4주차 CNN과 RNN의 이해

7차시: CNN(합성곱 신경망)의 원리

○ CNN 탄생과 흐름

○ CNN의 기본 개념: 합성곱 연산, 필터, 풀링

○ 이미지 데이터의 처리와 CNN 적용 방법

○ CNN을 활용한 이미지 분류 문제 실습

○ CNN 적용사례

○ CNN(Convolutional Neural Network, 합성곱 신경망 탄생과 흐름

CNN은 딥러닝 분야에서 특히 컴퓨터 비전 분야에서 중요한 역할을 하는 신경망 아키텍처입니다. CNN은 이미지, 비디오, 음성, **시계열 데이터**와 같은 데이터를 처리하는 데 매우 효과적입니다.

1. 신경망 초기 역사 (1950~1980년대)

- 신경망 자체는 1950년대부터 연구되기 시작했습니다. 그중 퍼셉트론(Perceptron)**이라는 모델이 1958년 프랭크 로젠블랫(Frank Rosenblatt)에 의해 제안되었습니다.
- 퍼셉트론은 간단한 뉴런 모델로, 입력 데이터를 선형적으로 분리하는 데 사용되었습니다. 그러나 퍼셉트론은 비선형 문제를 해결하지 못하는 한계가 있었으며, 특히 XOR 문제와 같은 비선형 분류 문제에서는 실패했습니다.
- 이후 다층 퍼셉트론(MLP, Multi-Layer Perceptron)이 제안되어 이 한계를 극복하려는 시도가 이루어졌습니다.

2. 합성곱 개념의 도입 (1960~1980년대)

- **합성곱(Convolution)의 개념**은 컴퓨터 비전 분야에서 이미지 처리에 사용되기 시작했습니다. 1960년대부터 생물학적 신경망 연구가 진행되면서, 시각 피질에서 특정 자극에 반응하는 뉴런들의 특성을 연구하던 데이비드 휴벨(David Hubel)과 토르스텐 비셀(Torsten Wiesel)의 연구가 CNN의 기초 개념을 형성하는데 기여했습니다.
- 그들은 시각 피질에서 뉴런들이 국소적으로 시야의 작은 영역에 반응한다는 사실을 발견했습니다. 즉, 뉴런들이 필터 역할을 하며 입력 이미지의 작은 부분(지역적인 정보)에 반응하는 국소 수용 영역(Receptive Field)을 가지고 있다는 개념이 형성되었습니다.
- 이 연구는 CNN의 합성곱 필터 개념의 기초를 마련했습니다.
-

3. 1980년대: CNN 개념의 시작 - 네오코그니트론(Neocognitron)

- CNN의 직접적인 전신은 1980년에 일본의 쿠히코 후쿠시마(Kunihiko Fukushima)에 의해 제안된 네오코그니트론(Neocognitron)입니다.
- 네오코그니트론: 시각 패턴을 인식하기 위한 계층적 모델로, 여러 층의 필터를 사용해 이미지를 처리하는 방식을 제안했습니다.
- 네오코그니트론은 합성곱과 풀링 층 개념을 도입하였으며, 오늘날의 CNN과 유사한 계층적 구조를 가지고 있었습니다.
- 하지만, 역전파 알고리즘을 사용하지 않았기 때문에 가중치 학습이 불가능했고, 수동으로 필터를 설정해야 하는 한계가 있었습니다.

4. 1990년대: CNN의 탄생 - 얀 르쿤(Yann LeCun)의 LeNet-5

- CNN의 탄생과 실질적인 발전은 1990년대에 얀 르쿤(Yann LeCun)에 의해 이루어졌습니다. 르쿤은 역전파 알고리즘을 사용하여 CNN의 가중치를 자동으로 학습할 수 있는 방법을 제안했고, 이는 LeNet이라는 CNN 모델로 발전하게 됩니다.
- LeNet-5 (1998년)
- LeNet-5는 얀 르쿤과 그의 연구팀이 1998년에 발표한 CNN 모델로, **손글씨 숫자(MNIST 데이터셋)를 인식하기 위해 개발**되었습니다.
- LeNet-5는 **합성곱 층**, **풀링 층**(서브샘플링 층), **완전 연결층(Fully Connected Layer)**으로 구성된 최초의 CNN 모델 중 하나입니다.
- 합성곱 층: 이미지의 특정 패턴을 인식하기 위한 필터(커널)를 적용하여 특징을 추출합니다.
- 풀링 층: 데이터를 다운샘플링하여 계산량을 줄이고, 공간적인 크기를 축소합니다.
- LeNet-5는 그 당시 이미지 분류 작업에서 탁월한 성능을 보였고, CNN의 효용성을 입증한 모델이었습니다.

5. 2010년대: 딥러닝과 CNN의 부활 - AlexNet (2012년)

- CNN의 발전은 2000년대까지 크게 주목받지 못했지만, 2010년대 초반에 다시 주목을 받기 시작했습니다.
- 특히, 2012년 ImageNet 대회에서 알렉스 크리제브스키(Alex Krizhevsky), 일리아 수츠케버(Ilya Sutskever), 그리고 제프리 힌튼(Geoffrey Hinton)이 개발한 AlexNet이 CNN을 기반으로 압도적인 성능을 기록하면서 CNN은 다시 부활하게 되었습니다.
- **AlexNet (2012년)** : AlexNet은 ImageNet 데이터셋에서 1000개 카테고리의 이미지를 분류하기 위해 개발된 CNN 모델입니다.
- AlexNet은 LeNet과 유사한 구조를 가지고 있었지만, 더 깊고 복잡한 네트워크였으며, 더 많은 합성곱 층과 필터를 사용했습니다.

주요 특징:

- ReLU 활성화 함수 사용: 비선형성을 추가하여 학습을 가속화.
- Dropout 기법 도입: 과적합을 방지하기 위한 기법.
- GPU 사용: 대량의 데이터와 복잡한 모델을 학습시키기 위해 GPU를 적극 활용.

AlexNet의 성공은 CNN과 딥러닝의 재부흥을 이끌었으며, 이후 많은 CNN 기반 모델들이 개발되었습니다.

6. 현대의 CNN 발전

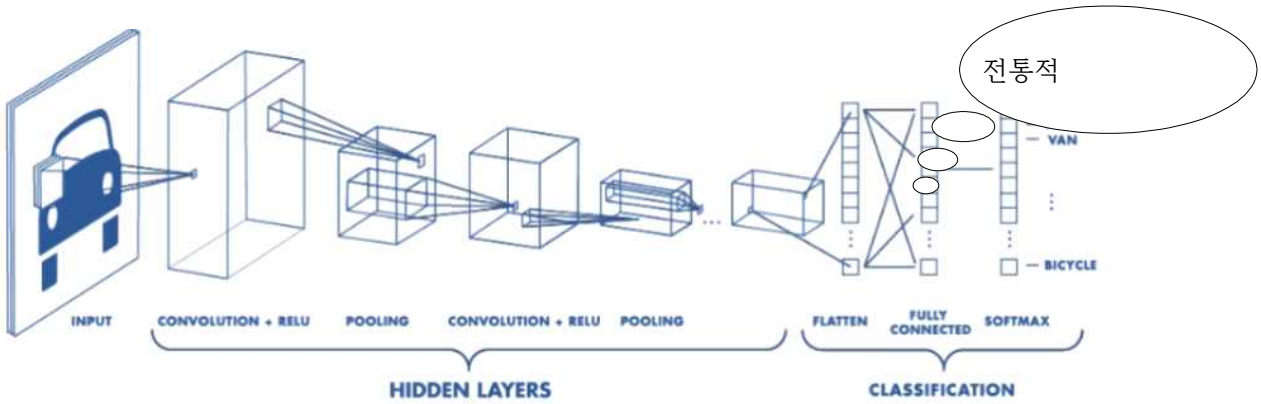
- AlexNet 이후 CNN은 급속히 발전하여 다양한 아키텍처가 제안되었습니다. 주요 CNN 모델들의 발전 과정은 다음과 같습니다.
- VGGNet (2014년)
- VGGNet은 2014년에 개발된 CNN 모델로, 더 깊은 신경망을 사용하여 성능을 개선한 모델입니다. VGGNet은 3x3 필터만을 사용하여 더 많은 합성곱 층을 추가하면서도 계산의 단순성을 유지하였습니다.
- GoogLeNet/Inception (2014년)
- GoogLeNet은 Inception 모듈을 도입하여 CNN의 깊이와 너비를 모두 확장한 모델입니다. 병렬 필터를 통해 서로 다른 크기의 필터와 풀링 층을 결합함으로써 다양한 수준의 특징을 추출할 수 있었습니다.

ResNet (2015년)
ResNet(Residual Network)은 2015년 Kaiming He와 동료들이 개발한 CNN 아키텍처로, 잔차 연결(Residual Connection)**을 도입하여 매우 깊은 신경망도 효과적으로 학습할 수 있게 하였습니다. ResNet은 152층의 매우 깊은 네트워크로도 학습이 가능했으며, 기울기 소실 문제를 극복한 혁신적인 모델로 평가받습니다.

○ CNN의 기본 개념: 합성곱 연산, 필터, 풀링

CNN(Convolutional Neural Network, 합성곱 신경망)

주로 이미지 처리와 같은 2차원 데이터를 다루는 데 효과적인 딥러닝 모델입니다.
CNN은 이미지나 다른 데이터의 중요한 특징을 자동으로 추출할 수 있는 능력을 가지고 있어, **이미지 분류**, 객체 인식, 영상 처리 등 다양한 분야에서 널리 사용됩니다.



CNN은 **이미지의 특징을 추출하는 부분**과 **클래스를 분류하는 부분**으로 나뉘어지는데,
특징 추출 영역에서 Convolution Layer가 여러 겹을 쌓는 형태로 구성된다.
Convolution Layer는 입력된 데이터에 필터를 적용시킨 후, 활성화 함수를 반영한다.

CNN은 이 입력 이미지를 받아서 특징 추출 과정을 거칩니다. 이때, 벡터화는 CNN에서 마지막에 완전 연결층(Fully Connected Layer)에 입력 데이터를 전달하기 전에 이루어집니다.

■ 1. Convolutional Layer (합성곱 층)

1) 과정
CNN의 핵심 구성 요소입니다. **필터(Filter, 커널)**[1]를 사용하여 **입력 이미지**[2] **(픽셀)**[3]의 작은 영역을 스캔하면서 **특징맵**(feature map)을 생성합니다.
필터는 이미지의 특정 패턴이나 특징(예: 엣지, 코너 등)을 인식하며, 이를 통해 입력 데이터의 특징을 학습합니다.

---

1) 주석1
2) 주석2
3) 주석3

2) 합성곱 계층(Convolutional Layer) 이미지를 처리하는 전반적인 과정

- 필터(Filter)를 사용하여 이미지의 국소 영역[4]에서 특징 맵(Feature Map)을 추출합니다.
- 필터는 이미지 위를 슬라이딩하면서 이미지의 중요한 패턴(모서리, 질감 등)을 학습합니다.
- 이 과정을 거쳐 입력 이미지는 여러 개의 특징 맵으로 변환됩니다. 특징 맵은 여전히 2차원 또는 3차원 배열로 유지됩니다.

2) -1 필터 이해

- CNN에서 필터(커널)는 이미지의 패턴이나 특징을 학습하기 위해 사용되며, 학습 과정에서 자동으로 최적화되는 파라미터입니다
- 초기 값: CNN이 처음 학습을 시작할 때 필터(커널)는 일반적으로 무작위로 초기화됩니다. 이 무작위 값은 주로 평균이 0이고 분산이 작은 값으로 설정되며, 이를 통해 학습이 시작됩니다.
- 필터는 이미지의 작은 부분(패치)을 스캔하면서 합성곱 연산(Convolution)을 수행합니다.

2) -2 입력이미지

- 입력 이미지를 수치 벡터화(벡터로 변환)하는 과정은 이미지 데이터를 머신러닝 및 딥러닝 모델에서 사용할 수 있는 수치 데이터로 변환하는 것을 의미합니다.
- 이 과정에서 이미지는 각 픽셀의 값으로 변환되며, 이 값들은 벡터 또는 행렬로 표현됩니다. 이미지의 픽셀 값은 그레이스케일 이미지와 컬러 이미지에 따라 다르게 표현되며, 다양한 차원의 수치 데이터로 벡터화됩니다.

▶ 이미지 데이터란?

1) 그레이스케일 이미지

- 각 픽셀은 밝기(0 ~ 255)로 표현되며, 2차원 행렬로 나타낼 수 있습니다.
- 값이 클수록 더 밝은 색상을 나타냅니다.
- 예: 28(행)x28(열) 크기의 그레이스케일 이미지는 28×28=784
- 28×28=784개의 픽셀로 이루어진 2차원 배열입니다.

[34, 45, 67, ..., 120]
[255, 254, 200, ..., 0]
[12, 34, 67, ..., 255]
...
[0, 78, 12, ..., 34]
각 숫자는 특정 픽셀의 밝기값을 나타내며, 전체 배열이 이미지를 나타냅니다.
특징
크기: 28×28 = 784개의 픽셀
표현 방식: 흑백 이미지의 경우 0~255의 밝기값으로 각 픽셀의 값을 설정
응용: 이미지 인식, 특히 글자나 숫자 인식에서 자주 사용

---

4)

2) 컬러 이미지

- : 각 픽셀은 RGB 값(Red, Green, Blue)으로 구성된 3차원 배열로, 각 색상의 채널에 대한 값이 0부터 255 사이의 값을 가집니다.
- 컬러 이미지는 3차원 배열로 표현됩니다.

예를 들어, 28×28의 컬러 이미지라면 28x28x3 배열이 됩니다.
첫 번째 차원은 행(row)의 개수
두 번째 차원은 열(column)의 개수 (
세 번째 차원은 색상 채널(R, G, B)의 개수입니다.
각 색상 채널에는 0부터 255까지의 값이 들어가며, 이 값들이 조합되어 특정 픽셀의 색상을 결정합니다.
예시: 28x28x3 컬러 이미지 배열

```
[   [[255, 0, 0], [0, 255, 0], [0, 0, 255], ...],
    [[128, 128, 128], [64, 64, 64], [192, 192, 192], ...],
    ...
    [[255, 255, 255], [0, 0, 0], [120, 60, 30], ...]]
```

위의 구조에서, 각 픽셀은 [R, G, B] 값으로 이루어져 있습니다.
[255, 0, 0]은 빨간색, [0, 255, 0]은 초록색, [0, 0, 255]은 파란색을 나타냅니다.
[128, 128, 128]은 회색, [255, 255, 255]은 흰색, [0, 0, 0]은 검은색을 나타냅니다.
이 배열에서 28개의 행이 있고, 각 행에 28개의 열이 있으며, 각 픽셀마다 RGB 채널이 존재하므로 총 28x28x3 크기의 데이터가 됩니다.

2) -3 특징 맵(feature map) 형성 방법

필터(커널)를 입력 이미지에 적용하여 특징 맵(feature map)을 생성하는 과정을 의미합니다.
이 과정은 필터가 이미지의 각 영역을 스캔하며 합성곱 연산을 수행해 특정 패턴이나 특징을 감지한 후, 새로운 출력을 만들어내는 방식으로 진행됩니다.

[맵 형성의 단계별 과정]

1. 입력 이미지와 필터 설정

1) 입력 이미지: CNN에 입력되는 이미지입니다.
보통 이미지의 크기와 채널 수가 주어집니다.
예를 들어, 32x32 크기의 RGB 이미지라면 입력 이미지의 차원은 (32, 32, 3)입니다. 그레이스케일 이미지의 경우는 (높이, 너비, 1) 형식이 됩니다.
2) 필터(커널): 합성곱 연산을 수행하는 작은 크기의 매트릭스로, 주로 3x3, 5x5와 같은 크기를 가집니다. 필터는 처음에는 무작위로 초기화되지만, 학습 과정에서 점진적으로 최적화됩니다.
예시:
입력 이미지 크기: (32, 32, 3) (RGB 이미지)

필터 크기: 3x3x3 (3채널 이미지에 맞춘 필터)

2. 패치 추출 및 합성곱 연산

- 필터는 입력 이미지의 작은 영역(패치)을 슬라이딩하면서 합성곱 연산을 수행합니다.
- 필터와 이미지 패치 간의 합성곱 연산(Convolution)은 이미지의 각 픽셀 값과 필터의 값을 곱한 후 더하는 방식(합성곱)으로 이루어집니다.
- 필터는 스트라이드(Strides) 설정에 따라 한 번에 한 칸씩, 혹은 여러 칸씩 이동하며 연산을 수행합니다.

합성곱 연산 예시: 입력 이미지의 일부 패치(3x3 크기):

합성곱 연산에서, 필터 K와 이미지 I에 대해 각 포지션에서의 출력 $O(i, j)$는 다음과 같이 계산됩니다.

$$O(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n) \cdot I(i+m, j+n)$$

K는 $M \times N$ 필터(커널)입니다.

I는 입력 이미지입니다.

i, j는 현재 필터가 적용되는 이미지의 위치

(사례)

예제: 3x3 필터를 적용한 5x5 입력 이미지

1. 입력 이미지 (5x5)

입력 이미지는 각 픽셀의 밝기값으로 구성된 5x5 배열

```
[  [1, 2, 0, 1, 3],
   [4, 1, 0, 2, 2],
   [3, 2, 1, 0, 1],
   [1, 1, 3, 2, 0],
   [0, 2, 3, 1, 1]
]
```

2. 필터(커널) (3x3)

합성곱 연산에 사용되는 3x3 크기의 필터입니다.

필터 (3x3):

```
[  [1, 0, -1],
   [1, 0, -1],
   [1, 0, -1]
]
```

합성곱 연산 과정

합성곱 연산은 필터를 입력 이미지 위에서 왼쪽 위부터 오른쪽 아래까지 이동시키면서 각 위치에서 이미지와 필터의 요소를 곱하고 합산합니다.

보통 스트라이드(stride)는 1로 설정하여 한 번에 한 칸씩 이동하며,

패딩(padding)은 하지 않아서 출력 크기는 (5-3+1)x(5-3+1) = 3x3가 됩니다.

합성곱 연산 예시 (첫 번째 위치에서 계산)
필터를 (0, 0) 위치에 적용합니다.
입력 이미지의 첫 번째 3x3 부분과 필터의 각 요소를 곱한 후 합산합니다.
계산:
(1*1) + (2*0) + (0*-1) +
(4*1) + (1*0) + (0*-1) +
(3*1) + (2*0) + (1*-1) =
1 + 0 + 0 + 4 + 0 + 0 + 3 + 0 - 1 = 7

첫 위치에서의 출력값은 7입니다.

4. 전체 연산 결과 (출력 특징 맵)
이렇게 필터를 이미지 전체에 적용해 각 위치에서의 출력을 구합니다.

입력 이미지 (5x5) , 필터(커널) (3x3)

출력 특징 맵 (3x3): 스트라이드(stride)= 1, 패딩(padding)=0
[[7, 4, 2],
[4, 2, 3],
[2, 1, -1]
]

[1, 2, 1]
[0, 1, 2]
[1, 0, 1]
필터(3x3):

코드 복사
[1, 0, -1]
[1, 0, -1]
[1, 0, -1]
합성곱 연산 결과:
코드 복사
(1*1) + (2*0) + (1*-1) + (0*1) + (1*0) + (2*-1) + (1*1) + (0*0) + (1*-1) = 0
이 연산이 이미지 전체에서 반복되면서 특징 맵이 형성됩니다.

파이썬 실습 참고 : CNN 모델 (합성곱.패딩..)

3. 특징 맵 형성

합성곱 연산 결과는 특징 맵(feature map)으로 나타나며, 필터가 입력 이미지에서 감지한 특정 패턴의 강도를 표현합니다.

필터가 특정 패턴(예: 가장자리, 선 등)을 감지하면 해당 위치에서 높은 값을 출력하고, 감지하지 못하면 낮은 값을 출력합니다.

이 과정을 통해 CNN은 이미지에서 중요한 패턴을 발견하고, 이를 새로운 맵으로 변환합니다.

예시:

필터가 이미지에서 수직선을 감지하는 필터라면, 수직선이 있는 부분에서 높은 값이 출력되고, 없는 부분에서는 낮은 값이 출력됩니다.

4. 패딩과 스트라이드

4.1 Padding(패딩)

→ 입력 이미지에 대해 ***합성곱***을 수행하면, ***출력 이미지의 크기는 입력 이미지의 크기보다 작아지게 된다.***

만역, 4x4 였던 입력 이미지가 합성곱을 수행하여 2x2 의 출력 이미지의 크기를 갖는다. 그러므로 합성곱 계층을 거치면서 이미지의 크기는 점점 작아지게 되고, 이미지의 가장자리에 위치한 픽셀들의 정보는 점점 사라지게 된다. 이러한 문제점을 해결하기 위해 이용되는 것이 패딩 (padding)이다.

- 패딩은 입력 이미지의 가장자리에 특정 값으로 설정된 픽셀들을 추가함으로써 입력 이미지와 출력 이미지의 크기를 같거나 비슷하게 만드는 역할을 수행한다.
- 이미지의 가장자리에 0의 값을 갖는 픽셀을 추가하는 것을 zero-padding이라고 하며, CNN에서는 주로 이러한 zero-padding이 이용된다.

- 필터가 입력의 경계를 처리할 때 데이터를 확장하거나 유지하는 방식입니다.
- 주로 출력 크기를 조절하거나 정보를 보존하기 위해 사용됩니다.
- 패딩은 필터가 이미지의 가장자리를 처리할 때 사용할 수 있는 기술로, 입력 이미지의 경계에 0 또는 특정 값을 추가하는 방식입니다.
- 패딩을 사용하면 필터가 이미지의 가장자리에서도 완전히 스캔할 수 있으며, 출력 크기를 입력 크기와 같게 유지할 수 있습니다.

패딩을 사용하면 출력 크기가 줄어드는 것을 방지할 수 있으며, 입력 이미지의 정보를 더 잘 보존할 수 있습니다.

패딩의 종류

- **Valid 패딩**: 패딩을 사용하지 않고 필터가 입력 이미지 내에서만 합성곱을 수행합니다. 이 경우 출력 크기가 입력 크기보다 작아집니다.
- Same 패딩: 입력 이미지의 경계에 패딩을 추가하여 출력 크기를 입력 크기와 동일하게 유지합니다.

예시:

입력 이미지가 5x5 크기이고, 3x3 필터와 스트라이드 1을 사용하는 경우, 패딩이 없으면 출

력은 3x3 크기로 줄어듭니다.
하지만 패딩을 사용하여 입력 이미지의 경계를 확장하면 출력 크기가 5x5로 유지됩니다.

(파이썬 실습 사례 참조 할 것)

4.2 스트라이드(Strides)[5)]

- 필터가 한 번에 이동하는 칸 수를 결정하는 값입니다.
- 스트라이드가 1이면 필터가 한 칸씩 이동하며, 스트라이드가 2면 두 칸씩 건너뛰면서 이동합니다. 스트라이드 값이 클수록 출력 크기는 작아집니다.

스트라이드와 패딩에 따른 특징 맵 크기:

출력 크기 = ((입력 크기 - 필터 크기 + 2 * 패딩) / 스트라이드) + 1
예시: 입력 이미지: 32x32
필터 크기: 3x3
패딩: 1
스트라이드: 1

코드 복사
출력 크기 = ((32 - 3 + 2 * 1) / 1) + 1 = 32
이 경우 출력 맵의 크기는 32x32로 유지됩니다.

5. 비선형성 도입 (활성화 함수)

- 합성곱 연산을 통해 생성된 특징 맵에 비선형 활성화 함수를 적용하여 모델이 비선형 패턴을 학습할 수 있게 합니다.
- 가장 많이 사용되는 활성화 함수는 ReLU(Rectified Linear Unit)입니다. ReLU는 0 이하의 값을 모두 0으로 변환하고, 0 이상의 값은 그대로 유지합니다.
- 이를 통해 특징 맵에서 음수 값을 제거하고, 중요한 패턴을 더 부각시킵니다.

ReLU 적용 예시:

입력 특징 맵: [-2, 3, -1, 5]
ReLU 적용 후: [0, 3, 0, 5]

6. 여러 필터의 사용

CNN은 각 층에서 여러 개의 필터를 사용하여 서로 다른 패턴을 감지합니다.
예를 들어, 첫 번째 합성곱 층에서 32개의 필터를 사용한다면, 이 층은 32개의 다른 특징 맵을 생성하게 됩니다. 각각의 필터는 이미지에서 서로 다른 특징(예: 수직선, 수평선, 모서리 등)을 감지할 수 있습니다.
각 필터는 고유한 특징을 추출하며, 다양한 필터를 사용함으로써 CNN은 더 복잡한 패턴을 학습할 수 있습니다.

---

5) 주석4

예시:
32개의 필터를 사용한 합성곱 층:
- 필터 1: 수직선 감지 → 수직선 특징 맵
- 필터 2: 수평선 감지 → 수평선 특징 맵
- 필터 3: 곡선 감지 → 곡선 특징 맵

7. 계층적 특징 학습
- CNN은 계층적 구조를 가지고 있기 때문에, 첫 번째 층에서는 단순한 특징(예: 가장자리, 선)을 학습하고, 더 깊은 층에서는 이러한 특징을 결합해 더 복잡한 패턴(예: 모양, 객체)을 학습합니다.
- 각 층에서의 필터는 앞선 층의 특징 맵을 입력으로 받아 더 고차원적인 패턴을 추출합니다.

예시:
- 첫 번째 층: 수직선, 수평선 등의 저차원 패턴을 감지.
- 두 번째 층: 저차원 패턴을 조합해 객체의 윤곽선이나 형태를 감지.
- 마지막 층: 객체의 고차원적인 패턴을 감지하고 분류.

요약:
맵 형성은 필터가 이미지의 각 영역을 스캔하면서 합성곱 연산을 수행해 **특징 맵(feature map)**을 생성하는 과정입니다.
필터는 이미지의 패턴(선, 가장자리, 모양 등)을 감지하고, 각 필터는 고유한 특징을 학습합니다.
패딩과 스트라이드 설정을 통해 출력 맵의 크기를 조절하며, 활성화 함수(ReLU)를 사용해 비선형성을 도입하여 모델의 학습 능력을 향상시킵니다.
여러 개의 필터를 사용해 다양한 특징을 추출하고, CNN의 계층적 구조를 통해 점차 더 복잡한 특징을 학습합니다.
이러한 과정은 CNN이 이미지에서 의미 있는 정보를 추출하여 최종적으로 분류나 객체 인식과 같은 작업을 수행할 수 있도록 합니다.
-

■ CNN에서 슬라이딩(Sliding)[6)]

- CNN에서 슬라이딩(Sliding)은 필터(커널)가 입력 이미지의 각 위치에서 합성곱 연산(Convolution)을 수행하는 과정을 말합니다.
- 슬라이딩은 필터가 이미지를 좌측 상단에서 시작해 우측 하단까지 일정한 규칙에 따라 이동하면서 이루어집니다.
- 이때 필터는 입력 이미지의 작은 영역을 스캔하고, 해당 영역에서 합성곱 연산을 통해 특징을 추출합니다.

6) 주석5

슬라이딩 방식의 주요 요소

슬라이딩 방식은 크게 **필터 크기, 스트라이드(Strides), 패딩(Padding)에 의해 결정**됩니다.

- 필터(커널) 크기

필터는 입력 이미지에서 특징을 추출하기 위한 작은 매트릭스입니다. 필터의 크기는 일반적으로 3x3, 5x5, 7x7 등으로 설정됩니다.

필터가 입력 이미지에서 작은 영역을 한 번에 스캔하고, 이 영역에서 합성곱(Convolution) 연산을 수행합니다.

- **스트라이드(Strides)**

스트라이드는 필터가 입력 이미지 위에서 이동하는 간격을 말합니다.

예시:

스트라이드 1: 필터가 입력 이미지에서 한 픽셀씩 이동하며 모든 영역을 스캔합니다.

입력 이미지가 5x5 크기이고, 필터 크기가 3x3, 스트라이드가 1이라면 필터는 총 9번(3x3 크기의 출력 맵) 스캔합니다.

스트라이드 2: 필터가 두 픽셀씩 건너뛰면서 이동합니다. 이 경우 출력 맵의 크기가 더 작아집니다.

입력 이미지가 5x5 크기이고, 필터 크기가 3x3, 스트라이드가 2라면 필터는 총 4번(2x2 크기의 출력 맵) 스캔합니다.

■ 2. Pooling Layer (풀링 층) 동작원리

Pooling은 CNN(Convolutional Neural Network) 모델에서 차원을 축소하고, 연산량을 줄이며, 중요한 특징을 유지하면서 모델의 과적합을 방지하기 위해 사용되는 중요한 연산입니다.

- Pooling Layer는 **입력된 특징 맵(feature map)을 작은 영역으로 나누고,** 각 영역에서 특정 값을 선택하여 **특징 맵의 크기를 줄이는 역할**을 한다. → 계산 비용을 줄이기 위해 사용합니다.
- 특징 맵(feature map)의 공간적 크기를 줄여주면서 중요한 정보를 유지하는 역할
- 풀링을 통해 모델의 과적합(overfitting)을 방지하는 동시에, 중요한 특징을 추출하여 모델이 더 효율적으로 학습할 수 있도록 돕습니다.
- 풀링 레이어는 비학습 층으로, 가중치 업데이트 없이 데이터를 처리합니다.

2-1. Pooling의 목적

- 차원 축소: 풀링은 입력의 공간적 크기를 줄임으로써 모델의 계산량을 감소시키고, 메모리 사용량을 줄이는 역할을 합니다.
- 불변성 제공: 풀링은 위치 변화나 왜곡에 대해 더 강건한 특성을 가지도록 도와줍니다. 이는 모델이 이미지의 작은 변화에도 민감하게 반응하지 않고, 중요한 특징을 잘 추출할 수 있도록 합니다. → 이 과정을 거친 후에도 2차원 또는 3차원 배열의 형태로 데이터가 유지됩니다.
- 과적합 방지: 차원을 줄임으로써 모델이 복잡한 패턴에 과적합되는 것을 방지합니다.

2-2. 주요 Pooling 방법

Pooling은 입력 이미지의 특정 영역을 요약하는 방법으로, 가장 많이 사용되는 풀링 방식에는

최대 풀링(Max Pooling)과 **평균 풀링(Average Pooling)**이 있습니다.

1) Max Pooling (최대 풀링)

- Max Pooling은 풀링 창(필터) 내에서 가장 큰 값을 선택하는 방법
- 입력 데이터에서 가장 강한 활성화를 선택하여 중요한 특징만을 남깁니다.

작동 방식: 주석 참조
N x N 풀링 창을 사용하여 입력 데이터의 일부분을 선택.
그 안에서 가장 큰 값을 추출하여 출력 데이터에 할당.
장점: 중요한 특징을 선택하고, 작은 변화(잡음 등)에 강건하게 반응함.
사용 예시: 일반적으로 2x2 필터를 사용하며, 2칸씩 건너뛰며(pooling stride=2) 계산합니다.

2) Average Pooling (평균 풀링)

- Average Pooling은 풀링 창 내의 평균값을 계산하여 출력하는 방식
- 입력 데이터에서 각 부분의 값을 평균화하여 추출하는 방법

작동 방식:
N x N 풀링 창을 사용하여 입력 데이터의 일부분을 선택.
그 안에 있는 모든 값의 평균을 구하여 출력 데이터에 할당.
장점: 특정 값에 의존하지 않고, 모든 값을 고려하는 방식이므로 더 부드러운 특징을 추출.
사용 예시: 이미지의 세부 정보를 덜 민감하게 처리하고, 노이즈 제거 효과가 있는 경우에 사용.

3) Global Pooling (글로벌 풀링)

- Global Pooling은 특징 맵 전체에 대한 풀링을 수행하는 방법
- 하나의 특징 맵 전체에서 최대값 또는 평균값을 계산하여 단일 값으로 줄입니다.
- 주로 출력층 직전에 사용되어, 마지막 특징 맵을 고정된 크기의 벡터로 변환합니다.

Global Max Pooling: 전체 특징 맵에서 최대값을 선택.
Global Average Pooling: 전체 특징 맵에서 평균값을 계산.
장점: 데이터 크기와 상관없이 고정된 출력 차원을 제공합니다.
사용 예시: 분류 작업에서 출력층 직전에 사용하여 입력 크기와 상관없이 고정된 크기의 벡터로 변환.

2-3. Pooling 방법의 선택

- Max Pooling: 중요한 특징만 남기고 싶을 때, 이미지에서 가장 강한 특징을 추출하는 데 유리합니다. 일반적인 이미지 처리 및 CNN에서 자주 사용됩니다.
- Average Pooling: 전체적으로 평균적인 정보를 반영하여, 중요한 특징과 덜 중요한 특징을 함께 반영하고자 할 때 사용됩니다.
- Global Pooling: 고정된 크기의 출력 벡터가 필요할 때, 주로 출력층 직전에 사용됩니다.

보통 Max Pooling이 사용되며, 이는 작은 영역 내에서 최대값을 취하는 방식입니다. 이 과정을 통해 모델이 특징을 추출하는 동안 불필요한 정보(노이즈)를 줄일 수 있습니다.

파이썬 실습 참고 : CNN 모델 (합성곱.패딩..)

3. Activation Function (활성화 함수)

- 각 층에서 출력 값을 비선형으로 변환하여 모델이 복잡한 패턴을 학습할 수 있게 합니다.
- 이 단계는 CNN의 깊은 계층에서 이미지 데이터의 복잡한 패턴을 학습하는 데 필수적입니다.
- CNN에서 자주 사용하는 활성화 함수는 ReLU(Rectified Linear Unit)입니다.
- ReLU는 음수 값을 0으로 변환하고 양수는 그대로 통과시켜서 계산 속도를 높이고 학습 성능을 향상시킵니다.

■ 3. Flattening[7] (벡터화 과정)

- CNN에서 합성곱과 풀링 과정을 통해 얻어진 2차원 또는 3차원 특징 맵을 최종적으로 완전 연결층(fully connected layer) 에 전달하기 위해서는 1차원 벡터화(1D 벡터는 1차원 배열 또는 1차원 리스트) 과정이 필요합니다. 이 과정을 Flatten이라 합니다.
- CNN에서 특징 맵을 평평하게(Flatten) 만들어 벡터로 변환합니다.

→ Flatten의 동작 :Flatten은 다차원 배열(텐서)을 1차원 벡터로 변환하는 과정으로, 완전 연결층(Fully Connected Layer)은 1차원 벡터 형태의 입력만 처리할 수 있기 때문에 이 과정이 필요하다.

- Flatten은 주로 CNN의 합성곱 층(Convolutional Layer)과 완전 연결 층(Fully Connected Layer) 사이에서 사용되며, 합성곱 층을 통해 추출된 특징 맵을 1차원 벡터로 변환하여 분류 작업에 사용할 수 있도록 도와줍니다.

[Flatten의 주요 특징]

- 입력 형태 변환: 예를 들어, CNN의 마지막 합성곱 층에서 (7, 7, 64) 형태의 텐서가 출력되었다면, Flatten은 이를 길이 3136(=7×7×64)의 1차원 벡터로 변환합니다. 이 벡터는 이후의 완전연결층에 입력될 수 있습니다.
- 계층 연결 역할: Flatten은 데이터의 구조는 유지하면서 단순히 펼쳐 주기만 하므로, 학습에는 영향을 주지 않습니다. 다만, CNN의 합성곱층과 완전연결층을 연결하는 다리 역할을 합니다.
- 특징 맵의 정보 유지: Flatten은 정보의 손실 없이 텐서의 각 요소를 순차적으로 나열해 주기 때문에, 합성곱층과 풀링층을 통해 추출된 유용한 특징들이 그대로 완전연결층으로 전달됩니다.

1차원 (1D) : 스칼라 값들이 모여서 구성된 1차원 배열이며, 수학적으로는 단순한 숫자 리스트라고 할 수 있다. 데이터가 단순히 나열된 형태를 말한다. 1차원(1D) : 하나의 축(axis)만 가지는 공간이나 데이터를 나타냅니다. 컴퓨터 프로그래밍에서는 1차원 배열(array)이나 리스트(list)로 구현된다.. [1, 2, 3, 4, 5]는 1차원 배열이며, 크기(Shape)는 (5,)이다.

7) 주석 8

1차원 벡터
: 방향과 크기를 가진 단일 축의 데이터를 나타낸다.
1차원 벡터의 특징
방향과 크기를 가진다. [3,4]는 크기가 $+4^2$ =5 이다.
1차원 벡터 활용 예시
데이터 분석: 각 데이터 포인트의 특성(features)을 1차원 벡터로 표현할 수 있다.
데이터 포인트의 특징 벡터
기계 학습: 입력 데이터는 보통 1차원 벡터로 모델에 전달된다. CNN Flatten 출력
신호 처리: 시간에 따른 신호 데이터를 1차원 벡터로 표현할 수 있다.

(예시)

- 그레이스케일 이미지 경우

: 28x28 크기의 그레이스케일 이미지가 CNN을 통해 7x7x64 크기의 특징 맵을 생성했다고 가정하면, 벡터화 과정에서는 이를 7 x 7 x 64 = 3136개의 값으로 이루어진 1D 벡터로 변환합니다.

7x7x64 →1D 벡터(3136개)

- 컬러 이미지 경우

: 32x32 크기의 RGB 이미지가 CNN을 거쳐 8x8x128 크기의 특징 맵을 생성했다고 가정하면, 이 역시 8 x 8 x 128 = 8192개의 값으로 이루어진 1차원 벡터로 변환됩니다.

8x8x128 →1D 벡터(8192개)

■ 4. Fully Connected Layer (완전 연결 층)

- 합성곱 신경망(CNN)에서 주로 마지막 단계에 사용되는 신경망 층으로, 벡터화된 데이터는 이제 완전 연결층으로 전달됩니다.
- 모든 뉴런이 이전 층의 모든 뉴런과 연결된 구조입니다.
- 완전 연결층은 신경망의 마지막 부분으로, 출력층을 포함한 **전통적인 신경망의 형태**를 가집니다.
- 이미지의 고차원 특징을 입력으로 받아 최종 분류를 수행하는 층입니다.
-

{완전연결층의 주요 역할과 개념}

- 특징 조합 및 분류: CNN의 합성곱층과 풀링층을 거치며 생성된 이미지 특징을 기반으로 특정 패턴을 학습합니다. 이러한 과정을 통해 추출된 고차원의 특징 맵은 완전연결층을 통해 최종 예측이나 분류 작업을 수행하는 데 사용됩니다.

  완전연결층은 이러한 특징 맵을 일렬로 펼친(flatten) 후, 이를 입력으로 받아 특정 태스크(예: 분류, 회귀)와 관련된 패턴을 학습합니다.

즉, 추출된 특징들을 종합하여 최종 결정을 내리는 데 사용됩니다.

- 행렬 변환: 합성곱층과 풀링층의 출력인 2차원(또는 3차원) 텐서를 1차원 벡터 형태로 변환하여, 완전연결층에 전달합니다. 이 과정은 주로 Flatten 계층
-
- 을 통해 이루어지며, 이후 완전연결층은 이를 받아 다양한 가중치를 적용해 패턴을 학습합

니다.

- 계산 방식: 완전연결층에서는 이전 층의 모든 뉴런과 연결되어 있기 때문에, 일반적인 신경망처럼 가중치와 편향을 포함한 선형 결합을 통해 학습이 이루어집니다. 이를 통해 이미지를 특정 클래스(예: 개, 고양이 등)로 분류하는 확률을 출력합니다.
- 과적합 위험: 완전연결층은 많은 파라미터를 가지기 때문에 과적합(overfitting)의 위험이 있습니다. 이를 방지하기 위해 드롭아웃(Dropout) 같은 정규화 기법이 자주 사용됩니다.

마지막 계층에서의 소프트맥스: 분류 문제의 경우, 마지막 완전연결층에 **소프트맥스 활성화 함수를 적용**하여 각 클래스에 대한 확률을 계산하고, 가장 높은 확률을 가지는 클래스를 예측 결과로 선택합니다.

완전연결층은 CNN이 추출한 특징을 종합하여 최종 결과를 출력하는 결정 메커니즘"의 역할을 수행합니다.

합성곱층이 "특징 추출"에 집중한다면, 완전연결층은 이 특징을 기반으로 "결론 도출"을 담당합니다.

■ 5. CNN의 주요 장점

- 지역적인 패턴 학습: CNN은 이미지의 작은 부분(로컬 필터)을 통해 특징을 학습하므로, 이미지의 공간적 구조를 잘 파악할 수 있습니다.
- 파라미터 효율성: 필터를 공유함으로써 입력 데이터의 크기에 비해 학습해야 할 파라미터 수가 적습니다.
- Translation Invariance(변환 불변성): 이미지의 특정 패턴이 위치를 변경해도 CNN은 이를 인식할 수 있습니다.

○ 이미지 데이터의 처리와 CNN 적용 방법

1. 입력 이미지 값?

- 입력 이미지 값은 CNN 모델에 입력되는 실제 데이터입니다.
- 입력 이미지의 값은 이미지의 각 픽셀 값을 의미합니다.

2. 이미지 값 생성 과정

2.1. 이미지의 픽셀값

이미지는 보통 픽셀(Pixel)로 이루어져 있으며, 각 픽셀은 ~~색상 정보를 숫자 값으로 저~~장합니다.

1) 그레이스케일 이미지(Grayscale Image)[1] : 각 픽셀은 흑백 정보를 나타내는 단일 숫자로 표현됩니다. 보통 0(검은색)에서 255(흰색) 사이의 값으로 표현됩니다.

예: 0 = 검은색, 255 = 흰색, 중간 값들은 다양한 회색 음영을 나타냅니다.

2) 컬러 이미지((Color Image, RGB)[2] : 각 픽셀은 3개의 숫자로 표현되며, 각각 Red, Green, Blue 채널에 대한 값입니다. 각 채널의 값도 0에서 255 사이의 숫자로 표현됩니다.

2.2. 이미지 전처리 과정

CNN 모델에 입력하기 전에 이미지 데이터를 숫자 배열로 변환하는 전처리 과정을 거칩니다. 주요 과정은 다음과 같습니다:

1) 이미지 크기[8] 조정: CNN에 입력될 이미지 크기를 고정해야 하므로, 이미지를 지정된 크기로 리사이즈(Resizing, 크기 조정)[9]합니다.

예를 들어, 224x224 크기의 이미지를 사용한다면, 모든 이미지를 224x224 크기로 조정합니다.

2) 정규화(Normalization)

- 픽셀 값은 일반적으로 0~255 사이의 값을 가집니다.
- 그러나 CNN 모델이 학습할 때 숫자가 너무 크면 성능이 떨어질 수 있으므로, 이를 0과 1 사이의 값으로 정규화하는 것이 일반적입니다.
- 이를 위해 모든 픽셀 값을 255로 나누는 방식으로 정규화할 수 있습니다.

예: 만약 픽셀 값이 128이라면, 정규화된 값은 128 / 255 = 0.5019가 됩니다.

3) 배치(batch) 형성[10]: 여러 이미지를 한꺼번에 처리하기 위해 이미지 데이터가 배치 형태로 입력됩니다. 각 배치는 여러 개의 이미지로 구성된 배열로 변환됩니다.

2.3. 이미지가 CNN에 입력되는 과정 예시 : 그레이스케일 이미지

- 입력 이미지: 28x28 크기의 그레이스케일 이미지가 있다고 가정하겠습니다.
- 픽셀 값 추출: 이미지를 숫자로 변환하면, 28x28 크기의 배열이 생성됩니다. 배열의 각 요소는 이미지의 픽셀 값을 나타냅니다. 예를 들어, 아래와 같은 28x28 배열이 될 수 있습니다:

```
[[  0,  0,  0, ..., 255, 255, 255],
 [  0,  0,  0, ..., 128, 128, 128],
 [  0,  0,  0, ...,  64,  64,  64],
  ...
 [  0,  0,  0, ..., 255, 255, 255]]
```

- 정규화: 각 픽셀 값을 255로 나누면, 모든 값이 0과 1 사이로 변환됩니다.

```
[[0.0, 0.0, 0.0, ..., 1.0, 1.0, 1.0],
 [0.0, 0.0, 0.0, ..., 0.5, 0.5, 0.5],
 [0.0, 0.0, 0.0, ..., 0.25, 0.25, 0.25],
  ...
 [0.0, 0.0, 0.0, ..., 1.0, 1.0, 1.0]]
```

---

8) 주석6
9) 주석7
10) 주석8

이 배열이 CNN의 입력 이미지로 사용되며, Convolutional Layer에서 합성곱 연산을 통해 특징 맵이 생성됩니다.

2.4. **이미지가 CNN에 입력되는 과정 예시** : 컬러 이미지의 경우

- 컬러 이미지(RGB)는 각 픽셀마다 3개의 값(빨강, 초록, 파랑)을 가지며, 3개의 채널로 이루어져 있습니다.
- 따라서 컬러 이미지의 경우, CNN에 입력되는 배열은 (높이, 너비, 3) 형태로 구성됩니다.

예를 들어, 32x32 크기의 컬러 이미지는 (32, 32, 3) 배열로 변환되며, 각 채널은 개별적으로 CNN에 입력되어 필터와의 합성곱 연산을 거칩니다.

결론적으로,

- 입력 이미지의 값은 실제 이미지 데이터를 수치화한 값으로, 이미지의 픽셀 정보를 CNN 모델이 처리할 수 있도록 변환한 형태입니다.
- 이 값을 기반으로 CNN은 특징을 추출하여 학습을 진행합니다.

2.5 CNN에서 입력 이미지의 크기

- CNN에서 입력 이미지의 크기는 모델 설계 시 중요한 요소 중 하나입니다.
- 입력 이미지 크기는 모델의 성능과 계산 비용에 영향을 미치며, 일반적으로 고정된 크기로 설정되어 모델에 전달됩니다.
- CNN 모델을 만들 때, 입력 이미지의 크기는 고정된 크기로 맞춰야 합니다.
- 입력 이미지의 크기는 보통 모델이 학습할 데이터의 특성과 관련이 있습니다.

1) 데이터셋에 맞춘 크기 설정

데이터셋에 이미 고정된 크기가 있을 경우, 이를 그대로 사용합니다.

예시) MNIST 데이터셋의 경우, 손글씨 숫자 이미지는 28x28 크기의 그레이스케일 이미지입니다. 따라서 CNN의 입력 이미지 크기는 28x28로 고정됩니다.

2) 이미지 리사이즈(Resizing)

데이터셋에 포함된 이미지들이 다양한 크기일 경우, 모든 이미지를 동일한 크기로 변환하여 CNN에 입력할 수 있습니다.

예시) ImageNet 데이터셋의 이미지들은 서로 다른 크기를 가지고 있기 때문에, 일반적으로 224x224, 256x256 등의 크기로 리사이즈하여 CNN에 입력합니다.

이 과정에서 중요한 이미지는 유지되지만, 불필요한 정보를 줄여 효율적인 학습을 도모할 수 있습니다.

예시:

- CIFAR-10: 32x32 크기의 컬러 이미지로 이루어진 데이터셋.
- ImageNet: 보통 224x224 또는 256x256 크기로 조정된 컬러 이미지.
- MNIST: 28x28 크기의 그레이스케일 이미지.

2.6. 입력 이미지 크기 설정 시 고려사항

1) 크기가 클수록 더 많은 정보가 포함됨

- 큰 이미지는 더 많은 세부 정보를 포함할 수 있지만, 계산 비용이 크게 증가합니다.

(예시) 512x512 크기의 이미지를 입력으로 사용하면, 모델이 더 많은 패턴을 학습할 수 있지

만, 메모리 사용량과 계산 시간이 늘어나므로 GPU 자원 등 하드웨어의 요구 사항이 높아집니다.

- 작은 이미지는 계산 비용이 적지만, 이미지의 세부 정보가 손실될 수 있습니다.

2) 입력 이미지의 비율 유지

- 이미지를 리사이즈할 때 가로 세로 비율이 유지되는 것이 중요합니다.
- 비율을 무시하고 임의로 크기를 조정하면 이미지가 왜곡되어 모델 학습에 문제가 발생할 수 있습니다.
- 비율을 유지하면서 크기를 변경하는 경우, 패딩(padding)을 사용하여 부족한 영역을 채울 수 있습니다.

3) 그레이스케일 vs 컬러 이미지

- 그레이스케일 이미지는 1개의 채널을 가지므로 (H x W x 1) 형태로 입력됩니다.
- 컬러 이미지는 RGB 채널을 가지며, (H x W x 3) 형태로 CNN에 입력됩니다. 예시) 32x32 크기의 컬러 이미지라면, (32, 32, 3) 형태로 모델에 입력됩니다.

2.7. 입력 이미지 크기와 CNN 구조

1) 첫 번째 합성곱 층

CNN의 첫 번째 합성곱 층은 입력 이미지 크기에 따라 크기와 연산량이 달라집니다.

예시) 입력 이미지가 32x32일 경우 첫 번째 층에서는 이 크기에 맞춰 필터가 적용되며, 출력 크기는 필터 크기, 스트라이드, 패딩 설정에 따라 결정됩니다.

2) 풀링 층과 다운샘플링

CNN에서는 이미지의 크기를 점진적으로 줄여가며 중요한 특징만 남기는 방식으로 학습합니다. 이때 풀링 층(Max Pooling) 등을 사용해 이미지 크기를 줄이면서 정보 손실을 최소화합니다.

예시) 처음 224x224 크기의 이미지가 풀링 층을 거치면서 112x112, 56x56, 28x28 등의 크기로 줄어들게 됩니다.

3. 입력 이미지 크기 설정의 실제 사례

1) MNIST 데이터셋

MNIST는 28x28 크기의 흑백 이미지로 이루어져 있으므로, CNN의 입력 이미지 크기도 28x28로 설정됩니다.

입력 이미지: (28, 28, 1) (그레이스케일 이미지)

2) CIFAR-10 데이터셋

CIFAR-10은 32x32 크기의 RGB 컬러 이미지로 구성되어 있습니다.

따라서 CNN에 입력되는 이미지의 크기는 (32, 32, 3)입니다.

입력 이미지: (32, 32, 3) (컬러 이미지)

3) ImageNet 데이터셋

ImageNet은 다양한 크기의 이미지를 포함하고 있지만, 일반적으로 CNN 학습을 위해 모든 이미지를 224x224 또는 256x256 크기로 리사이즈합니다.

입력 이미지: (224, 224, 3) 또는 (256, 256, 3) (컬러 이미지)

4. 입력 이미지 크기와 모델 성능 간의 관계

입력 이미지의 크기를 너무 작게 설정하면, 중요한 세부 정보가 손실될 수 있습니다.
반대로, 너무 큰 이미지를 사용하면 모델이 복잡해져 학습 속도가 느려지고, 계산 비용이 증가할 수 있습니다.
적절한 입력 이미지 크기를 선택하는 것은 데이터의 특성, 하드웨어 자원, 모델의 복잡도에 따라 결정됩니다.

5. CNN에서 필터(커널)가 학습되는 과정과 그 결과를 사례

예시: MNIST (Modified National Institute of Standards and Technology database) 데이터셋에서 숫자 분류

https://github.com/mbornet-hl/MNIST/tree/master/IMAGES/GROUPS

목표: CNN을 이용해 28x28 크기의 흑백 손글씨 숫자 이미지(0~9)를 분류하는 모델을 만들고, 필터가 어떻게 학습되는지 살펴봅니다.

5.1. 초기 입력

입력 이미지: 28x28 크기의 흑백 이미지입니다. 이미지의 각 픽셀은 0에서 255 사이의 값을 가집니다. (0은 검은색, 255는 흰색, 그 중간 값은 회색 계열)
예를 들어, 숫자 '8'이 포함된 이미지는 아래와 같이 생길 수 있습니다.

```
[ [ 0, 0, 0, ..., 0, 0, 0],
  [ 0, 0, 128, ..., 128, 0, 0],
  [ 0, 255, 255, ..., 255, 255, 0],
  ...
  [ 0, 128, 255, ..., 255, 128, 0],
  [ 0, 0, 0, ..., 0, 0, 0]]
```

5.2. CNN 초기 설정
첫 번째 합성곱 층에서는 32개의 3x3 필터를 사용하여 이미지의 로컬 패턴을 학습합니다.
필터는 처음에 랜덤한 값으로 초기화됩니다. 이때 필터는 아무런 의미 없는 무작위 패턴입니다.
5.3. 학습 과정에서의 필터 변화
CNN 모델이 학습을 진행할 때, 필터는 역전파(backpropagation) 알고리즘에 의해 점진적으로 업데이트되며, 이미지의 유용한 패턴을 인식하도록 조정됩니다.
필터 학습 사례:
초기 필터 상태:
학습이 시작되기 전에 필터는 랜덤한 값으로 초기화되므로 이미지의 패턴을 제대로 감지하지 못합니다. 필터는 아래와 같이 랜덤한 패턴을 가질 수 있습니다.

```
[ [ 0.2, -0.3, 0.1 ],
```

[0.5, -0.1, 0.7],
[-0.6, 0.4, 0.2]]

이 상태에서는 필터가 이미지의 특정한 특징을 감지하지 못하므로 합성곱 연산 결과가 유용하지 않을 수 있습니다.
학습 중간 상태:
CNN이 여러 에포크(반복 학습)를 거치면서, 필터는 손글씨 숫자의 특징적인 부분(가장자리, 선, 둥근 부분 등)을 감지하도록 최적화됩니다.
예를 들어, 손글씨 숫자에서 수직선이나 수평선, 곡선을 감지하는 필터로 발전할 수 있습니다.
학습 중 필터가 조정된 후 다음과 같은 값을 가질 수 있습니다.

수직선 필터:
[[1, 0, -1],
[1, 0, -1],
[1, 0, -1]]
수평선 필터:
[[1, 1, 1],
[0, 0, 0],
[-1, -1, -1]]

이런 필터는 입력 이미지에서 수직선이나 수평선을 효과적으로 감지할 수 있습니다. 예를 들어, 숫자 '1'에서는 수직선 패턴을 잘 감지하고, 숫자 '7'이나 '9'에서는 수평선을 감지할 수 있습니다.
학습 후 필터 상태:
학습이 완료된 후, 필터는 특정 숫자의 특징을 인식할 수 있는 형태로 발전합니다. 예를 들어, 하나의 필터는 숫자 '8'의 곡선 부분을 감지할 수 있고, 다른 필터는 숫자 '1'의 수직선 부분을 감지할 수 있습니다.
학습 후 최적화된 필터는 다음과 같은 패턴을 가질 수 있습니다.
곡선 감지 필터:
[[0, 1, 0],
[1, -4, 1],
[0, 1, 0]]

이 필터는 둥근 패턴을 감지하도록 학습된 것입니다. 예를 들어, 숫자 '8'이나 '0'과 같은 숫자의 곡선을 감지할 수 있습니다.

5.4. 필터 적용 결과
첫 번째 합성곱 층에서 필터들이 학습된 후, 입력 이미지의 패치를 필터와 합성곱 연산을 하면 특징 맵이 생성됩니다.
각 필터는 이미지에서 특정한 특징(가장자리, 선, 곡선 등)을 감지하며, 이 특징들을 모아 CNN이 손글씨 숫자를 분류하는 데 사용됩니다.

예를 들어, 필터가 숫자 '8' 이미지에서 둥근 패턴과 선을 감지하면, 그 결과 CNN은 이 이미지를 '8'로 분류할 가능성이 높아집니다.

5.5. CNN 학습 결과

CNN이 여러 층을 거치면서 필터는 점점 더 복잡한 패턴을 학습합니다. 초기 층에서는 단순한 패턴(예: 가장자리, 선)을 학습하고, 이후 층에서는 숫자 전체의 형태를 학습하게 됩니다.

최종적으로, 학습된 필터는 손글씨 숫자의 다양한 특징을 효과적으로 감지하여 정확한 분류 결과를 제공합니다.

○ CNN을 활용한 이미지 분류 문제 실습

CNN(Convolutional Neural Network)을 활용한 이미지 분류 문제는 딥러닝에서 매우 인기 있는 응용 분야입니다. CNN은 이미지 데이터를 처리하고, 컨볼루션 연산을 통해 특징을 추출하여 분류 작업을 수행합니다. 여기서는 CNN을 이용해 간단한 MNIST 손글씨 숫자 분류 문제를 해결하는 실습을 진행하겠습니다.

1. 실습 개요

목표: CNN을 사용하여 MNIST 데이터셋에서 손글씨 숫자 이미지를 분류하는 모델을 학습하고 평가합니다.

데이터셋: MNIST 데이터셋 (28x28 픽셀, 0~9까지의 숫자를 포함하는 10개의 클래스)

모델 구성: CNN을 사용하여 입력 이미지에서 특징을 추출하고, Dense 층을 사용해 분류 작업을 수행합니다.

2. 주요 단계

▷ 필요한 라이브러리 설치 및 데이터셋 로드

▷ CNN 모델 정의

▷ 모델 컴파일 및 학습

▷ 모델 평가 및 예측

1) 라이브러리 설치 및 데이터셋 로드

먼저 필요한 라이브러리를 설치하고, MNIST 데이터셋을 로드하고 정규화하는 작업을 수행합니다.

2) CNN 모델 정의

CNN 모델은 컨볼루션 층(특징 추출)과 풀링 층(특징 선택)을 포함합니다. 그런 다음, Dense 층(완전 연결 층)으로 분류 작업을 수행합니다.

3) 모델 컴파일 및 학습

모델을 컴파일하고 손실 함수와 최적화 알고리즘을 설정한 후 학습을 시작합니다.

설명:

손실 함수: Categorical_Cross Entropy는 다중 클래스 분류 문제에 적합한 손실 함수

최적화 알고리즘: Adam은 학습 속도를 빠르게 하고, 안정적인 성능을 제공하는 최적화 알고리즘입니다.

배치 크기: 각 배치에 64개의 샘플을 사용하여 학습하며, 5번의 에포크 동안 모델을 학습니

다.
검증 데이터: 학습 데이터의 10%를 검증 데이터로 사용하여 학습 성능을 모니터링합니다.
4) 모델 평가 및 예측
학습이 완료된 모델을 테스트 데이터로 평가하고, 예측 결과를 확인합니다.

5) 모델 성능 그래프 시각화
학습 과정 중 정확도와 손실 값을 그래프로 시각화하여 성능 변화를 확인할 수 있습니다.
6) 결과 분석
테스트 정확도: 학습이 완료된 모델을 테스트 데이터에 적용해, 모델의 일반화 성능을 평가합니다.
예측 결과 시각화: 예측 결과를 이미지와 함께 출력하여 모델이 어떻게 예측하는지 시각적으로 확인할 수 있습니다.
모델 성능 그래프: 학습 과정에서의 정확도와 손실 값의 변화를 그래프로 확인하며, 과적합 여부를 분석할 수 있습니다.

○ CNN 모델 적용사례

CNN(Convolutional Neural Network) 모델은 주로 컴퓨터 비전 분야에서 큰 성공을 거두었으며, 다양한 실제 응용 사례에서 매우 중요한 역할을 하고 있습니다.

1. 이미지 분류 (Image Classification)
CNN의 가장 대표적인 사용 사례는 이미지 분류입니다. 이는 이미지가 어떤 클래스에 속하는지 예측하는 작업으로, ImageNet 대회를 통해 유명해졌습니다.
CNN은 이미지의 특징을 자동으로 추출하고, 이를 바탕으로 고양이, 개, 자동차와 같은 카테고리로 이미지를 분류하는 데 뛰어난 성능을 발휘합니다.
사례:
- ImageNet: 2012년에 AlexNet이 ImageNet 대회에서 우승하면서 CNN이 이미지 분류에서 혁신적인 성능을 보여줬습니다. 이후 VGGNet, ResNet과 같은 CNN 모델들이 등장하여 더욱 깊은 신경망 구조를 통해 이미지 분류의 정확도를 높였습니다.
- Google Photos: 구글 포토는 사진을 분석하여 자동으로 사람, 동물, 사물 등 다양한 카테고리로 사진을 분류하고, 사용자가 원하는 이미지를 쉽게 검색할 수 있도록 돕습니다.

2. 객체 탐지(Object Detection)
객체 탐지는 이미지나 비디오 내에서 특정 객체가 어디에 있는지를 찾아내는 작업입니다. CNN 기반의 **YOLO(You Only Look Once)**나 R-CNN(Regions with Convolutional Neural Networks) 모델은 이미지를 통해 객체의 위치와 클래스(사람, 자동차 등)를 동시에 예측할 수 있습니다. 이는 자율주행차, 보안 시스템, 의료 영상 분석 등에서 많이 사용됩니다.
사례:
- 자율주행차: 자율주행차는 CNN 기반의 객체 탐지 알고리즘을 사용해 도로 상황에서 자동차, 사람, 장애물 등을 감지하고, 실시간으로 운전 결정을 내립니다. 테슬라와 같은 자율주

행차 개발 기업은 CNN을 활용해 차량 주변의 사물을 감지합니다.

- 보안 시스템: CCTV 영상을 분석하여 침입자 탐지나 특정 사람의 움직임을 추적하는 데 사용됩니다. 공항, 상업용 건물 등의 보안 카메라에서 CNN을 활용한 객체 탐지 기술을 통해 위험 요소를 감지하고 경고할 수 있습니다.

3. 얼굴 인식 (Facial Recognition)

얼굴 인식(Facial Recognition) 기술은 사람의 얼굴 이미지를 분석하여 개인을 식별하는 작업입니다. CNN 모델은 사람의 얼굴에서 눈, 코, 입 등 중요한 특징을 추출하고 이를 학습하여, 사용자 인증, 보안, 감시 시스템 등에 널리 사용됩니다.

사례:

- 스마트폰 잠금 해제: 애플의 Face ID나 삼성의 얼굴 인식 기능은 CNN을 기반으로 사용자의 얼굴을 인식하여 잠금을 해제합니다. 이는 얼굴의 3D 정보를 분석하고 학습하여, 사용자만이 자신의 기기에 접근할 수 있도록 보안 기능을 제공합니다.
- 공항 및 보안 시스템: 공항에서 안면 인식 시스템을 통해 승객의 신원을 빠르게 확인하고, 보안 검색 절차를 간소화합니다. 또한, 범죄자 식별 시스템에서도 안면 인식이 중요한 역할을 하고 있습니다.

4. 의료 영상 분석 (Medical Image Analysis)

CNN은 의료 영상 분석에서 매우 중요한 역할을 하고 있으며, CT 스캔, MRI, X-ray 이미지에서 질병을 자동으로 진단하는 데 사용됩니다. 특히 암 탐지나 폐 질환 진단 등에서 CNN 모델이 탁월한 성능을 보이고 있습니다.

사례:

- 암 진단: CNN은 유방암, 폐암 등 다양한 종류의 암을 조기에 진단하는 데 사용됩니다. 병리학 이미지에서 암세포를 자동으로 탐지하고, 의사가 빠르고 정확하게 진단할 수 있도록 돕습니다.
- COVID-19 진단: CNN 모델은 흉부 X-ray 이미지를 분석하여 COVID-19와 같은 폐 질환을 진단하는 데 사용되었습니다. CNN을 통해 감염 여부를 자동으로 분석하여 의료진이 빠르게 대처할 수 있도록 합니다.

5. 자율주행 (Autonomous Driving)

CNN은 자율주행 차량에서 중요한 역할을 하며, 차량이 주변 환경을 이해하고 안전하게 주행할 수 있도록 돕습니다. CNN은 카메라를 통해 들어오는 영상 데이터를 실시간으로 분석하여 차선 인식, 신호등 감지, 보행자 감지 등을 수행합니다.

사례:

- 테슬라 자율주행: 테슬라(Tesla)는 CNN 기반의 딥러닝 모델을 사용해 차량 주변 환경을 인식하고, 차선 유지, 장애물 회피, 자율 주행 모드 등을 구현합니다. 차량 카메라에서 수집된 데이터를 CNN이 실시간으로 분석하여, 안전한 주행을 가능하게 합니다.
- 우버 자율주행: 우버(Uber)도 자율주행 차량 개발에서 CNN을 활용하여, 도로 상황을 인식하고 주변 차량 및 보행자를 감지하여 주행 경로를 실시간으로 조정하는 시스템을 구축하고 있습니다.

6. 스타일 변환 (Style Transfer)

CNN은 이미지의 스타일을 변환하는 데도 사용됩니다. **스타일 변환(Style Transfer)**은 한 이미지의 스타일(예: 화가의 그림 스타일)을 다른 이미지에 적용하는 작업으로, 예술 작품을 재창조하거나 사진을 예술적으로 변환하는 데 사용됩니다.

사례:

- 디지털 아트 생성: 구글의 DeepArt는 CNN을 사용하여 사진에 화가의 그림 스타일을 적용하는 기술입니다. 이 기술은 사진을 고흐, 피카소와 같은 유명 화가의 스타일로 변환하여, 새로운 예술 작품을 생성합니다.
- 필터 적용: Prisma와 같은 앱은 CNN을 활용해 사용자의 사진에 다양한 예술적 필터를 실시간으로 적용합니다. 이 필터는 CNN이 사진의 콘텐츠와 스타일을 분석하여 예술적인 이미지를 만듭니다.

7. 이미지 생성 (Image Generation)

CNN 기반의 생성적 적대 신경망(GAN)은 이미지 생성에서 매우 중요한 역할을 합니다.

GAN은 CNN을 이용하여 실제와 매우 유사한 가짜 이미지를 생성할 수 있으며, 이 기술은 게임 그래픽, 영화 특수 효과, 얼굴 생성 등에 사용됩니다.

사례:

- DeepFake: DeepFake는 GAN을 사용해 사람의 얼굴을 다른 얼굴로 합성하는 기술입니다. 이 기술은 영상을 조작하거나, 새로운 얼굴을 생성하는 데 사용됩니다.
- 이미지 생성: NVIDIA는 CNN 기반의 GAN을 사용하여 고해상도 얼굴 이미지를 자동으로 생성하는 데 성공했습니다. 이 모델은 실제 존재하지 않는 사람의 얼굴을 매우 자연스럽게 생성할 수 있습니다.

8. 영상 처리 및 강화 (Video Processing and Enhancement)

CNN은 영상 처리 작업에서도 매우 중요한 역할을 합니다. 노이즈 제거, 해상도 향상, 영상 복원과 같은 작업에서 CNN은 기존의 영상 처리 방법보다 더 높은 성능을 보입니다.

사례:

- 슈퍼 해상도(Super Resolution): CNN은 저해상도의 이미지를 고해상도로 변환하는 데 사용됩니다. 이를 통해 오래된 사진이나 동영상의 화질을 높이는 작업을 수행할 수 있습니다.
- 노이즈 제거: CNN 기반 필터는 사진이나 영상의 노이즈를 자동으로 제거하여 선명한 이미지를 만들 수 있습니다. 이는 의료 영상이나 위성 영상에서 매우 유용하게 사용됩니다.

(주석 용어정리)

주1. 필터(Filter)

1. 필터의 초기화

초기 값: CNN이 처음 학습을 시작할 때 필터(커널)는 일반적으로 무작위로 초기화됩니다.
이 무작위 값은 주로 평균이 0이고 분산이 작은 값으로 설정되며, 이를 통해 학습이 시작됩니다. 초기화 방법으로는 He 초기화, Xavier 초기화 등이 자주 사용됩니다.
필터는 합성곱 층에서 학습해야 할 매개변수로, 모델이 학습 데이터를 통해 점진적으로 최적의 값을 찾아가는 방식입니다.

2. 필터의 역할

- 필터는 이미지의 작은 부분(패치)을 스캔하면서 합성곱 연산(Convolution)을 수행합니다.
- 필터는 입력 이미지의 특정 패턴(예: 가장자리, 선, 모서리, 텍스처 등)을 감지하는 역할을 합니다.

예를 들어, 이미지에서 수직선을 감지하려면 필터가 특정한 수직선 패턴을 학습해야 합니다. 처음에는 필터가 랜덤한 값을 가지지만, 학습 과정에서 필터는 다양한 패턴을 인식하는 형태로 조정됩니다.

2-1. 특징 추출 (Feature Extraction)

- 필터는 CNN에서 이미지의 중요한 특징을 자동으로 추출하는 역할을 합니다.
- 이미지의 픽셀 데이터를 스캔하면서 패턴을 감지하고, 이를 특징 맵(feature map)으로 변환합니다.

예를 들어, 초기에 필터는 가장자리, 수직선, 수평선 등과 같은 저차원적인 특징을 감지하며, 이후 층에서는 복잡한 패턴이나 객체의 형태와 같은 고차원적인 특징을 학습합니다.
예시:
필터가 이미지에서 수직선을 감지할 수 있습니다. 이 필터는 수직선이 있는 이미지의 영역에서 강한 출력을 생성하고, 그렇지 않은 영역에서는 낮은 출력을 생성합니다.

2-2. 합성곱 연산 (Convolution)

필터는 이미지의 작은 영역을 슬라이딩하며, 해당 영역과 필터 간의 합성곱 연산을 수행합니다. 이때 필터의 값과 이미지의 픽셀 값을 곱한 후, 그 합을 계산하여 새로운 출력 값을 만듭니다.
이 과정을 통해 필터는 이미지의 특정 패턴이 존재하는지를 인식합니다.
필터가 패턴을 감지하면, 해당 위치에서 높은 값을 출력하여 그 패턴을 강조합니다.

3. 필터 학습 과정

CNN 모델은 역전파 알고리즘(backpropagation)을 통해 필터를 학습합니다. 학습이 진행되면서 필터의 값은 손실 함수(loss function)를 최소화하는 방향으로 업데이트됩니다. 이 과정은 다음과 같이 이루어집니다:

- 순전파(Forward Propagation): 입력 이미지가 합성곱 층을 통과할 때, 필터와 이미지의 패치에 대해 합성곱 연산을 수행합니다. 이때 필터는 이미지의 로컬 패턴을 추출하고, 그 결과는 특징 맵(feature map)으로 표현됩니다.
- 손실 계산(Loss Calculation): CNN의 마지막 출력은 예측값을 제공하며, 이 예측값과 실제 값(정답) 사이의 차이를 측정하는 손실 함수가 계산됩니다. 손실 함수는 예측 오류를 정

량화하여 모델이 학습해야 할 정도를 나타냅니다.

- 역전파(Backpropagation): 손실 함수의 값을 기반으로 각 필터의 값이 얼마나 잘못되었는지를 계산합니다. 이를 통해 필터의 가중치가 조정됩니다. 필터의 가중치 업데이트는 **기울기(Gradient)**와 **학습률(Learning Rate)**을 사용하여 수행됩니다.

가중치 업데이트: 필터의 가중치는 위의 기울기를 따라 조금씩 조정됩니다. 이 과정이 반복되면서 필터는 이미지의 중요한 패턴(예: 엣지, 질감 등)을 학습하게 됩니다.

기울기 계산(Gradient Calculation): 필터가 어떻게 업데이트될지 결정하는 것은 손실 함수의 기울기에 따라 달라집니다. 기울기는 필터 값이 손실에 미치는 영향을 나타내며, 이 값이 크면 필터가 크게 업데이트되고, 작으면 미세하게 조정됩니다.

4. 다양한 필터가 학습되는 과정

CNN에서는 여러 개의 필터가 동시에 학습됩니다. 예를 들어, 첫 번째 합성곱 층에서 32개의 필터를 사용하면, 이 층은 입력 이미지로부터 32개의 다른 패턴을 학습하게 됩니다.

각 필터는 서로 다른 패턴(예: 수직선, 수평선, 모서리, 둥근 패턴 등)을 학습합니다. 처음에는 무작위 값에서 시작하지만, 학습이 진행됨에 따라 이미지의 다양한 특징을 인식하는 필터로 변환됩니다.

일반적으로 첫 번째 층에서는 저수준 특징(가장자리, 간단한 패턴)을 학습하고, 더 깊은 층에서는 고수준 특징(객체의 형태, 복잡한 패턴)을 학습합니다.

5. 필터 크기와 수의 결정

CNN의 각 합성곱 층에서 사용하는 필터의 크기와 개수는 설계자가 지정합니다.

필터 크기: 일반적으로 3x3, 5x5, 7x7 크기의 필터가 자주 사용됩니다. 작은 필터는 세밀한 패턴을 학습하고, 큰 필터는 더 넓은 범위의 패턴을 학습할 수 있습니다.

필터 개수: 각 합성곱 층에서 필터의 개수는 특징 맵의 깊이에 영향을 미칩니다. 많은 필터를 사용할수록 더 다양한 특징을 학습할 수 있습니다.

6. 필터의 실제 예시

예를 들어, CNN이 이미지에서 사람의 얼굴을 인식하려고 한다면:

첫 번째 층의 필터는 눈, 코, 입과 같은 단순한 특징을 학습할 수 있습니다.

더 깊은 층의 필터는 얼굴 전체의 형상을 인식하는 것처럼, 고차원 특징을 학습할 수 있습니다.

7. 학습된 필터 시각화

학습이 완료된 후, 필터를 시각화할 수 있습니다. 초기에는 랜덤한 값이었던 필터가 학습 후 특정 패턴을 인식하는 형태로 변형됩니다. 예를 들어, 필터가 이미지의 엣지, 선, 질감 등 특정 패턴을 인식할 수 있도록 조정됩니다.

요약:

필터는 CNN의 합성곱 층에서 학습되는 가중치 매개변수입니다.

필터는 처음에 무작위로 초기화되지만, 학습 과정에서 이미지의 특정 패턴을 감지하도록 최적화됩니다.

필터의 가중치는 손실 함수의 값을 기반으로 업데이트되며, 이를 통해 CNN은 이미지에서 의미 있는 특징을 추출할 수 있습니다.

필터는 CNN 모델의 가장 중요한 부분 중 하나로, 각 필터가 학습된 후 이미지의 패턴을 효과적으로 감지할 수 있습니다.

8. 합성곱 연산 예시

8-1. 입력 이미지의 작은 패치:

[1, 2, 1]
[0, 1, 2]
[1, 0, 1]
필터(3x3):
[1, 0, -1]
[1, 0, -1]
[1, 0, -1]

8-2. 합성곱 연산 결과:

(1*1) + (2*0) + (1*-1) + (0*1) + (1*0) + (2*-1) + (1*1) + (0*0) + (1*-1) = 0
이 결과는 필터가 해당 패턴을 감지하지 못했음을 의미하며, 필터가 특정 패턴을 인식하지 않으면 낮은 값을 출력하게 됩니다.

8-3. 특징 맵 (Feature Map) 생성

필터가 이미지의 모든 영역에 대해 합성곱 연산을 수행하면, 결과적으로 새로운 출력 이미지가 만들어지는데, 이를 **특징 맵(feature map)**이라고 합니다. 특징 맵은 필터가 감지한 패턴의 "강도"를 나타내며, 이미지에서 해당 패턴이 어디에서 나타나는지 알려줍니다.
여러 필터가 사용되면 각 필터는 이미지의 다른 특징을 감지하고, 이를 기반으로 여러 개의 특징 맵이 생성됩니다.
예시:
첫 번째 필터는 수직선을 감지하고, 두 번째 필터는 수평선을 감지하며, 세 번째 필터는 원형 패턴을 감지할 수 있습니다. 각 필터는 서로 다른 패턴을 인식하여 고유한 특징 맵을 생성합니다.

8-4. 필터의 학습

필터는 초기에는 무작위 값으로 시작하지만, 학습 과정에서 역전파(Backpropagation) 알고리즘을 통해 최적화됩니다. 즉, CNN은 학습 데이터를 통해 필터가 특정 패턴을 잘 감지할 수 있도록 가중치를 업데이트합니다.
학습이 진행되면서 필터는 이미지의 중요한 패턴을 점점 더 잘 감지하도록 조정됩니다. 이를 통해 모델이 더 정확한 예측을 할 수 있게 됩니다.
예시:
처음에는 필터가 무작위 값으로 설정되지만, 학습이 진행되면서 특정 필터는 이미지에서 가장자리, 곡선, 텍스처 등을 잘 감지하도록 조정됩니다.

8-5. 다양한 필터의 사용

CNN에서는 여러 개의 필터를 동시에 사용하여 다양한 특징을 학습합니다. 각 필터는 서로 다른 특징을 감지하도록 설계되며, 이를 통해 CNN은 이미지를 더 깊이 이해할 수 있습니다.
예를 들어, 첫 번째 합성곱 층에서 32개의 필터를 사용한다면, 이 층은 이미지의 32가지 서로 다른 특징을 학습하게 됩니다. 이후 층에서는 더 복잡한 필터가 학습되어 더 고차원적인 패턴

을 인식하게 됩니다.

8-6. 계층적 특징 추출

CNN에서 필터는 여러 층에서 점진적으로 더 복잡한 특징을 학습합니다. 초기에 사용되는 필터는 간단한 패턴(가장자리, 선)을 학습하고, 더 깊은 층에서는 복잡한 객체나 형태를 인식하게 됩니다.

이렇게 계층적 특징 추출을 통해 CNN은 이미지를 단계적으로 이해하고, 최종적으로 객체를 인식하거나 분류할 수 있습니다.

예시:

첫 번째 층의 필터는 선, 가장자리와 같은 단순한 패턴을 감지하고, 두 번째 층의 필터는 이 정보를 바탕으로 사물의 일부 형태를 학습합니다. 마지막 층에서는 고차원적인 패턴을 학습해 객체를 인식하게 됩니다.

8-7. 차원 축소

필터는 입력 이미지의 크기를 줄이는 데에도 기여합니다. 필터를 통해 중요한 특징만 추출하고, 나머지 불필요한 정보는 무시함으로써 차원 축소가 이루어집니다.

이렇게 이미지의 크기가 줄어들면 연산량이 줄어들어 학습 속도와 효율성을 높일 수 있습니다.

주2. 입력 이미지

입력 이미지를 수치 벡터화(벡터로 변환)하는 과정은 이미지 데이터를 머신러닝 및 딥러닝 모델에서 사용할 수 있는 수치 데이터로 변환하는 것을 의미합니다. 이 과정에서 이미지는 각 픽셀의 값으로 변환되며, 이 값들은 벡터 또는 행렬로 표현됩니다. 이미지의 픽셀 값은 그레이스케일 이미지와 컬러 이미지에 따라 다르게 표현되며, 다양한 차원의 수치 데이터로 벡터화됩니다.

1. 이미지 데이터란?

1) 그레이스케일 이미지

- 각 픽셀은 밝기(0 ~ 255)로 표현되며, 2차원 행렬로 나타낼 수 있습니다.
- 값이 클수록 더 밝은 색상을 나타냅니다.
- 예: 28x28 크기의 그레이스케일 이미지는 28×28=784
- 28×28=784개의 픽셀로 이루어진 2차원 배열입니다.

2) 컬러 이미지

- : 각 픽셀은 RGB 값(Red, Green, Blue)으로 구성된 3차원 배열로, 각 색상의 채널에 대한 값이 0부터 255 사이의 값을 가집니다.
- 컬러 이미지는 3차원 배열로 표현됩니다.
- 예: 32x32 크기의 컬러 이미지는 32×32×3 픽셀로 이루어진 3차원 배열입니다.

2. 입력 이미지를 벡터로 변환하는 과정

이미지를 모델에 입력하기 전에, 이미지를 벡터 형태로 변환하여 수치적으로 표현하는 과정이 필요합니다. 이 과정은 다음 단계를 거칩니다.

1) 이미지의 픽셀 데이터를 추출

이미지는 픽셀(pixel)로 구성된 2D 배열(행렬)로 저장됩니다.
각 픽셀은 해당 이미지의 색상 또는 명도 정보를 포함하고 있으며, 이 값을 추출하여 수치 데이터로 변환합니다.

- 그레이스케일 이미지: 각 픽셀은 0에서 255 사이의 값으로 표현되며, 밝기(명도)를 나타냅니다.
- 컬러 이미지(RGB 이미지): 각 픽셀은 세 개의 값(R, G, B)으로 구성되며, 각 값도 0에서 255 사이의 숫자로 표현됩니다.

3. 이미지의 벡터화

이미지를 벡터화하려면 이미지의 픽셀 값을 1차원 벡터로 변환합니다. 이는 2D 이미지 배열을 평평하게(flatten) 만들어 1D 벡터로 변환하는 것을 의미합니다.

수치 벡터화 과정 요약

1) 픽셀 데이터 추출 : 이미지의 각 픽셀 값을 추출하여 수치 데이터로 변환.
-그레이스케일: 각 픽셀은 단일 값으로 표현됨.
-컬러 이미지: 각 픽셀은 RGB 값(3개의 값)으로 표현됨.
2) 벡터화 : 2차원 또는 3차원 이미지를 1차원 벡터로 변환.
그레이스케일: $h \times w$ 크기의 이미지는 $1 \times (h \times w)$ 의 벡터로 변환.
컬러 이미지: $h \times w \times 3$h×w×3 크기의 이미지는 $1 \times (h \times w \times 3)$ 의 벡터로 변환.
3) 정규화: 각 픽셀 값을 0에서 1 사이의 값으로 정규화하여 모델이 쉽게 학습할 수 있도록 함.
4) 배치 처리: 다수의 이미지를 배치로 묶어 학습 과정에 입력

그레이스케일 이미지:

예시) 28x28 크기의 그레이스케일 이미지가 있다고 가정하면, 이 이미지는 784개의 픽셀(28x28 = 784)로 구성됩니다.
이 2차원 배열을 벡터로 변환하면 1x784 크기의 벡터가 됩니다.

$$\text{벡터화 전} : \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mn} \end{bmatrix} \quad \rightarrow \quad \text{벡터화 후} : [p_{11}, p_{12}, \ldots, p_{1n}, p_{21}, \ldots, p_{mn}]$$

이때 각 p는 해당 위치의 픽셀 값을 의미합니다.

컬러 이미지:

컬러 이미지는 **세 개의 채널(RGB)**로 구성되어 있습니다. 예를 들어, 32x32 크기의 RGB 이미지는 32x32x3의 차원을 가집니다. 이를 벡터로 변환하면, 각 픽셀의 RGB 값이 모두 포함된 1x3072(32x32x3 = 3072) 크기의 벡터로 변환됩니다.

$$\text{벡터화 후} : [R_{11}, G_{11}, B_{11}, R_{12}, G_{12}, B_{12}, \ldots, R_{32,32}, G_{32,32}, B_{32,32}]$$

각 값은 픽셀의 빨간색(R), 초록색(G), 파란색(B) 성분을 의미합니다.

배치 처리

이미지 벡터화를 통해 변환된 데이터를 **배치(batch)**로 묶어 모델에 입력합니다. 예를 들어, 100개의 이미지를 학습할 때, 각 이미지는 벡터화된 형태로 배치 처리됩니다.

예시:

만약 그레이스케일 이미지 100개(각 이미지가 28x28 크기)를 학습 데이터로 사용한다면, 이 데이터는 100x784의 크기를 가지는 입력 데이터로 변환됩니다.

컬러 이미지 100개(각 이미지가 32x32x3 크기)를 사용한다면, 이 데이터는 100x3072의 크기를 가지는 입력 데이터로 변환됩니다.

□ 그레이스케일 이미지(Grayscale Image)

흑백 이미지로, 각 픽셀이 밝기 값만을 가지는 이미지를 말합니다. 일반적으로 그레이스케일은 흑색에서 백색까지의 단일 채널 값을 가지며, 다양한 회색 음영을 통해 이미지가 표현됩니다. 이는 컬러 이미지와 달리 **색상 정보(RGB 값)**를 포함하지 않으며, 밝기만을 기준으로 이미지가 나타납니다.

그레이스케일 이미지의 주요 특징:

1. 단일 채널

그레이스케일 이미지는 단일 채널을 가집니다. 이는 각 픽셀이 하나의 밝기 값만으로 표현된다는 뜻입니다.

각 픽셀 값은 일반적으로 0에서 255 사이의 숫자로 표현되며, 이는 **0(검정)**에서 **255(흰색)**까지의 밝기 정보를 나타냅니다.

0: 완전한 검정.

255: 완전한 흰색.

중간 값들은 다양한 회색 음영을 나타냅니다.

예시:

28x28 크기의 그레이스케일 이미지는 (28, 28, 1) 형태의 배열로 표현됩니다. 여기서 1은 단일 채널을 의미하며, 각 픽셀은 밝기 값 하나로 나타납니다.

2. 픽셀 값

그레이스케일 이미지는 각 픽셀마다 밝기 정보만을 가지고 있습니다. 이 값은 이미지의 흑백 수준을 결정하며, 일반적으로 8비트(0~255 범위)로 저장됩니다.

0에 가까울수록 검은색에 가까운 픽셀이며, 255에 가까울수록 흰색에 가까운 픽셀입니다.
예시:
css
코드 복사
[0, 50, 100, 150, 200, 255]
이 값들은 어두운 픽셀(0)에서 밝은 픽셀(255)까지의 밝기 범위를 나타냅니다.
3. 컬러 이미지와의 차이점
컬러 이미지는 각 픽셀이 RGB 채널로 이루어져 있으며, 각 채널마다 (R, G, B) 각각의 값을 가지고 있어 복잡한 색상 표현이 가능합니다.
예: RGB 이미지의 경우 (28, 28, 3) 형태로 저장되며, 여기서 3은 Red, Green, Blue의 3가지 채널을 나타냅니다.
반면, 그레이스케일 이미지는 단일 채널로, 각 픽셀마다 하나의 밝기 값만 저장되어 단순한 흑백 이미지로 표현됩니다.
4. 이미지 처리에서의 사용
간단한 이미지 처리: 그레이스케일 이미지는 색상 정보 없이 밝기 값만으로 이미지를 표현하기 때문에, 엣지 검출이나 모양 인식과 같은 간단한 이미지 처리 작업에서 사용됩니다. 이는 연산량을 줄이고 처리 속도를 높일 수 있습니다.
CNN 모델에서의 사용: CNN과 같은 딥러닝 모델에서 그레이스케일 이미지는 입력 데이터로 자주 사용됩니다. 예를 들어, MNIST 데이터셋은 28x28 크기의 그레이스케일 이미지로 이루어진 손글씨 숫자 이미지 데이터셋입니다.
예시:
MNIST 데이터셋: MNIST는 28x28 크기의 그레이스케일 이미지를 사용하여 0~9까지의 숫자를 분류하는 데이터셋입니다. 각 이미지는 단일 채널(그레이스케일)로 이루어져 있으며, (28, 28, 1) 형식으로 입력됩니다.
5. 그레이스케일 이미지로 변환
컬러 이미지를 그레이스케일 이미지로 변환하는 방법은 각 픽셀의 RGB 값을 평균 내거나, 특정 가중치를 부여하여 밝기 값을 계산하는 방식으로 이루어집니다.
일반적으로 다음과 같은 가중치를 사용하여 밝기를 계산할 수 있습니다:
mathematica
코드 복사
Grayscale = 0.299 * Red + 0.587 * Green + 0.114 * Blue
이는 인간의 시각이 녹색에 더 민감하고 파란색에 덜 민감하다는 사실을 반영한 가중치입니다.
6. 활용 사례
컴퓨터 비전: 그레이스케일 이미지는 컴퓨터 비전 작업에서 자주 사용됩니다. 특히 이미지의 색상 정보가 중요한 경우가 아닌 엣지 검출, 객체 인식, 텍스처 분석 등에서는 그레이스케일 이미지만으로도 충분한 정보를 얻을 수 있습니다.
문자 인식(OCR): 문자인식 작업에서도 그레이스케일 이미지를 주로 사용하여 텍스트를 처리합니다. 예를 들어, 문서 스캔 이미지에서 글자를 추출하는 작업에서 그레이스케일 이미지는 연산 비용을 줄이고 정확도를 높일 수 있습니다.
의료 영상 처리: 의료 영상(예: X-ray, MRI)에서 그레이스케일 이미지는 필수적입니다. 이런

이미지는 흑백으로 표현되며, 병변 탐지나 분석에 사용됩니다.
요약:
그레이스케일 이미지는 단일 채널로, 각 픽셀이 0(검정)에서 255(흰색) 사이의 밝기 값으로 표현됩니다.
그레이스케일 이미지는 RGB와 같은 색상 정보 없이 밝기만을 표현하므로, 이미지 처리에서 연산량을 줄이고 더 간단하게 처리할 수 있습니다.
딥러닝 모델에서도 자주 사용되며, 특히 컴퓨터 비전, 문자 인식, 의료 영상 처리 등에서 중요한 역할을 합니다.
그레이스케일 이미지는 간단하지만, 이미지의 패턴과 모양을 효과적으로 분석하는 데 충분한 정보를 제공합니다.

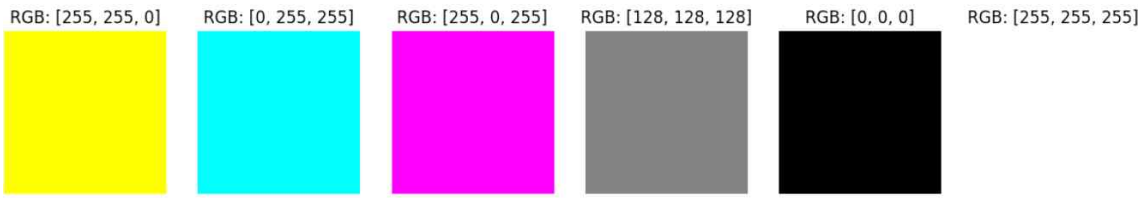
□ 컬러 이미지(Color Image)
각 픽셀이 여러 색상 정보를 포함하는 이미지로, 보통 RGB(Red, Green, Blue) 색상 모델을 사용하여 표현됩니다. 각 픽셀은 세 가지 채널(빨강, 초록, 파랑)의 값을 가지며, 이 값들의 조합으로 다양한 색상이 만들어집니다. 이러한 색상 정보 덕분에 컬러 이미지는 그레이스케일 이미지보다 더 풍부한 시각적 정보를 제공합니다.
1. RGB 색상 모델
RGB 모델은 컬러 이미지를 표현하는 가장 일반적인 방식입니다.
각 픽셀은 세 가지 색상 값(빨강, 초록, 파랑)을 가지며, 각 값은 보통 0~255 범위의 숫자로 표현됩니다.

▶ 0은 해당 색상의 강도가 전혀 없는 상태(즉, 어두움)를 의미합니다.
▶ 255는 해당 색상이 최대한 강하게 표현되는 상태(즉, 밝음)를 의미합니다.
픽셀의 RGB 값에 따라 다양한 색상을 표현할 수 있습니다. 예를 들어:

- (255, 0, 0)은 빨강이 최대, 나머지 색상이 없는 상태로 순수한 빨간색을 나타냅니다.
- (0, 255, 0)은 초록이 최대, 나머지 색상이 없는 상태로 순수한 초록색을 나타냅니다.
- (0, 0, 255)는 파랑이 최대, 나머지 색상이 없는 상태로 순수한 파란색을 나타냅니다.
- (0, 0, 0)은 모든 색상이 없는 상태로 검정색을,
- (255, 255, 255)는 모든 색상이 최대인 흰색을 나타냅니다.

이러한 세 가지 채널이 결합되어 다양한 색상을 나타내며, RGB의 비율에 따라 색상이 달라집니다.
(예시)
RGB 값 예시:

```
[255, 255, 0], # 노란색
[0, 255, 255], # 청록색
[255, 0, 255], # 자홍색
[128, 128, 128], # 회색
[0, 0, 0], # 검은색
[255, 255, 255]  # 흰색
```



파이썬 실습 파일 참조 (CNN입문 .... 참고사항)

2. 컬러 이미지의 구조
컬러 이미지는 다중 채널 이미지입니다. 즉, 각 픽셀은 세 개의 값(R, G, B)을 가지며, 이미지 전체는 세 개의 2D 배열로 이루어집니다.
예를 들어, 32x32 크기의 컬러 이미지가 있을 경우, 이 이미지는 (32, 32, 3) 형태로 표현됩니다. 여기서 32, 32는 이미지의 너비와 높이, 3은 RGB의 세 가지 채널을 나타냅니다.
예시:
코드 복사
이미지 크기: 32 x 32
RGB 채널: 3
배열 구조: (32, 32, 3)
각 채널은 이미지의 색상 정보를 포함하고 있으며, 픽셀마다 RGB 값이 저장됩니다.
3. 컬러 이미지의 픽셀 값
각 픽셀은 세 가지 색상 값으로 구성됩니다. 각각의 값은 0에서 255까지의 숫자로 표현되며, 0은 해당 색상이 없음, 255는 최대 강도를 의미합니다.
예를 들어, 빨간색 픽셀의 RGB 값은 (255, 0, 0)이며, 초록색은 (0, 255, 0), 파란색은 (0, 0, 255)로 표현됩니다.
예시:
픽셀 값: (34, 200, 100)
R = 34, G = 200, B = 100
이 픽셀은 적은 양의 빨강, 많은 양의 초록, 중간 양의 파랑이 섞인 색상을 나타냅니다.
4. 컬러 이미지의 처리
딥러닝 모델이나 이미지 처리 알고리즘은 컬러 이미지의 각 채널을 개별적으로 처리합니다.
CNN(합성곱 신경망)은 컬러 이미지를 입력으로 받을 때 각 채널(R, G, B)을 따로 처리한 후, 이 정보를 종합하여 이미지를 이해합니다.
예를 들어, 이미지의 한 부분에서 빨강 채널이 강하면, CNN은 이 부분이 빨간색일 가능성이 크다고 해석합니다.
CNN에서의 컬러 이미지 처리:
컬러 이미지의 입력 형태는 (높이, 너비, 채널)로 표현됩니다. 예를 들어, 64x64 크기의 컬러 이미지를 CNN에 입력하면 (64, 64, 3) 형식으로 입력됩니다.
각 채널(R, G, B)은 필터와 합성곱 연산을 통해 처리되며, 각 채널의 정보를 기반으로 CNN은 이미지의 특징을 학습합니다.

5. 컬러 이미지와 그레이스케일 이미지 비교
컬러 이미지는 RGB와 같은 색상 정보를 포함하며, 더 많은 시각적 정보를 제공하지만 처리 비용이 더 큽니다.
그레이스케일 이미지는 밝기 정보만을 가지는 단일 채널 이미지로, 컬러 이미지에 비해 정보가 단순하지만 계산량이 적어 처리 속도가 빠릅니다.
컬러 이미지를 그레이스케일 이미지로 변환할 때는 RGB 값을 결합하여 하나의 밝기 값으로 변환합니다. 일반적으로 가중치를 적용해 각 채널의 비율을 반영하여 밝기 값을 계산합니다:
css
코드 복사
그레이스케일 값 = 0.299 * R + 0.587 * G + 0.114 * B
이 방법은 인간의 시각이 녹색에 더 민감하다는 사실을 반영한 변환 방식입니다.
6. 컬러 이미지의 변환 및 처리
컬러 공간 변환: 컬러 이미지는 RGB 외에도 다른 색상 모델로 변환될 수 있습니다.
예를 들어, HSV(Hue, Saturation, Value)나 YUV와 같은 다른 색상 모델도 이미지 처리에 사용됩니다. RGB는 하드웨어와 호환성이 좋고 직관적이지만, 다른 색상 모델은 조명 변화에 더 강건한 특징을 가질 수 있습니다.
데이터 증강: 컬러 이미지를 처리할 때 학습 성능을 높이기 위해 데이터 증강 기법을 사용할 수 있습니다. 예를 들어, 이미지를 회전하거나 자르는 방식으로 새로운 이미지를 생성하여 학습 데이터를 늘릴 수 있습니다.
7. 컬러 이미지의 활용 사례
컴퓨터 비전: 얼굴 인식, 객체 탐지, 이미지 분류와 같은 컴퓨터 비전 작업에서 컬러 이미지는 매우 중요한 역할을 합니다. 다양한 색상 정보 덕분에 CNN은 이미지의 더 복잡한 패턴을 학습할 수 있습니다.
의료 영상: MRI, CT 스캔, 초음파 등 의료 영상에서도 컬러 이미지를 사용하여 더 많은 정보를 얻습니다. 예를 들어, 염색된 조직 샘플 이미지는 특정 색상을 기반으로 질병을 분석하는 데 사용됩니다.
영상 및 사진 처리: 비디오 편집, 필터 적용, 컬러 보정과 같은 작업에서도 컬러 이미지는 필수적입니다.
요약:
컬러 이미지는 각 픽셀이 RGB 값으로 표현되는 다중 채널 이미지입니다. 각 픽셀은 빨강, 초록, 파랑 세 가지 값으로 색상을 나타냅니다.
그레이스케일 이미지와 달리, 컬러 이미지는 더 풍부한 시각적 정보를 제공하며, 다양한 색상 조합을 표현할 수 있습니다.
CNN과 같은 모델은 컬러 이미지의 각 채널을 따로 처리하여 이미지를 이해하고 학습합니다.
컬러 이미지는 일상적인 사진이나 영상, 그리고 딥러닝 모델에서 중요한 시각 정보를 전달하는 매체로 사용됩니다.

주3. 픽셀(Pixel)

- 디지털 이미지의 가장 작은 단위로, "picture element"의 약어입니다.
- 하나의 픽셀은 이미지의 색상 또는 밝기 정보를 나타내며, 모든 디지털 이미지는 수많은

픽셀들로 이루어져 있습니다.

- 이미지를 확대하면 작은 사각형들이 보이는데, 이것이 바로 픽셀입니다.
- 픽셀은 2D 이미지에서 위치와 색상 값을 가지며, 이를 기반으로 이미지를 구성합니다.

1. 픽셀의 역할

- 이미지의 구성 요소: 디지털 이미지는 가로와 세로의 픽셀 배열로 표현됩니다. 예를 들어, 100x100 크기의 이미지는 가로 100 픽셀, 세로 100 픽셀로 구성된 2D 배열입니다. 각 픽셀은 이미지의 특정 위치에 대응하며, 그 위치에서 색상 또는 밝기를 나타냅니다.
- 색상 정보: 픽셀은 색상 정보를 가지고 있으며, 이 색상은 RGB(Red, Green, Blue)와 같은 색상 모델을 통해 표현됩니다. 각 픽셀은 적, 녹, 청의 값으로 이루어져 있으며, 이 값의 조합에 따라 다양한 색상이 나타납니다.
- 그레이스케일 이미지: 그레이스케일 이미지는 밝기만을 나타내는 픽셀들로 구성됩니다. 픽셀 값은 일반적으로 0(검은색)에서 255(흰색)까지의 값을 가집니다.
- 컬러 이미지: RGB 이미지의 경우, 각 픽셀은 R(적색), G(녹색), B(청색) 3개의 값으로 구성되며, 각각 0에서 255 사이의 값을 가집니다. 예를 들어, [255, 0, 0]은 완전한 빨간색을 나타냅니다.

2. 픽셀의 수와 해상도

해상도(Resolution)는 이미지의 픽셀 수를 나타냅니다. 해상도가 높을수록 이미지에 더 많은 픽셀이 포함되어 있고, 그만큼 더 세밀한 디테일을 표현할 수 있습니다.

예를 들어, 1920x1080 해상도의 이미지는 가로에 1920개의 픽셀, 세로에 1080개의 픽셀이 배열된 형태로, 총 2,073,600개의 픽셀로 구성됩니다.

해상도가 낮으면 이미지를 확대할 때 픽셀이 보이기 쉽고, 이미지가 거칠어 보이게 됩니다.

3. 픽셀 값

- 그레이스케일 이미지에서 하나의 픽셀은 단일 밝기 값(0~255)으로 표현됩니다. 0은 검정색, 255는 흰색, 그 중간 값들은 다양한 회색을 나타냅니다.
- 컬러 이미지의 픽셀 값은 보통 RGB 값으로 표현되며, 각 값은 0에서 255 사이의 숫자입니다. 예를 들어, [255, 0, 0]은 빨간색 픽셀을 나타내고, [0, 255, 0]은 녹색을, [0, 0, 255]는 파란색을 나타냅니다.

4. 픽셀의 구조

- 2D 배열: 픽셀은 2차원 배열의 형태로 저장됩니다. 예를 들어, 28x28 크기의 그레이스케일 이미지는 28개의 가로 픽셀과 28개의 세로 픽셀로 이루어진 2D 배열로 표현됩니다. 이러한 배열은 이미지의 좌표에 따라 픽셀 값을 배치한 것입니다.
- 컬러 이미지의 구조: RGB 이미지의 경우, 각 픽셀은 세 가지 채널(R, G, B)을 가집니다. 이미지의 각 채널은 독립적으로 존재하며, 각 채널마다 픽셀 값을 할당해 색상을 결정합니다. 즉, 하나의 이미지가 (높이, 너비, 3)의 차원을 가지게 됩니다.

5. CNN에서의 픽셀 처리

CNN(합성곱 신경망)은 이미지를 처리할 때 픽셀 데이터를 입력으로 받습니다. CNN은 픽셀 값에 필터를 적용해 합성곱 연산을 수행하며, 이를 통해 이미지의 패턴을 감지하고 특징을 추출합니다.

픽셀 값 정규화

픽셀 값은 일반적으로 0에서 255 사이의 값을 가지므로, 이를 모델이 더 잘 학습할 수 있도록 정규화하는 과정이 필요합니다. 정규화는 각 픽셀 값을 0과 1 사이의 값으로 변환하는 과정입니다.

정규화 방법: 각 픽셀 값을 255로 나누어 정규화합니다.

$$\text{정규화된 픽셀 값} = \frac{\text{픽셀 값}}{255}$$

정규화 이유: 딥러닝 모델은 큰 값을 다루기 어려우며, 0에서 1 사이의 값을 다루는 것이 학습 속도와 성능을 향상시킬 수 있기 때문에 정규화가 필요합니다.

6. 픽셀과 이미지 분석

이미지에서 각 픽셀의 값은 매우 중요한 의미를 지니며, CNN과 같은 모델은 이 값을 기반으로 이미지를 분석하고 학습합니다. CNN은 이미지에서 필터를 사용해 특정 픽셀 패턴(예: 가장자리, 모서리, 질감)을 학습합니다.

주석4. 국소 영역(Local Receptive Field)과 글로벌 영역(Global Receptive Field)

- 국소 영역(Local Receptive Field)과 글로벌 영역(Global Receptive Field)은 모델이 입력 데이터(주로 이미지)를 처리하는 방식에서의 시야 범위를 나타냅니다.
- 이를 통해 CNN이 어떻게 이미지를 이해하고 분석하는지 이해할 수 있습니다.

1. 국소 영역 (Local Receptive Field)

정의: 국소 영역은 입력 이미지의 작은 영역을 의미하며, CNN의 각 필터가 이 국소 영역에서 특징을 추출합니다.

역할: 국소 영역은 이미지의 세부적인 저수준(low-level) 특징을 인식합니다.

예시) CNN의 첫 번째 합성곱 계층에서는 작은 영역에서 모서리, 선, 텍스처와 같은 기본 패턴을 인식하는 데 사용됩니다.

특징:

- 지역적 패턴 감지: 국소 영역은 이미지의 특정 작은 부분을 인식하기 때문에 이웃 픽셀 사이의 관계를 포착하는 데 효과적입니다.
- 연산 효율성: 필터가 작은 국소 영역에서만 적용되므로 전체 이미지를 한번에 처리하는 것보다 효율적입니다.

예시: 얼굴 이미지에서 눈, 코, 입과 같은 작은 특징을 국소 영역에서 인식할 수 있습니다.

2. 글로벌 영역 (Global Receptive Field)

정의: 글로벌 영역은 네트워크가 점차 깊어지면서 확장된 시야로, 최종적으로 입력 전체를 포함하는 범위입니다.

역할: 글로벌 영역은 이미지의 큰 구조를 파악하고 고수준(high-level) 특징을 인식하는 데 사

용됩니다. CNN의 계층을 거치며 국소 영역에서의 저수준 특징이 결합되어 이미지의 전체적 맥락을 이해하게 됩니다.
특징:
전역적 이해: CNN이 최종 계층에서 이미지의 전체 맥락을 이해하며, 특정 객체가 어느 위치에 있으며, 다른 객체와 어떤 관계가 있는지를 파악할 수 있습니다.
고수준 특징 결합: 여러 국소 영역에서 추출된 저수준 특징들을 결합해 전체 이미지를 종합적으로 이해합니다.
예시: 자동차 이미지를 예로 들면, 글로벌 영역을 통해 CNN은 바퀴, 창문 등 개별적인 특징을 결합하여 전체적으로 자동차라는 객체를 인식하게 됩니다.
CNN에서 국소 영역과 글로벌 영역의 작용
초기 합성곱 계층에서는 작은 국소 영역에서 세부적인 저수준 특징을 추출합니다.
중간 계층을 거치면서 국소 영역에서 추출된 정보가 더 넓은 영역으로 통합되어 중간 수준의 특징을 포착합니다.
마지막 계층에서는 이미지의 전체적인 특징을 포착하는 글로벌 영역이 형성되어, 객체나 장면의 고수준 의미를 이해하게 됩니다.

구분	로컬 문제(Local Problem)	글로벌 문제(Global Problem)
정의	국소 정보에 한정되어 전체 맥락을 놓치는 문제	이미지의 전체적인 관계나 구성 정보를 잃는 문제
원인	작은 필터 크기 및 국소 영역만 바라보는 구조	CNN 구조의 특성상 전역 정보 결합이 제한됨
예시	자동차 바퀴는 인식하되, 그것이 전체적으로 자동차인지는 모름	복잡한 장면에서 객체 간 관계를 이해하지 못함
해결 방안	더 깊은 네트워크, 큰 필터 및 스트라이드, 멀티 스케일 적용	전역 풀링, 어텐션 메커니즘, Residual Connections 사용

CNN의 로컬 문제와 글로벌 문제를 해결하는 다양한 접근법들이 발전하고 있으며, 특히 어텐션 메커니즘과 같은 기술이 전역 정보 인식의 부족을 보완하는 데 많은 기여를 하고 있습니다.

주석5. 슬라이딩 과정 예시
입력 이미지(5x5)와 필터(3x3), 스트라이드가 1인 경우, 슬라이딩 과정은 다음과 같습니다:

입력 이미지:
1 2 3 0 1
0 1 2 3 0
1 0 1 2 1
2 1 0 1 2
1 2 3 1 0

필터(3x3):
1 0 1
0 -1 0
1 0 1

슬라이딩 단계:
첫 번째 위치 (좌측 상단):
필터는 좌측 상단에서 시작하여 입력 이미지의 첫 번째 3x3 영역과 합성곱 연산을 수행합니다.
필터가 스캔하는 영역
1 2 3
0 1 2
1 0 1
합성곱 결과: (1*1) + (2*0) + (3*1) + (0*-1) + (1*0) + (2*0) + (1*1) + (0*0) + (1*1) = 6

두 번째 위치 (오른쪽으로 한 칸 이동):
필터는 오른쪽으로 한 칸 이동하여 다음 3x3 영역과 합성곱 연산을 수행합니다.
필터가 스캔하는 영역:

2 3 0
1 2 3
0 1 2

합성곱 결과: (2*1) + (3*0) + (0*1) + (1*-1) + (2*0) + (3*0) + (0*1) + (1*0) + (2*1) = 5

계속해서 오른쪽 및 아래로 이동하면서 연산:
필터는 입력 이미지의 각 위치에서 슬라이딩하며 합성곱을 수행합니다.
전체 과정이 끝나면, 필터가 스캔한 영역에 대한 합성곱 결과로 새로운 특징 맵이 생성됩니다.

5. 출력 크기 계산 방법
출력 이미지의 크기는 입력 이미지 크기, 필터 크기, 스트라이드, 패딩에 따라 계산됩니다. 아래는 출력 크기를 계산하는 공식입니다:

출력 크기 = ((입력 크기 - 필터 크기 + 2 * 패딩) / 스트라이드) + 1

예시:
입력 이미지: 5x5
필터 크기: 3x3
스트라이드: 1
패딩: 0 (Valid 패딩)

출력 크기 = ((5 - 3 + 2 * 0) / 1) + 1 = 3

따라서 출력 크기는 3x3이 됩니다.

6. 슬라이딩의 결과

슬라이딩 과정을 통해 필터는 입력 이미지의 모든 위치에서 합성곱 연산을 수행하고, 특징 맵(feature map) 이라는 새로운 출력을 생성합니다. 이 특징 맵은 필터가 입력 이미지에서 감지한 패턴이나 특징을 나타냅니다. CNN은 이 특징 맵을 다음 층으로 전달해 더 복잡한 패턴을 학습합니다.

요약:

슬라이딩은 CNN에서 필터가 입력 이미지의 작은 영역을 차례로 스캔하며 합성곱 연산을 수행하는 과정입니다.

스트라이드는 필터가 이동하는 간격을 의미하며, 패딩은 경계 처리를 위한 방법입니다.

슬라이딩 과정을 통해 필터는 이미지에서 특징을 추출하고, 이를 바탕으로 특징 맵을 생성합니다.

CNN의 성능은 필터 크기, 스트라이드, 패딩 등의 슬라이딩 매개변수에 의해 크게 영향을 받습니다.

주석6. Stride (스트라이드)

합성곱 신경망(CNN, Convolutional Neural Network)에서 필터(커널)가 입력 이미지 위에서 얼마나 이동할지를 결정하는 중요한 파라미터입니다.

필터가 입력 이미지의 픽셀을 스캔하면서 합성곱 연산을 수행하는데, 이때 한 번의 연산 후 필터가 다음 연산으로 몇 칸 이동하는지를 스트라이드라고 합니다.

1. Stride의 정의

스트라이드는 필터가 이미지 위에서 이동하는 칸 수를 의미합니다.

일반적으로 스트라이드가 1이면 필터가 한 번에 한 칸씩 이동하고, 스트라이드가 2이면 필터가 두 칸씩 이동하면서 합성곱 연산을 수행합니다.

스트라이드는 출력 맵의 크기에 영향을 미치며, 스트라이드 값이 클수록 출력 크기는 작아집니다.

예시:

Stride = 1: 필터가 한 칸씩 이동합니다.

Stride = 2: 필터가 두 칸씩 건너뜁니다.

2. Stride의 역할

스트라이드는 합성곱 연산을 통해 이미지에서 특징을 추출할 때 다음과 같은 중요한 역할을 합니다:

1) 출력 크기 조정

스트라이드 값에 따라 출력 맵의 크기가 조정됩니다.

스트라이드가 클수록 필터가 더 많은 픽셀을 건너뛰기 때문에, 연산 횟수가 줄어들고, 그 결

과 출력 맵의 크기가 작아집니다.
스트라이드가 작으면 필터가 더 촘촘하게 이동하여 더 많은 정보를 포함한 큰 출력 맵을 생성하게 됩니다.
출력 크기 공식:
scss
코드 복사
출력 크기 = ((입력 크기 - 필터 크기 + 2 * 패딩) / 스트라이드) + 1
예시:
입력 이미지 크기: 5x5
필터 크기: 3x3
패딩: 0 (Valid Padding)
Stride = 1:
scss
코드 복사
출력 크기 = ((5 - 3 + 2 * 0) / 1) + 1 = 3x3
→ 출력 맵은 3x3 크기.
Stride = 2:
scss
코드 복사
출력 크기 = ((5 - 3 + 2 * 0) / 2) + 1 = 2x2
→ 출력 맵은 2x2 크기.
요약:
스트라이드 값이 크면 필터가 더 멀리 이동하여 적은 연산만 수행하므로, 출력 크기는 작아집니다.
스트라이드 값이 작으면 필터가 촘촘히 이동하며 더 많은 정보를 추출하게 되어, 출력 크기는 커집니다.
시각적 예시:
Stride = 1:
필터가 첫 번째 위치에서 시작하고, 한 칸씩 이동하면서 이미지 전체를 촘촘하게 스캔합니다.
Stride = 2:
필터가 첫 번째 위치에서 시작한 후, 두 칸씩 건너뛰며 연산을 수행하므로, 이미지의 작은 부분만 스캔합니다.
2) 연산량 조절
스트라이드를 늘리면 필터가 한 번에 더 많은 칸을 건너뛰기 때문에, 연산량이 줄어듭니다. 이는 학습 시간과 메모리 사용량을 줄이는 데 도움을 줍니다.
스트라이드가 클수록 필터가 더 적은 연산을 수행하므로, 모델이 더 빠르게 처리할 수 있습니다.
3) 특징 추출의 세밀함 조절
스트라이드가 작으면 필터가 더 세밀하게 이미지의 각 부분을 살펴볼 수 있습니다. 즉, 더 많은 특징을 추출할 수 있습니다.
스트라이드가 크면 이미지의 일부 세부 정보를 건너뛰게 되어, 더 큰 패턴을 감지할 수 있지

만 세부적인 특징은 놓칠 수 있습니다.
예시:
Stride = 1: 이미지의 작은 패턴(예: 가장자리, 세부적인 형태)을 세밀하게 추출합니다.
Stride = 2: 이미지의 더 큰 패턴(예: 전체 윤곽선)을 감지하지만, 세부적인 정보는 손실될 수 있습니다.
3. 스트라이드의 종류
CNN에서 스트라이드는 주로 다음과 같이 설정됩니다:
1) Stride = 1
가장 많이 사용되는 기본 값입니다. 필터가 한 칸씩 이동하면서 이미지의 거의 모든 부분을 처리합니다.
이 설정은 더 많은 정보를 추출하고, 더 큰 출력 맵을 생성합니다.
2) Stride = 2 이상
필터가 두 칸 또는 그 이상을 건너뛰며 이동하는 방식입니다. 더 적은 정보만 추출하게 되어 출력 크기가 작아지고 연산량이 줄어듭니다.
스트라이드 값이 2 이상인 경우는 주로 출력 크기를 줄이고 싶을 때 사용됩니다.
4. Stride와 패딩의 관계
스트라이드와 패딩은 함께 사용되어 출력 크기를 조정할 수 있습니다. 패딩은 입력 이미지의 가장자리에 0을 추가하여 입력 크기를 확장하는 방법입니다. 스트라이드와 패딩을 조합하면 다양한 출력 크기를 얻을 수 있습니다.
예시:
Stride = 1, 패딩 = 1 (Same Padding): 입력 크기를 유지하면서 필터를 적용할 수 있습니다. 출력 맵 크기는 입력 이미지와 동일하게 유지됩니다.
Stride = 2, 패딩 = 0 (Valid Padding): 입력 이미지의 일부만 처리하며, 출력 맵 크기가 입력보다 작아집니다.
5. 스트라이드와 풀링 층
스트라이드는 합성곱 층뿐만 아니라 풀링 층에서도 사용됩니다. Max Pooling이나 Average Pooling과 같은 풀링 연산에서도 스트라이드 값을 설정하여, 한 번에 몇 칸씩 이동할지를 결정할 수 있습니다.
풀링 층에서 스트라이드 값을 2로 설정하면, 출력 크기가 절반으로 줄어듭니다. 이는 차원 축소와 계산 효율성을 높이는 데 기여합니다.
요약:
Stride(스트라이드)는 CNN에서 필터가 입력 이미지 위에서 한 번에 몇 칸씩 이동할지를 결정하는 파라미터입니다.
스트라이드 값이 클수록 필터가 더 많이 이동하므로 출력 맵 크기는 작아지고, 연산량이 줄어듭니다.
스트라이드 값이 작을수록 필터가 더 세밀하게 이미지를 처리하여 더 많은 정보를 추출하고, 출력 크기는 커집니다.
스트라이드는 출력 크기, 연산량, 그리고 모델이 감지하는 패턴의 세밀함에 영향을 미치며, 패딩과 함께 사용해 다양한 출력 크기를 얻을 수 있습니다.
스트라이드는 CNN의 성능과 효율성에 중요한 영향을 미치는 파라미터로, 출력 크기와 학습 속도에 맞게 적절하게 설정해야 합니다.

CNN에서 입력 이미지에 필터(커널)를 곱하는 이유

CNN에서 입력 이미지에 필터(커널)를 곱하는 이유는 특정 패턴이나 특징(feature)을 감지하기 위해서입니다.
필터는 이미지의 작은 영역을 스캔하면서 합성곱 연산을 수행하여, 이미지 내의 유용한 정보(예: 가장자리, 선, 텍스처 등)를 추출하는 역할을 합니다.
이 과정은 CNN이 이미지를 이해하고 학습하는 데 매우 중요한 단계입니다.
아래에서 필터를 곱하는 이유와 그 과정에 대해 더 구체적으로 설명하겠습니다.

1. 특정 패턴 감지
필터를 곱하는 이유는 이미지의 특정 패턴을 감지하기 위함입니다.
예를 들어, 이미지의 수직선, 수평선, 엣지(가장자리)와 같은 기본적인 특징을 감지하는 것이 CNN의 중요한 목표 중 하나입니다.
필터는 작은 크기의 매트릭스로, 이미지의 특정 패턴을 강조하거나 추출하는 역할을 합니다. 이때 필터가 이미지의 각 픽셀과 곱해지는 연산이 바로 합성곱 연산(Convolution)입니다.
예시:
수직선 필터:
css
코드 복사
[1, 0, -1]
[1, 0, -1]
[1, 0, -1]
이 필터는 이미지에서 수직선을 감지하는 데 사용됩니다. 입력 이미지의 특정 영역에서 수직선과 일치하는 패턴이 있으면, 합성곱 연산을 통해 높은 값을 출력하여 그 패턴을 강조합니다.
수평선 필터:
css
코드 복사
[1, 1, 1]
[0, 0, 0]
[-1, -1, -1]
이 필터는 이미지에서 수평선을 감지하는 데 사용됩니다.

2. 특징 추출(feature extraction)
필터를 곱하는 또 다른 이유는 이미지에서 중요한 특징을 추출하기 위해서입니다. CNN은 단순히 이미지를 압축하는 것이 아니라, 중요한 패턴(특징)을 추출해 점점 더 고차원적인 특징을 학습합니다.
입력 이미지가 여러 층을 통과하면서 다양한 필터가 이미지의 저차원적 특징(엣지, 선, 모서리)을 감지하고, 이를 기반으로 더 복잡한 패턴을 학습합니다. 필터가 곱해지는 합성곱 연산은 이 특징 추출의 첫 단계입니다.

3. 합성곱 연산의 목적

합성곱 연산은 필터와 이미지의 각 영역에서 이루어집니다. 이때 필터는 입력 이미지의 각 픽셀 값과 자신의 가중치 값을 곱한 후 합하여 새로운 값을 생성합니다. 이 연산을 통해 CNN은 입력 이미지의 중요한 정보만을 추출하고 불필요한 정보를 제거하는 역할을 합니다. 이렇게 계산된 값은 **특징 맵(feature map)**으로 출력됩니다.

예시: 합성곱 연산

입력 이미지의 일부 영역:

css

코드 복사

[1, 2, 3]

[0, 1, 2]

[1, 0, 1]

필터:

css

코드 복사

[1, 0, 1]

[0, -1, 0]

[1, 0, 1]

두 행렬의 각 원소를 곱한 후 합산하면:

scss

코드 복사

(1*1) + (2*0) + (3*1) + (0*0) + (1*-1) + (2*0) + (1*1) + (0*0) + (1*1) = 6

이 결과 값은 해당 영역에서 필터가 감지한 패턴의 "강도"를 나타냅니다.

4. 필터의 학습

CNN에서 필터의 값은 학습을 통해 결정됩니다. 학습 초기에 필터는 무작위로 초기화되지만, 학습 과정에서 점진적으로 이미지에서 유용한 특징을 감지할 수 있도록 업데이트됩니다. 이 과정은 역전파(Backpropagation) 알고리즘을 통해 이루어집니다.

학습이 진행됨에 따라, 필터는 점점 더 복잡한 패턴(예: 사람의 얼굴 형태, 고양이의 귀, 자동차의 윤곽선 등)을 감지할 수 있게 됩니다.

5. 차원 축소 및 계산 효율성

CNN의 필터는 차원 축소에도 기여합니다. 입력 이미지가 매우 크면, 모든 픽셀에 대해 각각 처리하는 것은 매우 비효율적입니다. 필터를 사용하여 이미지를 스캔함으로써 중요한 정보만 남기고 불필요한 정보는 제거할 수 있습니다. 이로 인해 이미지의 크기는 점차 줄어들고, 계산량도 줄어듭니다.

필터를 곱하는 과정은 이미지의 크기를 점진적으로 줄이면서 동시에 중요한 특징을 유지하는 방법으로 CNN의 효율성을 높입니다.

6. 비선형 활성화 함수와 결합

필터를 곱하는 합성곱 연산 후에는 일반적으로 비선형 활성화 함수(예: ReLU)가 적용됩니다. 활성화 함수는 비선형성을 모델에 도입하여 CNN이 더욱 복잡한 패턴을 학습할 수 있게 합니다.

필터를 통해 특정 패턴을 감지한 후, ReLU 활성화 함수는 음수를 0으로 변환하여 중요한 특

징만 남기고 노이즈나 불필요한 정보를 제거하는 데 도움을 줍니다.
요약:
CNN에서 입력 이미지에 필터를 곱하는 이유는 이미지에서 특정 패턴이나 특징을 추출하기 위해서입니다.
필터는 이미지의 작은 영역을 스캔하면서 합성곱 연산을 통해 수직선, 수평선, 엣지 등의 기본적인 패턴을 감지합니다.
이 과정은 CNN이 이미지를 점진적으로 이해하고 중요한 특징을 학습하는 데 필수적인 단계입니다.
필터는 학습 과정에서 점차 최적화되어 더 복잡한 패턴을 감지하도록 조정되며, 이를 통해 이미지의 고차원적인 특징을 추출합니다.
필터를 곱하는 합성곱 연산 덕분에 CNN은 이미지의 중요한 정보를 추출하고, 나아가 더 복잡한 이미지 인식 작업을 수행할 수 있게 됩니다.

Padding(패딩)

CNN(Convolutional Neural Network)에서 필터가 이미지의 가장자리까지 연산을 수행할 수 있도록 입력 이미지의 경계에 값을 추가하는 방법을 의미합니다. 주로 0을 추가하지만, 다른 값도 사용할 수 있습니다. 패딩은 필터가 이미지 경계 부분에서도 특징을 추출할 수 있게 하고, 출력 맵의 크기를 조절하는 역할을 합니다.
1. Padding의 역할
경계 정보 보존: 필터가 입력 이미지의 가장자리를 처리할 때, 패딩이 없으면 필터가 이미지 경계에서 적용되는 픽셀 수가 줄어듭니다. 패딩을 사용하면 이미지의 가장자리까지 필터를 온전하게 적용할 수 있어 경계 정보를 보존할 수 있습니다.
출력 크기 조정: 패딩을 통해 입력 이미지의 크기를 인위적으로 늘릴 수 있기 때문에, 출력 특징 맵의 크기를 조절할 수 있습니다. 패딩을 사용하지 않으면 필터가 이미지 내부로만 이동하므로, 출력 크기는 입력 크기보다 작아지게 됩니다.

2. Padding의 종류
1) Valid Padding (패딩 없이 합성곱)
패딩을 전혀 사용하지 않는 방식입니다. 필터가 이미지 내부에서만 적용되고, 경계 부분에서 처리할 수 있는 정보가 줄어듭니다. 이로 인해 출력 크기는 입력 크기보다 작아집니다.
경계 부분을 무시하고, 필터가 온전히 적용될 수 있는 영역에서만 연산을 수행합니다.
예시:
입력 이미지 크기: 32x32
필터 크기: 3x3
스트라이드: 1
Valid Padding 사용 시, 패딩은 0이므로 경계 부분을 처리하지 않고, 필터가 완전히 이미지 내부에서만 연산을 수행합니다.
출력 이미지 크기: 30x30 (입력 크기보다 작아짐)

Valid Padding은 출력 크기를 줄이거나, 패딩이 필요 없는 상황에서 사용됩니다.

출력 크기 계산

$$\text{Output Size} = \left(\frac{\text{Input Size} - \text{Filter Size}}{\text{Stride}} \right) + 1$$

- Input Size: 입력 이미지의 크기
- Filter Size: 필터의 크기
- Stride: 필터가 이동하는 간격

2) Same Padding

출력 맵의 크기를 입력 이미지의 크기와 동일하게 유지하기 위해 패딩을 추가하는 방식입니다. 이때 필터가 이미지 경계를 벗어나지 않고도 모든 영역을 처리할 수 있게, 이미지의 가장자리에 0(또는 다른 값)을 추가하여 크기를 조정합니다.
패딩의 양은 필터 크기에 따라 자동으로 계산되며, 출력 크기가 입력 크기와 동일합니다.

▶ Zero-padding

정의: 입력의 가장자리에 0으로 채워진 픽셀을 추가하는 방식입니다. "패딩을 한다"는 표현으로 흔히 쓰이며, 입력 이미지의 크기를 키우는 효과를 냅니다.
목적: 패딩을 통해 출력 크기를 입력 크기와 동일하게 유지하거나, 너무 급격하게 축소되지 않도록 할 수 있습니다.
출력 크기: Zero-padding을 적용하면 출력 크기가 커지며, Convolution 레이어를 여러 번 거칠 때 정보 손실을 줄일 수 있습니다.

예시:
입력 이미지 크기: 32x32
필터 크기: 3x3
스트라이드: 1
Same Padding 사용 시, 패딩은 1픽셀씩 추가됩니다.
출력 이미지 크기: 32x32 (입력 크기와 동일)
Same Padding은 출력 맵 크기를 일정하게 유지하려는 경우에 주로 사용됩니다.

출력 크기 계산 (Same Padding):

$$\text{Output Size} = \left(\frac{\text{Input Size} - \text{Filter Size} + 2 \times P}{\text{Stride}} \right) + 1$$

여기서 P는 패딩 크기입니다.

구분	Zero-padding	Valid 패딩
패딩 유무	가장자리에 0을 추가	패딩 없음
출력 크기	입력 크기와 비슷하거나 더 큼	입력 크기보다 작아짐
적용 목적	가장자리 정보 포함, 입력 크기 유지	정보 추출에 집중하며 크기 축소
계산 효율성	약간 추가적인 계산 필요	연산량 감소

3. Padding 계산 공식
패딩의 크기는 필터 크기와 스트라이드에 따라 결정됩니다. 패딩이 필요한 경우는 입력 이미지의 크기와 출력 크기를 맞추거나, 필터가 이미지 경계 부분까지 연산을 수행할 수 있게 하는 경우입니다.
패딩 크기 계산 공식:
scss
코드 복사
패딩(P) = ((출력 크기 - 1) * 스트라이드 + 필터 크기 - 입력 크기) / 2
이 공식에 따라 패딩을 적용하여 입력 크기를 조정하게 됩니다.

4. Padding의 필요성
패딩은 CNN에서 다음과 같은 이유로 필요합니다:
1) 출력 크기 유지
필터가 합성곱 연산을 수행할 때, 패딩이 없으면 필터가 경계 부분에서 처리할 수 있는 영역이 줄어들기 때문에 출력 맵의 크기가 입력 맵보다 작아집니다.
Same Padding을 사용하면 출력 크기를 입력 크기와 동일하게 유지할 수 있습니다. 이는 특히 **전이 학습(Transfer Learning)**에서 사전 학습된 모델과 동일한 입력 크기를 맞추고자 할 때 중요합니다.
2) 경계 정보 보존
패딩을 사용하지 않으면, 이미지의 경계 부분에 있는 특징이 무시되거나 덜 처리될 수 있습니다. 패딩은 경계 근처의 특징도 동일하게 처리할 수 있게 하여, 중요한 정보가 손실되지 않도록 합니다.
3) 차원 축소 방지
CNN에서 여러 층을 거칠 때, 패딩이 없으면 차원이 점차 축소됩니다. 이를 방지하기 위해 패딩을 사용해 입력 크기를 유지하면, 신경망이 더 많은 특징을 학습할 수 있습니다.
5. Padding과 Stride의 관계
패딩과 스트라이드는 CNN에서 출력 크기에 영향을 주는 두 가지 중요한 요소입니다.
Stride가 크면 필터가 한 번에 많은 픽셀을 건너뛰기 때문에 출력 크기가 작아지고, 패딩을 사용하면 입력 크기를 유지하거나 출력 크기를 크게 할 수 있습니다.
Stride와 패딩을 적절히 조합하면 출력 맵의 크기를 유연하게 조절할 수 있습니다.
예시:
입력 이미지 크기: 5x5

필터 크기: 3x3
스트라이드: 1
Valid Padding (패딩 없음):
scss
코드 복사
출력 크기 = ((5 - 3 + 2 * 0) / 1) + 1 = 3x3
출력 크기는 3x3으로 줄어듭니다.
Same Padding (패딩 1):
scss
코드 복사
출력 크기 = ((5 - 3 + 2 * 1) / 1) + 1 = 5x5
출력 크기는 입력 크기와 동일하게 5x5로 유지됩니다.

Pooling 사례

1. 2×2 Max Pooling

사용 예시: 일반적으로 2x2 필터를 사용하며, 2칸씩 건너뛰며(pooling stride=2) 계산합니다.

입력:

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Max Pooling 결과:

$$\begin{bmatrix} 6 & 8 \\ 14 & 16 \end{bmatrix}$$

2. 2×2 Average Pooling

입력:

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Average Pooling 결과:

$$\begin{bmatrix} 3.75 & 5.25 \\ 11.25 & 12.75 \end{bmatrix}$$

주7. 이미지 크기

이미지 크기는 픽셀 단위로 측정되며, 다양한 분야와 용도에 따라 여러 표준 크기가 사용됩니다.

6.1. 소셜 미디어 이미지 크기

소셜 미디어 플랫폼에서는 이미지가 특정한 크기나 비율로 업로드되도록 요구합니다.
이러한 표준 크기는 플랫폼별로 다르며, 사용자 인터페이스에 맞추기 위해 지정됩니다.
예시:

- Facebook 프로필 사진: 180x180 픽셀
- Instagram 포스트: 1080x1080 픽셀 (정사각형), 1080x1350 픽셀 (세로), 1080x566 픽셀 (가로)
- Twitter 헤더 이미지: 1500x500 픽셀
- LinkedIn 배너 이미지: 1584x396 픽셀

6.2. 웹 디자인 이미지 크기

- 웹사이트 디자인에서 이미지 크기는 페이지 로딩 속도, 해상도, 레이아웃을 고려하여 결정됩니다.
- 일반적으로 고해상도 이미지를 사용하면 더 많은 데이터 전송이 필요하기 때문에, 크기와 용량을 최적화하는 것이 중요합니다.

예시:

- 웹페이지 배경 이미지: 1920x1080 픽셀 (전체 화면)
- 웹사이트 로고: 250x100 픽셀 (웹사이트마다 다를 수 있음)
- 썸네일 이미지: 150x150 픽셀

6.3. 디지털 카메라 이미지 크기

- 디지털 카메라는 다양한 해상도로 이미지를 촬영할 수 있으며, 해상도는 보통 메가픽셀(MP) 단위로 표시됩니다.
- 해상도가 높을수록 더 많은 픽셀을 포함하므로, 더 높은 품질의 이미지를 제공합니다.

예시:

- 스마트폰 카메라: 12MP (4032x3024 픽셀) 정도의 해상도
- DSLR 카메라: 24MP (6000x4000 픽셀) 이상의 고해상도 이미지

6.4. 표준 디스플레이 해상도

- 디지털 화면에서 이미지를 표시할 때 디스플레이 해상도가 중요합니다.
- 해상도는 화면의 크기와 픽셀 밀도에 따라 다르며, 다양한 기기에서 자주 사용되는 해상도는 다음과 같습니다.

예시:

- HD (High Definition): 1280x720 픽셀
- Full HD: 1920x1080 픽셀
- 2K 해상도: 2560x1440

주8. 이미지 크기 조정(Resizing)

- 이미지 크기 조정(Resizing)은 딥러닝 모델, 특히 CNN과 같은 모델을 학습할 때 필수적인 과정입니다.
- 이미지의 크기를 일정하게 맞추는 것은 학습 속도를 향상시키고 모델이 효율적으로 학습할 수 있도록 돕습니다.
-

[이미지 크기 조정이 필요한 주요 이유]

1. 입력 크기의 일관성

- CNN과 같은 딥러닝 모델은 고정된 입력 크기를 요구합니다. 즉, 모든 입력 데이터(이미지)는 동일한 크기여야 모델이 처리할 수 있습니다.
- 만약 학습 데이터셋의 이미지들이 서로 다른 크기를 가지고 있다면, 이를 모델에 바로 입력할 수 없습니다. 따라서 모든 이미지를 일정한 크기로 변환하여 입력 크기를 일관되게 맞춰야 합니다.

예시:

모델이 224x224 크기의 이미지를 입력으로 받도록 설계되었다면, 모든 이미지를 이 크기로 맞추어야 합니다. 데이터셋의 이미지를 224x224 크기로 리사이즈(크기 조정)하는 과정이 필요합니다.

2. 메모리 및 계산 효율성

- 이미지 크기가 너무 크면 학습에 많은 메모리와 계산 자원이 필요하게 됩니다.
- 이를 방지하기 위해 이미지를 적절한 크기로 줄이면, 메모리 사용량을 줄이고 계산 효율성을 높일 수 있습니다.
- GPU와 같은 하드웨어에서 메모리 제한이 있는 경우, 작은 크기의 이미지를 사용하는 것이 유리합니다.

예시:

1024x1024 크기의 이미지를 사용하면 메모리 사용량과 계산 비용이 크게 증가하지만, 이를

224x224 또는 256x256 크기로 줄이면 훨씬 더 적은 메모리와 계산 자원으로 학습을 수행할 수 있습니다.

3. 고차원의 데이터 문제 해결

- 이미지 크기가 커질수록 데이터의 차원이 높아지며 모델이 처리해야 할 데이터의 양도 커집니다. 이는 학습 속도를 느리게 하고, 모델의 성능을 저하시킬 수 있습니다.
- 이미지의 크기를 줄이면 데이터 차원을 축소할 수 있어, 모델이 더 효율적으로 학습할 수 있습니다.

4. 모델의 학습 속도 개선

- 크기가 큰 이미지는 처리 속도가 느려질 수 있습니다.
- 이미지 크기를 줄이면 연산량이 줄어들고 모델의 학습 속도가 개선됩니다.

예를 들어, 이미지의 크기를 절반으로 줄이면, 모델이 처리해야 할 픽셀 수는 4배로 줄어들게 됩니다. 따라서 계산량이 크게 줄어들어 학습 속도가 빨라집니다.

5. 데이터 정규화

이미지 크기가 일정하지 않으면, 모델이 학습할 때 특징의 일관성을 유지하기 어렵습니다. 이미지 크기를 정규화하여 모델이 일관된 입력을 받도록 하면, 모델이 더 쉽게 학습할 수 있습니다.

예를 들어, 사람 얼굴을 인식하는 모델에서 얼굴 크기가 매우 작은 이미지와 매우 큰 이미지가 섞여 있으면 모델이 패턴을 인식하기 어렵습니다. 크기를 동일하게 맞추면 모델이 학습하기 쉬워집니다.

6. 전이 학습(Transfer Learning)에서의 요구

전이 학습을 사용할 때, 이미 사전 학습된 모델들은 특정 크기의 이미지를 입력으로 받도록 설계되어 있습니다. 예를 들어, VGGNet, ResNet과 같은 사전 학습된 모델은 보통 224x224 크기의 이미지를 사용합니다.

따라서 전이 학습을 활용하려면, 데이터셋의 이미지를 사전 학습된 모델의 입력 크기에 맞추어야 합니다.

예시:

VGG16과 같은 사전 학습된 모델은 224x224 크기의 이미지를 입력으로 받습니다. 다른 크기의 이미지를 사용하려면 반드시 리사이즈 과정을 거쳐야 합니다.

7. 비율 유지 및 왜곡 방지

- 이미지를 리사이즈할 때 가로세로 비율(aspect ratio)을 유지하는 것이 중요합니다.
- 비율을 유지하지 않고 이미지를 리사이즈하면 왜곡이 발생하여 이미지의 중요한 정보가 손실될 수 있습니다.
- 비율을 유지하면서 이미지를 변환하려면 빈 공간을 패딩(padding)으로 채우는 방법을 사용할 수 있습니다.
- 이 방식은 이미지의 크기를 조정하더라도 왜곡이 발생하지 않도록 도와줍니다.

예시:

640x480 크기의 이미지를 224x224로 리사이즈할 때, 비율을 유지하면 224x168으로 리사이즈한 후 상하단에 패딩을 추가하여 224x224로 만들 수 있습니다.

8. 데이터 증강(Data Augmentation)에서의 이미지 크기 조정

데이터 증강 과정에서 이미지 크기를 변경하는 것은 모델의 일반화 성능을 향상시키는 데 도움을 줍니다. 크기 조정을 통해 모델이 다양한 크기의 이미지에서 동일한 객체를 인식할 수

있도록 학습시킬 수 있습니다.
예를 들어, 이미지의 크기를 랜덤하게 변경하거나 확대, 축소하는 방식으로 데이터를 증강할 수 있습니다.
9. 특징 추출의 일관성 유지
CNN과 같은 딥러닝 모델은 이미지에서 특징 맵(feature map)을 추출하는 과정에서 입력 크기에 따라 달라진 결과를 도출할 수 있습니다.
이미지의 크기를 일정하게 유지하면, 추출된 특징이 일관성을 갖게 되어 모델의 성능이 개선될 수 있습니다.

주석9. 배치(batch) 형성

- 배치(batch) 형성은 딥러닝 모델을 훈련할 때 데이터를 일정한 크기로 묶어 한 번에 처리하는 기법을 말합니다.
- CNN과 같은 모델을 학습할 때, 전체 데이터를 한꺼번에 처리하지 않고, 여러 개의 데이터 샘플을 묶어 배치(batch) 단위로 처리함으로써 학습 속도와 효율성을 높입니다.
- 이 과정은 배치 학습(Batch Learning)의 핵심이며, 배치의 크기를 설정하는 것이 중요한 요소입니다.

8.1. 배치(batch)의 정의
배치란, 학습 과정에서 한 번에 모델에 입력되는 데이터 샘플들의 묶음입니다.
예를 들어, 이미지 분류 모델을 학습할 때 100개의 이미지를 동시에 처리하려면, 이 100개의 이미지를 하나의 배치로 묶습니다. 이때 100개는 배치 크기(batch size)라고 부릅니다.
전체 학습 데이터셋을 여러 배치로 나누어 학습하게 됩니다.
8.2. 배치의 역할
딥러닝 모델을 학습할 때, 배치를 사용하면 메모리 효율성을 높이고, 학습 속도를 개선할 수 있습니다.
특히 대용량 데이터셋을 사용할 때 전체 데이터를 한꺼번에 처리하는 것은 비효율적이거나 불가능할 수 있기 때문에, 작은 배치로 나누어 학습합니다.
주요 역할:
- 메모리 효율성: 데이터가 너무 많아 한 번에 처리할 수 없을 때, 배치로 나누어 학습하면 메모리 사용량을 조절할 수 있습니다.
- 학습 안정성: 한 번에 모든 데이터를 처리하는 대신, 작은 단위로 모델을 업데이트하면서 학습할 수 있어 모델이 더 빨리 수렴하도록 도울 수 있습니다.
- 병렬 처리: GPU와 같은 하드웨어를 활용해 여러 데이터를 동시에 처리할 수 있으므로, 연산 효율성이 높아집니다.

8.3. 배치 크기(batch size)의 설정
배치 크기는 학습 시 한 번에 모델에 입력되는 데이터 샘플의 개수입니다.
배치 크기는 매우 중요하며, 학습 속도, 메모리 사용량, 모델의 성능에 영향을 미칩니다.
배치 크기 선택:
- 작은 배치 크기: 예를 들어, 배치 크기를 16 또는 32로 설정하면, 모델은 적은 양의 데이

터로 여러 번 학습하면서 빠르게 업데이트할 수 있습니다. 그러나 작은 배치는 학습 과정의 변동성이 클 수 있습니다.

- 큰 배치 크기: 배치 크기가 128, 256 또는 더 클 경우, 모델은 한 번에 더 많은 데이터를 처리하므로 업데이트가 더 안정적이지만, 학습 속도는 느려질 수 있습니다. 또한 더 많은 메모리가 필요합니다.

배치 크기 선택에 대한 고려사항:

- 메모리 제한: 배치 크기가 크면 많은 데이터를 한 번에 처리하기 때문에 GPU 메모리 사용량이 증가합니다. 따라서 GPU 메모리에 맞춰 배치 크기를 조정해야 합니다.
- 속도와 성능의 균형: 큰 배치 크기는 학습이 더 안정적이지만, 작은 배치 크기는 업데이트가 더 자주 일어나므로 빠르게 최적화될 수 있습니다. 그러나 작은 배치는 학습 과정에서 변동이 커질 수 있으므로, 적절한 균형이 필요합니다.

8.4. 배치 학습의 종류

- 미니 배치 학습(Mini-Batch Learning): 전체 데이터셋을 여러 작은 배치로 나누어 학습하는 방법입니다. 이 방식이 가장 일반적이며, 딥러닝 모델에서 가장 널리 사용됩니다.
- 전체 배치 학습(Batch Learning): 전체 데이터를 한 번에 모델에 입력하여 학습하는 방식입니다. 대용량 데이터셋에서는 메모리 문제로 인해 거의 사용되지 않습니다.
- 온라인 학습(Stochastic Gradient Descent, SGD): 한 번에 하나의 데이터를 사용하여 학습하는 방식입니다. 이는 학습이 매우 불안정할 수 있지만, 실시간 데이터 처리에서 유용할 수 있습니다.

8.5. 배치 학습의 과정

학습 데이터셋을 배치로 나누어 학습하는 과정은 다음과 같습니다:

1) 데이터셋 분할

: 전체 학습 데이터셋을 미리 지정된 배치 크기만큼 나누어 여러 개의 배치로 만듭니다.

2) 각 배치마다 순전파(Forward Propagation): 한 번의 학습에서 각 배치에 있는 데이터들을 모델에 입력해 순전파를 수행하여 예측값을 계산합니다.

3) 손실 계산(Loss Calculation): 각 배치에 대한 손실 함수 값을 계산합니다.

4) 역전파(Backpropagation): 손실을 줄이기 위해 역전파 알고리즘을 적용하여 모델의 가중치를 업데이트합니다.

5) 다음 배치 처리: 다음 배치를 가져와 위 과정을 반복합니다.

8.6. 에포크(epoch)와 배치(batch)의 관계

에포크(epoch): 하나의 에포크는 전체 데이터셋을 한 번 모두 학습한 것을 의미합니다.

데이터셋을 여러 개의 배치로 나누어 학습하기 때문에, 한 번의 에포크가 끝나려면 여러 배치를 처리해야 합니다.

예를 들어, 1,000개의 데이터가 있고 배치 크기를 100으로 설정한 경우, 한 에포크는 10번의 배치 처리를 통해 완성됩니다.

8.7. CNN에서 배치 형성

CNN 모델을 훈련할 때는 입력 이미지 데이터를 배치로 묶어 한 번에 처리합니다. 예를 들어, CIFAR-10과 같은 데이터셋에서는 32x32 크기의 이미지를 사용하며, 이 이미지들을 배치로 묶어 처리하게 됩니다.

CNN에서 배치 형성 과정 예시:
CIFAR-10 데이터셋에는 60,000개의 32x32 크기의 컬러 이미지가 있습니다.
배치 크기를 64로 설정하면, 모델은 한 번의 학습에서 64개의 이미지를 동시에 처리하게 됩니다.
이 배치의 크기는 (64, 32, 32, 3) 형태가 되며, 여기서:
64: 배치 크기 (한 번에 처리할 이미지 수)
32, 32: 각 이미지의 너비와 높이
3: 각 이미지의 RGB 채널 (컬러 이미지)

주석10. flatten

1. flatten의 역할
CNN에서 합성곱 층과 풀링 층은 입력 이미지로부터 다차원 특징 맵을 생성합니다. 이 특징 맵은 2차원 또는 3차원 배열의 형태로 유지되며, 이를 완전 연결 층(Fully Connected Layer)에 전달하기 위해 Flatten을 사용하여 1차원 벡터로 변환해야 합니다.
완전 연결 층은 입력을 1차원 형태로 받아야 하므로, 합성곱 층에서 추출된 다차원 데이터를 Flatten을 통해 변환한 후, 최종적으로 분류하거나 회귀 작업에 사용할 수 있습니다.
(예시)
입력 이미지 크기: 32x32x3 (RGB 이미지)
합성곱 층 출력: 8x8x64 (64개의 8x8 크기의 특징 맵)
Flatten 결과: 8 * 8 * 64 = 4096 (1차원 벡터로 변환)
이 1차원 벡터는 이후 완전 연결 층에 전달되어 각 뉴런과 연결된 가중치와 함께 분류 또는 회귀 작업을 수행하게 됩니다.
2. Flatten의 필요성
CNN은 합성곱 층과 풀링 층을 통해 이미지에서 특징 맵(feature map)을 추출한 후, 이러한 특징을 기반으로 분류(classification) 또는 예측(prediction)을 수행합니다.
분류 작업을 수행하는 마지막 단계에서, 완전 연결 층에 데이터를 전달하려면 1차원 데이터가 필요하므로 Flatten을 사용하여 다차원 데이터를 변환합니다.
Flatten의 주요 역할:
다차원에서 1차원으로 변환: 합성곱 층과 풀링 층을 통해 나온 2D 또는 3D 배열을 1차원 벡터로 변환합니다.
완전 연결 층으로 연결: 변환된 1차원 벡터는 완전 연결 층에서 학습되어 분류나 회귀 작업에 사용됩니다.
3. Flatten의 작동 방식
Flatten은 단순히 다차원 배열의 값을 일렬로 펼쳐서 1차원 배열로 변환합니다.
Flatten은 실제로 데이터를 변환하거나 바꾸지 않고, 그저 배열의 형태를 바꿔주기만 합니다.
예시:
입력 배열:
[[[1, 2], [3, 4]],
 [[5, 6], [7, 8]]]

Flatten 적용:
[1, 2, 3, 4, 5, 6, 7, 8]
4. Flatten이 적용되는 단계
Flatten은 주로 CNN의 마지막 합성곱 층과 완전 연결 층 사이에서 적용됩니다.
합성곱 층을 거쳐 이미지에서 특징을 추출한 후, 이 특징을 Flatten을 통해 1차원 벡터로 변환하여 완전 연결 층에 전달합니다.

예시로 본 CNN의 흐름:
입력 이미지: 32x32 크기의 RGB 이미지 (32x32x3).
합성곱 층과 풀링 층: 이미지로부터 특징을 추출하여 다차원 배열(예: 8x8x64)을 생성.
Flatten 층: 8x8x64를 1차원 벡터로 변환 (4096차원).
완전 연결 층: 변환된 1차원 벡터를 사용하여 예측을 수행 (예: 10개의 클래스에 대한 확률 출력).
5. Flatten이 사용되는 이유
CNN에서 다차원 특징 맵을 그대로 분류 단계에서 사용하기 어렵기 때문에, 이를 1차원 형태로 변환하는 것이 필요합니다.
완전 연결 층은 각각의 뉴런이 모든 입력 노드에 연결되기 때문에, 입력이 반드시 1차원 벡터여야만 합니다. 이를 위해 Flatten이 필요한 것입니다.
Flatten을 사용하는 이유:
합성곱 층에서 추출한 특징을 활용: 합성곱 층에서 추출한 패턴, 모양 등의 정보를 최종적으로 완전 연결 층에서 사용하려면 1차원 벡터로 변환해야 합니다.
분류 및 예측에 사용: 변환된 1차원 벡터는 클래스 예측 또는 회귀 작업에서 사용될 수 있습니다.
6. Flatten 이후의 과정
Flatten으로 1차원 벡터가 완성되면, 그 다음으로는 완전 연결 층에 전달되어 분류 작업을 수행하게 됩니다. 완전 연결 층은 이 벡터를 입력으로 받아 각각의 뉴런과 가중치를 연결하고, 마지막에 Softmax 또는 Sigmoid와 같은 활성화 함수가 적용되어 각 클래스에 대한 확률을 출력합니다.

(파이썬 실습)

파이썬 실습 참고 : CNN 모델 입문(합성곱..패딩 .)

https://github.com/freejyb/deeplearning/blob/main/CNN%20%EB%AA%A8%EB%8D%B8%EC%9E%85%EB%AC%B8(%ED%95%A9%EC%84%B1%EA%B3%B1%2C%ED%8C%A8%EB%94%A9%2C%ED%92%80%EB%A7%81..).ipynb

CNN 모델

https://github.com/freejyb/deeplearning/blob/main/CNN(%ED%95%A9%EC%84%B1%EA%B3%B1%20%EC%8B%A0%EA%B2%BD%EB%A7%9D).ipynb

---