.32
ResNeXt-50 [47]	22.2	-	22.11	5.90	4.24	$21.10_{(1.01)}$	$5.49_{(0.41)}$	4.25
ResNeXt-101 [47]	21.2	5.6	21.18	5.57	7.99	$20.70_{(0.48)}$	$5.01_{(0.56)}$	8.00
VGG-16 [39]	-	-	27.02	8.81	15.47	$25.22_{(1.80)}$	$7.70_{(1.11)}$	15.48
BN-Inception [16]	25.2	7.82	25.38	7.89	2.03	$24.23_{(1.15)}$	$7.14_{(0.75)}$	2.04
Inception-ResNet-v2 [42]	$19.9^{\dagger}$	$4.9^{\dagger}$	20.37	5.21	11.75	$19.80_{(0.57)}$	$4.79_{(0.42)}$	11.76

- MobileNet & ShuffleNet 결과

	original		re-implementation				SENet			
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	MFLOPs	Million Parameters	top-1 err.	top-5 err.	MFLOPs	Million Parameters
MobileNet [13]	29.4	-	29.1	10.1	569	4.2	$25.3_{(3.8)}$	$7.9_{(2.2)}$	572	4.7
ShuffleNet [52]	34.1	-	33.9	13.6	140	1.8	$31.7_{(2.2)}$	$11.7_{(1.9)}$	142	2.4

Table 3: Single-crop error rates (%) on the ImageNet validation set and complexity comparisons. Here, MobileNet refers to "1.0 MobileNet-224" in [13] and ShuffleNet refers to "ShuffleNet $1 \times (g = 3)$" in [52].