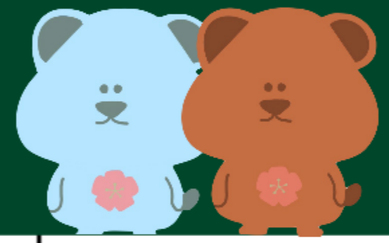
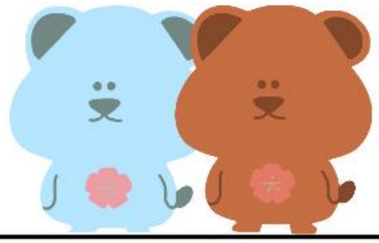


3주차 발표

ML팀 박지운 오연재 최하경

EWHA EURON

목차



EWHA EURON

1. 합성곱 신경망(CNN)

EWHA EURON

#1.1 합성곱 신경망 구조

합성공 신경망(CNN)

- 음성 이미지/영상 인식에 많이 사용

크게 5개의 층으로 구성

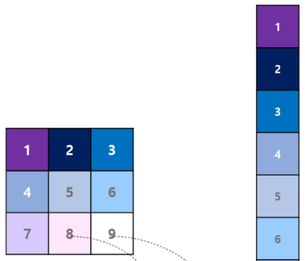
1) 입력+ 2) 합성곱층(렐루)+ 3) 풀링층+ 4) 완전연결층+ 5) 출력층

합성곱층과 풀링층을 거치면서 입력이미지의 특성 벡터를 추출

특성 벡터는 완전 연결층을 거치면서 1차원 벡터로 변환+출력층에서 소프트맥스 함수 이용

사용하는 이유: 3x3 배열을 오른쪽과 같이 펼쳐서(flatten) 하여 각 픽셀에 가중치를 곱하는 형식

Flatten하면서 공간적 구조를 무시하는데 이런 공간적인 특성을 유지하고자 CNN이 등장하게 됨



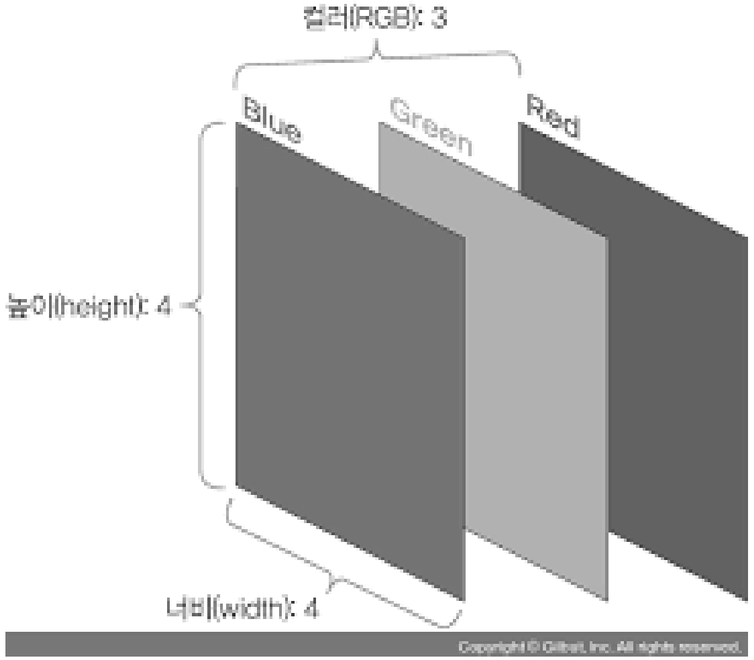
#1.1 합성곱 신경망 구조

입력층 - 입력 이미지가 최초로 들어오는 층

형식: (weight, width, channel)

channel: 3차원 데이터(그레이 스케일(흑백) 이면 1, 컬러(RGB)면 3, 컬러인 경우 R,G,B 각각에 대해 생기므로 3개)

높이4, 너비4, RGB 채널인 경우: (4,4,3) 으로 표시



https://yjjo.tistory.com/8

EWHA EURON

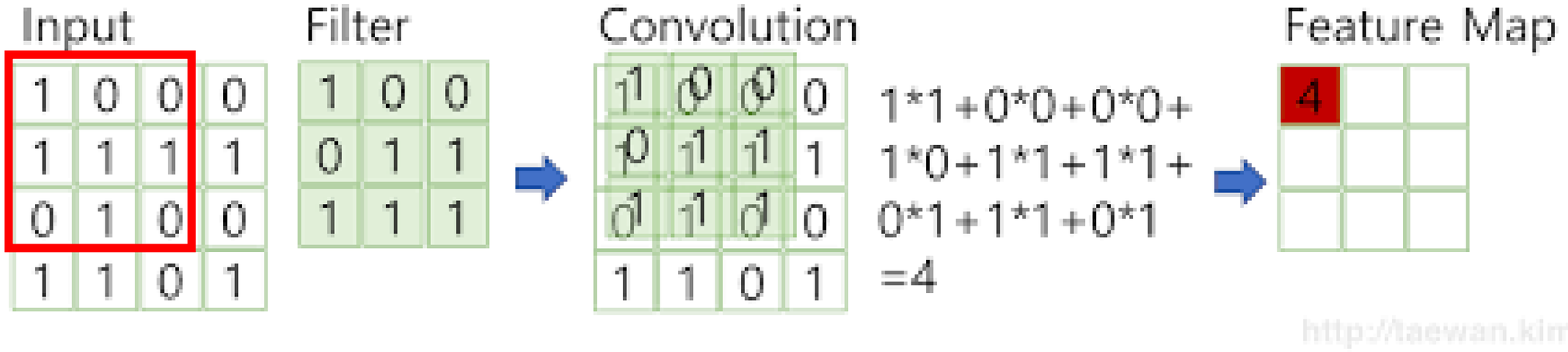
#1.1 합성곱 신경망 구조

합성곱층 – 입력 데이터에서 특성을 추출하는 역할

특성 추출: 입력 이미지가 입력층에 들어온 후, 특성을 감지하기 위해 커널/필터 를 이용한다.

커널/필터는 이미지의 모든 영역을 훑으면서 특성을 추출. 추출된 결과물을 특성맵이라고 한다. 커널은 (3,3)이나 (5,5)가 일반적이다. Stride(지정된 간격) 의 크기에 따라 크기 만큼 이동하면서 특성을 추출.

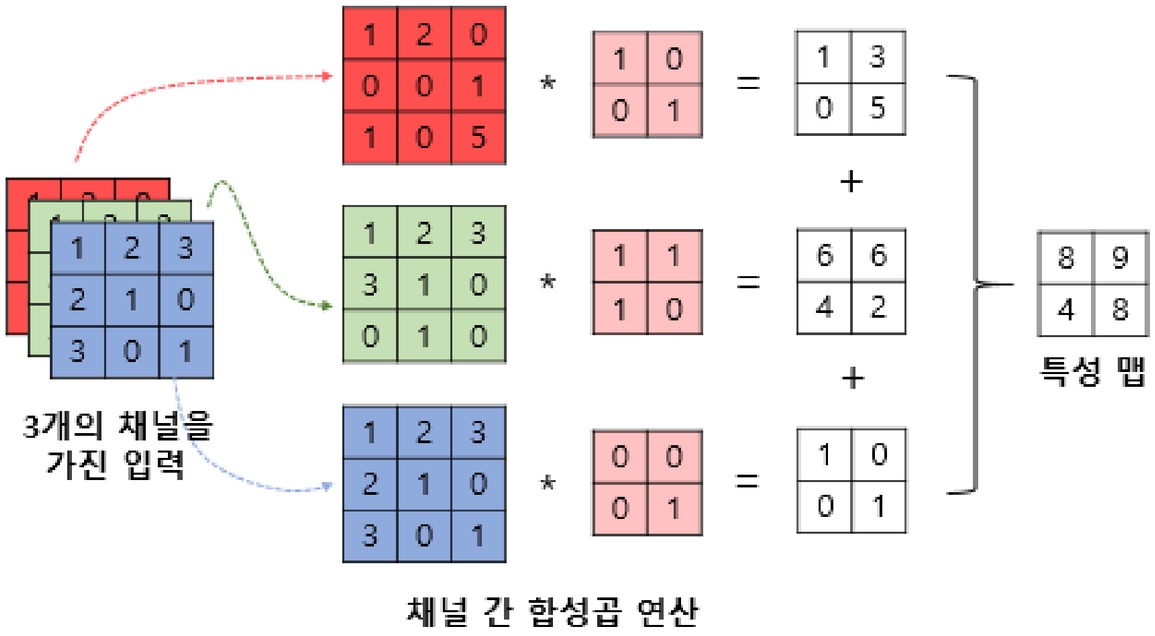
Ex) 입력 이미지에 3X3 필터, stride는 1로 적용, **그레이 스케일에 적용**(channel:1, 필터 개수: 1)



#1.1 합성곱 신경망 구조

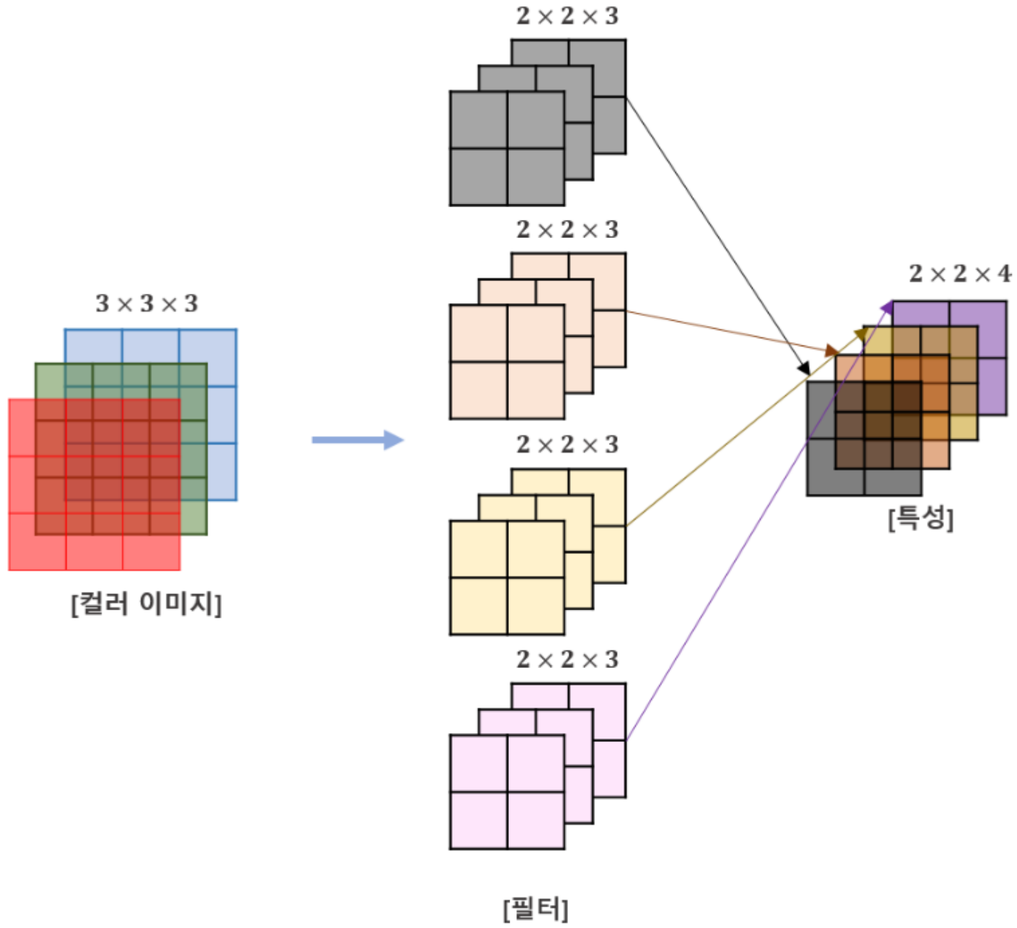
Ex) 입력 이미지에 2x2 필터, stride는 1로 적용,

컬러 이미지에 적용(channel:3, 필터 개수: 1)



Ex) 입력 이미지에 2X2 필터, stride는 1로 적용,

컬러 이미지에 적용(channel:3, 필터 개수: 4)

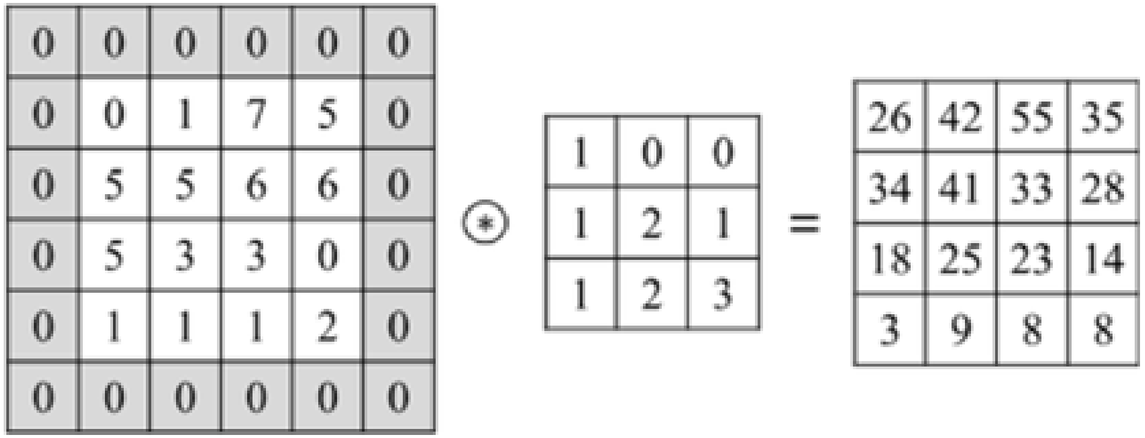


#1.1 합성곱 신경망 구조

*** 패딩**

- 합성곱 연산을 수행하기 전, 입력데이터 주변을 특정값으로 채워 늘리는 것을 말한다. 주로 출력 데이터의 공간적 크기를 조절하기 위해 사용, 주로 zero-padding

- 이미지의 각 모서리 부분에 특정 값(ex: 0)을 채워 넣는 것
- 필터를 거쳐 나온 특성맵의 크기가 작아지는 것을 방지
- Edge pixel data를 충분히 활용하기 위함



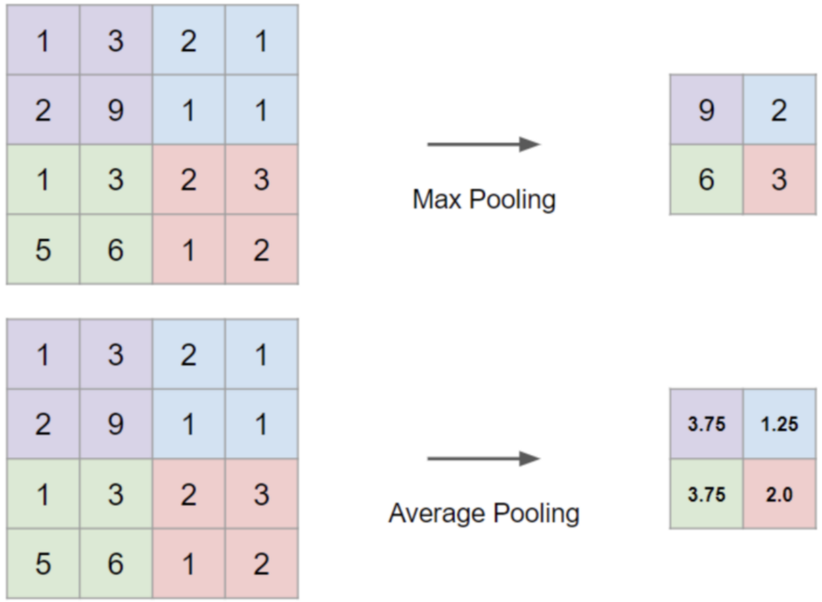
입력 데이터: W1 x H1 x D1
하이퍼 파라미터: **필터 개수**(K), **필터 크기**(F), **스트라이드**(S), **패딩**(P)
출력 데이터: W2=(W1-F+2P)/S+1, H2=(H1-F+2P)/S+1, D2=K

#1.1 합성곱 신경망 구조

풀링층

특성 맵의 차원을 다운 샘플링하여 연산량을 감소, 주요한 특성 벡터를 추출하여 학습을 효과적으로 한다.
최대 풀링(max pooling): 대상 영역에서 최댓값
평균 풀링(average pooling): 대상 영역에서 평균 반환
* 주로 CNN에서는 특성 값을 유지하고자 하여 평균 풀링 보다는 최대 풀링을 이용한다.

Ex) 필터 크기: 2, 스트라이드: 2



입력 데이터: W1 x H1 x D1
하이퍼 파라미터: 필터 크기(F), 스트라이드(S)
출력 데이터: W2=(W1-F)/S+1, H2=(H1-F)/S+1, D2=D1
파라미터 수: 0 (max를 취하거나 average를 취하는 형식이므로 사이즈가 바뀔 분, 새로운 learning이 발생하지 않는다.)

EWHA EURON

#1.1 합성곱 신경망 구조

완전 연결층(Fully connected layer)

합성곱층과 풀링층을 거치면서 축소된 특성맵은 완전연결층으로 전달된다.

3차원 벡터가 1차원 벡터로 펼쳐지게 된다.

1. flatten : 각 레이어를 1차원 벡터로 변환

2. fully-conneced layer : 1차원 벡터로 변환된 레이어를 하나의 벡터로 연결 (각 층의 노드들은 하나로 연결)

3. Softmax 함수를 이용해 가장 확률이 높은 class를 output으로 분류합니다.

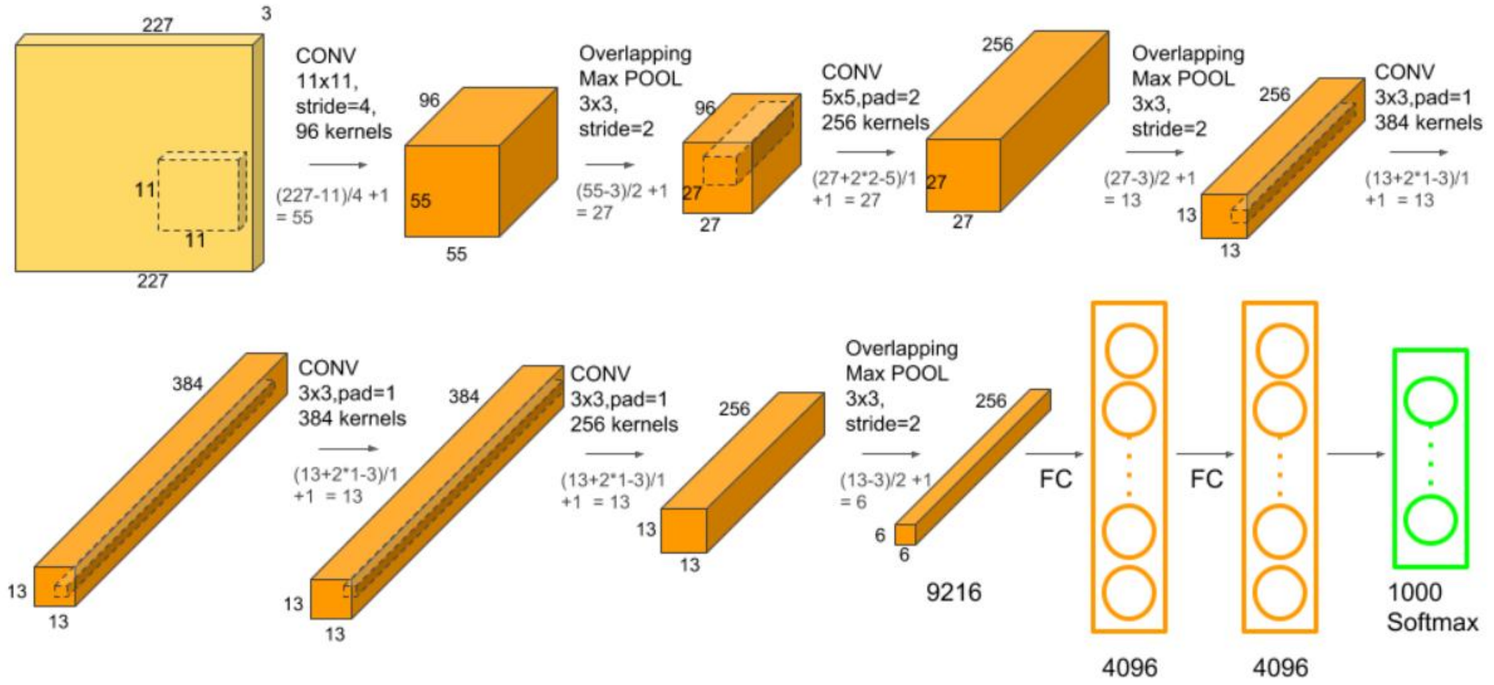
출력층

소프트맥스 활성화 함수 이용, 입력 받은 값을 0~1사이로 출력/이미지가 각 레이블에 속할 확률 값이 출력/가장 높은 확률 레이블이 선택

#1.1 합성곱 신경망 구조

Alex Net 파라미터 계산

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**



https://seongkyun.github.io/study/2019/01/25/num_of_parameters/

#1.1 합성곱 신경망 구조

1D, 2D, 3D 합성곱(이동하는 방향의 수와 출력 형태에 따라 분류)

1. 1D 합성곱: 필터가 시간을 축으로 좌우로만 이동이 가능한 합성곱, 출력 형태가 1D형태이다.
2. 2D 합성곱: 필터가 방향 두 개로 이동하는 형태이다, 출력 형태가 2D형태이다.
3. 3D 합성곱: 필터가 세 개의 방향으로 이동한다, 출력 형태가 3D 형태이다.
4. 3D 입력을 갖는 2D 합성곱: 입력이 3D 형태임에도 출력 형태는 2D 형태

필터에 대한 길이(L)이 입력 채널의 길이와 같아야 하기 때문에 이와 같은 합성곱 형태가 만들어진다. 필터는 두 방향으로 이동하며 출력 형태는 2D 행렬이 된다. 대표적인 합성곱으로는 LeNet-5, VGG가 있다.

5. 1x1 합성곱: 3D 형태로 입력된다. 연산량이 감소되는 효과가 있고, 대표적인 사례로는 GoogleNet이 있다.

2. 합성곱 신경망 맛보기

EWHA EURON

#2.1 환경 설정

1. 필요 라이브러리 호출 (각 라이브러리는 코드에서 쓰일때 설명)

```
#라이브러리 호출
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.functional as F

import torchvision
import torchvision.transforms as transforms #데이터 전처리를 위해 사용하는 라이브러리
from torch.utils.data import Dataset, DataLoader
```

2.GPU 장치 확인

```
#GPU 장치 확인
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
device
```

```
device(type='cpu')
```

#2.1 환경 설정

3. fashion_mnist 데이터셋 다운

```
#fashion_mnist dataset
train_dataset = torchvision.datasets.FashionMNIST("/chap05/data", download=True, transform=transforms.Compose([transforms.ToTensor()]))
test_dataset = torchvision.datasets.FashionMNIST("/chap05/data", download=True, train = False, transform=transforms.Compose([transforms.ToTensor()]))
```

- torch vision.datasets 하위에 다양한 데이터셋 존재 (CIFAR, COCO, ImageNet 등)
- transforms.ToTensor() : 이미지를 텐서(0~1) 형태로 변환

4.데이터셋 데이터로더로 전달

```
#fashion_mnist 데이터를 데이터로더에 전달
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size = 100)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size = 100)
```

- DataLoader()를 사용하여 원하는 크기의 배치 단위로 데이터를 불러오거나, 데이터를 shuffle할 수 있음
- 위의 코드에선 batch_size = 1000이므로, 100개 단위로 데이터를 묶어서 불러옴

#2.2 분류에 사용될 클래스 정의

```
#분류에 사용될 클래스 정의
labels_map = {0: 'T-shirt', 1: 'Trouser', 2: 'Pullover', 3: 'Dress', 4: 'Coat', 5: 'Sandal', 6: 'Shirt', 7: 'Sneaker', 8: 'Bag', 9: 'Ankle Boot'}

fig = plt.figure(figsize = (8,8))
columns = 4
rows = 5
for i in range(1, columns*rows + 1):
  img_xy = np.random.randint(len(train_dataset))
  img = train_dataset[img_xy][0][0,:,:]
  fig.add_subplot(rows, columns, i)
  plt.title(labels_map[train_dataset[img_xy][1]])
  plt.axis('off')
  plt.imshow(img, cmap='gray')
plt.show()
```

train_dataset을 이용해 3차원 배열 생성 후 20개의 이미지 시각적 생성

#2.3 심층신경망 모델 생성

```python
#심층 신경망 모델 생성
class FashionDNN(nn.Module):
  def __init__(self):
    super(FashionDNN, self).__init__()
    self.fc1 = nn.Linear(in_features=784, out_features=256)
    self.drop = nn.Dropout(0.25)
    self.fc2 = nn.Linear(in_features = 256, out_features = 128)
    self.fc3 = nn.Linear(in_features=128, out_features=10)

  def forward(self, input_data):
    out = input_data.view(-1, 784)
    out = F.relu(self.fc1(out))
    out = self.drop(out)
    out = F.relu(self.fc2(out))
    out = self.fc3(out)
    return out
```

- nn은 딥러닝 모델 구성에 필요한 모듈이 모여 있는 패키지 (nn.Linear은 단순 선형 회귀 모델)
  - in_features는 입력의 크기, out_features는 출력의 크기

- nn.Dropout(p)는 p만큼의 비율로 텐서가 0이 되고, 0이 되지 않는 값들은 기존 값에 (1/(1-p)) 만큼 곱해져 커짐

- forward()함수는 학습 데이터를 입력 받아서 순전파 학습 진행 -> 반드시 forward라는 이름의 함수이어야 한다!

#2.4 심층신경망 파라미터 정의

```
#심층 신경망에서 필요한 파라미터 정의
learning_rate = 0.001
model = FashionDNN()
model.to(device)

criterion = nn.CrossEntropyLoss() #분류 문제에서 사용되는 손실 함수
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
print(model)
```

```
FashionDNN(
  (fc1): Linear(in_features=784, out_features=256, bias=True)
  (drop): Dropout(p=0.25, inplace=False)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=10, bias=True)
)
```

- 학습률은 0.001
- 최적화 방식은 Adam
- 옵티마이저에 앞서 생성했던 DNN 모델 넣어주기

#2.5 심층신경망 모델 학습

```
#심층 신경망을 이용한 모델학습
num_epochs = 5
count = 0
loss_list = []
iteration_list = []
accuracy_list = []

predictions_list = []
labels_list = []

for epoch in range(num_epochs):
  for images, labels in train_loader:
    images, labels = images.to(device), labels.to(device)

    train = Variable(images.view(100,1,28,28)) #autograd -> 역전파를 위한 미분값 자동으로 계산
    labels = Variable(labels)

    outputs = model(train)
    loss = criterion(outputs, labels)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    count += 1

    if not (count % 50): #count를 50으로 나눴을 때 나머지가 0이 아니라면 실행
      total = 0
      correct = 0
      for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        labels_list.append(labels)
        test = Variable(images.view(100,1,28,28))
        outputs = model(test)
        predictions = torch.max(outputs, 1)[1].to(device)
        predictions_list.append(predictions)
        correct += (predictions == labels).sum()
        total += len(labels)

      accuracy = correct * 100 / total
      loss_list.append(loss.data)
      iteration_list.append(count)
      accuracy_list.append(accuracy)

    if not (count % 500):
      print("iteration:{}, loss: {}, accuracy: {}%".format(count, loss.data, accuracy))
```

- Autograd : 자동 미분을 수행하는 파이토치의 패키지 => Variable을 사용해서 역전파를 위한 미분 값을 자동으로 계산해주는 유용한 패키지!

- 분류 문제는 예측 성능을 '정확도'로 표기

```
iteration:500, loss: 0.4907730519771576, accuracy: 85.54000091552734%
iteration:1000, loss: 0.42999595403671265, accuracy: 85.4800033569336%
iteration:1500, loss: 0.2851185202598572, accuracy: 86.08999633789062%
iteration:2000, loss: 0.31032606959342957, accuracy: 86.5199966430664%
iteration:2500, loss: 0.23627763986587524, accuracy: 86.81999969482422%
iteration:3000, loss: 0.22662794589996338, accuracy: 87.05999755859375%
```

Accuracy는 86%정도 기록..

#2.6 합성곱 네트워크 생성

```python
#합성곱 네트워크 생성
class FashionCNN(nn.Module):
  def __init__(self):
    super(FashionCNN, self).__init__()
    self.layer1 = nn.Sequential(
        nn.Conv2d(in_channels = 1, out_channels = 32, kernel_size = 3, padding=1),
        nn.BatchNorm2d(32),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2)
    )
    self.layer2 = nn.Sequential(
        nn.Conv2d(in_channels = 32, out_channels = 64, kernel_size = 3),
        nn.BatchNorm2d(64),
        nn.ReLU(),
        nn.MaxPool2d(2)
    )
    self.fc1 = nn.Linear(in_features=64*6*6, out_features=600)
    self.drop = nn.Dropout(0.25)
    self.fc2 = nn.Linear(in_features = 600, out_features = 120)
    self.fc3 = nn.Linear(in_features=120, out_features=10) #마지막 계층의 out_features는 클래스 개수를 의미

  def forward(self, x):
    out = self.layer1(x)
    out = self.layer2(out)
    out = out.view(out.size(0), -1)
    out = self.fc1(out)
    out = self.drop(out)
    out = self.fc2(out)
    out = self.fc3(out)
    return out
```

- nn.Sequential : 계층을 차례대로 쌓을 수 있도록 수식과 활성화 함수 연결해주는 역할
- 합성곱층(Conv2d) : 합성곱 연산을 통해 이미지의 특징 추출
  - in_channels: 입력 채널 수 (흑백은 1, 컬러는 3)
  - out_channels: 출력 채널 수
  - kernel_size: 커널 크기, 커널은 입력 데이터를 스트라이드 간격으로 순회하며 합성곱 계산 = 필터
- BatchNorm2d: 배치 단위별로 데이터를 정규화
- MaxPool2d: 풀링 -> 이미지 크기를 축소시키는 용도로 사용

#2.7 합성곱 네트워크 파라미터 및 모델 학습

```
#합성곱 네트워크에서 필요한 파라미터 정의
learning_rate = 0.001
model = FashionCNN()
model.to(device)

criterion = nn.CrossEntropyLoss() #분류 문제에서 사용되는 손실 함수
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
print(model)
```

```
FashionCNN(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=2304, out_features=600, bias=True)
  (drop): Dropout(p=0.25, inplace=False)
  (fc2): Linear(in_features=600, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

```
iteration:500, loss: 0.45735305547714233, accuracy: 87.98999786376953%
iteration:1000, loss: 0.4097321033477783, accuracy: 87.73999786376953%
iteration:1500, loss: 0.3181893825531006, accuracy: 87.44999694824219%
iteration:2000, loss: 0.1479538381099701, accuracy: 89.5199966430664%
iteration:2500, loss: 0.15383018553256989, accuracy: 90.08000183105469%
iteration:3000, loss: 0.16585248708724976, accuracy: 90.12000274658203%
```

=> 심층신경망과 비교해 정확도가 약간 높음
실제로 이미지 데이터가 많아지면 단순 DNN으로는 정확한 분류가 불가능하므로 CNN이 필요함

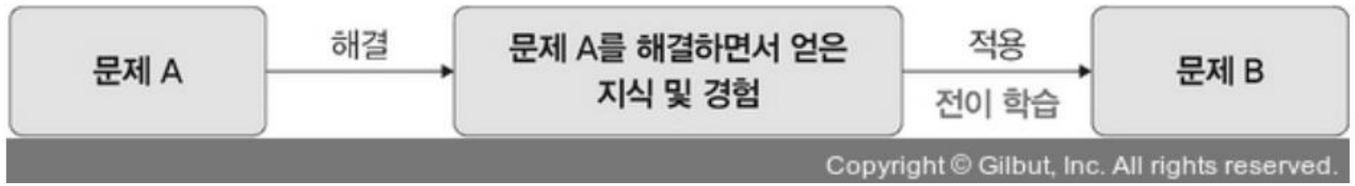
3. 전이 학습

EWHA EURON

#3.1.1 전이 학습

📌 전이 학습(transfer learning)이란?

- 아주 큰 데이터셋으로 훈련된 모델의 가중치를 가져와 우리가 해결하려는 과제에 맞게 보정해 사용하는 것
- 비교적 적은 수의 데이터를 갖고 우리가 원하는 과제 수행 가능



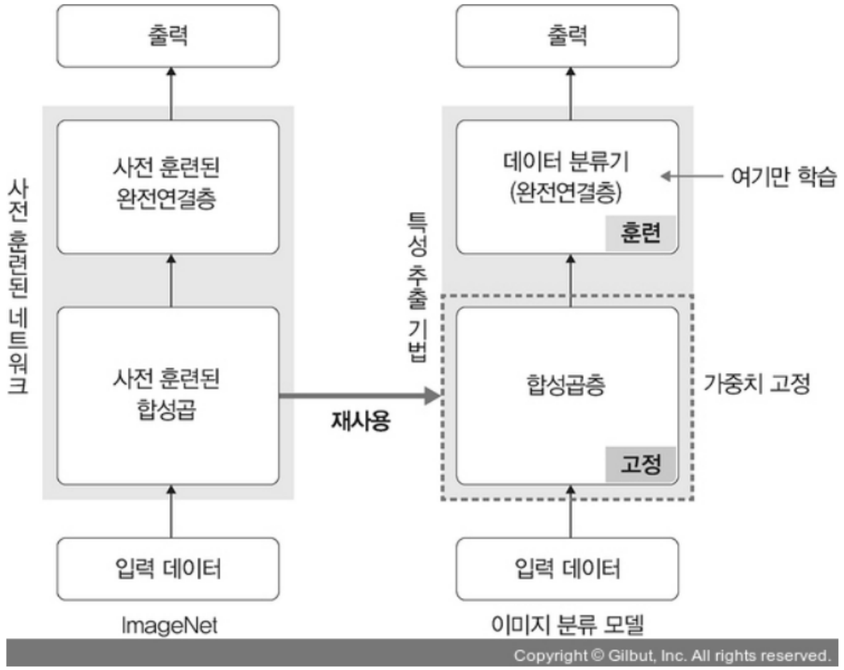
- 전이 학습을 위한 두 가지 방법 – 특성 추출과 미세 조정 기법

#3.1.2 특성 추출

📌 특성 추출 (feature extraction)

- ImageNet 데이터셋으로 사전 훈련된 모델을 가져온 후 완전연결층(FC layer) 부분만 새로 만드는 방법

  -> 학습할 때 마지막 FC layer만 학습하고 나머지 계층들은 학습 X
- ex. Xception, Inception V3, ResNet50, VGG16, VGG19, MobileNet



▲ 그림 5-30 특성 추출 기법

#3.1.2 특성 추출

📌 특성 추출 예시

① 훈련 데이터 전처리

```
## train data 전처리

data_path = 'ML_5장/train/'
transform = transforms.Compose(
                [
                    transforms.Resize([256, 256]),
                    transforms.RandomResizedCrop(224),
                    transforms.RandomHorizontalFlip(),
                    transforms.ToTensor(),
                ])
train_dataset = torchvision.datasets.ImageFolder(
    data_path,
    transform=transform
)
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=32,
    num_workers=8,
    shuffle=True
)

print(len(train_dataset))
```

```
385
```

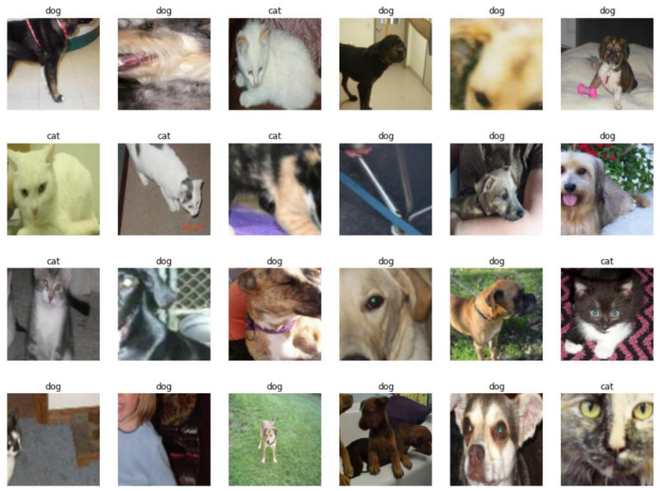
- torchvision.transform
  - 이미지 데이터를 모델의 입력으로 사용할 수 있게끔 변환
- torch.datasets.ImageFolder()
  - 데이터로더가 데이터를 불러올 대상 및 경로와 전처리 방법을 정의
- torch.utils.data.DataLoader()
  - 불러올 데이터셋 + 한번에 얼마나 불러올지 (batch_size) + shuffle

#3.1.2 특성 추출

📌 특성 추출 예시

② 이미지와 레이블 정보 출력

```
import numpy as np
samples, labels = iter(train_loader).next()
classes = {0:'cat', 1:'dog'}
fig = plt.figure(figsize=(16,24))
for i in range(24):
    a = fig.add_subplot(4,6,i+1)
    a.set_title(classes[labels[i].item()])
    a.axis('off')
    a.imshow(np.transpose(samples[i].numpy(), (1,2,0)))
plt.subplots_adjust(bottom=0.2, top=0.6, hspace=0)
```



- Iter()와 next() -> train_loader에서 데이터를 하나씩 꺼냄
- np.transpose()
  - 샘플의 [32, 3, 224, 224] 형태를 (224, 224, 3) 형태로 변환

#3.1.2 특성 추출

📌 특성 추출 예시

③ 사전 훈련된 ResNet18 모델 내려받기

- Pretrained=True : 사전 학습된 가중치 사용

```
resnet18 = models.resnet18(pretrained=True)
```

```
C:\Users\hakyu\anaconda3\lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and will be removed in 0.15, please use 'weights' instead.
  warnings.warn(
C:\Users\hakyu\anaconda3\lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and will be removed in 0.15. The current behavior is equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to C:\Users\hakyu/.cache\torch\hub\checkpoints\resnet18-f37072fd.pth

100% 44.7M/44.7M [00:00<00:00, 54.7MB/s]
```

```
def set_parameter_requires_grad(model, feature_extracting=True):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False

set_parameter_requires_grad(resnet18)
```

- FC layer 추가

```
resnet18.fc = nn.Linear(512, 2)
```

- 파라미터 확인

```
for name, param in resnet18.named_parameters():
    if param.requires_grad:
        print(name, param.data)
```

```
fc.weight tensor([[ 0.0386,  0.0267,  0.0129,  ...,  0.0141, -0.0286, -0.0161],
        [-0.0268, -0.0362,  0.0220,  ...,  0.0242,  0.0145, -0.0015]])
fc.bias tensor([ 0.0085, -0.0200])
```

#3.1.2 특성 추출

📌 특성 추출 예시

④ 훈련

- 모델 객체 생성 후 손실 함수 정의

```
model = models.resnet18(pretrained = True)

for param in model.parameters():
    param.requires_grad = False

model.fc = torch.nn.Linear(512, 2)
for param in model.fc.parameters():
    param.requires_grad = True

optimizer = torch.optim.Adam(model.fc.parameters())
cost = torch.nn.CrossEntropyLoss()
print(model)
```

```
(layer4): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=512, out_features=2, bias=True)
```

#3.1.2 특성 추출

📌 특성 추출 예시

④ 훈련

- 훈련 함수 생성

```
def train_model(model, dataloaders, criterion, optimizer, device, num_epochs=13, is_train=True):
    since = time.time()
    acc_history = []
    loss_history = []
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in dataloaders:
            inputs = inputs.to(device)
            labels = labels.to(device)

            model.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / len(dataloaders.dataset)
        epoch_acc = running_corrects.double() / len(dataloaders.dataset)

        print('Loss: {:.4f} Acc: {:.4f}'.format(epoch_loss, epoch_acc))

        if epoch_acc > best_acc:
            best_acc = epoch_acc

        acc_history.append(epoch_acc.item())
        loss_history.append(epoch_loss)
        torch.save(model.state_dict(), os.path.join('ML_5장/train/', '{0:0=2d}.pth'.format(epoch)))
        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
    print('Best Acc: {:4f}'.format(best_acc))
    return acc_history, loss_history
```

- FC layer은 학습하도록 설정

```
params_to_update = []
for name,param in resnet18.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)
        print("\t",name)

optimizer = optim.Adam(params_to_update)
```

```
        fc.weight
        fc.bias
```

EWHA EURON

#3.1.2 특성 추출

📌 특성 추출 예시

④ 훈련

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
criterion = nn.CrossEntropyLoss()
train_acc_hist, train_loss_hist = train_model(resnet18, train_loader, criterion, optimizer, device)
```

```
Epoch 6/12
----------
Loss: 0.2349 Acc: 0.9039

Epoch 7/12
----------
Loss: 0.2224 Acc: 0.8987

Epoch 8/12
----------
Loss: 0.2269 Acc: 0.9091

Epoch 9/12
----------
Loss: 0.2452 Acc: 0.8857

Epoch 10/12
----------
Loss: 0.2662 Acc: 0.8779

Epoch 11/12
----------
Loss: 0.2035 Acc: 0.9091

Epoch 12/12
----------
Loss: 0.2130 Acc: 0.9143

Training complete in 3m 47s
Best Acc: 0.919481
```

⑤ 평가

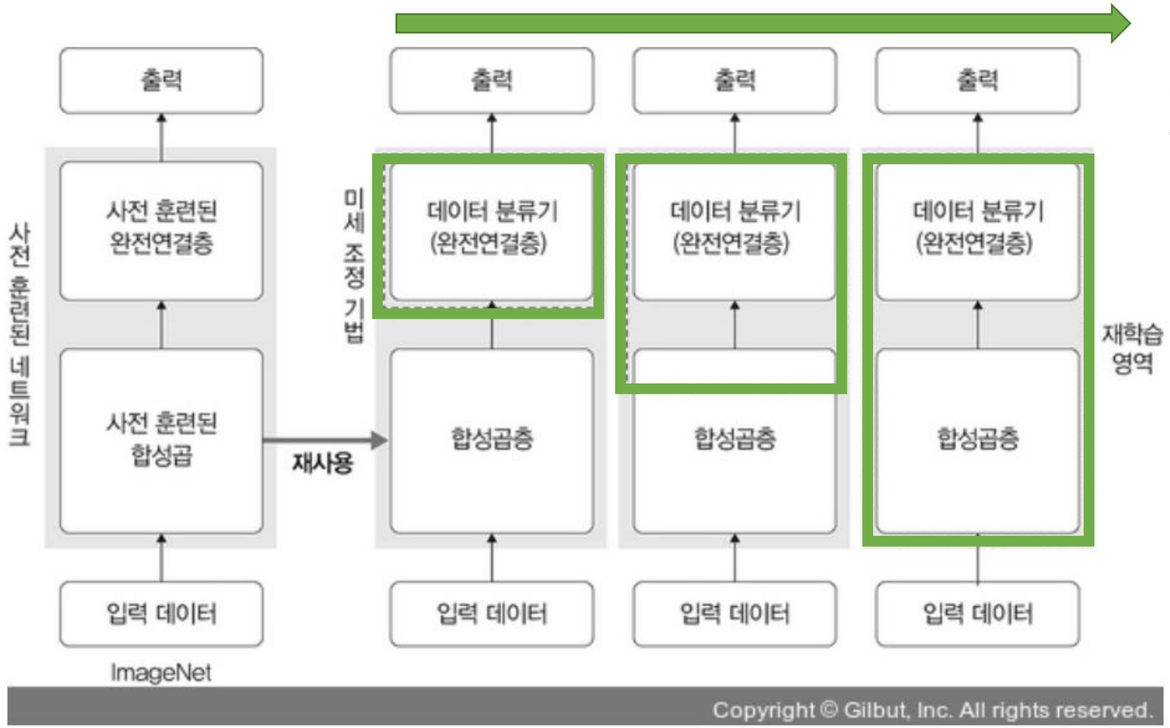
(코드 생략)

```
Loading model catanddog/06.pth
Acc: 0.9388

Loading model catanddog/07.pth
Acc: 0.9184

Loading model catanddog/08.pth
Acc: 0.9286

Loading model catanddog/09.pth
Acc: 0.9184

Loading model catanddog/10.pth
Acc: 0.9286

Loading model catanddog/11.pth
Acc: 0.9184

Loading model catanddog/12.pth
Acc: 0.9184

Validation complete in 0m 8s
Best Acc: 0.938776
```

#3.1.3 미세 조정 기법

📌 미세 조정 (fine-tuning) 기법

- 사전 훈련된 모델을 목적에 맞게 재학습시키거나 학습된 가중치의 일부를 재학습시키는 방법
- 사전 훈련된 데이터셋의 특성과 학습시키려는 데이터셋의 특성이 다를 경우 가중치를 업데이트해 특성을 다시 추출
- 데이터셋의 크기와 유사성에 따라 재훈련되는 layer 범위가 달라짐



▲ 그림 5-44 미세 조정 기법

4. 설명 가능한 CNN

EWHA EURON

#4.1.1 설명 가능한 CNN

📌 미세 조정 (fine-tuning) 기법

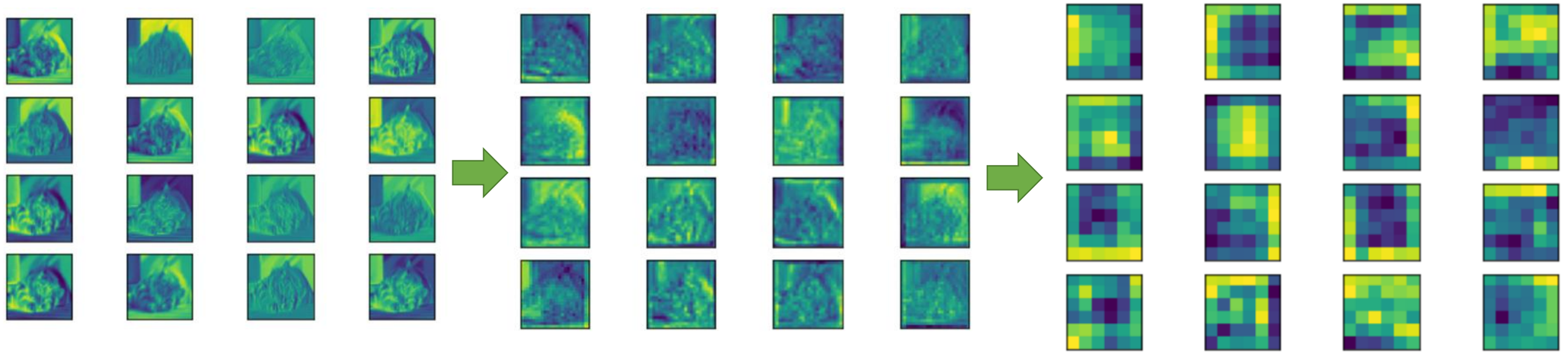
- 딥러닝 처리 결과를 사람이 이해할 수 있는 방식으로 제시하는 기술
- CNN을 구성하는 각 중간 계층부터 최종 분류까지 입력된 이미지에서 특성이 어떻게 추출되고 학습하는지를 시각적으로 설명
- 종류
  - 필터에 대한 시각화
  - 특성 맵에 대한 시각화

📌 특성 맵에 대한 시각화

- 특성 맵에서 입력 특성을 감지하는 방법을 이해할 수 있도록 돕는 것

#4.1.1 설명 가능한 CNN

📌 특성 맵에 대한 시각화



- 학습이 진행될수록 원래 이미지에 대한 형태를 전혀 찾아볼 수 없음
- 이미지 특징들만 전달
- But! 출력층과 가까울수록 high-level feature을 학습

- Nested Conv-layers

  - high-level feature : 모델 class와 가장 직접적인 피처들을 학습 (가장 중요)
  - mid-level feature : high-level feature을 구분하는데 중요한 피처들을 학습
  - low-level feature : mid-level feature을 구분하는데 필요한 피처들을 학습

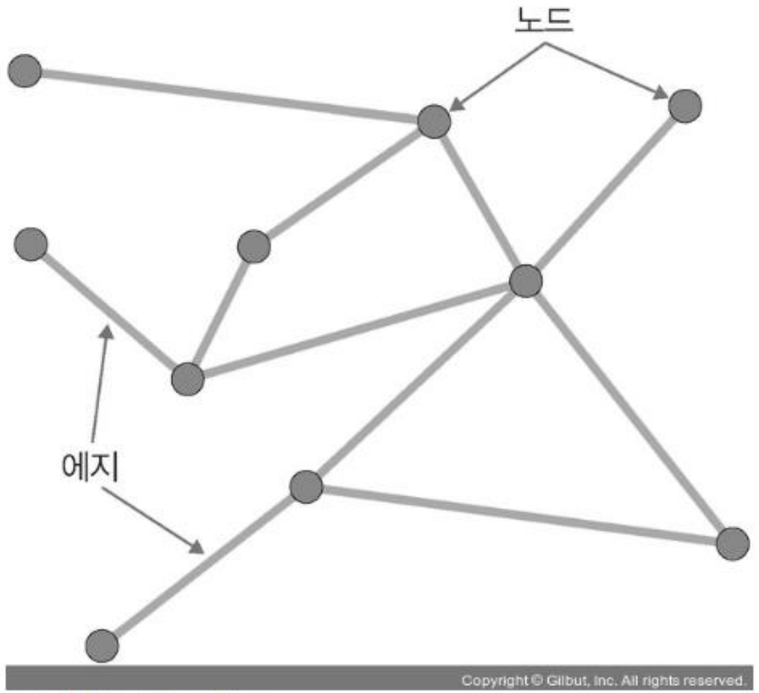
5. 그래프 합성곱 네트워크

EWHA EURON

#5.1.1. 그래프 합성곱 네트워크

📌 그래프

- 방향성이 있거나 없는 edge로 연결된 node들의 집합
- 그래프의 구성 요소
  - Edge : 두 노드를 연결한 선 (결합 방법을 의미)
  - Node : 원소



▲ 그림 5-50 그래프

📌 그래프 신경망

- 방향성이 있거나 없는 edge로 연결된 node들의 집합
- 그래프의 구성 요소
  - Edge : 두 노드를 연결한 선 (결합 방법을 의미)
  - Node : 원소

#5.1.1. 그래프 합성곱 네트워크

📌 그래프 신경망

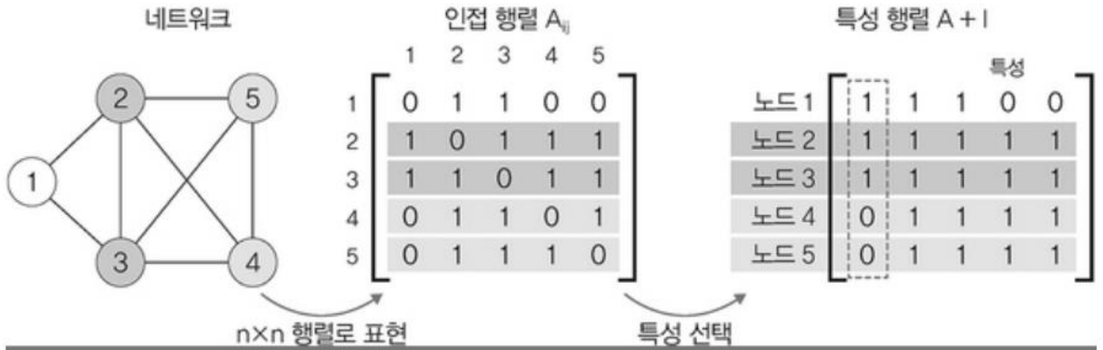
- 그래프 구조에서 사용하는 신경망
- 그래프 데이터에 대한 표현

① 인접 행렬 (adjacent matrix)

- 노드 n개를 n x n 행렬로 표현
- edge로 연결되어 있다면 1, 아니라면 0

② 특성 행렬 (feature matrix)

- 인접 행렬만으로 특성을 파악하기 어려울 때 사용
- 행 : 노드를 의미
- 열 : 데이터 특성 -> 해당 노드가 갖고 있다면 1, 아니라면 0



▲ 그림 5-51 특성 행렬[5]

#5.1.1. 그래프 합성곱 네트워크

📌 그래프 합성곱 네트워크 (Graph Convolutional Network, GCN)

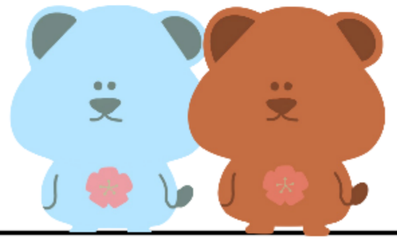
- 이미지에 대한 합성곱을 그래프로 확장한 알고리즘
- 구조

▲ 그림 5-52 그래프 합성곱 네트워크

- 리드아웃 (readout)
  - 특성 행렬을 하나의 벡터로 변환하는 함수
  - 전체 노드의 특성 벡터에 대해 평균을 구한 후 그래프 전체를 표현

- 그래프 합성곱층 (graph convolutional layer)
  - 그래프 형태의 데이터를 행렬 형태의 데이터로 변환

THANK YOU



EWHA EURON