과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

➢Intro

•학습열기

•학습목표

➢학습하기

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

➢적용하기

➢Outro

•문제풀기

- 왜 고양이와 강아지를 구분할 수 있을까요?

우리 눈에는 너무나 당연하게 느껴지는 일이 있습니다. 예를 들어, 우리는 사진 속 고양이와 강아지를 한눈에 구분합니다. 하지만 컴퓨터에게 이 일을 맡긴다면 어떨까요? 머신러닝이 처음 등장했을 때, 고양이 사진 수천 장을 학습시켜도 모델은 개와 구분하지 못하곤 했습니다. 그 이유는 무엇일까요? 바로 이미지 속 픽셀 정보만으로는 패턴을 파악하는 것이 쉽지 않기 때문입니다. 이를 해결하기 위해 우리는 합성곱 신경망(CNN)이라는 강력한 도구를 사용하게 되었습니다. CNN은 이미지를 구성하는 각 요소인 윤곽, 색상, 질감 등을 점진적으로 학습하여 의미 있는 특성을 추출합니다. 하지만 이 CNN도 학습의 효율과 정확도를 높이기 위해 여러 기술적 최적화가 필요합니다. 이번 학습에서는 데이터 증강, 학습률 조정, 과적합 방지 기법, 그리고 클래스 불균형 대응 전략 등을 통해 실제로 모델이 어떻게 똑똑해지는지를 체험하게 됩니다. 인간의 시각을 흉내 내는 인공 지능의 첫걸음을 여러분과 함께 디버깅하며 확인해보고 싶습니다.

① 본 학습 내용으로 들어가기 전, 학습 주제의 흥미를 이끌 만한 도입부의 내용이 있다면 제시해주세요.

② ex. 관련 뉴스기사, 실생활과 관련된 이야기 등

③ 저작권 침해가 되지 않도록 내용을 구성해 주세요.

④ 출처가 있을 경우 반드시 작성해 주세요.

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

➢ **Intro**

•학습열기

•학습목표

➢ **학습하기**

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

➢ **적용하기**

➢ **Outro**

•문제풀기

◈ 학습목표

1. 데이터 증강 및 학습률 조정을 적용할 수 있다.
2. 네트워크 깊이와 필터 크기를 조정하여 성능을 비교할 수 있다.
3. 불균형 데이터 문제 해결 기법을 적용할 수 있다.

◈ 학습내용

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

① 학습내용과 학습목표는 강의계획서와 일치해야 하며, 필요시 강의계획서를 수정할 수 있습니다.

② **학습목표**

✓ 각 레슨에 맞는 학습목표를 2~3개 작성해 주세요.

③ **학습내용**

✓ 1회차 당 25분 분량이 되도록 2~3개 레슨으로 구성해주세요.

✓ 학습내용과 레슨명은 일치해야 합니다.

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- ➢ **Intro**
  - •학습열기
  - •학습목표
- ➢ **학습하기**
  1. CNN 학습 최적화
  2. CNN 구조 최적화
  3. Dataset 최적화
- ➢ **적용하기**
- ➢ **Outro**
  - •문제풀기

간지

CNN 학습 최적화

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 고도화된 CNN 모델 구현

필요 라이브러리 메모리 로드

```
# PyTorch를 이용한 고도화된 CNN 모델 구현
# CIFAR-10 데이터셋을 기반으로 모델 성능 향상을 위해 다양한 기법을 적용한 예제

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from torchsummary import summary
from torch.utils.data import WeightedRandomSampler
from collections import Counter
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 데이터 증강 강화 1

ColorJitter()를 통해 학습 데이터 다양성 증가

이미지의 색상 속성(밝기, 대비, 채도)을 무작위로 변경해 다양한 광학 조건에 대응 가능한 학습을 유도함

모델은 다양한 색상 조건에서도 견고한 예측을 할 수 있게 됨

brightness=0.2

밝기를 ±20% 범위 내에서 무작위로 조절, 픽셀이 어두운 사진, 밝은 사진 모두 학습

contrast=0.2

대비를 ±20% 범위로 조정

명암 차가 뚜렷한 사진과 흐릿한 사진 모두 대응하도록 함

saturation=0.2

채도를 ±20% 조정해서 색감이 풍부하거나 밋밋한 이미지 모두 학습하게 함

```
transforms.ColorJitter(brightness=0.2,          # 밝기 조절
                       contrast=0.2,            # 대비 조절
                       saturation=0.2),         # 채도 조절
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 데이터 증강 강화 2

RandomRotation 을 통해 학습 데이터 다양성 증가

이미지를 -10도에서 +10도 사이로 무작위 회전시킴

$-10° \leq \theta \leq +10°$ 범위 내에서 무작위 회전

객체가 항상 정방향으로만 존재하는 건 아니므로 약간 기울어진 이미지를 학습하도록 만듦

```
transforms.RandomRotation(10),  # -10도 ~ +10도 범위에서 무작위 회전
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 학습 데이터 적용한 데이터 증강 기법 코드

```
# 학습 데이터에 다양한 데이터 증강 기법을 적용해 모델의 일반화 성능을 향상
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),              # 이미지를 좌우로 무작위 반전
    transforms.RandomCrop(32, padding=4),           # 4픽셀 패딩 후 32x32 크기로 무작위 크롭
    transforms.ColorJitter(brightness=0.2,          # 밝기 조절
                           contrast=0.2,            # 대비 조절
                           saturation=0.2),         # 채도 조절
    transforms.RandomRotation(10),                  # -10도~+10도 범위에서 무작위 회전
    transforms.ToTensor(),                          # PIL 이미지를 Tensor로 변환
    transforms.Normalize((0.4914, 0.4822, 0.4465), # 각 채널별 평균과 표준편차로 정규화
                         (0.2023, 0.1994, 0.2010))
])

# 테스트 데이터는 데이터 증강 없이 정규화만 수행
…

# CIFAR-10 학습/테스트 데이터셋 로드
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform_train)
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

➢Intro

•학습열기

•학습목표

➢학습하기

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

➢적용하기

➢Outro

•문제풀기

- 데이터 수는 증가하지 않지만 매 에폭마다 다양한 데이터로 학습

  예를 들어, RandomHorizontalFlip()은 50% 확률로 좌우 반전

  RandomRotation(10)은 -10°~+10° 사이에서 무작위 각도로 회전

  ColorJitter(...)는 무한히 다양한 조합의 밝기/채도/대비 변화를 생성함

- 위 조합은 epoch마다 다르게 적용되므로 같은 이미지가 매 epoch마다 다르게 보여짐

  예로, 고양이 이미지 하나

  → 첫 epoch: 좌우 반전 + 약간 밝게

  → 둘째 epoch: 원본 그대로

  → 셋째 epoch: 약간 회전 + 대비 낮춤

  → 넷째 epoch: 좌우 반전 + 회전

  → ...

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 옵티마이저를 설정

AdamW

Adam의 변형으로, 과적합(overfitting) 을 막기 위해 적용한 버전

더 안정적이고 일반화 잘 됨

net.parameters() → 네트워크의 학습 가능한 파라미터들을 optimizer에 전달

lr=0.001 → 초기 학습률 설정

구성 요소	설명
AdamW	L2 정규화가 개선된 Adam 옵티마이저
ReduceLROnPlateau	성능 개선이 정체되면 학습률을 자동으로 줄이는 스케줄러

```
criterion = nn.CrossEntropyLoss()  # 다중 클래스 분류용 손실 함수
optimizer = optim.AdamW(net.parameters(), lr=0.001)  # Adam의 정규화 강화 버전
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 학습률 조절 설정

ReduceLROnPlateau() 사용

손실 값이 줄어들지 않으면 학습률을 자동으로 줄여주는(이전의 10%로) 스케줄러

파라미터	의미
optimizer	조정 대상 옵티마이저
mode='min'	모니터링할 지표가 낮을수록 좋은 경우 (ex: loss)
patience=3	성능이 개선되지 않아도 3번 참았다가 학습률 줄임

```
criterion = nn.CrossEntropyLoss()  # 다중 클래스 분류용 손실 함수
optimizer = optim.AdamW(net.parameters(), lr=0.001)  # Adam의 L2 정규화 강화 버전
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=3)
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- ➢ **Intro**
  - •학습열기
  - •학습목표
- ➢ **학습하기**
  1. CNN 학습 최적화
  2. CNN 구조 최적화
  3. Dataset 최적화
- ➢ **적용하기**
- ➢ **Outro**
  - •문제풀기

간지

CNN 구조 최적화

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

➢ **Intro**

•학습열기

•학습목표

➢ **학습하기**

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

➢ **적용하기**

➢ **Outro**

•문제풀기

- CNN 네트워크 구조의 고도화
  - 합성곱 블록 1 = 합성곱 층 2개 + 최대풀링
  - 합성곱 블록 2 = 합성곱 층 2개 + 최대풀링
  - 합성곱 블록 3 = 합성곱 층 1개 + 최대풀링
  - 완전 연결 층 = ANN 1개(256 * 4 * 4 -> 512) + 10개 클래스 분류 층(512 -> 10))
- 필터는 3으로 고정

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 특징 추출 부(feature extractor)

  이미지 입력을 받아 점점 더 복잡한 특징을 추출하도록 설계된 합성곱 계층 블록

  3채널(RGB) 입력 이미지를 64 채널의 특징 맵(feature map)으로 변환

  커널 크기: 3×3 → 공간적으로 인접한 9픽셀 기반 특징 추출

  padding=1 → 출력 크기를 입력과 동일하게 유지 (32×32 → 32×32)

```
class ImprovedCNN(nn.Module):
    def __init__(self):
        super(ImprovedCNN, self).__init__()

        # 특징 추출을 위한 Convolutional Layer 블록 정의
        self.features = nn.Sequential(
            # 첫 번째 블록: 3채널 입력 → 64채널 출력
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Dropout(0.25),
```

용어설명

내레이션	

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 두 번째 블록과 세 번째 블록

  합성곱 블록 2 = 합성곱 층 2개 + 최대풀링

  합성곱 블록 3 = 합성곱 층 1개 + 최대풀링

```
# 두 번째 블록: 64채널 → 128채널
nn.Conv2d(64, 128, kernel_size=3, padding=1),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Conv2d(128, 128, kernel_size=3, padding=1),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2),
nn.Dropout(0.25),

# 세 번째 블록: 128채널 → 256채널
nn.Conv2d(128, 256, kernel_size=3, padding=1),
nn.BatchNorm2d(256),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2),
nn.Dropout(0.3)
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- CNN 합성곱 블록 구조 요약

블록	계층 종류	입력 채널	출력 채널	커널 크기	출력 크기 변화	역할 요약
1	Conv2d	3	64	3×3	32×32 → 32×32	기본 특징 추출
	BatchNorm2d	-	64	-	동일	정규화
	ReLU	-	-	-	동일	비선형성 추가
	Conv2d	64	64	3×3	32×32 → 32×32	특징 심화
	BatchNorm2d	-	64	-	동일	정규화
	ReLU	-	-	-	동일	비선형성 추가
	MaxPool2d	-	-	2×2	32×32 → 16×16	다운샘플링
	Dropout(0.25)	-	-	-	동일	과적합 방지
2	Conv2d	64	128	3×3	16×16 → 16×16	중간 수준 특징 추출
	BatchNorm2d	-	128	-	동일	정규화
	ReLU	-	-	-	동일	비선형성 추가
	Conv2d	128	128	3×3	16×16 → 16×16	특징 심화
	BatchNorm2d	-	128	-	동일	정규화
	ReLU	-	-	-	동일	비선형성 추가
	MaxPool2d	-	-	2×2	16×16 → 8×8	다운샘플링
	Dropout(0.25)	-	-	-	동일	과적합 방지
3	Conv2d	128	256	3×3	8×8 → 8×8	고수준 특징 추출
	BatchNorm2d	-	256	-	동일	정규화
	ReLU	-	-	-	동일	비선형성 추가
	MaxPool2d	-	-	2×2	8×8 → 4×4	최종 다운샘플링
	Dropout(0.3)	-	-	-	동일	과적합 방지

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 완전 연결층 구성

계층 종류	입력 크기	출력 크기	설명
Linear	256×4×4 = 4096	512	특징 맵 flatten 후 FC 연결
ReLU	-	-	활성화 함수
Dropout	-	-	과적합 방지 (50%)
Linear	512	10	CIFAR-10 클래스 수만큼 출력

```
# 전결합층 (Flatten 후 Classifier)
self.classifier = nn.Sequential(
    nn.Linear(256 * 4 * 4, 512),  # Conv 결과를 FC 입력으로 변환
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(512, 10)            # CIFAR-10 클래스 수만큼 출력
)
```

용어설명

내레이션

• CNN 구조 요약 결과

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = ImprovedCNN().to(device)   # 모델을 GPU 또는 CPU로 이동
summary(net, (3, 32, 32))        # 모델 구조 요약 출력
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
       BatchNorm2d-2           [-1, 64, 32, 32]             128
              ReLU-3           [-1, 64, 32, 32]               0
            Conv2d-4           [-1, 64, 32, 32]          36,928
       BatchNorm2d-5           [-1, 64, 32, 32]             128
              ReLU-6           [-1, 64, 32, 32]               0
         MaxPool2d-7           [-1, 64, 16, 16]               0
           Dropout-8           [-1, 64, 16, 16]               0
            Conv2d-9          [-1, 128, 16, 16]          73,856
      BatchNorm2d-10          [-1, 128, 16, 16]             256
             ReLU-11          [-1, 128, 16, 16]               0
           Conv2d-12          [-1, 128, 16, 16]         147,584
      BatchNorm2d-13          [-1, 128, 16, 16]             256
             ReLU-14          [-1, 128, 16, 16]               0
        MaxPool2d-15            [-1, 128, 8, 8]               0
          Dropout-16            [-1, 128, 8, 8]               0
           Conv2d-17            [-1, 256, 8, 8]         295,168
      BatchNorm2d-18            [-1, 256, 8, 8]             512
             ReLU-19            [-1, 256, 8, 8]               0
        MaxPool2d-20            [-1, 256, 4, 4]               0
          Dropout-21            [-1, 256, 4, 4]               0
           Linear-22                  [-1, 512]       2,097,664
             ReLU-23                  [-1, 512]               0
          Dropout-24                  [-1, 512]               0
           Linear-25                   [-1, 10]           5,130
================================================================
Total params: 2,659,402
Trainable params: 2,659,402
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 5.32
Params size (MB): 10.14
Estimated Total Size (MB): 15.48
----------------------------------------------------------------
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- ➢ **Intro**
  - •학습열기
  - •학습목표
- ➢ **학습하기**
  1. CNN 학습 최적화
  2. CNN 구조 최적화
  3. Dataset 최적화
- ➢ **적용하기**
- ➢ **Outro**
  - •문제풀기

간지

Dataset 최적화

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

➢ **Intro**

- 학습열기
- 학습목표

➢ **학습하기**

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

➢ **적용하기**

➢ **Outro**

- 문제풀기

- 클래스 불균형(class imbalance) 문제란?

  클래스 불균형 문제는 어떤 클래스의 데이터가 매우 많고,

  다른 클래스는 아주 적은 경우 발생하는 머신러닝 학습 문제

  모델이 자주 등장하는 클래스에만 편향되어 드문 클래스를 무시하거나 제대로 분류하지 못하는 문제 발생 가능성

  어떤 클래스는 이미지가 많고 어떤 클래스는 적어서 모델이 자주 나오는 클래스에 편향되기 쉬운 문제를 막기 위한 방법이 필요

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

➢Intro

•학습열기

•학습목표

➢학습하기

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

➢적용하기

➢Outro

•문제풀기

- 클래스 불균형(class imbalance) 문제 해결을 위한 준비

  Counter()로 각 클래스별 이미지 개수를 센 다음

  Counter({0: 5000, 1: 5000, 2: 5000, ..., 9: 5000})

  CIFAR-10은 클래스 균형이 잘 맞아 있는 편이지만

  커스텀 데이터셋이나 필터링을 거친 경우엔 불균형이 생길 수 있음

  적은 클래스에 더 높은 확률을 주는 방식으로 샘플링 가중치(class_weights)를 생성

```
# 클래스별 샘플 수 세기 → 불균형 데이터 대응용
targets = trainset.targets
class_count = Counter(targets)  # {클래스: 샘플 수}
class_weights = [1.0 / class_count[i] for i in targets]  # 빈도에 반비례한 가중치 생성
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

➢ **Intro**

•학습열기

•학습목표

➢ **학습하기**

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

➢ **적용하기**

➢ **Outro**

•문제풀기

- 불균형 문제를 해결하는 샘플링 전략 수립

  WeightedRandomSampler를 통해 데이터를 고르게 샘플링하는 전략

  class_weights에 따라 샘플을 확률적으로 선택

  num_samples=len(class_weights):→ 총 샘플 수는 원본 trainset과 동일하게 유지됨 (5만 개)

  즉, 전체 데이터 수는 유지하면서, 클래스 등장 비율만 조정

  replacement=True: 중복 허용 → 적은 클래스도 여러 번 뽑힐 수 있음

```
# 클래스별 샘플 수 세기 → 불균형 데이터 대응
targets = trainset.targets
class_count = Counter(targets)  # {클래스: 샘플 수}
class_weights = [1.0 / class_count[i] for i in targets]  # 빈도에 반비례한 가중치 생성
sampler = WeightedRandomSampler(class_weights, num_samples=len(class_weights),
replacement=True)

# WeightedRandomSampler를 이용한 불균형 보정 학습 데이터로더 생성
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, sampler=sampler,
num_workers=2)
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 수립된 샘플링 전략을 DataLoader에 설정

  DataLoader에 sampler를 지정하면, shuffle은 무시

  sampler는 우리가 만든 균형 샘플링 전략을 그대로 따름

  batch_size=128

  매 배치마다 다양한 클래스가 비슷한 비율로 포함되도록 샘플링됨

  num_workers=2

  데이터 준비 속도를 2배 병렬화해서 모델 학습 흐름을 끊기지 않게 만들어주는 설정

```
# 클래스별 샘플 수 세기 → 불균형 데이터 대응용
targets = trainset.targets
class_count = Counter(targets)  # {클래스: 샘플 수}
class_weights = [1.0 / class_count[i] for i in targets]  # 빈도에 반비례한 가중치 생성
sampler = WeightedRandomSampler(class_weights, num_samples=len(class_weights),
replacement=True)

# WeightedRandomSampler를 이용한 불균형 보정 학습 데이터로더 생성
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, sampler=sampler,
num_workers=2)
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 학습

에폭 수 30: - GPU 사용: 약 20분 소요, CPU 사용: 약 4~5시간 소요

```
epochs = 30
train_losses = []
test_accuracies = []

for epoch in range(epochs):
    net.train()
    running_loss = 0.0

    for i, (inputs, labels) in enumerate(trainloader):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()           # 이전 gradient 초기화
        outputs = net(inputs)           # 순전파
        loss = criterion(outputs, labels)  # 손실 계산
        loss.backward()                 # 역전파
        optimizer.step()                # 파라미터 갱신

        running_loss += loss.item()

    avg_loss = running_loss / len(trainloader)
    train_losses.append(avg_loss)

    # 테스트 정확도 평가
    …
```

용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 학습 시, 에폭마다 평가 실시

  학습 과정에서 train_losses, test_accuracies 계산해 저장

```
for epoch in range(epochs):
    # 학습 과정
    …
    avg_loss = running_loss / len(trainloader)
    train_losses.append(avg_loss)

    # 테스트 정확도 평가
    net.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in testloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = net(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    test_accuracies.append(accuracy)
    scheduler.step(avg_loss)

    print(f'[Epoch {epoch+1:2d}] Loss: {avg_loss:.3f} | Accuracy: {accuracy:.2f}%')
```

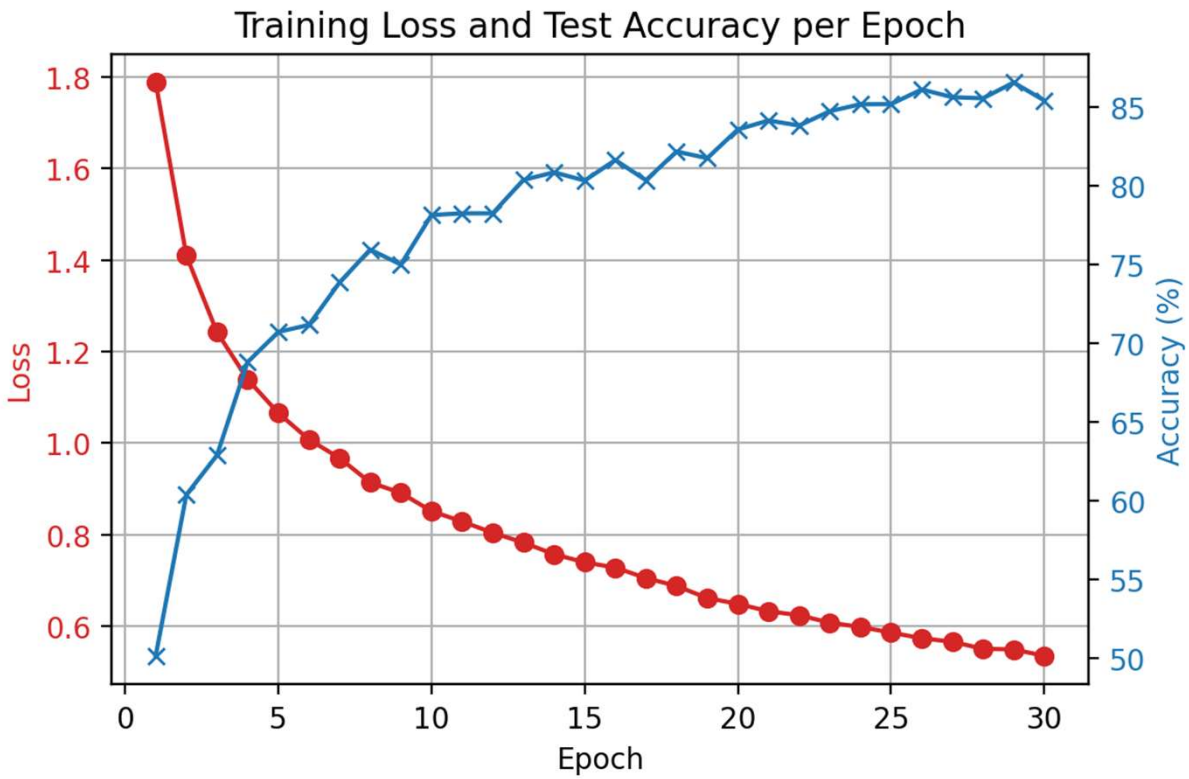
용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

- 손실과 정확도를 함께 시각화

  학습 과정에서 저장된 train_losses, test_accuracies를 사용



용어설명

내레이션

과정명	PyTorch로 배우는 머신러닝 알고리즘	회차명	10	화면설명

➢ **Intro**

•학습열기

•학습목표

➢ **학습하기**

1. CNN 학습 최적화
2. CNN 구조 최적화
3. Dataset 최적화

➢ **적용하기**

➢ **Outro**

•문제풀기

- PyTorch 기반의 CNN 이미지 분류 모델 학습 과정에서 다음과 같은 문제가 발생했다고 가정해 봅시다.

  학습 초기에는 손실이 빠르게 감소하지만 일정 에폭 이후에는 개선이 거의 없음

  모델이 특정 클래스에 과도하게 예측값을 집중함

  테스트 정확도는 일정 수준에서 정체되어 있음

1. ReduceLROnPlateau 스케줄러 적용: 손실 감소가 정체되는 경우, 학습률을 자동으로 낮춰서 더 미세한 학습을 유도함으로써 손실 감소 정체를 완화할 수 있습니다.
2. 데이터 증강 기법 강화: 좌우 반전, 회전, 밝기 조절 등의 데이터 증강을 적용하면 모델이 다양한 이미지 상황에 대응할 수 있어 과적합을 방지하고 테스트 정확도 정체를 완화할 수 있습니다.
3. 클래스 불균형 보정 (WeightedRandomSampler): 특정 클래스에 예측이 집중되는 문제는 데이터의 불균형에서 비롯되므로, 클래스별 샘플링 확률을 조정하여 균형 있게 학습하도록 유도할 수 있습니다.

① 학습 내용과 관련하여 실제 적용력을 높일 수 있는 문제, 혹은 주제를 작성해 주세요.

② ex. 사례 제시 후 전문가 의견, 실습과제, 응용 예시 시뮬레이션 등

③ 저작권 침해가 되지 않도록 내용을 구성해 주세요.

④ 출처가 있을 경우 반드시 작성해 주세요.

용어설명

내레이션