


DNN_02_1. CNN

🕒 생성일	@2022년 6월 10일 오후 1:48
📁 유형	머신러닝/딥러닝
👤 작성자	 동훈 오

핸즈온: 14. 합성곱 신경망을 사용한 컴퓨터 비전

CNN 의 구성 요소, 텐서플로우를 사용한 구현, CNN 구조 몇 가지, 객체 탐지, semantic segmentation

합성곱 층(convolutional layer)

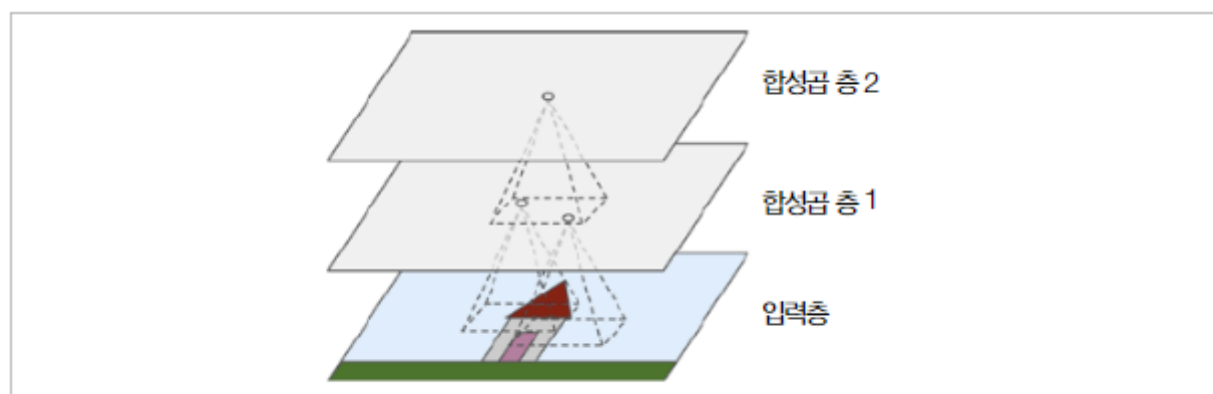
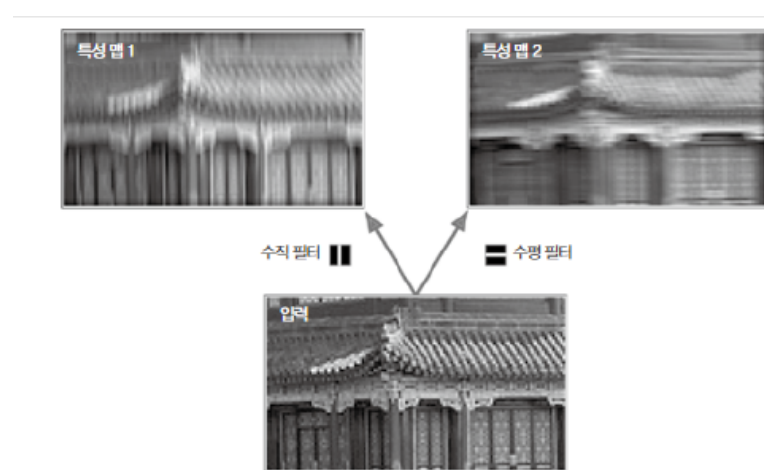


그림 14-2 사각 형태의 국부 수용장을 가진 CNN 층

위 이미지와 같은 구조는 네트워크가 첫 번째 은닉층에서는 작은 저수준 특성에 집중하고, 그다음 은닉층에서는 더 큰 고수준 특성으로 조합해 나가도록 도와준다.

필터

다음의 이미지는 필터 또는 합성곱 커널 이라 부르는 두 개의 가중치 세트를 보여준다.



층의 전체 뉴런에 적용된 하나의 필터는 하나의 특성 맵을 만든다. 이 맵은 필터를 가장 크게 활성화시키는 이미지의 영역을 강조한다. 훈련하는 동안 합성곱 층이 자동으로 해당 문제에 가장 유용한 필터를 찾고 상위층은 이들을 연결하여 더 복잡한 패턴을 학습한다.

여러 가지 특성맵 쌓기

각 특성 맵의 픽셀은 하나의 뉴런에 해당하고 하나의 특성 맵 안에서는 모든 뉴런이 같은 파라미터를 공유하지만, 다른 특성 맵에 있는 뉴런은 다른 파라미터를 사용한다. 따라서 하나의 합성곱 층이 입력에 여러 필터를 동시에 적용하여 입력에 있는 여러 특성을 감지할 수 있다.

입력 이미지는 컬러 채널마다 하나씩 여러 서브 층으로 구성되기도 한다. 컬러 채널은 전형적으로 RGB 이며, 흑백 이미지는 하나의 채널만 가진다.

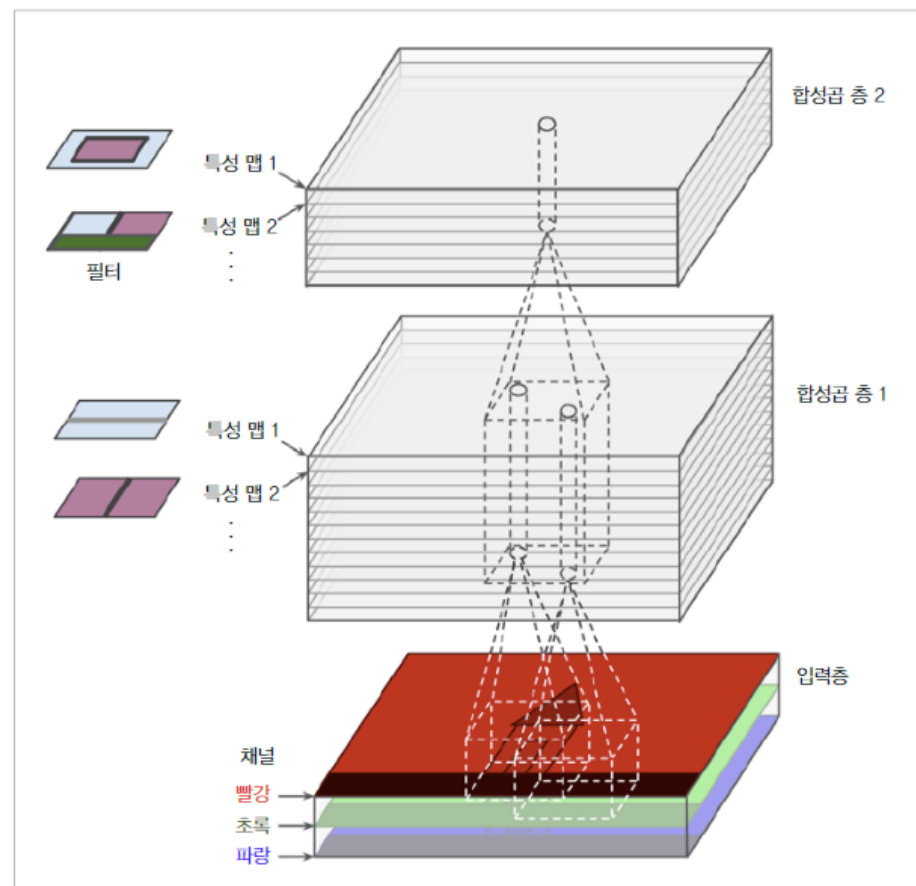


그림 14-6 여러 가지 특성 맵으로 이루어진 합성곱 층과 세 개의 컬러 채널을 가진 이미지

텐서플로우에서 합성곱층

텐서플로우에서 각 입력 이미지는 보통 [높이, 너비, 채널] 형태의 3d 텐서로 표현되며, 하나의 미니배치는 [미니배치 크기, 높이, 너비, 채널] 형태의 4d 텐서로 표현된다.

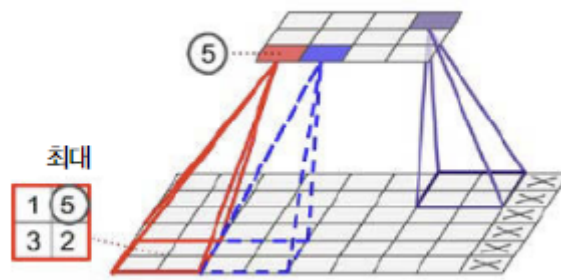
```
conv = keras.layers.Conv2D(filters=32, kernel_size=3, strides=1,
padding="same", activation="relu")
```

정리하면, 합성곱 층을 위해 다음의 하이퍼파라미터가 필요하다. 필터의 수, 필터의 높이와 너비, 스트라이드, 패딩 종류

풀링층 (pooling layer)

풀링층의 목적은 계산량과 메모리 사용량, 파라미터 수를 줄이기 위해 입력 이미지의 subsample (=축소본) 을 만드는 것이다.

풀링층에서도 동일하게 크기, 스트라이드, 패딩 유형을 지정해야 한다. 하지만 풀링 뉴런은 가중치가 없다. 최대나 평균 같은 합산 함수를 사용해 입력값을 더하는 것이 전부이다. (한즈온 예제에서는 풀링층에 패딩은 지정하지 않는다.)



텐서플로우에서 풀링층: Max, Average

다음 코드는 2*2 커널을 사용해 최대 풀링 층을 만든다. 스트라이드의 기본값은 커널 크기이므로 이 층은 수평, 수직 모두 스트라이드 2를 사용한다. 기본적으로 “valid” 패딩을 사용한다.

```
max_pool = keras.layers.MaxPool2D(pool_size=2)
```

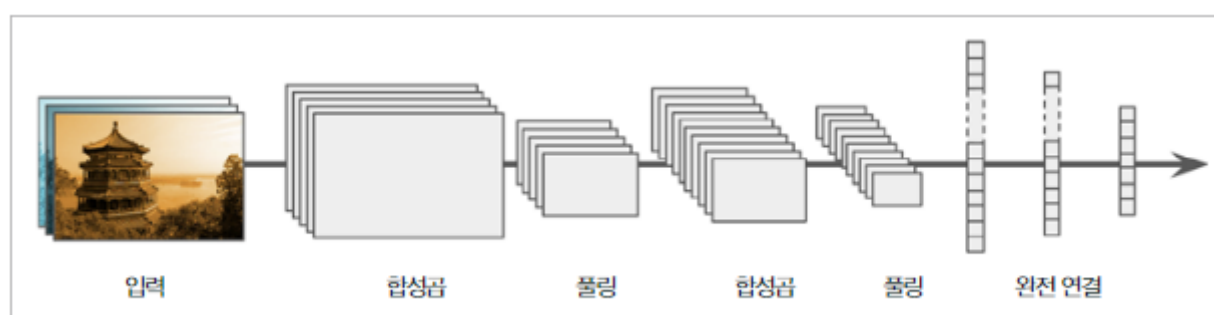
평균 풀링 층을 만들려면 AvgPool2D 를 사용하면 된다. 평균을 사용하면 최댓값을 이용하는 것보다 정보 손실이 적고, 최대 풀링층을 사용하면 의미 없는 것은 모두 제거하고 가장 큰 특징만 유지한다. 일반적으로 최대 풀링 층이 더 성능이 좋게 나온다. 이러한 이유에는, 최댓값을 사용하면, 다음 layer 에서 조금 더 명확한 신호로 작업할 수 있기 때문이다. 반대로 평균 풀링은 평균을 계산하기 때문에 특징을 희석시키는 효과를 낸다.

텐서플로우에서 풀링층: Global average

전역 평균 풀링 층은 현대적인 신경망 구조에서 종종 볼 수 있다. 이 층은 각 특성 맵의 평균을 계산한다. 즉, 각 샘플의 특성 맵마다 하나의 숫자를 출력한다. 특성 맵에 있는 대부분의 정보를 잃을지만 출력층에는 유용할 수 있다.

CNN 구조

전형적인 CNN 구조는 합성곱 층을 몇 개 쌓고, 그 다음에 풀링층을 쌓고, 그 다음에 또 합성곱 층을 몇 개 더 쌓고, 그 다음에 다시 풀링층을 쌓는 방식이다. 네트워크를 통과하여 진행할수록 이미지는 점점 작아지지만, 합성곱 층 때문에 일반적으로 점점 더 깊어진다. 맨 위층에는 몇 개의 완전 연결 층으로 구성된 일반적인 피드포워드 신경망이 추가되고 마지막 층에서 예측을 출력한다.



텐서플로우에서 위 네트워크를 다음의 모델로 구현한다.

```
model = keras.models.Sequential([
    # input layer
```

```

keras.layers.Conv2D(64, 7, activation='relu', padding='same',
                    input_shape=[28, 28, 1]),
keras.layers.MaxPooling2D(2),

# middel
keras.layers.Conv2D(128, 3, activation='relu', padding='same'),
keras.layers.Conv2D(128, 3, activation='relu', padding='same'),
keras.layers.MaxPooling2D(2),

keras.layers.Conv2D(256, 3, activation='relu', padding='same'),
keras.layers.Conv2D(256, 3, activation='relu', padding='same'),
keras.layers.MaxPooling2D(2),

# fully-connected layer
keras.layers.Flatten(),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dropout(0.5),
keras.layers.Dense(64, activation='relu'),
keras.layers.Dropout(0.5),

# output layer
keras.layers.Dense(10, activation='softmax')
])

```

- 28*28 픽셀 사이즈의 컬러 채널이 1인 흑백 이미지를 사용했다.
- input layer 는 64개의 7*7 크기의 필터 와 스트라이드 1을 사용한다. 그 다음 풀링 크기가 2인 maxpooling layer 을 추가하여 공간 방향 차원을 절반으로 줄인다.
- CNN 이 output layer 에 다다를수록 필터 개수가 늘어난다. 64 → 128 → 256. 저수준 특성의 개수는 적지만 이를 연결하여 고수준 특성을 만들 수 있는 방법이 많기에 이런 구조가 합리적이다.
- 풀링 층 다음에 필터 개수를 두 배로 늘리는 것이 일반적인 방법이다. 풀링 층이 공간 방향 차원을 절반으로 줄이므로 이어지는 층에서 파라미터 개수, 메모리 사용량, 계산 비용을 크게 늘리지 않고 깊이에 해당하는 특성 맵 개수를 두 배로 늘릴 수 있다.

LeNet-5

LeNet-5 의 layer 구조는 다음과 같다.

표 14-1 LeNet-5 구조

층	종류	특성 맵	크기	커널 크기	스트라이드	활성화 함수
출력	완전 연결	-	10	-	-	RBF
F6	완전 연결	-	84	-	-	tanh
C5	합성곱	120	1×1	5×5	1	tanh
S4	평균 풀링	16	5×5	2×2	2	tanh

층	종류	특성 맵	크기	커널 크기	스트라이드	활성화 함수
C3	합성곱	16	10×10	5×5	1	tanh
S2	평균 풀링	6	14×14	2×2	2	tanh
C1	합성곱	6	28×28	5×5	1	tanh
입력	입력	1	32×32	-	-	-

출력층은 입력과 가중치 벡터를 행렬 곱셈하는 대신, 각 뉴런에서 입력 벡터와 가중치 벡터 사이의 유클리드 거리를 출력한다. 각 출력은 이미지가 얼마나 특정 숫자 클래스에 속하는지 측정한다.

AlexNet

AlexNet CNN 구조는 2012년 이미지넷 대회에서 압도적인 성능으로 우승하여 유명해졌다. 구조는 다음 표와 같다.

표 14-2 AlexNet 구조

층	종류	특성 맵	크기	커널 크기	스트라이드	패딩	활성화 함수
출력	완전 연결	-	1,000	-	-	-	Softmax
F10	완전 연결	-	4,096	-	-	-	ReLU
F9	완전 연결	-	4,096	-	-	-	ReLU
F8	최대 풀링	256	6×6	3×3	2	valid	-
C7	합성곱	256	13×13	3×3	1	same	ReLU
C6	합성곱	384	13×13	3×3	1	same	ReLU
C5	합성곱	384	13×13	3×3	1	same	ReLU
S4	최대 풀링	256	13×13	3×3	2	valid	-
C3	합성곱	256	27×27	5×5	1	same	ReLU
S2	최대 풀링	96	27×27	3×3	2	valid	-
C1	합성곱	96	55×55	11×11	4	valid	ReLU
입력	입력	3 (RGB)	227 × 227	-	-	-	-

합성곱을 연속해서 쌓는 방식이 처음 도입된 모델이다.

과대적합을 줄이기 위해 두 가지 regularization 기법을 사용했다. 우선, 훈련하는 동안 F9과 F10의 출력에 Dropout을 50% 적용했고, 그리고 왜곡시킨 이미지를 사용하는 data augmentation 을 수행했다.

data augmentation 은 이미지의 크기를 변경하거나 이동, 회전시키거나, 색깔, 명암, 배경을 변화시키는 방법을 의미한다. 훈련 세트의 크기를 많이 늘릴 수 있을 뿐더러, 일부 특성에 민감하지 않은, 조금 더 general 한 모델을 만들 수 있다.

AlexNet 은 C1, C3 층의 ReLU 단계 후에 “local response normalization; LRN” 이라 부르는 경쟁적인 정규화 단계를 추가했다. 가장 강하게 활성화된 뉴런이 다른 특성 맵에 있는 같은 위치의 뉴런을 억제한다. 이는 특성 맵을 각기 특별하게 다른 것과 구분되게 하고, 더 넓은 시각에서 특징을 탐색하도록 만들어 결국 일반화 성능을 향상시킨다.

식 14-2 LRN

$$b_i = a_i \left(k + \alpha \sum_{j=j_{\text{low}}}^{j_{\text{high}}} a_j^2 \right)^{-\beta} \quad \text{여기서} \quad \begin{cases} j_{\text{high}} = \min \left(i + \frac{r}{2}, f_n - 1 \right) \\ j_{\text{low}} = \max \left(0, i - \frac{r}{2} \right) \end{cases}$$

- b_i 는 i 특성 맵, u 행, v 열에 위치한 뉴런의 정규화된 출력이다.
- a_i 는 ReLU 단계를 지나고 정규화 단계를 거치기 전인 뉴런의 활성화 값이다.
- k, α, β, r 은 하이퍼파라미터 이며, k 는 편향, r 은 깊이 반경이다.
- f_n 은 특성 맵의 수이다.

AlexNet 에서 하이퍼파라미터는 $r=2, \alpha=0.00002, \beta=0.75, k=1$ 로 설정되었다. 이 단계는 `tf.nn.local_response_normalization()` 연산을 사용하여 구현할 수 있다.