영상 및 이미지 기초

이미지와 영상의 기본 개념

- 이미지 : 픽셀로 이루어진 2차원 배열
  - RGB 이미지의 경우, 세 개의 채널(R, G, B)을 가짐
- 영상 : 시간에 따라 변화하는 이미지의 연속
  - 프레임 단위로 처리

데이터 전처리

- Normalization(정규화) : 픽셀 값을 0-1 범위로 조정
- Augmentation(증강) : 데이터를 다양하게 변형하여 학습 데이터를 늘림
  - 회전, 자르기, 색상변화 등

컨볼루션 층

- 필터(Filter)와 커널(Kernel)
  - 필터(또는 커널) : 작은 크기의 행렬로, 일반적으로 3 x 3, 5 × 5, 7 × 7 크기를 가짐
  - 필터는 이미지의 일부 영역을 스캔하면서 점곱 연산을 수행하여 결과를 생성

    ▼ 3 × 3 필터 예시

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

- 스트라이드
  - 필터가 이미지 위를 이동하는 단계의 크기
  - 스트라이드가 1이면 필터가 한 칸씩 이동, 스트라이드가 2이면 두 칸씩 이동
  - 스트라이드가 클수록 출력 크기는 줄어듬

- 패딩
  - 입력 이미지의 경계 부분을 처리하기 위해 여백을 추가하는 방법
  - 패딩 유형
    - Valid Padding : 패딩을 추가하지 않음, 출력 크기가 줄어듬
    - Same Padding : 출력 크기가 입력 크기와 동일하게 유지되도록 패딩을 추가
- 출력 크기 계산
  - 컨볼루션 층의 출력 크기는 아래 공식을 사용하여 계산
    - $\text{출력크기} = \frac{(\text{입력크기} - \text{필터크기} + 2) * \text{패딩}}{\text{스트라이드}} + 1$
- 컨볼루션 연산
  - 입력 이미지에 필터를 적용하여 특성 맵(feature map)을 생성
  - 각 특성 맵은 필터가 이미지에서 감지한 특정 패턴을 나타냄
- 활성화 함수
  - 컨볼루션 연산 후, 비선형을 추가하기 위해 ReLU와 같은 활성화 함수를 적용
    - 모델이 복잡한 패턴을 학습할 수 있게 도와줌

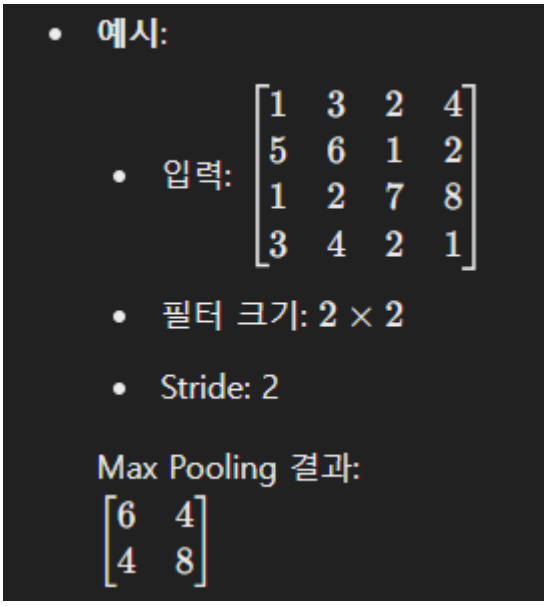
```
컨볼루션  예시코드
import torch
import torch.nn as nn

# 입력 이미지 생성 (배치 크기 1, 채널 1개, 높이 32, 너비 32)
input_image = torch.randn(1, 1, 32, 32)

# 컨볼루션 레이어 정의 (입력 채널 1, 출력 채널 16, 필터 크기 3x3, 스트리
conv_layer = nn.Conv2d(in_channels=1, out_channels=16, kernel_
conv_output = conv_layer(input_image)

print("Conv Layer Output Shape:", conv_output.shape)
```

풀링

- 컨볼루션 신경망에서 중요한 역할을 하는 연산 중 하나
  - 이미지의 공간적인 크기를 줄이고, 계산 비용을 줄이며, 특정 특징을 강화하기 위해 사용

    - 풀링은 필터를 사용하여 이미지의 특정 영역에서 추출하는 방식으로 작동
- 풀링의 종류
    - Max Pooling(최대 풀링)
        - Max Pooling은 필터가 커버하는 영역 내에서 가장 큰 값을 선택 → 주요 특징을 강조하고 노이즈를 줄이는데 도움

        ▼ 예시



    - Average Pooling(평균 풀링)
        - Average Pooling은 필터가 커버하는 영역 내의 값들의 평균을 계산 → 이미지의 축소된 버전에서 평균적인 특성을 유지

        ▼ 예시

- **예시:**
  - 입력: $\begin{bmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 1 & 2 \\ 1 & 2 & 7 & 8 \\ 3 & 4 & 2 & 1 \end{bmatrix}$
  - 필터 크기: 2×2
  - Stride: 2

  Average Pooling 결과:
  $\begin{bmatrix} 3.75 & 2.25 \\ 2.5 & 4.5 \end{bmatrix}$

- 풀링의 효과
  - 차원 축소 : 풀링은 이미지의 공간적 크기를 줄여 계산량을 감소시키고, 메모리 사용량을 줄임

    → 모델을 더 효율적으로 학습시키고,과적합(overfitting)을 방지하는데 도움이 됨
  - 불변성 제공 : 풀링은 입력 이미지의 작은 변화(예: 이동, 회전 등)에 대해 모델이 더 안정적이고 불변한 특성을 가지도록 함
  - 주요특징 : Max Pooling의 경우, 중요한 특징을 강조하여 모델이 더 중요한 정보를 학습할 수 있게 도움
- 풀링의 출력 크기 계산
  - 출릭 크기 공식(풀링)
    - 출력크기 $= \frac{\text{입력크기} - \text{필터크기}}{\text{스트라이드}} + 1$

    ▼ 예시

    - 입력 이미지 크기: 32×32
    - 풀링 필터 크기: 2×2
    - 스트라이드: 2

    출력 크기 $= \frac{(32-2)}{2} + 1 = 16$

풀링 예시코드

```
# Max Pooling 레이어 정의 (필터 크기 2x2, 스트라이드 2)
max_pool_layer = nn.MaxPool2d(kernel_size=2, stride=2)
max_pooled_output = max_pool_layer(conv_output)

print("Max Pool Layer Output Shape:", max_pooled_output.shape

# Average Pooling 레이어 정의 (필터 크기 2x2, 스트라이드 2)
avg_pool_layer = nn.AvgPool2d(kernel_size=2, stride=2)
avg_pooled_output = avg_pool_layer(conv_output)

print("Average Pool Layer Output Shape:", avg_pooled_output.s

import torch
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=3

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        return x

# 모델 인스턴스 생성
model = SimpleCNN()

# 임의의 입력 이미지 생성 (배치 크기 1, 채널 1개, 높이 32, 너비 32)
input_image = torch.randn(1, 1, 32, 32)

# 모델을 통해 예측 수행
output = model(input_image)
```

```
print("Model Output Shape:", output.shape)
```