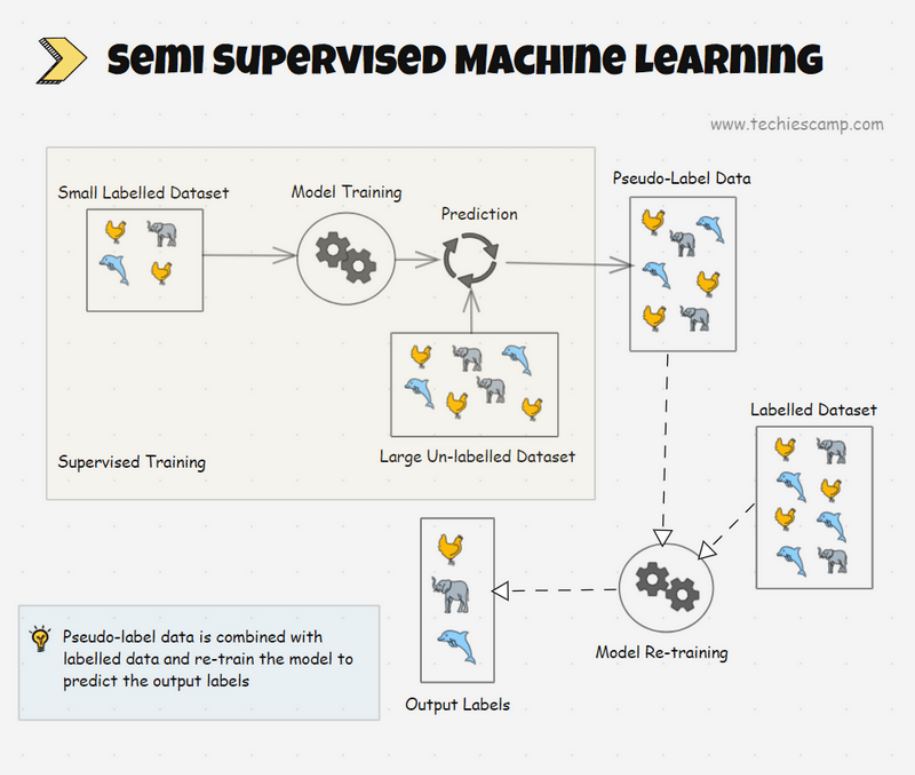
AI VIET NAM – AIO2024

Introduction to Semi-Supervised Learning: Co-training Method

Hoang-Thien Nguyen and Thanh-Huy Nguyen

Ngày 11 tháng 3 năm 2025

I. Giới thiệu



Hình 1: Hình minh họa quy trình học máy bán giám sát: Mô hình được huấn luyện ban đầu trên tập dữ liệu nhỏ có gán nhãn (Small Labelled Dataset) để tạo ra dự đoán cho tập dữ liệu lớn chưa gán nhãn (Large Un-labelled Dataset). Các dự đoán này trở thành nhãn giả (Pseudo-Label Data) và được kết hợp với dữ liệu gán nhãn ban đầu để tạo ra tập dữ liệu hoàn chỉnh hơn. Sau đó, mô hình được huấn luyện lại (Model Re-training) trên tập dữ liệu mở rộng này nhằm cải thiện độ chính xác và khả năng tổng quát khi dự đoán nhãn đầu ra (Output Labels). [1]

Học bán giám sát (Semi-Supervised Learning - SSL) là một hướng quan trọng trong học máy, nổi bật nhờ khả năng tận dụng dữ liệu không nhãn (unlabeled data) kết hợp với dữ liệu có nhãn (labeled data) để cải thiện hiệu suất mô hình (được mô tả thông qua hình 1). Trong thực tế, dữ liệu có nhãn thường khan hiếm và tốn kém để thu thập, trong khi dữ liệu không nhãn lại rất phổ biến. Đây là lý do khiến SSL trở thành một giải pháp hiệu quả cho nhiều bài toán như nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên (NLP),....

Một số hướng tiếp cận trong SSL:

- Self-Training: Mô hình huấn luyện trên dữ liệu có nhãn, sau đó tự gán nhãn dữ liệu không nhãn và tiếp tục học. Tuy nhiên, nếu nhãn giả (pseudo labels) sai, lỗi sẽ tích lũy và làm suy giảm hiệu suất.
- Graph-Based Learning: Xây dựng mạng đồ thị để lan truyền nhãn từ dữ liệu có nhãn sang dữ liệu không nhãn. Hạn chế chính là yêu cầu cấu trúc đồ thị chất lượng cao và dễ bị ảnh hưởng bởi nhiễu trong dữ liệu.
- Consistency Regularization: Khuyến khích mô hình đưa ra dự đoán nhất quán khi dữ liệu đầu vào bị biến đổi hoặc thêm nhiễu. Tuy nhiên, hiệu quả phụ thuộc mạnh vào việc lựa chọn dạng biến đổi và mức độ nhiễu phù hợp.

Những hạn chế này đã thúc đẩy sự ra đời của **Co-Training**, một phương pháp mạnh mẽ trong SSL tận dụng sức mạnh bổ trợ từ nhiều mô hình khác nhau.

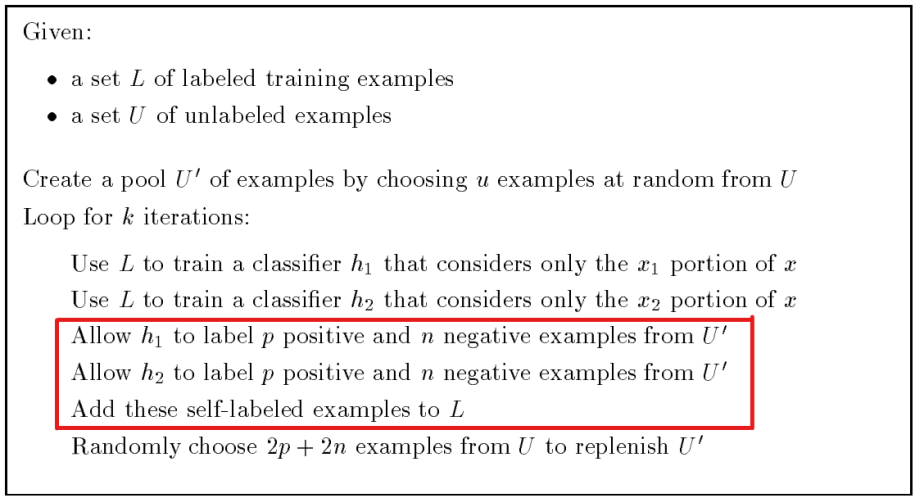
II. Phương Pháp

A. Co-Training

Co-training là một phương pháp phổ biến trong semi-supervised learning dựa trên ý tưởng sử dụng hai bộ phân loại (classifiers) được huấn luyện trên hai tập đặc trưng (feature views) khác nhau của cùng một tập dữ liệu. Cách thức hoạt động sơ khai của co-training bao gồm:

1. Hai mô hình học trên tập dữ liệu có nhãn.
2. Hai mô hình dùng dữ liệu không nhãn (unlabeled data) để dự đoán.
3. Hai mô hình trao đổi những dự đoán tự tin nhất để mở rộng tập dữ liệu có nhãn (labeled data).
4. Huấn luyện lại bộ mô hình bằng tập dữ liệu được mở rộng.

Ý tưởng ban đầu của Co-training có thể được biểu diễn thông qua hình 2, được giới thiệu bởi Avrim Blum và Tom Mitchell (1998) [2], xuất phát từ nhu cầu giảm phụ thuộc vào dữ liệu có nhãn (vốn tốn kém) bằng cách sử dụng hiệu quả dữ liệu không nhãn.



Hình 2: Hình minh họa ý tưởng cơ bản của co-training: Hai mô hình được huấn luyện trên các phần khác nhau của dữ liệu. Mỗi mô hình tự gán nhãn một số mẫu từ tập chưa gán nhãn và thêm vào tập dữ liệu huấn luyện để cải thiện dần qua từng vòng lặp.

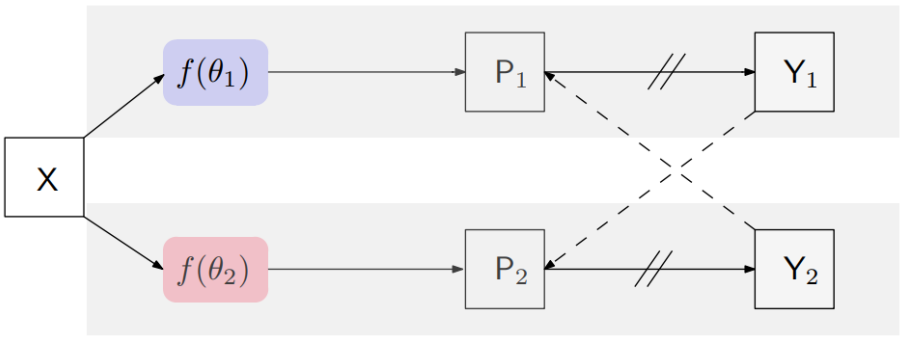
Có thể hiểu động lực chính của Co-training được dùng để để:

- Tận dụng dữ liệu không nhãn: Trong thực tế, dữ liệu có nhãn thường khan hiếm hoặc tốn kém để thu thập, trong khi dữ liệu không nhãn lại phong phú.
- Bổ sung tri thức từ các khía cạnh khác nhau: Nếu một mẫu dữ liệu được mô tả dưới hai khía cạnh khác nhau nhưng đều có khả năng phân loại tốt, việc kết hợp hai khía cạnh này sẽ giúp cải thiện độ chính xác.
- Giảm lệch và tăng độ tổng quát: Hai bộ phân loại độc lập sẽ tự bù trừ các điểm yếu của nhau, dẫn đến một mô hình chính xác và ổn định hơn.

Tổng quan về Co-training: Phương pháp Co-training tiêu chuẩn Được giới thiệu bởi Blum và Mitchell (1998) [2], phương pháp này sử dụng hai bộ phân loại học từ hai khía cạnh (views) khác nhau của dữ liệu. Hai bộ phân loại sẽ lần lượt gán nhãn cho dữ liệu chưa có nhãn và bổ sung vào tập huấn luyện của nhau. Co-EM [3] là sự kết hợp giữa Co-training và thuật toán EM (Expectation-Maximization). Hai bộ phân loại được huấn luyện trên các khía cạnh khác nhau và cùng tối ưu hoá hàm khả năng tối đa (likelihood) trong quá trình lặp. Tri-training [4] là biến thể mở rộng của co-training, tri-training sử dụng ba bộ phân loại và không yêu cầu hai khía cạnh độc lập. Bộ phân loại thứ ba được huấn luyện từ các mẫu mà hai bộ còn lại đồng thuận. Multi-view Learning [5] là sự khái quát của co-training, multi-view learning sử dụng nhiều hơn hai khía cạnh (views) của dữ liệu, với nguyên lý kết hợp thông tin từ nhiều khía cạnh để nâng cao hiệu suất học, phù hợp với các tập dữ liệu có nhiều đặc trưng.

Từ phương pháp Co-training tiêu chuẩn [2] dựa trên hai khía cạnh độc lập đến Tri-training [4] với ba bộ phân loại, các phương pháp này đều tập trung vào việc khai thác sự đồng thuận giữa nhiều bộ phân loại. Tuy nhiên, những hạn chế như giả định tính độc lập giữa các khía cạnh và nguy cơ lan truyền nhãn sai đã thúc đẩy sự phát triển của các phương pháp tiên tiến hơn. Một trong những tiến bộ đáng chú ý là phương pháp Cross Pseudo Supervision (CPS) [6], mở rộng nguyên lý đồng thuận bằng cách kết hợp các bộ phân loại học tập chéo và tối ưu hóa thông qua các nhãn giả (pseudo labels). Phương pháp này không chỉ kế thừa ý tưởng từ Co-training mà còn khắc phục các hạn chế, giúp nâng cao độ chính xác trong các bài toán học sâu hiện đại.

B. Cross-Pseudo Supervision (CPS)



Hình 3: Sơ đồ minh họa Cross Pseudo Supervision (CPS): Hai mô hình học song song từ cùng tập dữ liệu. Mỗi mô hình dự đoán và gán nhãn giả cho mô hình còn lại.

Ý tưởng chính của CPS (Hình 3): sử dụng hai mô hình giống nhau (hoặc tương tự nhau) nhưng khác nhau về góc nhìn để giám sát lẫn nhau thông qua các nhãn giả (pseudo labels), từ đó cải thiện chất lượng học từ dữ liệu không nhãn. Sơ lược về cách thức hoạt động

1. Khởi tạo hai mô hình: Sử dụng hai mô hình có cùng kiến trúc nhưng khởi tạo khác nhau với trọng số khác nhau.

2. Huấn luyện bằng dữ liệu có nhãn (Labeled Data): Cả hai mô hình được huấn luyện bình thường bằng dữ liệu có nhãn với hàm mất mát giám sát (supervised loss), thường là Cross-Entropy Loss.

3. Tạo nhãn giả (Pseudo labels) từ dữ liệu không nhãn (Unlabeled Data): Khi đưa dữ liệu không nhãn vào, mỗi mô hình tạo ra nhãn giả (pseudo labels) dựa trên đầu ra của chính nó. Tuy nhiên, thay vì sử dụng nhãn giả từ chính mô hình để cập nhật trọng số của nó, mỗi mô hình sử dụng nhãn giả từ mô hình còn lại. Đây chính là cơ chế "cross" supervision.

4. Huấn luyện bằng nhãn giả (Unlabeled Loss): Sử dụng nhãn giả từ mô hình A để tính hàm mất mát cho mô hình B, và ngược lại. Thông thường, hàm mất mát này là Consistency Loss, nhằm đảm bảo hai mô hình dự đoán tương đồng trên dữ liệu không nhãn.

Hàm Loss của CPS được biểu hiện dưới dạng:

$$\mathcal{L} = \mathcal{L}_{sup}^{A} + \mathcal{L}_{sup}^{B} + \lambda(\mathcal{L}_{unsup}^{A \to B} + \mathcal{L}_{unsup}^{B \to A}) \tag{1}$$

Trong đó:

- A, B là hai mô hình có được khởi tạo với trọng số khác nhau.
- $\mathcal{L}_{sup}$: supervised loss được tính dựa trên prediction với groundtruth của labeled set.
- $\mathcal{L}_{unsup}$: loss được tính dựa trên việc trao đổi chéo pseudo labels giữa hai model trên unlabeled set.
- λ được gọi là hệ số cân bằng giữa $\mathcal{L}_{sup}$ và $\mathcal{L}_{unsup}$.

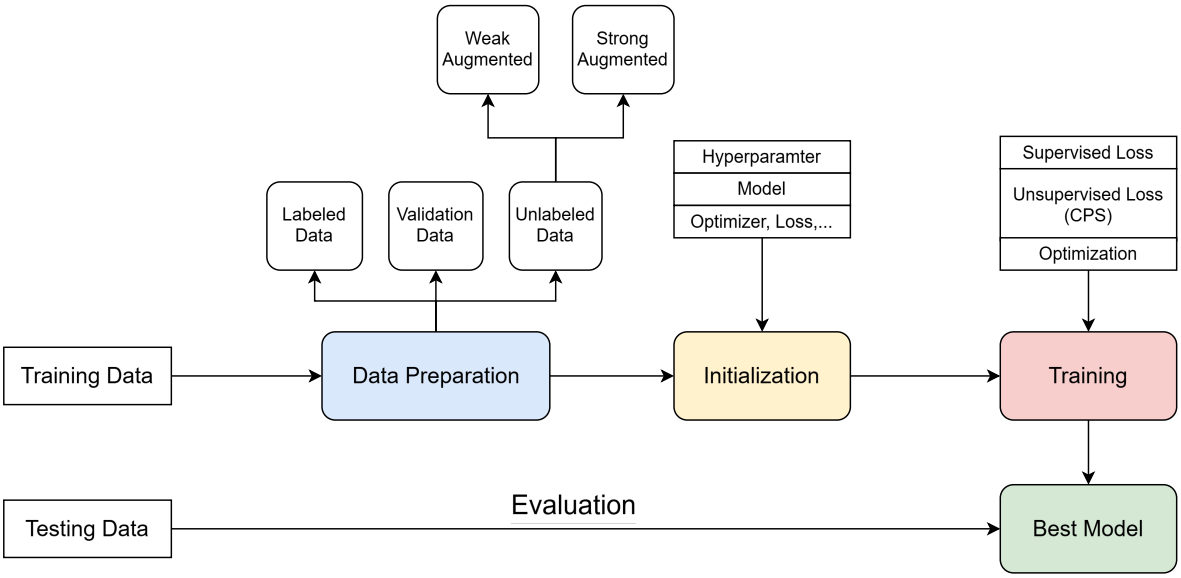
Ưu điểm của CPS có thể thấy:

- Tăng độ chính xác: Nhãn giả từ mô hình thứ hai đóng vai trò giám sát chéo, giúp giảm hiện tượng "self-confirmation bias"(thiên lệch do tin vào nhãn giả của chính mình). Hiện tưởng self-confirmation bias là hiện tượng khi một mô hình được học các nhãn giả (pseudo labels) mà chính nó tạo ra nhưng nếu pseudo labels được tạo ra không đúng sẽ dẫn đến việc củng cố và lặp lại các lỗi sai.
- Dễ triển khai: Không cần mô hình phức tạp hơn như teacher-student, chỉ cần hai mô hình giống nhau.
- Khả năng tổng quát tốt: Phương pháp này có thể áp dụng cho nhiều kiến trúc mạng khác nhau như DeepLabV3+, PSPNet, U-Net...

Một số hạn chế của CPS:

- Tốn tài nguyên: Cần gấp đôi tài nguyên tính toán do sử dụng hai mô hình.
- Chất lượng nhãn giả phụ thuộc vào hiệu suất ban đầu: Nếu cả hai mô hình khởi đầu không tốt, nhãn giả sẽ không chính xác.

III. Cài đặt chương trình - CPS



Hình 4: Sơ đồ minh họa quy trình huấn luyện trong học bán giám sát với phương pháp CPS.

Sơ đồ hình 4 mô tả quy trình huấn luyện trong học bán giám sát với phương pháp Cross Pseudo Supervision (CPS), bao gồm các bước chính như sau:

1. Chuẩn bị dữ liệu (Data Preparation):
   - Dữ liệu huấn luyện được chia thành ba tập: dữ liệu gán nhãn, dữ liệu chưa gán nhãn và dữ liệu kiểm định.
   - Dữ liệu chưa gán nhãn được áp dụng hai kiểu tăng cường: weak augmentation và strong augmentation để phục vụ huấn luyện với CPS.
2. Khởi tạo (Initialization): Thiết lập các tham số quan trọng như hyperparameter, mô hình, bộ tối ưu và các hàm mất mát.
3. Huấn luyện (Training): Mô hình được huấn luyện với hai loại hàm mất mát: supervised loss trên dữ liệu gán nhãn và unsupervised loss từ Cross Pseudo Supervision (CPS) trên dữ liệu không có nhãn, cùng với các kỹ thuật optimization.
4. Đánh giá (Evaluation): Sau khi huấn luyện, mô hình được đánh giá trên tập kiểm thử để chọn ra mô hình tốt nhất.

A. Thư viện cần thiết

Cài đặt các thư viện cần thiết. Bao gồm các bộ thư viện của torch, numpy, random, itertools và copy, phục vụ cho việc tính toán và load dữ liệu.

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader, Subset
import torch.backends.cudnn as cudnn
import numpy as np
import random
from itertools import cycle
from copy import deepcopy
```

Cài đặt mặc định các tham số ngẫu nhiên. Đoạn code sẽ đảm bảo tính reproducibility qua mỗi lần thực nghiệm, bao gồm thiết lập cứng tham số ngẫu nhiên của các thư viện sử dụng cơ chế nhẫu nhiên.

```python
seed = 2024

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

B. MNIST

1. Thiết kế dataloader

Chia số lượng cho labeled, validated, unlabeled set. Mục dích để chia tập dữ liệu thành ba phần: có nhãn (500 mẫu), kiểm định (1000 mẫu) và chưa gán nhãn. Chỉ số dữ liệu được xáo trộn ngẫu nhiên trước khi phân chia, giúp tổ chức dữ liệu hiệu quả cho huấn luyện, đánh giá và các phương pháp học bán giám sát. Bao gồm:

1. Khởi tạo số lượng gắn cho labeled samples và validation samples thông qua các biến *labeled_num* và *val_num*.

2. Lấy index của toàn bộ samples của training dataset, biến *dataset* được định nghĩa ở bước tiếp theo, đồng thời xáo trộn (shuffle) các index một cách ngẫu nhiên.

3. Lấy các index cho từng loại samples bao gồm labeled samples (*labeled_idx*), validation samples (*validation_idx*) và unlabeled samples (*unlabeled_idx*).

```python
# Set the number of labeled, validated
labeled_num = 500
val_num = 1000

indices = np.arange(len(dataset))
np.random.shuffle(indices)

```

```
# The number of labeled samples
labeled_idx = indices[:labeled_num]
# The number of validated samples
validation_idx = indices[labeled_num:labeled_num + val_num]
# The number of validated samples
unlabeled_idx = indices[labeled_num + val_num:]
```

Load tập dữ liệu labeled và validated. Thực hiện việc tải tập dữ liệu MNIST, tạo tập có nhãn và tập kiểm định dựa trên các chỉ số đã xác định. Dữ liệu được chuyển thành tensor trước khi chia thành các tập con. Các bộ tải dữ liệu (DataLoader) được thiết lập với batch size 512, trong đó tập có nhãn được trộn ngẫu nhiên, còn tập kiểm định giữ nguyên thứ tự. Bao gồm:

1. Khởi tạo phương pháp biến đổi dữ liệu (augmentation) bằng cách sử dụng hàm *transforms.Compose* với phép biến đổi cơ bản là *transforms.ToTensor()*, giúp chuẩn hóa dữ liệu hình ảnh thành dạng tensor phục vụ huấn luyện mô hình.

2. Tải tập dữ liệu MNIST bằng *datasets.MNIST*, chỉ định đường dẫn lưu trữ, trạng thái huấn luyện (train=True), cho phép tải về nếu chưa tồn tại (download=True) và áp dụng phép biến đổi đã khởi tạo thông qua tham số *transform*.

3. Tạo tập dữ liệu có nhãn (*labeled_set*) bằng cách sử dụng lớp *Subset* với tập dữ liệu gốc và các chỉ số được xác định trước (*labeled_idx*).

4. Tạo tập kiểm định (*validation_set*) tương tự bằng *Subset* với các chỉ số tương ứng (*validation_idx*).

5. Thiết lập bộ tải dữ liệu cho tập có nhãn (*labeled_loader*) với batch size 512 và trộn ngẫu nhiên dữ liệu (shuffle=True) để tăng tính đa dạng trong quá trình huấn luyện.

6. Thiết lập bộ tải dữ liệu cho tập kiểm định (*validation_loader*) cũng với batch size 512 nhưng giữ nguyên thứ tự dữ liệu (shuffle=False) để đảm bảo tính nhất quán khi đánh giá mô hình.

```
# Base Augmentation
transform = transforms.Compose([transforms.ToTensor()])
# Load MNIST dataset
dataset = datasets.MNIST(root="./data", train=True, download=True, transform=
    transform)

# Get labeled set by predefined indices
labeled_set = Subset(dataset, labeled_idx)
# Get validated set by predefined indices
validation_set = Subset(dataset, validation_idx)

# Set up loader for labeled set
labeled_loader = DataLoader(labeled_set, batch_size=512, shuffle=True)
# Set up loader for validated set
validation_loader = DataLoader(validation_set, batch_size=512, shuffle=False)
```

Load tập dữ liệu unlabeled. Mục đích để load tập dữ liệu chưa gán nhãn (unlabeled set) và áp dụng hai dạng biến đổi dữ liệu: biến đổi yếu (weak augmentation) và biến đổi mạnh (strong augmentation). Biến đổi yếu gồm lật ngang, cắt ngẫu nhiên và chuyển đổi thành tensor, trong khi biến đổi mạnh bổ sung phép xoay ngẫu nhiên. Hai tập dữ liệu MNIST được tải với hai dạng

biến đổi này, sau đó tạo thành tập con dựa trên các chỉ số của tập chưa gán nhãn. Một lớp PairedDataset được định nghĩa để ghép cặp ảnh từ hai tập biến đổi, giúp tạo tập dữ liệu có cả phiên bản yếu và mạnh của cùng một mẫu. Cuối cùng, DataLoader được thiết lập để nạp dữ liệu với batch size 512 và trộn ngẫu nhiên. Bao gồm:

1. Định nghĩa biến đổi yếu (*weak_transform*) bằng cách sử dụng *transforms.Compose* kết hợp các phép biến đổi: lật ngang ngẫu nhiên (*RandomHorizontalFlip*), cắt ngẫu nhiên với phần đệm (*RandomCrop*), và chuyển đổi thành tensor (*ToTensor*).

2. Định nghĩa biến đổi mạnh (*strong_transform*) với các phép biến đổi tương tự như biến đổi yếu, nhưng bổ sung thêm phép xoay ngẫu nhiên (*RandomRotation*) để tăng cường độ đa dạng của dữ liệu.

3. Tải tập dữ liệu MNIST hai lần với hai dạng biến đổi riêng biệt, tạo thành tập *weak transformed dataset* và *strong transformed dataset*, đảm bảo mỗi tập áp dụng đúng phương pháp augmentation đã định nghĩa.

4. Tạo tập con chưa gán nhãn với hai phiên bản yếu và mạnh bằng cách sử dụng lớp *Subset* với chỉ số *unlabeled_idx* để lấy mẫu dữ liệu từ hai tập biến đổi.

5. Định nghĩa lớp *PairedDataset* để ghép cặp dữ liệu từ tập biến đổi yếu và mạnh. Phương thức *getitem* trả về hai phiên bản của cùng một mẫu dữ liệu, giúp mô hình học được tính bất biến qua các biến đổi khác nhau.

6. Thiết lập bộ tải dữ liệu (*unlabeled_loader*) với batch size 512 và trộn ngẫu nhiên (shuffle=True), đảm bảo quá trình huấn luyện đa dạng và không bị lệ thuộc vào thứ tự ban đầu của dữ liệu.

```
# Weak-Strong augmentation
weak_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(28, padding=4),
    transforms.ToTensor()
])

strong_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(28, padding=4),
    transforms.RandomRotation(15),
    transforms.ToTensor()
])
```

```
# Load dataset with weak-strong augmentation
weak_transformed_dataset = datasets.MNIST(root="./data", train=True, download=
    True, transform=weak_transform)
strong_transformed_dataset = datasets.MNIST(root="./data", train=True,
    download=True, transform=strong_transform)
```

```
class PairedDataset(torch.utils.data.Dataset):
    def __init__(self, weak_set, strong_set):
        self.weak_set = weak_set
        self.strong_set = strong_set
```

```

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

    def __getitem__(self, index):
        weak_img, weak_lab = self.weak_set[index]
        strong_img, strong_lab = self.strong_set[index]
        return (weak_img, weak_lab), (strong_img, strong_lab)
```

```
# Get weak, strong augmented set by predefined indices
weak_unlabeled_set = Subset(weak_transformed_dataset, unlabeled_idx)
strong_unlabeled_set = Subset(strong_transformed_dataset, unlabeled_idx)

# Set up loader for unlabeled data
paired_unlabeled_set = PairedDataset(weak_unlabeled_set, strong_unlabeled_set)
unlabeled_loader = DataLoader(paired_unlabeled_set, batch_size=512, shuffle=
    True)
```

C. CIFAR-10

Tương tự như MNIST, tập dữ liệu CIFAR-10 cũng có thể được chia thành các tập có nhãn, kiểm định và chưa gán nhãn. Các kỹ thuật tăng cường dữ liệu yếu và mạnh được áp dụng để tạo ra hai phiên bản khác nhau của cùng một hình ảnh, giúp mô hình học tốt hơn trong các phương pháp học bán giám sát. Việc sử dụng lớp PairedDataset để ghép cặp ảnh từ hai phiên bản biến đổi cũng được áp dụng tương tự, cho phép nạp dữ liệu hiệu quả bằng DataLoader. Điều này giúp cải thiện khả năng tổng quát hóa của mô hình khi làm việc với dữ liệu hình ảnh phức tạp hơn như CIFAR-10.

1. Thiết kế dataloader

Chia số lượng cho labeled, validated, unlabeled set.

```
# Set the number of labeled, validated
labeled_num = 2000
val_num = 1000

indices = np.arange(len(dataset))
np.random.shuffle(indices)

# The number of labeled samples
labeled_idx = indices[:labeled_num]
# The number of validated samples
validation_idx = indices[labeled_num:labeled_num + val_num]
# The number of validated samples
unlabeled_idx = indices[labeled_num + val_num:]
```

Load tập dữ liệu labeled và validated

```
# Base Augmentation
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize
    ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
# Load CIFAR10 dataset
```

```
dataset = datasets.CIFAR10(root="./data", train=True, download=True, transform=transform)

# Get labeled set by predefined indices
labeled_set = Subset(dataset, labeled_idx)
# Get validated set by predefined indices
validation_set = Subset(dataset, validation_idx)

# Set up loader for labeled set
labeled_loader = DataLoader(labeled_set, batch_size=512, shuffle=True)
# Set up loader for labeled set
labeled_loader = DataLoader(labeled_set, batch_size=512, shuffle=True)
# Set up loader for validated set
validation_loader = DataLoader(validation_set, batch_size=512, shuffle=False)
```

Load tập dữ liệu unlabeled

```
# Weak-Strong augmentation
weak_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

strong_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

```
# Load dataset with weak-strong augmentation
weak_transformed_dataset = datasets.CIFAR10(root="./data", train=True, download=True, transform=weak_transform)
strong_transformed_dataset = datasets.CIFAR10(root="./data", train=True, download=True, transform=strong_transform)
```

```
class PairedDataset(torch.utils.data.Dataset):
    def __init__(self, weak_set, strong_set):
        self.weak_set = weak_set
        self.strong_set = strong_set

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

    def __getitem__(self, index):
        weak_img, weak_lab = self.weak_set[index]
        strong_img, strong_lab = self.strong_set[index]
        return (weak_img, weak_lab), (strong_img, strong_lab)
```

```
# Get weak, strong augmented set by predefined indices
weak_unlabeled_set = Subset(weak_transformed_dataset, unlabeled_idx)
strong_unlabeled_set = Subset(strong_transformed_dataset, unlabeled_idx)
```

```
4
5 # Set up loader for unlabeled data
6 paired_unlabeled_set = PairedDataset(weak_unlabeled_set, strong_unlabeled_set)
7 unlabeled_loader = DataLoader(paired_unlabeled_set, batch_size=512, shuffle=
    True)
```

D. Thiết kế mô hình

Để đơn giản hóa kiến trúc mô hình, một CNN đơn giản được thiết kế để tập trung vào ứng dụng của học bán giám sát (semi-supervised learning). Hai mô hình CNN được thiết kế cho tập dữ liệu MNIST và CIFAR-10 có cấu trúc tương tự nhau, nhưng có một số điều chỉnh phù hợp với đặc điểm của từng tập dữ liệu.

Với MNIST, mô hình xử lý ảnh xám (1 kênh), sử dụng hai lớp tích chập (Conv2d) để trích xuất đặc trưng, theo sau là các lớp kích hoạt ReLU và pooling (max_pool2d). Đầu ra sau khi flatten được đưa vào hai lớp kết nối đầy đủ (Linear) để phân loại 10 chữ số. Với CIFAR-10, kiến trúc tương tự nhưng đầu vào là ảnh màu (3 kênh), và kích thước dữ liệu sau mỗi lớp pooling được điều chỉnh cho phù hợp với độ phân giải ảnh lớn hơn. Chi tiết bao gồm:

- Khởi tạo lớp *CNN* kế thừa từ *nn.Module*, định nghĩa kiến trúc của mạng trong hàm *init* và triển khai quá trình lan truyền xuôi trong hàm *forward(x)*.
- Với MNIST, mô hình xử lý ảnh xám (1 kênh), sử dụng hai lớp tích chập (*Conv2d*) với 32 và 64 kênh đầu ra, kernel size 3x3, stride 1 và padding 1. Kết hợp các lớp kích hoạt *ReLU* và pooling (*max_pool2d*) để giảm kích thước đầu ra.
- Sau hai lớp tích chập và pooling, đầu ra được làm phẳng (*view*) và đưa qua hai lớp kết nối đầy đủ (*Linear*). Lớp đầu tiên ánh xạ đặc trưng vào không gian 128 chiều, và lớp cuối cùng dự đoán xác suất cho 10 lớp (các chữ số).
- Với CIFAR-10, mô hình tương tự nhưng xử lý ảnh màu (3 kênh). Do độ phân giải ảnh cao hơn (32x32), kích thước đầu ra sau pooling thay đổi, nên số chiều đầu vào của lớp kết nối đầy đủ đầu tiên được điều chỉnh thành 64×8×8.
- Hàm *forward* triển khai quá trình truyền xuôi: dữ liệu đi qua các lớp tích chập, kích hoạt ReLU, pooling, làm phẳng và cuối cùng là các lớp kết nối đầy đủ. Đầu ra là logits chưa chuẩn hóa cho 10 lớp.

1. MNIST

```
1 class CNN(nn.Module):
2     def __init__(self):
3         super(CNN, self).__init__()
4         self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
5         self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
6         self.fc1 = nn.Linear(64 * 7 * 7, 128)
7         self.fc2 = nn.Linear(128, 10)
8
9     def forward(self, x):
10        x = F.relu(self.conv1(x))
```

```
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

2. CIFAR-10

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(64 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

E. Thiết lập luồng huấn luyện

1. Training

Khởi tạo. Khởi tạo các thành phần quan trọng cho quá trình huấn luyện mô hình. Cụ thể, mô hình CNN được khởi tạo và đưa vào thiết bị (device), sử dụng Adam làm trình tối ưu với learning rate 0.001, và CrossEntropyLoss làm hàm mất mát. Bao gồm:

1. Khởi tạo hai mô hình CNN riêng biệt, *model1* và *model2*, đưa vào thiết bị bằng phương thức *to(device)*.

2. Thiết lập hai trình tối ưu *optimizer1* và *optimizer2*, mỗi trình tối ưu quản lý tham số của một mô hình bằng phương thức *model.parameters()*, sử dụng Adam với learning rate 0.001.

3. Khai báo hàm mất mát *criterion* là CrossEntropyLoss, phù hợp cho các bài toán phân loại đa lớp.

4. Xác định các tham số kiểm soát vòng lặp huấn luyện: *iter_num* là số vòng lặp hiện tại, *max_iter_num* là tổng số vòng lặp huấn luyện.

5. Thiết lập ngưỡng *threshold* bằng 0.95, dùng để chọn ra các dự đoán có độ tin cậy cao trong quá trình huấn luyện bán giám sát.

6. Khởi tạo biến lưu trữ mô hình tốt nhất (*best_model*) và độ chính xác cao nhất (*best_acc*), cùng với *iter_per_eval* xác định số vòng lặp giữa các lần đánh giá mô hình.

```
# Init model, optimizer, loss
model1 = CNN().to(device)
model2 = CNN().to(device)
optimizer1 = optim.Adam(model1.parameters(), lr=0.001)
optimizer2 = optim.Adam(model2.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

# Number of iterations
iter_num = 0
max_iter_num = 200

# Threshold (Threshold's hyperparameter)
threshold = 0.95

# Variables for storing best model
iter_per_eval = 5
best_model = None
best_acc = 0
```

Thiết lập hàm validate. Hàm validate(model) được thiết lập để đánh giá độ chính xác của mô hình trên tập kiểm định (validation set). Bao gồm:

1. Chuyển mô hình sang chế độ đánh giá (*eval mode*) bằng cách gọi *model.eval()*, tắt các thành phần như dropout hoặc batch normalization, giúp mô hình hoạt động ổn định khi dự đoán.

2. Khởi tạo biến *correct* để đếm số lượng dự đoán đúng trên tập kiểm định.

3. Tắt việc tính gradient bằng *torch.no_grad()* nhằm tiết kiệm bộ nhớ và tăng tốc quá trình suy luận.

4. Duyệt qua từng batch dữ liệu từ *validation_loader*, chuyển dữ liệu và nhãn sang thiết bị GPU (*cuda()*) nếu khả dụng.

5. Thực hiện dự đoán bằng cách truyền dữ liệu qua mô hình và lấy chỉ số lớp có xác suất cao nhất bằng *argmax(dim=1)*.

6. So sánh dự đoán với nhãn thật, tăng biến *correct* với số lượng dự đoán đúng qua mỗi batch bằng *sum().item()*.

7. Tính toán độ chính xác bằng cách chia số lượng dự đoán đúng cho tổng số mẫu trong tập kiểm định, lưu vào biến *acc_score*.

8. In ra độ chính xác với định dạng 4 chữ số thập phân để dễ quan sát kết quả.

9. Chuyển mô hình trở lại chế độ huấn luyện (*train mode*) bằng *model.train()* để tiếp tục cập nhật tham số trong các bước huấn luyện tiếp theo.

10. Trả về *acc_score* như đầu ra của hàm, phục vụ việc theo dõi và đánh giá hiệu năng mô hình trong quá trình huấn luyện.

```python
def validate(model):
    # Move to eval model
    model.eval()
    correct = 0
    with torch.no_grad():
        for data, target in validation_loader:
            data, target = data.cuda(), target.cuda()
            output = model(data)
            pred = output.argmax(dim=1)
            correct += pred.eq(target).sum().item()

    acc_score = correct / len(validation_set)
    print(f"Validation Accuracy: {acc_score:.4f}")
    # Turn back to training model
    model.train()
    return acc_score
```

Thiết lập luồng training. Thiết lập luồng huấn luyện theo phương pháp Cross Pseudo Supervision (CPS) với hai mô hình đồng huấn luyện (co-training) nhằm khai thác dữ liệu chưa gán nhãn trong học bán giám sát. Quy trình huấn luyện như sau:

1. Huấn luyện trên dữ liệu có nhãn (Supervised Loss):
   - Cả hai mô hình model1 và model2 dự đoán nhãn từ dữ liệu có nhãn (logits_x1, logits_x2).
   - Tính toán hàm mất mát giám sát (loss_x1, loss_x2) bằng CrossEntropyLoss.
2. Tạo nhãn giả và huấn luyện chéo trên dữ liệu chưa gán nhãn (Cross Pseudo Supervision):
   - Tạo nhãn giả (Pseudo Labels):
     - Mô hình model1 dự đoán trên ảnh biến đổi yếu (logits_u_w1, probs_u_w1), model2 làm tương tự (logits_u_w2, probs_u_w2).
     - Mỗi mô hình sử dụng dự đoán của mô hình còn lại làm nhãn giả:
       * pseudo_labels1 (cho model1) lấy từ dự đoán của model2.
       * pseudo_labels2 (cho model2) lấy từ dự đoán của model1.
     - Chỉ giữ lại các nhãn giả có độ tin cậy cao hơn threshold (tạo mặt nạ mask1, mask2).
   - Huấn luyện trên ảnh biến đổi mạnh (Unsupervised Loss):
     - Hai mô hình dự đoán trên ảnh biến đổi mạnh (logits_u_s1, logits_u_s2).
     - Tính hàm mất mát không giám sát (loss_u1, loss_u2), chỉ áp dụng cho các mẫu đạt ngưỡng tin cậy.
3. Tối ưu hóa mô hình:
   - Tổng hợp mất mát cho từng mô hình"
     - loss1 = loss_x1 + loss_u1 (cho model1).
     - loss2 = loss_x2 + loss_u2 (cho model2).
   - Cập nhật trọng số từng mô hình riêng biệt bằng optimizer1 và optimizer2.

4. Đánh giá mô hình:

   - Sau mỗi iter_per_eval vòng lặp, mô hình được đánh giá trên tập kiểm định (validate(model1)).
   - Nếu độ chính xác cao hơn kết quả tốt nhất trước đó, lưu lại model1 làm mô hình tốt nhất (best_model).

```
# Train model
model1.train()
model2.train()
for _ in range(100):
    for (labeled_data, labeled_target), ((weak_unlabeled_data, _), (strong_unlabeled_data, _)) in zip(cycle(labeled_loader), unlabeled_loader):
        labeled_data, labeled_target = labeled_data.to(device), labeled_target.to(device)
        weak_unlabeled_data, strong_unlabeled_data = weak_unlabeled_data.to(device), strong_unlabeled_data.to(device)

        # Supervised loss for both models
        logits_x1 = model1(labeled_data)
        logits_x2 = model2(labeled_data)
        loss_x1 = criterion(logits_x1, labeled_target)
        loss_x2 = criterion(logits_x2, labeled_target)

        # Unsupervised loss with cross-pseudo labels
        logits_u_w1 = model1(weak_unlabeled_data)
        logits_u_w2 = model2(weak_unlabeled_data)

        probs_u_w1 = torch.softmax(logits_u_w1, dim=1)
        probs_u_w2 = torch.softmax(logits_u_w2, dim=1)

        max_probs1, pseudo_labels1 = torch.max(probs_u_w2, dim=1)  # Use model2's predictions for model1
        max_probs2, pseudo_labels2 = torch.max(probs_u_w1, dim=1)  # Use model1's predictions for model2

        mask1 = max_probs1.ge(threshold).float()
        mask2 = max_probs2.ge(threshold).float()

        logits_u_s1 = model1(strong_unlabeled_data)
        logits_u_s2 = model2(strong_unlabeled_data)

        loss_u1 = (criterion(logits_u_s1, pseudo_labels1) * mask1).mean()
        loss_u2 = (criterion(logits_u_s2, pseudo_labels2) * mask2).mean()

        loss1 = loss_x1 + loss_u1
        loss2 = loss_x2 + loss_u2

        optimizer1.zero_grad()
        loss1.backward()
        optimizer1.step()

        optimizer2.zero_grad()
        loss2.backward()
```

```
        optimizer2.step()

        iter_num += 1
        print(f"Iteration {iter_num}, Model1 Loss: {loss1.item():.4f}, Model2
    Loss: {loss2.item():.4f}")

        if iter_num % 5 == 0:
            acc_score = validate(model1)
            if acc_score >= best_acc:
                best_acc = acc_score
                best_model = deepcopy(model1)
                print(f'Save new best model at iter: {iter_num}')

        if iter_num == max_iter_num:
            break
     if iter_num == max_iter_num:
            break
```

2. Testing

Sau khi quá trình huấn luyện hoàn tất, mô hình được kiểm tra trên tập kiểm tra để đánh giá khả năng tổng quát hóa thông qua hàm test(model). Hàm này được áp dụng cho cả hai tập dữ liệu MNIST và CIFAR-10, giúp đánh giá hiệu suất của mô hình trên các bài toán phân loại khác nhau. Độ chính xác thu được phản ánh mức độ hiệu quả của phương pháp học bán giám sát, cho thấy mô hình có thể tận dụng dữ liệu chưa gán nhãn để cải thiện kết quả so với chỉ sử dụng dữ liệu có nhãn. Bao gồm:

1. Chuyển mô hình sang chế độ đánh giá (*eval mode*) bằng cách gọi *model.eval()*, giúp vô hiệu hóa các thành phần như dropout hoặc batch normalization, đảm bảo tính ổn định khi suy luận.

2. Tải tập kiểm tra MNIST hoặc CIFAR-10 thông qua lớp *MNIST()* hoặc *CIFAR10()*, sử dụng tham số *train=False* để lấy tập kiểm tra và áp dụng biến đổi đã định nghĩa (*transform*).

3. Tạo DataLoader cho tập kiểm tra với batch size 512 và không trộn dữ liệu (*shuffle=False*) để giữ nguyên thứ tự mẫu.

4. Khởi tạo biến *correct* để đếm số lượng dự đoán đúng trên tập kiểm tra.

5. Tắt việc tính gradient với *torch.no_grad()* nhằm tiết kiệm bộ nhớ và tăng tốc suy luận.

6. Duyệt qua từng batch dữ liệu từ *test_loader*, chuyển dữ liệu và nhãn sang thiết bị huấn luyện (*device*).

7. Truyền dữ liệu qua mô hình để tạo dự đoán, lấy chỉ số lớp có xác suất cao nhất bằng *argmax(dim=1)*.

8. So sánh dự đoán với nhãn thật và cộng số lượng dự đoán đúng vào biến *correct* qua mỗi batch bằng *sum().item()*.

9. Tính toán và in ra độ chính xác bằng cách chia số lượng dự đoán đúng cho tổng số mẫu trong tập kiểm tra, định dạng với 4 chữ số thập phân.

10. Thực hiện hàm *test(best_ model)* để đánh giá mô hình tốt nhất thu được trong quá trình huấn luyện trên từng tập dữ liệu.

MNIST

```
# Evaluate model
def test(model):
    model.eval()
    test_dataset = datasets.MNIST(root="./data", train=False, transform=
    transform)
    test_loader = DataLoader(test_dataset, batch_size=512, shuffle=False)
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            pred = output.argmax(dim=1)
            correct += pred.eq(target).sum().item()
    print(f"Test Accuracy: {correct / len(test_dataset):.4f}")

test(best_model)
```

CIFAR-10

```
# Evaluate model
def test(model):
    model.eval()
    test_dataset = datasets.CIFAR10(root="./data", train=False, download=True,
    transform=transform)
    test_loader = DataLoader(test_dataset, batch_size=512, shuffle=False)
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            pred = output.argmax(dim=1)
            correct += pred.eq(target).sum().item()
    print(f"Test Accuracy: {correct / len(test_dataset):.4f}")

test(best_model)
```

IV. Đánh giá kết quả

Kết quả đánh giá cho thấy phương pháp Cross Pseudo Supervision (CPS) giúp cải thiện độ chính xác so với mô hình chỉ sử dụng dữ liệu có nhãn (Sup only), khẳng định hiệu quả của cơ chế học bán giám sát trong việc khai thác thông tin từ dữ liệu chưa gán nhãn.

Trên tập MNIST (Bảng 1), phương pháp CPS đạt độ chính xác vượt trội so với Sup only, ngay cả khi số lượng dữ liệu có nhãn hạn chế. Điều này cho thấy cơ chế Cross Pseudo Labels, trong đó hai mô hình trao đổi nhãn giả để học lẫn nhau, giúp tăng khả năng khái quát của mô hình.

Đối với tập CIFAR-10 (Bảng 2), mặc dù mức cải thiện không quá lớn, phương pháp CPS vẫn mang lại hiệu quả rõ ràng, đặc biệt khi tăng số lượng dữ liệu có nhãn. Kết quả này nhấn mạnh khả năng của CPS trong việc xử lý các tập dữ liệu phức tạp hơn.

Nhìn chung, kết quả thực nghiệm khẳng định rằng CPS là một phương pháp hiệu quả trong học bán giám sát, giúp tận dụng dữ liệu chưa gán nhãn để cải thiện hiệu suất mô hình, đặc biệt khi dữ liệu có nhãn khan hiếm.

	500 labeled samples	1000 labeled samples
Sup only	91.44	93.21
CPS	94.34	95.63

Bảng 1: Bảng kết quả tham khảo trên dữ liệu MNIST

	1000 labeled samples	2000 labeled samples
Sup only	42.73	47.7
CPS	43.07	48.49

Bảng 2: Bảng kết quả tham khảo trên dữ liệu CIFAR-10

V. Hướng nghiên cứu trong tương lai

Co-training truyền thống. Các phương pháp tập trung vô việc khai thác hiệu quả thông tin dưới nhiều góc nhìn (view) khác nhau. Ví dụ như: Bằng cách kết hợp hiệu quả giữa co-training với uncertainty modeling và dynamic data augmentation, UCC [7] giải quyết các thách thức phổ biến như class imbalance và noisy trong pseudo-labels, từ đó có thể cải thiện hiệu suất của model. CCVC [8] đã đề xuất Cross-View Consistency (CVC) và Conflict-Based Pseudo-Labelling (CPL) nhằm đảm bảo được sự ổn định và đa dạng khi mô hình học trên hai góc nhìn toàn toàn khác nhau.

Multi-task learning. Việc sử dụng co-training trên một nhiệm vụ cố định sẽ giới hạn khả năng học cũng như mức độ đa dạng mà các phương pháp co-training có thể mang lại. Do đó, việc kết hợp giữa multi-task learning và co-training có thể giúp cho model khai thác được đặc trưng và tổng hợp thông tin từ nhiều task khác nhau, từ đó có thể tạo ra nhãn giả tốt hơn và cải thiện quá trình học cũng như mức độ khái quát (generalization) của model. Một số phương pháp có thể tham khảo như: [9], SemiMTL [10], [11],...

VI. Code tham khảo

- Link chứa code thực nghiệm của SupOnly tại đây: MNIST và CIFAR10.
- Link chứa code thực nghiệm của CPS tại đây MNIST và CIFAR10.

Tài liệu

[1] *Semi-Supervised Machine Learning*, https://blog.techiescamp.com/docs/semi-supervised-machine-learning/.

[2] A. Blum **and** T. Mitchell, "Combining labeled and unlabeled data with co-training," **in***Proceedings of the Eleventh Annual Conference on Computational Learning Theory* **jourser** COLT' 98, Madison, Wisconsin, USA: Association for Computing Machinery, 1998, 92–100, ISBN: 1581130570. DOI: 10.1145/279943.279962.

[3] K. Nigam **and** R. Ghani, "Analyzing the effectiveness and applicability of co-training," **in***Proceedings of the Ninth International Conference on Information and Knowledge Management* **jourser** CIKM '00, McLean, Virginia, USA: Association for Computing Machinery, 2000, 86–93, ISBN: 1581133200. DOI: 10.1145/354756.354805.

[4] Z.-H. Zhou **and** M. Li, "Tri-training: exploiting unlabeled data using three classifiers," *IEEE Transactions on Knowledge and Data Engineering*, **jourvol** 17, **number** 11, **pages** 1529–1541, 2005. DOI: 10.1109/TKDE.2005.186.

[5] C. Xu, D. Tao **and** C. Xu, *A Survey on Multi-view Learning*, 2013. arXiv: 1304.5634 [cs.LG].

[6] X. Chen, Y. Yuan, G. Zeng **and** J. Wang, "Semi-Supervised Semantic Segmentation with Cross Pseudo Supervision," **in***2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2021, **pages** 2613–2622. DOI: 10.1109/CVPR46437.2021.00264.

[7] J. Fan, B. Gao, H. Jin **and** L. Jiang, "UCC: Uncertainty guided Cross-head Cotraining for Semi-Supervised Semantic Segmentation," **in***2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2022, **pages** 9937–9946. DOI: 10.1109/CVPR52688.2022.00971.

[8] Z. Wang, Z. Zhao, X. Xing, D. Xu, X. Kong **and** L. Zhou, "Conflict-Based Cross-View Consistency for Semi-Supervised Semantic Segmentation," **in***2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2023, **pages** 19 585–19 595. DOI: 10.1109/CVPR52729.2023.01876.

[9] Q. Liu, X. Liao **and** L. Carin, "Semi-Supervised Multitask Learning," **in***Advances in Neural Information Processing Systems* J. Platt, D. Koller, Y. Singer **and** S. Roweis, **editors**, **volume** 20, Curran Associates, Inc., 2007.

[10] Y. Wang, Y.-H. Tsai, W.-C. Hung, W. Ding, S. Liu **and** M.-H. Yang, "Semi-supervised Multi-task Learning for Semantics and Depth," **in***2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* 2022, **pages** 2663–2672. DOI: 10.1109/WACV51458.2022.00272.

[11] F. Spinola, P. Benz, M. Yu **and** T. hoon Kim, *Knowledge Assembly: Semi-Supervised Multi-Task Learning from Multiple Datasets with Disjoint Labels*, 2023. arXiv: 2306.08839 [cs.CV]. **url**: https://arxiv.org/abs/2306.08839.