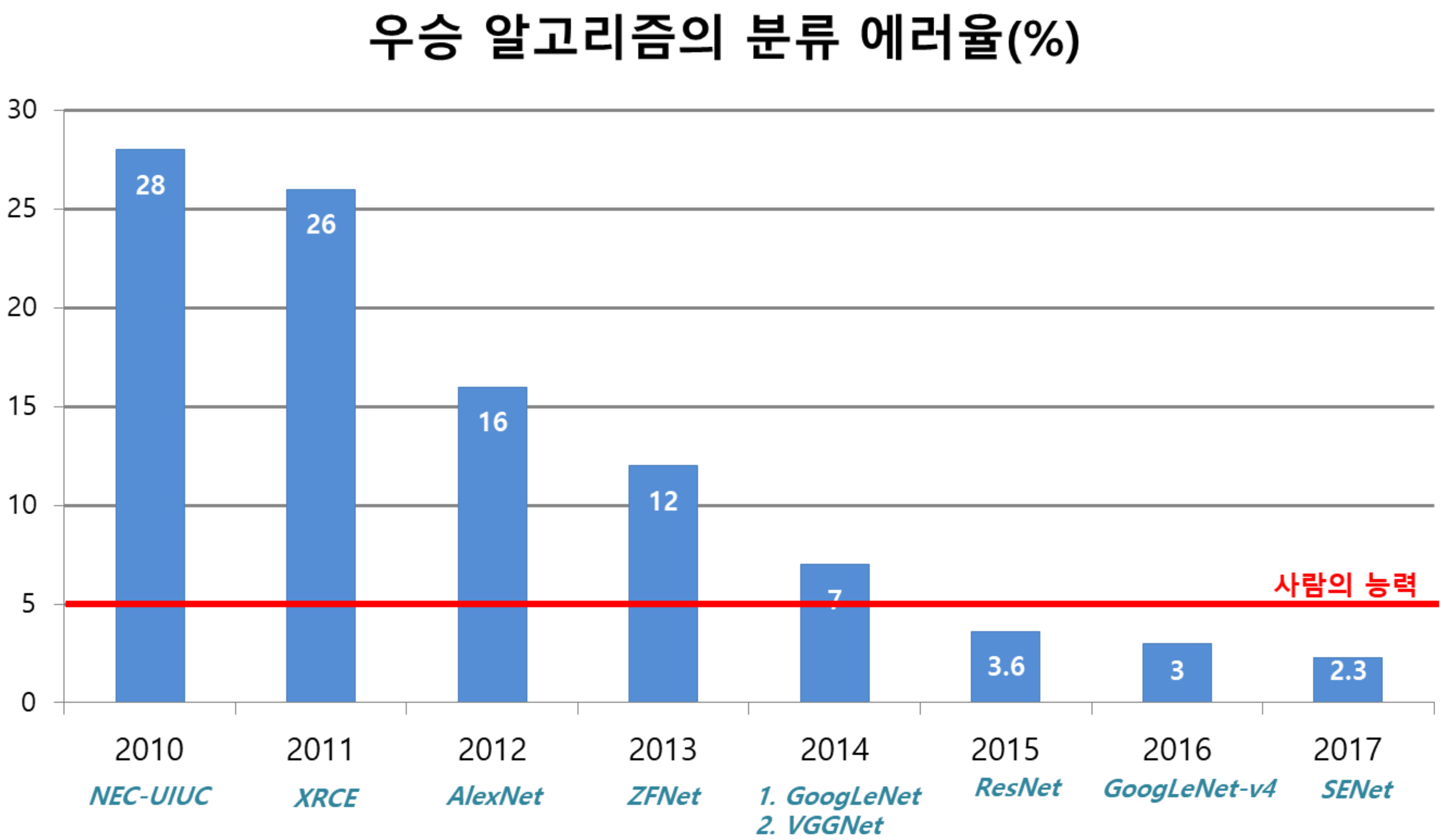
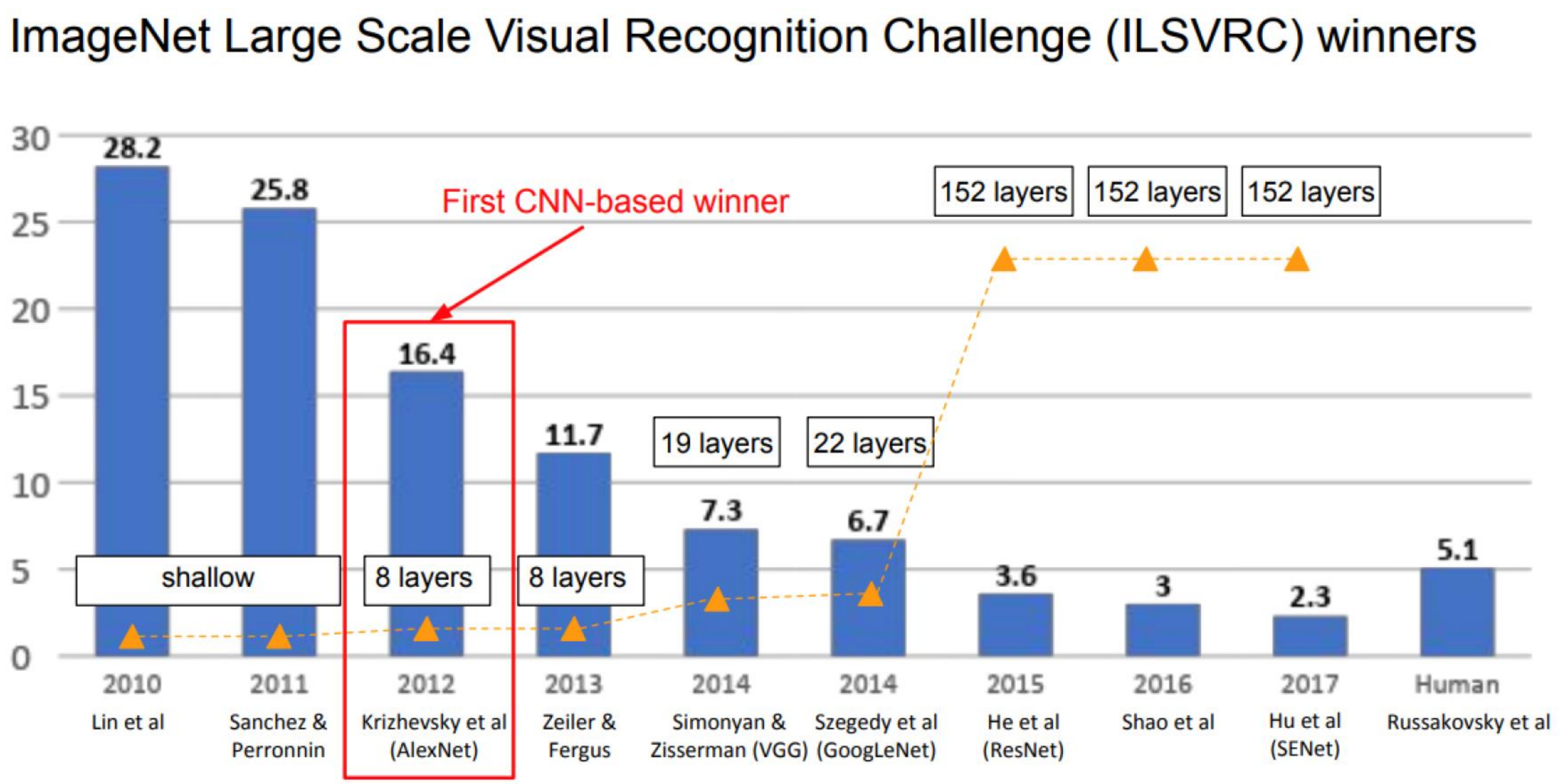
Advanced CNN

ESC 1조
김민희 김수연 백채빈 손지우 이성우

01 ILSVRC

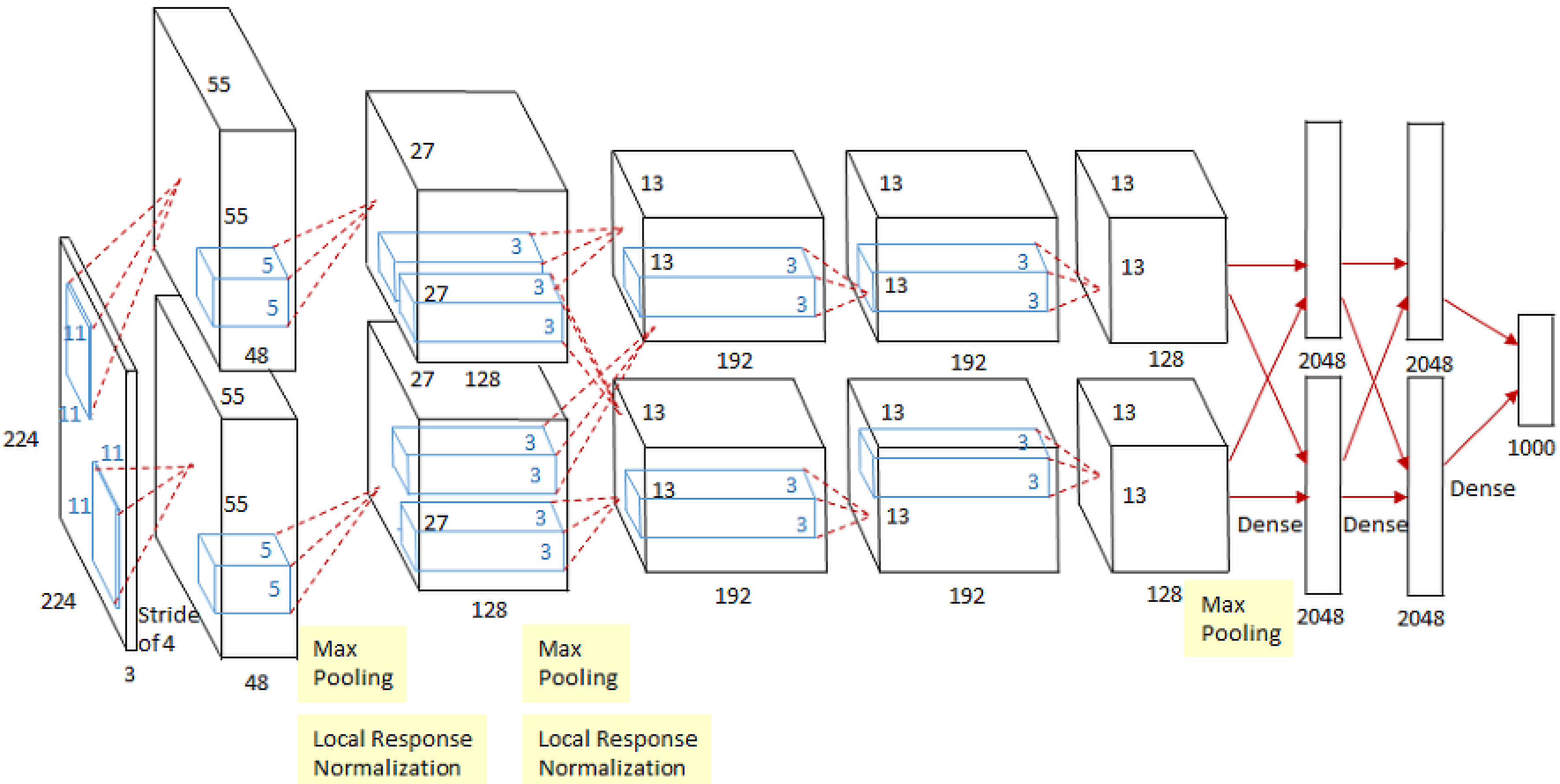

우승 알고리즘의 분류 에러율(%)
30
25
20
15
10
5
0
28
26
16
12
7
3.6
3
2.3
사람의 능력
2010
NEC-UIUC
2011
XRCE
2012
AlexNet
2013
ZFNet
2014
1. GoogLeNet
2. VGGNet
2015
ResNet
2016
GoogLeNet-v4
2017
SENet

AlexNet

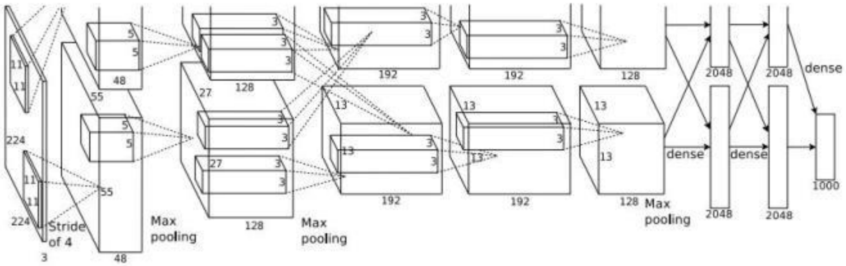

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
3.6
3
2.3
5.1
First CNN-based winner
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
152 layers
152 layers
2010
Lin et al
2011
Sanchez & Perronnin
2012
Krizhevsky et al (AlexNet)
2013
Zeiler & Fergus
2014
Simonyan & Zisserman (VGG)
2014
Szegedy et al (GoogLeNet)
2015
He et al (ResNet)
2016
Shao et al
2017
Hu et al (SENet)
Human
Russakovsky et al

02 AlexNet

◆ Structure



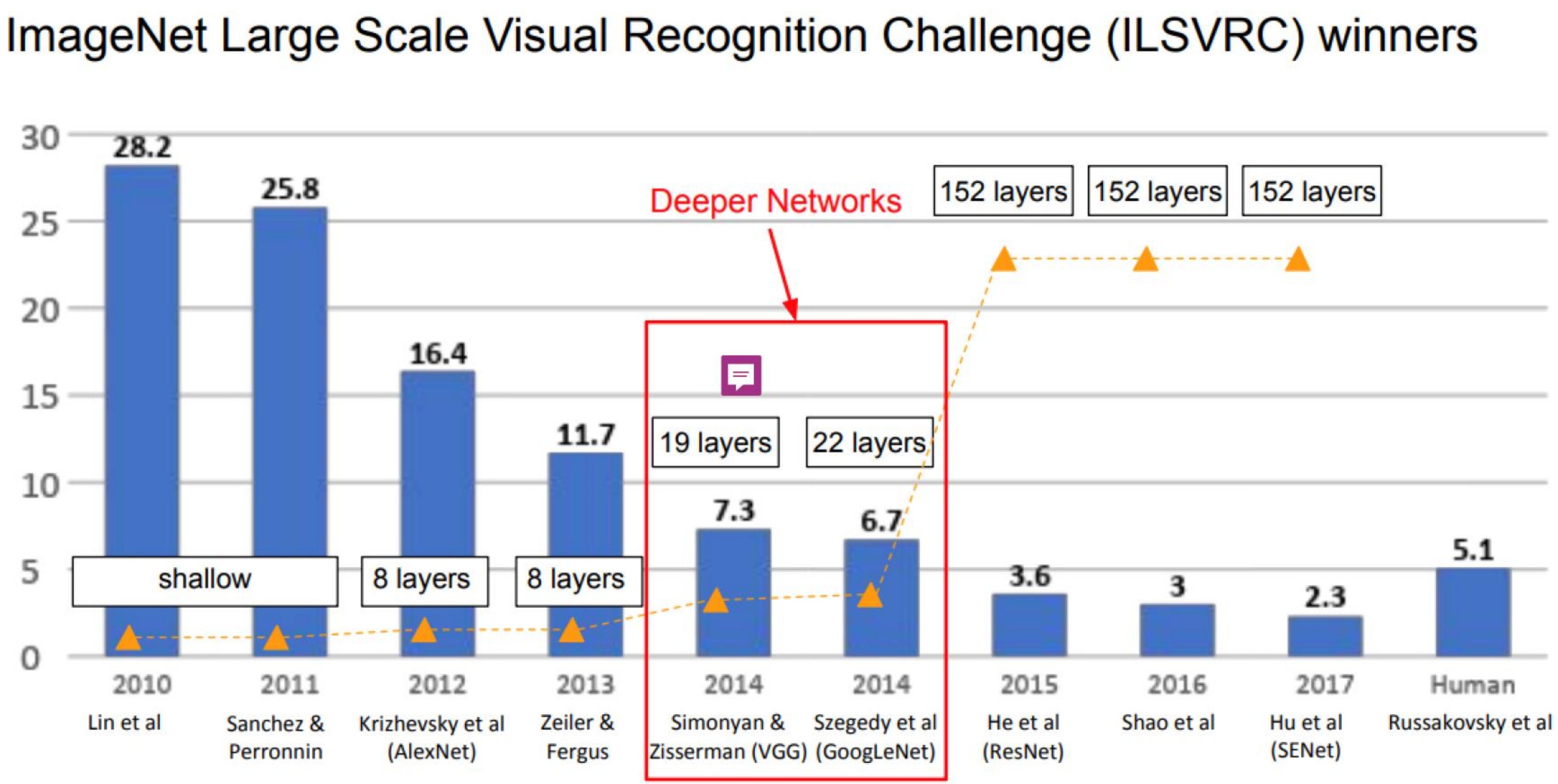
AlexNet



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

03

VGGNet



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
3.6
3
2.3
5.1
Deeper Networks
152 layers
152 layers
152 layers
19 layers
22 layers
shallow
8 layers
8 layers
2010
Lin et al
2011
Sanchez & Perronnin
2012
Krizhevsky et al (AlexNet)
2013
Zeiler & Fergus
2014
Simonyan & Zisserman (VGG)
2014
Szegedy et al (GoogLeNet)
2015
He et al (ResNet)
2016
Shao et al
2017
Hu et al (SENet)
Human
Russakovsky et al

03

VGGNet

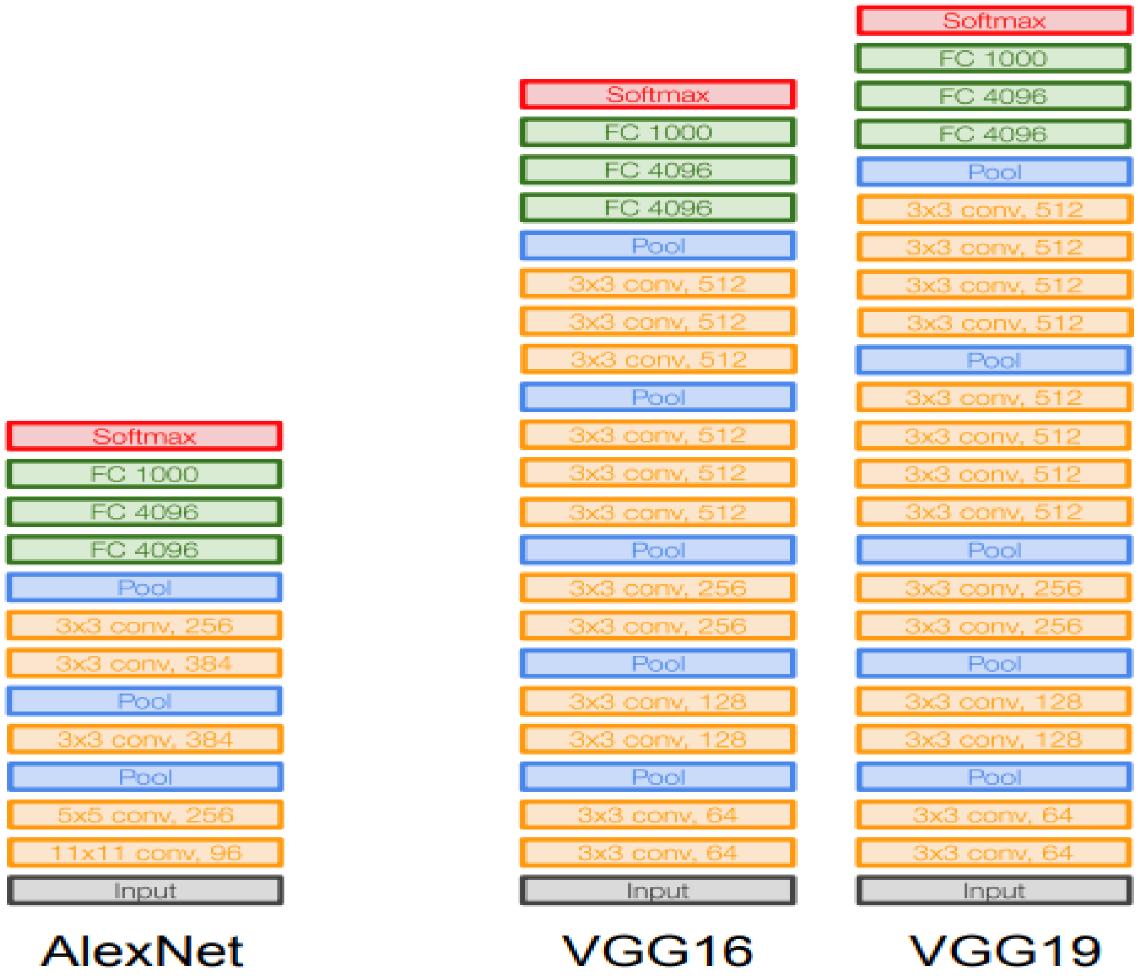
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Karen Simonyan & Andrew Zisserman, 「VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION」

03

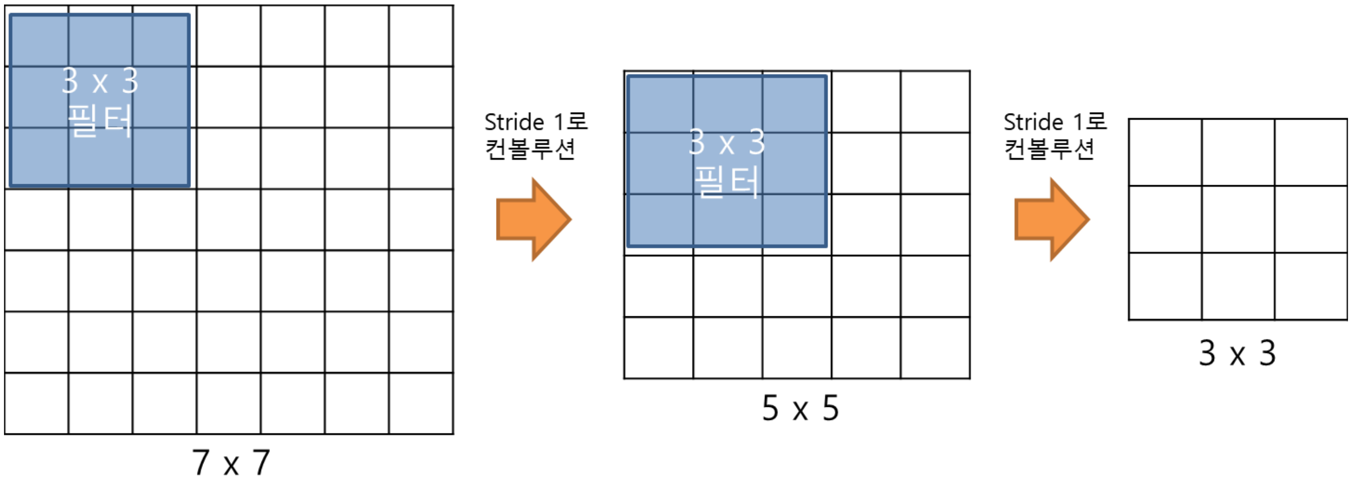
VGGNet

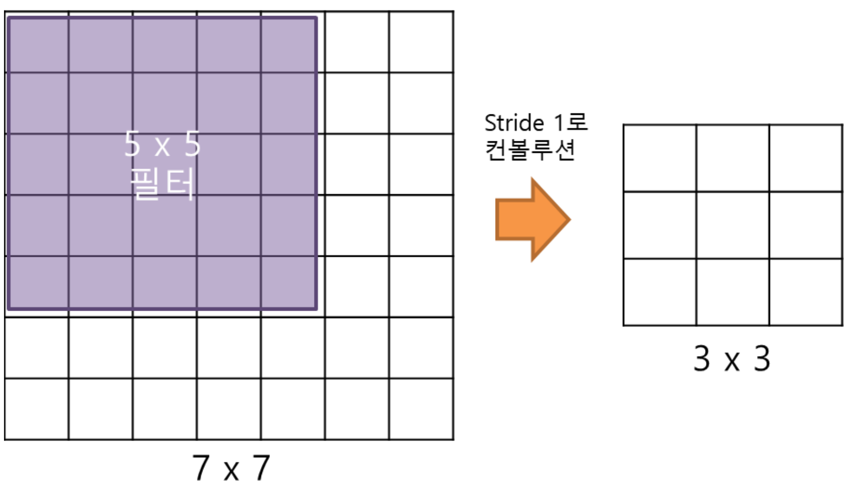
- Compare with AlexNet



03 VGGNet

◆ Filter

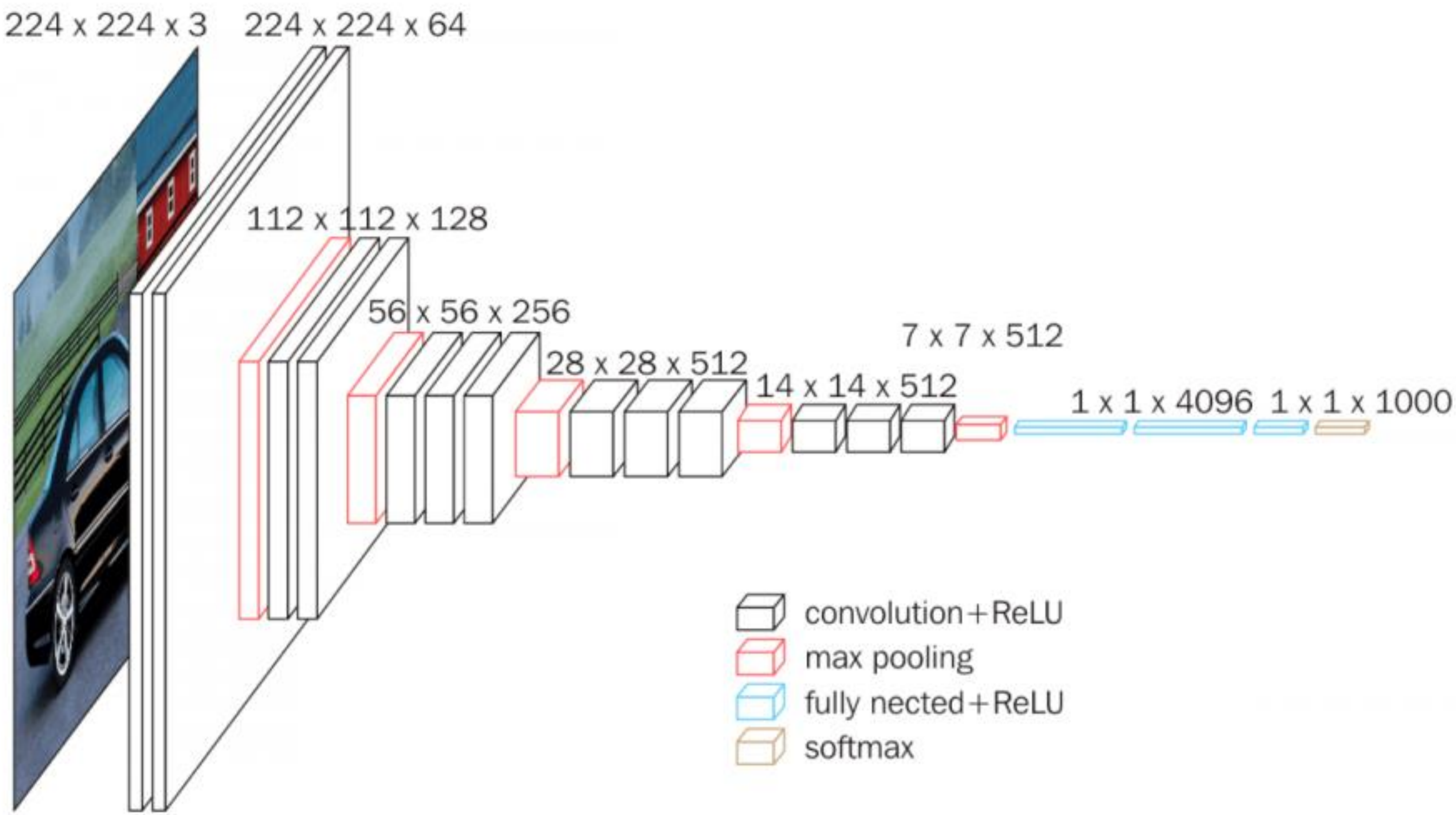




03

VGGNet

◆ Structure



03

VGGNet

```
class VGG(nn.Module):
    def __init__(self, features, num_classes=1000, init_weights=True):
        super(VGG, self).__init__()

        self.features = features #convolution layers (we are building)

        self.avgpool = nn.AdaptiveAvgPool2d((7, 7))

        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )#FC layer
```

Adaptive Average Pooling이란?

- 기존 average pooling: kernel size와 stride를 hyperparameter로서 제시해주어야 함.
- **Adaptive** average pooling은 output size를 지정해주면 알아서 kernel size와 stride가 결정된다.
  - Stride = (input_size//output_size)
  - Kernel size = input_size - (output_size-1)*stride
  - Padding = 0

VGGNet

◆ Code

```
def forward(self, x):
    x = self.features(x) #Convolution
    x = self.avgpool(x) #avgpool
    x = x.view(x.size(0), -1) #일렬로 펼치기
    x = self.classifier(x) #FC layer
    return x

def _initialize_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
            # kaiming normalization은 아래 사진 또는 지난주 세션자료 참고
            if m.bias is not None:
                nn.init.constant_(m.bias, 0)  #initialize 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)
```

n_{in} = layer의 input 개수
n_{out} = layer의 output 갯수

2015년

■ He Normal initialization

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

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

03 VGGNet

◆ Code

```
def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 3  #input channel

    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v  #중요* (conv layer 통과하고 나오게 되면 channel #이 변경)

    return nn.Sequential(*layers)
```

03

VGGNet

◆ Code

```
'A' = [64,'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M']

### VGG-11

#Convolutional Layer1
conv2d = nn.Conv2d(3, 64, kernel_size=3, padding=1)
nn.ReLU(inplace=True) #in_channels = 64
nn.MaxPool2d(kernel_size=2, stride=2)

#Convolutional Layer2
nn.Conv2d(64, 128, kernel_size=3, padding=1)
nn.ReLU(inplace=True) #in_channels = 128
nn.MaxPool2d(kernel_size=2, stride=2)

#Convolutional Layer3,4
nn.Conv2d(128, 256, kernel_size=3, padding=1)
nn.ReLU(inplace=True) #in_channels = 256
nn.Conv2d(256, 256, kernel_size=3, padding=1)
nn.ReLU(inplace=True) #in_channels = 256
nn.MaxPool2d(kernel_size=2, stride=2)
```

```
#Convolutional Layer5,6
nn.Conv2d(256, 512, kernel_size=3, padding=1)
nn.ReLU(inplace=True) #in_channels = 512
nn.Conv2d(512, 512, kernel_size=3, padding=1)
nn.ReLU(inplace=True) #in_channels = 512
nn.MaxPool2d(kernel_size=2, stride=2)

#Convolutional Layer7,8
nn.Conv2d(512, 512, kernel_size=3, padding=1)
nn.ReLU(inplace=True) #in_channels = 512
nn.Conv2d(512, 512, kernel_size=3, padding=1)
nn.ReLU(inplace=True) #in_channels = 512
nn.MaxPool2d(kernel_size=2, stride=2)

#FC Layer 3
```

03

VGGNet

◆ Code

```
'custom' : [64,64,64,'M',128,128,128,'M',256,256,256,'M']
```

```
conv = make_layers(cfg['custom'], batch_norm=True)
conv
```

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): ReLU(inplace=True)
  (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (12): ReLU(inplace=True)
  (13): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (14): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (15): ReLU(inplace=True)
  (16): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (17): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (18): ReLU(inplace=True)
  (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (20): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (22): ReLU(inplace=True)
  (23): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (24): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (25): ReLU(inplace=True)
  (26): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (28): ReLU(inplace=True)
  (29): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

03

VGGNet

◆ Code

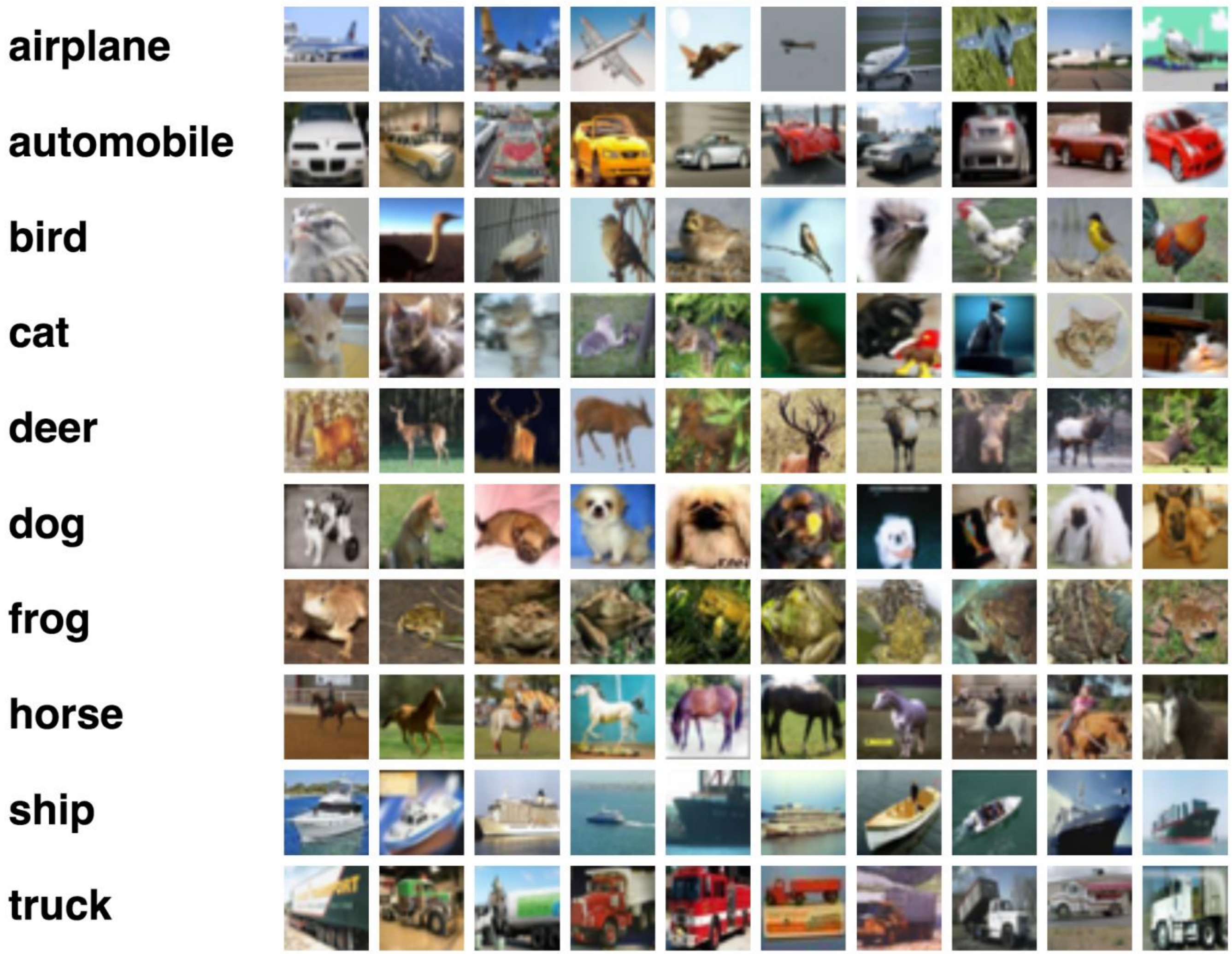
```
'custom' : [64,64,64,'M',128,128,128,'M',256,256,256,'M']
```

```
conv = make_layers(cfg['custom'], batch_norm=True)
conv
```

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): ReLU(inplace=True)
  (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (12): ReLU(inplace=True)
  (13): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (14): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (15): ReLU(inplace=True)
  (16): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (17): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (18): ReLU(inplace=True)
  (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (20): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (22): ReLU(inplace=True)
  (23): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (24): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (25): ReLU(inplace=True)
  (26): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (28): ReLU(inplace=True)
  (29): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

04

Cifar10 with VGGNet



04 Cifar10 with VGGNet

◆ Code (기본적으로 앞과 똑같다)

```
transform = transforms.Compose(
    [transforms.ToTensor(),  ### 데이터 폼 텐서로 변환
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])  ### 정규화

trainset = torchvision.datasets.CIFAR10(root='./cifar10', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=512, ### batch_size 512로 수정
                                          shuffle=True, num_workers=0)  ### num_workers: 멀티 프로세스 값 (default=0: 메인프로세스만)

testset = torchvision.datasets.CIFAR10(root='./cifar10', train=False,
                                       download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=0)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')  ### label 값들
```

Cifar10 with VGGNet

◆ Code

```
class VGG(nn.Module):

    def __init__(self, features, num_classes=1000, init_weights=True):
        super(VGG, self).__init__()
        self.features = features
        #self.avgpool = nn.AdaptiveAvgPool2d((7, 7))  ### vgg.py 에서는 (7,7)을 하도록 되어 있는데 이미지가 7by7보다 작으므로 굳이 할
        self.classifier = nn.Sequential(
            nn.Linear(512 * 4 * 4, 4096),  # 첫번째 classifier / 여기 역시 7 by 7이 아니라 4 by 4로 수정
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),  # 두번째 classifier
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),  # 세번째 classifier
        )
        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = self.features(x)
        #x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                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)
```

https://nittaku.tistory.com/269

04 Cifar10 with VGGNet

◆ Code

```
print(len(trainloader))
epochs = 30

for epoch in range(epochs):  # loop over the dataset multiple times
    running_loss = 0.0
    lr_sche.step()
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = vgg16(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 30 == 29:    # print every 30 mini-batches
            loss_tracker(loss_plt, torch.Tensor([running_loss/30]), torch.Tensor([i + epoch*len(trainloader) ]))
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 30))
            running_loss = 0.0


print('Finished Training')
```

04 Cifar10 with VGGNet

◆ Code

```
[25,    30] loss: 0.465
[25,    60] loss: 0.488
[25,    90] loss: 0.495
[26,    30] loss: 0.443
[26,    60] loss: 0.441
[26,    90] loss: 0.477
[27,    30] loss: 0.419
[27,    60] loss: 0.436
[27,    90] loss: 0.418
[28,    30] loss: 0.387
[28,    60] loss: 0.392
[28,    90] loss: 0.396
[29,    30] loss: 0.344
[29,    60] loss: 0.358
[29,    90] loss: 0.356
[30,    30] loss: 0.312
[30,    60] loss: 0.318
[30,    90] loss: 0.324
Finished Training
```

◆ Code

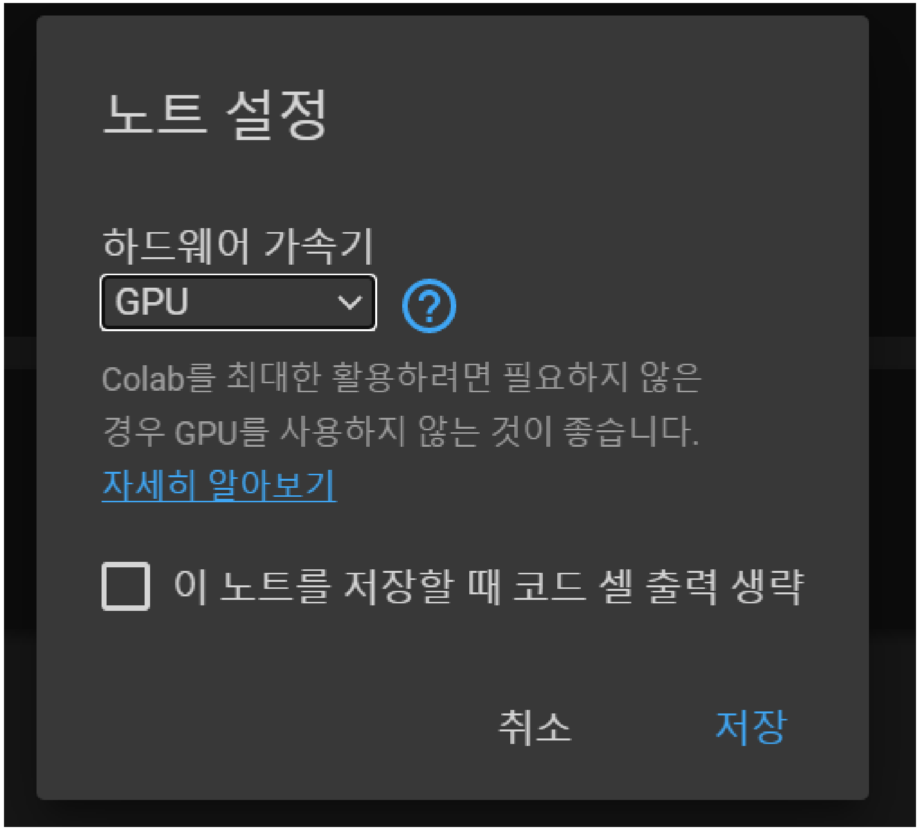
```
correct = 0
total = 0

with torch.no_grad():
    for data in testloader:
        images, labels = data
        images = images.to(device)
        labels = labels.to(device)
        outputs = vgg16(images)

        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)

        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 73 %
```

Cifar10 with VGGNet

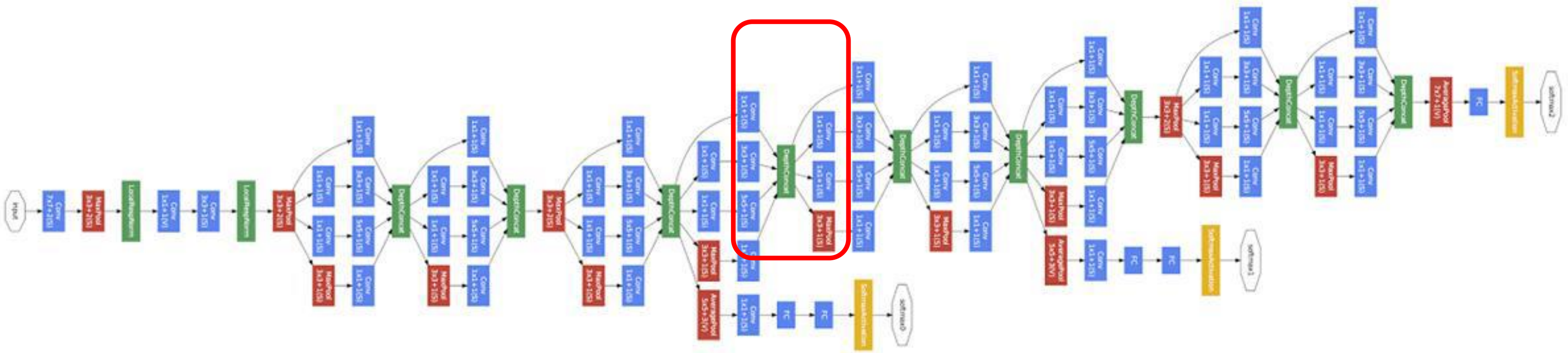
◆ Tip

코랩에서 GPU 설정으로 돌리면 훨씬훨씬 빠르게 돌아갑니다!

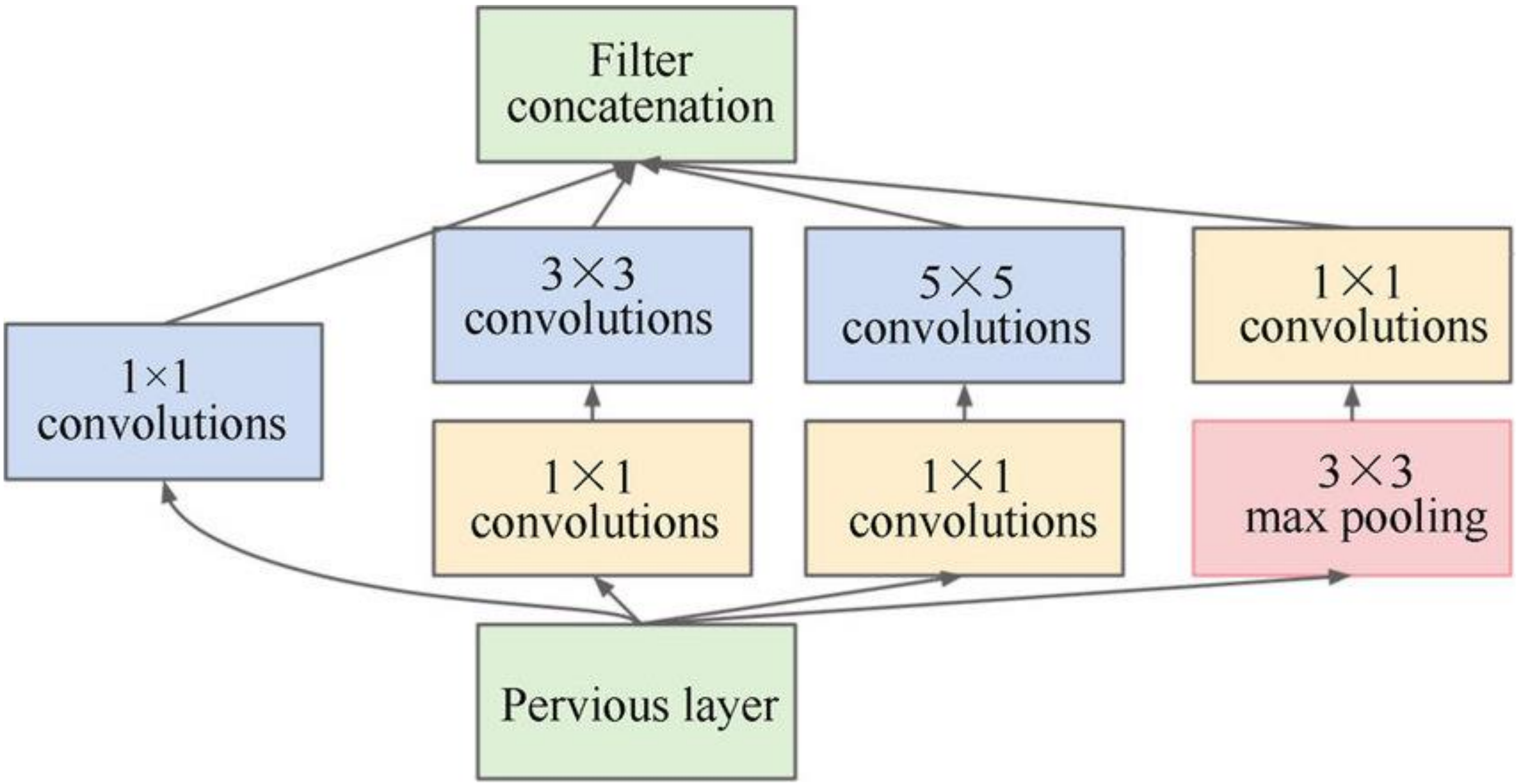


GooLeNet/Inception

GooLeNet/Inception

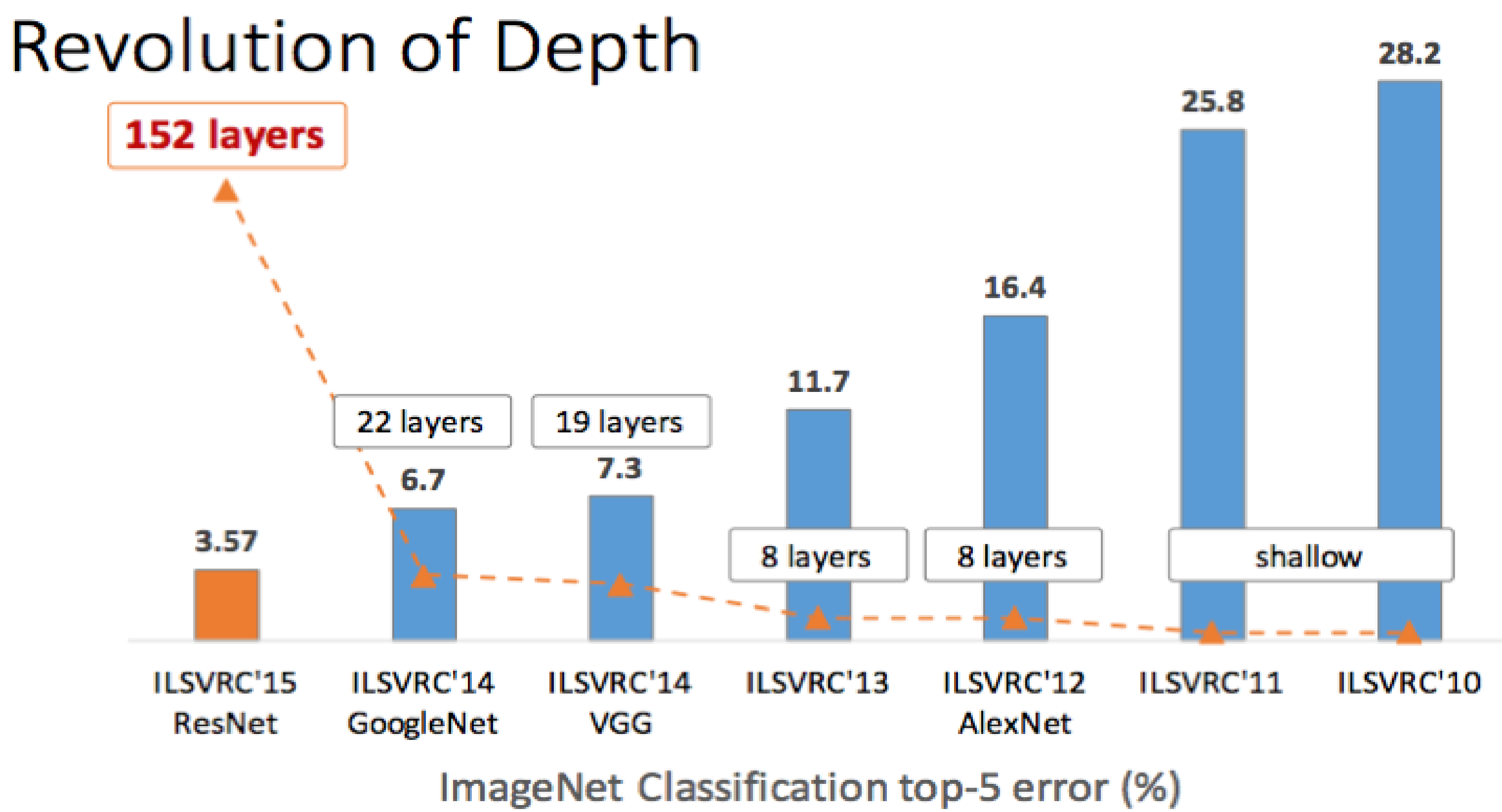


04 GooLeNet/Inception



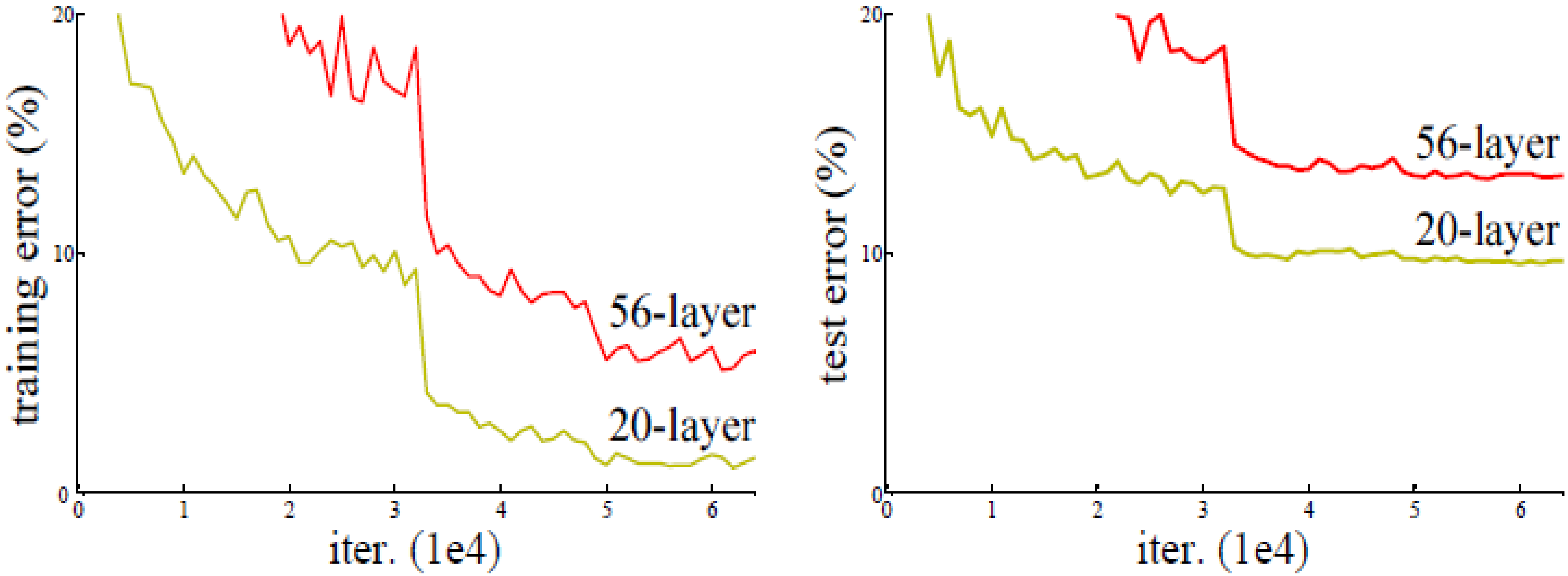
ResNet

- ◆ Background

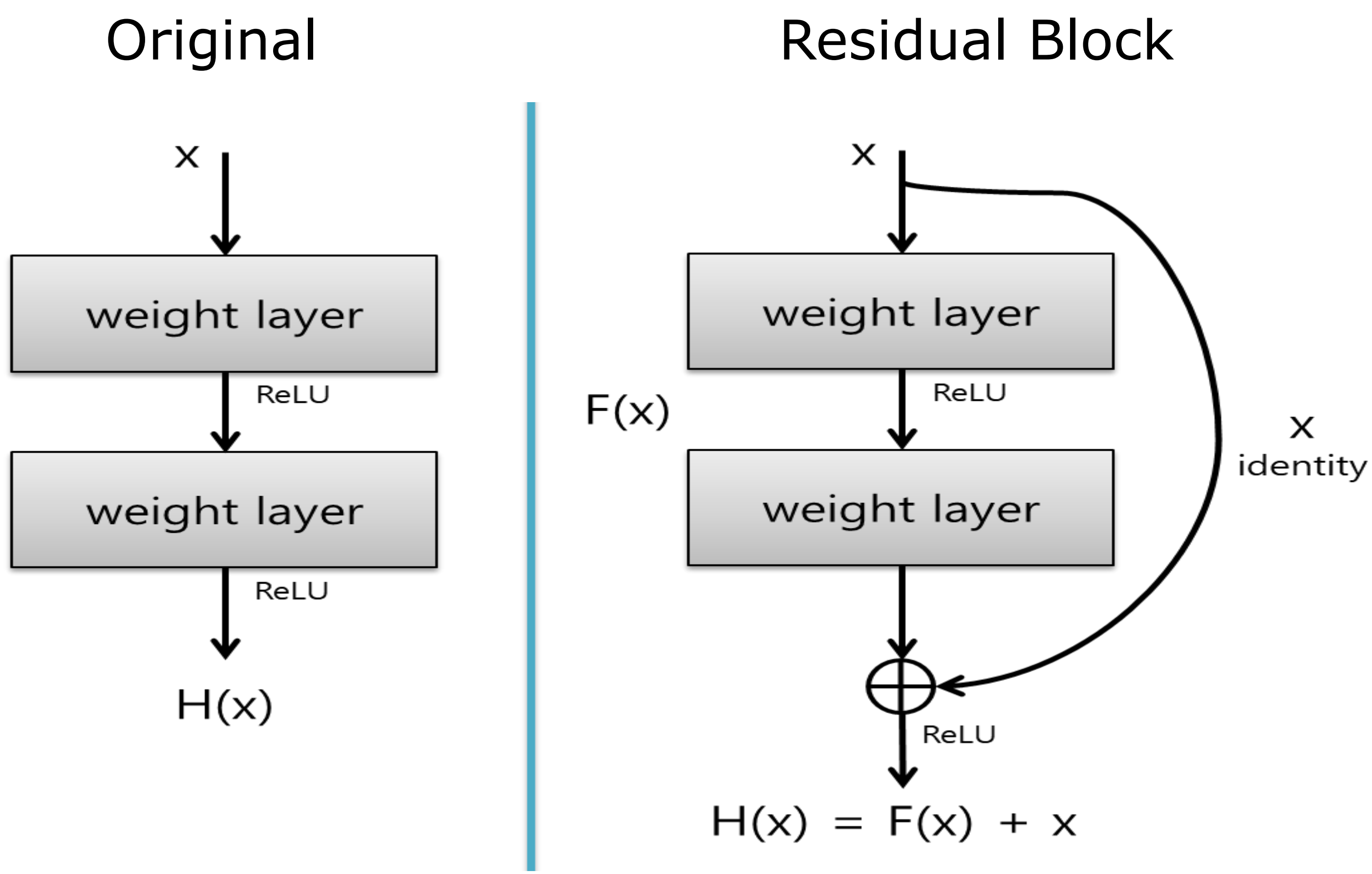


05 ResNet

- The depth of network and its error

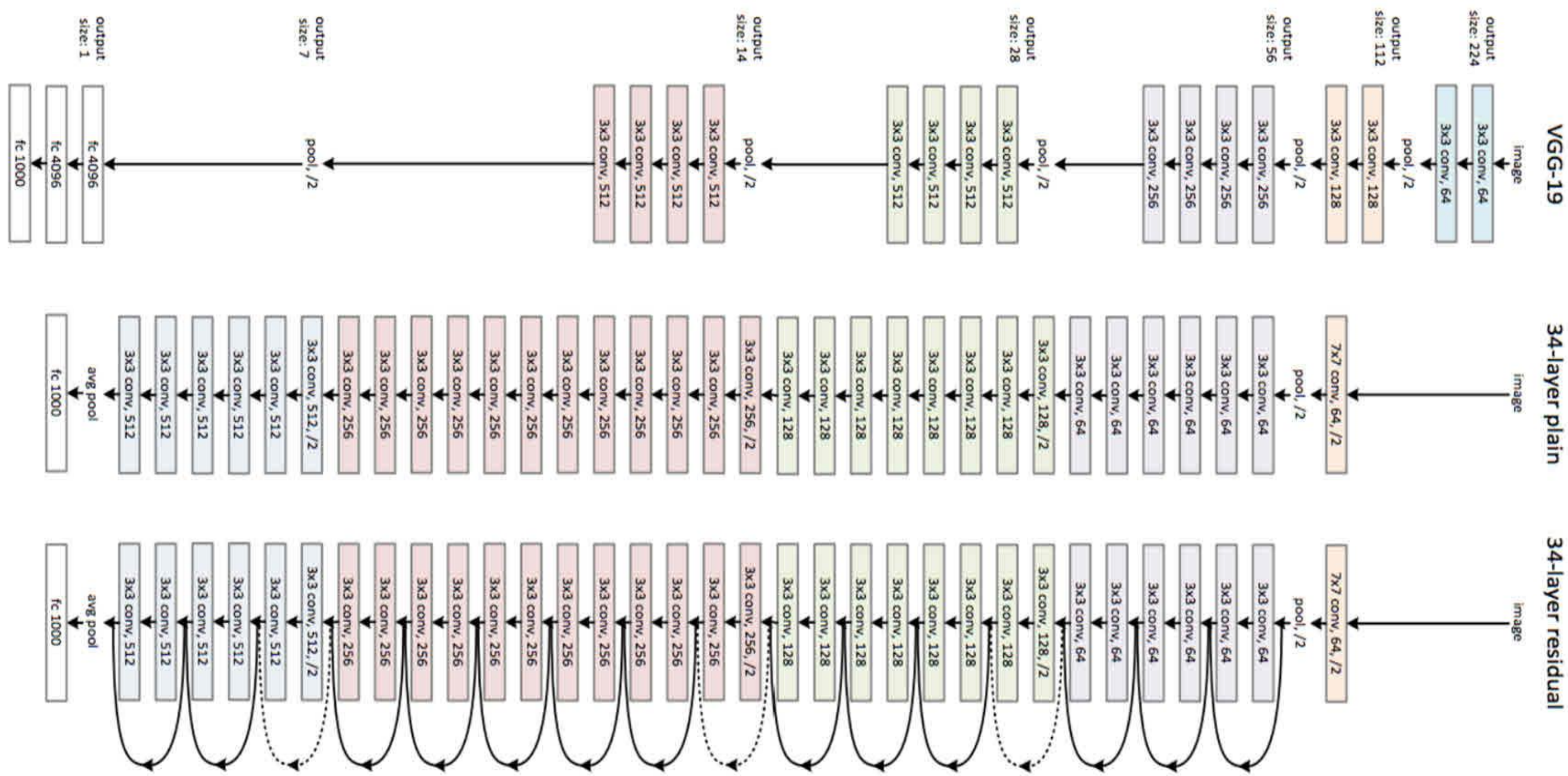


◆ Residual Block



05 ResNet

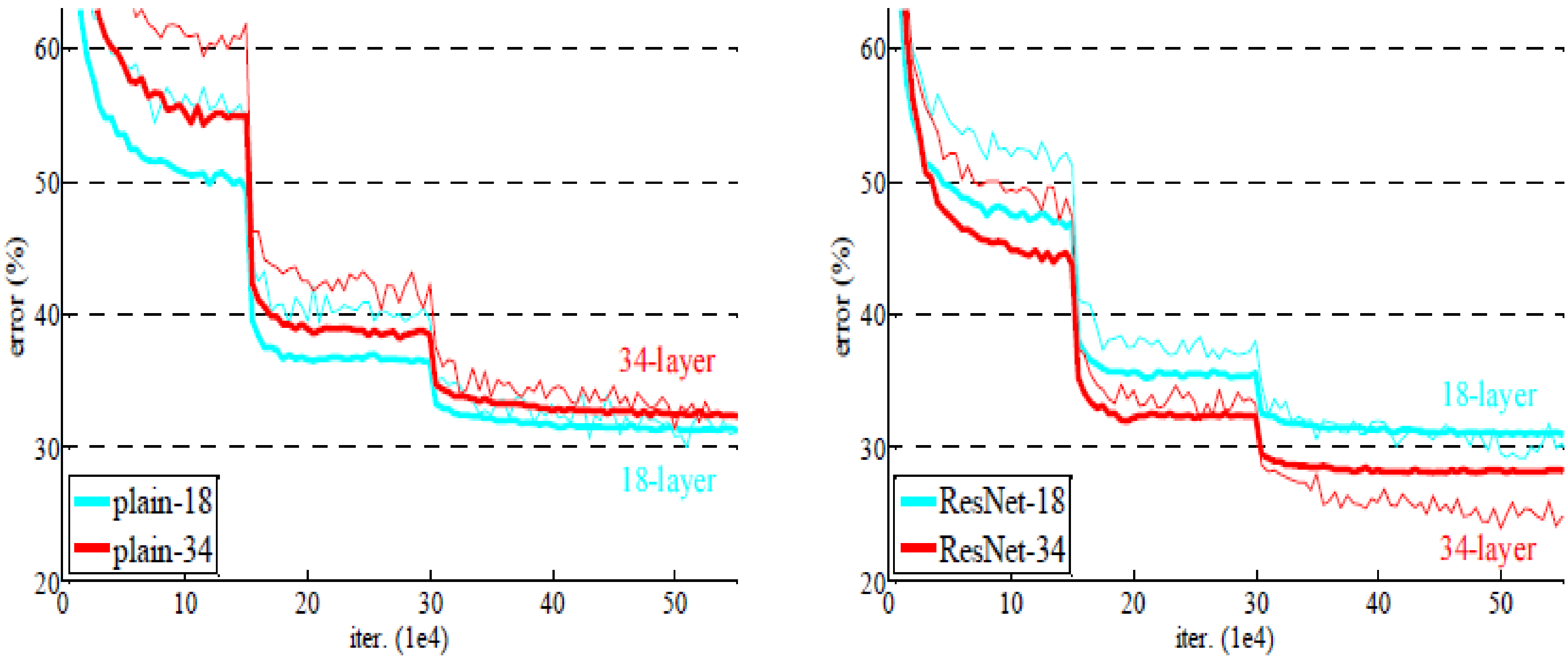
◆ The Architecture of ResNet



05

ResNet

◆ The Performance of ResNet



05

ResNet

◆ Code

```
for m in self.modules():
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
    elif isinstance(m, nn.BatchNorm2d):
        nn.init.constant_(m.weight, 1)
        nn.init.constant_(m.bias, 0)

# Zero-initialize the last BN in each residual branch,
# so that the residual branch starts with zeros, and each residual block behaves like an identity.
# This improves the model by 0.2~0.3% according to https://arxiv.org/abs/1706.02677
if zero_init_residual:
    for m in self.modules():
        if isinstance(m, Bottleneck):
            nn.init.constant_(m.bn3.weight, 0)
        elif isinstance(m, BasicBlock):
            nn.init.constant_(m.bn2.weight, 0)
```

ResNet

◆ Code

#Layer 구성

```
def _make_layer(self, block, planes, blocks, stride=1):
    downsample = None
    if stride != 1 or self.inplanes != planes * block.expansion:
        downsample = nn.Sequential(
            conv1x1(self.inplanes, planes * block.expansion, stride),
            nn.BatchNorm2d(planes * block.expansion),
        )

    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample))
    self.inplanes = planes * block.expansion
    for _ in range(1, blocks):
        layers.append(block(self.inplanes, planes))

    return nn.Sequential(*layers)
```

self.inplanes 값과 planes * block.expansion 값 비교
downsample 진행
layer append 이후 self inplanes 값 재설정

◆ Code

#Convolution 지난 후 size

```
def forward(self, x):
    x = self.conv1(x)
    #x.shape =[1, 16, 32,32]
    x = self.bn1(x)
    x = self.relu(x)
    #x = self.maxpool(x)

    x = self.layer1(x)
    #x.shape =[1, 128, 32,32]
    x = self.layer2(x)
    #x.shape =[1, 256, 32,32]
    x = self.layer3(x)
    #x.shape =[1, 512, 16,16]
    x = self.layer4(x)
    #x.shape =[1, 1024, 8,8]

    x = self.avgpool(x)
    x = x.view(x.size(0), -1)
    x = self.fc(x)

    return x
```

conv1 통과 이전 사이즈
x.shape=[batch,3*32*32]

ResNet

◆ Code

#ResNet 50

```
In [ ]: resnet50 = ResNet(resnet.Bottleneck, [3, 4, 6, 3], 10, True).to(device)
        #1(conv1) + 9(layer1) + 12(layer2) + 18(layer3) + 9(layer4) +1(fc)= ResNet50
```

```
In [ ]: resnet50
```

```
In [ ]: a=torch.Tensor(1,3,32,32).to(device)
        out = resnet50(a)
        print(out)
```

```
In [ ]: criterion = nn.CrossEntropyLoss().to(device)
        optimizer = torch.optim.SGD(resnet50.parameters(), lr = 0.1, momentum = 0.9, weight_decay=5e-4)
        lr_sche = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
```

#Loss & Accuracy

```
In [ ]: loss_plt = vis.line(Y=torch.Tensor(1).zero_(),opts=dict(title='loss_tracker', legend=['loss'], showlegend=True))
        acc_plt = vis.line(Y=torch.Tensor(1).zero_(),opts=dict(title='Accuracy', legend=['Acc'], showlegend=True))
```

◆ Code

#Acc Check Function 정의

```
In [ ]: def acc_check(net, test_set, epoch, save=1):
            correct = 0
            total = 0
            with torch.no_grad():
                for data in test_set:
                    images, labels = data
                    images = images.to(device)
                    labels = labels.to(device)
                    outputs = net(images)

                    _, predicted = torch.max(outputs.data, 1)

                    total += labels.size(0)
                    correct += (predicted == labels).sum().item()

            acc = (100 * correct / total)
            print('Accuracy of the network on the 10000 test images: %d %%' % acc)
            if save:
                torch.save(net.state_dict(), "./model/model_epoch_{}_acc_{}.pth".format(epoch, int(acc)))
            return acc
```

05 ResNet

◆ Code

#모델 학습 with (acc_check + model save)

In [ ]:
```
print(len(trainloader))
epochs = 150

for epoch in range(epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    lr_sche.step()
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = resnet50(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

#Optimizer, Criterion 설정

◆ Code

#모델 학습 with (acc_check + model save)

```
        # print statistics
        running_loss += loss.item()
        if i % 30 == 29:    # print every 30 mini-batches
            value_tracker(loss_plt, torch.Tensor([running_loss/30]), torch.Tensor([i + epoch*len(trainloader) ]))
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 30))
            running_loss = 0.0

    #Check Accuracy
    acc = acc_check(resnet50, testloader, epoch, save=1)
    value_tracker(acc_plt, torch.Tensor([acc]), torch.Tensor([epoch]))


print('Finished Training')
```

ResNet

◆ Code

#모델 성능 test

In [ ]:
```
correct = 0
total = 0

with torch.no_grad():
    for data in testloader:
        images, labels = data
        images = images.to(device)
        labels = labels.to(device)
        outputs = resnet50(images)

        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)

        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

06 Assignment

◆ VGG 19 – CIFAR10

- 코드 : https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-10_5_2_Aadvance-CNN(VGG_cifar10).ipynb

- 위의 코드에서 cfg 리스트를 **VGG 19**에 맞게 수정 후 **epoch=2**로 트레이닝
  16개의 convolution layer + 3개의 fully connected layer

- 정확도 보고할 필요 X

◆ ResNet 34 – CIFAR10

- 코드 : https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-10_6_2_Advance-CNN(ResNet_cifar10).ipynb

- block과 layer를 **ResNet 34**에 맞게 수정 후 **epoch=2**로 트레이닝
- BasicBlock + [3, 4, 6, 3] layer (layer는 ResNet50과 동일)

Reference

- Karen Simonyan & Andrew Zisserman, 「VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION」
- https://bskyvision.com/421
- https://bskyvision.com/504
- Stanford University School of Engineering 「 Lecture 9; CNN Architectures 」, https://www.youtube.com/watch?v=DAOcjicFr1Y

Thank You