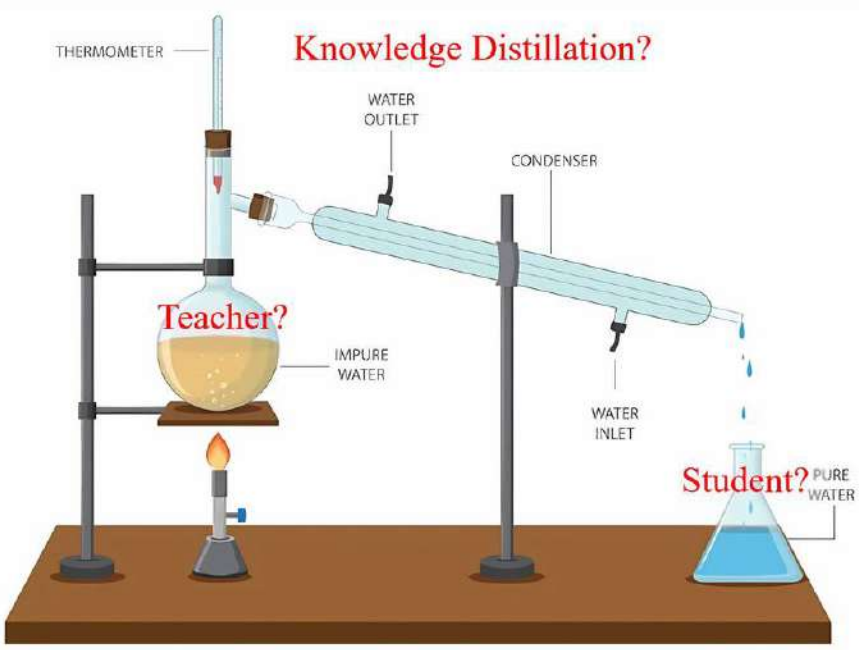
AI VIET NAM – AI COURSE 2024

Tutorial: Basic Knowledge Distillation

Yen-Linh Vu, Anh-Khoi Nguyen, Dinh-Thang Duong, Quang-Vinh Dinh

I. Giới thiệu

Khi nhắc đến **distillation** (tạm dịch: chưng cất), hẳn nhiều người sẽ nghĩ ngay đến quá trình chưng cất để tách rượu khỏi nước hay tạo ra những chất tinh khiết từ hỗn hợp phức tạp trong hóa học. Bạn có bao giờ nghĩ rằng **distillation** cũng được áp dụng trong trí tuệ nhân tạo (AI) – và ở đây, thứ được "chưng cất" chính là... tri thức? Vậy khái niệm **Knowledge Distillation** (tạm dịch: chưng cất tri thức) trong AI là gì? Nó có gì khác biệt với quá trình chưng cất trong hóa học? Và điều gì khiến khái niệm này trở nên quan trọng trong lĩnh vực học máy? Sau đây chúng ta sẽ lần lượt giải đáp các vấn đề này.



Hình 1: Mối liên hệ giữa quá trình chưng cất trong hóa học và trong học máy.

Hãy bắt đầu bằng một so sánh đơn giản: Trong hóa học, chưng cất là quá trình tách các thành phần không cần thiết để thu được một chất tinh khiết. Nhưng trong học máy, tri thức không bị tách rời hay loại bỏ, thay vào đó là tập trung vào việc truyền tải tri thức ở dạng cô đọng từ một mô hình lớn (Teacher model) sang một mô hình nhỏ gọn hơn (Student model). Tri thức

(knowledge) trong AI là sự hiểu biết, kiến thức mà một Teacher model tích lũy được từ dữ liệu lớn thông qua quá trình huấn luyện mô hình. Từ đó, mô hình này đóng vai trò như một chuyên gia có khả năng đưa ra dự đoán chính xác. Tuy nhiên, việc triển khai toàn bộ sức mạnh của mô hình lớn trong thực tế thường rất phức tạp và đòi hỏi nguồn tài nguyên lớn. Từ đó, ý tưởng về việc "chưng cất" tri thức có được từ những mô hình lớn sang các mô hình nhỏ nhằm đạt được tốc độ xử lý nhanh nhưng vẫn đảm bảo độ chính xác cao được ra đời. Theo đó, mục tiêu của việc áp dụng kỹ thuật nêu trên không phải là để Student model sao chép hoàn toàn mọi tri thức từ Teacher model, mà là học được những gì cốt lõi và quan trọng nhất. Như vậy, ta phân biệt được, "chưng cất" trong hóa học **không hoàn toàn giống** với "chưng cất" trong học máy. Trong học máy, tri thức không bị tách riêng, mà được truyền tải ở dạng cô đọng, giữ lại những gì tinh túy nhất, quá trình này được gọi là **Knowledge Distillation (KD)**.

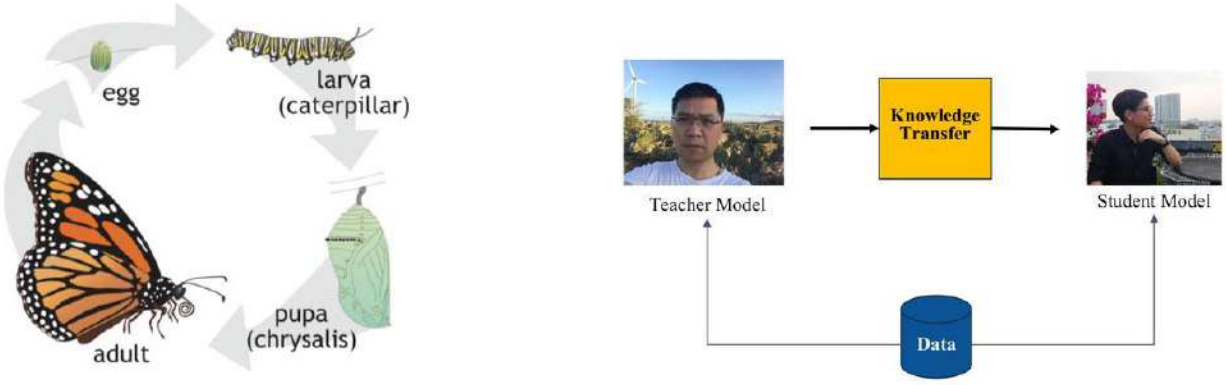
Hinton et al. (2015):

> "The process of distilling knowledge from one network into another can be thought of as transferring knowledge in a condensed form, similar to how nature optimizes energy and growth in biological processes."

Tạm dịch:

> "Quá trình chưng cất tri thức từ một kiến trúc mạng này sang một kiến trúc mạng khác có thể được xem như việc chuyển giao tri thức ở dạng cô đọng, tương tự cách mà tự nhiên tối ưu hóa năng lượng và sự phát triển trong các quá trình sinh học."

Hinton và cộng sự đã chỉ ra trong *"Distilling the Knowledge in a Neural Network"* (2015) – một trong những bài báo tiên phong áp dụng hiệu quả ý tưởng Knowledge Distillation (KD) trong deep learning, rằng KD là một quá trình không chỉ đơn thuần mang tính kỹ thuật, mà còn phản ánh sự hữu hiệu trong cách tự nhiên tối ưu hóa năng lượng và phát triển. Ý tưởng này được minh họa rõ ràng qua cách mà thiên nhiên vận hành. Các quá trình sinh học luôn chắt lọc, tập trung nguồn lực để đạt hiệu quả cao nhất.



Hình 2: Quá trình tối ưu hóa trong tự nhiên (trái) và chuyển giao tri thức (phải). Nguồn: link.

Hình minh họa bên trái hình 2 cho thấy vòng đời của loài bướm – từ trứng, sâu bướm, nhộng cho đến bướm trưởng thành – mỗi giai đoạn đều được tối ưu hóa cho một mục tiêu cụ thể. Tương tự, Knowledge Distillation là quá trình chuyển giao tri thức ở dạng cô đọng từ Teacher model cồng

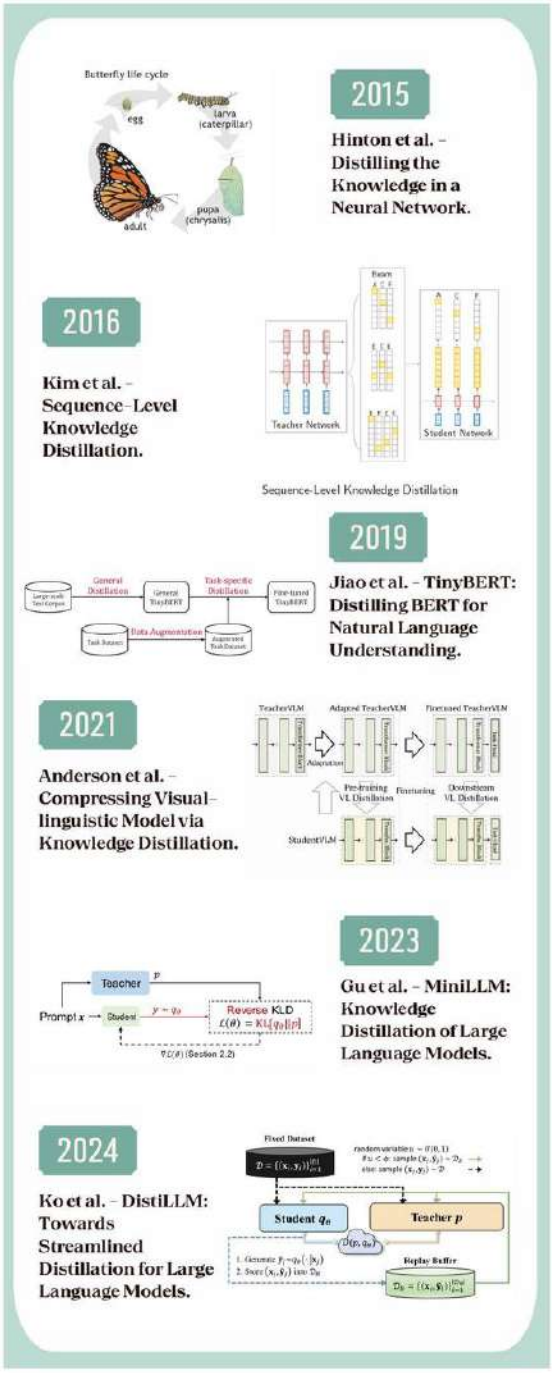
kênh, phức tạp sang Student model nhỏ gọn hơn. Hình minh họa bên phải hình 2 cho thấy cách Teacher model, một mô hình được đào tạo từ nguồn dữ liệu lớn, truyền tải phần tinh túy nhất của tri thức qua quá trình Knowledge Transfer. Thay vì truyền tải toàn bộ cấu trúc cồng kềnh của Teacher model, Knowledge Distillation tập trung vào những gì thực sự cần thiết, chẳng hạn như các dự đoán xác suất hoặc mối liên hệ giữa đầu vào và đầu ra. Điều này cho phép Student model có khả năng đạt hiệu suất gần như tương đương Teacher model, nhưng với kích thước và chi phí tính toán giảm đi đáng kể.

Model Name	n_{params}
GPT-3 Small	125M
GPT-3 Medium	350M
GPT-3 Large	760M
GPT-3 XL	1.3B
GPT-3 2.7B	2.7B
GPT-3 6.7B	6.7B
GPT-3 13B	13.0B
GPT-3 175B or "GPT-3"	175.0B

Hình 3: Số lượng tham số của GPT-3. Nguồn: link

Đặt trong bối cảnh hiện nay, các mô hình ngôn ngữ lớn (LLMs) với số lượng tham số rất cao, ví dụ như GPT-3 với 175 tỷ tham số, mang lại sức mạnh vượt trội, nhưng đồng thời cũng đi kèm những thách thức lớn về tài nguyên. Đầu tiên là tiêu tốn lượng lớn RAM và GPU trong quá trình huấn luyện và thứ hai là khó triển khai trên thực tế do yêu cầu cao về tốc độ xử lý và tối ưu hóa tài nguyên. Đặc biệt, các ứng dụng trên thiết bị di động hoặc hệ thống thời gian thực thường không đủ sức đáp ứng. Trong bối cảnh này, Knowledge Distillation trở thành một trong những giải pháp lý tưởng, mở ra cánh cửa để áp dụng LLMs hiệu quả vào thế giới thực.

Khi nói về tính ứng dụng, kỹ thuật Knowledge Distillation (KD) còn có thể được áp dụng vào trong nhiều bài toán, từ xử lý ngôn ngữ tự nhiên (NLP) đến nhận diện hình ảnh hay nhận diện giọng nói như một cách thức nhằm tăng độ chính xác và giảm độ phức tạp của mô hình mục tiêu. Điều này đóng vai trò quan trọng trong nghiên cứu về AI cũng như việc ứng dụng các mô hình AI phức tạp vào thực tế.



Hình 4: Khảo sát nghiên cứu Knowledge Distillation trong giai đoạn từ 2015–2024.

Hình 4 minh họa sự phát triển của một số nghiên cứu nổi bật về KD từ năm 2015 đến 2024. Khởi đầu, Hinton et al. (2015) giới thiệu KD như một phương pháp truyền tri thức từ mô hình lớn (Teacher) sang mô hình nhỏ (Student) thông qua *soft logits*, giúp mô hình nhỏ đạt hiệu suất gần tương đương ban đầu.

Kim et al. (2016) mở rộng KD với *Sequence-Level Knowledge Distillation*, hỗ trợ các bài toán xử lý chuỗi như dịch máy. Tiếp nối, Jiao et al. (2019) phát triển *TinyBERT*, tối ưu hóa BERT cho các thiết bị tài nguyên hạn chế nhưng vẫn đảm bảo hiệu quả trên các tác vụ NLP.

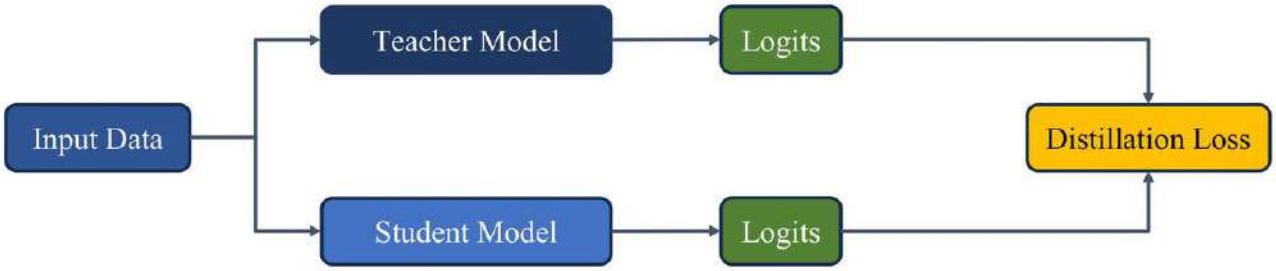
Đến năm 2021, Anderson et al. ứng dụng KD vào mô hình đa phương thức với *Compressing Visual-linguistic Models*, hỗ trợ bài toán như VQA và chú thích hình ảnh, cân bằng giữa kích thước mô hình và hiệu suất.

Năm 2023, Gu et al. giới thiệu *MiniLLM*, cải tiến *SeqKD* để nén các LLMs mà vẫn duy trì hiệu suất cao. Cuối cùng, Ko et al. (2024) phát triển *DistiLLM*, tối ưu quy trình KD cho ứng dụng thời gian thực như trợ lý ảo và hệ thống NLP đối thoại.

Các nghiên cứu này không chỉ ghi nhận những cột mốc quan trọng mà còn thể hiện vai trò của KD trong việc đưa các hệ thống AI từ lý thuyết đến thực tiễn, đáp ứng nhu cầu ngày càng cao của thế giới hiện đại.

Tri thức trong **Knowledge Distillation (KD)** được truyền tải thông qua ba dạng chính, mỗi dạng lại cung cấp một cách tiếp cận độc đáo để khai thác và truyền tải thông tin:

- **Response-based Knowledge**: Sử dụng đầu ra (logits) của mô hình lớn (**Teacher**) để hướng dẫn mô hình nhỏ (**Student**). Phương pháp này được đánh giá cao nhờ sự đơn giản và khả năng ứng dụng rộng rãi, trở thành nền móng cho các nghiên cứu KD hiện đại.
- **Feature-based Knowledge**: Khai thác thông tin từ các tầng trung gian của Teacher, giúp Student học cách biểu diễn đặc trưng dữ liệu một cách chi tiết và chính xác.
- **Relation-based Knowledge**: Tập trung vào việc mô phỏng các mối quan hệ giữa các điểm dữ liệu, từ đó hỗ trợ Student hiểu rõ cách Teacher tổ chức và xử lý dữ liệu trong không gian đặc trưng.



Hình 5: Pipeline mô phỏng hướng tiếp cận theo Response-based Knowledge trong KD.

Trong số ba dạng trên, **Response-based Knowledge** được xem như một trong những viên gạch đầu tiên của KD, đồng thời là phương pháp được sử dụng phổ biến nhờ tính dễ hiểu và hiệu quả trong thực tiễn.

Tri thức này được truyền tải thông qua các logits – đầu ra trước khi áp dụng hàm softmax của Teacher. Điểm đặc biệt của logits nằm ở việc chúng không chỉ chứa thông tin về nhãn đúng mà còn phản ánh mức độ tự tin của Teacher đối với các nhãn khác. Chính điều này tạo nên một nguồn tri thức mềm, giúp Student học không chỉ từ đáp án đúng mà còn từ cách Teacher "nhìn nhận" các đáp án khác.

Tuy nhiên, có một thách thức là logits thường quá sắc nét (sharp), khiến thông tin về các nhãn khác bị làm mờ đi. Để khắc phục, kỹ thuật **Temperature Scaling** được áp dụng nhằm "làm mềm" logits. Bằng cách chia logits cho tham số $T > 1$, phân phối xác suất trở nên mượt mà hơn, tạo điều kiện để Student học hiệu quả hơn từ Teacher.

Một thành phần không thể thiếu trong Response-based Knowledge là **KL-Divergence (Kullback-Leibler Divergence)**. KL-Divergence không chỉ là một thước đo sự khác biệt giữa hai phân phối xác suất, mà còn đóng vai trò quan trọng trong việc giúp Student điều chỉnh logits sao cho mô phỏng sát nhất cách Teacher hiểu dữ liệu.

$$D_{KL}(P_{teacher} \| P_{student}) = \sum_i P_{teacher,i} \cdot (\log P_{teacher,i} - \log P_{student,i}) \tag{1}$$

- $P_{teacher,i}$: Xác suất của Teacher cho lớp i, được làm mềm bởi tham số T.
- $P_{student,i}$: Xác suất của Student cho lớp i, cũng làm mềm bởi T.

Nhờ có KL-Divergence, Student không chỉ sao chép được các logits mà còn nắm bắt được cách Teacher đánh giá cấu trúc tổng thể của dữ liệu, từ đó cải thiện khả năng học sâu và tổng quát hóa.

Distillation loss trong Response-based KD là sự kết hợp hai thành phần: **Cross-Entropy Loss**, nhằm đảm bảo Student học đúng nhãn mục tiêu, và **KL-Divergence**, giúp truyền tải tri thức mềm từ Teacher:

$$L_{KD} = (1 - \alpha) \cdot L_{CE}(y, z_s) + \alpha \cdot T^2 \cdot L_{KL}(p(z_t, T) \| p(z_s, T)) \tag{2}$$

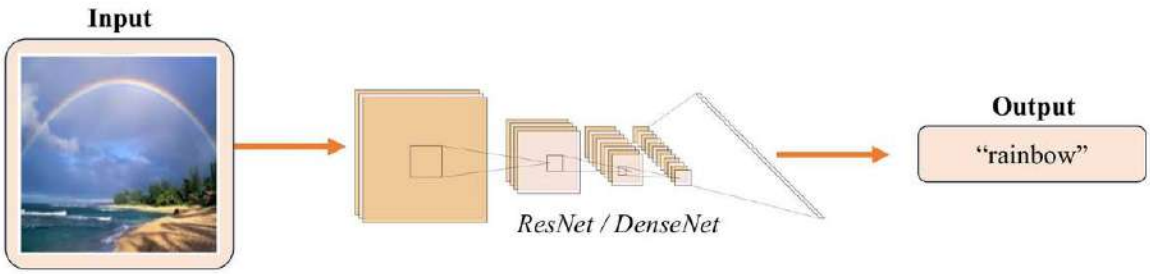
- $L_{CE}(y, z_s)$: Cross-Entropy Loss giữa nhãn đúng y và logits của Student.
- α: Tham số cân bằng giữa hai thành phần loss.
- T: Tham số nhiệt độ làm mềm logits của cả Teacher và Student.
- L_{KL}: Kullback-Leibler Divergence Loss giữa teacher logits và student logits

Phương pháp này không chỉ đảm bảo rằng Student học được nhãn đúng mà còn giúp mô phỏng cách Teacher phân phối trọng số giữa các nhãn, từ đó tăng cường hiệu suất trên dữ liệu chưa từng gặp.

KL-Divergence và Temperature Scaling không chỉ giúp Student bắt chước Teacher mà còn hỗ trợ việc truyền tải tri thức mềm – điều mà các phương pháp truyền thống khó có thể đạt được. Điều này đặc biệt hữu ích trong các bài toán yêu cầu khả năng tổng quát hóa cao, như xử lý ngôn ngữ tự nhiên trên thiết bị di động, nhận diện hình ảnh, và các hệ thống thời gian thực. Với tính đơn giản và hiệu quả, **Response-based Knowledge Distillation** không chỉ đặt nền móng cho KD mà còn mở ra nhiều hướng phát triển đầy hứa hẹn.

II. Cài đặt chương trình

Trong phần này, chúng ta sẽ từng bước cài đặt kỹ thuật Knowledge Distillation (KD) đã đề cập ở phần trước sử dụng thư viện PyTorch. Bài toán phân loại ảnh thời tiết sẽ được chọn làm bài toán cần giải quyết trong bài này. Theo đó, Input và Output của tác vụ này có thể được mô tả như sau:



Hình 6: Input và output của bài toán phân loại ảnh thời tiết bằng mô hình ResNet hoặc DenseNet.

Để mô phỏng phần nào ngữ cảnh sử dụng KD, chúng ta sẽ chọn kiến trúc mô hình cho Student và Teacher với điều kiện kích thước mô hình của Student phải nhỏ hơn so với Teacher. Ở đây, vì là bài toán phân loại ảnh, ResNet18 ($\sim$ 11.7 triệu tham số) sẽ được sử dụng cho mô hình Student và DenseNet169 ($\sim$ 14.3 triệu tham số) cho Teacher.

Các bước cài đặt code sẽ được mô tả theo ba giai đoạn huấn luyện mô hình như sau:

1. **Student only:** Bước này được thực hiện chỉ để có một sự so sánh kết quả giữa mô hình student không được áp dụng KD và được áp dụng KD. Bạn đọc có thể bỏ qua bước này nếu chỉ quan tâm đến KD. Trong bài viết này, việc train riêng mô hình student sẽ được đề cập.

2. **Teacher only:** Bước này được thực hiện để tạo ra một mô hình teacher đủ tốt, từ đó tham gia vào quá trình học của mô hình student trong KD để cải thiện hiệu suất.

3. **Knowledge Distillation (KD):** Bước này được thực hiện để huấn luyện mô hình student với mô hình teacher đã huấn luyện ở bước trên, ứng dụng kỹ thuật Knowledge Distillation.

II.I. Huấn luyện riêng mô hình Student và Teacher

Phần code này dùng để huấn luyện cho cả Student và Teacher, với các code của Teacher được đặt trong comment.

1. **Tải bộ dữ liệu:** Trước hết, cần tải xuống tập dữ liệu Weather và tiến hành giải nén.

```
# https://drive.google.com/file/d/1fnJMMw0LvDgl-GS4FTou5qAgLx0E2KQ0/view?usp=drive_link
!gdown 1fnJMMw0LvDgl-GS4FTou5qAgLx0E2KQ0
!unzip img_cls_weather_dataset.zip  # Giải nén

```

Dưới đây là một số hình ảnh bên trong bộ dữ liệu này:

Hình 7: Một số hình ảnh từ bộ dữ liệu.

2. **Import các thư viện cần thiết:** Kế tiếp là thêm các thư viện cần dùng.

```
import os
import timm
import torch
import random
import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt

from PIL import Image
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split

```

3. **Cài đặt tham số ngẫu nhiên cố định:** Nhằm đảm bảo kết quả có thể tái tạo lại, việc cần làm là chỉ định một seed như code sau:

```
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
```

```
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

seed = 59
set_seed(seed)

```

4. **Đọc bộ dữ liệu:** Trong bước này, những thông tin cần thu thập là các loại thời tiết (classes) có trong bộ dữ liệu, sau đó, tổng hợp các đường dẫn (img_paths) và nhãn (labels) tương ứng của các hình ảnh lại để chuẩn bị cho bước tiếp theo.

```
root_dir = 'weather-dataset/dataset'
classes = {
    label_idx: class_name \
        for label_idx, class_name in enumerate(
            sorted(os.listdir(root_dir))
        )
}

img_paths = []
labels = []
for label_idx, class_name in classes.items():
    class_dir = os.path.join(root_dir, class_name)
    for img_filename in os.listdir(class_dir):
        img_path = os.path.join(class_dir, img_filename)
        img_paths.append(img_path)
        labels.append(label_idx)

```

5. **Chia tập train/val/test:** Các thông tin vừa được tổng hợp sẽ dùng để xây dựng các bộ train, validation, test theo tỉ lệ lần lượt là 0.7/0.2/0.1 theo đoạn code bên dưới:

```
val_size = 0.2
test_size = 0.125
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    img_paths, labels,
    test_size=val_size,
    random_state=seed,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=test_size,
    random_state=seed,
    shuffle=is_shuffle
)

```

6. **Xây dựng PyTorch Dataset:** Tiếp đến, sử dụng lớp Dataset của Pytorch để quản lý và tổ chức của bộ dữ liệu, ở đây là bộ Weather. Lớp WeatherDataset bên dưới sẽ được khởi tạo bằng danh sách các đường dẫn *(X)* và nhãn tương ứng *(y)* để tạo thành các mẫu (sample), với mỗi sample gồm hình ảnh đã được đọc và biến đổi (transform) cùng với nhãn tương ứng (như kết quả trả về từ *__getitem__()*). Theo đó, khởi tạo các bộ train/validation/test bằng các danh sách *X, y* tương ứng, cùng với các biến đổi (transform) cần thiết như thay đổi kích thước ảnh (resize) về 224x244 và chuẩn hoá về khoảng [0,1].

```
class WeatherDataset(Dataset):
    def __init__(
        self,
        X, y,
        transform=None
    ):
        self.transform = transform
        self.img_paths = X
        self.labels = y

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, idx):
        img_path = self.img_paths[idx]
        img = Image.open(img_path).convert("RGB")

        if self.transform:
            img = self.transform(img)

        return img, self.labels[idx]

def transform(img, img_size=(224, 224)):
    img = img.resize(img_size)
    img = np.array(img)[..., :3]
    img = torch.tensor(img).permute(2, 0, 1).float()
    normalized_img = img / 255.0

    return normalized_img

train_dataset = WeatherDataset(
    X_train, y_train,
    transform=transform
)
val_dataset = WeatherDataset(
    X_val, y_val,
    transform=transform
)
test_dataset = WeatherDataset(
    X_test, y_test,
    transform=transform
)

```

7. **Xây dựng PyTorch DataLoader:** Nối tiếp bước trên, DataLoader sẽ chịu trách nhiệm hỗ trợ tải dữ liệu từ Dataset một cách hiệu quả bằng cách việc chia dữ liệu thành các lô

(batches), xáo trộn (shuffling) và quản lý tải dữ liệu song song.

```
# Student
train_batch_size = 256
# Teacher
# train_batch_size = 32

test_batch_size = 128

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batch_size,
    shuffle=True
)
val_loader = DataLoader(
    val_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
test_loader = DataLoader(
    test_dataset,
    batch_size=test_batch_size,
    shuffle=False
)

```

8. **Xây dựng mô hình:** Sau khi xây dựng hoàn chỉnh bộ dữ liệu, sử dụng thư viện timm để nhanh chóng triển khai mô hình cần thiết như ResNet và DenseNet. Theo code bên dưới, mô hình sẽ được khởi tạo với các tham số đã được huấn luyện trước và thư viện sẽ tạo mới lớp linear cuối cùng với số lượng node đầu ra bằng với tham số *num_ classes*.

```
n_classes = len(list(classes.keys()))
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# Student
model = timm.create_model(
    'resnet18',
    pretrained=True,
    num_classes=n_classes
).to(device)

# Teacher
# model = timm.create_model(
#     'densenet169',
#     pretrained=True,
#     num_classes=n_classes
# ).to(device)

```

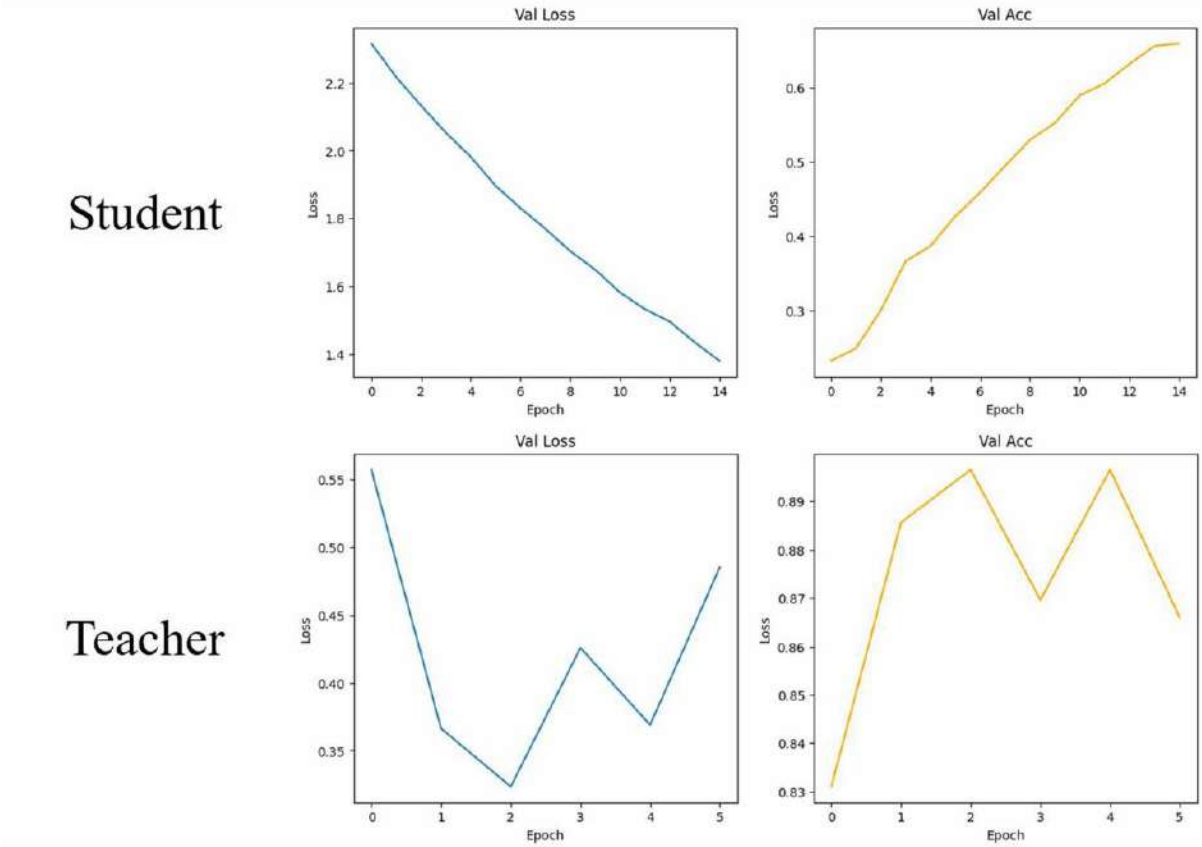
Khi cài đặt, trong trường hợp sử dụng Google Colab, các bạn có thể tách việc huấn luyện mô hình Student và Teacher ở hai file khác nhau hoặc trên cùng một file. Với từng loại model, các bạn hãy sử dụng đoạn code tương ứng cho phần đó (trong trường hợp đoạn code trên đang khởi tạo cho model Student, các bạn cần làm tương tự cho Teacher ở một file khác hoặc đổi tên biến khai báo hai model này khi code chung một file).

9. **Huấn luyện mô hình:** Tại bước huấn luyện này, cần phải xây dựng một hàm *evaluate()* để đánh giá mô hình trên tập validation trong quá trình huấn luyện và hàm *fit()* dùng để huấn luyện mô hình trên theo dạng bài toán phân loại ảnh. Cuối cùng là định nghĩa cấu hình huấn luyện như số epoch, learning rate, ... và tiến hành đào tạo mô hình với kỹ thuật early stopping để dừng sớm quá trình huấn luyện nếu không có cải thiện trên validation loss.

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            losses.append(loss.item())
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    loss = sum(losses) / len(losses)
    acc = correct / total

    return loss, acc

def fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs,
    patience
):
    train_losses = []
    val_losses = []
    val_accs = []

    best_val_loss = float('inf')
    patience_counter = 0

    for epoch in range(epochs):
        batch_train_losses = []

        model.train()
        for idx, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
```

```
            optimizer.step()

            batch_train_losses.append(loss.item())

        train_loss = sum(batch_train_losses) / len(batch_train_losses)
        train_losses.append(train_loss)

        val_loss, val_acc = evaluate(
            model, val_loader,
            criterion, device
        )
        val_accs.append(val_acc)
        val_losses.append(val_loss)

        print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal
    loss: {val_loss:.4f}\tVal acc: {val_acc:.4f}')

        # Check early stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience:
            print(f"Early stopping triggered after {epoch + 1} epochs.")
            break

    return train_losses, val_losses, val_accs

lr = 1e-2
epochs = 15
patience = 3

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(
    model.parameters(),
    lr=lr
)

train_losses, val_losses, val_accs = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs,
    patience
)

fig, ax = plt.subplots(1, 2, figsize=(12, 5))
ax[0].plot(val_losses)
ax[0].set_title('Val Loss')
ax[0].set_xlabel('Epoch')
```

```
ax[0].set_ylabel('Loss')
ax[1].plot(val_accs, color='orange')
ax[1].set_title('Val Acc')
ax[1].set_xlabel('Epoch')
ax[1].set_ylabel('Loss')
plt.show()
```



Hình 8: Kết quả sau khi huấn luyện Student và Teacher.

10. **Đánh giá mô hình:** Cuối cùng là bước đánh giá mô hình trên tập test và validation để có kết quả cuối cùng.

```
val_loss, val_acc = evaluate(
    model,
    val_loader,
    criterion,
    device
)
test_loss, test_acc = evaluate(
    model,
    test_loader,
    criterion,
    device
```

```
)

print('Evaluation on val/test dataset')
print('Val accuracy: ', val_acc)
print('Test accuracy: ', test_acc)

# Student:
# Val accuracy:  0.6598689002184996
# Test accuracy:  0.6462882096069869

# Teacher:
# Val accuracy:  0.8659868900218499
# Test accuracy:  0.8879184861717613

```

Như vậy, thông qua kết quả đánh giá, mô hình Teacher đạt hiệu suất cao hơn mô hình Student (chưa áp dụng KD) trên cùng một cài đặt các tham số huấn luyện mô hình. Đây là điều mà chúng ta mong muốn bởi lẽ, một mô hình Teacher tốt sẽ có thể hỗ trợ mô hình Student học tốt hơn trên cài đặt KD phần ở sau.

11. **Lưu mô hình (chỉ Teacher):** Để chuẩn bị cho phần Knowledge Distillation kế tiếp, cần phải lưu lại mô hình Teacher bằng code sau:

```
# Teacher
save_path = '/content/teacher_wt.pt'
torch.save(model.state_dict(), save_path)

```

II.II. Huấn luyện mô hình Student với KD

1. **Xây dựng mô hình:** Thay đổi chính ở phần này là cần khai báo 2 mô hình, trong đó, sẽ tải lại Teacher từ file.pt đã lưu và khởi tạo Student từ đầu.

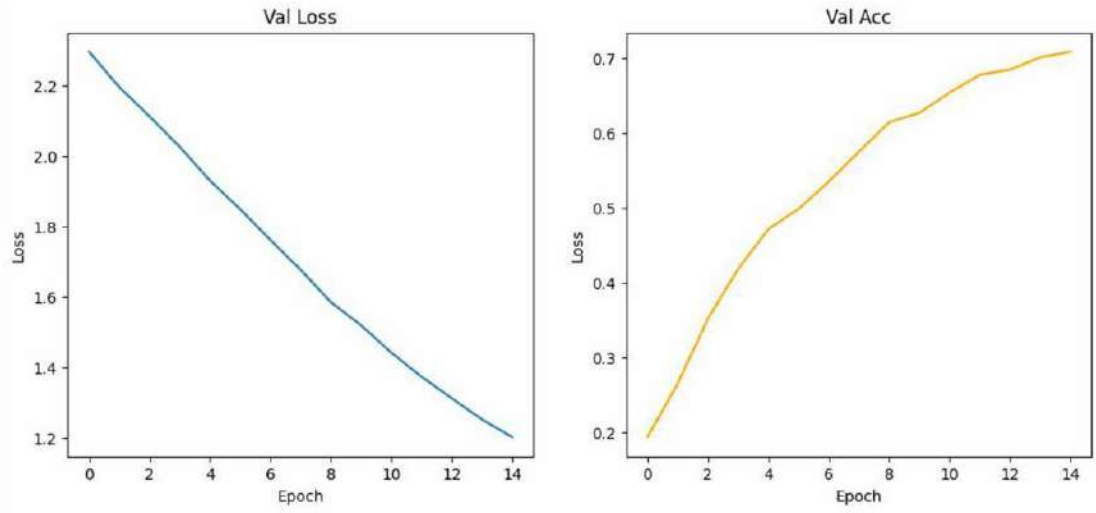
```
n_classes = len(list(classes.keys()))
device = 'cuda' if torch.cuda.is_available() else 'cpu'

teacher_model = timm.create_model(
    'densenet169',
    pretrained=True,
    num_classes=n_classes
).to(device)

teacher_model.load_state_dict(
    torch.load(
        'teacher_wt.pt',
        map_location=device
    )
)

student_model = timm.create_model(
    'resnet18',
    pretrained=True,
    num_classes=n_classes
```

```
).to(device)

```

2. **Huấn luyện mô hình theo KD:** Để chuyển giao tri thức từ Teacher sang Student, ta chỉ sử dụng Teacher với mục đích lấy output mà không làm ảnh hưởng đến các tham số bên trong bằng cách chuyển Teacher sang chế độ đánh giá bằng hàm teacher.eval(). Hàm Loss cuối cùng sẽ bao gồm Soft Target Loss và Cross Entropy Loss như đã đề cập tại phần lý thuyết. Với Target Loss sử dụng output từ 2 mô hình, trong khi Cross Entropy Loss sử dụng output từ Student và nhãn đúng.

```
# ... Tương tự phần code trước

def fit(
    teacher,
    student,
    T,
    soft_target_loss_weight,
    ce_loss_weight,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs,
    patience
):
    train_losses = []
    val_losses = []
    val_accs = []
    teacher.eval()

    best_val_loss = float('inf')
    patience_counter = 0

    for epoch in range(epochs):
        batch_train_losses = []

        student.train()
        for idx, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()

            with torch.no_grad():
                teacher_logits = teacher(inputs)
            student_logits = student(inputs)

            soft_targets = nn.functional.softmax(teacher_logits / T, dim
    =-1)
            soft_prob = nn.functional.log_softmax(student_logits / T, dim
    =-1)

            soft_targets_loss = torch.sum(soft_targets * (soft_targets.
    log() - soft_prob)) / soft_prob.size()[0] * (T**2)
```

```
            label_loss = criterion(student_logits, labels)

            loss = soft_target_loss_weight * soft_targets_loss + ce_loss_weight * label_loss

            loss.backward()
            optimizer.step()

            batch_train_losses.append(loss.item())

        train_loss = sum(batch_train_losses) / len(batch_train_losses)
        train_losses.append(train_loss)

        val_loss, val_acc = evaluate(
            student, val_loader,
            criterion, device
        )
        val_accs.append(val_acc)
        val_losses.append(val_loss)

        print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}\tVal acc: {val_acc:.4f}')

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience:
            print(f"Early stopping triggered after {epoch + 1} epochs.")
            break

    return train_losses, val_losses, val_accs

# ... Tương tự phần code trước

train_losses, val_losses, val_accs = fit(
    teacher_model,
    student_model,
    2,
    0.25,
    0.75,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs,
    patience
)

# Tương tự như Student
...

```



Hình 9: Kết quả kỳ vọng sau khi huấn luyện với Knowledge Distillation.

3. **Đánh giá mô hình:** Kết quả trên Student được huấn luyện thông qua KD đem lại hiệu suất cao hơn Student được huấn luyện như thông thường.

```
# ... Tương tự phần code trước

# KD Student
# Val accuracy:  0.7079388201019665
# Test accuracy:  0.6783114992721979

```

Tổng hợp lại các kết quả huấn luyện trên, ta có một bảng đánh giá hiệu suất các mô hình như sau:

Methods	Val Accuracy	Test Accuracy
Student	0.6599	0.6463
Teacher	0.8660	0.8879
KD Student	0.7079	0.6783

Bảng 1: Bảng tổng hợp kết quả đánh giá mô hình trong nội dung bài hướng dẫn này. Trong đó, Student/Teacher chỉ mô hình được huấn luyện một cách thông thường (supervised learning). KD Student chỉ mô hình Student được huấn luyện với cài đặt KD.

Có thể thấy, kỹ thuật Knowledge Distillation (KD) đã cải thiện đáng kể hiệu suất của mô hình Student so với việc huấn luyện truyền thống, đúng với kỳ vọng trong phần lý thuyết mà chúng ta đã thảo luận. Dù vậy, thuật toán KD được sử dụng trong bài chưa thể làm cho mô hình Student đạt được tiệm cận kết quả của Teacher, cho thấy mặt hạn chế cũng như tiềm năng để phát triển tiếp đối với nhóm kỹ thuật KD này.

- Hết -