Computer Vision

Lecture 02: 파이썬 설치 및 파이토치 실습

Jaesung Lee

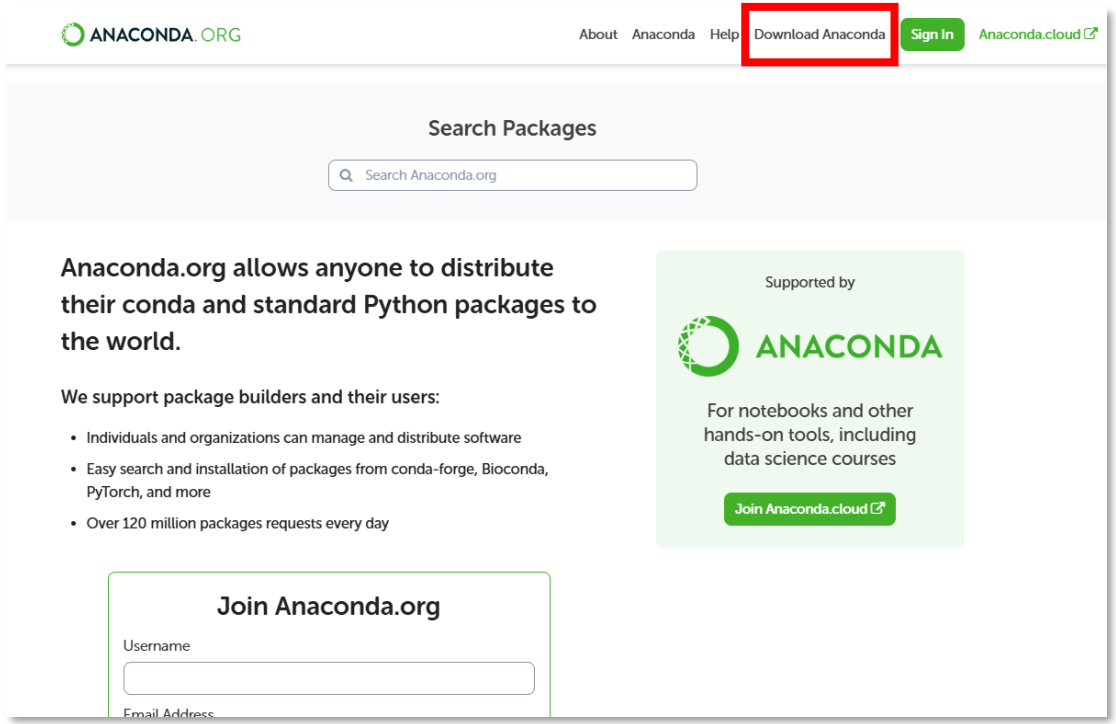
curseor@cau.ac.kr

2025. 03. 12.

Contents

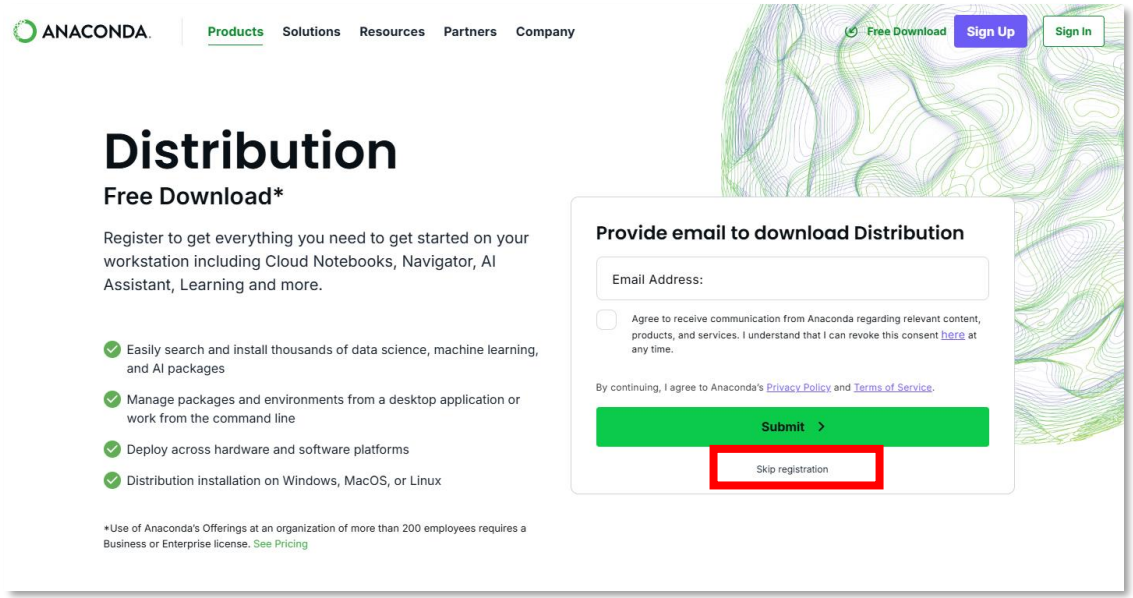
1. 아나콘다 설치

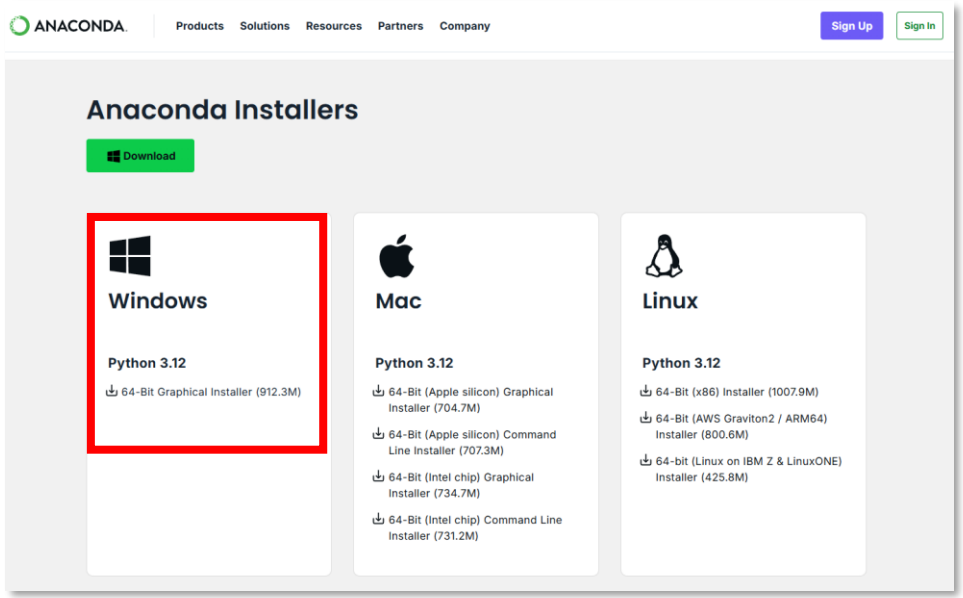
- 아나콘다(Anaconda)를 설치하면 머신러닝, 딥러닝 등의 개발 환경을 쉽게 구축할 수 있으며, 파이썬(Python), 주피터 노트북(Jupyter Notebook) 등 주요 개발 도구도 함께 자동으로 설치됨
- 아나콘다 공식 홈페이지(https://anaconda.org/)에 접속하여 다운로드 페이지에 접속함



1. 아나콘다 설치

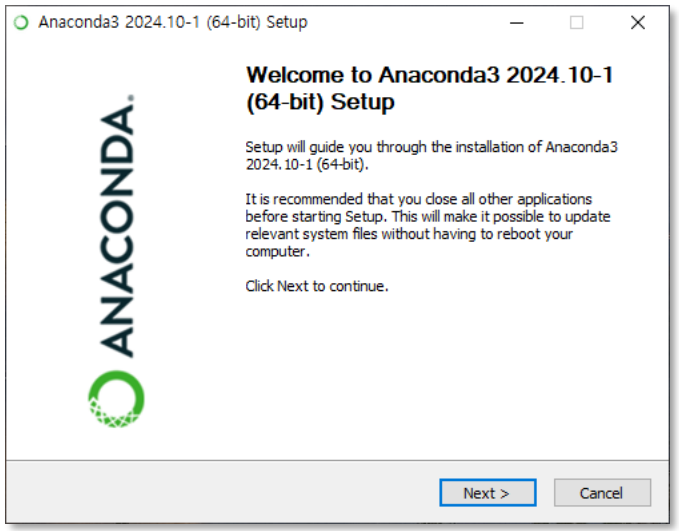
- 다운로드 페이지에서 하단 [Skip registration]을 선택함
- 설치 화면에서 운영체제에 맞는 아나콘다를 설치함

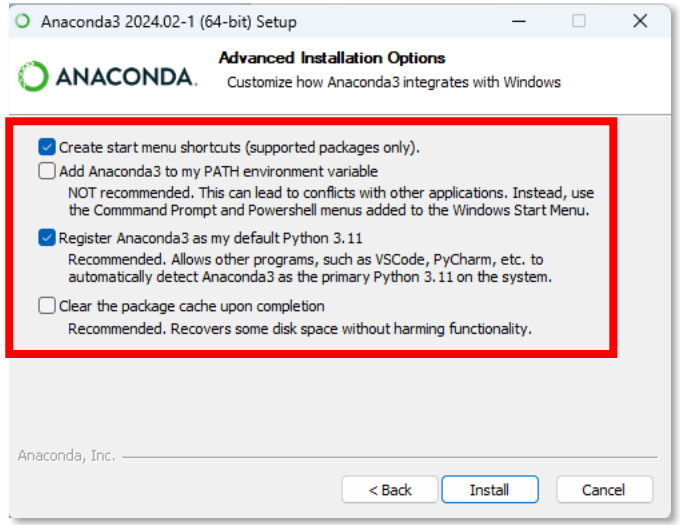


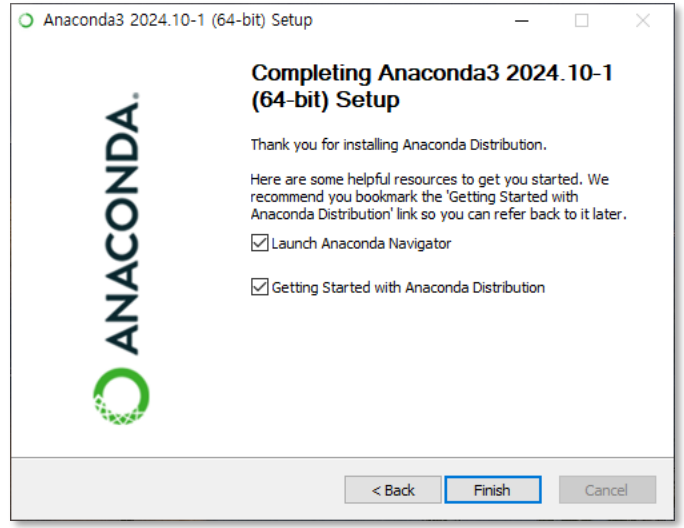


1. 아나콘다 설치

- 다운로드한 아나콘다 설치 파일로 설치를 진행함
- 이때, 추가 설치 옵션에서 아래의 그림처럼 선택하여 설치함
- 설치가 완료되면 Finish 버튼을 눌러 설치를 마무리함

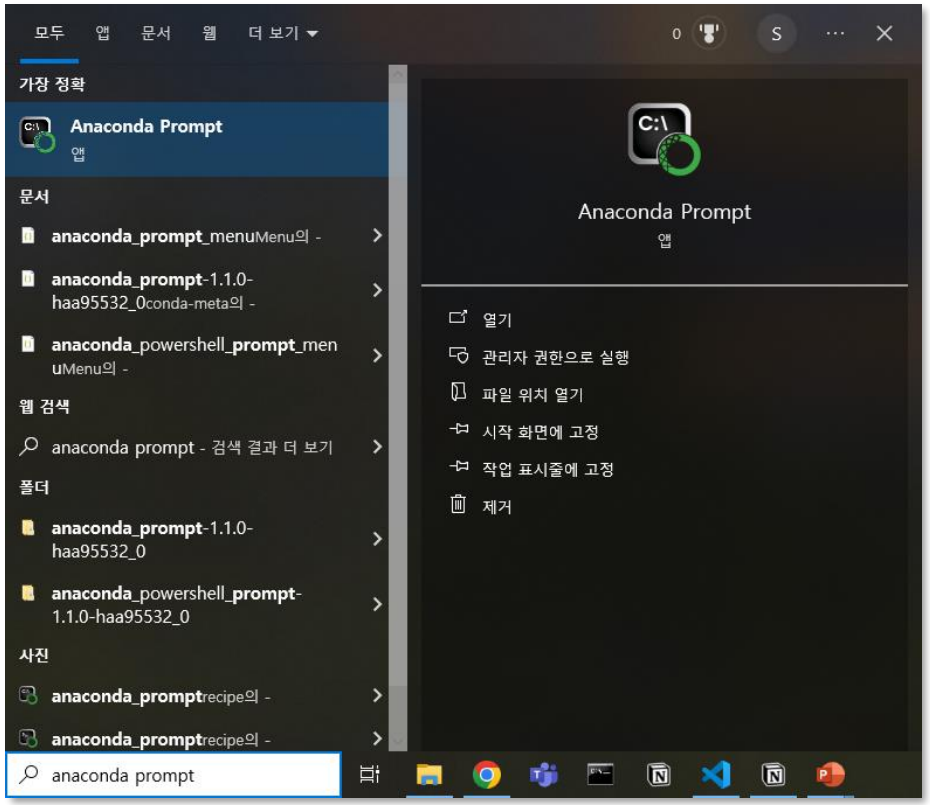


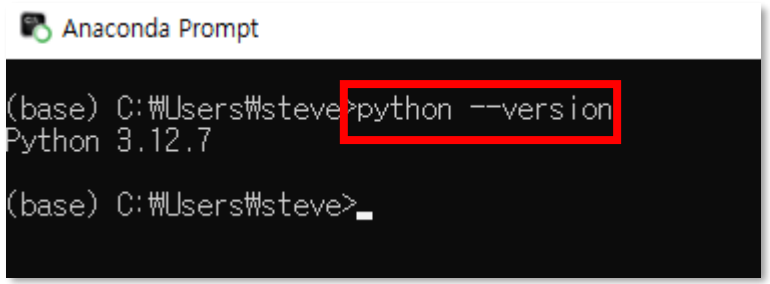




1. 아나콘다 설치

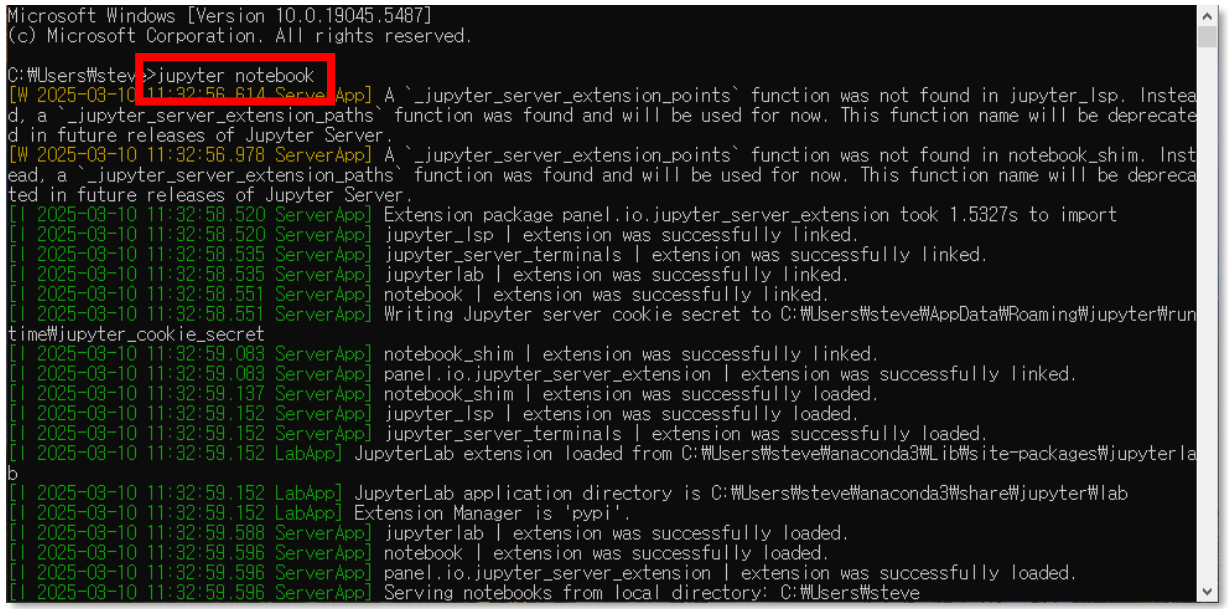
- 아나콘다 설치가 완료되면 패키지를 통해 파이썬이 함께 설치됨
- 아나콘다 프롬프트에서 ‘python --version’ 명령어를 통해 현재 설치된 파이썬 버전을 확인할 수 있음





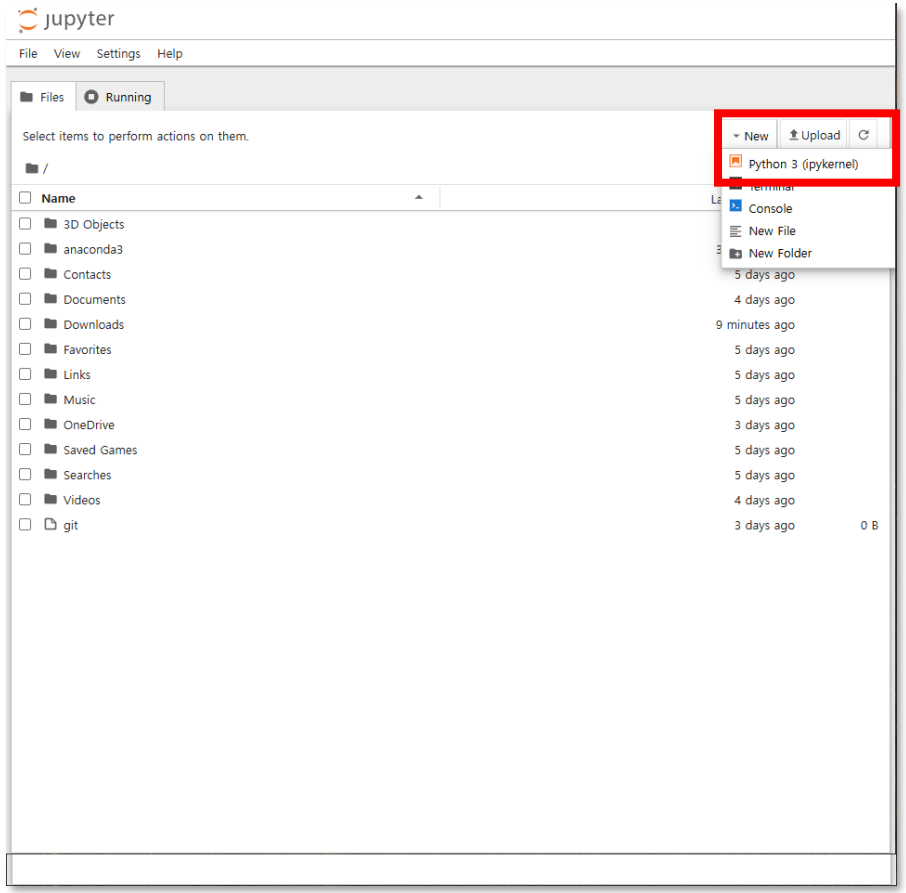
1. 아나콘다 설치

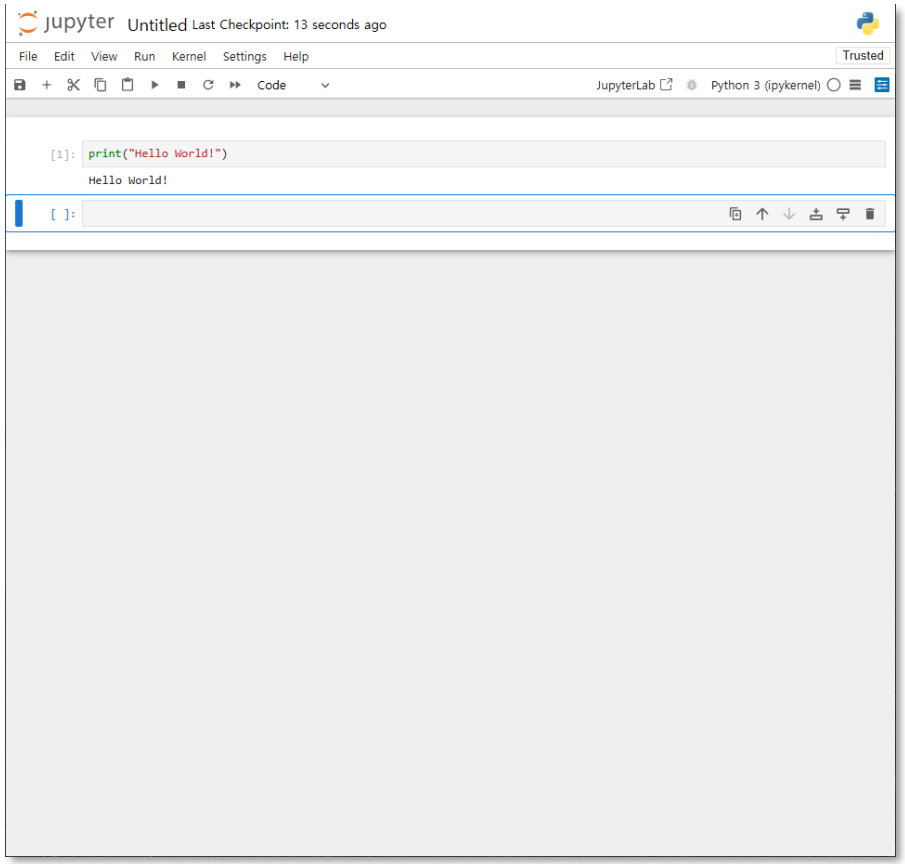
- 아나콘다를 설치하면 주피터 노트북도 함께 설치됨
- 따라서 별도의 추가 설치 없이 아래 그림과 같이 터미널에서 ‘jupyter notebook’ 명령어를 통해 주피터 노트북을 실행할 수 있음



1. 아나콘다 설치

- 주피터 노트북이 실행되면 화면 오른쪽 위에 보이는 [New] – [Python3 (ipykernel)]을 선택하여 새 노트북을 생성함
- 생성된 파이썬 노트북에서 코드를 셀 별로 작성하고 실행할 수 있음





2. PyTorch 실습

- 아래의 github repository 주소로 접속하여, 가상 환경 설정 및 파이토치 실습을 진행함

https://github.com/tkdgur658/CAU_25_Spring_ComputerVision/tree/main

2. PyTorch 실습

- 활성화한 가상환경에서 PyTorch를 비롯해 기본적으로 실험에 사용할 모듈을 불러옴

```
# Pytorch basic modules
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data

# Data loading and augmentation
import torchvision.transforms as transforms
import torchvision.datasets as datasets

# Visualization and math operations
from tqdm.notebook import trange, tqdm
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np

import copy
import random
import time
```

- 재현성을 보장하기 위해서는 고정된 시드를 설정하는 것이 중요함. 난수를 제어하여 실행마다 일관된 결과를 유지하도록 하며, 실험의 비교 및 검증을 정확하게 수행할 수 있도록 도움

```
SEED = 1234

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

2. PyTorch 실습

데이터셋 준비

- MNIST 데이터셋을 불러오기 위한 경로 설정 및 데이터셋 객체를 생성함

```
ROOT = 'data'

train_data = datasets.MNIST(root=ROOT,
                            train=True,
                            download=True)
```

- 학습을 단순화하고 가속하며 지역 최소점을 피하기 위해서는 학습 데이터 정규화가 필요함. 이에 사용되는 평균과 표준 편차를 계산함

```
mean = train_data.data.float().mean() / 255
std = train_data.data.float().std() / 255
```

```
print(f'Calculated mean: {mean}')
print(f'Calculated std: {std}')
```

```
Calculated mean: 0.13066047430038452
Calculated std: 0.30810779333114624
```

2. PyTorch 실습

데이터셋 준비

- 모델 성능 향상을 위해 학습 데이터의 다양성을 높이는 것이 중요함. PyTorch의 transform 모듈을 활용한 데이터 증강을 통해 회전, 뒤집기, 크기 조정 등의 변환을 적용하면 모델의 일반화 성능과 강건성이 향상됨

```
train_transforms = transforms.Compose([
                            transforms.RandomRotation(5),
                            transforms.RandomCrop(28, padding=2),
                            transforms.ToTensor(),
                            transforms.Normalize(mean=[mean], std=[std])
                                    ])

test_transforms = transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize(mean=[mean], std=[std])
                                    ])
```

- 정의된 변환들을 적용하여 학습 및 테스트 데이터셋을 불러옴

```
train_data = datasets.MNIST(root=ROOT,
                            train=True,
                            download=True,
                            transform=train_transforms)

test_data = datasets.MNIST(root=ROOT,
                           train=False,
                           download=True,
                           transform=test_transforms)
```

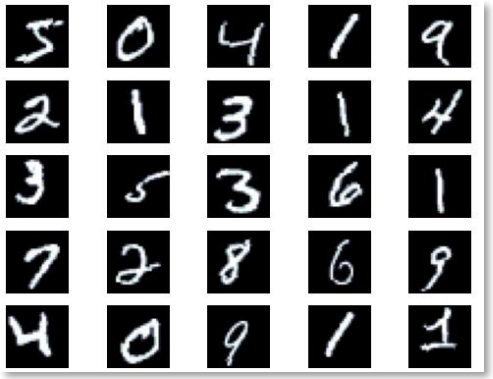
2. PyTorch 실습

데이터셋 준비

- 실제 작업 시작 전에 데이터셋을 시각화함. 데이터의 구조와 특성을 이해하는 데 도움이 되며, 유용한 정보를 얻을 수 있게 함. MNIST 데이터셋은 손글씨 숫자로 구성된 대표적인 기계학습 데이터셋임

```python
def plot_images(images):

    n_images = len(images)

    rows = int(np.sqrt(n_images))
    cols = int(np.sqrt(n_images))

    fig = plt.figure()
    for i in range(rows*cols):
        ax = fig.add_subplot(rows, cols, i+1)
        ax.imshow(images[i].view(28, 28).cpu().numpy(), cmap='bone')
        ax.axis('off')
```

```python
N_IMAGES = 25

images = [image for image, label in [train_data[i] for i in range(N_IMAGES)]]

plot_images(images)
```



2. PyTorch 실습

데이터셋 준비

- 검증 데이터셋은 학습 중 모델의 일반화 성능을 검증하고, 하이퍼파라미터 튜닝을 위한 기준을 제공함. 검증 데이터셋은 반드시 학습 데이터셋에서 분할하며, 본 예제에서는 학습 데이터의 10%를 검증용으로 할당함

```
VALID_RATIO = 0.1              # 10%
TRAIN_RATIO = 1 - VALID_RATIO  # 100% - VALID_RATIO

n_train_examples = int(len(train_data) * TRAIN_RATIO)
n_valid_examples = len(train_data) - n_train_examples
```

```
train_data, valid_data = data.random_split(train_data,
                                           [n_train_examples, n_valid_examples])
```

```
print(f'Number of training examples: {len(train_data)}')
print(f'Number of validation examples: {len(valid_data)}')
print(f'Number of testing examples: {len(test_data)}')
```

```
Number of training examples: 54000
Number of validation examples: 6000
Number of testing examples: 10000
```

- 검증 데이터는 성능 평가를 위한 것이므로 테스트 데이터와 마찬가지로 데이터 증강을 적용하지 않음

```
valid_data = copy.deepcopy(valid_data)
valid_data.dataset.transform = test_transforms
```

2. PyTorch 실습

학습 단계

- 학습 시작 전, 배치 크기와 같은 주요 하이퍼파라미터를 정의하며, 각 데이터셋에 대한 데이터 로더를 설정함. 데이터 로더는 미니 배치를 생성하고 데이터를 순차적으로 제공하는 역할을 함

```
BATCH_SIZE = 64

train_iterator = data.DataLoader(train_data,
                                 shuffle=True,
                                 batch_size=BATCH_SIZE)

valid_iterator = data.DataLoader(valid_data,
                                 batch_size=BATCH_SIZE)

test_iterator = data.DataLoader(test_data,
                                batch_size=BATCH_SIZE)
```

2. PyTorch 실습

학습 단계

- 입력층, 출력층 그리고 512개의 뉴런을 가진 은닉층 하나로 구성된 간단한 MLP 모델을 정의함

```python
class MLP(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.output_dim = output_dim

        hidden_layer_dimension = 512

        self.input_layer = nn.Linear(input_dim, hidden_layer_dimension)
        self.hidden_layer = nn.Linear(hidden_layer_dimension, hidden_layer_dimension)
        self.output_layer = nn.Linear(hidden_layer_dimension, output_dim)

    def forward(self, x):

        # x = [batch size (B), height (H), width (W)]

        batch_size = x.shape[0]

        x = x.view(batch_size, -1)

        # x = [B, H * W]

        out_1 = F.tanh(self.input_layer(x))

        # out_1 = [B, hidden_layer_dimension (512)]

        out_2 = F.tanh(self.hidden_layer(out_1))

        # out_2 = [B, hidden_layer_dimension (512)]

        prediction = self.output_layer(out_2)

        # prediction = [B, output dim]

        return prediction
```

2. PyTorch 실습

학습 단계

- 선언한 모델을 아래와 같이 초기화 함

```
INPUT_DIM = 28 * 28
OUTPUT_DIM = 10

model = MLP(INPUT_DIM, OUTPUT_DIM)
```

- 각 반복에서 경사 하강법을 활용해 모델의 파라미터를 최적화하기 위해 Adam 옵티마이저를 사용함

```
optimizer = optim.Adam(model.parameters())
```

- 모델의 예측값과 실제값 간의 차이를 측정하기 위해 손실함수를 정의하며, 본 모델에서는 분류 작업에 적합한 Cross Entropy Loss를 사용함

```
loss_function = nn.CrossEntropyLoss()
```

2. PyTorch 실습

학습 단계

- 모델이 학습될 디바이스를 지정함

```
device = torch.device('cpu')
```

- ‘to()’ 메서드는 모델이나 데이터를 지정된 디바이스로 전송하는 역할을 함

```
model = model.to(device)
loss_function = loss_function.to(device)
```

- 학습 데이터 배치에서 올바르게 예측된 샘플 수의 평균을 계산하여 정확도를 측정함

```
def calculate_accuracy(y_pred, y):
    top_pred = y_pred.argmax(1, keepdim=True)
    correct = top_pred.eq(y.view_as(top_pred)).sum()
    acc = correct.float() / y.shape[0]
    return acc
```

2. PyTorch 실습

학습 단계

- 이제 학습 함수를 정의함. 학습 시에는 train() 모드를 사용하며, 앞서 정의한 옵티마이저, 손실함수를 사용하여 모델의 파라미터를 업데이트 함

```python
def train(model, iterator, optimizer, criterion, device):

    epoch_loss = 0
    epoch_acc = 0

    model.train()

    for (x, y) in tqdm(iterator, desc="Training", leave=False):

        x = x.to(device)
        y = y.to(device)

        optimizer.zero_grad()

        y_pred = model(x)

        loss = criterion(y_pred, y)

        acc = calculate_accuracy(y_pred, y)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

2. PyTorch 실습

학습 단계

- 검증 함수를 정의함. 검증 시에는 eval() 모드를 사용하며, torch.no_grad()를 활용해 그래디언트 계산을 비활성화하고, 모델의 예측값과 실제 값의 차이를 기반으로 손실과 정확도를 평가함

```python
def evaluate(model, iterator, criterion, device):

    epoch_loss = 0
    epoch_acc = 0

    model.eval()

    with torch.no_grad():

        for (x, y) in tqdm(iterator, desc="Evaluating", leave=False):

            x = x.to(device)
            y = y.to(device)

            y_pred = model(x)

            loss = criterion(y_pred, y)

            acc = calculate_accuracy(y_pred, y)

            epoch_loss += loss.item()
            epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

2. PyTorch 실습

학습 단계

- 모델은 정의된 횟수만큼 에포크를 반복하며 학습됨. 각 에포크에서 학습 데이터를 사용해 모델을 훈련하고, 매 반복이 끝날 때 검증 손실을 측정하여 성능을 평가함. 검증 손실이 최저 값을 기록하면 모델 가중치를 저장함

```
EPOCHS = 10

best_valid_loss = float('inf')

for epoch in trange(EPOCHS):

    start_time = time.perf_counter()

    train_loss, train_acc = train(model, train_iterator, optimizer, loss_function, device)
    valid_loss, valid_acc = evaluate(model, valid_iterator, loss_function, device)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'mlp-model-mnist.pt')

    end_time = time.perf_counter()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

- 학습이 완료되면 최저 검증 손실을 기록한 모델을 불러와 테스트 데이터셋에서 성능을 평가함

```
model.load_state_dict(torch.load('mlp-model-mnist.pt', weights_only=True))

test_loss, test_acc = evaluate(model, test_iterator, loss_function, device)
```

2. PyTorch 실습

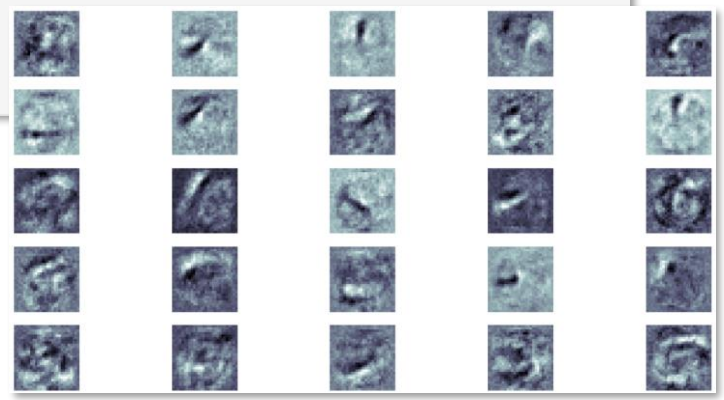
분석 단계

- 모델이 학습한 가중치를 시각화하면, 학습 과정에서 모델이 어떤 패턴을 학습했는지 이해하는 데 도움이 될 수 있음

```
def plot_weights(weights, n_weights):

    rows = int(np.sqrt(n_weights))
    cols = int(np.sqrt(n_weights))

    fig = plt.figure(figsize=(20, 10))
    for i in range(rows*cols):
        ax = fig.add_subplot(rows, cols, i+1)
        ax.imshow(weights[i].view(28, 28).cpu().numpy(), cmap='bone')
        ax.axis('off')
```

- 예컨대 MNIST 데이터셋에서 숫자 인식을 위한 모델을 학습한 경우, 초기 레이어는 주로 엣지, 곡선, 형태와 같은 기본적인 특징을 학습하게 됨

```
N_WEIGHTS = 25

weights = model.input_layer.weight.data

plot_weights(weights, N_WEIGHTS)
```



2. PyTorch 실습

분석 단계

- 모델의 학습 가능한 파라미터 수를 계산함. 이는 모델의 복잡도와 필요한 자원량을 파악하는 데 도움이 될 수 있음

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
```

```
print(f'The model has {count_parameters(model):,} trainable parameters')
```

```
The model has 669,706 trainable parameters
```

2. PyTorch 실습

분석 단계

- 모델이 어려움을 겪는 데이터를 이해하기 위해 혼동 행렬을 이용할 수 있음. 아래는 혼동 행렬을 계산하기 전 모델의 예측을 얻는 단계임

```
def get_predictions(model, iterator, device):

    model.eval()

    images = []
    labels = []
    probs = []

    with torch.no_grad():

        for (x, y) in iterator:

            x = x.to(device)

            y_pred = model(x)

            y_prob = F.softmax(y_pred, dim=-1)

            images.append(x.cpu())
            labels.append(y.cpu())
            probs.append(y_prob.cpu())

    images = torch.cat(images, dim=0)
    labels = torch.cat(labels, dim=0)
    probs = torch.cat(probs, dim=0)

    return images, labels, probs
```

```
images, labels, probs = get_predictions(model, test_iterator, device)

pred_labels = torch.argmax(probs, 1)
```

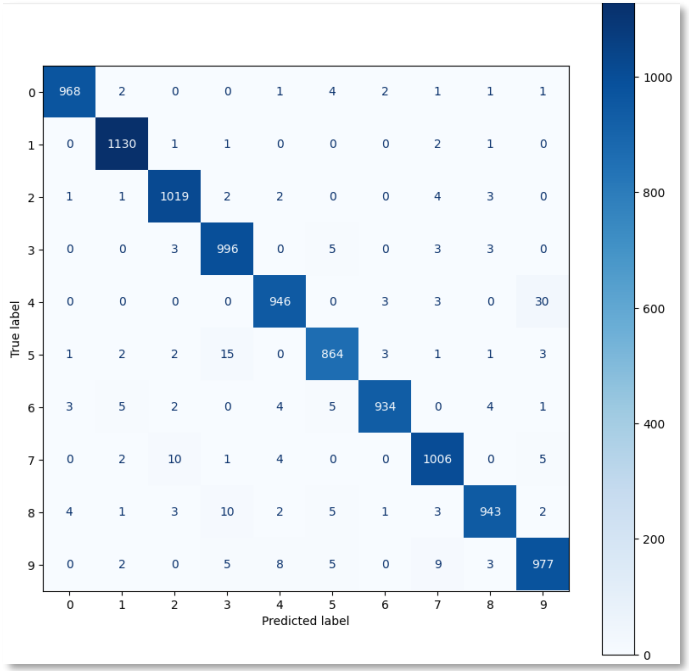
2. PyTorch 실습

분석 단계

- 혼동행렬은 예측된 레이블을 바탕으로, 실제 클래스와 비교하여 정답과 오분류된 샘플 수를 시각적으로 표현함. 대각선 요소는 올바르게 예측된 샘플을, 비대각선 요소는 잘못 분류된 샘플을 나타냄

```
def plot_confusion_matrix(labels, pred_labels):

    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(1, 1, 1)
    cm = metrics.confusion_matrix(labels, pred_labels)
    cm = metrics.ConfusionMatrixDisplay(cm, display_labels=range(10))
    cm.plot(values_format='d', cmap='Blues', ax=ax)
```

```
plot_confusion_matrix(labels, pred_labels)
```



2. PyTorch 실습

분석 단계

- 마지막으로, 모델이 올바르게 분류하지 못한 데이터를 확인할 수 있음. 아래 정의한 함수는 실제 레이블과 예측된 레이블을 비교하여 오분류된 샘플을 추출하고, 이를 시각화 함

```
corrects = torch.eq(labels, pred_labels)
```

```
incorrect_examples = []

for image, label, prob, correct in zip(images, labels, probs, corrects):
    if not correct:
        incorrect_examples.append((image, label, prob))

incorrect_examples.sort(reverse=True,
                        key=lambda x: torch.max(x[2], dim=0).values)
```

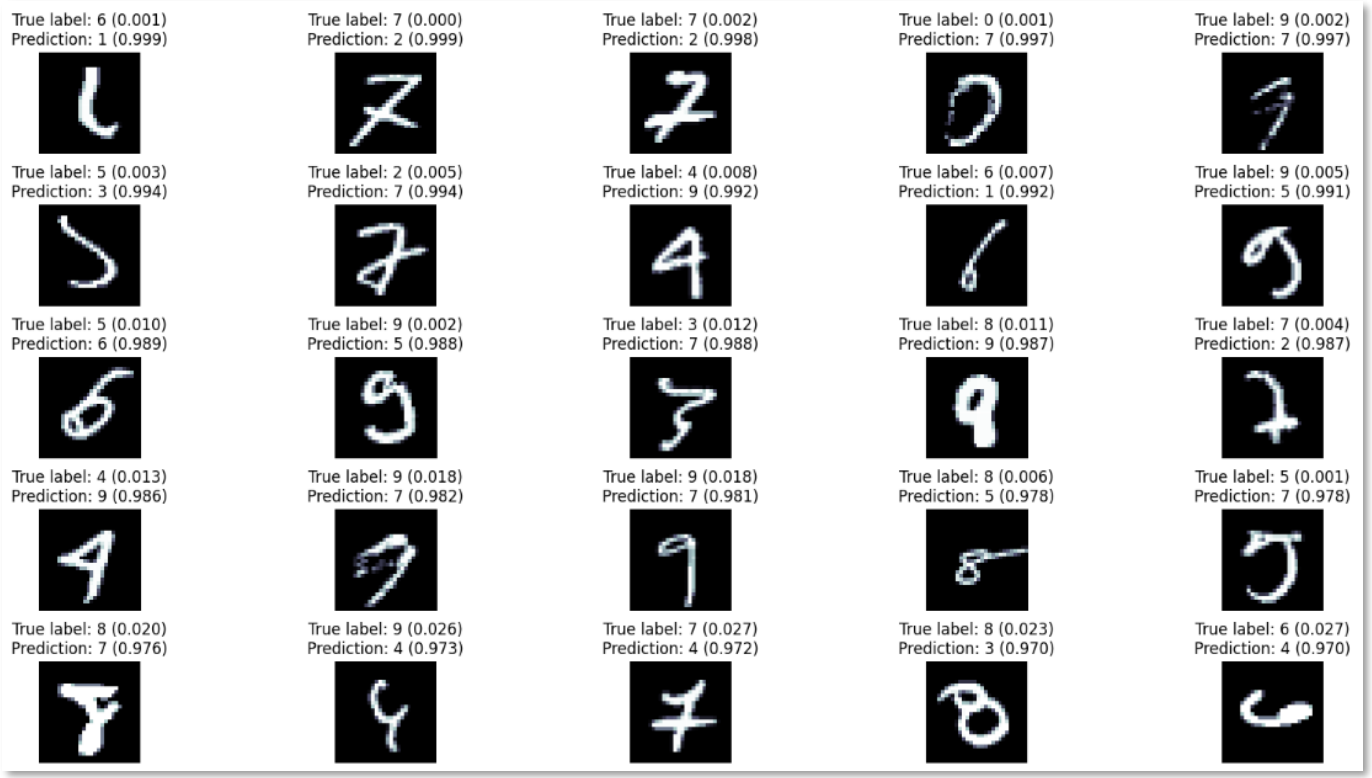
```
def plot_most_incorrect(incorrect, n_images):

    rows = int(np.sqrt(n_images))
    cols = int(np.sqrt(n_images))

    fig = plt.figure(figsize=(20, 10))
    for i in range(rows*cols):
        ax = fig.add_subplot(rows, cols, i+1)
        image, true_label, probs = incorrect[i]
        true_prob = probs[true_label]
        incorrect_prob, incorrect_label = torch.max(probs, dim=0)
        ax.imshow(image.view(28, 28).cpu().numpy(), cmap='bone')
        ax.set_title(f'True label: {true_label} ({true_prob:.3f})\n'
                     f'Prediction: {incorrect_label} ({incorrect_prob:.3f})')
        ax.axis('off')
    fig.subplots_adjust(hspace=0.5)
```

2. PyTorch 실습

분석 단계

- 앞서 정의한 함수를 실행하여, 모델이 잘못 예측한 샘플을 시각화 함. 이는 어떤 유형의 데이터에서 오류가 발생하였는지 분석하는 데 도움을 줌

```
N_IMAGES = 25

plot_most_incorrect(incorrect_examples, N_IMAGES)
```



Thank you