chapter 7 – 합성공 신경망(CNN)

7.1 전체구조

- CNN에서는 1) 합성곱 계층 2) 풀링 계층이 새로이 등장합니다.
- 즉, Affine – ReLU 연결이 Conv – ReLU – (Pooling)으로 바뀌었다고 보면 됩니다.
- CNN에서 주목할 또 다른 점은 * 출력에 가까운 층에서는 지금까지의 'Affine – ReLU' 구성을 사용할 수 있다느 것입니다.
- * 마지막 출력 계층에서는 'Affine – Softmax' 조합을 그대로 사용합니다.

7.2 합성곱 계층

- CNN에서는 * 패딩, * 스트라이드 등 CNN 고유의 용어가 등장합니다.
- 또, 각 계층 사이에는 3차원 데이터같이 입체적인 데이터가 흐른다는 점에서 완전연결 신경망과 다릅니다.

7.2.1 완전연결 계층의 문제점

- 완전연결계층에서는 인접하는 계층의 뉴런이 모두 연결되고 출력의 수는 임의로 정할 수 있습니다.
- 완전연결 계층의 문제점은 무엇일까요? 바로 [데이터의 형상이 무시]된다는 사실입니다.
- 한편, 합성곱 계층은 형상을 유지합니다. 이미지도 3차원 데이터로 입력받으며, 마찬가지로 다음 계층에서도 3차원 데이터로 전달합니다.
- CNN에서는 합성곱 계층의 입출력 데이터를 [특징 맵]이라고도 합니다.
- 합성곱 계층의 입력 데이터를 [입력 특징 맵], 출력 데이터를 [출력 특징 맵]이라고 하는 식이죠.

7.2.2 합성곱 연산

- [합성곱 연산]은 이미지 처리에서 말하는 필터 연산에 해당됩니다.
- 문헌에 따라 필터를 [커널]이라 칭하기도 합니다.
- 합성곱 연산은 필터의 [윈도우]를 일정 간격으로 이동해가며 입력 데이터에 적용합니다.

7.2.3 패딩

- 합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값(예컨대 0)으로 채우기도 합니다. 이를 [패딩]이라고 하며, [그림 7-6]은 (4,4) 크기의 입력 데이터에 폭이 1인 패딩을 적용한 모습입니다.

7.2.4 스트라이드

- 필터를 적용하는 위치의 간격을 [스트라이드]라고 합니다.
- 스트라이드를 2로 하면 필터를 적용하는 윈도우가 두 칸씩 이동합니다.
- 이처럼 스트라이드를 키우면 출력 크기는 작아집니다.
- 딥러닝 프레임워크 중에는 값이 딱 나눠떨어지지 않을 때는 가장 가까운 정수로 반올림하는 등, 특별히 에러를 내지 않고 진행하도록 구현하는 경우도 있습니다.

7.2.5 3차원 데이터의 합성곱 연산

- 이미지만 해도 세로, 가로에 더해서 채널까지 고려한 3차우너 데이터입니다.
- 2차원일 때와 비교하면, 길이 방향(채널 방향)으로 특징 맵이 늘어났습니다.

- 3차원의 합성곱 연산에서 주의할 점은 입력 데이터의 채널 수와 필터의 채널 수가 같아야 한다는 것입니다.
- 한편, 필터 자체의 크기는 원하는 값으로 설정할 수 있습니다.(단, 모든 채널의 필터가 같은 크기여야 합니다.)

7.2.6 블록으로 생각하기

- 자, 이 예에서 출력 데이터는 한 장의 특징 맵입니다.
- 그럼 합성곱 연산의 출력으로 다수 채널을 내보내려면 어떻게 해야 할까요? 그 답은 필터(가중치)를 다수 사용하는 것입니다.

7.3 풀링 계층

- 풀링은 세로, 가로 방향의 공간을 줄이는 연산입니다.
- 참고로, 풀링의 윈도우 크기와 스트라이드는 같은 값으로 설정하는 것이 보통입니다. 예를 들어, 윈도우가 3X3이면 스트라이드를 3, 윈도우가 4X4이면 스트라이드를 4로 설정합니다.

7.3.1 풀링 계층의 특징

- 1) 학습해야 할 매개변수가 없다
- 2) 채널 수가 변하지 않는다 : 데이터의 태널 수 그대로 출력 데이터로 내보냅니다.
- 3) 입력의 변화에 영향을 적게 받는다(강건하다)
  : 입력 데이터가 조금 변해도 풀리의 결과는 잘 변하지 않습니다.

7.4 합성곱 / 풀링 계층 구현하기

7.4.1 4차원 배열

- CNN에서 계층 사이를 흐르는 데이터는 4차원입니다. 예를 들어, (10, 1, 28, 28)이라면, 이는 높이 28, 너비 28, 채널 1개인 데이터가 10개라는 이야깁니다.
- 그래서 합성곱 연산의 구현은 복잡해질 것 같지만, 다음 절에서 설명하는 im2col이라는 '트릭'이 문제를 단순하게 만들어줍니다.

7.4.2 im2col로 데이터 전개하기

- for문 대신 im2col이라는 편의 함수를 사용해 간단하게 구현해보겠습니다.
- im2col은 입력 데이터를 필터링(가중치 계산)하기 좋게 전개하는 (펼치는) 함수입니다.
- 3차원 입력 데이터에 im2col을 적용하면 2차원 행렬로 바뀝니다.
- 4차원 데이터를 2차원으로 변환하는 역할을 함!
- 구체적으로는 입력 데이터에서 필터를 적용하는 영역을 한 줄로 늘어놓습니다.
- 그래서 im2col을 사용해 구현하면 평소보다 메모리를 많이 소비하는 단점이 있습니다.
- 하지만 컴퓨터는 큰 행렬을 묶어서 계산하는 데 탁월합니다.

- im2col로 입력 데이터를 전개한 다음에는 합성곱 계층의 필터를 1열로 전개하고, 두 행렬의 내적을 계산하면 됩니다. 이는 완전연결 계층의 Affine 계층에서 한 것과 거의 같습니다.
- im2col 방식으로 출력한 결과는 2차원 행렬입니다.
- CNN은 데이터를 4차우너 배열로 저장하므로 2차원인 출력 데이터 4차원으로 변형합니다.

7.4.3 합성곱 계층 구현하기

* im2col(input_data, filter_h, filter_w, stride = 1, pad = 0)
- filter_h : 필터의 높이 / filter_w : 필터의 너비

```
import sys, os
sys.path.append(os.pardir)
from common.util import im2col

x1 = np.random.rand(1, 3, 7, 7) # (데이터 수, 채널 수, 높이, 너비)
col1 = im2col(x1, 5, 5 stride = 1, pad = 0)
print(col1.shape)

x2 = np.random.rand(10, 3, 7, 7) # 데이터 10개
col2 = im2col(x2, 5, 5, stride = 1, pad = 0)
print(col2.shape)
```

```
class Convolution:
        def __init__(self, W, b, stride = 1, pad = 0):
                self.W = W
                self.b = b
                self.stride = stride
                self.pad = pad

        def forward(self, x):
                FN, C, FH, FW = self.W.shape
                N, C, H, W = x.shape
                out_h = int(1 + (H + 2*self.pad - FH) / self.stride)
                out_w = int(1 + (W + 2*self.pad - FW) / self.stride)

                col = im2col(x, FH, FW, self.stride, self.pad)
                col_W = self.W.reshape(FN, -1).T
                out = np.dot(col, col_W) + self.b

                out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

                return out
```

- 합성곱 계층은 필터(가중치), 편향, 스트라이드, 패딩을 인수로 받아 초기화합니다
- 입력 데이터를 im2col로 전개하고, 필터도 reshape을 사용해 2차원 배열로 전개합니다.
- 그리고 이렇게 전개한 두 행렬의 내적을 구합니다
- 이때 reshape의 두 번째 인수를 -1로 지정했는데, 이는 reshape이 제공하는 편의 기능입니다. reshape에 -1을 지정하면 다차원 배열의 원소 수가 변환 후에도 똑같이 유지되도록 적절히 묶어줍니다.
- 앞의 코드에서 (10, 3, 5, 5) 형상을 한 다차원 배열 W의 원소 수는 총 750개죠? 이 배열에 reshape(10, -1)을 호출하면 750개의 원소를 10묶음으로, 즉 형상이 (10, 75)인 배열로 만듭니다.
- 이상이 합성곱 계층의 forward 구현입니다. im2col로 전개한 덕분에 완전연결 계층의 Affine 계층과 거의 똑같이 구현할 수 있습니다.
- 합성곱 계층의 역전파에서는 im2col을 역으로 처리해야 합니다. 이는 이 책이 제공하는 col2im 함수를 사용하면 됩니다.
- col2im을 사용한다는 점을 제외하면 합성곱 계층의 역전파는 Affine 계층과 똑같습니다.

7.4.4 풀링 계층 구현하기

- 풀링 계층 구현도 합성곱 계층과 마찬가지로 im2col을 사용해 입력 데이터를 전개합니다. 단, 풀링의 경우엔 채널 쪽이 독립적이라는 점에서 합성곱 계층 때와 다릅니다. 구체적으로는 [그림 7-21]과 같이 풀링 적용 영역을 채널마다 독립적으로 전개합니다.
- [풀링 계층 구현]은 1) 입력 데이터를 전개한다 2) 행별 최댓값을 구한다 3) 적절한 모양으로 성형
- 최댓값 계산에는 넘파이의 np.max 메서드를 사용할 수 있습니다. np.max는 인수로 축(axis)을 지정할 수 있는데, 이 인수로 지정한 축마다 최댓값을 구할 수 있습니다. 가령 np.max(x, axis = 1)과 같이 쓰면 입력 x의 1번째 차원의 축마다 최댓값을 구합니다.

7.5 CNN 구현하기

- 데이터 - [Conv - ReLU - Pooling] - [Affine - ReLU] - [Affine - Softmax]
- 이 CNN을 SimpleConvNet이라고 한다.

7.6.1.1

- [그림 7-24]의 오른쪽과같이 규칙성 있는 필터는 '무엇을 보고 있는'걸까요? 그것은 에지(색상이 바뀐 경계선)와 블롭(국소적으로 덩어리진 영역) 등을 보고 있습니다.
- 이처럼 합성곱 계층의 필터는 에지나 블롭 등의 원시적인 정보를 추출할 수 있습니다.

7.6.2 층 깊이에 따른 추출 정보 변화

- 앞 절의 결과는 1번째 층의 합성곱 계층을 대상으로 한 것이었습니다.
- 1번째 층의 합성곱 계층에서는 에지나 블롭 등의 저수준 정보가 추출됩니다.
- 그럼 층이 깊어지면 어떤 정보들이 추출될까요? 딥러닝 시각화에 대한 연구에 따르면, 계층이 깊어질수록 추출되는 정보(정확히는 강하게 반응하는 뉴런)는 더 복잡하고 추상화된다는 것을 알 수 있습니다. (에지 -> 텍스터 -> 더 복잡한 사물의 일부에 반응)

- 즉, 층이 깊어질수록 뉴런이 반응하는 대상이 단순한 모양에서 '고급' 정보로 변화해갑니다.
- 다시 말하면, 사물의 '의미'를 이해하도록 변화하는 것입니다.

7.7 대표적인 CNN

7.7.1 LeNet

- LeNet은 손글씨 숫자로 인식하는 네트워크입니다
- 합성곱 계층과 풀링 계층을 반복하고, 마지막으로 완전연결 계층을 거치면서 결과를 출력합니다.
- LeNet과 '현재의 CNN'의 차이는 1) 활성화 함수 - LeNet은 시그모이드 함수, 현재는 주로 ReLU를 사용. / LeNet은 서브샘플링을 하여 중간 데이터의 크기가 작아지지만 현재는 최대 풀링이 주류입니다.

7.7.2 AlexNet

- 활성화 함수로 ReLU를 이용
- LRN이라는 국소적 정규화를 실시하는 계층을 이용한다.
- 드롭아웃을 사용