# 2025.08.28
# 머신러닝 세미나

High Energy Nuclear Physics
Lee Jiwon

INHA
University

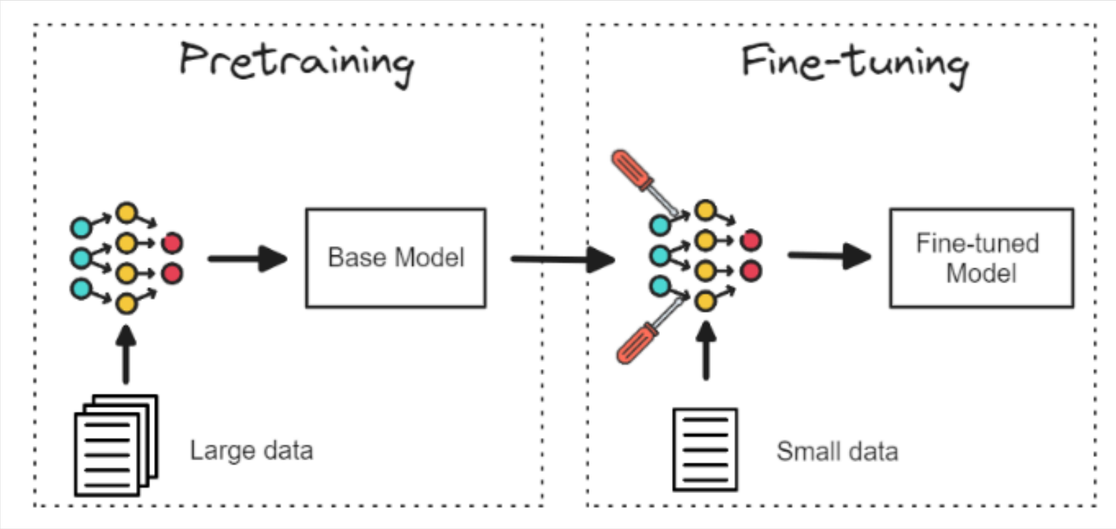
# Contents

01 fine tuning 이란?

02 Timm이란?

03 코드 설명

# Fine-tuning이란?




## 01 개념

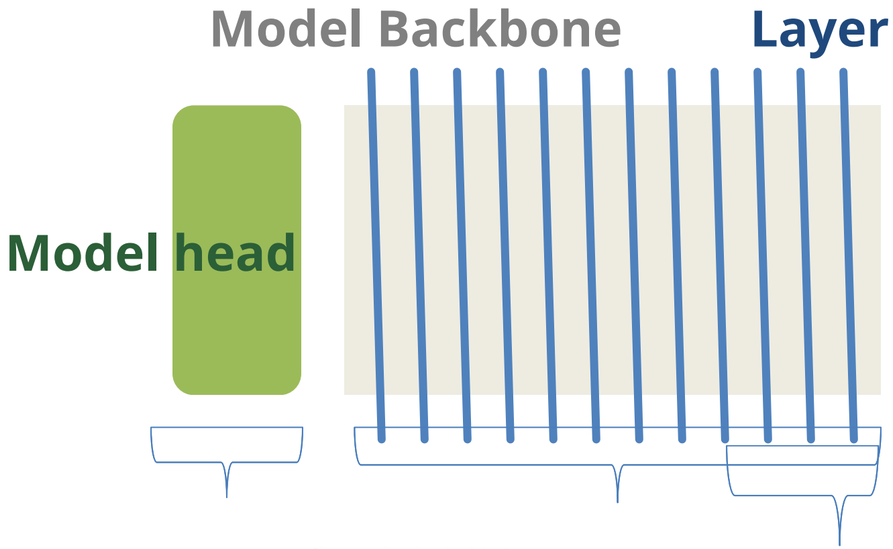
- 사전 학습된 모델에 새로운 데이터셋으로 추가 학습시키는 과정이다.
- Timm 라이브러리에 사전 학습된 모델들이 있다.
- 대부분 이미지넷에서 가져온 대용량 데이터로 학습을 시켜 둔 상태이다.

## 02 왜 필요할까?

- 학습시키려는 데이터가 적을 때
- 나의 데이터에 더 맞게 모델을 학습시키고 싶을 때
- 빠르게 학습시켜서 모델의 정확도를 높이고 싶을 때

# Fine-tuning이란?

## Fine-tuning 방법




**헤드 교체 후 학습**

- 기존의 헤드를 새로운 task에 맞게 헤드 교체 후 그 부분만 학습
- 데이터가 극도로 적거나 빠른 실험이 필요할 때 사용
- 성능이 제한적

**전체 파라미터 학습 (Full Fine-tuning)**

- 모델 backbone 전체를 학습
- 도메인 차이가 크거나 데이터셋이 충분히 있을 때
- 연산 비용 큼, 과적합 위험 있음

**부분 파라미터 학습 (Partial Fine-tuning)**

- 초기 계층은 그대로 두고 상위 계층만 학습
- 자원 절약 + 안정적 수렴

# Timm이란?

**PyTorch 기반의 이미지 모델 라이브러리**

- Vision 모델을 편리하게 불러오고 사용할 수 있다.
- ViT(Vision Transformer), Swin Transformer, ConvNeXt, EfficientNet, ResNet 등 최신/고전 모델을 로드할 수 있다.
- 모델을 로드하는 코드가 편하다.
- 커스터마이징에 유연하다.
- 모델의 backbone만 가져와 다른 task을 하기에 좋다.
- 성능을 최적화하기 위한 최신 기법들을 반영한다. Ex) DropPath, EMA, mixed precision, weight init 등

# 실제 코드 설명

## Timm에서 모델 가져오는 방법

```
!pip install --quiet timm torch torchvision pandas scikit-learn

import math
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from timm import create_model
from timm.models.vision_transformer import PatchEmbed
```

```
model = create_model('vit_base_patch16_224', pretrained=True, num_classes=num_classes)

optimizer = torch.optim.AdamW(model.parameters(), lr=3e-5)
criterion = nn.CrossEntropyLoss()
num_epochs = 5
```

Timm 라이브러리를 설치한다.
코드에 필요한 함수를 불러온다.

Create_model 함수를 이용하여 timm에서 불러오고 싶은 모델을 선택한다. 사전학습된 가중치를 사용할 것인지 아닌지와 분류하는 class의 수를 정해준다.

모델이 배우는 속도와 방식을 정한다.
옵티마이저, 한 번의 업데이트에서 가중치를 얼마나 조정할지 정해주는 학습률 lr(fine-tuning에서는 작은 학습률을 사용)과 어떤 손실 함수를 사용할 것인지, 전체 데이터셋을 몇 번 반복해서 학습할지를 정하는 epoch 수를 정해준다.

# 실제 코드 설명

## 가져온 모델 학습시키는 방법

```
# 1) 모든 파라미터 동결
for param in model.parameters():
    param.requires_grad = False

# 2) 마지막 6개 Transformer 블록과 분류기 헤드만 학습
for param in model.blocks[-6:].parameters():   # 뒤쪽 절반 블록
    param.requires_grad = True

for param in model.head.parameters():          # 분류기 헤드
    param.requires_grad = True

# 3) optimizer는 requires_grad=True인 파라미터만 포함
optimizer = torch.optim.AdamW(
    filter(lambda p: p.requires_grad, model.parameters()),
    lr=3e-5
)
```

학습하여 파라미터를 업데이트 할 블록수를 정해준다.

```
# fine-tune 완료 후
torch.save(model.state_dict(), "/content/drive/MyDrive/signal vs
background/data/xic_vit_finetuned.pth")
torch.save({
    "scaler_mean": scaler.mean_,
    "scaler_scale": scaler.scale_
}, "/content/drive/MyDrive/signal vs background/data/xic_vit_scaler.pth")
```

Fine tuning 완료 후 업데이트 된 가중치를 저장하여 나중에 다시 사용할 때 불러올 수 있도록 한다.