AI Programming

Machine learning / Deep Learning

05. CNN, RNN

first coding

CNN (Convolutional Neural Network) 개념과 원리

CNN (Convolutional Neural Network) 개념과 원리

- **기초 딥러닝의 문제점**

  - 복잡한 데이터 처리 한계문제
    - 초기 신경망은 입력 데이터가 고정된 크기와 구조를 가져야만 했음
    - 이미지, 텍스트, 시계열 데이터와 같은 복잡한 데이터에 대한 처리 능력이 부족
    - 예: 이미지의 위치 불변성(Spatial Invariance)이나 순서 정보(Time Dependency)를 학습하기 어려움.

  - 비선형 패턴 학습의 부족문제
    - 단순한 모델에서는 비선형 관계나 복잡한 패턴을 학습하기 어려웠음
    - 은닉층이 부족하거나 활성화 함수가 비효율적일 경우, 모델의 표현력(Representation Power)이 제한

CNN (Convolutional Neural Network) 개념과 원리

- **기초 딥러닝의 문제점**

  - 과적합(Overfitting) 문제
    - 복잡한 문제를 해결하기 위해 뉴런 수를 늘리면, 훈련 데이터에 지나치게 적합하여 일반화 성능이 떨어짐
    - 예: 학습 데이터에서는 높은 정확도를 보이지만 새로운 데이터에서는 성능이 저하됨.

  - 기울기 소멸(Vanishing Gradient) 문제
    - 역전파 알고리즘에서 기울기가 점차 작아져, 초깊은 신경망의 가중치 업데이트가 어려워졌음
    - 주로 Sigmoid와 같은 활성화 함수에서 발생하며, 모델 학습을 방해

  - 연산량 문제문제
    - 계산량과 메모리 사용량이 매우 큼 → 학습 속도가 느려지고, 모델 학습이 어려워 짐

CNN (Convolutional Neural Network) 개념과 원리

- 문제를 해결하기 위해 등장한 CNN, RNN
  - **합성곱 신경망(CNN: Convolutional Neural Network)**등장
    - CNN은 이미지 데이터의 공간적 패턴을 효율적으로 처리하기 위해 설계
    - 단순 신경망(ANN)으로는 이미지의 구조적 특징(예: 선, 모서리)을 학습하기 어려웠음.
    - 개선된 점
      - 지역 연결(Local Connectivity)
        - » 이미지의 일부 영역만 처리하여 중요한 지역적 특징을 학습.
        - » 전체 데이터를 처리할 필요가 없어 연산량이 감소.
      - 가중치 공유(Weight Sharing)
        - » 같은 필터를 이미지 전체에서 사용하므로 학습해야 할 가중치 수가 크게 줄어듦
      - 공간 불변성(Spatial Invariance) : 위치나 크기가 약간 달라져도 동일한 패턴을 학습 가능.
    - 적용 분야 : 이미지 분류, 객체 탐지, 얼굴 인식, 영상 처리.

CNN (Convolutional Neural Network) 개념과 원리

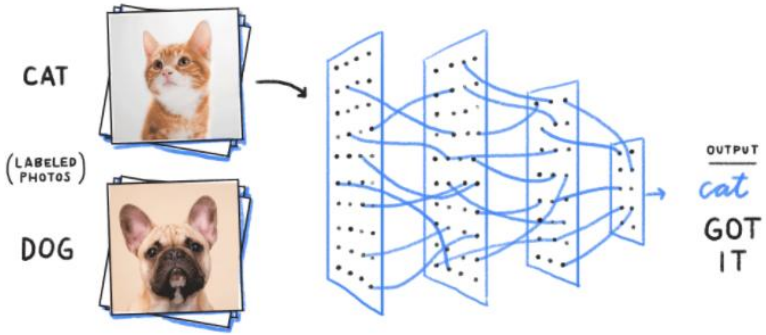
- 문제를 해결하기 위해 등장한 CNN, RNN
  - **순환 신경망(RNN: Recurrent Neural Network)**
    - 순서와 시간 의존성이 중요한 데이터를 처리하기 위해 RNN이 등장.
    - 일반 신경망은 시퀀스 데이터에서 과거 데이터를 기억하거나 문맥을 이해하지 못함.
    - 개선된 점
      - 기억 능력 : 은닉 상태(hidden state)를 통해 이전 데이터를 현재 학습에 활용.
      - 순서 정보 학습 : 시간적 관계를 고려해 데이터의 흐름을 이해.
      - 반복 구조 : 출력 값이 다시 입력으로 연결되어 데이터의 연속성을 학습 가능.
    - 적용 분야 : 자연어 처리(NLP), 음성 인식, 시계열 예측, 번역, 텍스트 생성.
    - 한계와 개선
      - 기본 RNN은 기울기 소멸 문제로 긴 시퀀스 데이터를 처리하기 어려움
      - 이를 극복하기 위해 **LSTM(Long Short-Term Memory)과 GRU(Gated Recurrent Unit)**가 등장.

CNN (Convolutional Neural Network) 개념과 원리

- 문제를 해결하기 위해 등장한 CNN, DNN, RNN
  - **심층 신경망(DNN: Deep Neural Network)**
    - 기존 신경망의 표현력 부족 문제를 해결하기 위해 여러 은닉층을 쌓은 심층 구조가 등장
    - 데이터의 복잡한 패턴과 고차원 특징을 학습
    - 개선된 점
      - 비선형성 학습
        - » 여러 은닉층과 활성화 함수를 통해 비선형 관계를 학습 가능.
      - 표현 학습(Representation Learning)
        - » 사람이 직접 특징을 설계할 필요 없이, 데이터에서 중요한 패턴을 자동으로 학습.
      - 모듈화 가능성
        - » 특정 문제에 맞는 구조로 쉽게 확장 가능.
    - 적용 분야 : 이미지, 텍스트, 음성, 신호 처리 등 전반적인 AI 문제.

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 개념



  - CNN (Convolutional Neural Network)

    - 주로 이미지와 같은 시각적 데이터를 처리하고 분석하기 위해 설계된 딥러닝 모델
      → 데이터를 분석하고 중요한 특징을 자동으로 추출할 수 있는 구조로 설계

    - CNN은 데이터의 공간적 구조(Spatial Structure)를 활용해 학습
      → 이미지 내의 위치 정보와 패턴(특징)을 유지하면서 처리하는 것이 특징

CNN (Convolutional Neural Network) 개념과 원리

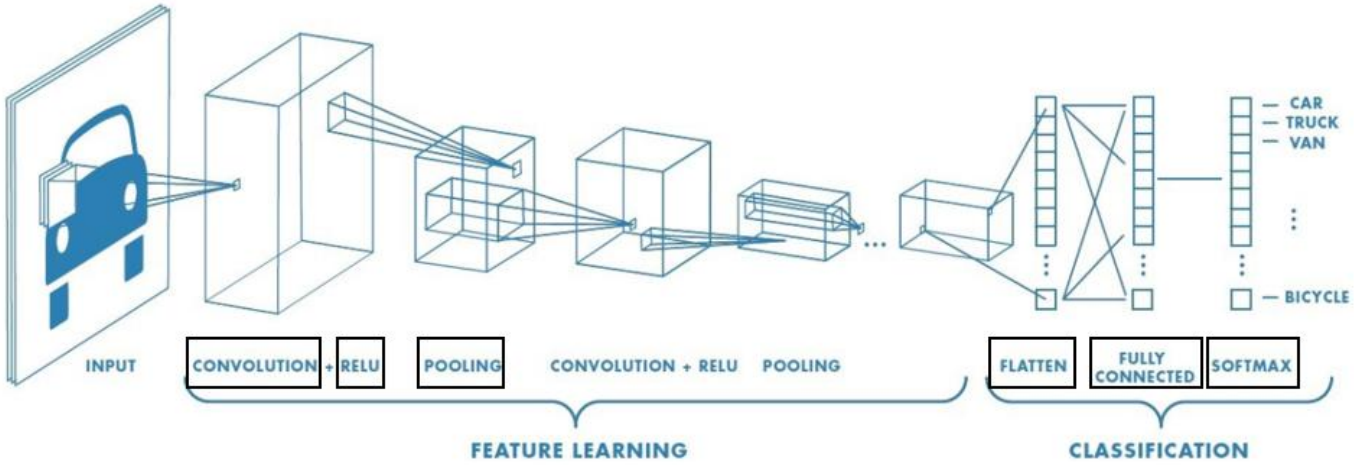
- CNN의 특징

  - 공간적 패턴 학습
    - 이미지를 입력받을 때 위치 정보와 패턴을 유지
  - 효율성
    - 풀링을 통해 데이터 크기를 줄이고 계산량을 감소 시킴
  - 다양한 적용 분야
    - 얼굴 인식, 자율 주행 자동차의 도로 객체 인식, 의료 영상 분석 등에 활용

CNN (Convolutional Neural Network) 개념과 원리

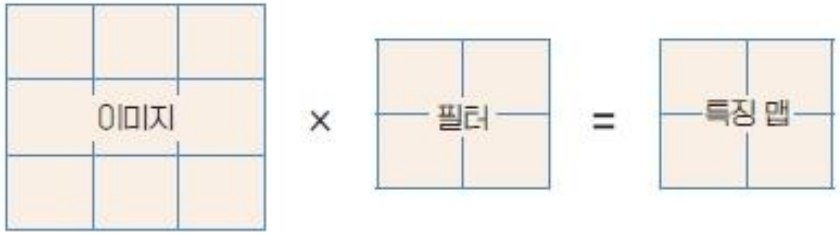
- CNN의 주요 구성 요소

  - Convolution Layer (합성곱 층)
  - 활성화 함수 (ReLU, Rectified Linear Unit)
  - Fully Connected Layer (완전 연결 층)



CNN (Convolutional Neural Network) 개념과 원리

- CNN의 주요 구성 요소



  - Convolution Layer (합성곱 층)
    - 이미지에서 특정 패턴(특징)을 찾아내는 역할
    - 핵심 개념
      - 이미지는 픽셀 값으로 이루어진 행렬(2D 배열)
      - 합성곱 층에서는 작은 **필터(또는 커널)**가 이미지 위를 이동하며 특정 패턴을 추출
        예를 들어, 필터는 수직선, 가로선, 경계 등을 탐지할 수 있음
    - 실제 사례
      - 얼굴 인식 시스템에서 필터가 눈, 코, 입 같은 패턴을 탐지하는 역할
      - 필터가 다양한 크기와 방향의 특징을 학습하여 얼굴의 윤곽을 추출

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 주요 구성 요소

  - Convolution Layer (합성곱 층)

-1	0	+1
-2	0	+2
-1	0	+1

Sobel-X
(vertical)

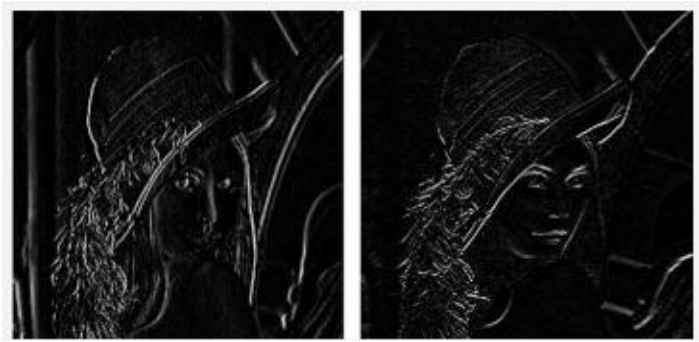
+1	+2	+1
0	0	0
-1	-2	-1

Sobel-Y
(horizontal)

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 주요 구성 요소

  - Convolution Layer (합성곱 층)



CNN (Convolutional Neural Network) 개념과 원리

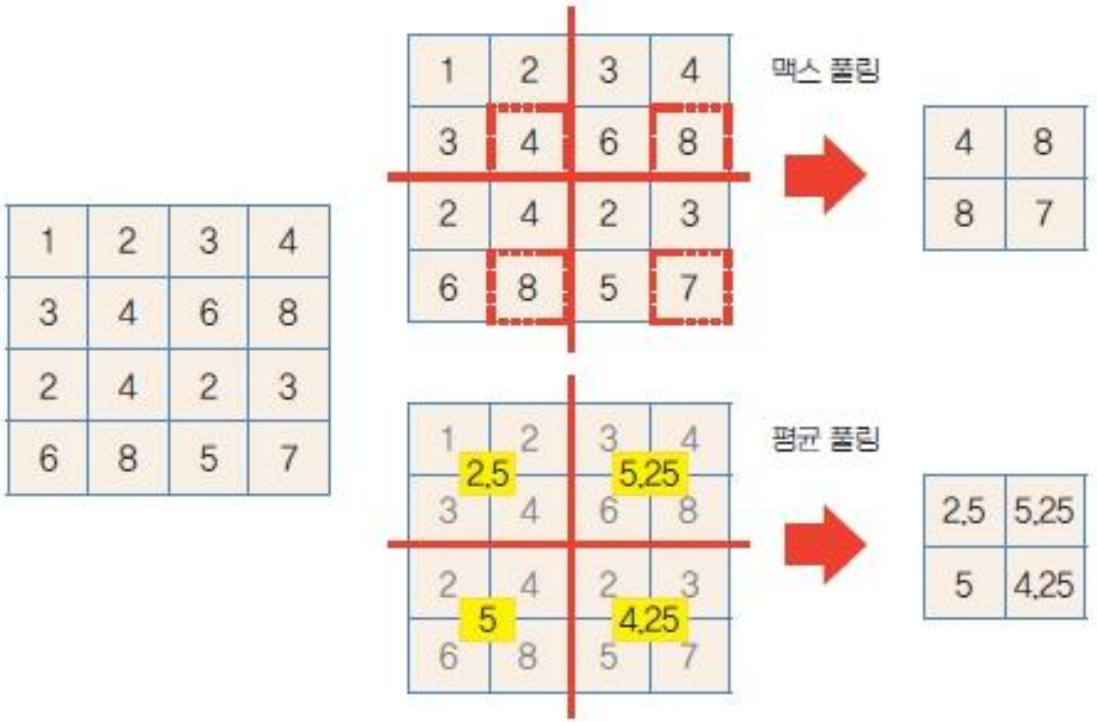
- CNN의 주요 구성 요소

  - Pooling Layer (**풀링 층**)
    - 이미지의 크기를 줄이고 중요한 정보를 유지하는 역할
    - 핵심 개념
      - 풀링은 데이터의 차원을 축소하여 계산량을 줄이고 과적합을 방지
      - 일반적으로 Max Pooling(**최대값 추출**)과 Average Pooling(**평균값 추출**)을 사용
    - 실제 사례
      - 이미지를 줄이면서도 중요한 정보(예: 가장 뚜렷한 경계)를 유지
      - 작은 이미지를 효율적으로 처리하여 스마트폰에서 사진 검색 속도를 높여줌

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 주요 구성 요소

  – Pooling Layer (풀링 층)



CNN (Convolutional Neural Network) 개념과 원리

- CNN의 주요 구성 요소

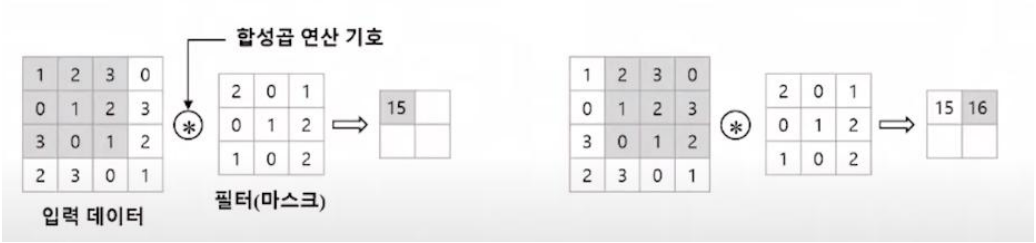
  - Fully Connected Layer (완전 연결 층)
    - 추출된 특징을 기반으로 최종 결과를 분류하거나 예측하는 역할
    - 핵심 개념
      - 합성곱 층과 풀링 층에서 학습한 정보를 활용해 클래스(종류)를 구분
      - 마지막 출력층에서 각 클래스의 확률 값을 계산
    - 실제 사례
      - 고양이와 개 이미지를 분류하는 모델에서, Fully Connected Layer는 "고양이일 확률 90%, 개일 확률 10%"과 같은 결과를 출력

first coding

CNN 동작 방식

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 동작 방식

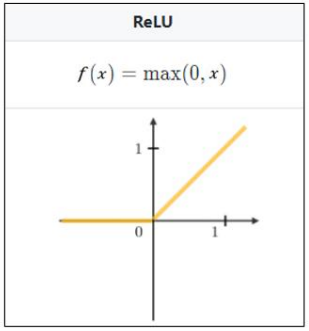


  - Convolution Layer (합성곱 층)
    - 이미지에서 특정 패턴(특징)을 찾아내는 역할
    - 작동 방식
      ① 작은 크기의 필터(커널)가 입력 데이터 위를 움직이며 겹침
      ② 필터와 입력 데이터의 값이 곱해진 합(점곱, convolution)을 계산하여 특징 맵을 생성
      ③ 특징 맵은 입력 데이터의 중요한 패턴(예: 이미지의 경계선, 색상 변화)을 나타냄
    - 사례
      - 이미지 데이터 → 고양이 사진에서 합성곱 층이 귀의 모양, 눈의 위치, 털 패턴 등의 특징을 감지

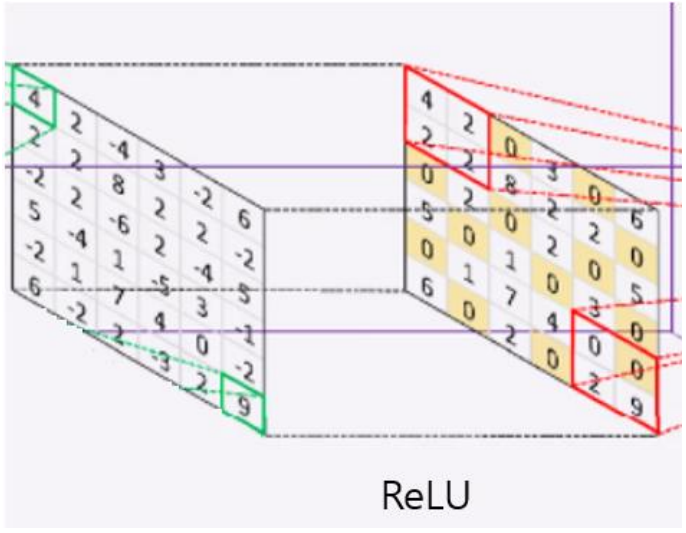
CNN (Convolutional Neural Network) 개념과 원리

- CNN의 동작 방식

$$f = \begin{cases} (x < 0) & f(x) = 0 \\ (x \geq 0) & f(x) = x \end{cases}$$



  - **활성화 함수 (ReLU, Rectified Linear Unit)**
    - 비선형성을 추가하여 모델이 복잡한 패턴을 학습할 수 있도록 함
    - 핵심 개념
      - 입력 값이 0보다 크면 그대로 출력하고,
        0보다 작으면 0으로 출력
      - 수학적 표현 : $f(x)=\max(0,x)$
    - 실제 사례
      - 특징 맵에서 음수 값은 제거하고 양수 값만 남김
        → 노이즈 제거 효과



CNN (Convolutional Neural Network) 개념과 원리

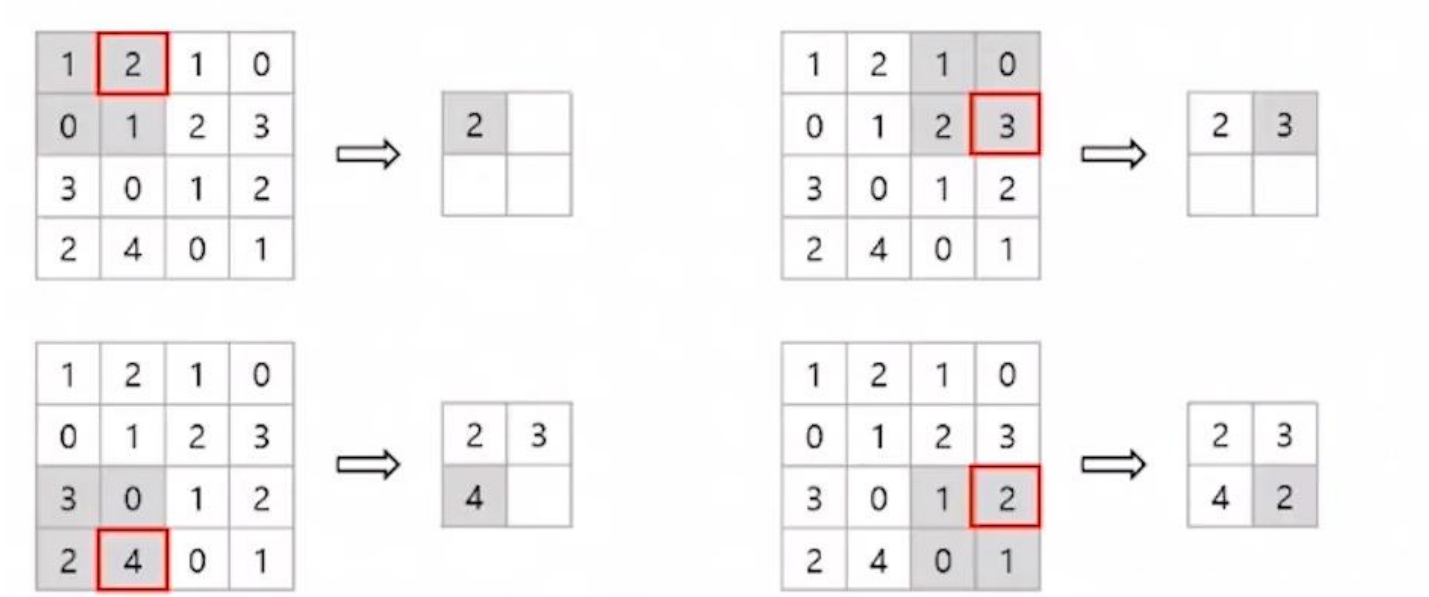
- CNN의 동작 방식

  - Pooling Layer (풀링 층)
    - 이미지의 크기를 줄이고 중요한 정보를 유지하는 역할
    - 핵심 개념
      - 풀링은 데이터의 차원을 축소하여 계산량을 줄이고 과적합을 방지
      - 일반적으로 Max Pooling(최대값 추출)과 Average Pooling(평균값 추출)을 사용
    - 실제 사례
      - 이미지를 줄이면서도 중요한 정보(예: 가장 뚜렷한 경계)를 유지
      - 작은 이미지를 효율적으로 처리하여 스마트폰에서 사진 검색 속도를 높여줌

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 동작 방식

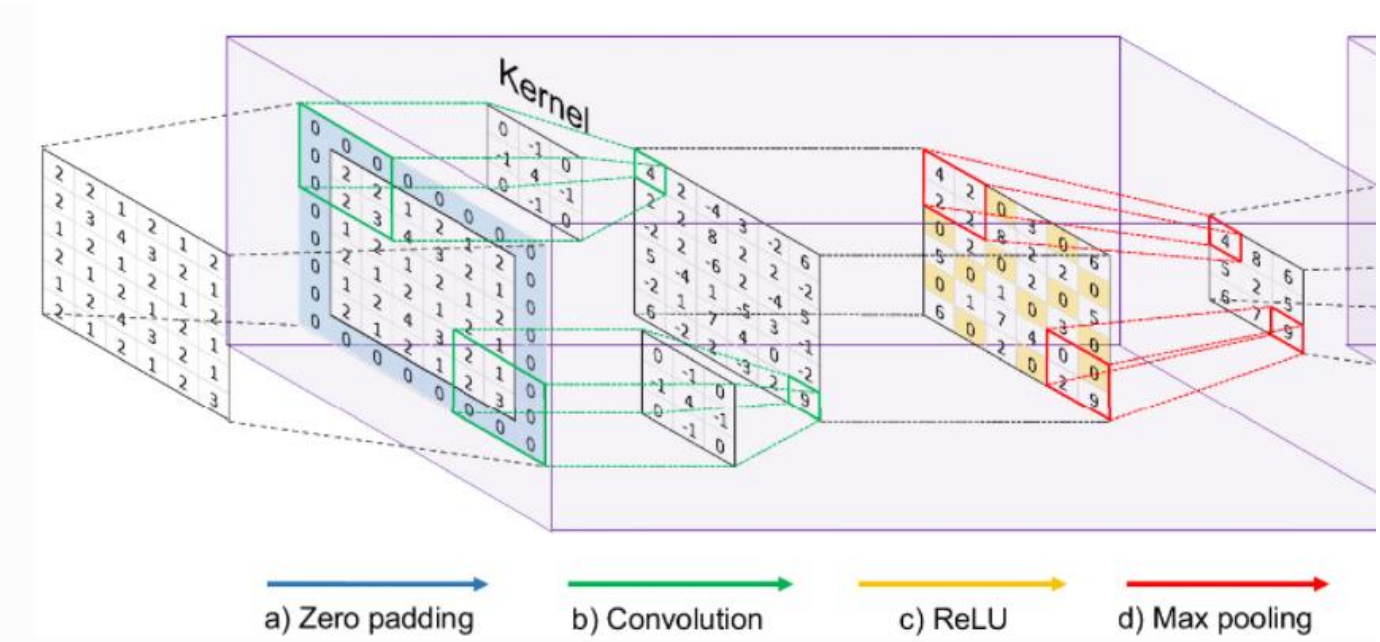
  – Pooling Layer (풀링 층)



CNN (Convolutional Neural Network) 개념과 원리

- CNN의 작동방식

  – Pooling Layer (풀링 층)



CNN (Convolutional Neural Network) 개념과 원리

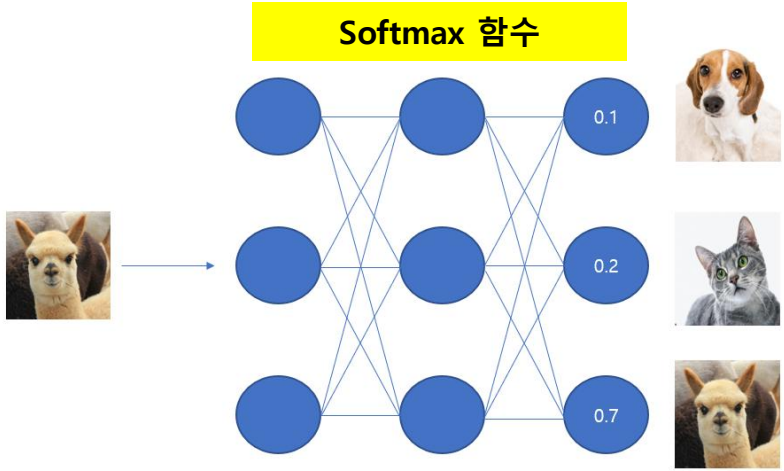
- CNN의 작동방식

  - Fully Connected Layer (완전 연결 층)
    - 추출된 특징을 기반으로 최종 결과를 분류하거나 예측하는 역할
    - 핵심 개념
      - 합성곱 층과 풀링 층에서 학습한 정보를 활용해 클래스(종류)를 구분
      - 마지막 출력층에서 각 클래스의 확률 값을 계산
    - 실제 사례
      - 고양이와 개 이미지를 분류하는 모델에서,
        Fully Connected Layer는 "고양이일 확률 90%, 개일 확률 10%"과 같은 결과를 출력

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 작동방식

  – Fully Connected Layer (완전 연결 층)

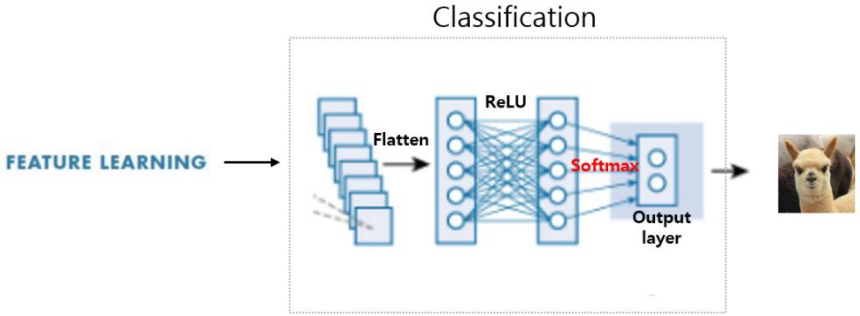


1	1	0
4	2	1
0	2	1

Pooled Feature Map

Flattening

1
1
0
4
2
1
0
2
1



CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름

① 입력(Input)

② 합성곱 층 (Convolution Layer)

③ 활성화 함수 (Activation Function)

④ 풀링 층 (Pooling Layer)

⑤ 완전 연결 층 (Fully Connected Layer)

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름

  - 입력(Input)
    - 이미지 데이터를 입력으로 받음
    - 이미지는 픽셀 값(0~255)로 이루어진 2D 배열(그레이스케일) 또는 3D 배열(RGB)

    - 예시
      - 28x28 크기의 숫자 이미지 (MNIST 데이터셋)
      - 224x224 크기의 고양이 사진 (컬러 이미지)

CNN (Convolutional Neural Network) 개념과 원리

- **CNN의 구조와 처리 흐름**

  - **합성곱 층 (Convolution Layer)**
    - 이미지에서 특정 패턴(선, 모서리, 질감)을 자동으로 추출
    - 합성곱 연산을 통해 작은 필터(커널)가 이미지 위를 이동하며 특징 맵(Feature Map)을 생성
    - 처리 흐름
      ① 작은 크기의 필터(3x3, 5x5 등)가 이미지 위를 한 칸씩 이동
      ② 각 필터는 이미지의 특정 패턴(예: 수직선, 수평선)을 탐지
      ③ 결과로 생성된 값들이 새로운 행렬(특징 맵)을 생성
    - 예시
      - 필터가 숫자 "3" 이미지에서 곡선 모양을 탐지.
      - 고양이 이미지에서 귀의 경계선이나 눈 모양을 탐지.

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름

  - **활성화 함수 (Activation Function)**
    - 특징 맵의 비선형성을 추가하고 중요하지 않은 값을 제거
      - 주로 ReLU(Rectified Linear Unit)를 사용하여 음수 값을 0으로 변환
      - 이를 통해 학습 과정에서 비선형 관계를 학습
    - 예시
      - 고양이의 귀를 탐지한 결과에서 음수 값을 제거해 신호를 강화

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름

  - 풀링 층 (Pooling Layer)
    - 데이터 크기를 줄이고(차원 축소) 주요 정보를 유지
      - Max Pooling : 영역 내에서 가장 큰 값을 선택
      - Average Pooling : 영역 내 값들의 평균을 계산
    - 처리 흐름
      ① 풀링 필터(2x2, 3x3 등)가 이미지 위를 이동하며 각 영역의 값을 축소.
      ② 데이터 크기를 줄여 계산 효율성을 높이고 과적합을 방지.
    - 예시
      - 4x4 이미지를 Max Pooling(2x2 필터)으로 처리하면 2x2 이미지로 크기가 축소
      - 고양이 귀의 윤곽을 더 간결한 형태로 유지

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름

  - 완전 연결 층 (Fully Connected Layer)
    - 추출된 특징을 기반으로 최종 분류 또는 예측을 수행
      - Flatten : 특징 맵을 1D 벡터로 변환
      - Dense Layer : 1D 벡터를 이용해 예측 결과를 생성
    - 예시
      - MNIST 숫자 분류 : 0~9 중 하나로 분류.
      - 고양이 vs 개 분류 : 고양이일 확률(90%)과 개일 확률(10%) 출력.

CNN (Convolutional Neural Network) 개념과 원리

- **CNN의 구조와 처리 흐름**
  - 예시 : 숫자 이미지 분류 (MNIST 데이터셋)
  - 목표 : 손글씨 숫자(0~9)를 CNN으로 분류.

① **입력 이미지**

28x28 크기의 숫자 이미지가 CNN에 입력

예: 숫자 "3"의 픽셀 값 배열.

② **합성곱 연산**

첫 번째 필터는 "곡선" 패턴을 탐지.

두 번째 필터는 "수평선"을 탐지.

결과로 여러 개의 특징 맵이 생성

③ **ReLU 활성화**: 음수 값을 0으로 변환해 주요 신호만 유지.

④ **풀링 연산**: 특징 맵의 크기를 줄이며 중요한 정보를 보존.

⑤ **완전 연결 층**

특징 맵을 기반으로 "3일 확률 98%"를 출력.

- **CNN의 구조와 처리 흐름**
  - 예시 : 고양이 vs 개 이미지 분류
  - 목표 : 손글씨 숫자(0~9)를 CNN으로 분류.

① **입력 이미지**

224x224 크기의 컬러 이미지.

② **합성곱 연산**

첫 번째 층 : 간단한 선과 모서리를 탐지.

두 번째 층 : 귀, 털 패턴과 같은 구체적인 특징 탐지.

세 번째 층 : 고양이 전체의 윤곽을 파악.

③ **풀링 연산**

이미지의 크기를 줄이며 주요 정보를 유지.

④ **완전 연결 층**

"고양이일 확률 90%, 개일 확률 10%" 출력.

CNN (Convolutional Neural Network) 개념과 원리

- **CNN의 구조와 처리 흐름 : 단계별 데이터**

① **입력(Input)**

- 이미지 데이터는 일반적으로 (높이, 너비, 채널) 형식의 3차원 배열로 표현
  - 28x28 크기의 흑백 이미지(채널=1)를 입력으로 하면, 데이터의 형식은 (28, 28, 1)
  - 컬러 이미지의 경우 채널은 3(RGB)으로 (높이, 너비, 3) 형식

② 합성곱 층 (Convolution Layer)

③ 활성화 함수 (Activation Function)

④ 풀링 층 (Pooling Layer)

⑤ 완전 연결 층 (Fully Connected Layer)

⑥ 출력계층

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름 : 단계별 데이터

① 입력(Input)

② 합성곱 층 (Convolution Layer)

$$\text{출력 크기} = \left(\frac{\text{입력 크기} - \text{필터 크기} + 2 \times \text{패딩}}{\text{스트라이드}} \right) + 1$$

- 필터(커널)를 사용해 입력 데이터와 합성곱 연산을 수행하여 출력 특징 맵(Feature Map)을 생성
  - 필터
    - » 작은 크기의 행렬(예: 3x3 또는 5x5)로 구성
    - » 이미지에서 특정 패턴(모서리, 선 등)을 감지
  - Stride
    - » 필터가 이동하는 간격.
  - Padding
    - » 입력 데이터의 가장자리를 유지하거나 확대하는 방법.

입력 데이터: 28x28x1

필터 크기: 3x3

스트라이드: 1

패딩: 없음

→ 출력 크기: 26x26x1

CNN (Convolutional Neural Network) 개념과 원리

- **CNN의 구조와 처리 흐름 : 단계별 데이터**

① **입력(Input)**

② **합성곱 층 (Convolution Layer)**

③ **활성화 함수 (Activation Function)**

- ReLU의 역할: 음수 값을 0으로 설정하고, 양수 값은 그대로 유지

  입력: [-1, 2, -3, 4] → ReLU 적용 후: [0, 2, 0, 4]

④ **풀링 층 (Pooling Layer)**

⑤ **완전 연결 층 (Fully Connected Layer)**

⑥ 출력계층

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름 : 단계별 데이터

① 입력(Input)

② 합성곱 층 (Convolution Layer)

③ 활성화 함수 (Activation Function)

④ 풀링 층 (Pooling Layer)

- 일반적으로 최대 풀링(Max Pooling) 사용

  [[1, 3], [2, 4]] → 2x2 Max Pooling → 출력: [[4]]

⑤ 완전 연결 층 (Fully Connected Layer)

⑥ 출력계층

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름 : 단계별 데이터

① 입력(Input)

② 합성곱 층 (Convolution Layer)

③ 활성화 함수 (Activation Function)

④ 풀링 층 (Pooling Layer)

⑤ 완전 연결 층 (Fully Connected Layer)

- Flatten 과정: 특징 맵을 1차원 벡터로 변환.
  입력 특징 맵: (7, 7, 64) → Flatten 후: (7*7*64 = 3136) 크기의 벡터.

⑥ 출력계층

CNN (Convolutional Neural Network) 개념과 원리

- CNN의 구조와 처리 흐름 : 단계별 데이터

① 입력(Input)

② 합성곱 층 (Convolution Layer)

③ 활성화 함수 (Activation Function)

④ 풀링 층 (Pooling Layer)

⑤ 완전 연결 층 (Fully Connected Layer)

⑥ 출력계층

- Softmax 함수
  - 다중 클래스 분류에서 확률 값으로 변환.

예시 : 3개의 클래스가 있다면
출력은 [0.1, 0.7, 0.2] 확률 분포

이미지 분류 작업 (MNIST 숫자 데이터)

1. 입력: 28x28 크기의 숫자 이미지.
2. 합성곱 계층: 32개의 3x3 필터 적용 → 출력: (26, 26, 32)
3. ReLU: 비선형성 추가.
4. Max Pooling: 2x2 필터로 풀링 → 출력: (13, 13, 32)
5. 합성곱 + 풀링 반복: (13, 13, 64) → (7, 7, 64)
6. Flatten: (7*7*64 = 3136) 크기의 벡터.
7. 완전 연결 계층: 128개의 노드 → 출력: [128]
8. 출력 계층: 10개의 노드(숫자 0~9) → 출력: [10]

TensorFlow/Keras를 사용하여 간단한 CNN 모델을 구현

- CNN 모델을 구현

① 데이터 전처리

- MNIST 데이터셋을 사용
- 이미지 데이터를 로드하고, 전처리를 수행

```python
# TensorFlow와 Keras를 사용한 CNN 모델 구현
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# MNIST 데이터셋 로드
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 데이터 확인
print("훈련 데이터 크기:", x_train.shape)
print("테스트 데이터 크기:", x_test.shape)

# 데이터 정규화
# (0~255 범위를 0~1 범위로 스케일링)
x_train = x_train / 255.0
x_test = x_test / 255.0

# 데이터 차원 변경
# (CNN은 4차원 입력을 요구: [샘플 수, 높이, 너비, 채널])
x_train = x_train.reshape(-1, 28, 28, 1)  # 채널 추가
x_test = x_test.reshape(-1, 28, 28, 1)

# 라벨을 One-Hot Encoding으로 변환
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

TensorFlow/Keras를 사용하여 간단한 CNN 모델을 구현

- CNN 모델을 구현

  ② CNN 모델 정의 및 레이어 추가

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Sequential API를 사용한 CNN 모델 정의
model = Sequential()

# 첫 번째 합성곱 레이어와 활성화 함수
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)))
# 최대 풀링 레이어
model.add(MaxPooling2D(pool_size=(2, 2)))

# 두 번째 합성곱 레이어와 최대 풀링 레이어
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 평탄화 레이어 (Fully Connected 레이어에 입력하기 위해 1차원으로 변환)
model.add(Flatten())

# 완전 연결층과 출력층
model.add(Dense(128, activation='relu'))  # 은닉층
model.add(Dense(10, activation='softmax'))  # 출력층 (10개 클래스)

# 모델 요약
model.summary()
```

TensorFlow/Keras를 사용하여 간단한 CNN 모델을 구현

- CNN 모델을 구현

③ 모델 컴파일 및 학습

```
# 모델 컴파일
model.compile(optimizer='adam',  # 최적화 함수
              loss='categorical_crossentropy',  # 손실 함수
              metrics=['accuracy'])  # 평가 메트릭

# 모델 학습
history = model.fit(x_train, y_train, epochs=10, batch_size=32,
validation_split=0.2)
```

TensorFlow/Keras를 사용하여 간단한 CNN 모델을 구현

- CNN 모델을 구현

④ 모델 학습 결과 시각화

```
# 학습 결과 시각화
plt.figure(figsize=(12, 4))

# 정확도 그래프
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# 손실 그래프
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

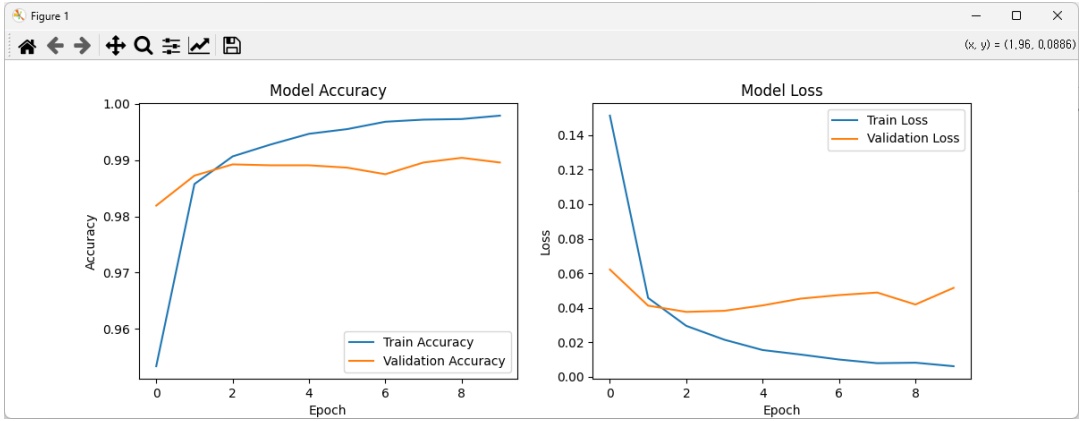
TensorFlow/Keras를 사용하여 간단한 CNN 모델을 구현

- CNN 모델을 구현

⑤ 모델 평가

테스트 데이터로 모델의 성능 평가.

```
# 테스트 데이터로 성능 평가
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"테스트 손실: {test_loss:.4f}")
print(f"테스트 정확도: {test_accuracy:.4f}")
```




```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9876 - loss: 0.0597
테스트 손실: 0.0451
테스트 정확도: 0.9906
```

first
coding

DCNN(Deep Convolutional Neural Network)

DCNN(Deep Convolutional Neural Network)

- DCNN(Deep Convolutional Neural Network)

  - DCNN(Deep Convolutional Neural Network)
    - 기존 CNN(Convolutional Neural Network)을 확장한 형태로, 모델의 깊이가 더 깊어진 합성곱 신경망

  - CNN과의 차이점
    - CNN
      - 몇 개의 합성곱 레이어와 풀링 레이어로 구성된 상대적으로 얕은 네트워크.
        사용 예시: 간단한 이미지 분류(손글씨 숫자 분류 등).
    - DCNN
      - CNN의 구조를 확장하여 수십, 수백 개의 레이어를 쌓은 심층 네트워크.
        사용 예시: 복잡한 이미지 분류, 객체 탐지, 영상 처리.

DCNN(Deep Convolutional Neural Network)

- DCNN(Deep Convolutional Neural Network)

  - 모델 깊이가 깊어질수록 얻을 수 있는 이점
    - 복잡한 특성 학습
      - 더 깊은 레이어는 데이터의 고차원 특징을 학습하여 복잡한 패턴을 더 잘 인식.

        예: 이미지에서 단순한 가장자리(초기 레이어) → 특정 사물의 형태(중간 레이어) → 전체 객체(최종 레이어).
    - 성능 향상
      - 데이터가 많을수록 깊은 네트워크가 더 높은 정확도를 제공.

        예: 고해상도 이미지 분류에서는 얕은 네트워크보다 심층 네트워크가 더 높은 성능을 보임.
    - 추론 능력 강화
      - 심층 네트워크는 다양한 데이터 변형(크기, 각도 등)에 더 강인함.

DCNN(Deep Convolutional Neural Network)

- **DCNN의 발전**

  - AlexNet (2012)
    - DCNN의 첫 성공 사례로, ImageNet 대회에서 우승하며 딥러닝의 가능성을 입증.
    - 특징
      - ReLU 활성화 함수 도입: 학습 속도 향상.
      - 드롭아웃 적용: 과적합 방지.
      - GPU 병렬 처리: 빠른 학습 가능.
    - 사례
      - ImageNet 데이터셋에서 1,000개의 클래스 분류 정확도를 기존보다 10% 이상 향상.

DCNN(Deep Convolutional Neural Network)

- **DCNN의 발전**

  - VGGNet (2014)
    - 네트워크 깊이가 성능에 미치는 영향을 확인하기 위해 설계된 모델.
    - 특징
      - 단순한 구조: 3x3 필터를 사용하는 합성곱 레이어를 깊게 쌓음.
      - 레이어 수: 16~19개의 깊은 구조(VGG16, VGG19).
      - 필터 크기(3x3)를 고정하여 더 많은 파라미터를 활용하고 학습 성능을 높임.
    - 사례
      - 고해상도 이미지 분류에서 우수한 성능 제공.

DCNN(Deep Convolutional Neural Network)

- **DCNN의 발전**

  - VGGNet (2014)
    - 네트워크 깊이가 성능에 미치는 영향을 확인하기 위해 설계된 모델.
    - 특징
      - 단순한 구조: 3x3 필터를 사용하는 합성곱 레이어를 깊게 쌓음.
      - 레이어 수: 16~19개의 깊은 구조(VGG16, VGG19).
      - 필터 크기(3x3)를 고정하여 더 많은 파라미터를 활용하고 학습 성능을 높임.
    - 사례
      - 고해상도 이미지 분류에서 우수한 성능 제공.

DCNN(Deep Convolutional Neural Network)

- **DCNN의 발전**

  - ResNet (2015)
    - 네트워크가 깊어질수록 발생하는 기울기 소실 문제를 해결.
    - 특징
      - Residual Connection(잔차 연결)
        - » 출력값이 바로 다음 레이어로 전달되도록 함.
        - » 깊은 네트워크에서도 학습이 가능하도록 돕는 구조.
        - » 수식: $y=F(x)+x$ (F는 학습해야 할 함수, x는 입력값)
      - 152개의 레이어로 구성된 ResNet-152는 ImageNet 대회에서 최고 성능 기록.
    - 사례
      - 의료 영상 분석에서 병변 탐지. 자율주행 차량의 객체 인식.

DCNN(Deep Convolutional Neural Network)

- Residual Connection

  - Residual Connection(잔차 연결)
    - 입력값(x)을 다음 레이어로 직접 연결해, 레이어에서 학습할 내용(F(x))과 입력값을 더해 출력.
    - 장점
      - **기울기 소실 문제 해결** : 역전파 시, 정보 손실 없이 기울기를 전달 가능.
      - **더 깊은 네트워크 구현 가능** : 수백 개의 레이어로도 학습 성능 유지.
      - **효율적인 학습** : 간단한 구조를 학습하며 복잡한 패턴까지 점진적으로 학습 가능.
    - 사례
      - **자율주행** : 다양한 환경에서의 객체 탐지(보행자, 도로 표지판 등)
      - **의료 데이터 분석** : MRI, CT 영상에서 암 병변을 고정밀 탐지.

DCNN(Deep Convolutional Neural Network)

- DCNN의 주요 구성 요소DCNN의 주요 구성 요소

  - Convolution Layer (합성곱 층)
  - Pooling Layer (풀링 층)
  - Fully Connected Layer (완전 연결 층)
  - Activation Function (활성화 함수)
  - 드롭아웃(Dropout)

DCNN(Deep Convolutional Neural Network)

- **DCNN의 주요 구성 요소DCNN의 주요 구성 요소**

  - Convolution Layer (합성곱 층)
    - 입력 이미지에서 작은 영역(커널 또는 필터)을 활용하여 특징을 추출
      - 초기 층에서는 엣지, 선, 색상 등 단순한 특징을 학습, 깊이감이 더해질수록 형태, 구조, 패턴 등 복잡한 특징을 학습
    - 동작
      - 필터(커널)가 입력 데이터를 슬라이딩하며 작은 영역에서 연산을 수행
      - 필터는 특정 특징(예: 수직선, 원형 등)을 감지하도록 학습
    - 출력
      - 특징 맵(Feature Map), 입력 데이터의 변형된 형태.
    - 사례:입력
      - 28x28 크기의 손글씨 숫자 이미지

DCNN(Deep Convolutional Neural Network)

- **DCNN의 주요 구성 요소DCNN의 주요 구성 요소**

  - Pooling Layer (풀링 층)
    - 이미지의 공간 크기를 줄이고, 중요한 정보를 추출하며 계산량을 줄임

      예: 최대 풀링(Max Pooling)을 통해 가장 강한 신호를 선택

      → 데이터 크기를 줄여 연산량을 감소시키고 특징을 강화하며, 과적합(overfitting)을 방지
    - 동작
      - 가장 흔한 방식은 맥스 풀링(Max Pooling)으로, 작은 영역에서 최댓값을 선택
    - 출력
      - 압축된 특징 맵.

DCNN(Deep Convolutional Neural Network)

- **DCNN의 주요 구성 요소DCNN의 주요 구성 요소**

  - Fully Connected Layer (완전 연결 층)
    - 마지막으로 모든 뉴런을 연결하여 분류를 수행
      예: 입력 이미지를 고양이 또는 개로 분류
    - 최종적으로 분류 또는 예측 결과를 출력
    - 동작
      - 모든 뉴런이 연결되어, 이전 레이어에서 학습된 특징을 활용해 결과를 도출
    - 출력 : 클래스 확률 또는 결과값.

DCNN(Deep Convolutional Neural Network)

- **DCNN의 주요 구성 요소DCNN의 주요 구성 요소**

  - Activation Function (활성화 함수)
    - 비선형성을 추가하여 더 복잡한 모델링이 가능하도록 만듬
    - 주요 함수: ReLU(Rectified Linear Unit), Sigmoid, Softmax

      → 주로 ReLU(Rectified Linear Unit)를 사용

      ReLU는 음수를 0으로 만들고, 양수는 그대로 둠

    - 사례

      입력: −2, 3, −1, 5 → ReLU 함수 적용: 0, 3, 0, 5

DCNN(Deep Convolutional Neural Network)

- **DCNN의 주요 구성 요소DCNN의 주요 구성 요소**

  - 드롭아웃(Dropout)
    - 특정 뉴런을 임의로 비활성화하여 과적합을 방지
    - 동작
      - 학습 과정에서 일부 뉴런을 무작위로 0으로 설정

DCNN(Deep Convolutional Neural Network)

- **DCNN의 동작 과정과 데이터 흐름**
  - 사례: 고양이와 강아지 이미지를 구분하는 DCNN
    - ① 입력 데이터 (Input Layer) : 예: 28x28 크기의 손글씨 숫자 이미지.
    - ② 컨볼루션 레이어 (Convolution Layer)
      - 필터를 사용해 이미지의 저수준 특징(엣지, 텍스처)을 추출 → 출력: 여러 개의 특징 맵 (예: 26x26 크기, 32개의 맵)
    - ③ 풀링 레이어 (Pooling Layer)
      - 특징 맵의 크기를 줄이고 중요한 특징만 남김 → 출력: 13x13 크기의 맵
    - ④ 여러 컨볼루션/풀링 레이어 반복
      - 점차 고수준 특징을 추출 (예 : 손글씨의 곡선, 숫자 모양 등)
    - ⑤ 완전 연결 레이어 (Fully Connected Layer)
      - 학습된 특징을 바탕으로 최종 분류를 수행 → 출력: 숫자 0~9의 확률.
    - ⑥ 출력 레이어 (Output Layer)
      - 가장 높은 확률을 가진 클래스를 예측 결과로 선택

DCNN(Deep Convolutional Neural Network)

- **DCNN의 동작 과정과 데이터 흐름**

  - 사례: 손글씨 숫자 분류

    ① 입력 : 28x28 손글씨 숫자 이미지 (숫자 5)

    ② 컨볼루션 레이어 : 3x3 필터를 사용해 엣지를 감지 → 특징 맵 생성

    ③ 풀링 레이어 : 특징 맵의 크기를 줄여 압축된 정보 생성

    ④ 컨볼루션/풀링 반복 : 점점 더 복잡한 특징(숫자의 곡선과 형태)을 학습

    ⑤ 완전 연결 레이어 : "5"일 확률이 가장 높은 결과로 출력

    ⑥ 출력 : 예측값 → 숫자 5.

TensorFlow/Keras를 사용하여 DCNN 구현 예제

- **DCNN 모델을 구현**

  - 전이 학습 개요
    - 전이 학습(Transfer Learning)
      - 사전 학습된 모델(이미 학습된 가중치와 구조)을 활용하여 새로운 작업에 적합하게 재 학습하는 기법.
    - Fine-Tuning
      - 사전 학습된 모델의 상위 레이어(일반적으로 Fully Connected Layer)를 새 데이터에 맞게 조정하거나, 일부 하위 레이어를 미세 조정하는 과정.

TensorFlow/Keras를 사용하여 DCNN 구현 예제

- 사전 학습된 VGG16 을 사용한 이미지 분류

① 데이터 셋 준비

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# 1. CIFAR-10 데이터셋 로드
# CIFAR-10 데이터셋은 10개의 클래스(비행기, 자동차, 새 등)를 포함한 32x32 크기의 이미지들로 구성
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# 데이터 정규화: 픽셀 값을 0~1 사이로 스케일링
# CIFAR-10 데이터는 0~255 범위의 픽셀 값을 가지므로, 255로 나누어 0~1 범위로 정규화
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# 레이블을 원-핫 인코딩
# 0~9의 정수 형태를 10차원의 벡터로 변환합니다.
# to_categorical 함수는 num_classes 매개변수로 클래스를 명시할 수 있으며, 기본값은 데이터에서 자동으로 추정됩니다.
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

TensorFlow/Keras를 사용하여 DCNN 구현 예제

- 사전 학습된 VGG16 을 사용한 이미지 분류

② 사전 학습된 VGG16 모델 로드

```
# 2. 사전 학습된 VGG16 모델 로드
# VGG16 모델을 불러옴
# include_top=False로 설정하여 마지막 Fully Connected Layer를 제거
# weights='imagenet'은 ImageNet 데이터셋으로 학습된 가중치를 사용
# input_shape는 입력 데이터의 형상을 정의하며
# (32, 32, 3)은 CIFAR-10 데이터셋에 맞는 크기입니다.
vgg16_base = VGG16(weights='imagenet’,
                   include_top=False,
                   input_shape=(32, 32, 3))

# 모델의 가중치를 고정 (전이 학습용)
# trainable=False로 설정하면 VGG16의 가중치가 고정되어 업데이트되지 않습니다.
vgg16_base.trainable = False
```

TensorFlow/Keras를 사용하여 DCNN 구현 예제

- 사전 학습된 VGG16 을 사용한 이미지 분류

③ 새로운 모델 정의

```
# 3. 새로운 모델 정의
# VGG16의 출력 위에 우리의 새로운 분류기를 추가
# Sequential 모델은 레이어를 순차적으로 쌓아올리는 단순한 방식의 모델
model = models.Sequential([
    vgg16_base,  # 사전 학습된 VGG16 모델
    layers.Flatten(),  # 평탄화: 다차원 데이터를 1차원으로 변환
    layers.Dense(256, activation='relu'),  # Fully Connected Layer 추가, 256개의 뉴런 사용
    # activation: 활성화 함수, 'relu' 외에도 'sigmoid', 'tanh' 등
    layers.Dropout(0.5),  # 드롭아웃: 과적합 방지를 위해 일부 뉴런을 무작위로 비활성화
    # rate: 드롭아웃 비율, 0.5는 50%의 뉴런을 비활성화
    layers.Dense(10, activation='softmax')  # CIFAR-10 클래스에 맞는 출력, 10개의 뉴런과 softmax 활성화 함수
    # softmax는 출력값을 확률로 변환하여 다중 클래스 분류에 사용
])

# 모델 컴파일
# optimizer: 최적화 알고리즘, 'adam', 'sgd', 'rmsprop' 등
# loss: 손실 함수, 다중 클래스 분류에서는 'categorical_crossentropy'를 사용
# metrics: 평가 지표, 'accuracy' 외에도 'precision', 'recall' 등을 추가할 수 있음
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

TensorFlow/Keras를 사용하여 DCNN 구현 예제

- 사전 학습된 VGG16 을 사용한 이미지 분류

④ 모델 학습

```
# 4. 모델 학습
# fit 함수는 모델을 학습시키는 데 사용
# validation_data: 검증 데이터 (x_test, y_test)를 지정하여 학습 중 검증 정확도를 확인
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, batch_size=64)

# epochs: 학습 반복 횟수, 기본값은 1
# batch_size: 한 번에 학습에 사용하는 데이터 샘플 수, 기본값은 32
```

TensorFlow/Keras를 사용하여 DCNN 구현 예제

- 사전 학습된 VGG16 을 사용한 이미지 분류

⑤ Fine-Tuning (세부 조정)

```
# 5. Fine-Tuning (세부 조정)
# VGG16의 일부 층을 학습 가능하도록 설정
for layer in vgg16_base.layers[-4:]:  # 마지막 4개 층만 학습 가능하도록 설정
    layer.trainable = True

# 모델 재컴파일 (Fine-Tuning 적용 후)
# 학습률(learning_rate)을 낮추어 가중치를 더 세밀하게 조정
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5), loss='categorical_crossentropy',
metrics=['accuracy'])

# Fine-Tuning 학습 진행
fine_tune_history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5, batch_size=64)
```

TensorFlow/Keras를 사용하여 DCNN 구현 예제

- 사전 학습된 VGG16 을 사용한 이미지 분류

⑦ 모델 평가

```
# 6. 모델 평가
# 학습 후 테스트 데이터셋에서 모델 성능 평가
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

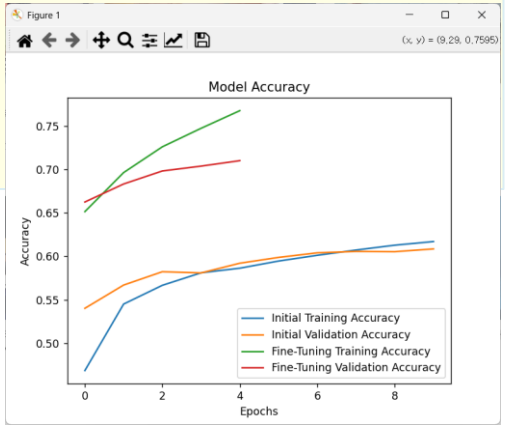
TensorFlow/Keras를 사용하여 DCNN 구현 예제

- 사전 학습된 VGG16 을 사용한 이미지 분류

⑥ 모델 평가

```
# 7. 결과 시각화
import matplotlib.pyplot as plt

# 학습 및 검증 정확도 시각화
plt.plot(history.history['accuracy'], label='Initial Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Initial Validation Accuracy')
plt.plot(fine_tune_history.history['accuracy'], label='Fine-Tuning Training Accuracy')
plt.plot(fine_tune_history.history['val_accuracy'], label='Fine-Tuning Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



first coding

RNN (Recurrent Neural Network)

RNN (Recurrent Neural Network)

- RNN의 기본 개념
  - RNN(Recurrent Neural Network)은 순환 신경망
  - 시퀀스 데이터(시간에 따라 변화하는 데이터)를 처리하고 학습하기 위해 설계된 딥러닝 모델
    - 입력 데이터가 시간 순서를 가지고 있는 경우 RNN이 효과적

      예 : 텍스트, 음성, 주식 데이터
  - 기존 신경망과의 주요 차이점은 순환 구조를 가지고 있다는 점

    → 이를 통해 이전 시점의 정보를 기억하고 이를 현재 시점의 예측에 활용

    기억의 능력을 가지고 있어서 이전의 입력 정보를 네트워크 내부에 유지하며

    다음 단계의 계산에 활용

RNN (Recurrent Neural Network)

- RNN (Recurrent Neural Network)

  - RNN의 주요 특징
    - 순환 구조
      - RNN의 각 노드는 자신으로부터 출력된 결과를 다시 입력으로 받아 이를 통해 시간의 흐름에 따라 데이터를 처리
    - 기억 능력
      - RNN은 이전 시점의 상태를 저장하며, 이를 통해 문맥(Context)을 이해하거나 패턴을 학습
    - 시퀀스 데이터 처리
      - RNN은 자연어 처리, 주가 예측, 음성 인식 등 연속적인 데이터를 처리하는 데 적합

RNN (Recurrent Neural Network)

- RNN (Recurrent Neural Network)

  - RNN의 사례
    - 텍스트 생성
      - 입력: "오늘 날씨는“
      - 출력: "맑고 화창합니다.“

        RNN은 입력된 텍스트의 단어를 순차적으로 처리하여, 문맥을 파악하고 자연스러운 다음 단어를 생성
    - 주가 예측
      - 입력: 특정 기간의 주가 데이터
      - 출력: 다음 날의 주가RNN은 이전 날짜의 주가 정보를 학습하여 미래 주가를 예측
    - 번역 시스템
      - 입력: "How are you?“
      - 출력: "잘 지내시나요?"RNN은 문장을 단어 단위로 처리하며 번역 시 문맥을 반영

RNN (Recurrent Neural Network)

- RNN (Recurrent Neural Network)

  - RNN의 한계
    - 장기 의존성 문제
      - 긴 시퀀스에서 초반 정보가 사라져 영향을 줄 수 없는 문제가 발생.
    - 기울기 소실(Gradient Vanishing)
      - 역전파 중 기울기가 작아져 학습이 제대로 이루어지지 않을 수 있음.
  - 개선된 모델
    - LSTM (Long Short-Term Memory)
      - RNN의 한계를 해결하기 위해 개발된 모델로, 기억을 제어하는 게이트 구조를 포함.
    - GRU (Gated Recurrent Unit)
      - LSTM의 간소화된 버전으로 계산 효율이 높음.

RNN (Recurrent Neural Network)

- RNN (Recurrent Neural Network)

  - 입력(Input)
    - 시간에 따라 변화하는 데이터 $X=(x1,x2,...,xT)$

      예를 들어, 텍스트 데이터에서 각 xt 는 한 단어에 해당
  - 순환 셀(Recurrent Cell)
    - 순환 신경망의 핵심으로, 입력과 이전 상태를 받아 새로운 상태를 계산

      $$ht=f(W\cdot xt+U\cdot ht-1+b)$$

      ht : 현재 상태
      $ht-1$: 이전 상태
      xt : 현재 입력
      W,U,b : 가중치와 편향
      f : 활성화 함수 (일반적으로 tanh 또는 ReLU 사용)
  - 출력(Output)
    - 각 시점의 상태를 기반으로 최종 출력 yt 를 생성

RNN (Recurrent Neural Network)

- RNN (Recurrent Neural Network)

  - **RNN의 동작 방식 (예: 텍스트 예측)**

    ① 입력 데이터 준비

      - "I love deep"이라는 문장이 주어졌다면 각 단어를 벡터로 변환

        → $x1$="I", $x2$="$love$", $x3$="$deep$"

    ② 순환 셀에 입력

      - 첫 번째 단어 "I"를 입력하면, RNN은 초기 상태 $h0$ 와 함께 계산하여 새로운 상태 $h1$ 을 생성
      - 이후 두 번째 단어 "love"를 입력하고, $h1$ 을 사용해 $h2$ 를 계산

    ③ 최종 출력

      - 마지막으로, 각 시점에서 상태를 기반으로 다음 단어를 예측하거나, 전체 시퀀스의 의미를 분석

        예 : "deep" 다음에 나올 단어를 예측 → "learning"

RNN (Recurrent Neural Network)

- RNN, LSTM, GRU 모델의 차이

  - RNN (Recurrent Neural Network)
    - 순환 신경망으로 시퀀스 데이터를 다루기 위해 설계되었음
    - 하지만 긴 시퀀스를 학습하는 데 문제가 있을 수 있음 (기울기 소실 문제).

  - LSTM (Long Short-Term Memory)
    - RNN의 개선된 버전으로, 기울기 소실 문제를 해결하며 장기 의존성(long-term dependency)을 학습

  - GRU (Gated Recurrent Unit)
    - LSTM보다 간단한 구조로 비슷한 성능을 제공하며 계산량이 적음

RNN 구현 예제

- **텍스트 데이터 예제**

1. 필요한 라이브러리 및 모듈 임포트
2. 셈플 데이터 정의

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, LSTM, GRU, Dense,
Embedding
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


# 샘플 텍스트 데이터
data = """나는 오늘 기분이 좋아.
나는 내일도 기분이 좋을 거야.
기분이 좋은 날엔 춤을 추고 싶어."""
```

RNN 구현 예제

- **텍스트 데이터 예제**

3. 토큰화 및 시퀀스 변환

```
# 1. 토큰화: 텍스트 데이터를 숫자로 변환
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data.split('\n'))
sequences = tokenizer.texts_to_sequences(data.split('\n’))
```

- Tokenizer(): Keras의 텍스트 전처리 도구로, 텍스트 데이터를 숫자로 변환
- fit_on_texts(): 텍스트 데이터를 학습시켜 단어의 고유 인덱스를 생성
- data.split('₩n'): 줄바꿈(₩n)을 기준으로 텍스트를 분리해 문장 리스트로 생성
- texts_to_sequences(): 문장을 단어의 고유 인덱스로 변환

  예 : "나는 오늘 기분이 좋아" → [1, 2, 3, 4] (인덱스는 학습된 결과에 따라 달라질 수 있음)

RNN 구현 예제

- **텍스트 데이터 예제**

4. 단어 인덱스 확인

```
# 2. 단어 인덱스 확인
word_index = tokenizer.word_index
print("단어 인덱스:", word_index)
```

– tokenizer.word_index: 학습된 단어와 그에 대응하는 인덱스를 딕셔너리 형태로 반환

출력 예 : {'나는': 1, '오늘': 2, '기분이': 3, '좋아': 4, ...}.

RNN 구현 예제

- **텍스트 데이터 예제**

5. 학습 데이터 시퀀스 생성

```
# 3. 시퀀스를 학습 데이터로 변환
input_sequences = []
for sequence in sequences:
    for i in range(1, len(sequence)):
        input_sequences.append(sequence[:i+1])
```

– 각 문장을 학습 가능한 데이터 형식으로 변환

– sequence[:i+1]: 첫 번째 단어부터 현재 단어까지의 시퀀스를 생성

예 : [1, 2, 3, 4] → [1, 2], [1, 2, 3], [1, 2, 3, 4].

RNN 구현 예제

- **텍스트 데이터 예제**

6. 패딩 처리

```
# 4. 패딩 처리
max_len = max(len(x) for x in input_sequences)
input_sequences = pad_sequences(input_sequences,
                                maxlen=max_len,
                                padding='pre')
```

- max_len: 가장 긴 시퀀스의 길이를 계산
- pad_sequences(): 시퀀스의 길이를 max_len으로 맞춤
  - maxlen: 시퀀스의 최종 길이.
  - padding='pre': 짧은 시퀀스의 앞쪽에 0을 추가
    예 : [1, 2] → [0, 0, 1, 2] (길이가 4로 맞춰짐).

RNN 구현 예제

- **텍스트 데이터 예제**

7. 입력(X)과 출력(y) 분리

```
# 5. 입력(X)과 출력(y) 분리
X = input_sequences[:, :-1]
y = input_sequences[:, -1]
```

- X : 입력 데이터로, 마지막 단어를 제외한 시퀀스.

  예 : [0, 0, 1, 2] → [0, 0, 1]

- y : 출력 데이터로, 시퀀스의 마지막 단어

  예 : [0, 0, 1, 2] → 2

RNN 구현 예제

- **텍스트 데이터 예제**

8. 출력 데이터 원-핫 인코딩

```
# 6. 출력(y)을 원-핫 인코딩
y = tf.keras.utils.to_categorical(y, num_classes=len(word_index) + 1)
```

- to_categorical() : 출력 데이터를 원-핫 벡터로 변환
  - num_classes: 클래스(단어) 개수

    예: 2 → [0, 1, 0, 0, ...].

RNN 구현 예제

- **텍스트 데이터 예제**

9. 모델 정의

```
# RNN 모델 정의
rnn_model = Sequential([
    Embedding(input_dim=len(word_index) + 1,
              output_dim=10,
              input_length=max_len - 1),  # 임베딩 층
    SimpleRNN(64, return_sequences=False),  # RNN 층
    Dense(len(word_index) + 1,
              activation='softmax')  # 출력 층
])
```

– **Embedding**

  - input_dim: 입력 단어의 총 개수

    (len(word_index) + 1)

  - output_dim: 임베딩 벡터의 차원(10)
  - input_length: 입력 시퀀스의 길이(max_len - 1)

– **SimpleRNN**

  - 64: RNN의 유닛 수(출력 차원)
  - return_sequences=False: 마지막 출력만 반환

– **Dense**

  - len(word_index) + 1: 출력 크기(단어 개수)
  - activation='softmax': 확률 분포를 반환.

RNN 구현 예제

- **텍스트 데이터 예제**

10. 모델 정의

```
# 모델 컴파일 및 학습
rnn_model.compile(loss='categorical_crossentropy’,
                  optimizer='adam’,
                  metrics=['accuracy'])
rnn_model.fit(X, y, epochs=20, verbose=1)
```

- compile
  - Loss=‘categorical_crossentropy’: 다중 클래스 분류에 적합한 손실 함수
  - Optimizer=‘adam’: Adam 옵티마이저
  - Metrics=[‘accuracy’]: 정확도를 평가 지표로 사용.
- fit
  - X, y: 학습 데이터와 라벨
  - epochs=20: 20번 반복 학습
  - verbose=1: 학습 진행 상황을 출력.

RNN 구현 예제

- **텍스트 데이터 예제**

  LSTM 모델 정의

```
# LSTM 모델 정의
lstm_model = Sequential([
    Embedding(input_dim=len(word_index) + 1,
              output_dim=10,
              input_length=max_len - 1),  # 임베딩 층
    LSTM(64, return_sequences=False),  # LSTM 층
    Dense(len(word_index) + 1,
          activation='softmax')  # 출력 층
])

# 모델 컴파일 및 학습
lstm_model.compile(loss='categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
lstm_model.fit(X, y, epochs=20, verbose=1)
```

  - Embedding 층
    - input_dim:단어의 개수를 나타냄
      - len(word_index) + 1로
        단어 사전에 존재하는 단어의 총 개수
      - +1은 패딩 토큰을 포함하기 위함
    - output_dim
      - 단어를 임베딩 벡터로 변환할 때 각 단어를 표현할 차원의 크기 여기서는 10으로 설정
        예를 들어, 각 단어가 10차원 벡터로 변환
    - input_length
      - 입력 시퀀스의 길이를 지정. 여기서는 max_len - 1로, 입력 시퀀스의 길이는 패딩 처리 후 일정함

RNN 구현 예제

- **텍스트 데이터 예제**

  LSTM 모델 정의

  - LSTM 층
    - units=64
      - LSTM 유닛의 개수
      - 출력 벡터의 차원 크기
      - 각 시점의 출력(hidden state)이 64차원 벡터로 표현
    - return_sequences=False
      - 기본적으로 False이며, 마지막 타임스텝의 출력만 반환
        예를 들어, 문장이 5개의 단어로 구성되어 있으면, 마지막 단어의 출력을 반환
      - True로 설정하면 모든 타임스텝의 출력을 반환하며, 이를 다른 RNN 계열 층으로 넘길 때 유용

```
# LSTM 모델 정의
lstm_model = Sequential([
    Embedding(input_dim=len(word_index) + 1,
              output_dim=10,
              input_length=max_len - 1),  # 임베딩 층
    LSTM(64, return_sequences=False),  # LSTM 층
    Dense(len(word_index) + 1,
          activation='softmax')  # 출력 층
])

# 모델 컴파일 및 학습
lstm_model.compile(loss='categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
lstm_model.fit(X, y, epochs=20, verbose=1)
```

RNN 구현 예제

- **텍스트 데이터 예제**

  GRU 모델 정의

  - GRU 층
    - LSTM보다 구조가 간단하지만 유사한 기능을 수행
    - units=64
      - GRU 유닛의 개수
      - 출력 벡터의 차원 크기
    - return_sequences=False
      - LSTM과 동일하게 마지막 타임스텝의 출력만 반환

```
# GRU 모델 정의
gru_model = Sequential([
    Embedding(input_dim=len(word_index) + 1,
              output_dim=10,
              input_length=max_len - 1), # 임베딩 층
    GRU(64, return_sequences=False),  # GRU 층
    Dense(len(word_index) + 1,
          activation='softmax')  # 출력 층
])

# 모델 컴파일 및 학습
gru_model.compile(loss='categorical_crossentropy’,
                  optimizer='adam’,
                  metrics=['accuracy'])
gru_model.fit(X, y, epochs=20, verbose=1)
```

RNN 구현 예제

- **텍스트 데이터 예제**

  추가 단어 생성 함수

```python
def generate_text(model, tokenizer, seed_text, next_words, max_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_len - 1, padding='pre')
        predicted = np.argmax(model.predict(token_list, verbose=0), axis=-1)
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                seed_text += " " + word
                break
    return seed_text
```

  - model
    - 학습된 딥러닝 모델(예: RNN, LSTM, GRU). 다음 단어를 예측하는 데 사용
  - tokenizer
    - 텍스트를 숫자로 변환하기 위해 사용되는 Tokenizer 객체임. 학습 시 사용한 단어 인덱스를 그대로 사용해야 함
  - seed_text
    - 텍스트 생성의 시작점이 되는 초기 문장. 모델은 이 문장을 기반으로 다음 단어를 예측
  - next_words
    - 생성할 단어의 수. 예를 들어, next_words=5라면 시드 텍스트 뒤에 다섯 개의 단어를 생성
  - max_len
    - 입력 시퀀스의 최대 길이. 학습 시 사용한 패딩 길이와 동일해야 함

RNN 구현 예제

- **텍스트 데이터 예제**

  추가 단어 생성 함수

```
def generate_text(model, tokenizer, seed_text, next_words, max_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_len - 1, padding='pre')
        predicted = np.argmax(model.predict(token_list, verbose=0), axis=-1)
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                seed_text += " " + word
                break
    return seed_text
```

  - range(next_words) : 생성할 단어의 수만큼 루프를 반복
  - tokenizer.texts_to_sequences :
    - 주어진 텍스트(seed_text)를 숫자 시퀀스로 변환
    - 반환값은 리스트의 리스트이므로 [0]을 사용해 첫 번째 리스트를 가져옴

      예 : "나는 오늘" → [1, 2] (단어 인덱스 기반).
  - pad_sequences
    - 시퀀스의 길이를 max_len - 1로 맞춤. 학습 시와 동일한 길이여야 함
    - padding='pre':시퀀스의 앞쪽에 0을 추가해 길이를 맞춤

      예 : [1, 2] → [0, 0, 1, 2] (길이가 4로 맞춰짐).

RNN 구현 예제

- **텍스트 데이터 예제**

  추가 단어 생성 함수

```
def generate_text(model, tokenizer, seed_text, next_words, max_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_len - 1, padding='pre')
        predicted = np.argmax(model.predict(token_list, verbose=0), axis=-1)
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                seed_text += " " + word
                break
    return seed_text
```

- tokenizer.word_index.items()
  - 단어와 인덱스의 쌍을 순회

    예: {'나는': 1, '오늘': 2, ...}
- if index == predicted
  - 모델이 예측한 단어의 인덱스(predicted)와 일치하는 단어를 검색
- seed_text += " " + word
  - 예측된 단어를 시드 텍스트에 추가

    예 : "나는 오늘" → "나는 오늘 기분이".

RNN 구현 예제

- **텍스트 데이터 예제**

  텍스트 생서 예제

```
# 텍스트 생성 예제
seed_text = "나는 오늘"
print("RNN 생성 결과:", generate_text(rnn_model, tokenizer, seed_text, next_words=5, max_len=max_len))
print("LSTM 생성 결과:", generate_text(lstm_model, tokenizer, seed_text, next_words=5, max_len=max_len))
print("GRU 생성 결과:", generate_text(gru_model, tokenizer, seed_text, next_words=5, max_len=max_len))
```