Week3예습과제

WEEK	3주차

5장 | 합성곱 신경망 I

5.1 합성곱 신경망

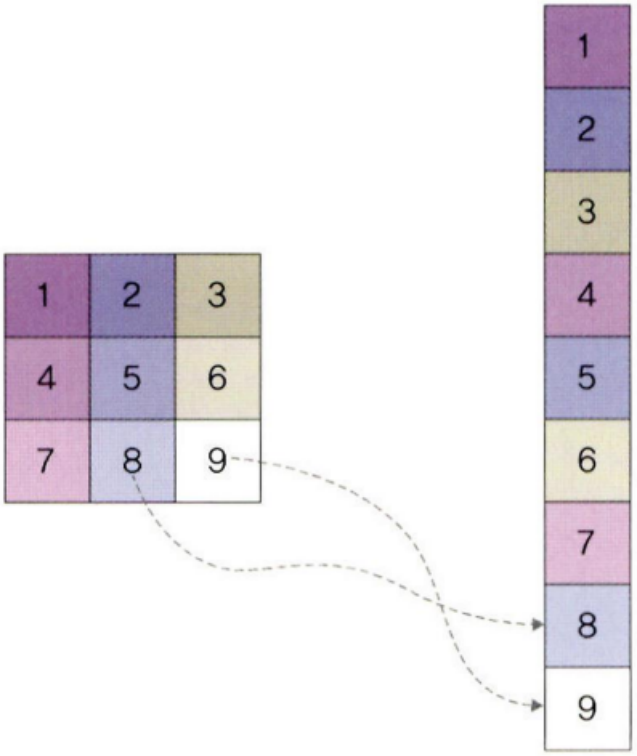
역전파 계산 과정 - 복잡 & 많은 자원 (CPU, GPU, 메모리) 요구

▶ 해결: 합성곱 신경망

합성곱 신경망: 이미지 전체를 한 번에 계산하는 것이 아닌 이미지의 국소적 부분을 계산함으로써 시간과 자원을 절약하여 이미지의 세밀한 부분까지 분석할 수 있는 신경망

5.1.1 합성곱층의 필요성

합성곱 신경망은 이미지나 영상 처리에 유용함



합성곱층 원리

이미지 분석 시 nxn 배열을 flattening해서 각 픽셀에 가중치를 곱하여 은닉층으로 전달

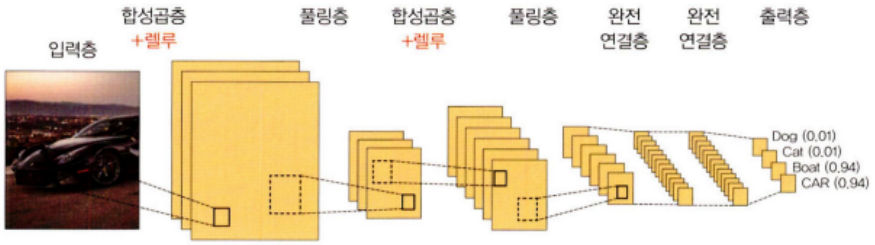
△ 이미지를 펼쳐서 분석하면 데이터의 공간적 구조를 무시하게 됨

이를 방지하고자 **합성곱층** 도입

5.1.2 합성곱 신경망 구조

합성곱 신경망(Convolutional Neural Network, CNN 또는 ConvNet)

- 음성 인식이나 이미지/영상 인식에서 주로 사용되는 신경망
- 다차원 배열 데이터를 처리하도록 구성됨 → 컬러이미지 같은 다차원 배열 처리에 특화
- 계층 5개 구성
    1. 입력층
    2. 합성곱층
    3. 풀링층
    4. 완전연결층
    5. 출력층



합성곱 신경망 구조

합성곱층과 폴링층을 거치면서 입력 이미지의 **주요 특성 벡터(feature vector)추출**

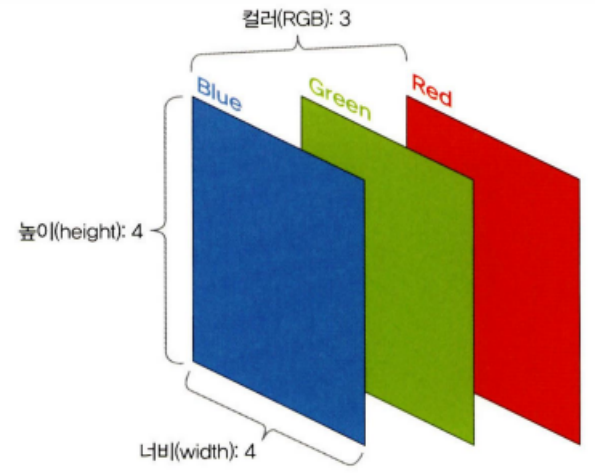
추출된 주요 특성 벡터들은 완전 연결층을 거치며 **1차원 벡터로 변환**

출력층에서 활성화 함수인 소프트맥스(softmax) 함수를 사용하여 최종 결과 출력

입력층(input layer)

이미지 데이터가 최초로 거치게 되는 계층

이미지는 높이,너비,채널의 값을 갖는 3차원 데이터 - 이 때 채널은 이미지가 그레이스케일(gray scale)이면 1, 컬러(RGB)면 3 값을 가짐



채널

위의 그림은 높이 5, 너비 5, 채널은 RGB를 가지므로 이미지 형태(shape)는 (4,4,3)으로 표현할 수 있음

합성곱층(convolutional layer)

입력 데이터에서 특성을 추출하는 역할 수행

입력 이미지가 들어왔을 때 이미지에 대한 특성을 감지하기 위해 **커널(kernel)**이나 **필터** 사용

커널/필터는 이미지의 모든 영역을 훑으면서 특성 추출 - 결과물: **특성 맵(feature map)**

커널은 **3x3**, **5×5** 크기로 적용되는 것이 일반적

스트라이드(stride)라는 지정 간격에 따라 순차적으로 이동

특성 추출 과정 (gray scale, stride=1)

1. 입력 이미지에 3x3 필터 적용

   입력 이미지와 필터를 포개 놓고 대응되는 숫자끼리 곱한 후 모두 더함

$$(1\times1)+(0\times0)+(0\times1)+(0\times0)+(1\times1)+(0\times0)+(0\times1)+(0\times0)+(1\times1)=3$$

입력 이미지에 3x3 필터 적용

2. 필터가 1만큼 이동 (stride)

$$(0\times1)+(0\times0)+(0\times1)+(1\times0)+(0\times1)+(0\times0)+(0\times1)+(1\times0)+(1\times1)=1$$

입력 이미지에 필터가 1만큼 이동

3. 필터가 1만큼 두 번째 이동

$$(0\times1)+(0\times0)+(0\times1)+(0\times0)+(0\times1)+(1\times0)+(1\times1)+(1\times0)+(0\times1)=1$$

입력 이미지에 필터가 1만큼 두 번째 이동

4. 필터가 1만큼 세 번째 이동

$$(0\times1)+(0\times0)+(1\times1)+(0\times0)+(1\times1)+(0\times0)+(1\times1)+(0\times0)+(0\times1)=3$$

입력 이미지에 필터가 1만큼 세 번째 이동

5. 필터가 1만큼 네 번째 이동

$$(0\times1)+(1\times0)+(0\times1)+(0\times0)+(0\times1)+(1\times0)+(1\times1)+(0\times0)+(0\times1)=1$$

입력 이미지에 필터가 1만큼 네 번째 이동

6. 필터가 1만큼 마지막으로 이동

$$(0\times1)+(1\times0)+(0\times1)+(0\times0)+(1\times1)+(0\times0)+(0\times1)+(1\times0)+(0\times1)=1$$

입력 이미지에 필터가 1만큼 마지막으로 이동

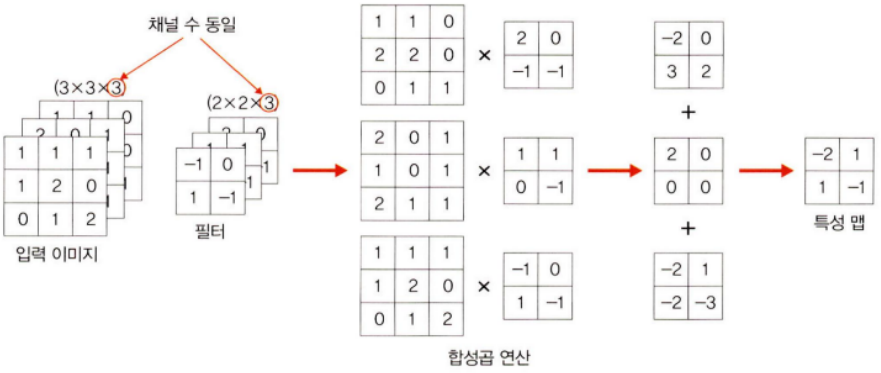
커널은 스트라이드 간격만큼 순회하면서 모든 입력 값과 합성곱 연산으로 새로운 특성 맵 만듦

위의 예시와 같이 커널과 스트라이드의 상호 작용으로 원본 (6,6,1) 크기가 (4,4,1) 크기의 특성 맵으로 줄어듦

RGB 이미지의 합성곱 (stride = 1)

★ grayscale 이미지와 구분되는 특징

1. 필터 채널 = 3 (필터 개수는 1개)
2. RGB 각각에 서로 다른 가중치로 합성곱을 적용한 후 결과를 더해 줌

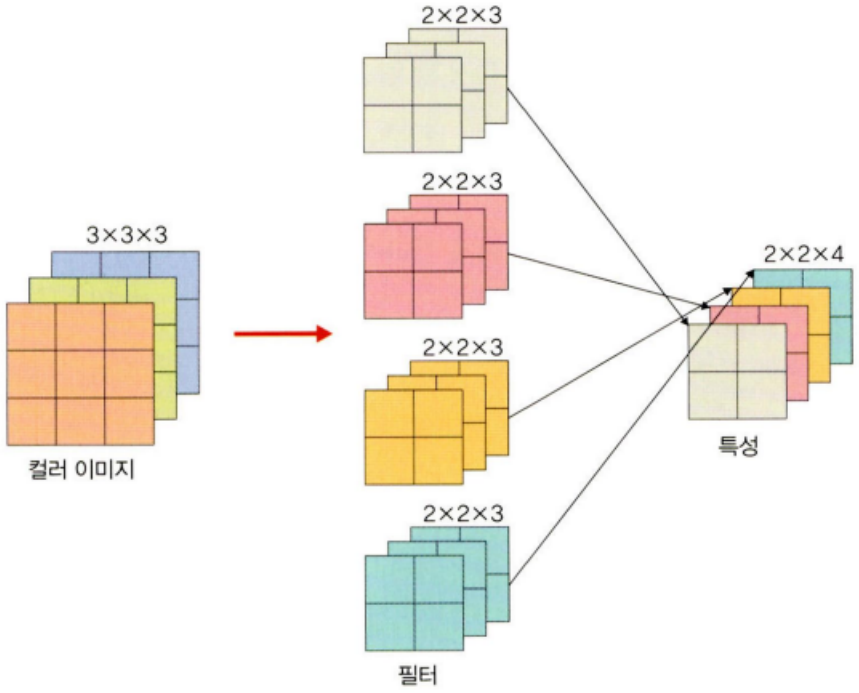


컬러 이미지 합성곱

Q. **필터**가 **두 개 이상**이라면?

필터 각각이 특성 추출 결과의 채널이 됨

각 계산은 기존 방법과 동일



필터가 2 이상인 합성곱

합성곱층 요약

- **입력 데이터**

  $W_1 \times H_1 \times D_1$

  W_1: 가로, $\times H_1$: 세로, $\times D_1$: 채널 또는 깊이

- **하이퍼파라미터**
  - 필터 개수: K
  - 필터 크기: F
  - 스트라이드: S
  - 패딩: P
- **출력 데이터**
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$

풀링층(pooling layer)

합성곱층과 유사하게 특성 맵의 차원을 다운 샘플링(이미지 축소)하여 연산량을 감소시키고, 주요 특성 벡터를 추출하여 학습을 효과적으로 할 수 있게 함

두 가지 풀링 연산

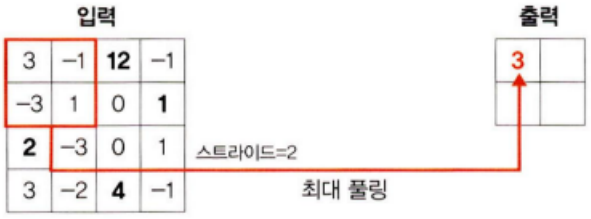
1. **최대 풀링(**max pooling): 대상 영역에서 최댓값을 추출
2. **평균 풀링**(average pooling): 대상 영역에서 평균을 반환

대부분의 합성곱 신경망 - **최대 풀링** 사용

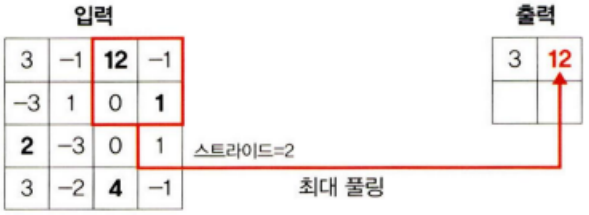
(∵ 평균 풀링은 각 커널 값을 평균화 시켜 중요 가중치 갖는 값의 **특성 희미**해질 수 있음)

최대 풀링 연산 과정

1. 3,-1,-3,1 값 중에서 최댓값(3) 선택



2. 12,-1,0,1 값 중에서 최댓값(12) 선택

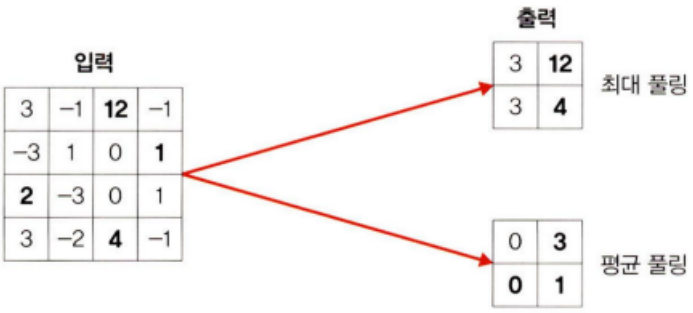


3. 2,-3,3,-2 값 중에서 최댓값(3) 선택

4. 0,1,4,-2 값 중에서 최댓값(4) 선택

평균 풀링 계산 과정은 최대 풀링과 유사한 방식으로 진행하되 각 필터의 평균으로 계산함

$$0=(3+(-1)+(-3)+1)/4$$
$$3=(12+(-1)+0+1)/4$$
$$0=(2+(-3)+3+(-2))/4$$
$$1=(0+1+4+(-1))/4$$



최대 풀링과 평균 풀링 비교

최대 풀링과 평균 풀링 요약 (계산 과정은 다르지만 사용하는 파라미터 동일)

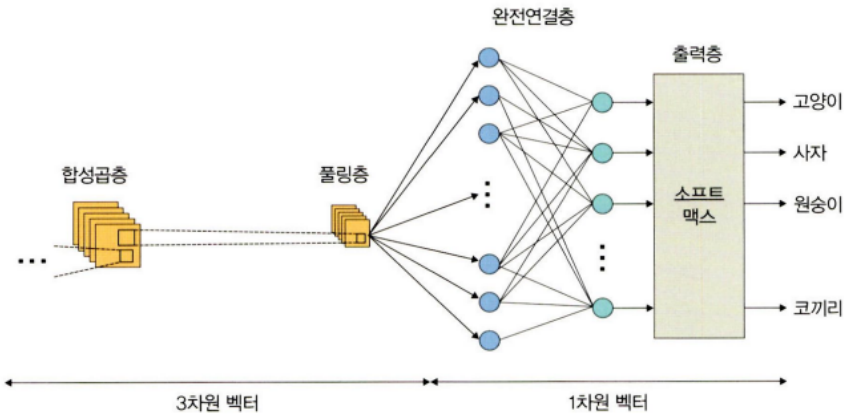
- **입력 데이터**

  $W_1 \times H_1 \times D_1$
- **하이퍼파라미터**
  - 필터 크기: F
  - 스트라이드: S
- **출력 데이터**
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$

완전연결층 (fully-connected layer)

합성곱층과 풀링층을 거치며 차원 축소된 특성 맵이 최종적으로 전달되는 층

이 과정에서 이미지는 3차원 벡터에서 1차원 벡터로 펼쳐지게(flatten) 됨



완전연결층

출력층(output layer)

소프트맥스 활성화 함수 사용됨: 입력 받은 값을 0~1 사이의 값으로 출력

▶ 이미지가 각 레이블에 속할 확률 값이 출력 됨

▶ 가장 높은 확률 값을 갖는 레이블이 최종 값으로 선정됨

5.1.3 1D, 2D, 3D 합성곱

합성곱 이동 방향의 수와 출력 형태에 따라 1D, 2D, 3D로 분류

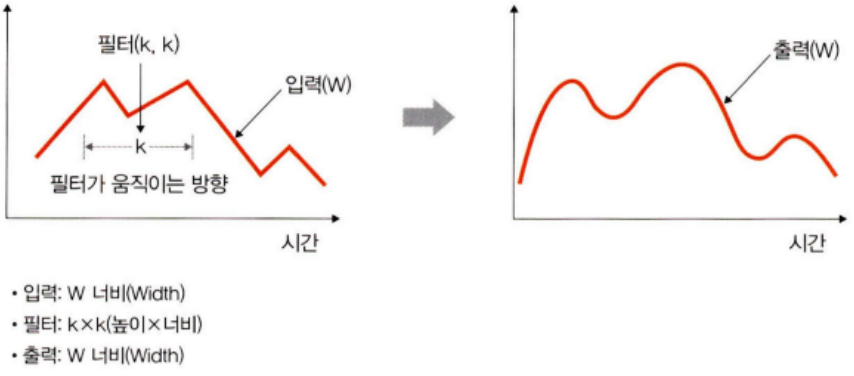
1D 합성곱

필터가 시간을 축으로 좌우로만 이동할 수 있는 합성곱

입력(W)과 필터(K)에 대한 출력은 W가 됨 e.g.) 입력 = [1,1,1,1,1], 필터 = [0.25, 0.25, 0.25], 출력 = [1,1,1]

출력 형태는 1D 배열

그래프 곡선 완화할 때 사용 多

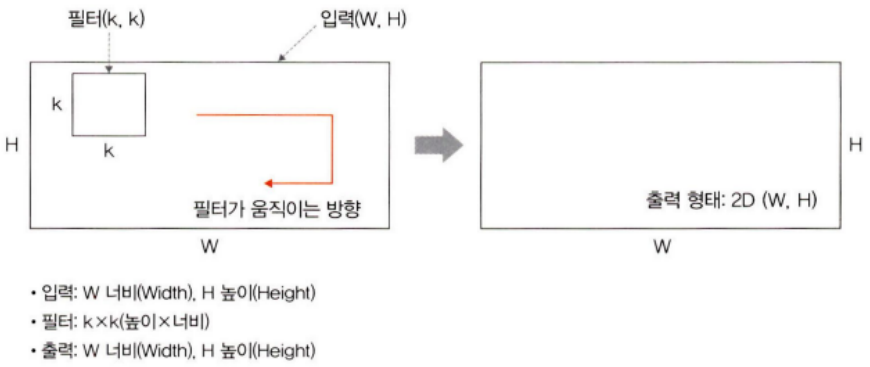


2D 합성곱

필터가 방향 두 개(좌우, 상하)로 움직이는 형태

입력(W,H)과 필터(k,k)에 대한 출력은 (W,H)가 됨

출력 형태는 2D 행렬



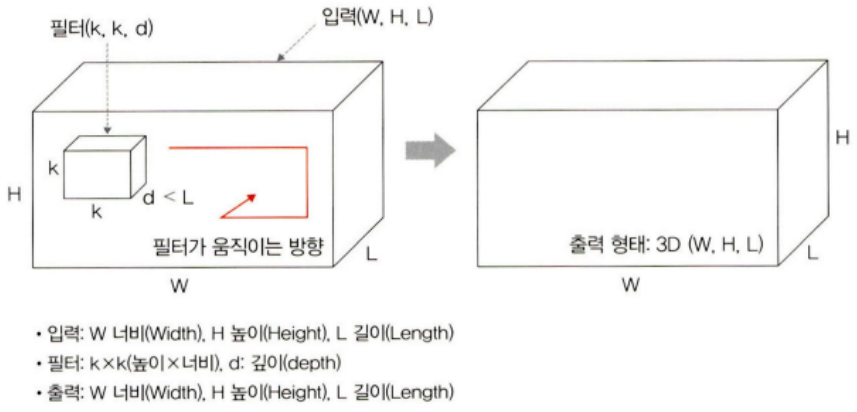
3D 합성곱

필터가 방향 세 개(좌우,상하,전후)로 움직임

입력(W,H,L)에 대해 필터(k,k,d)를 적용하면 출력으로 (W,H,L)을 가짐

출력 형태는 3D

d<L을 유지하는 것이 중요



3D 입력을 갖는 2D 합성곱

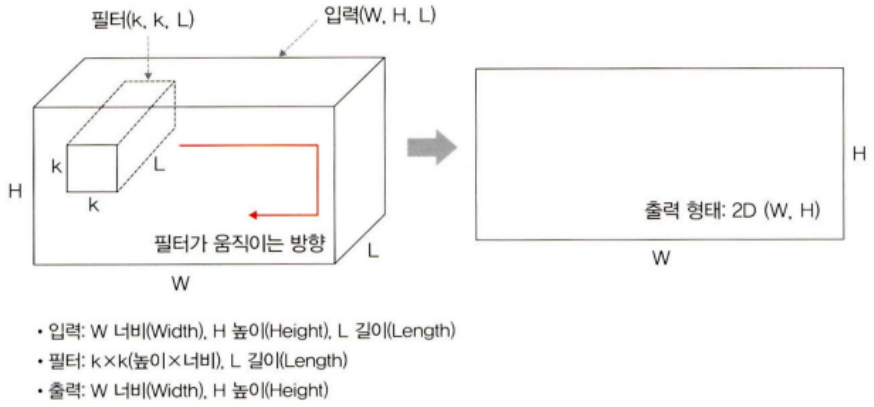
필터에 대한 길이(L)과 입력 채널의 길이(L)이 같아야 해서 만들어지는 합성곱 형태

입력(W,H,L)에 필터(k,k,L)을 적용하면 출력은 (W,H)가 됨

필터는 두 개 방향(상하,좌우)으로 움직임

출력 형태는 2D 행렬

대표적 사례: LeNet-5, VGG



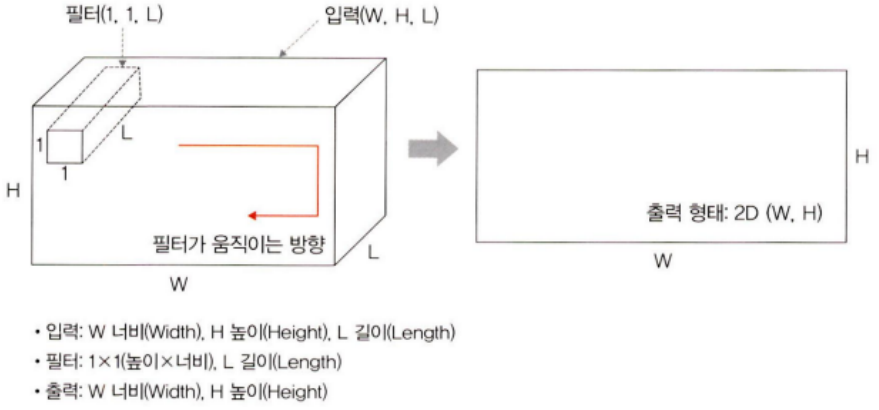
1×1 합성곱

3D 형태 입력

입력(W,H,L)에 필터(1,1,L)를 적용하면 출력은 (W,H)가 됨

1×1 합성곱에서 채널 수를 조정해서 연산량이 감소되는 효과

대표적 사례: GoogLeNet



5.2 합성곱 신경망 맛보기

코드 5-3

```
# fashion_mnist 데이터셋 내려받기
train_dataset = torchvision.datasets.FashionMNIST("/content/drive/MyDrive/Colab Notebooks/EURON/Week3/data",download=True,transform=transforms.Compose([transforms.ToTensor()]))
test_dataset = torchvision.datasets.FashionMNIST("/content/drive/MyDrive/Colab Notebooks/EURON/Week3/data",download=True,train = False,transform=transforms.Compose([transforms.ToTensor()]))
```

- `torchvision.datasets`
  - `torch.utils.data.Dataset` 의 하위 클래스

- 다양한 데이터셋 (CIFAR, MNIST, ImageNet 등)을 포함
- 주요 파라미터
  - 첫 번째 파라미터: FashionMNIST를 내려받을 위치를 정함
  - `download`: download를 True로 변경해 주면 첫 번째 파라미터의 위치에 해당 데이터셋이 있는지 확인한 후 내려받음
  - `transform`: 이미지를 텐서(0~1)로 변경

코드 5-4

```
# 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)
```

- `torch.utils.data.DataLoader()`
  - 이를 사용하여 원하는 크기의 배치 단위로 데이터를 불러오거나 순서가 무작위로 섞이도록(shuffle) 할 수 있음
  - 파라미터
    - 첫 번째 파라미터: 데이터를 불러올 데이터셋을 지정
    - `batch_size`: 데이터를 배치로 묶어 줄 때 단위 개수 지정

코드 5-5

```
# 분류에 사용된 클래스 정의

labels_map = {0: 'T-Shirt', 1: 'Trouser', 2: 'Pullover', 3: 'Dress', 4: 'Co
at', 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()
```

- `np.random`
  - 무작위 데이터 생성을 위해 사용

    - `np.random.randint()`
        - 이산형 분포를 갖는 데이터에서 무작위 표본을 추출할 때 사용

- `train_dataset [img_xy][0][0, :, :]`
    - `train_dataset`을 이용한 3차원 배열 생성
    - 의미: `train_dataset`에서 [img_xy][0][0, :, :]에 해당하는 요소 값을 가져옴

코드 5-6

```
# 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
```

- 클래스(class) 형태의 모델은 항상 `torch.nn.Module`을 상속 받음
- `__init__()`
    - 객체가 갖는 속성 값을 초기화 하는 역할
    - 객체 생성 시 자동 호출

➕ **객체(object)**

메모리를 할당 받아 프로그램에서 사용되는 모든 데이터 (e.g. 변수, 함수 등)

재사용성을 이유로 객체 지향 프로그래밍 사용 多

➕ **함수(function)**

하나의 특정 작업을 수행하기 위해 독립적으로 설계된 프로그램 코드

함수의 결괏값을 계속 사용하기 위해서는 따로 값을 저장해야만 함

함수를 포함한 프로그램 코드의 일부를 재사용 하는 모든 경우에 마찬가지

▶ **클래스(class)**: 함수뿐만 아니라 관련된 변수까지도 한꺼번에 묶어서 관리하고 재사용 할 수 있게 해줌

- `nn`
  - 딥러닝 모델(네트워크) 구성에 필요한 모듈이 모여 있는 패키지
  - `Linear`
    - 단순 선형 회귀 모델 만들 때 사용
    - 파라미터
      - `in_features`: 입력의 크기
      - `out_features`: 출력의 크기
    - 실제로 데이터 연산이 진행되는 `forward()` 부분에는 첫 번째 파라미터 값만 넘겨주게 되고, 두 번째 파라미터에서 정의된 크기가 `forward()` 연산의 결과가 됨
- `torch.nn.Dropout(p)`
  - p만큼의 비율로 텐서의 값이 0이 됨
  - 0이 되지 않은 값들은 기존 값에 (1/(1-p))만큼 곱해져 커짐
- `forward()`
  - 모델이 학습 데이터를 입력 받아서 순전파 연산을 진행하는 함수
    - 순전파 연산: H(x) 식에 입력 x로부터 예측된 y를 얻는 것
  - 반드시 forward라는 이름의 함수여야 함
  - 객체를 데이터와 함께 호출하면 자동으로 실행됨
- `view()`
  - 텐서의 크기(shape)를 변경해 주는 역할 (넘파이의 `reshape`과 같은 역할)
- **활성화 함수 지정 방법**
  - `F.relu()`
    - `forward()` 함수에서 정의
  - `nn.ReLU()`
    - `__init__()` 함수에서 정의
  - 차이
    - nn.Conv2d에서는 input_channel과 output_channel을 사용해서 연산
    - F.conv2d에서는 입력과 가중치 자체를 직접 넣어줌 (= 가중치를 전달해야 할 때마다 가중치 값을 새로 정의해야 함)

구분	nn.xx	nn.functional.xx
형태	nn.Conv2d: 클래스 nn.Module: 클래스를 상속받아 사용	nn.functional.conv2d: 함수 def function (input) 으로 정의된 순수한 함수
호출 방법	먼저 하이퍼파라미터를 전달한 후 함수 호출을 통해 데이터 전	함수를 호출할 때 하이퍼파라미터, 데이터 전달
위치	nn.Sequential 내에 위치	nn.Sequential에 위치할 수 없음
파라미터	파라미터를 새로 정의할 필요 없음	가중치를 수동으로 전달해야 할 때마다 자체 가중치를 정의

코드 5-8

```
# 심층 신경망을 이용한 모델 학습

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))
    labels = Variable(labels)

    outputs = model(train) # 학습 데이터를 모델에 적용
    loss = criterion(outputs, labels)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    count += 1

    if not (count % 50):
      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,a
```

- **Autograd**
  - 자동 미분을 수행하는 파이토치 핵심 패키지
  - 자동 미분에 대한 값을 저장하기 위해 테이프(tape)를 사용
  - `Variable`을 사용해서 역전파를 위한 미분 값을 자동으로 계산해 줌

코드 5-9

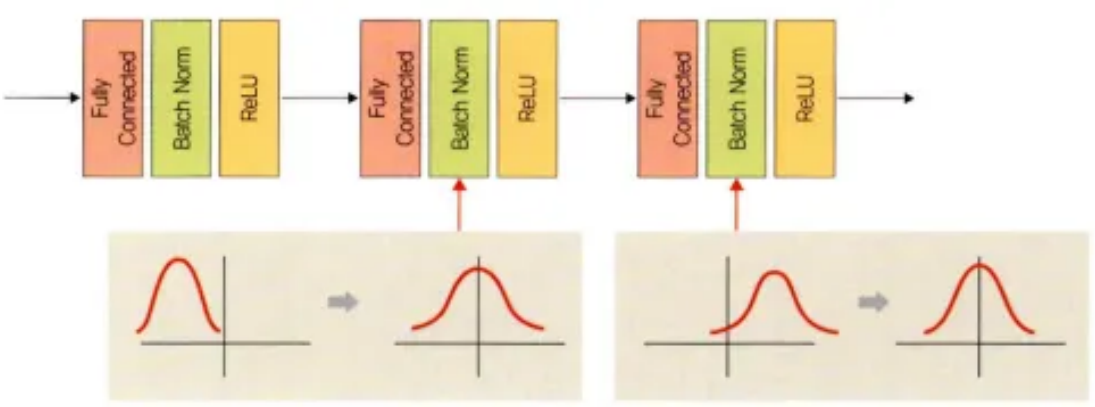
```
# 합성곱 네트워크 생성

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.Dropout2d(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`
  - `__init__()`에서 사용할 네트워크 모델들을 정의해 줌
  - `forward()` 함수에서 구현될 순전파를 계층 형태로 좀 더 가독성이 뛰어난 코드로 작성할 수 있음
  - ∴ 여러 개의 계층을 하나의 컨테이너에 구현하는 방법
- **합성곱층(conv layer)**
  - 합성곱 연산을 통해 이미지의 특징 추출
  - 커널(또는 필터)이라는 n x m 크기의 행렬이 높이(height) x 너비 (width) 크기의 이미지를 처음부터 끝까지 훑으면서 각 원소 값끼리 곱한 후 모두 더한 값을 출력
  - 커널은 일반적으로 3x3이나 5x5를 사용
  - 파라미터
    - `in_channels`
      - 입력 채널(깊이)의 수 - 흑백 이미지 = 1, RGB 이미지 = 3
    - `out_channels`
      - 출력 채널의 수
    - `kernel_size`
      - 커널의 크기 (정사각형은 `kernel_size = 3` (3×3), 직사각형은 `kernel_size = (3,5)` (3×5))
      - 커널
        - 이미지 특징을 찾아내기 위한 공용 파라미터 (CNN에서 학습 대상은 필터 파라미터)
        - 입력 데이터를 스트라이드 간격으로 순회하면서 합성곱 계산
    - `padding`
      - 패딩(출력 크기 조정을 위해 입력 데이터 주위에 0 채우는 것) 크기
      - 패딩 값 ↑ ... 출력 크기 ↑
- `BatchNorm2d`
  - 학습 과정에서 각 배치 단위별로 데이터가 다양한 분포를 가지더라도 **평균과 분산을 이용하여 정규화**하는 것
  - **평균 0, 표준편차 1**로 데이터의 분포 조정



BatchNorm2d

- `MaxPool2d`
  - 이미지 크기 축소 용도
  - 풀링 계층
    - 합성곱층의 출력 데이터를 입력으로 받아서 출력 데이터(activation map)의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용
    - 처리 방법
      1. 최대 풀링(max pooling)
      2. 평균 풀링(average pooling)
      3. 최소 풀링(min pooling)
  - 파라미터
    - kernel_size = m x n 행렬로 구성된 가중치
    - stride
      - 입력 데이터에 커널(필터)을 적용할 때 이동할 간격 의미
      - 값 ↑ ... 출력 크기 ↓
- **클래스 분류를 위한 완전연결층 전달 과정**
  - 클래스를 분류하기 위해서는 이미지 형태의 데이터를 배열 형태로 변환하여 작업해야 함
  - Conv2d에서 사용하는 하이퍼파라미터(패딩, 스트라이드) 값들에 따라 출력 크기가 달라짐
  - 줄어든 출력 크기를 최종적으로 분류를 담당하는 완전연결층(fully connected layer)으로 전달
  - 파라미터
    - `in_features`
      - 입력 데이터의 크기
      - 이전까지 수행한 Conv2d, MaxPool2d는 이미지 데이터를 입력으로 받아 처리했지만 출력 결과를 완전연결층으로 보내려면 1차원으로 변경해 주어야 함
      - 1차원 변경 공식
        - Conv2d 계층에서의 출력 크기 구하는 공식

$$출력\ 크기 = (W - F + 2P)/S + 1$$

- W: 입력 데이터의 크기(input_volume_size)
- F: 커널 크기(kernel_size)
- P: 패딩 크기(padding_size)
- S: 스트라이드(strides)

    - MaxPool2d 계층에서의 출력 크기 구하는 공식

$$출력\ 크기 = IF/F$$

- IF: 입력 필터의 크기(input_filter_size, 또한 바로 앞의 Conv2d의 출력 크기이기도 합니다)
- F: 커널 크기(kernel_size)

- `out_features`
    - 출력 데이터의 크기

합성곱 신경망은 심층 신경망과 비교하여 **정확도 약간 높음**

심층 신경망과 별 차이가 없기 때문에 조금 더 간편한 심층 신경망만 사용해도 무난할 것 같지만,

실제로 이미지 데이터가 많아지면 **단순 심층 신경망**으로는 정확한 특성 추출 및 분류가 **불가능**

→ **합성곱 신경망을 생성할 수 있도록 학습해야 함**