Homework 3 - Convolutional Neural Network

This is the example code of homework 3 of the machine learning course by Prof. Hung-yi Lee.

In this homework, you are required to build a convolutional neural network for image classification, possibly with some advanced training tips.

There are three levels here:

Easy: Build a simple convolutional neural network as the baseline. (2 pts)

Medium: Design a better architecture or adopt different data augmentations to improve the performance. (2 pts)

Hard: Utilize provided unlabeled data to obtain better results. (2 pts)

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
%cd /content/drive/MyDrive/NTU_colab/hw3
```

```
/content/drive/MyDrive/NTU_colab/hw3
```

```
import gc
import torch
gc.collect()
torch.cuda.empty_cache()
```

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('Not connected to a GPU')
else:
    print(gpu_info)
'''
import urllib.request

url = 'https://download.pytorch.org/models/resnet34-333f7ec4.pth'
urllib.request.urlretrieve(url, 'resnet34-333f7ec4.pth')
'''
```

```
Sat Feb 17 07:22:04 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla V100-SXM2-16GB           Off | 00000000:00:04.0 Off |                    0 |
| N/A   34C    P0              41W / 300W |    502MiB / 16384MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
+---------------------------------------------------------------------------------------+
'\nimport urllib.request\n\nurl = 'https://download.pytorch.org/models/resnet34-333f7ec4.pth'\nurllib.request.urlretri
eve(url, 'resnet34-333f7ec4.pth')\n'
```

About the Dataset

The dataset used here is food-11, a collection of food images in 11 classes.

For the requirement in the homework, TAs slightly modified the data. Please DO NOT access the original fully-labeled training data or testing labels.

Also, the modified dataset is for this course only, and any further distribution or commercial use is forbidden.

```
#
!gdown --id '1awF7pZ9Dz7X1jn1_QAiKN-_v56veCEKy' --output food-11.zip
!unzip -q food-11.zip
"""
# Download the dataset
# You may choose where to download the data.

# Google Drive
gdown --id '1awF7pZ9Dz7X1jn1_QAiKN-_v56veCEKy' --output food-11.zip

# Dropbox
# !wget https://www.dropbox.com/s/m9q6273jl3djall/food-11.zip -O food-11.zip

# MEGA
# !sudo apt install megatools
# !megadl "https://mega.nz/#!zt1TTIhK!ZuMbg5ZjGWzWX1I6nEUbfjMZgCmAgeqJlwDkqdIryfg"

# Unzip the dataset.
# This may take some time.
!unzip -q food-11.zip
"""
```

```
/usr/local/lib/python3.10/dist-packages/gdown/cli.py:138: FutureWarning: Option `--id` was deprecated in version 4.3.1
  warnings.warn(
Downloading...
From (original): https://drive.google.com/uc?id=1awF7pZ9Dz7X1jn1_QAiKN-_v56veCEKy
From (redirected): https://drive.google.com/uc?id=1awF7pZ9Dz7X1jn1_QAiKN-_v56veCEKy&confirm=t&uuid=957c51c9-3e84-4179-
To: /content/drive/MyDrive/NTU_colab/hw3/food-11.zip
100% 963M/963M [00:05<00:00, 178MB/s]
'\n# Download the dataset\n# You may choose where to download the data.\n\n# Google Drive\ngdown --id \'1awF7pZ9Dz7X1j
n1_QAiKN-_v56veCEKy\' --output food-11.zip\n\n# Dropbox\n# !wget https://www.dropbox.com/s/m9q6273jl3djall/food-11.zip
-O food-11.zip\n\n# MEGA\n# !sudo apt install megatools\n# !megadl "https://mega.nz/#!zt1TTIhK!ZuMbg5ZjGWzWX1I6nEUbfjM
ZgCmAgeqJlwDkqdIryfg"\n\n# Unzip the dataset.\n# This may take some time.\n!unzip -q food-11.zip\n'
```

Import Packages

1. 清單項目
2. 清單項目

First, we need to import packages that will be used later.

In this homework, we highly rely on **torchvision**, a library of PyTorch.

```
# Import necessary packages.
import numpy as np
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from PIL import Image
# "ConcatDataset" and "Subset" are possibly useful when doing semi-supervised learning.
from torch.utils.data import ConcatDataset, DataLoader, Subset, Dataset
from torchvision.datasets import DatasetFolder

# This is for the progress bar.
#from tqdm.auto import tqdm #may have some bug
from tqdm import tqdm
```

Change Dir to google drive

```
"""
from google.colab import drive
drive.mount('/content/drive')

import os
os.chdir('/content/drive/My Drive/Colab Notebooks/hw3_data') #切換該目錄
os.listdir() #確認目錄內容
"""
```

```
'\nfrom google.colab import drive\ndrive.mount('/content/drive')\n\nimport os\nos.chdir('/content/drive/My Drive/Colab Notebooks/hw3_data') #切換該目錄\nos.listdir() #確認目錄內容\n'
```

Dataset, Data Loader, and Transforms

Torchvision provides lots of useful utilities for image preprocessing, data wrapping as well as data augmentation.

Here, since our data are stored in folders by class labels, we can directly apply **torchvision.datasets.DatasetFolder** for wrapping data without much effort.

Please refer to [PyTorch official website] for details about different transforms.

```
# It is important to do data augmentation in training.
# However, not every augmentation is useful.
# Please think about what kind of augmentation is helpful for food recognition.
train_tfm = transforms.Compose([
        # Resize the image into a fixed shape (height = width = 128)
        transforms.Resize((128, 128)),
        # You may add some transforms here.
        # ToTensor() should be the last one of the transforms.
        transforms.ToTensor(),
        #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

train_tfm2 = transforms.Compose([
        # Resize the image into a fixed shape (height = width = 128)
        transforms.Resize((128, 128)),
        # ToTensor() should be the last one of the transforms.
        transforms.RandomHorizontalFlip(p=1.0),
        transforms.ToTensor(),
        #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

train_tfm3 = transforms.Compose([
        # Resize the image into a fixed shape (height = width = 128)
        transforms.Resize((128, 128)),
        # ToTensor() should be the last one of the transforms.
        transforms.RandomRotation((30,180)),
        transforms.ToTensor(),
        #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

train_tfm4 = transforms.Compose([
        transforms.Resize((128, 128)),
        transforms.RandomPerspective(p=1.0),
        transforms.ToTensor(),
        #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

train_tfm5 = transforms.Compose([
        # Resize the image into a fixed shape (height = width = 128)
        transforms.Resize((128, 128)),
        transforms.RandomHorizontalFlip(p=1.0),
        transforms.ToTensor(),
```

```
        #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

train_tfm6 = transforms.Compose([
        # Resize the image into a fixed shape (height = width = 128)
        transforms.Resize((128, 128)),
        transforms.ColorJitter(brightness=(0, 5), contrast=(
        0, 5), saturation=(0, 5), hue=(-0.1, 0.1)),
        transforms.ToTensor(),
        #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# We don't need augmentations in testing and validation.
# All we need here is to resize the PIL image and transform it into Tensor.
test_tfm = transforms.Compose([
        transforms.Resize((128, 128)),
        transforms.ToTensor(),
        #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

```python
# Batch size for training, validation, and testing.
# A greater batch size usually gives a more stable gradient.
# But the GPU memory is limited, so please adjust it carefully.
batch_size = 128

# Construct datasets.
# The argument "loader" tells how torchvision reads the data.
train_set1 = DatasetFolder("food-11/training/labeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm)
train_set2 = DatasetFolder("food-11/training/labeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm2)
train_set3 = DatasetFolder("food-11/training/labeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm3)
train_set4 = DatasetFolder("food-11/training/labeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm4)
train_set5 = DatasetFolder("food-11/training/labeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm5)
train_set6 = DatasetFolder("food-11/training/labeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm6)


train_set = ConcatDataset([train_set1, train_set2, train_set3, train_set4, train_set5, train_set6])
valid_set = DatasetFolder("food-11/validation", loader=lambda x: Image.open(x), extensions="jpg", transform=test_tfm)
#unlabeled_set = DatasetFolder("food-11/training/unlabeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm)

unlabeled_set1 = DatasetFolder("food-11/training/unlabeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm)
unlabeled_set2 = DatasetFolder("food-11/training/unlabeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm2)
unlabeled_set3 = DatasetFolder("food-11/training/unlabeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm3)
unlabeled_set4 = DatasetFolder("food-11/training/unlabeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm4)
unlabeled_set5 = DatasetFolder("food-11/training/unlabeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm5)
unlabeled_set6 = DatasetFolder("food-11/training/unlabeled", loader=lambda x: Image.open(x), extensions="jpg", transform=train_tfm6)
unlabeled_set = ConcatDataset([unlabeled_set1, unlabeled_set2, unlabeled_set3, unlabeled_set4, unlabeled_set5, unlabeled_set6])


test_set = DatasetFolder("food-11/testing", loader=lambda x: Image.open(x), extensions="jpg", transform=test_tfm)

# Construct data loaders.
# pin_memory = false, data will be swapped to driver if necessary
# default num_workers = 8
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=16, pin_memory=True)
valid_loader = DataLoader(valid_set, batch_size=batch_size, shuffle=True, num_workers=16, pin_memory=True)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 16 worker processes in tot
  warnings.warn(_create_warning_msg(
```

Model

The basic model here is simply a stack of convolutional layers followed by some fully-connected layers.

Since there are three channels for a color image (RGB), the input channels of the network must be three. In each convolutional layer, typically the channels of inputs grow, while the height and width shrink (or remain unchanged, according to some hyperparameters like stride and padding).

Before fed into fully-connected layers, the feature map must be flattened into a single one-dimensional vector (for each image). These features are then transformed by the fully-connected layers, and finally, we obtain the "logits" for each class.

WARNING -- You Must Know

You are free to modify the model architecture here for further improvement. However, if you want to use some well-known architectures such as ResNet50, please make sure **NOT** to load the pre-trained weights. Using such pre-trained models is considered cheating and therefore you will be punished. Similarly, it is your responsibility to make sure no pre-trained weights are used if you use **torch.hub** to load any modules.

For example, if you use ResNet-18 as your model:

model = torchvision.models.resnet18(pretrained=**False**) → This is fine.

model = torchvision.models.resnet18(pretrained=**True**) → This is **NOT** allowed.

```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        # The arguments for commonly used modules:
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        # out_channels = feature filter num, by adding 3 RGB input channel result within 1 filter
        # torch.nn.MaxPool2d(kernel_size, stride, padding)
        # after 2x2 kernel maxpool, result will be intput witdth/2 * height/2

        # input image size: [3, 128, 128]
        """" original
        self.cnn_layers = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),

            nn.Conv2d(64, 128, 3, 1, 1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),

            nn.Conv2d(128, 256, 3, 1, 1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(4, 4, 0),
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(256 * 8 * 8, 256),
            nn.ReLU(),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.BatchNorm1d(256),
            nn.Dropout(p=0.5),
            nn.Linear(256, 11)
        )
        """
        # add 2 Conv layers
        self.cnn_layers = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
```

```
            nn.MaxPool2d(2, 2, 0),

            nn.Conv2d(64, 128, 3, 1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),

            nn.Conv2d(128, 256, 3, 1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),

            nn.Conv2d(256, 512, 3, 1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),

            nn.Conv2d(512, 1024, 3, 1),
            nn.BatchNorm2d(1024),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0)
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(4096, 1024),
            nn.ReLU(),
            nn.BatchNorm1d(1024),
            nn.Dropout(0.6),
            nn.Linear(1024, 256),
            nn.ReLU(),
            nn.BatchNorm1d(256),
            nn.Dropout(0.4),
            nn.Linear(256, 11)
        )

    def forward(self, x):
        # input (x): [batch_size, 3, 128, 128]
        # output: [batch_size, 11]

        # Extract features by convolutional layers.
        x = self.cnn_layers(x)

        # The extracted feature map must be flatten before going to fully-connected layers.
```

```
x = x.flatten(1)

# The features are transformed by fully-connected layers to obtain the final logits.
x = self.fc_layers(x)
return x
```

Training

You can finish supervised learning by simply running the provided code without any modification.

The function "get_pseudo_labels" is used for semi-supervised learning. It is expected to get better performance if you use unlabeled data for semi-supervised learning. However, you have to implement the function on your own and need to adjust several hyperparameters manually.

For more details about semi-supervised learning, please refer to Prof. Lee's slides.

Again, please notice that utilizing external data (or pre-trained model) for training is **prohibited**.

```
class PseudoDataset(Dataset):
    def __init__(self, x, y):
        self.x = x
        self.y = y

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

    def __getitem__(self, id):
        return self.x[id][0], self.y[id]

def get_pseudo_labels(dataset, model, threshold=0.9):
    # This functions generates pseudo-labels of a dataset using given model.
    # It returns an instance of DatasetFolder containing images whose prediction confidences exceed a given threshold.
    # You are NOT allowed to use any models trained on external data for pseudo-labeling.
    # from https://github.com/1am9trash/Hung_Yi_Lee_ML_2021/blob/main/hw/hw3/hw3_code.ipynb
    device = "cuda" if torch.cuda.is_available() else "cpu"

    # Construct a data loader.
    data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=False)

    # Make sure the model is in eval mode.
    model.eval()
    # Define softmax function.
    softmax = nn.Softmax(dim=-1)

    idx = []
    labels = []

    # Iterate over the dataset by batches.
    for i, batch in enumerate(data_loader):
        img, _ = batch
        with torch.no_grad():
            logits = model(img.to(device))
        probs = softmax(logits)

        for j, x in enumerate(probs):
            if torch.max(x) > threshold:
                idx.append(i * batch_size + j)
                labels.append(int(torch.argmax(x)))

    model.train()
```

```
        print ("\nNew pseudo label data: {:5d}\n".format(len(idx)))
        dataset = PseudoDataset(Subset(dataset, idx), labels)
        return dataset


    # "cuda" only when GPUs are available.
    device = "cuda" if torch.cuda.is_available() else "cpu"

    # Initialize a model, and put it on the device specified.
    #model = Classifier().to(device)

    """
    #resnet18
    model = torchvision.models.resnet18(pretrained=False).to(device)
    model_weight_path = "./resnet18_pre.pth"
    model.load_state_dict(torch.load(model_weight_path), strict=False)
    """

    #resnet34
    import torchvision.models as models

# Load the pre-trained ResNet-34 model
    model = models.resnet34(pretrained=False).to(device)
    #model = torchvision.models.resnet34(pretrained=False).to(device)
    model_weight_path = "./resnet34-333f7ec4.pth"
    model.load_state_dict(torch.load(model_weight_path), strict=False)

    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, 11).to(device)
    model.device = device
    # fix para in model layers
    #for param in model.parameters():
    #       param.requires_grad = True

    model.device = device

    # For the classification task, we use cross-entropy as the measurement of performance.
    criterion = nn.CrossEntropyLoss()

    # Initialize optimizer, you may fine-tune some hyperparameters such as learning rate on your own.
    optimizer = torch.optim.Adam(model.parameters(), lr=0.0003, weight_decay=1e-5)
```

```
# The number of training epochs.
n_epochs = 20

# Whether to do semi-supervised learning.
do_semi = True
best_acc = 0.0
train_loss_record = []
valid_loss_record = []
train_acc_record = []
valid_acc_record = []


for epoch in range(n_epochs):
    # ---------- TODO ----------
    # In each epoch, relabel the unlabeled dataset for semi-supervised learning.
    # Then you can combine the labeled dataset and pseudo-labeled dataset for the training.
    if do_semi and best_acc > 0.75:
        # Obtain pseudo-labels for unlabeled data using trained model.
        pseudo_set = get_pseudo_labels(unlabeled_set, model)

        # Construct a new dataset and a data loader for training.
        # This is used in semi-supervised learning only.
        concat_dataset = ConcatDataset([train_set, pseudo_set])
        train_loader = DataLoader(concat_dataset, batch_size=batch_size, shuffle=True, num_workers=16, pin_memory=True, drop_last=

    # ---------- Training ----------
    # Make sure the model is in train mode before training.
    model.train()

    # These are used to record information in training.
    train_loss = []
    train_accs = []

    # Iterate the training set by batches.
    for batch in tqdm(train_loader):

        # A batch consists of image data and corresponding labels.
        imgs, labels = batch

        # Forward the data. (Make sure data and model are on the same device.)
        logits = model(imgs.to(device))

        # Calculate the cross-entropy loss.
```

```
    # We don't need to apply softmax before computing cross-entropy as it is done automatically.
    loss = criterion(logits, labels.to(device))

    # Gradients stored in the parameters in the previous step should be cleared out first.
    optimizer.zero_grad()

    # Compute the gradients for parameters.
    loss.backward()

    # Clip the gradient norms for stable training.
    # 避免梯度爆炸or消失，將梯度限制在某個範圍
    grad_norm = nn.utils.clip_grad_norm_(model.parameters(), max_norm=10)

    # Update the parameters with computed gradients.
    optimizer.step()

    # Compute the accuracy for current batch.
    acc = (logits.argmax(dim=-1) == labels.to(device)).float().mean()

    # Record the loss and accuracy.
    train_loss.append(loss.item())
    train_accs.append(acc)

# The average loss and accuracy of the training set is the average of the recorded values.
train_loss = sum(train_loss) / len(train_loss)
train_acc = sum(train_accs) / len(train_accs)

# Print the information.
print(f"[ Train | {epoch + 1:03d}/{n_epochs:03d} ] loss = {train_loss:.5f}, acc = {train_acc:.5f}")

# ---------- Validation ----------
# Make sure the model is in eval mode so that some modules like dropout are disabled and work normally.
model.eval()

# These are used to record information in validation.
valid_loss = []
valid_accs = []

# Iterate the validation set by batches.
for batch in tqdm(valid_loader):

    # A batch consists of image data and corresponding labels.
    imgs, labels = batch
```

```
        imgs, labels = batch

        # We don’t need gradient in validation.
        # Using torch.no_grad() accelerates the forward process.
        with torch.no_grad():
            logits = model(imgs.to(device))

        # We can still compute the loss (but not the gradient).
        loss = criterion(logits, labels.to(device))

        # Compute the accuracy for current batch.
        acc = (logits.argmax(dim=-1) == labels.to(device)).float().mean()

        # Record the loss and accuracy.
        valid_loss.append(loss.item())
        valid_accs.append(acc)

# The average loss and accuracy for entire validation set is the average of the recorded values.
valid_loss = sum(valid_loss) / len(valid_loss)
valid_acc = sum(valid_accs) / len(valid_accs)

# Print the information.
print(f”[ Valid | {epoch + 1:03d}/{n_epochs:03d} ] loss = {valid_loss:.5f}, acc = {valid_acc:.5f}”)

if valid_acc > best_acc:
    best_acc = valid_acc
    torch.save(model.state_dict(), ’./model.ckpt’)
    print(’saving model with acc {:.5f}’.format(best_acc))

#record for visualization
train_loss_record.append(train_loss)
valid_loss_record.append(valid_loss)
train_acc_record.append(train_acc)
valid_acc_record.append(valid_acc)
```

```
100%|██████████| 297/297 [04:11<00:00,  1.18it/s]
[ Train | 002/020 ] loss = 0.36731, acc = 0.88336
100%|██████████| 6/6 [00:08<00:00,  1.40s/it]
[ Valid | 002/020 ] loss = 0.74181, acc = 0.79036
saving model with acc 0.79036

New pseudo label data: 23815

100%|██████████| 330/330 [04:38<00:00,  1.19it/s]
[ Train | 003/020 ] loss = 0.26241, acc = 0.91551
100%|██████████| 6/6 [00:07<00:00,  1.26s/it]
[ Valid | 003/020 ] loss = 0.85470, acc = 0.80781
saving model with acc 0.80781

New pseudo label data: 26423

100%|██████████| 350/350 [04:40<00:00,  1.25it/s]
[ Train | 004/020 ] loss = 0.25360, acc = 0.91955
100%|██████████| 6/6 [00:07<00:00,  1.20s/it]
[ Valid | 004/020 ] loss = 0.68226, acc = 0.81667
saving model with acc 0.81667

New pseudo label data: 26920

100%|██████████| 354/354 [04:53<00:00,  1.21it/s]
[ Train | 005/020 ] loss = 0.20765, acc = 0.93278
100%|██████████| 6/6 [00:05<00:00,  1.01it/s]
[ Valid | 005/020 ] loss = 0.85378, acc = 0.78021

New pseudo label data: 28241

100%|██████████| 365/365 [04:45<00:00,  1.28it/s]
[ Train | 006/020 ] loss = 0.20786, acc = 0.93305
100%|██████████| 6/6 [00:05<00:00,  1.03it/s]
[ Valid | 006/020 ] loss = 0.91519, acc = 0.77448

New pseudo label data: 28943

100%|██████████| 370/370 [04:42<00:00,  1.31it/s]
[ Train | 007/020 ] loss = 0.19510, acc = 0.93839
100%|██████████| 6/6 [00:07<00:00,  1.17s/it]
[ Valid | 007/020 ] loss = 0.87018, acc = 0.77839

New pseudo label data: 29055

 87%|████████▋ | 322/371 [04:13<00:38,  1.27it/s]
```

```
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-35-387d53302423> in <cell line: 50>()
     96 
     97         # Compute the accuracy for current batch.
---> 98         acc = (logits.argmax(dim=-1) == labels.to(device)).float().mean()
     99 
    100         # Record the loss and accuracy.

KeyboardInterrupt: 
```

Testing

For inference, we need to make sure the model is in eval mode, and the order of the dataset should not be shuffled ("shuffle=False" in test_loader).

Last but not least, don't forget to save the predictions into a single CSV file. The format of CSV file should follow the rules mentioned in the