홍익대학교

Google Colab을 이용한 CNN 기본 코드의 성능 개선

Hanjin Cho

Electronic & Electrical Convergence Engineering
Hongik University
Republic of Korea

강의 내용

HONGIK UNIVERSITY

- 실습에 필요한 기본 개념
- 실습 준비
- CNN 학습 기본 코드 설명
- CNN 학습 기본 코드 실행 결과
- CNN 추론 기본 코드 설명
- CNN 추론 기본 코드 실행 방법 및 결과
- CNN 기본 코드 개선 (과제)

실습에 필요한 기본 개념

Deep Learning

HONGIK UNIVERSITY

▪ 딥러닝(Deep Learning) 이란?

- 정의
  - 인공지능(AI)의 한 분야.
  - 머신러닝(기계학습)의 하위 개념.
  - 인공신경망(Artificial Neural Network)을 여러 층(Layer) 쌓아올려 '깊게(Deep)' 만든 모델을 이용해 학습하는 방법.
- 원리
  - 입력 데이터(이미지, 음성, 텍스트 등)를 신경망에 넣음.
  - 각 층의 뉴런이 가중치(Weight)와 활성화 함수(Activation Function)를 통해 특징을 추출.
  - 오차(Error)를 계산하고, 역전파(Backpropagation)와 최적화 알고리즘(Optimizer)으로 가중치를 조정하며 학습.
- 특징
  - 사람이 직접 특징(Feature)을 설계하지 않아도 됨 → 자동 특징 학습.
  - 데이터와 연산 자원이 많을수록 성능이 좋아짐.
- 대표 모델
  - CNN (Convolutional Neural Network) → 영상 인식.
  - RNN (Recurrent Neural Network), LSTM, GRU → 시계열/텍스트.
  - Transformer, BERT, GPT → 자연어 처리, 생성형 AI.

딥러닝(심층학습)Deep Learning - YouTube

프로그래밍 및 실행 환경

HONGIK UNIVERSITY

- **Python**
  - 파이썬은 딥러닝 모델을 코딩하는 데 널리 사용되는 프로그래밍 언어로, 배우기 쉬운 문법과 직관적인 구조를 가지고 있음.
  - 다양한 딥러닝 프레임워크(PyTorch, TensorFlow 등)와 데이터 처리·시각화 라이브러리(NumPy, Pandas, Matplotlib 등)를 지원해 연구와 개발에 최적화되어 있음.
  - 코드가 간결해 아이디어를 빠르게 구현하고 수정할 수 있으며, 전 세계적으로 커뮤니티와 자료가 풍부해 학습과 활용이 용이.
  - Colab, Jupyter Notebook, VS Code 등 여러 실행 환경에서 동일하게 사용할 수 있어 호환성과 확장성이 뛰어나 최근 딥러닝 분야에서 가장 많이 사용되는 언어로 자리 잡음.
- **Google Colab**
  - 클라우드에서 Python을 실행할 수 있는 환경. 무료 GPU 지원, 드라이브와 연동 가능.
- **Jupyter Notebook 인터페이스**
  - Colab이 기반으로 하는 실행 방식. 셀 단위 실행, 출력 즉시 확인 가능 (Interpreter 기반).

주요 라이브러리

HONGIK UNIVERSITY

- **PyTorch (vs TensorFlow)**
  - Meta Platforms(구, Facebook) AI Research에서 개발한 오픈소스 딥러닝 프레임워크.
  - 동적 연산 그래프를 지원하여 직관적이고 유연한 모델 개발 가능.
  - 텐서 연산, 자동 미분, GPU 가속을 지원.
  - 다양한 신경망 구조를 간단히 구현할 수 있어 연구와 교육에 적합.
- **Torchvision**
  - PyTorch와 함께 제공되는 보조 라이브러리.
  - CIFAR10, MNIST, ImageNet 등 주요 데이터셋을 쉽게 불러올 수 있음.
  - ResNet, VGG, AlexNet 등 유명 모델을 사전 학습된 가중치와 함께 제공.
  - transforms 모듈을 통해 데이터 전처리와 증강 작업을 간단히 적용 가능.
- **PyTorch와 Torchvision의 실습 활용**
  - PyTorch로 CNN의 구조와 학습 과정을 직관적으로 이해할 수 있음.
  - Torchvision으로 데이터 불러오기와 전처리를 단순화해 교육 시간을 절약할 수 있음.
  - 두 라이브러리를 함께 사용하면 기초 딥러닝 실습을 효과적으로 진행 가능.

하드웨어 및 실행 장치

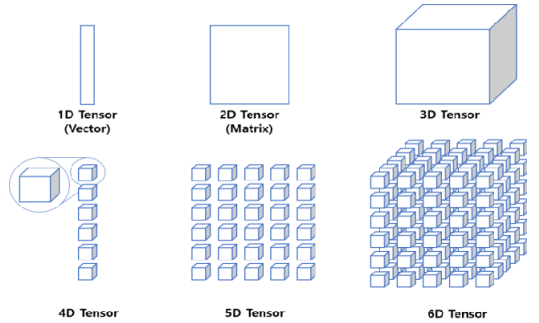
HONGIK UNIVERSITY

- **GPU (Graphics Processing Unit)**
  - 병렬 연산에 특화된 장치. 딥러닝 학습 속도를 CPU보다 크게 향상.
- **CUDA**
  - NVIDIA GPU에서 연산을 수행할 수 있게 하는 플랫폼.
    torch.cuda.is_available() 로 확인.
  - Colab을 이용하면 Colab 서버에 이미 CUDA가 설치되어 있고, NVIDIA GPU 드라이버와 환경 설정까지 다 되어 있어서, 학습자들은 따로 CUDA를 설치하거나 복잡하게 신경 쓸 필요 없음.
- **Device 설정**
  - 코드에서 CPU와 GPU 중 어떤 장치를 사용할지 선택.
    (device = torch.device("cuda" if torch.cuda.is_available() else "cpu"))

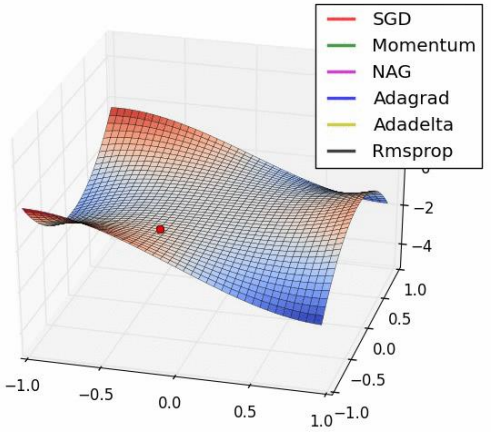
딥러닝 기본 개념

HONGIK UNIVERSITY

- **Tensor**
  - 다차원 배열. PyTorch에서 데이터와 파라미터를 표현하는 기본 단위.
- **Gradient / Backpropagation**
  - 손실 함수에 따라 가중치를 자동으로 업데이트하는 과정.



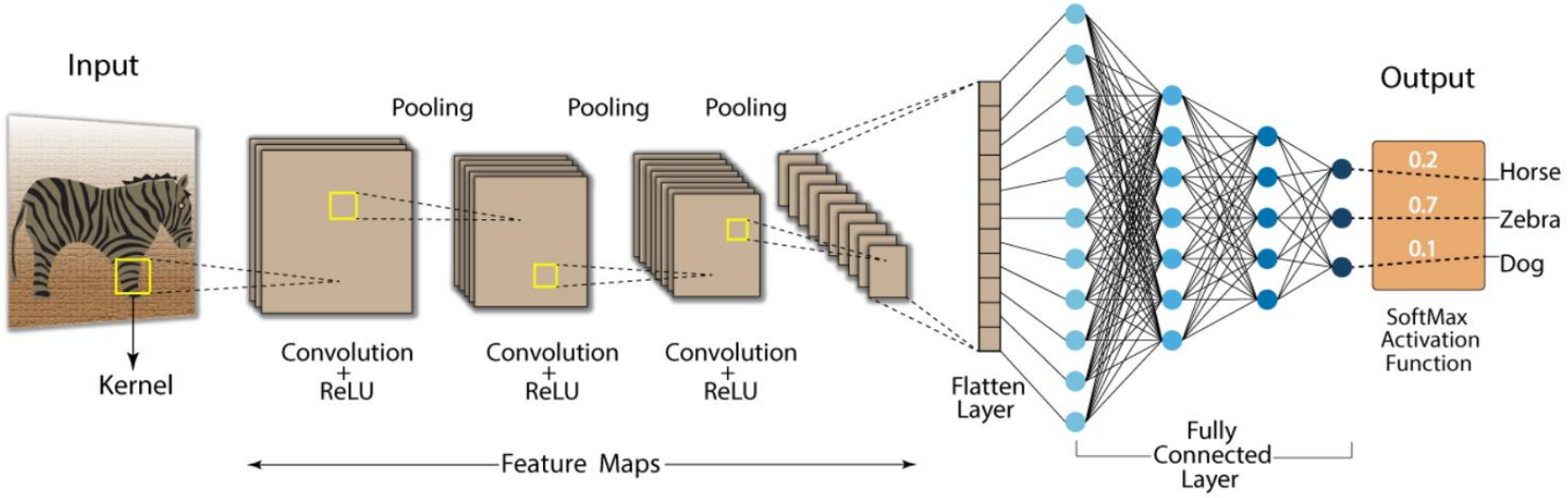
- **Optimizer**
  - 손실을 줄이기 위해 파라미터를 업데이트하는 알고리즘 (Adam, SGD 등)
- **Loss Function**
  - 모델의 예측과 정답 차이를 수치화하는 함수 (CrossEntropyLoss).



합성곱 신경망 핵심 개념

HONGIK UNIVERSITY

- **합성곱 신경망(CNN) 이란?**
  - CNN은 이미지 인식, 영상 분석, 객체 탐지 등 시각적 데이터를 처리하는 데 최적화된 딥러닝 구조임.
  - 데이터의 공간적 구조를 인식하고 학습할 수 있도록 설계됨.
  - 주요 구성
    - 합성곱 층(Convolutional Layer), 풀링 층(Pooling Layer), 완전연결 층(Fully Connected Layer).



합성곱 신경망 핵심 개념

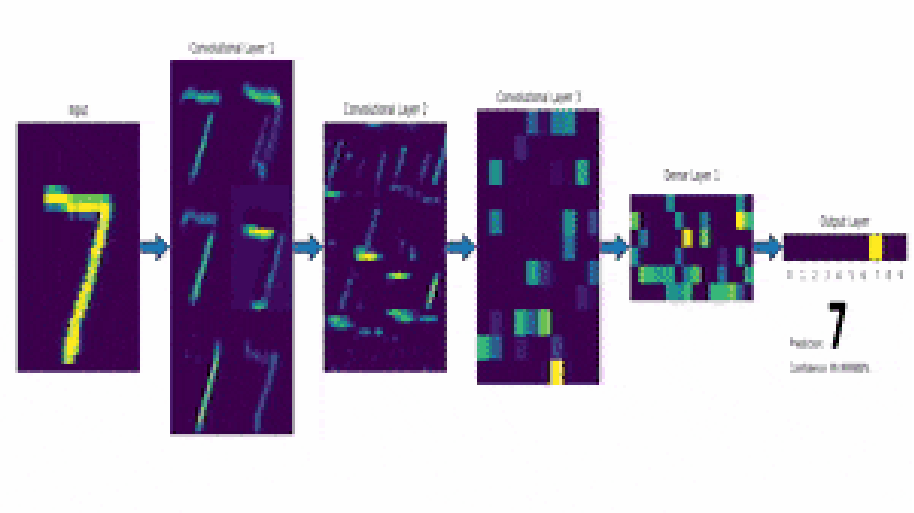
HONGIK UNIVERSITY

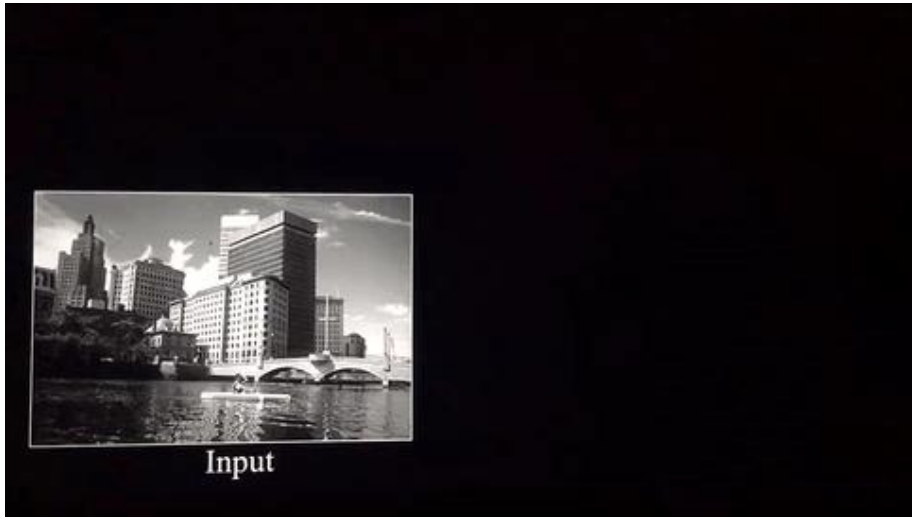
- **Convolution Layer**
  - 이미지의 특징(에지, 패턴)을 추출하는 핵심 연산.
- **Activation Function (ReLU, SoftMax)**
  - 비선형성을 추가해 복잡한 패턴 학습 가능하게 하거나 출력층에서 문제 유형에 따라 최종 출력을 결정.
- **Pooling Layer (MaxPool)**
  - 특징 맵 크기를 줄여 연산 효율과 불변성 확보.
- **Fully Connected Layer**
  - 최종적으로 분류를 수행하는 단계.

합성곱 신경망 핵심 개념

HONGIK UNIVERSITY

- **합성곱 층 (Convolutional Layer)**
  - 합성곱 층은 이미지의 공간 구조를 유지한 채 유용한 특징을 추출하는 층.
  - 입력 데이터에 필터(커널)를 적용해 합성곱 연산을 수행하고, 특징맵(feature map)을 생성.
  - 작은 영역을 슬라이딩하며 곱셈-합산을 수행해 국소적인 패턴을 감지.
  - 자주 사용하는 커널 크기: 1×1, 3×3, 5×5, 7×7 등.





합성곱 신경망 핵심 개념

HONGIK UNIVERSITY

- **합성곱 연산**
  - 입력 픽셀과 커널 간 곱셈 결과를 통해 특징을 추출하는 구조 시각화.
  - 각 커널은 선, 에지, 방향 등의 저수준 특징을 추출하도록 설계됨.
  - 여러 개의 커널이 동시에 다양한 특징을 병렬적으로 추출함.

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

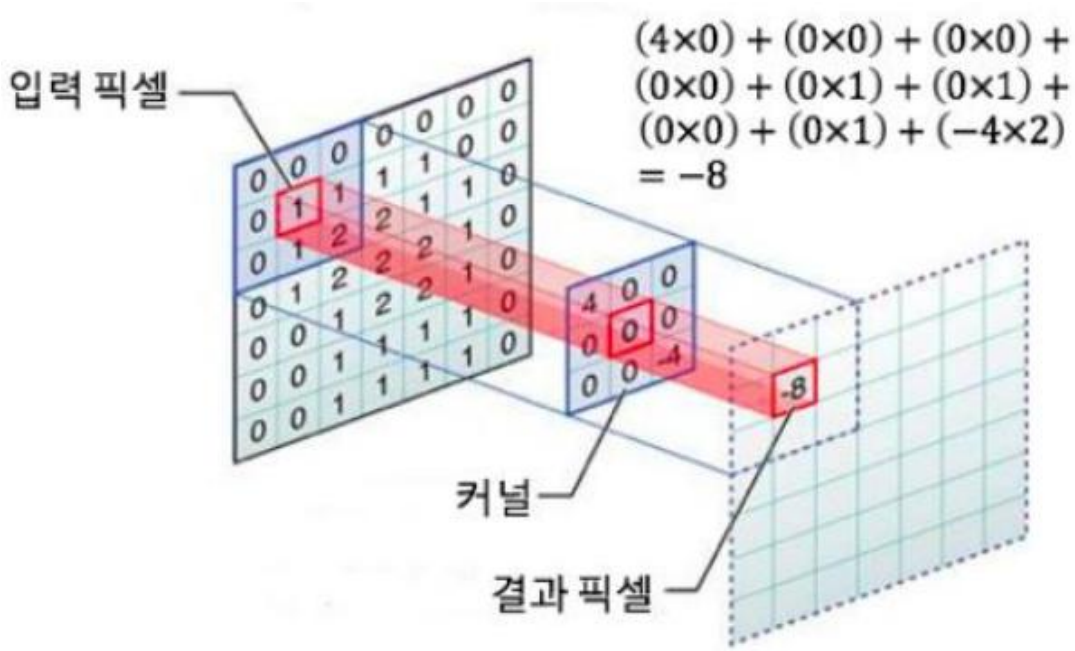
*

1	0	-1
1	0	-1
1	0	-1

=

6		

7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6



데이터셋 및 전처리

HONGIK UNIVERSITY

- **CIFAR-10**
  - 32×32 RGB 컬러 이미지, 10개 사물 클래스(비행기, 자동차, 개, 고양이 등).
- **Fashion-MNIST**
  - 28×28 흑백 이미지, 10개 의류 클래스.
- **DataLoader**
  - 배치 단위로 데이터를 공급하는 PyTorch 모듈.
- **Transforms**
  - 데이터 전처리 및 증강 도구 (Crop, Flip, Normalize 등).

학습 과정 이해

HONGIK UNIVERSITY

- **Epoch**
  - 전체 데이터셋을 1번 학습하는 단위.
- **Batch**
  - 학습할 때 한 번에 처리하는 데이터 묶음.
- **Validation Accuracy**
  - 모델이 학습 데이터가 아닌 검증 데이터에서 어느 정도 성능을 내는지 확인하는 지표.
- **Overfitting**
  - 학습 데이터에는 성능이 좋지만 새로운 데이터에는 성능이 떨어지는 현상.

하이퍼 파라미터 튜닝 및 모델 개선

HONGIK UNIVERSITY

- **하이퍼 파라미터란?**
  - 학습률, 배치 크기, epoch 수, 최적화 알고리즘 등 모델 성능에 직접 영향을 주는 값을 의미.
  - 학습 과정에서 자동으로 결정되지 않으므로 사용자가 직접 조정 필요.

- **하이퍼 파라미터 및 모델 개선의 목적과 방법**
  - 딥러닝 모델의 정확도 향상과 일반화 성능 강화.
    - (예) 기본 CNN이 CIFAR10에서 60~70 퍼센트 수준 성능을 내는 경우, 튜닝과 개선을 통해 성능 향상 가능.
  - 하이퍼 파라미터 튜닝 방법
    - 다양한 조합을 실험하며 가장 성능이 좋은 값을 탐색하여 적용.
  - 모델 개선 방법
    - 데이터 증강 기법 적용으로 데이터 다양성 확보 및 과적합 방지.
    - 합성곱 층 수나 채널 수 변경 등 모델 구조 조정.
    - Dropout 같은 규제 기법 활용.

- **하이퍼 파라미터 및 모델 개선의 교육적 의미**
  - 단순히 결과만 보는 것이 아니라 어떤 이유에 의해 성능이 향상되는지 원리 이해 가능.
  - 반복적 실험과 개선 과정을 통해 연구자가 실제 딥러닝 코드의 성능을 향상하는 방식 학습.

실습 준비

비전공자가 AI 코드를 배우는 이유

HONGIK UNIVERSITY

- **AI가 데이터를 이해하고 판단하는 원리를 배우는 것**
  - 대부분의 AI 서비스는 CNN과 같은 기본 모델 구조에서 출발하기 때문에 이 원리를 이해하면 업무나 일상에 적용할 수 있는 아이디어를 떠올릴 수 있음.
  - 즉, AI에서의 **데이터 중요성**을 이해하고, **기본 용어와 개념**을 익히며, **학습과 추론의 구조**를 자연스럽게 이해할 수 있기 때문에 AI 모델의 구조를 이해하면 **나에게 최적화된 AI 서비스를 설계**하거나, **기존 AI 서비스를 더욱 효과적으로 활용**할 수 있음.

Google Colab

HONGIK UNIVERSITY

- **Google Colab 이란?**
  - 구글에서 무료로 제공하는 클라우드 기반 주피터 노트북 환경.
  - 웹 브라우저만 있으면 파이썬 코드 작성과 실행 가능.
  - GPU와 TPU 같은 고성능 연산 자원을 무료 또는 저렴하게 제공.
  - 개인 PC 성능이 낮아도 딥러닝 학습과 같은 연산 작업 수행 가능.
  - 구글 드라이브와 연동되어 작업 결과가 자동 저장되고 공유도 용이.
  - TensorFlow, PyTorch, Scikit-learn, OpenCV 등 주요 라이브러리가 기본 포함되어 바로 실습 진행 가능.

※ 해당 강의에서는 Google Colab을 이용하여 CNN 코드 실습 진행.

Google 계정 생성

HONGIK UNIVERSITY

▪ Google 계정 생성 방법

- Google Colab 사용을 위해서는 Google 계정이 필요.
- 기존 계정이 있다면 사용 가능하며 만약 없다면 아래의 절차에 따라 계정 생성.

- 계정 생성 절차
  - 웹 브라우저에서 accounts.google.com/signup 접속
  - 이름, ID(이메일 주소), 비밀번호 입력
  - 휴대폰 번호 인증, 보안 옵션 설정, 약관 동의 절차 진행

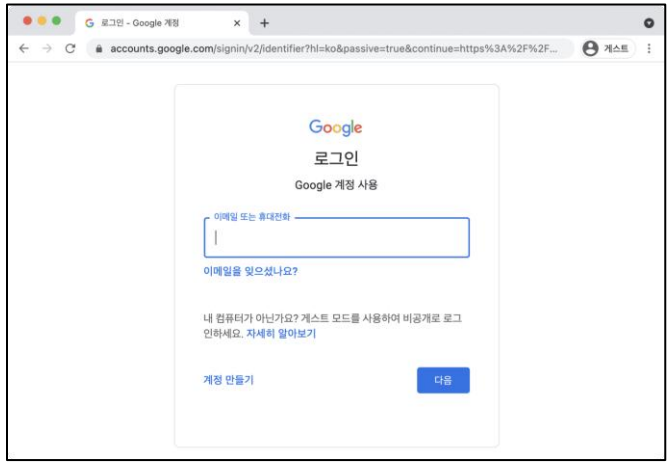
- 계정 생성 후 Google Colab을 비롯하여 Gmail, Google Drive, YouTube 등 다양한 서비스 사용 가능.
- 계정이 준비되면 Colab 노트북을 열고 실습 진행.

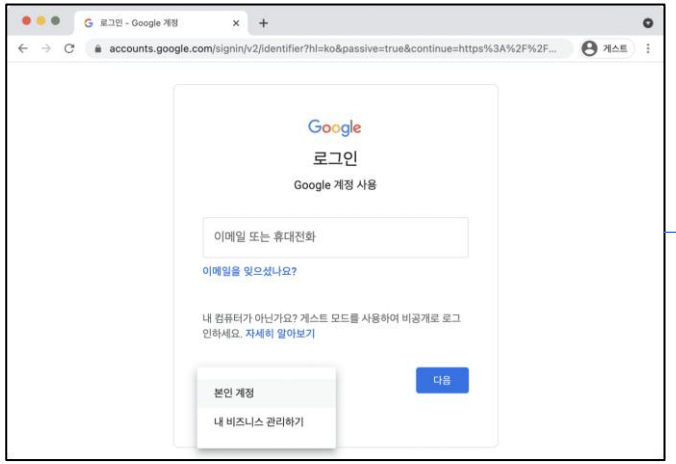
Google 계정 생성

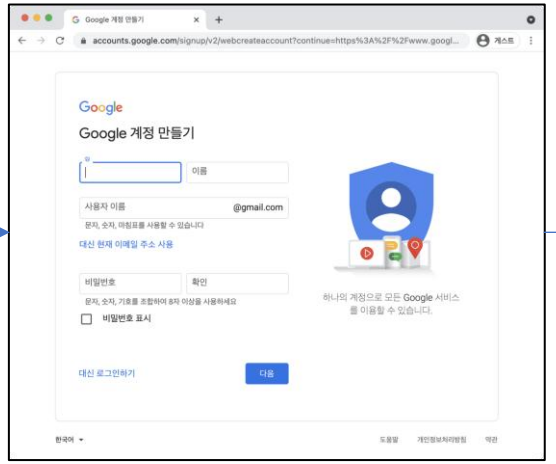
HONGIK UNIVERSITY

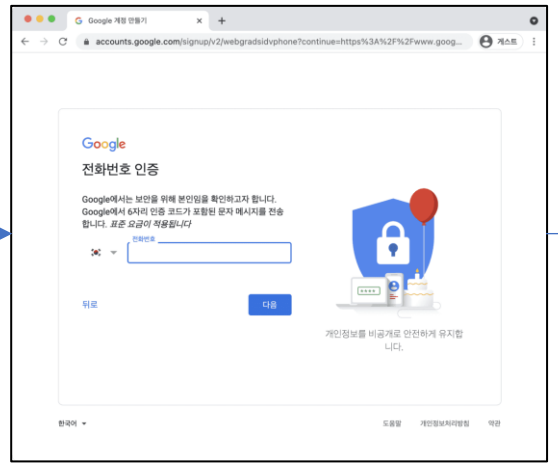
- **Google 계정 생성 과정**

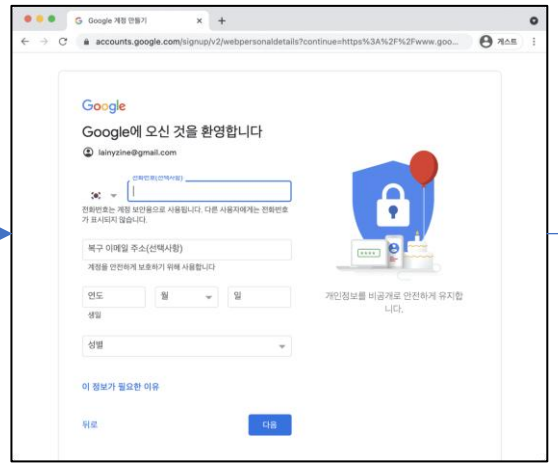
https://www.lainyzine.com/ko/article/how-to-create-google-account/

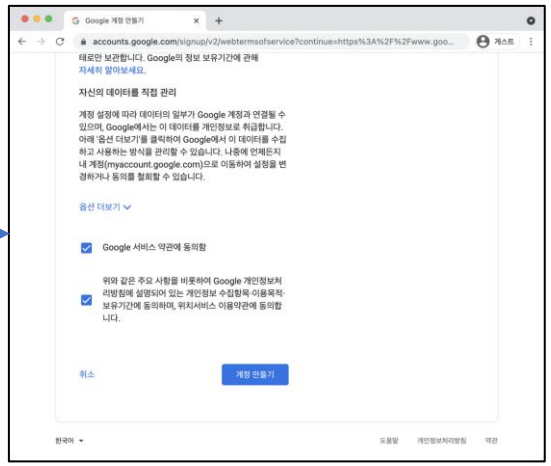












Google Colab 사용 방법

HONGIK UNIVERSITY

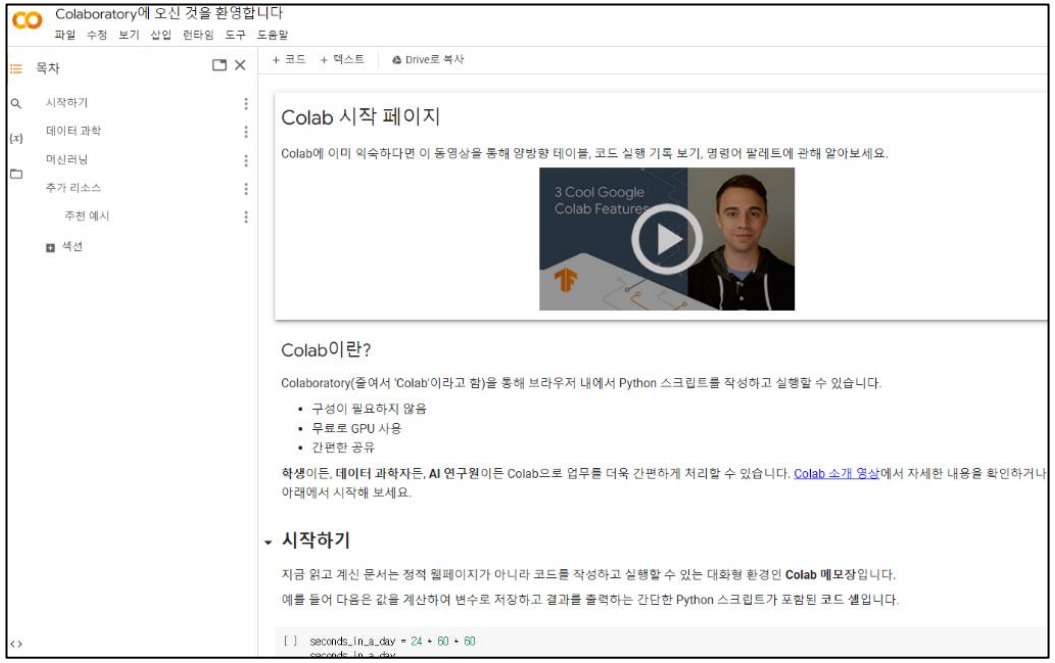
- **Colab 접속**
  - colab.research.google.com 주소 입력 후 접속
- **노트북 생성**
  - 메뉴에서 [파일 → 새 노트] 선택
- **실행 환경 설정**
  - 메뉴에서 [런타임 → 런타임 유형 변경] 선택
    - 런타임 유형 : Python 3
    - 하드웨어 가속기 : T4 GPU
    - 런타임 버전 : 최선 버전(권장)
- **코드 실행**
  - 코드 셀에 작성 후 Shift Enter 키로 실행
- **파일 관리**
  - Colab과 Google Drive 연동으로 노트북 저장 및 불러오기 가능

※ 해당 강의에서는 미리 생성된 Colab 노트북 링크를 제공하고 파일 메뉴에서 드라이브에 복사하여 개인 환경에서 실행 예정.

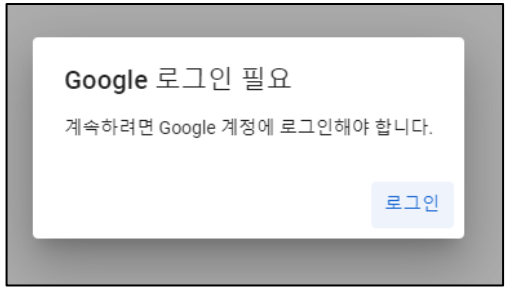
Google Colab 사용 방법

HONGIK UNIVERSITY

- 새 노트 열기



[Colab 시작 화면]



[새 노트 열기]

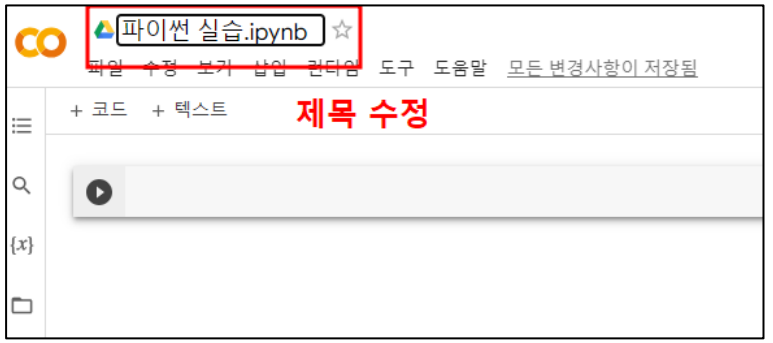
▲ 실행 상황에 따라 Google 로그인 필요

Google Colab 사용 방법

HONGIK UNIVERSITY

- 간단한 코드 실행

[새노트북 시작 화면]



[파일명 수정]

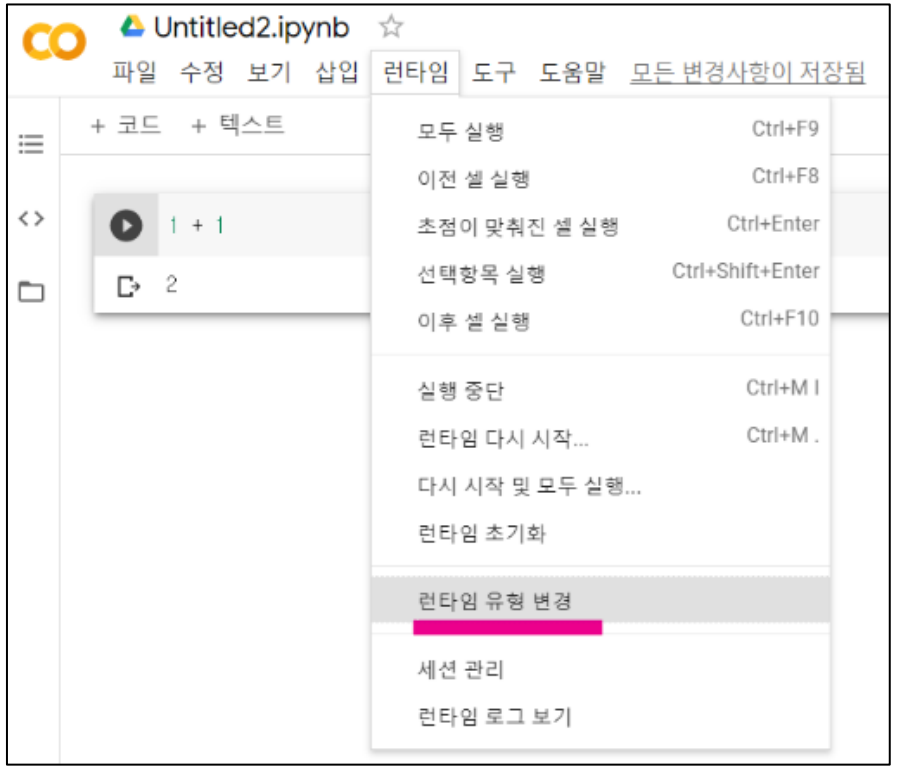
[간단한 코드 실행]

▲ 코드 왼쪽에 실행 버튼을 누르거나 Shift+Enter로 코드 실행

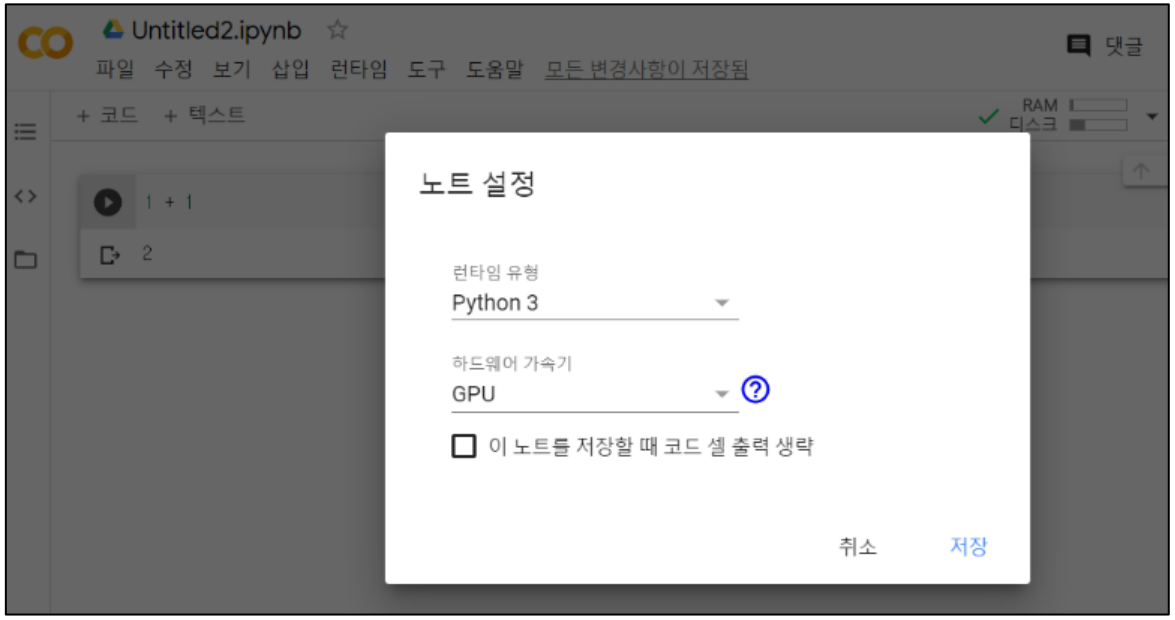
Google Colab 사용 방법

HONGIK UNIVERSITY

- 실행 환경 설정



[런타임 유형 변경]



[실행 환경 설정]

CNN 학습 기본 코드 설명

CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

- **TinyCNN 코드 링크**
  - https://colab.research.google.com/drive/1dhhfvRFSqMlOmQx2OCM7FJRasOreAKg-?usp=sharing

※ 주의

위 링크 실행 후
파일 → Drive에 사본 저장 → 기존 코드 창 닫기

```
# 0) 환경
!pip -q install torch torchvision

import torch, torch.nn as nn, torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("device:", device)

from tqdm.auto import tqdm
import matplotlib.pyplot as plt

train_losses, val_accs = [], []

# 1) 설정
USE_CIFAR10 = True    # False로 바꾸면 Fashion-MNIST
EPOCHS = 5
BATCH = 128
LR = 3e-3

# 2) 데이터
if USE_CIFAR10:
    mean, std = (0.4914,0.4822,0.4465), (0.2023,0.1994,0.2010)
    train_tf = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(), transforms.Normalize(mean, std),
    ])
    test_tf  = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean, std)])
    train_ds = datasets.CIFAR10("./data", train=True,  download=True, transform=train_tf)
    test_ds  = datasets.CIFAR10("./data", train=False, download=True, transform=test_tf)
    in_ch, num_classes = 3, 10
else:
    mean, std = (0.2861,), (0.3530,)
    train_tf = transforms.Compose([
        transforms.RandomCrop(28, padding=3),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(), transforms.Normalize(mean, std),
    ])
    test_tf  = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean, std)])
    train_ds = datasets.FashionMNIST("./data", train=True,  download=True, transform=train_tf)
    test_ds  = datasets.FashionMNIST("./data", train=False, download=True, transform=test_tf)
    in_ch, num_classes = 1, 10

train_ld = DataLoader(train_ds, batch_size=BATCH, shuffle=True,  num_workers=0)
test_ld  = DataLoader(test_ds,  batch_size=BATCH, shuffle=False, num_workers=0)
```

```
# 3) 모델
class TinyCNN(nn.Module):
    def __init__(self, in_ch=3, ncls=10):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_ch, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),   # /2
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),      # /4
        )
        # CIFAR: 32x32 -> 8x8, FMNIST: 28x28 -> 7x7
        fc_in = 64 * (8 if in_ch==3 else 7) * (8 if in_ch==3 else 7)
        self.fc  = nn.Sequential(
            nn.Flatten(),
            nn.Linear(fc_in, 128), nn.ReLU(),
            nn.Linear(128, ncls)
        )
    def forward(self, x): return self.fc(self.conv(x))

model = TinyCNN(in_ch=in_ch, ncls=num_classes).to(device)
opt = torch.optim.Adam(model.parameters(), lr=LR)
criterion = nn.CrossEntropyLoss()
```

```
# 4) 학습/평가 루프
def eval_acc():
    model.eval(); correct=total=0
    with torch.no_grad():
        for x,y in test_ld:
            x,y = x.to(device), y.to(device)
            pred = model(x).argmax(1)
            correct += (pred==y).sum().item()
            total += y.size(0)
    return correct/total

for ep in range(1, EPOCHS+1):
    model.train()
    running_loss = 0.0

    # 에폭 진행률만 실시간 표시
    pbar = tqdm(train_ld, desc=f"Epoch {ep}/{EPOCHS}", leave=False)
    for x,y in pbar:
        x,y = x.to(device), y.to(device)
        logits = model(x)
        loss = criterion(logits, y)
        opt.zero_grad(); loss.backward(); opt.step()
        running_loss += loss.item()

        pbar.set_postfix(loss=f"{loss.item():.4f}")

    # 에폭 종료 후 검증
    avg_loss = running_loss / len(train_ld)
    acc = eval_acc()
    train_losses.append(avg_loss)
    val_accs.append(acc)

    print(f"[{ep}/{EPOCHS}] loss={avg_loss:.4f}  val_acc={acc:.3f}")


# --- 학습 종료 후 그래프 출력 ---
plt.figure(figsize=(6,4))
plt.plot(train_losses, label="Train Loss")
plt.plot(val_accs, label="Val Accuracy")
plt.xlabel("Epoch"); plt.ylabel("Value")
plt.title("Training Curves")
plt.legend(); plt.grid(True)
plt.show()
```

CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

▪ 하이퍼 파라미터 설정

- 하이퍼파라미터란 모델 학습 과정에서 사용자가 직접 정해주는 값으로, 학습 성능과 속도에 큰 영향을 줌.
  - EPOCHS는 전체 학습 데이터를 몇 번 반복할지 결정.
  - BATCH는 한 번에 모델에 입력되는 데이터 묶음 크기, 학습 속도와 메모리에 영향.
  - LR은 학습률, 가중치를 업데이트하는 크기. 너무 크면 발산, 너무 작으면 수렴이 느려짐.

```
EPOCHS = 5
BATCH = 128
LR = 3e-3
```

▪ 데이터셋 불러오기

- 데이터셋은 모델이 학습하고 평가하는 데 사용하는 입력 데이터와 정답 라벨의 집합을 의미.
  - torchvision.datasets 모듈을 사용해 CIFAR10 데이터셋 자동 다운로드 후 ./data 폴더에 저장.
  - download=True 옵션으로 처음 실행 시만 내려받음.
  - transform 매개변수로 데이터 전처리와 증강 적용.

```
train_ds = datasets.CIFAR10("./data", train=True,  download=True, transform=train_tf)
test_ds  = datasets.CIFAR10("./data", train=False, download=True, transform=test_tf)
```

CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

▪ 자동 다운로드

- 자동 다운로드는 코드 실행 시 필요한 데이터셋을 인터넷에서 내려받아 저장.
  - 처음 실행 시 인터넷에서 CIFAR10 데이터셋을 내려받아 ./data 폴더에 저장.
  - 이후 실행부터는 저장된 데이터를 불러옴, 다시 다운로드하지 않음.

```
train_ds = datasets.CIFAR10("./data", train=True,  download=True, transform=train_tf)
```

▪ 데이터 로더

- 데이터 로더는 데이터셋을 학습에 적합한 배치 단위로 잘라서 모델에 공급하는 도구.
  - DataLoader는 데이터를 batch 단위로 모델에 공급.
  - train DataLoader는 shuffle=True로 매 epoch마다 데이터 순서를 섞음 → 과적합 방지.
  - test DataLoader는 shuffle=False로 항상 같은 순서로 평가.

```
train_ld = DataLoader(train_ds, batch_size=BATCH, shuffle=True,  num_workers=0)
test_ld  = DataLoader(test_ds,  batch_size=BATCH, shuffle=False, num_workers=0)
```

CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

- **실습에서의 편의성**
  - 자동 다운로드와 DataLoader 로 인해 학습자들은 데이터 준비 과정을 직접 할 필요 없음.
  - 코드 실행만으로 데이터셋이 자동 준비.
  - 따라서, 학습자들은 CNN 구조와 학습 과정 이해에 집중 가능.

CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

▪ 모델 정의

- 모델은 입력 데이터를 받아 출력을 생성하는 신경망 구조로, 합성곱 신경망은 이미지 처리에 적합한 구조를 가지고 있음.
  - 합성곱(conv), 활성화(ReLU), 풀링(MaxPool)을 반복해 특징 추출.
  - 입력 이미지 크기를 점차 줄여가며 더 추상적인 특징을 학습.
  - Flatten으로 1차원으로 바꾼 뒤 Linear 층에서 분류 수행.
  - 간단한 구조지만 CNN의 기본 원리를 담고 있음.

```python
class TinyCNN(nn.Module):
    def __init__(self, in_ch=3, ncls=10):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_ch, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),    # /2
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),       # /4
        )
        # CIFAR: 32x32 -> 8x8, FMNIST: 28x28 -> 7x7
        fc_in = 64 * (8 if in_ch==3 else 7) * (8 if in_ch==3 else 7)
        self.fc  = nn.Sequential(
            nn.Flatten(),
            nn.Linear(fc_in, 128), nn.ReLU(),
            nn.Linear(128, ncls)
        )
    def forward(self, x): return self.fc(self.conv(x))
```

◀ ch : 입력 채널 수(3:컬러, 1:그레이), ncls : 분류할 클래스 수

◀ 합성곱 신경망 계층 : 입력 이미지를 받아 특징맵 추출

◀ CIFAR-10의 경우 입력 3×32×32을 받아 64×8×8 특징맵 출력
① 3×32×32 → 32×16×16 ② 32×16×16 → 64×8×8
◀ Fashion-MNIST의 경우 입력 1×28×28을 받아 64×7×7 특징맵 출력
① 1×28×28 → 32×14×14 ② 32×14×14 → 64×7×7

◀ 합성곱 결과를 펼친 후 입력되는 fully connected layer 크기 설정.

◀ 합성곱 신경망에서 추출한 특징맵을 펼치고(Flatten)
완전연결층(Linear)을 이용해 최종 클래스 확률을 출력하는 분류기(Classifier)

▶ 즉, TinyCNN 모델의 구조는 conv에서 특징맵을 추출하고 fc를 실행하여
총 10개(ncls)의 클래스로 구분하는 구조.

CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

- **손실함수와 옵티마이저**
  - 손실함수는 예측과 실제 값 차이를 수치로 표현하고, 옵티마이저는 이 손실을 줄이도록 모델 파라미터를 조정하는 역할.
    - CrossEntropyLoss는 '분류' 문제에서 널리 사용되는 손실 함수.
    - Adam 옵티마이저는 적응적 학습률을 사용해 빠르고 안정적인 학습을 지원.

```
opt = torch.optim.Adam(model.parameters(), lr=LR)
criterion = nn.CrossEntropyLoss()
```

CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

▪ 학습 및 평가 루프

- 학습 루프는 데이터를 반복 학습하는 과정이고, 평가 루프는 학습된 모델의 성능을 확인하는 과정.
  - 학습(train) 단계에서 손실 계산, 역전파 (backpropagation), 가중치 업데이트 수행.
  - 평가(eval) 단계에서 test set으로 정확도 계산.
  - 각 epoch이 끝날 때마다 검증 정확도를 출력해 성능 변화를 확인.

```
for ep in range(1, EPOCHS+1):  ◀ 설정된 epoch 동안 반복 학습
    model.train()
    running_loss = 0.0

    # 에폭 진행률만 실시간 표시
    pbar = tqdm(train_ld, desc=f"Epoch {ep}/{EPOCHS}", leave=False)
    for x,y in pbar:
        x,y = x.to(device), y.to(device) ◀ 데이터를 장치(GPU/CPU)로 이동
        logits = model(x)        ◀ 모델의 예측 출력
        loss = criterion(logits, y)     ◀ 예측 정답과 비교 : 손실 계산
        opt.zero_grad(); loss.backward(); opt.step() ◀ 이전 단계의 기울기 최소화
                                                      역전파로 기울기 계산
                                                      옵티마이저가 가중치 업데이트
        running_loss += loss.item()
              ▲ 이번 epoch 동안의 총 손실값 누적

        pbar.set_postfix(loss=f"{loss.item():.4f}")

    # 에폭 종료 후 검증
    avg_loss = running_loss / len(train_ld) ◀ 평균 손실
    acc = eval_acc()       ◀ 검증 정확도 계산
    train_losses.append(avg_loss)
    val_accs.append(acc)

    print(f"[{ep}/{EPOCHS}] loss={avg_loss:.4f}  val_acc={acc:.3f}")
              ▲ 현재 epoch 결과 출력
```

▶ 결론적으로 데이터 배치를 모델에 넣어 손실을 구하고, 역전파와 최적화를 통해 가중치를 업데이트하면서 학습 진행. 각 epoch마다 평균 손실과 정확도를 기록

CNN 학습 기본 코드 실행 결과

CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

▪ CNN 기본 코드 구조

- USE_CIFAR10 변수를 True/False로 바꿔 CIFAR-10과 Fashion-MNIST 의 데이터셋 중 하나를 선택하여 실행.

- **CIFAR-10**
  - 구성
    - → 사물 데이터를 모아 놓은 컬러 이미지 데이터셋, 크기 32×32 픽셀.
    - → 클래스는 10종류 : 비행기, 자동차, 새, 고양이, 사슴, 개, 개구리, 말, 배, 트럭.
    - → 학습용 데이터 5만 장, 테스트용 데이터 1만 장으로 구성.
  - 코드 실행 결과
    - → TinyCNN 기본 구조로 5 epoch 학습 → 약 60~70% 정확도.
    - → 데이터가 컬러이고 다양한 물체 클래스가 있어 작은 네트워크로는 높은 성능을 내기 어려움.
    - → 따라서 모델 개선이나 데이터 증강을 통해 성능을 더 끌어올려야 함.
  - 출력 결과
    - → 모델은 입력된 이미지를 10개 클래스 중 하나로 분류.
    - → 최종 결과는 각 클래스별 확률 분포(softmax 출력)와, 그중 가장 높은 값을 갖는 클래스 예측.

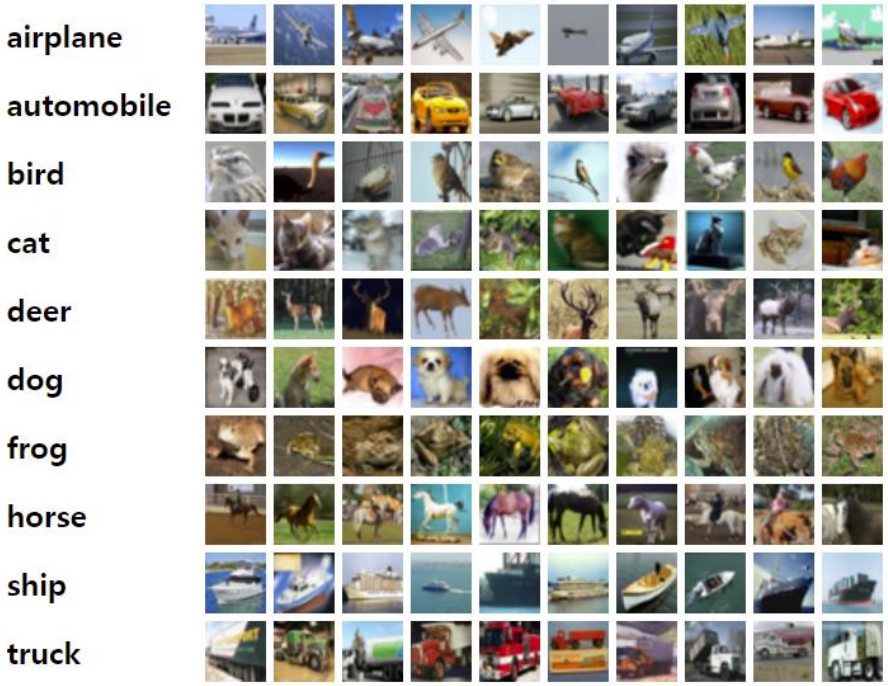
- **Fashion-MNIST**
  - 구성
    - → Fashion과 관련된 데이터를 모아놓은 흑백 이미지 데이터셋, 크기 28×28 픽셀.
    - → 클래스는 10종류 : 티셔츠/탑, 바지, 풀오버, 드레스, 코트, 샌들, 셔츠, 스니커즈, 가방, 앵클부츠.
    - → 학습용 데이터 6만 장, 테스트용 데이터 1만 장으로 구성.
  - 코드 실행 결과
    - → 동일한 TinyCNN 구조로 학습 시 85% 이상 정확도 달성.
    - → 데이터가 상대적으로 단순하고 클래스 간 차이가 명확해 CIFAR10보다 높은 성능을 얻음.
  - 출력 결과
    - → 모델은 입력된 패션 아이템 이미지를 10개 클래스 중 하나로 분류.
    - → 최종 결과는 softmax 확률 분포와 예측된 의류 클래스.

CNN 학습 기본 코드 설명

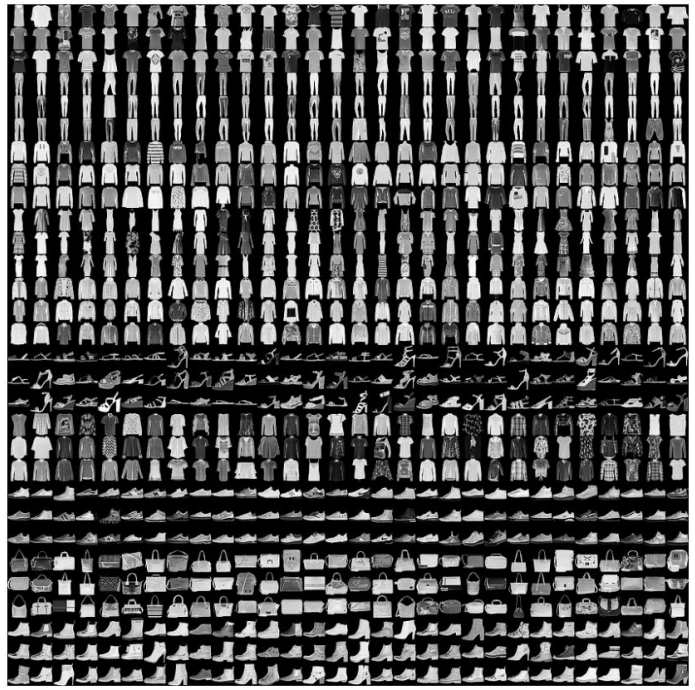
HONGIK UNIVERSITY

- **CIFAR-10 과 Fashion-MNIST 비교**

데이터셋	이미지 크기	채널	클래스 수	학습/테스트 수	특징
CIFAR-10	32x32	컬러(RGB)	10	50,000/10,000	작은 컬러 이미지와 다양한 클래스 구성, 학습 난이도 중간 이상
Fashion-MNIST	28x28	흑백	10	60,000/10,000	단순한 구조, 빠른 모델 학습과 결과 확인 가능 학습 난이도 낮음



[CIFAR-10]



[Fashion-MNIST]

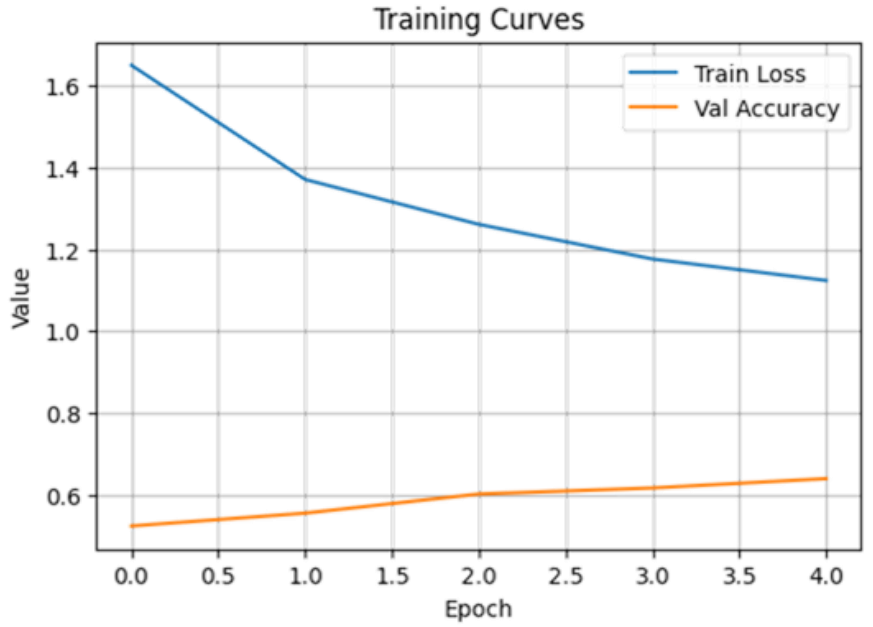
CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

▪ TinyCNN 학습 결과 (CIFAR-10)

- 학습 진행 내용
  - 5 에폭(epoch) 동안 학습 수행.
  - 진행률 바를 통해 배치 단위 손실값을 모니터링.
  - 각 에폭 종료 시 손실(loss)과 검증 정확도(val_acc) 출력.
- 최종 성능 요약
  - 학습 손실 : 에폭이 진행될수록 감소하는 추세.
  - 검증 정확도 : 점진적으로 향상, 단순한 모델임에도 CIFAR-10에서 ~0.4–0.5수준(약 40~50%).
  - 이는 TinyCNN의 구조적 한계로, 더 깊은 네트워크나 정규화 기법을 적용하면 향상 가능.
- 학습 곡선 해석
  - Train Loss : 꾸준히 감소 → 모델이 데이터 패턴을 학습하고 있음을 의미.
  - Val Accuracy : 증가하다가 일정 수준에서 수렴 → 단순 CNN의 표현력 한계.
  - 과적합(overfitting)은 아직 두드러지지 않음.
- 결론
  - 간단한 CNN도 이미지 분류 문제를 학습 가능.
  - 더 많은 학습(epoch)과 복잡한 구조(VGG, ResNet 등)를 적용하면 정확도 향상 가능.

```
device: cuda
[1/5] loss=1.6507  val_acc=0.523
[2/5] loss=1.3710  val_acc=0.554
[3/5] loss=1.2611  val_acc=0.601
[4/5] loss=1.1763  val_acc=0.616
[5/5] loss=1.1238  val_acc=0.639
```

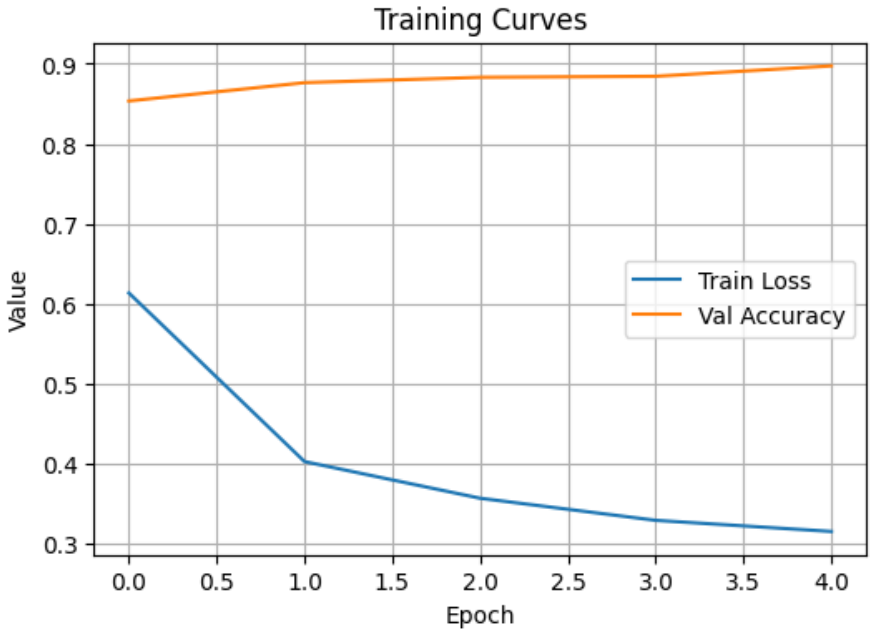


CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

- **TinyCNN 학습 결과 (Fashion-MNIST)**
  - 학습 진행 내용
    - 5 에폭(epoch) 동안 학습 수행.
    - 진행률 바를 통해 배치 단위 손실값을 모니터링.
    - 각 에폭 종료 시 손실(loss)과 검증 정확도(val_acc) 출력.
  - 최종 성능 요약
    - 학습 손실 : 에폭이 진행될수록 감소 (약 0.58 → 0.29).
    - 검증 정확도 : 빠르게 상승 후 안정화, 최종 약 89.6% 도달.
    - 단순한 CNN 구조임에도 Fashion-MNIST에서는 높은 성능을 기록.
  - 학습 곡선 해석
    - Train Loss : 꾸준히 감소 → 모델이 데이터 패턴을 안정적으로 학습하고 있음을 의미.
    - Val Accuracy : 에폭 초반부터 빠르게 증가 후 ~90% 부근에서 수렴.
    - 과적합(overfitting) 현상은 관찰되지 않음.
  - 결론
    - 간단한 CNN도 Fashion-MNIST와 같은 비교적 단순한 데이터셋에서는 높은 정확도를 달성 가능.
    - 더 많은 학습(epoch) 또는 복잡한 네트워크(VGG, ResNet 등)를 적용하면 추가적인 성능 향상 가능.

```
device: cuda
[1/5] loss=0.6130  val_acc=0.854
[2/5] loss=0.4019  val_acc=0.876
[3/5] loss=0.3560  val_acc=0.883
[4/5] loss=0.3283  val_acc=0.884
[5/5] loss=0.3144  val_acc=0.897
```



CNN 학습 기본 코드 설명

HONGIK UNIVERSITY

- **CIFAR-10 과 Fashion-MNIST 데이터셋 모두 분류 문제이며, 결과는 분류 정확도(accuracy)로 확인 가능.**
- **CIFAR-10은 복잡해 기본 모델로는 성능이 낮게 나오고, Fashion-MNIST는 단순해 높은 정확도를 보임.**
  - 이미지 크기와 채널 수
    - Fashion-MNIST: 28×28, 흑백(채널 1개) → 데이터가 단순.
    - CIFAR-10: 32×32, 컬러(RGB, 채널 3개) → 데이터가 더 복잡.
    - CNN은 채널이 많을수록 더 많은 파라미터가 필요하고, 학습 난이도도 올라감.
    - 따라서 같은 모델 구조라면 Fashion-MNIST에서 더 쉽게 높은 정확도를 달성.
  - 클래스 특성
    - Fashion-MNIST 클래스(티셔츠, 바지, 신발 등)는 형태 차이가 비교적 뚜렷함.
    - CIFAR-10 클래스(개, 고양이, 사슴 등)는 시각적으로 유사한 경우가 많아 분류가 어려움.
  - 데이터 다양성
    - CIFAR-10은 다양한 배경, 색상, 포즈가 섞여 있음 → 모델이 잡아내야 할 특징이 많음.
    - Fashion-MNIST는 배경이 단순하고 중심에 물체가 고정되어 있어 비교적 쉽고 안정적.
  - 학습 난이도 결과
    - Fashion-MNIST: 단순 구조의 CNN으로도 85% 이상 정확도 쉽게 달성.
    - CIFAR-10: 같은 모델로는 60~70% 수준, 성능을 높이려면 더 깊은 네트워크, 데이터 증강, 하이퍼파라미터 조정이 필요.
- **이 차이를 통해 학습자들은 데이터셋 특성이 모델 성능의 차이를 보일 수 있다는 것을 확인할 수 있음.**

CNN 추론 기본 코드 설명

```python
# 1) 데이터셋에 맞는 클래스 라벨 & 전처리 정의
if USE_CIFAR10:
    class_names = ['airplane','automobile','bird','cat','deer',
                   'dog','frog','horse','ship','truck']
    mean, std = (0.4914,0.4822,0.4465), (0.2023,0.1994,0.2010)
    to_img = lambda p: Image.open(p).convert("RGB")  # 3채널
    infer_tf = transforms.Compose([
        transforms.Resize((32,32)),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ])
else:
    class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat',
                   'Sandal','Shirt','Sneaker','Bag','Ankle boot']
    mean, std = (0.2861,), (0.3530,)
    to_img = lambda p: Image.open(p).convert("L")    # 1채널(그레이스케일)
    infer_tf = transforms.Compose([
        transforms.Resize((28,28)),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ])
```

```python
# 2) 이미지 경로 지정
img_path = "dog.jpg"   # 업로드한 파일명으로 바꾸세요

# 3) 전처리 & 추론
img_raw = to_img(img_path)
x = infer_tf(img_raw).unsqueeze(0).to(device)

model.eval()
with torch.no_grad():
    logits = model(x)
    probs = F.softmax(logits, dim=1)[0]
    top_p, top_i = probs.topk(3)

# 4) 결과 출력 + 시각화
print("[Top-1]", class_names[top_i[0].item()], f"({top_p[0].item():.2%})")
print("[Top-3]", [(class_names[i.item()], f"{p.item():.2%}") for p,i in zip(top_p, top_i)])

plt.imshow(img_raw if USE_CIFAR10 else img_raw, cmap=None if USE_CIFAR10 else "gray")
plt.title(f"Pred: {class_names[top_i[0].item()]}  ({top_p[0].item():.1%})")
plt.axis('off')
plt.show()
```

CNN 추론 기본 코드 설명

HONGIK UNIVERSITY

- **데이터셋에 맞는 클래스 이름과 전처리 정의**
  - 모델이 예측할 10개 클래스 이름을 정의
    - ‘airplane’, ‘automobile’, ‘bird’, ‘cat’, ‘deer’
      ‘dog’, ‘frog’, ‘horse’, ‘ship’, ‘truck’
  - 학습한 영상 형태와 같은 형태로 입력 영상을 변환
    - 컬러 or 그레이 스케일 영상.
    - 영상크기 Resize : (32 × 32) or (28 × 28)
    - 정규화.

```
# 1) 데이터셋에 맞는 클래스 라벨 & 전처리 정의
if USE_CIFAR10:
    class_names = ['airplane','automobile','bird','cat','deer',
                   'dog','frog','horse','ship','truck']
    mean, std = (0.4914,0.4822,0.4465), (0.2023,0.1994,0.2010)
    to_img = lambda p: Image.open(p).convert("RGB")  # 3채널
    infer_tf = transforms.Compose([
        transforms.Resize((32,32)),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ])
else:
    class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat',
                   'Sandal','Shirt','Sneaker','Bag','Ankle boot']
    mean, std = (0.2861,), (0.3530,)
    to_img = lambda p: Image.open(p).convert("L")    # 1채널(그레이스케일)
    infer_tf = transforms.Compose([
        transforms.Resize((28,28)),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ])
```

CNN 추론 기본 코드 설명

HONGIK UNIVERSITY

- **전처리 및 추론 실행**
  - 이전 슬라이드에서 정의한 영상의 전처리 실행.
  - 추론 실행
    - 모델을 학습모드에서 추론모드로 변경.
    - GPU(또는 CPU)로 텐서 이동.
    - 추론에서는 그래디언트 계산이 필요 없으므로 계산 제거.
    - 모델에 입력 이미지를 로드.
    - 활성화 함수인 softmax를 통해 각 클래스별 확률 계산.
    - 확률이 가장 높은 상위 3개의 클래스 추출.

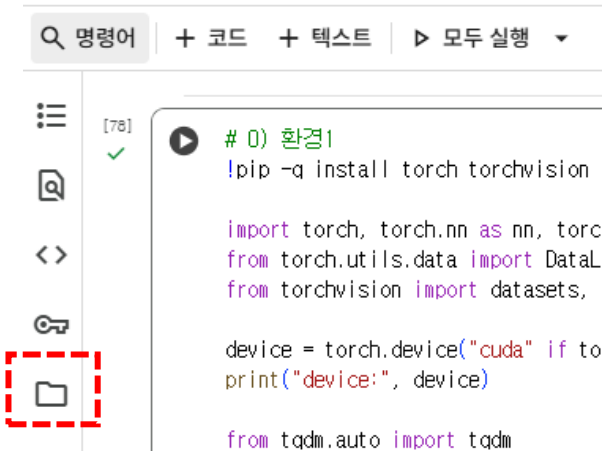
```
img_raw = to_img(img_path)
x = infer_tf(img_raw).unsqueeze(0).to(device)

model.eval()
with torch.no_grad():
    logits = model(x)
    probs = F.softmax(logits, dim=1)[0]
    top_p, top_i = probs.topk(3)
```

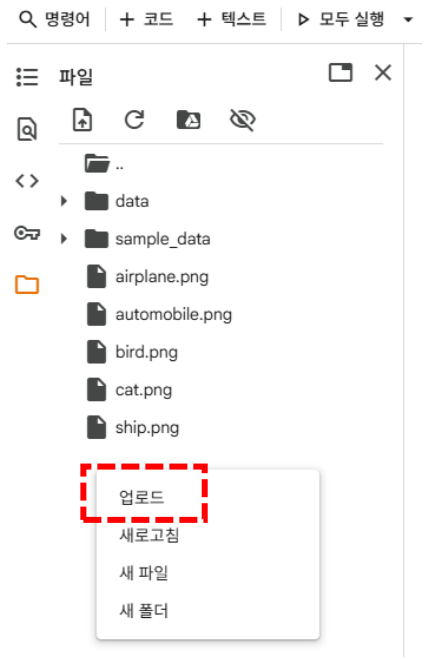
CNN 추론 기본 코드 실행 방법 및 결과

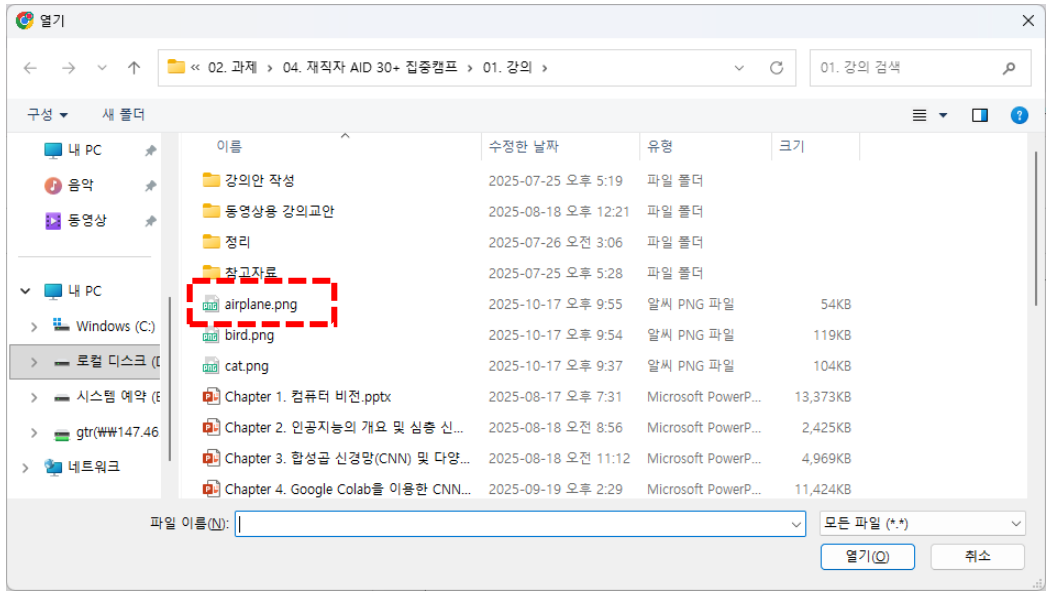
CNN 추론 기본 코드 실행 방법

HONGIK UNIVERSITY

- **분류할 이미지 업로드 및 파일 지정**
  - 인터넷 등에서 다운로드 받은 영상을 .jpg 또는 .png 등의 파일 형태로 저장하여 Colab에 업로드.







  - 소스코드에서 파일 지정

```
img_path = "dog.jpg"
```

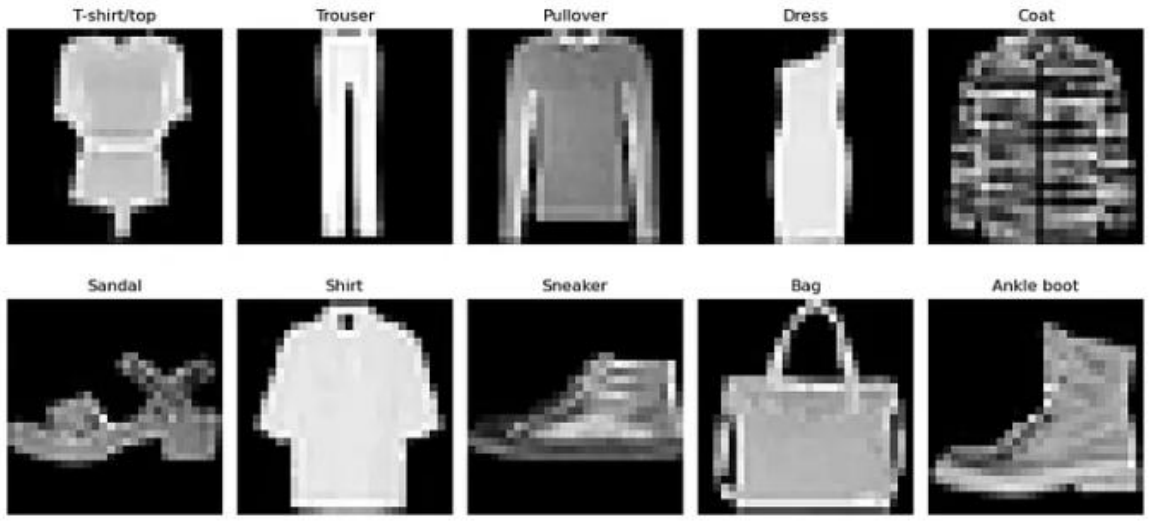
CNN 추론 기본 코드 실행 방법

HONGIK UNIVERSITY

▪ 추론에 사용할 영상 선택 방법

- CNN 모델 학습에 사용한 영상의 형식에 부합할 수 있는 영상을 선택
  - 영상의 형식 : 대상체의 위치 및 크기, 배경의 복잡도 등
  - CIFAR-10 : 대상체가 화면의 중심에 있고, 배경에 비해 대상체의 비중이 높은 영상을 선택.
  - Fashion-MNIST : 대상체가 화면의 중심에 있고, 대상체의 형상이 영상 내에서 모두 표현.
    또한 배경이 단색의 검정색 배경인 단순한 영상을 선택

[CIFAR-10]



[Fashion-MNIST]

CNN 추론 기본 코드 실행 결과

HONGIK UNIVERSITY

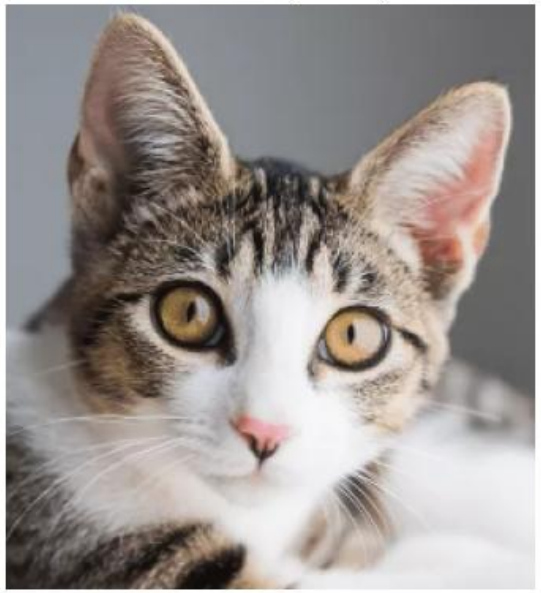
- **학습한 CNN 모델에 새로운 입력데이터를 넣은 추론 결과**
  - 학습데이터에 없는 새로운 입력데이터를 넣어서 추론을 해보면 클래스 별로 분류된 결과 확인 가능.
  - 하지만 강의에 사용한 모델은 간단한 형태로 구현된 모델이기 때문에 학습에 사용한 데이터와 형상이나 영상의 구조(배경과 대상체의 배치 등)에서 차이가 많이 발생하는 데이터는 분류 정확도가 저하될 수 있음.

```
[Top-1] airplane (83.16%)
[Top-3] [('airplane', '83.16%'), ('ship', '5.60%'), ('automobile', '5.55%')]
```

Pred: airplane (83.2%)

```
[Top-1] cat (23.60%)
[Top-3] [('cat', '23.60%'), ('bird', '19.19%'), ('dog', '17.69%')]
```

Pred: cat (23.6%)



CNN 기본 코드 개선 (과제)

CNN 기본 코드 개선 과제 예시 (1)

HONGIK UNIVERSITY

- **Epoch 수 조정**
  - Epoch은 전체 데이터셋을 몇 번 반복해서 학습할지를 의미하며, Epoch을 늘리면 모델이 데이터 패턴을 더 많이 학습 가능.
    - 기본 코드에서는 5 Epoch 학습 → CIFAR10 약 60~70% 정확도.
    - Epoch을 20으로 늘리면 CIFAR10의 경우 75~80%까지 성능 향상 가능.
    - 단, Epoch을 과도하게 늘리면 훈련 데이터에만 과적합(overfitting)될 수 있으므로 주의 필요.

```
EPOCHS = 20   # 기본 5에서 20으로 증가
```

CNN 기본 코드 개선 과제 예시 (2)

HONGIK UNIVERSITY

- **Optimizer 변경**
  - Optimizer는 손실 함수를 최소화하도록 가중치를 업데이트하는 알고리즘이며, 종류에 따라 학습 속도와 최종 성능 개선 가능.
    - Adam
      - → 적응적 학습률, 빠르고 안정적인 수렴.
      - → 소규모 네트워크에서 기본 선택지.
      - → CIFAR-10 약 75~78% 정확도(20 epoch 기준).
    - SGD + Momentum
      - → 초기 학습은 느리지만, 충분한 Epoch을 주면 높은 최종 정확도 가능.
      - → 일반화에서 성능이 Adam보다 나은 경우가 있음.
      - → CIFAR-10 약 80% 이상 가능.
    - RMSProp
      - → 학습률을 가중치마다 다르게 조정, RNN이나 시계열 데이터에서 강점.
      - → CNN에서도 안정적인 성능 제공.
    - AdamW
      - → Adam에 가중치 감쇠(weight decay) 개념을 명확히 적용.
      - → 과적합 억제 효과, 최신 딥러닝 모델에서 기본 선택지로 많이 사용됨.
      - → CIFAR-10에서 Adam 대비 약간 더 높은 일반화 성능 기대 가능.

```python
# Adam Optimizer (기본)
opt = torch.optim.Adam(model.parameters(), lr=0.001)

# SGD Optimizer (Momentum 포함)
opt = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

# RMSProp Optimizer
opt = torch.optim.RMSprop(model.parameters(), lr=0.001)

# AdamW Optimizer (정규화 포함)
opt = torch.optim.AdamW(model.parameters(), lr=0.001)
```

CNN 기본 코드 개선 과제 예시 (3)

HONGIK UNIVERSITY

▪ 손실 함수 변경

- 손실 함수는 예측값과 정답값 차이를 수치로 표현하는 기준이며, 문제 성격에 따라 선택 필요.
  - CrossEntropyLoss
    → 다중 클래스 분류에서 표준적으로 사용.
    → CIFAR-10, Fashion-MNIST 모두 기본 선택 가능.
  - CrossEntropyLoss + Label Smoothing
    → 정답 확률을 100%로 강제하지 않고 일부 분산.
    → 모델이 과도하게 확신하지 않게 만들어 일반화 성능 향상.
    → CIFAR-10에서 오분류 클래스 줄이는 효과 기대.
  - Focal Loss
    → 어려운 샘플에 더 집중하게 만드는 손실 함수.
    → 클래스 불균형 문제에서 유리, 예를 들어 고양이와 개 데이터가 한쪽에 치우친 경우.
    → Fashion-MNIST처럼 클래스가 균등한 데이터에는 효과 제한적.
  - MSE Loss
    → 평균 제곱 오차, 회귀 문제에 적합.
    → 분류 문제에서는 잘 쓰이지 않음 (출력 분포가 잘 맞지 않음).

```python
# 기본 CrossEntropyLoss
criterion = nn.CrossEntropyLoss()

# 라벨 스무딩 적용
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)

# Focal Loss (클래스 불균형 대응, 직접 구현 필요)
# 예시: from torchvision.ops import sigmoid_focal_loss

# MSE Loss (회귀 문제에 주로 사용, 분류에는 부적합)
criterion = nn.MSELoss()
```

CNN 기본 코드 개선 과제 예시 (4)

HONGIK UNIVERSITY

- **데이터 전처리 및 증강 추가**
  - 데이터 증강은 원본 데이터를 변형해 다양한 학습 샘플을 만들어 모델의 일반화 문제 개선 가능.
    - RandomCrop, HorizontalFlip : 이미지 다양성 확보 → 과적합 방지.
    - ColorJitter : 색상·밝기 변화 추가 → 조명 변화에 강건.
    - RandomRotation, RandomAffine : 회전·축소·확대·기울기 적용 → 다양한 시점에 대응.
    - RandomGrayscale : 색상 정보 의존 줄이고 형태 학습 강화.
    - RandomErasing : 이미지 일부 영역 제거 → 가림(occlusion) 상황에서도 성능 유지.
    - CIFAR-10 기준, 단순 Crop/Flip만 적용했을 때 5~7% 정확도 향상. 추가 기법까지 적용하면 최대 10% 가까운 성능 개선 가능.

```
train_tf = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.RandomRotation(15),
    transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)),
    transforms.RandomGrayscale(p=0.1),
    transforms.RandomErasing(p=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean, std),
])
```

CNN 기본 코드 개선 과제 예시 (5)

HONGIK UNIVERSITY

▪ 모델 구조 변경

- 모델 구조 변경은 합성곱 층 수, 채널 수, 규제 기법 등을 추가하여 더 강력한 표현력을 얻는 방법.
  - 합성곱 층 추가 : 더 복잡한 특징 학습 가능.
  - Batch Normalization : 층 출력 정규화 → 학습 안정화, 수렴 속도 향상.
  - Dropout : 뉴런 일부 제거 → 특정 패턴에 과도하게 의존하지 않도록 방지.
  - Residual Block(ResNet 스타일) : 깊은 네트워크 학습 시 기울기 소실 문제 완화.
  - 채널 수 증가 : 더 많은 특징 맵 학습 가능, 복잡한 패턴 분리 효과.
  - CIFAR-10 기준, 단순 합성곱 층 추가 시 약 10% 이상 성능 향상 가능. BatchNorm, Dropout을 함께 적용하면 안정성과 일반화 성능까지 강화.

```
self.conv = nn.Sequential(
    nn.Conv2d(in_ch, 32, 3, padding=1), nn.BatchNorm2d(32), nn.ReLU(),
    nn.Conv2d(32, 64, 3, padding=1), nn.BatchNorm2d(64), nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Conv2d(64, 128, 3, padding=1), nn.ReLU(), nn.Dropout(0.3),
    nn.MaxPool2d(2),
)
```

CNN 기본 코드 개선 과제 예시 (6)

HONGIK UNIVERSITY

▪ 학습 기법 확장

- 학습 과정 자체를 개선하는 방법으로, 학습률 스케줄러나 조기 종료 등을 적용 가능.
  - Learning Rate Scheduler : 일정 Epoch마다 학습률 감소 → 더 정밀한 수렴 가능.
  - Early Stopping : 검증 성능이 더 이상 좋아지지 않으면 학습 중단 → 과적합 방지.
  - Gradient Clipping : 기울기 폭발 방지 → 안정적인 학습 유지.
  - CIFAR-10 기준으로 스케줄러 적용 시 성능이 2~3% 더 향상되는 경우가 많음.

```
scheduler = torch.optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
```

CNN 기본 코드 개선 과제 예시 (6)

HONGIK UNIVERSITY

※ 스케줄러 추가 방법

- 옵티마이저 다음에 스케줄러를 정의하고 각 에폭마다 scheduler.step()을 호출

```python
from torch.optim.lr_scheduler import StepLR

model = TinyCNN(in_ch=in_ch, ncls=num_classes).to(device)
opt = torch.optim.Adam(model.parameters(), lr=LR)

# ★ 스케줄러 정의(옵티마이저 바로 아래)
scheduler = StepLR(opt, step_size=2, gamma=0.5)   # 매 2에폭마다 lr을 1/2로

criterion = nn.CrossEntropyLoss()
```

```python
for ep in range(1, EPOCHS+1):
    model.train()
    running_loss = 0.0
    pbar = tqdm(train_ld, desc=f"Epoch {ep}/{EPOCHS}", leave=False)
    for x,y in pbar:
        x,y = x.to(device), y.to(device)
        logits = model(x)
        loss = criterion(logits, y)
        opt.zero_grad(); loss.backward(); opt.step()
        running_loss += loss.item()
        pbar.set_postfix(loss=f"{loss.item():.4f}")

    avg_loss = running_loss / len(train_ld)
    acc = eval_acc()
    train_losses.append(avg_loss);  val_accs.append(acc)

    # ★ 에폭 종료 후 스케줄러 스텝
    scheduler.step()
    curr_lr = scheduler.get_last_lr()[0]
```

CNN 기본 코드 개선 과제의 예상 결과

HONGIK UNIVERSITY

- **Epoch 증가** : 반복 학습 기회 확대, 성능 향상 → 과적합 주의.
- **Optimizer 변경** : Adam은 빠르고 안정적, SGD는 더 높은 최종 성능, AdamW는 최신 모델에서 선호.
- **손실 함수 변경** : CrossEntropy 기본, Label Smoothing은 일반화 강화, Focal Loss는 불균형 대응.
- **데이터 증강** : 회전·기하학적 변형·Erasing 등으로 과적합 완화, 정확도 상승.
- **모델 구조 개선** : BatchNorm, Dropout, Residual Block으로 안정성 및 성능 향상.
- **학습 기법 확장** : 스케줄러, 조기 종료, 기울기 안정화로 효율적인 학습 지원.

From JPs