컴퓨터비전 프로젝트

최종발표

Deep Residual Learning for Image Recognition

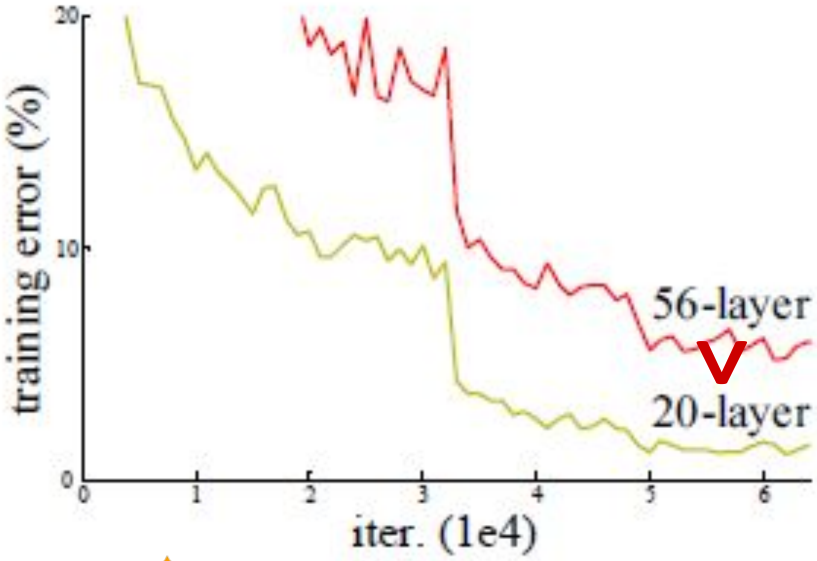
2022. 6. 9

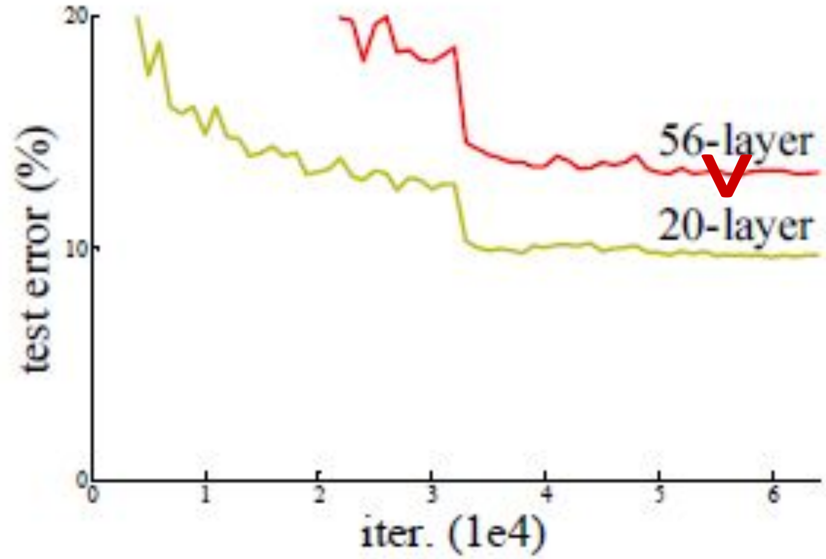
조 편성

기계공학부 20175806 김현우

소프트웨어학부 20192455 이주영

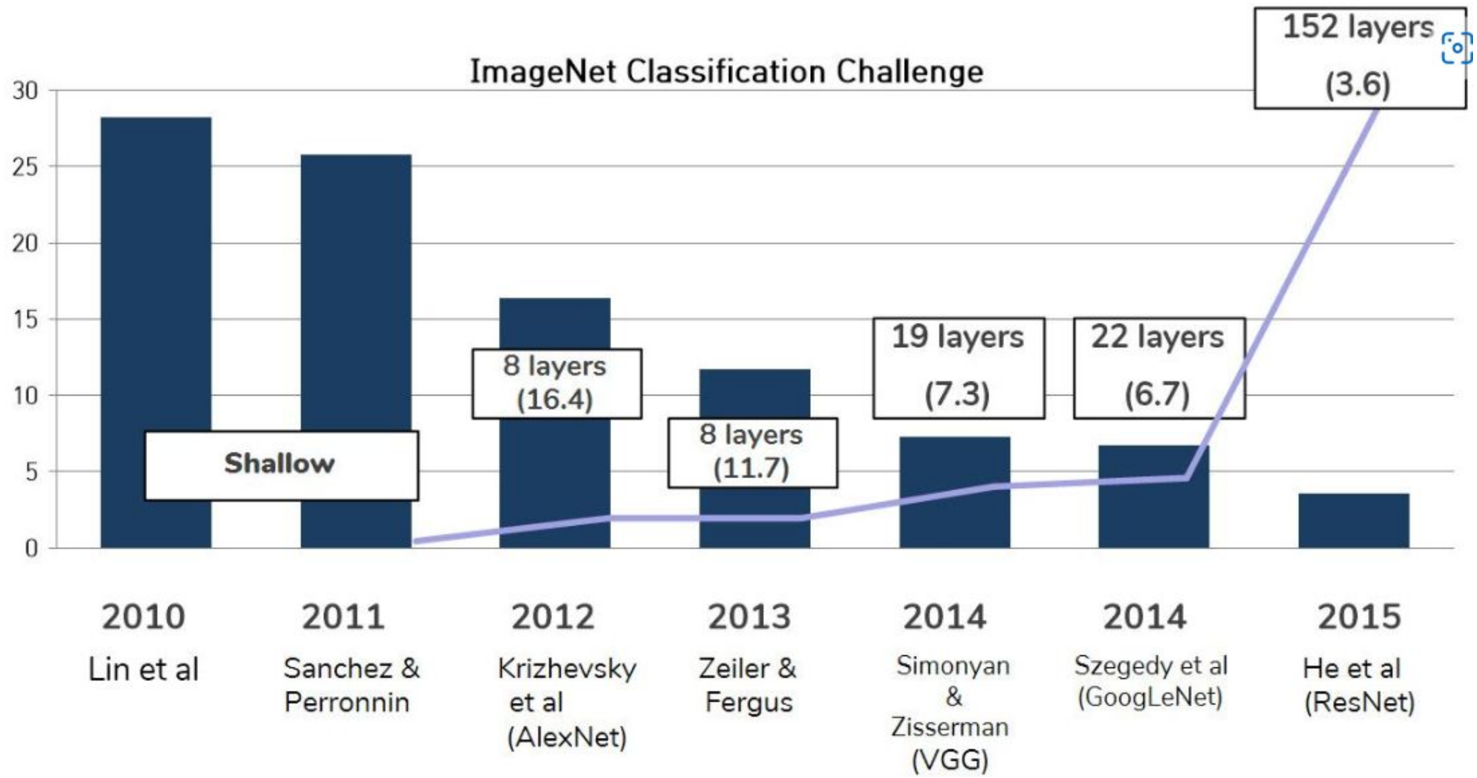
1. 문제 제기/필요성





- **1st** place on **ILSVRC 2015 classification** task
- easier to optimize, lower complexity

2. 내용

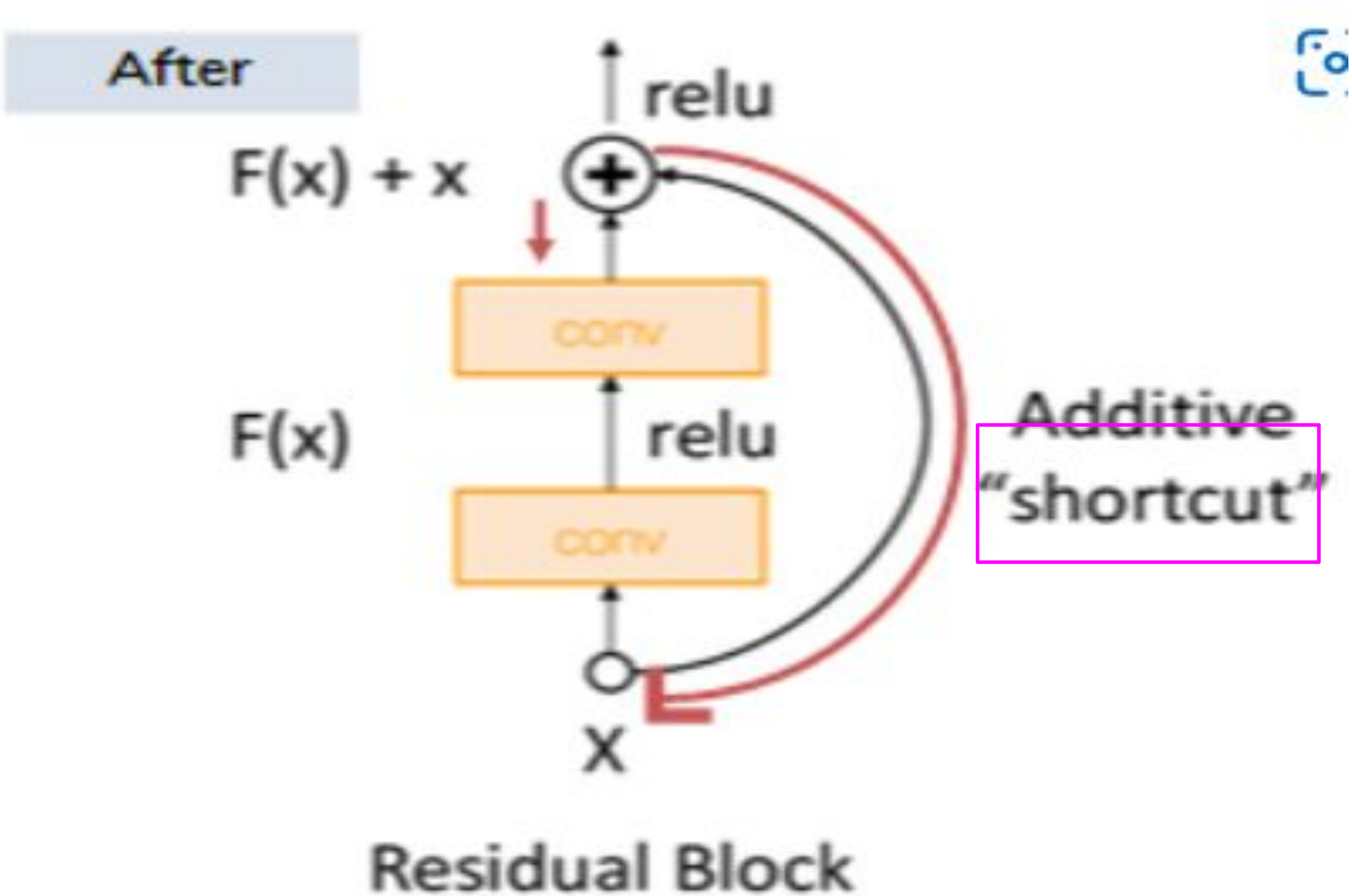


(1) **Shortcut Connection**

(2) **Bottleneck Block**

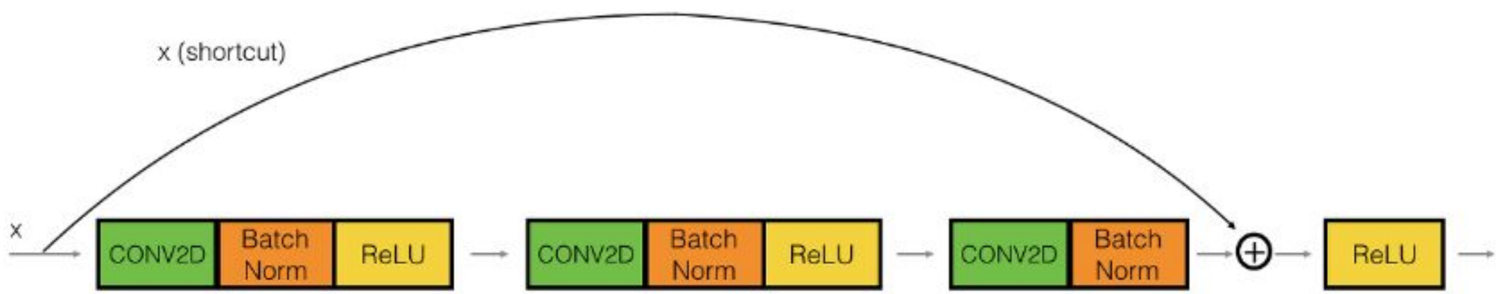
2. 내용 (1) **Shortcut Connection**

$$H(x) - x = F(x)$$

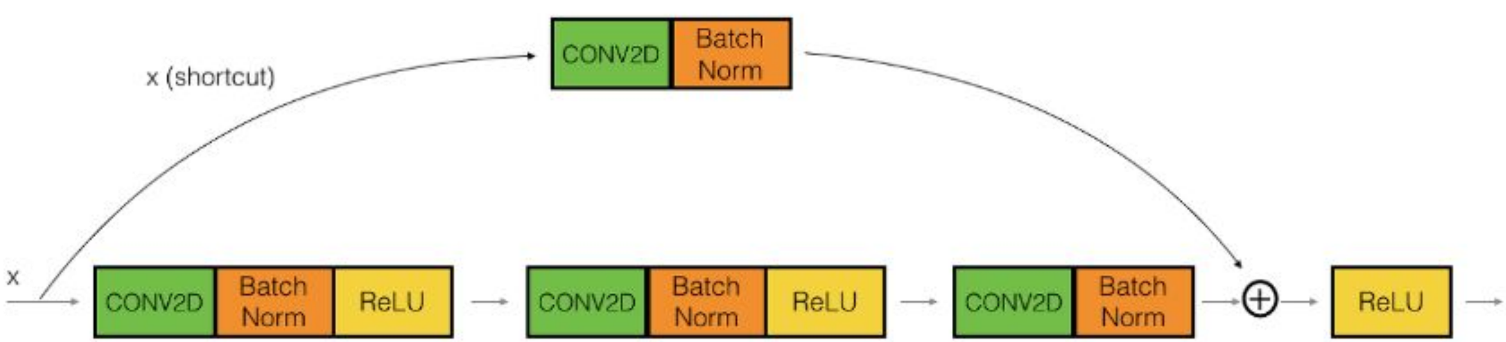


2. 내용 (1) **Shortcut Connection**

$$\frac{\partial H(x)}{\partial x} = \frac{\partial (F(x) + x)}{\partial x} = F'(x) + 1 \geq 1$$



identity block



convolution block

2. 내용 (1) **Shortcut Connection**

```
ResNetBasicBlock(
  (blocks): Sequential(
    (0): Sequential(
      (conv): Conv2dAuto(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): ReLU()
    (2): Sequential(
      (conv): Conv2dAuto(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (shortcut): Sequential(
    (conv): Conv2d(32, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

```
ResNetBottleNeckBlock(
  (blocks): Sequential(
    (0): Sequential(
      (conv): Conv2dAuto(32, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): ReLU()
    (2): Sequential(
      (conv): Conv2dAuto(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (3): ReLU()
    (4): Sequential(
      (conv): Conv2dAuto(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (shortcut): Sequential(
    (conv): Conv2d(32, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

2. 내용 (2) **Bottleneck Block**

3. 구현 내용 및 결과 분석

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```python
def resnet18(in_channels, n_classes):
    return ResNet(in_channels, n_classes, block=ResNetBasicBlock, deepths=[2, 2, 2, 2])

def resnet34(in_channels, n_classes):
    return ResNet(in_channels, n_classes, block=ResNetBasicBlock, deepths=[3, 4, 6, 3])

def resnet50(in_channels, n_classes):
    return ResNet(in_channels, n_classes, block=ResNetBottleNeckBlock, deepths=[3, 4, 6, 3])

def resnet101(in_channels, n_classes):
    return ResNet(in_channels, n_classes, block=ResNetBottleNeckBlock, deepths=[3, 4, 23, 3])

def resnet152(in_channels, n_classes):
    return ResNet(in_channels, n_classes, block=ResNetBottleNeckBlock, deepths=[3, 8, 36, 3])
```

3. 구현 내용 및 결과 분석

```python
def resnet18(in_channels, n_classes):
    return ResNet(in_channels, n_classes, block=ResNetBasicBlock, deepths=[2, 2, 2, 2])
```

```python
def resnet50(in_channels, n_classes):
    return ResNet(in_channels, n_classes, block=ResNetBottleNeckBlock, deepths=[3, 4, 6, 3])
```

```python
class ResNet(nn.Module):

    def __init__(self, in_channels, n_classes, *args, **kwargs):
        super().__init__()
        self.encoder = ResNetEncoder(in_channels, *args, **kwargs)
        self.decoder = ResnetDecoder(self.encoder.blocks[-1].blocks[-1].expanded_channels, n_classes)

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

3. 구현 내용 및 결과 분석

```
class ResNetBasicBlock(ResNetResidualBlock):
    expansion = 1
    def __init__(self, in_channels, out_channels, activation=nn.ReLU, *args, **kwargs):
        super().__init__(in_channels, out_channels, *args, **kwargs)
        self.blocks = nn.Sequential(
            conv_bn(self.in_channels, self.out_channels, conv=self.conv, bias=False, stride=self.downsampling)
            activation(),
            conv_bn(self.out_channels, self.expanded_channels, conv=self.conv, bias=False),
        )
```

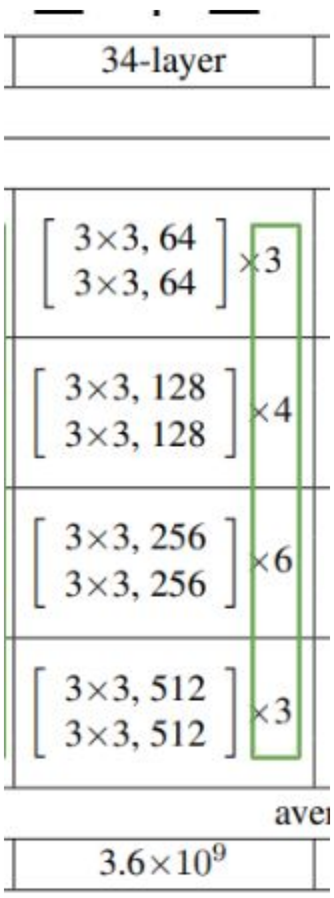
```
class ResNetBottleNeckBlock(ResNetResidualBlock):
    expansion = 4
    def __init__(self, in_channels, out_channels, activation=nn.ReLU, *args, **kwargs):
        super().__init__(in_channels, out_channels, expansion=4, *args, **kwargs)
        self.blocks = nn.Sequential(
           conv_bn(self.in_channels, self.out_channels, self.conv, kernel_size=1),
             activation(),
             conv_bn(self.out_channels, self.out_channels, self.conv, kernel_size=3, stride=self.downsampling
             activation(),
             conv_bn(self.out_channels, self.expanded_channels, self.conv, kernel_size=1),
        )
```

```
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
```

```
train_dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform_train)
```

3. 구현 내용 및 결과 분석

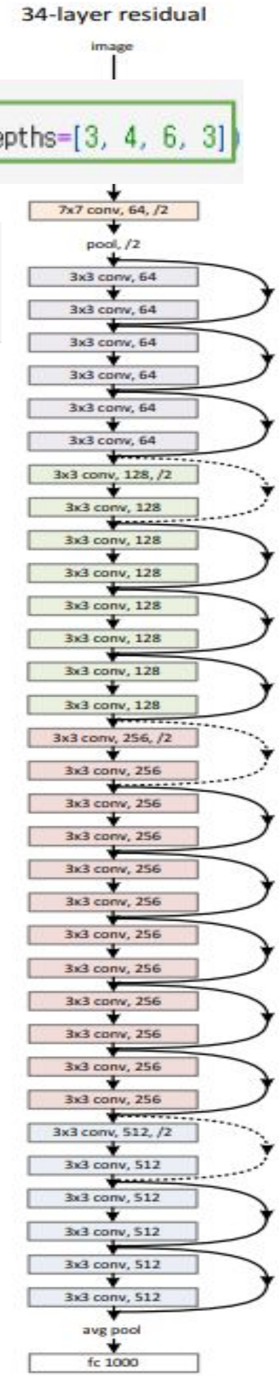
```python
def resnet34(in_channels, n_classes):
    return ResNet(in_channels, n_classes, block=ResNetBasicBlock, deepths=[3, 4, 6, 3])
```

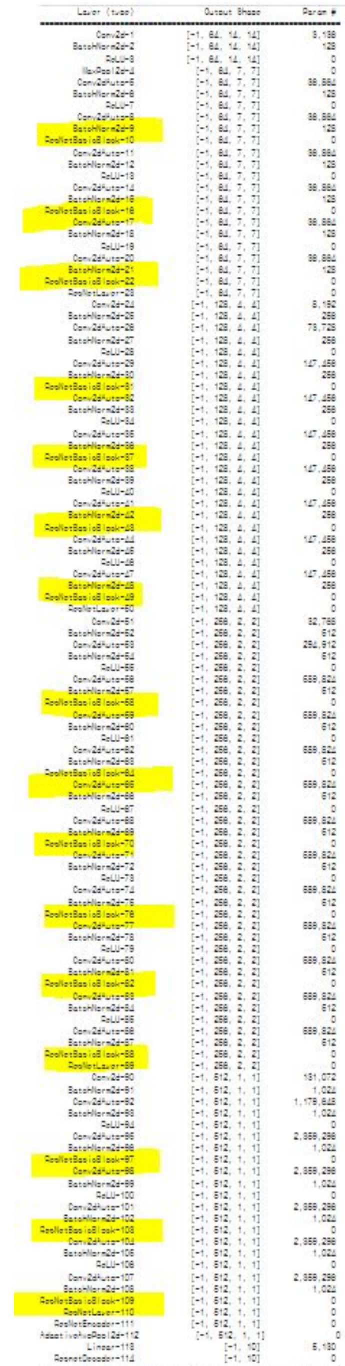



```python
from torchsummary import summary

model = resnet34(1, 10)
summary(model.cuda(), (1, 28, 28))
```

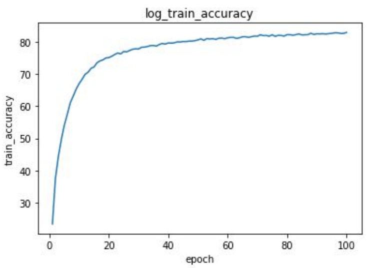
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 14, 14]           3,136
       BatchNorm2d-2           [-1, 64, 14, 14]             128
              ReLU-3           [-1, 64, 14, 14]               0
         MaxPool2d-4             [-1, 64, 7, 7]               0
        Conv2dAuto-5             [-1, 64, 7, 7]          36,864
       BatchNorm2d-6             [-1, 64, 7, 7]             128
              ReLU-7             [-1, 64, 7, 7]               0
        Conv2dAuto-8             [-1, 64, 7, 7]          36,864
       BatchNorm2d-9             [-1, 64, 7, 7]             128
ResNetBasicBlock-10             [-1, 64, 7, 7]               0
       Conv2dAuto-11             [-1, 64, 7, 7]          36,864
      BatchNorm2d-12             [-1, 64, 7, 7]             128
             ReLU-13             [-1, 64, 7, 7]               0
       Conv2dAuto-14             [-1, 64, 7, 7]          36,864
      BatchNorm2d-15             [-1, 64, 7, 7]             128
ResNetBasicBlock-16             [-1, 64, 7, 7]               0
```

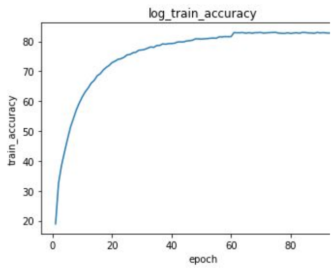


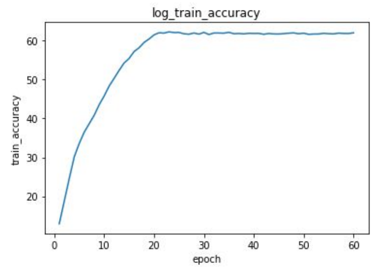


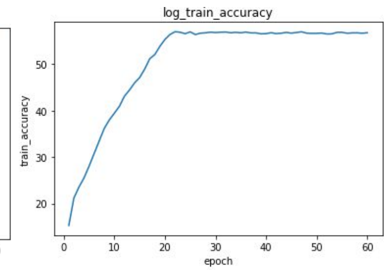
3. 구현 내용 및 결과 분석 - CIFAR-10 dataset

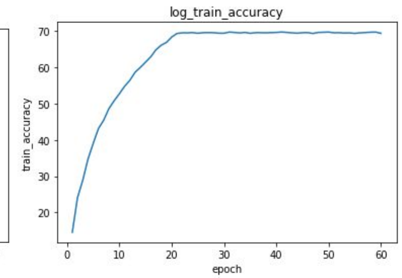
- Google Colab
- **Train Accuracy**



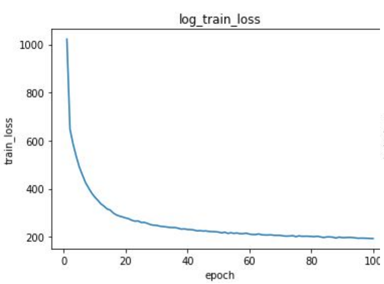


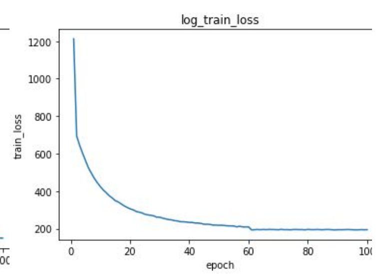


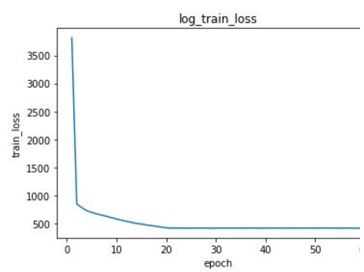


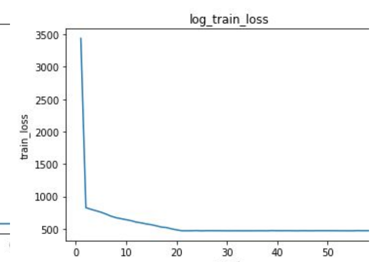


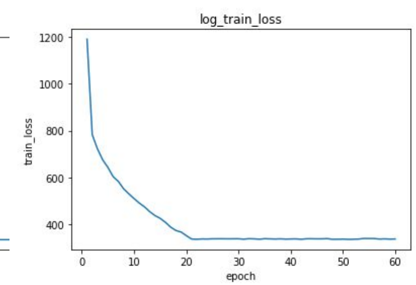
- **Train Loss**





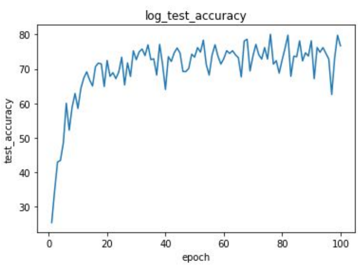


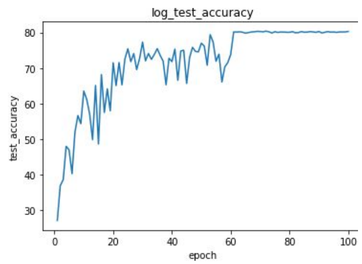


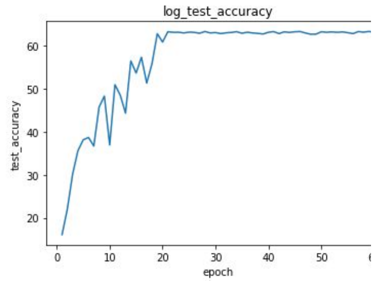


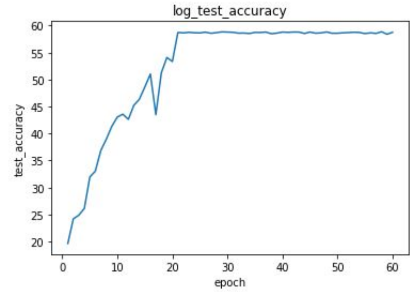
3. 구현 내용 및 결과 분석 - CIFAR-10 dataset

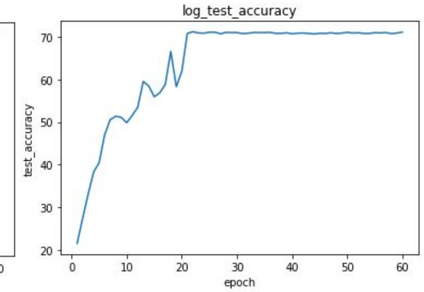
- **Test Accuracy**



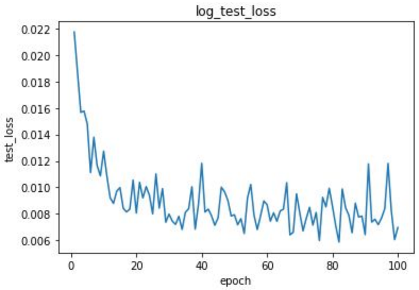


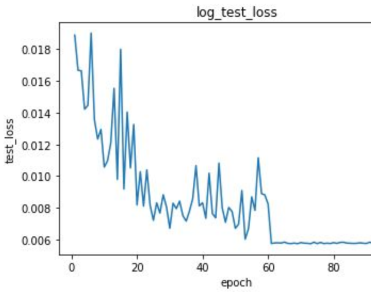


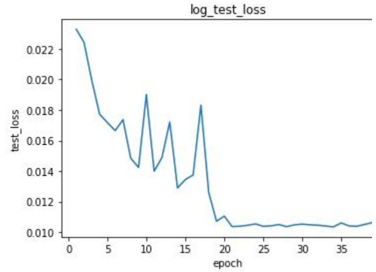


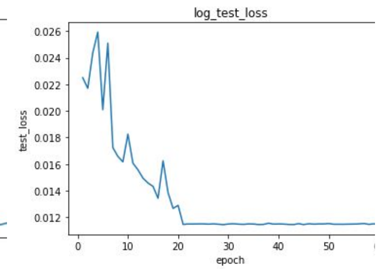


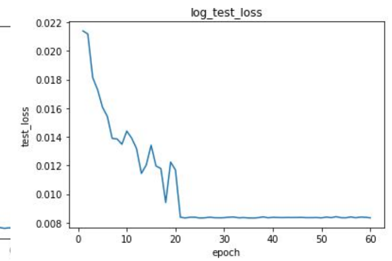
- **Test Loss**





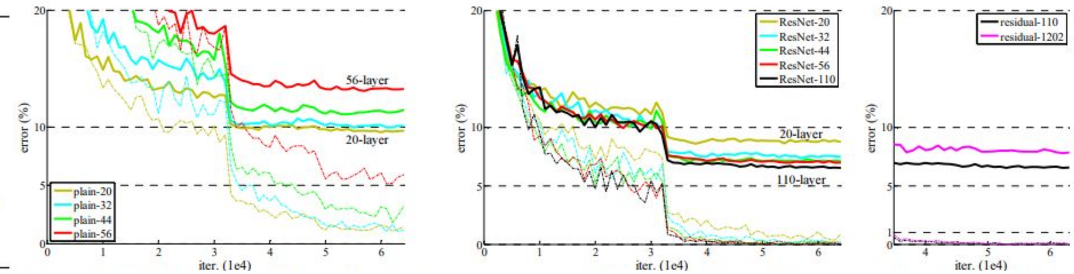






4. 개선 내용 - CIFAR-10 dataset

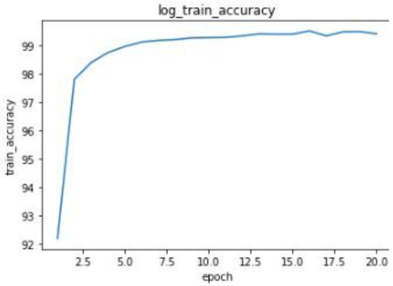
method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

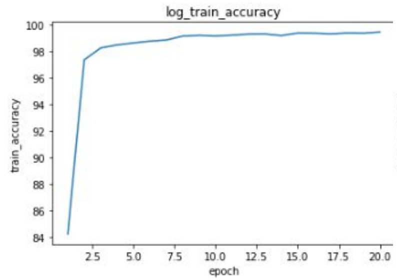


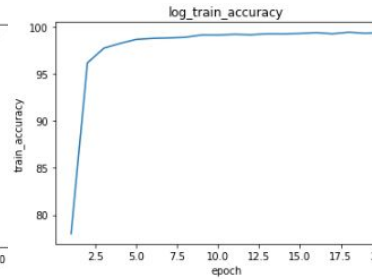
	18	34	50	101	152
train accuracy	83.022	82.722	61.974	56.776	69.376
train loss	193.057136774 0631	194.398092538 11836	416.790073037 1475]	472.024994432 9262	337.761715769 76776
test accuracy	76.69	80.28	63.21	58.76	71.05
test loss	0.00696204485 297203	0.00580733307 0039749	0.01044059162 1398926	0.01148222939 3720627	0.00833706557 1546554

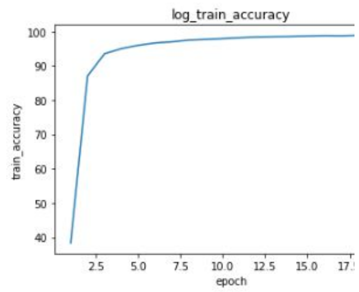
4. 개선 내용 - MNIST dataset

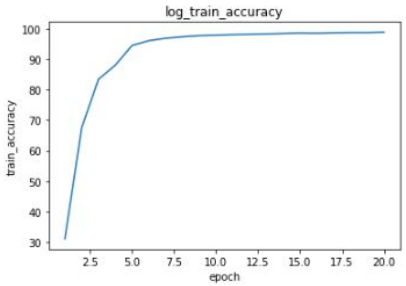
- **Train Accuracy**



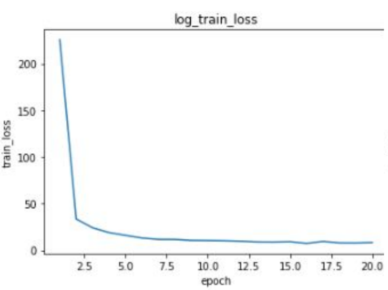


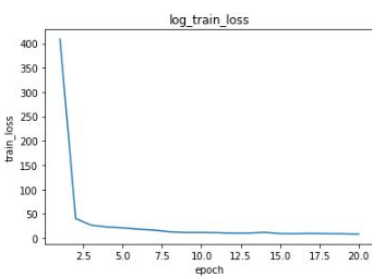


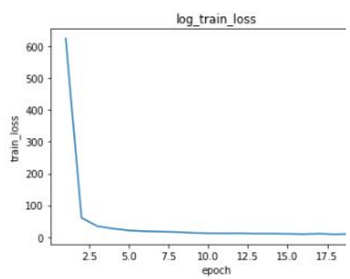


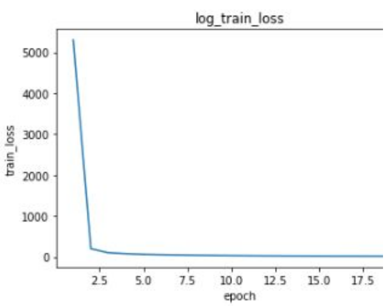


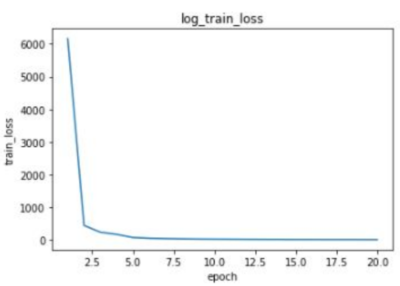
- **Train Loss**





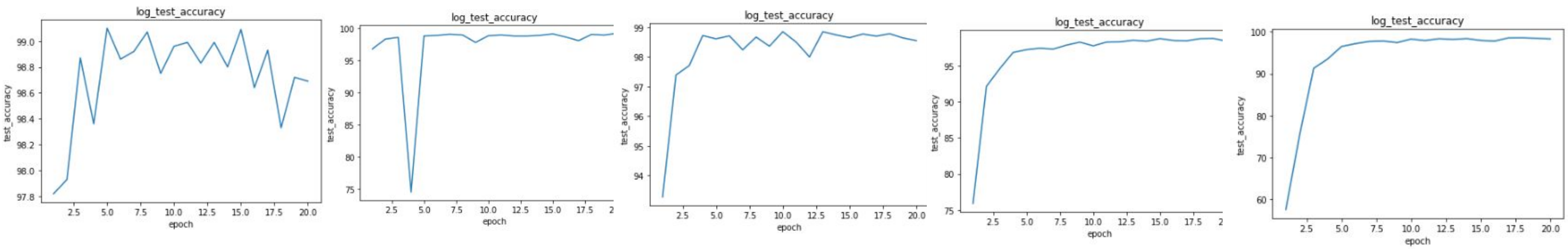




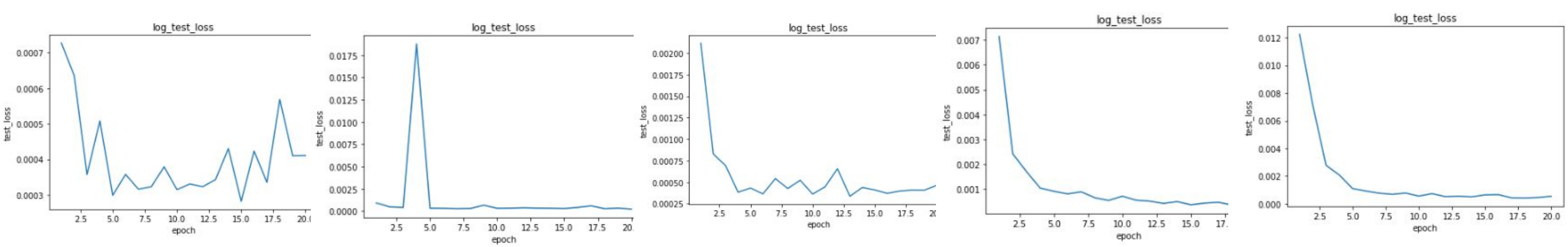


4. 개선 내용 - MNIST dataset

- **Train Accuracy**



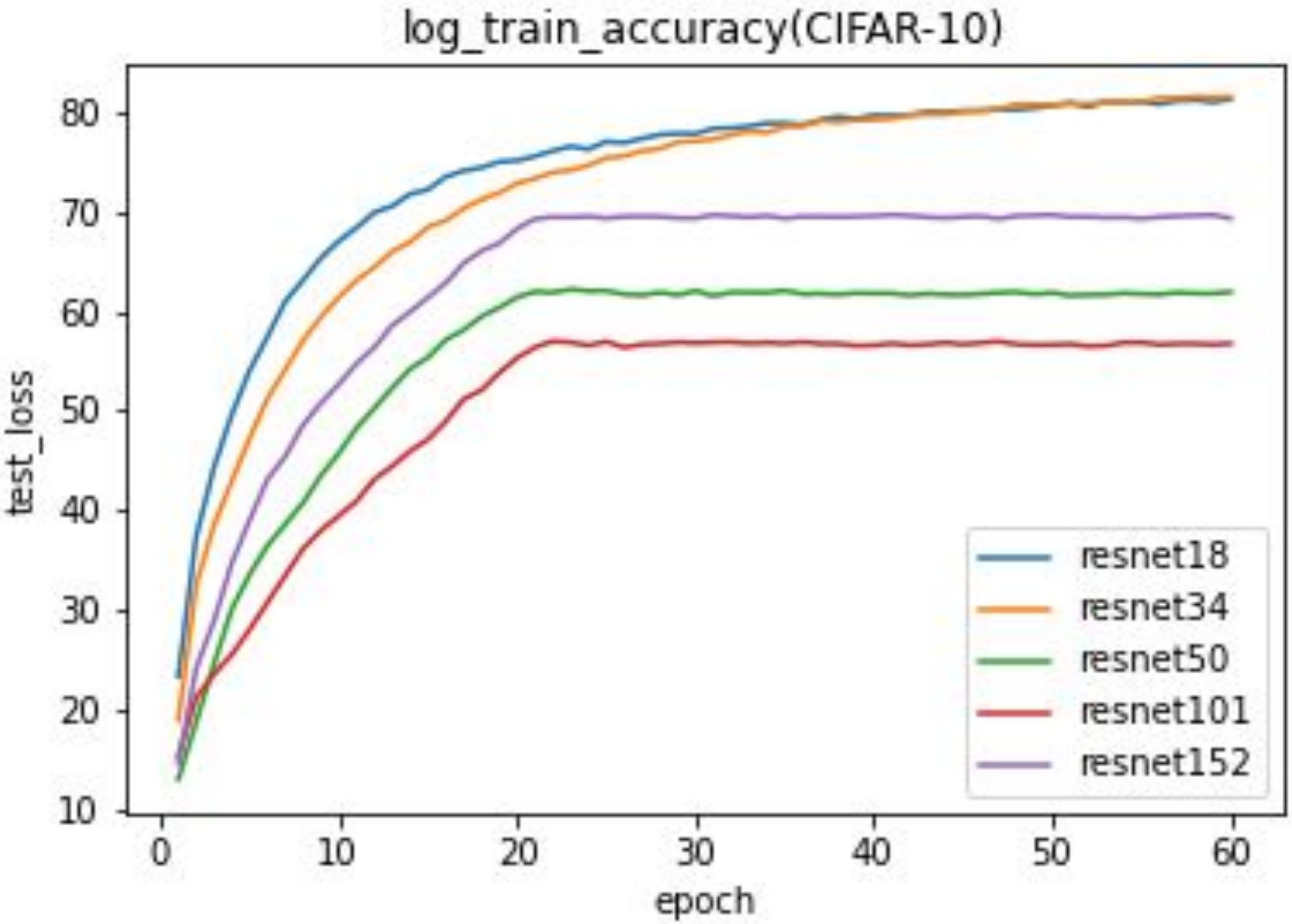
- **Test Loss**



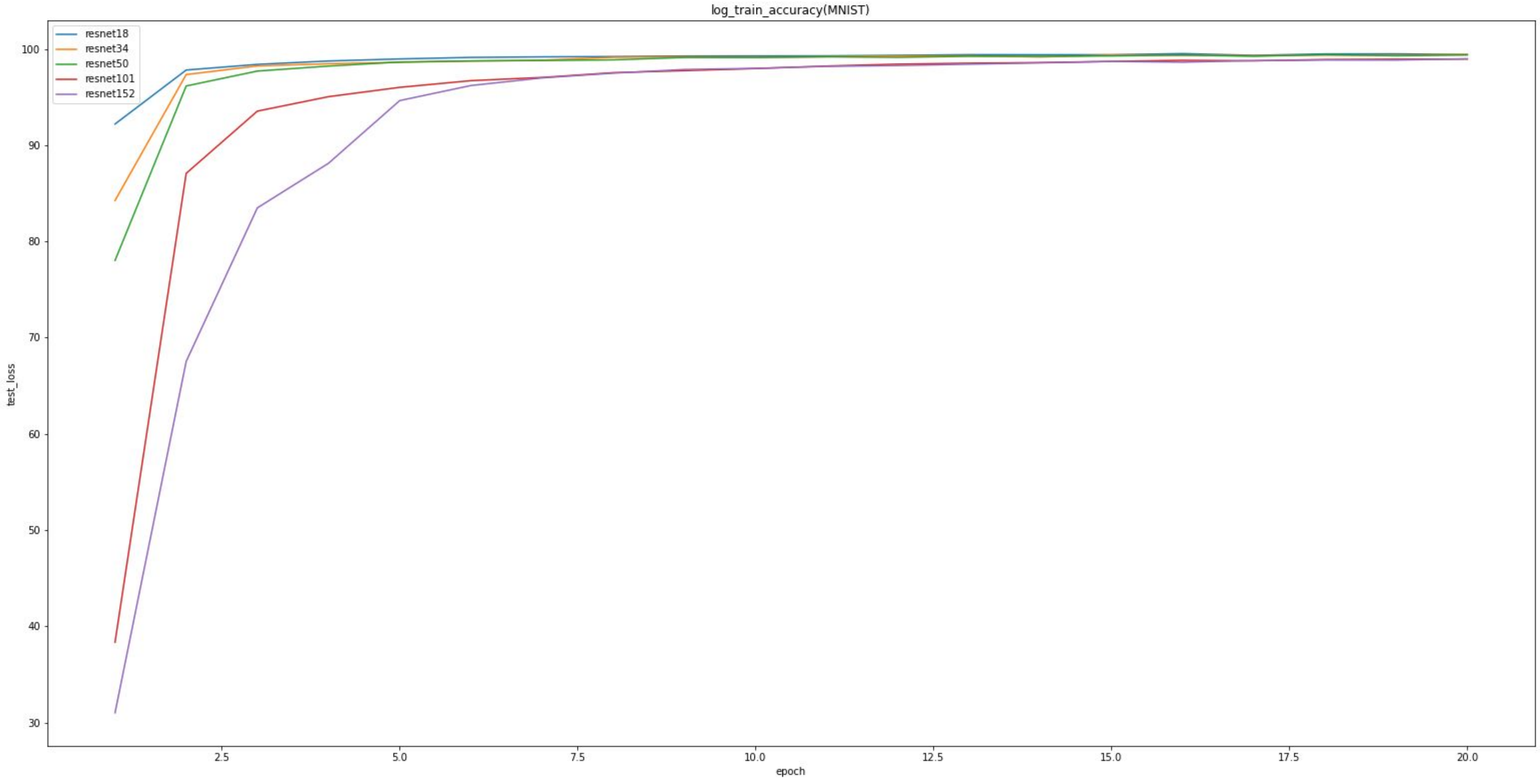
4. 개선 내용 - MNIST dataset

	18	34	50	101	152
train accuracy	99.40333333333334	99.425	99.36333333333333	98.945	98.965
train loss	8.203844760952052	8.523877266357886	9.164297751209233	16.18617395788897]	15.927942932932638
test accuracy	98.69	99.24	98.55	98.37	98.29
test loss	0.0004107659961387981	0.0002369845971959876	0.0004639747631139471	0.0005147263069520705]	0.0005391983608278679

4. 개선 내용


log_train_accuracy(CIFAR-10)
test_loss
epoch
10
20
30
40
50
60
70
80
0
10
20
30
40
50
60
resnet18
resnet34
resnet50
resnet101
resnet152

4. 개선 내용



log_train_accuracy(MNIST)
resnet18
resnet34
resnet50
resnet101
resnet152
test_loss
epoch
100
90
80
70
60
50
40
30
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

5. 결론

- layer 및 block을 정의하고 조합하는 방법 학습
- torchvision.datasets 사용 방법 학습
- Matplotlib 사용 Accuracy & Loss 그래프 작성 방법 학습
- 무작위적 layer 추가가 아닌 ResNet 모델의 깊이별 결과 비교
- 논문 구현 경험
- 논문 결과 검증 경험
- Pytorch 이해 경험

6. 참고문헌

- Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Microsoft Research
  Deep Residual Learning for Image Recognition
  CVPR 2016

- Deep-Learning-Paper-Review-and-Practice/ResNet18_CIFAR10_Train.ipynb at master · ndb796/Deep-Learning-Paper-Review-and-Practice (github.com)

- Deep-Learning-Paper-Review-and-Practice/ResNet18_MNIST_Train.ipynb at master · ndb796/Deep-Learning-Paper-Review-and-Practice (github.com)

- FrancescoSaverioZuppichini/ResNet: Clean, scalable and easy to use ResNet implementation in Pytorch (github.com)

- ResNet에 관한 이해 : 네이버 블로그 (naver.com)

- ResNet에 대하여 (velog.io)

https://github.com/jjjuurang/2022-1-ComputerVisionProject/tree/main/%EC%B5%9C%EC%A2%85%20%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8