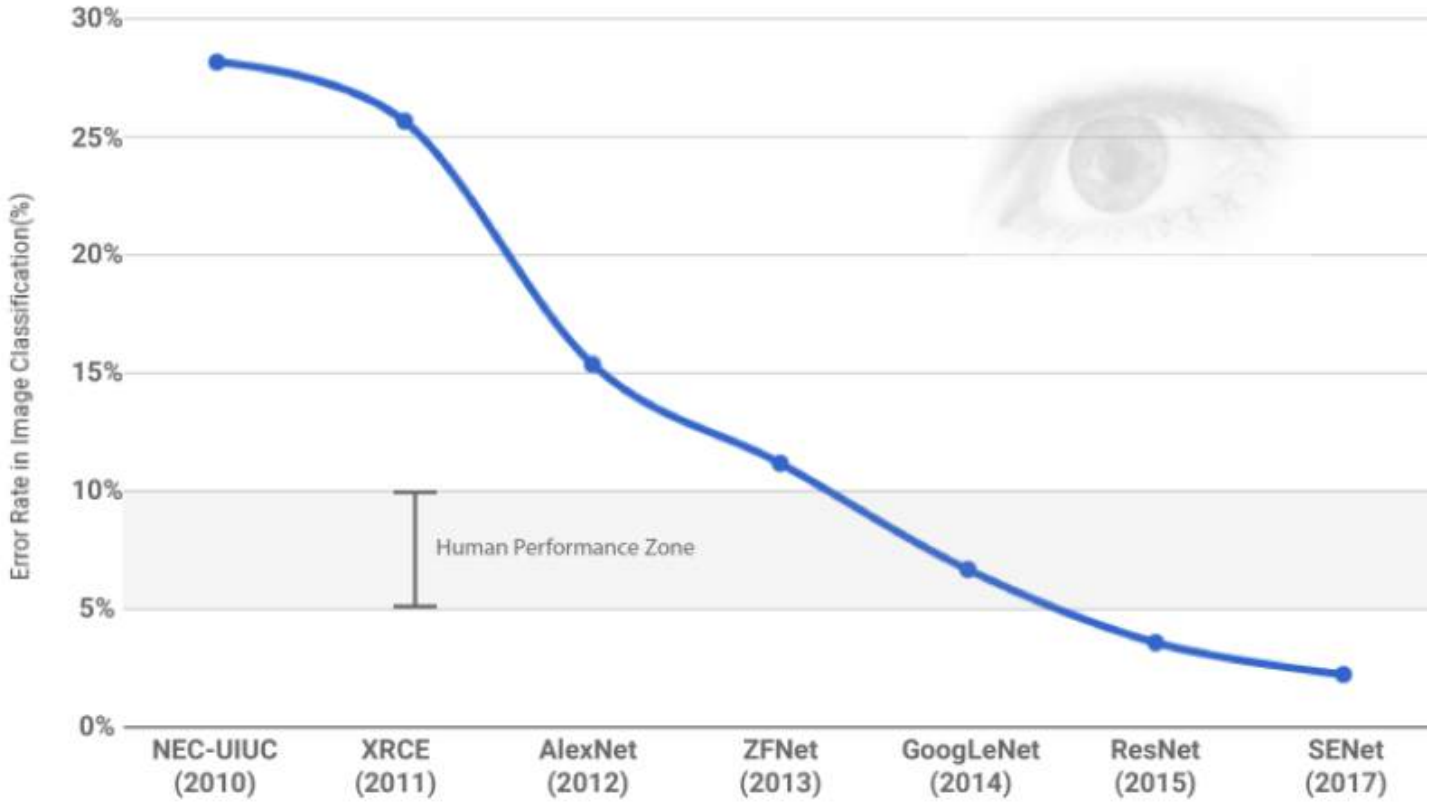
VGGNet 구현

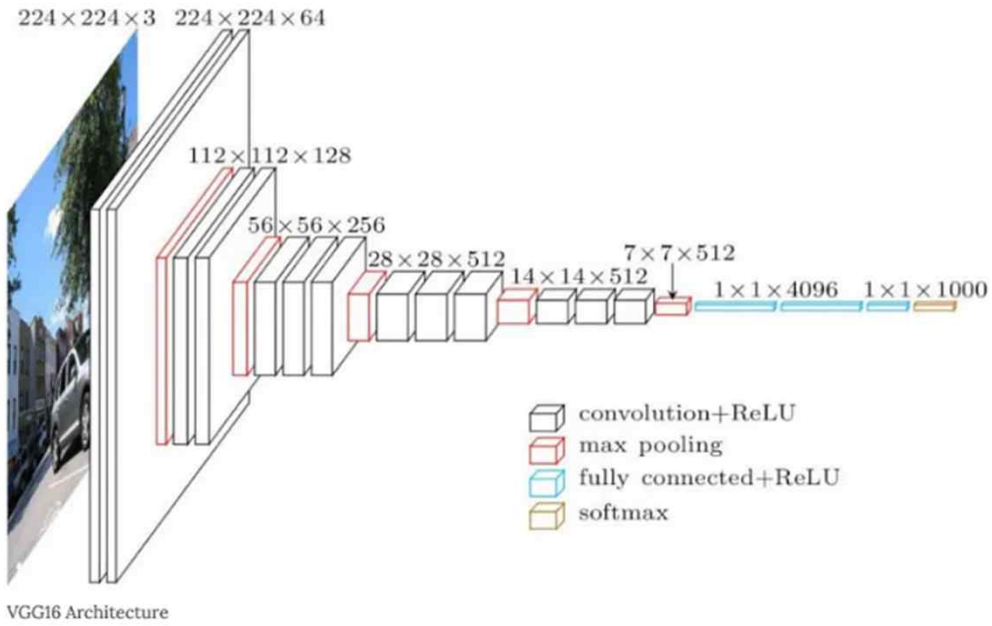
1. ILSVRC 자연영상 인식 경쟁

- CNN(Convolutional Neural Network)의 태동
  - ImageNet 데이터 셋을 이용한 영상 분류 세계대회 개최
    - 120만장의 학습 영상, 총 1000개의 class로 구성
  - 2012년 AlexNet: CNN이 DMLP보다우월한 성능을 보임을 증명



2. VGGNet

- VGGNet은 ISRVRC 대회에서 Top-5 정확도 92.7%를 달성, 대표적 딥러닝 모델로 자리매김
  - (참고 논문) VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION, ICLR2015
  - 층의 개수에 따라 VGG-13, VGG-16, VGG-19 모델이 있음(VGG-16이 가장 많이 활용됨)



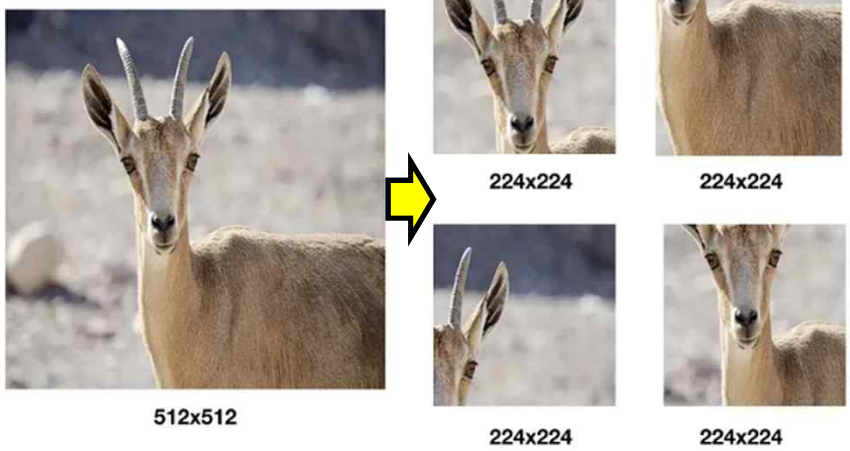
VGG16 Architecture

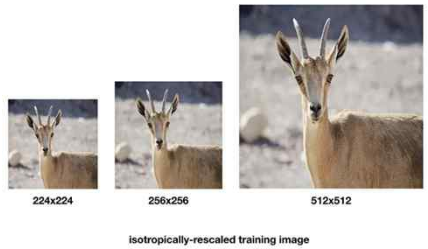
이미지 출처 : https://neurohive.io/en/popular-networks/vgg16/

https://medium.com/@msmapark2/vgg16-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-very-deep-convolutional-networks-for-large-scale-image-recognition-6f748235242a

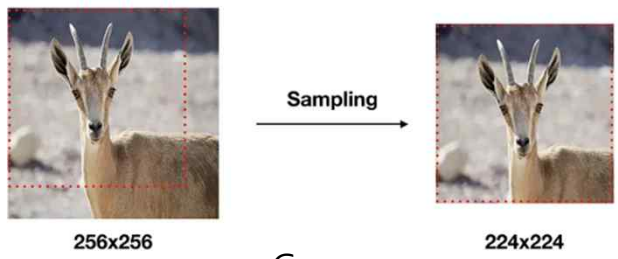
2. VGGNet

- VGG 모델의 특징
  - 2012년 ILSVRC 우승모델인 AlexNet보다 2배 이상 깊은 네트워크로, 학습에 성공하여 오차율을 획기적으로 감소(16.4% -> 7.3%)
- 깊은 신경망 학습이 성공할 수 있었던 요인
  - 모든 Convolution 층에 3x3필터를 사용
    - 7x7 filter 1개 대신 3x3필터 3개 사용 가능 → 비선형성 증가
  - Activation 함수로 ReLU를 사용
  - 데이터 증강기법(Resizing + Crop) 사용





Resizing

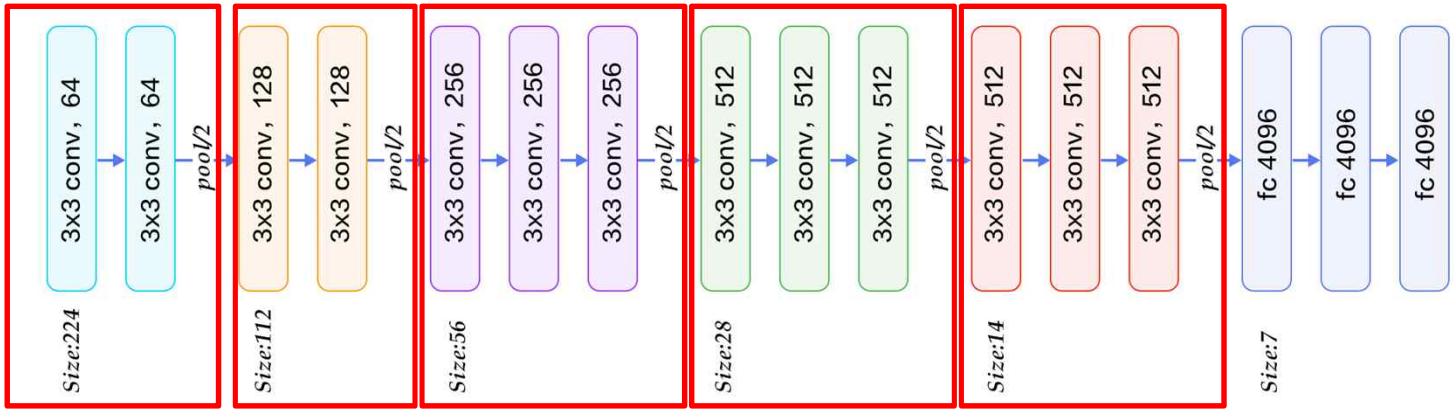


Crop

https://medium.com/@msmapark2/vgg16-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-very-deep-convolutional-networks-for-large-scale-image-recognition-6f748235242a

2. VGGNet

- VGG 모델 구조

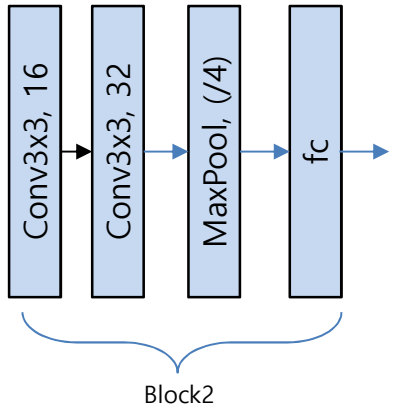


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide

2. VGGNet

- Simplified VGGNet 구현
  - 모델: block2 1개로 구성된 VGGNet
  - 데이터: CIFAR10



- Transforms
  - DataLoader에서 data를 불러 올 때 전처리 수행

```
1 import torch
2 import torch.nn as nn
3 import torchvision
4 import torchvision.transforms as transforms
5
6
7 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
8
9 num_epochs = 80
10 learning_rate = 0.001
11
12 transform = transforms.Compose([
13     transforms.Pad(4),
14     transforms.RandomHorizontalFlip(),
15     transforms.RandomCrop(32),
16     transforms.ToTensor()])
17
```

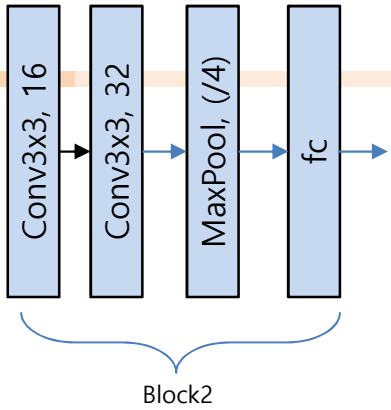
2. VGGNet

- CIFAR-10 data 불러오기
  - 참고: RandomCrop은 train_dataset에만 적용

```
18 # CIFAR-10 dataset
19 train_dataset = torchvision.datasets.CIFAR10(root='../../data/',
20                                               train=True,
21                                               transform=transform,
22                                               download=True)
23
24 test_dataset = torchvision.datasets.CIFAR10(root='../../data/',
25                                              train=False,
26                                              transform=transforms.ToTensor())
27
28 # Data loader
29 train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
30                                            batch_size=100,
31                                            shuffle=True)
32
33 test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
34                                           batch_size=100,
35                                           shuffle=False)
36
37
```

2. VGGNet

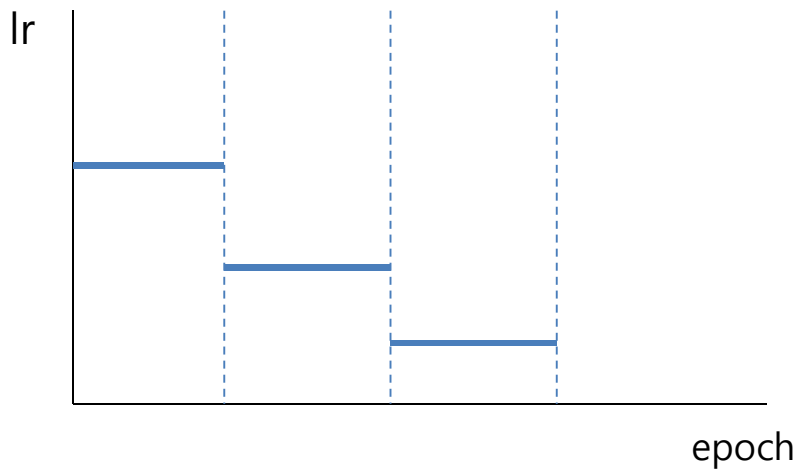
- VGG Model 생성
  - 참고: RandomCrop은 train_dataset에만 적용




```
38 class VGG(nn.Module):
39     def __init__(self):
40         super(VGG, self).__init__()
41         self.maxpool = nn.MaxPool2d(4)
42         self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1)
43         self.conv2 = nn.Conv2d(in_channels = 16, out_channels=32, kernel_size=3, stride=1)
44         self.relu = nn.ReLU(inplace = True)
45         self.fc = nn.Linear(1568 , 10)
46 
47     def forward(self, x):
48         x = self.relu(self.conv1(x))
49         x = self.relu(self.conv2(x))
50         x = self.maxpool(x)
51         x = x.view(x.size(0), -1)
52 #       print(x.shape)
53         x = self.fc(x)
54 
55         return x
56 
57 
58 model = VGG().to(device)
```

2. VGGNet

- 학습을 위한 하이퍼라리미터 세팅
  - Adam Optimizer 사용
  - update_lr(): Learning rate scheduling을 위해 학습 중 lr 업데이트 함수 구현




```
60 criterion = nn.CrossEntropyLoss()
61 optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
62
63 def update_lr(optimizer, lr):
64     for param_group in optimizer.param_groups:
65         param_group['lr'] = lr
66
```

2. VGGNet

- Training Phase

```
total_step = len(train_loader)
curr_lr = learning_rate
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        #print(images.size())
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 ==0:
            print ("Epoch [{}/{}], step [{}/{}] Loss: {:4f}".format(epoch+1, num_epochs, i+1, total_step, loss.item()))

    if (epoch+1) % 20 == 0:
        curr_lr /= 3
        update_lr(optimizer, curr_lr)

```

2. VGGNet

- Test Phase

```
88 model.eval()
89 with torch.no_grad():
90     correct = 0
91     total = 0
92     for images, labels in test_loader:
93         images = images.to(device)
94         labels = labels.to(device)
95         outputs = model(images)
96         _, predicted = torch.max(outputs.data, 1)
97         total += labels.size(0)
98         correct += (predicted == labels).sum().item()
99 
100     print('Accuracy of the odel on the test images: {} %'.format(100*correct/total))
```

실습

- 1. Training/Test를 for문 안에서 함께 수행하도록 코드를 변경하시오.
  - 아래의 for문이 동작하도록 train(), test()함수를 작성할 것

```
for epoch in range(1, 10):
    train(epoch)
    test()
```

실습

- 2. 데이터 전처리로 정규화(normalization)를 수행하시오.
  - (방법) R, G, B 각 채널에 대한 모든 Training data의 평균 pixel 밝기값(μ_R, μ_G , μ_B)과 표준편차(σ_R, σ_G , σ_B)를 구한다음, R, G, B 채널에 각각 독립적으로 정규화를 수행할 것
  - Transforms.Normalize() 함수를 사용할 것

(참고)

- 식 (5.9)의 정규화는 규모 문제와 양수 문제를 해결해줌
  - 특징별로 독립적으로 적용(**i.i.d. condition**: independent and identically distributed condition)

$$x_i^{new} = \frac{x_i^{old} - \mu_i}{\sigma_i} \qquad (5.9)$$

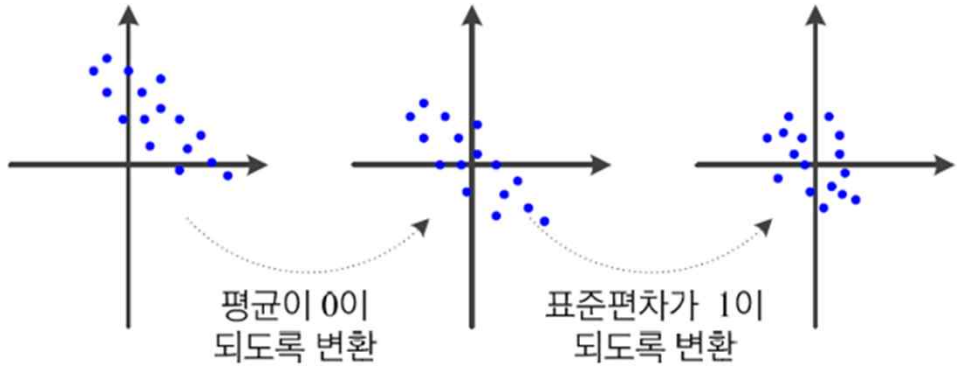
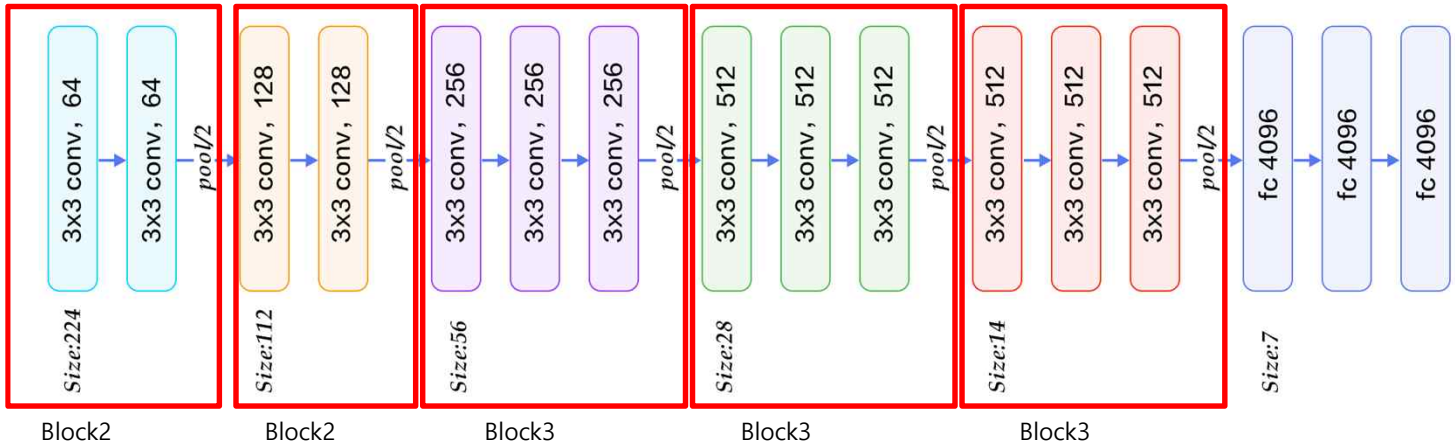


그림 5-7 표준점수로 변환

실습

- 3아래 VGG모델을 구현하고 CIFAR-100 데이터를 학습하시오. 학습 결과를 report 하시오
  - Block2->Block3->fc->fc 로 구성할 것(총 7개 layers)
  - (Block2는 2개의 convolution layer, Block3은 3개의 convolution layer로 구성됨)
  - (채널의 개수 및 MaxPooling의 stide를 적절하게 설정할 것)

(참고)



실습

- 4. 이전 문제에서 구현한 VGG모델을 튜닝하시오
  - 채널의 개수 및 MaxPooling의 stide에 대한 하이퍼파라미터 튜닝을 수행하시오(채널 개수 및 MaxPooling의 stride값에 대한 적절한 후보군을 설정한 다음 Search 수행)
  - 시간이 많이 걸릴 수 있으므로 경희대 AI센터의 GPU를 대여하여 학습에 활용할 것
    - 학교 공용 GPU 클러스터 활용법)은 과목 홈페이지를 참고할 것