Machine Learning

Challenge (Deep Learning)

Prof. Eunwoo Kim

What you have to submit

- ✓ Challenge: June 2 – June 16 (two weeks)
- ✓ Implementation codes including model weights written in Python (in a zip file) for two tasks, supervised learning and semi-supervised learning, as follows.
  - {StudentID}.zip
    - ◆ Supervised_Learning.py
    - ◆ Semi-Supervised_Learning.py
    - ◆ logs (folder)
      - ● Supervised_Learning (folder)
        - ■ best_model.pt
      - ● Semi-Supervised_Learning (folder)
        - ■ best_model_1.pt
        - ■ best_model_2.pt

  ** "logs" folder and subfolders are automatically saved (don't edit).

  ** Do not submit data folder.

  ** The name of submitted zip files should be your student ID.

  ** You should not change the structure of the provided folders

  ** You should only submit a single zip file that contains every result in a single folder.

  ** You can submit the learned code max 3 times during the challenging period.

- ✓ Submit it to TA (through e-mail)
  - Class-01: Jungkyoo Shin (neo293@cau.ac.kr)
  - Class-02: Jiho Lee (j2hoooo@cau.ac.kr)

Results on leaderboard

- ✓ You can check your results on a daily basis.
  - TA will evaluate the results of the codes and models submitted prior to midnight and will provide score updates at each subsequent day, 14:00.
  - Check it out in the following leaderboard: Link
  - Challenge scores will be based on the rank of the leaderboard (a total of 30 points).

Datasets

- ✓ Task 1: Supervised Learning
  - Image dataset of 96×96 size (number of classes: 50).
  - We provide 30,000 images (each class has 600 images) as a labeled training set and 2500 for a validation set.
  - We do not provide the test set, which will be used to evaluate student's submissions by TA.
- ✓ Task 2: Semi-Supervised Learning
  - Image dataset of 96×96 size (number of classes: 10).
  - We provide 6,000 images (each class has 600 images) as labeled training datasets, 6,000 images as unlabeled training sets, and 500 for validation sets.
- ✓ Since we do not provide the test sets, you can evaluate your learned models using the provided validation sets (but do NOT use the validation samples to learn your models).
- ✓ Link for Dataset & Baseline Codes: Link

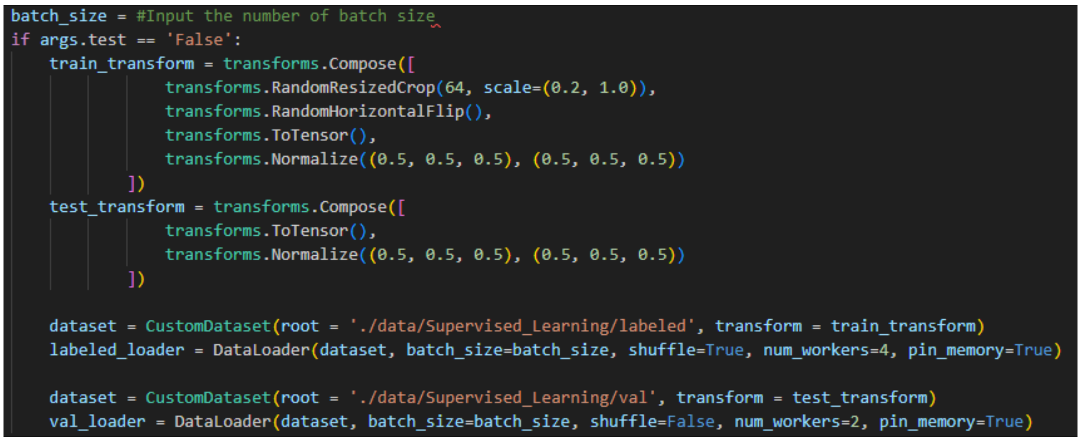
Training Process

✓ Task 1: Supervised Learning

- Students should train a deep learning model to classify images into 50 labels using the provided image datasets with annotated labels.

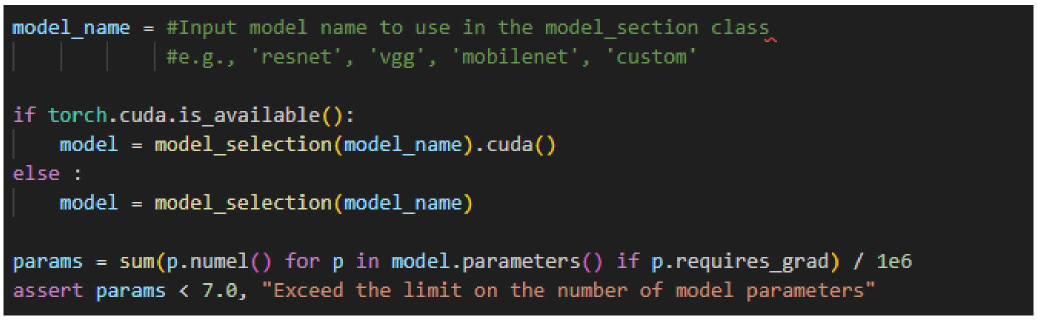
- We provide basic codes with blanks. Students should fill in the blank and make modifications to achieve higher classification performance.

- Preparation of data

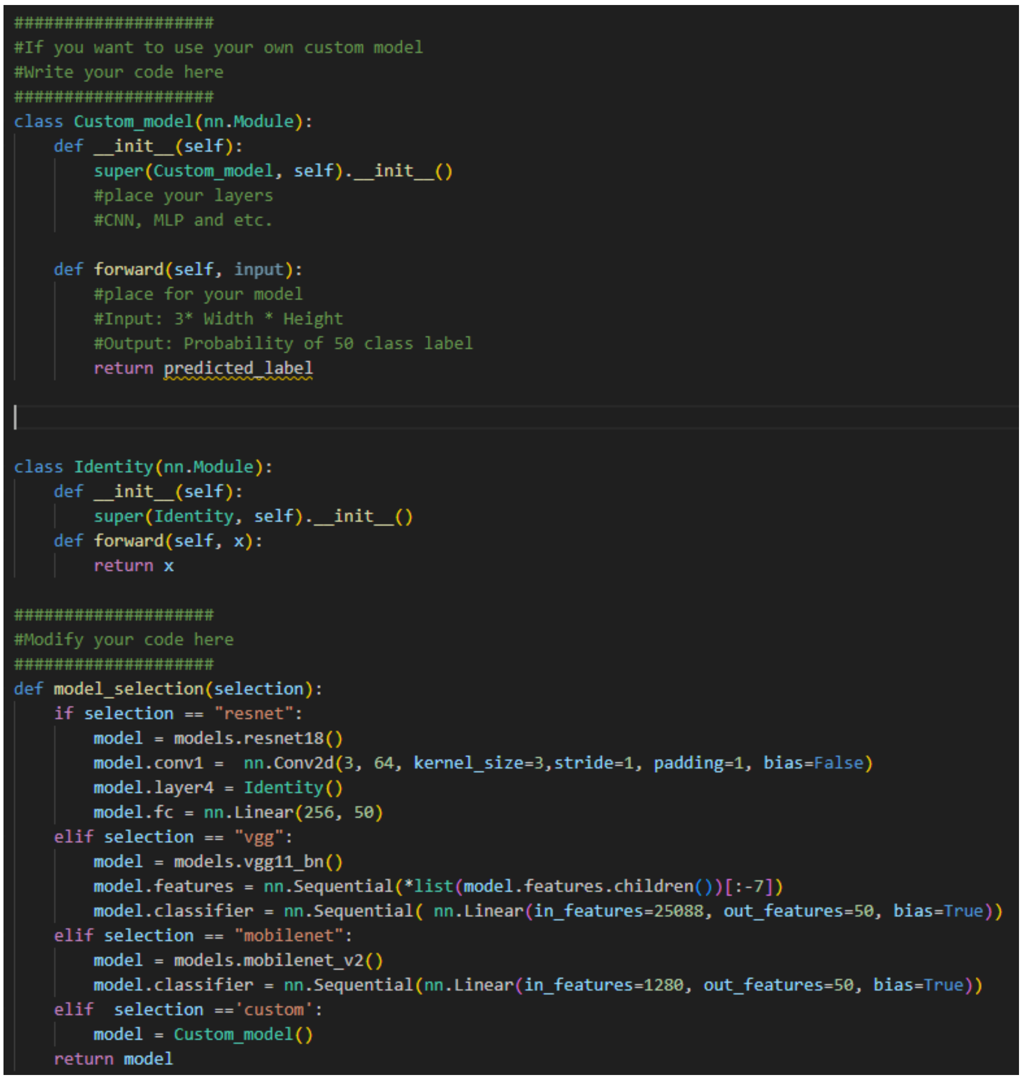


```
batch_size = #Input the number of batch size
if args.test == 'False':
    train_transform = transforms.Compose([
                transforms.RandomResizedCrop(64, scale=(0.2, 1.0)),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
            ])
    test_transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
            ])

    dataset = CustomDataset(root = './data/Supervised_Learning/labeled', transform = train_transform)
    labeled_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=4, pin_memory=True)

    dataset = CustomDataset(root = './data/Supervised_Learning/val', transform = test_transform)
    val_loader = DataLoader(dataset, batch_size=batch_size, shuffle=False, num_workers=2, pin_memory=True)
```

- Preparation of model
  - We provide three popular deep learning CNN models as default.
    - ResNet, MobileNet, VGG
    - Students can use the reference models or construct your own models.

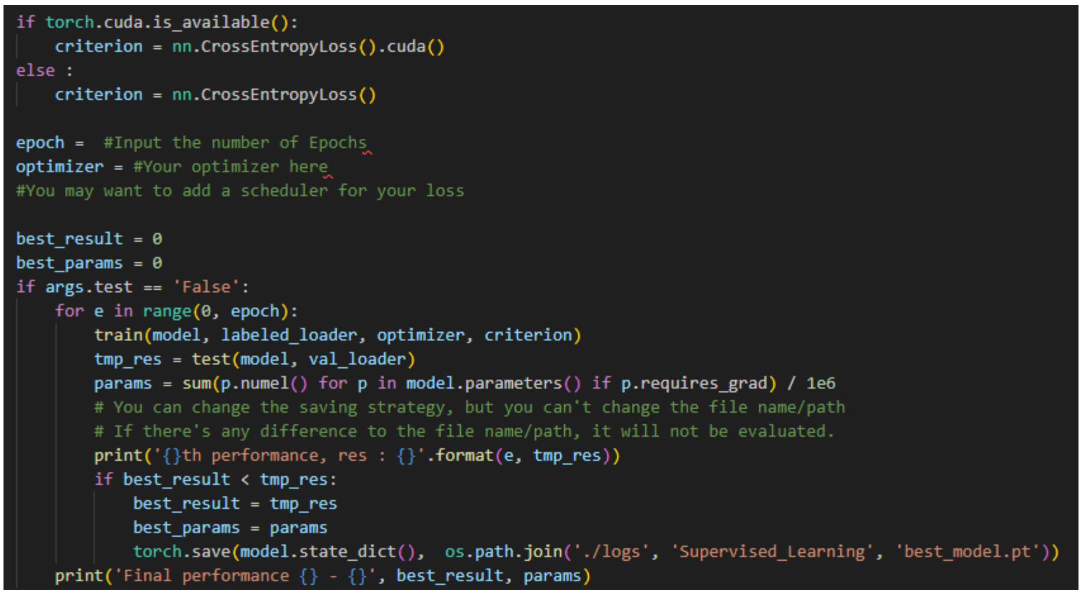


```python
model_name = #Input model name to use in the model_section class
             #e.g., 'resnet', 'vgg', 'mobilenet', 'custom'

if torch.cuda.is_available():
    model = model_selection(model_name).cuda()
else :
    model = model_selection(model_name)

params = sum(p.numel() for p in model.parameters() if p.requires_grad) / 1e6
assert params < 7.0, "Exceed the limit on the number of model parameters"
```



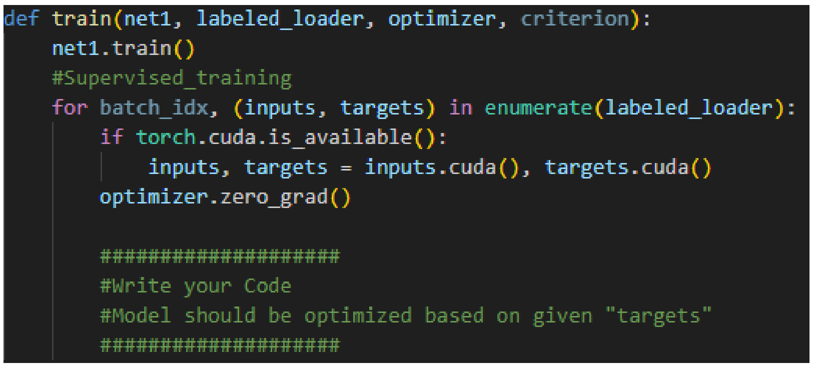
```python
####################
#If you want to use your own custom model
#Write your code here
####################
class Custom_model(nn.Module):
    def __init__(self):
        super(Custom_model, self).__init__()
        #place your layers
        #CNN, MLP and etc.

    def forward(self, input):
        #place for your model
        #Input: 3* Width * Height
        #Output: Probability of 50 class label
        return predicted_label


class Identity(nn.Module):
    def __init__(self):
        super(Identity, self).__init__()
    def forward(self, x):
        return x

####################
#Modify your code here
####################
def model_selection(selection):
    if selection == "resnet":
        model = models.resnet18()
        model.conv1 =  nn.Conv2d(3, 64, kernel_size=3,stride=1, padding=1, bias=False)
        model.layer4 = Identity()
        model.fc = nn.Linear(256, 50)
    elif selection == "vgg":
        model = models.vgg11_bn()
        model.features = nn.Sequential(*list(model.features.children())[:-7])
        model.classifier = nn.Sequential( nn.Linear(in_features=25088, out_features=50, bias=True))
    elif selection == "mobilenet":
        model = models.mobilenet_v2()
        model.classifier = nn.Sequential(nn.Linear(in_features=1280, out_features=50, bias=True))
    elif  selection =='custom':
        model = Custom_model()
    return model
```

- Main function



```python
if torch.cuda.is_available():
    criterion = nn.CrossEntropyLoss().cuda()
else :
    criterion = nn.CrossEntropyLoss()

epoch =  #Input the number of Epochs
optimizer = #Your optimizer here
#You may want to add a scheduler for your loss

best_result = 0
best_params = 0
if args.test == 'False':
    for e in range(0, epoch):
        train(model, labeled_loader, optimizer, criterion)
        tmp_res = test(model, val_loader)
        params = sum(p.numel() for p in model.parameters() if p.requires_grad) / 1e6
        # You can change the saving strategy, but you can't change the file name/path
        # If there's any difference to the file name/path, it will not be evaluated.
        print('{}th performance, res : {}'.format(e, tmp_res))
        if best_result < tmp_res:
            best_result = tmp_res
            best_params = params
            torch.save(model.state_dict(),  os.path.join('./logs', 'Supervised_Learning', 'best_model.pt'))
    print('Final performance {} - {}', best_result, params)
```

- Some training functions will be left as blank. Excluded components are as below.
  - ◆ Feeding input data to the model.
  - ◆ Calculating the objective function.
  - ◆ Backpropagation process.

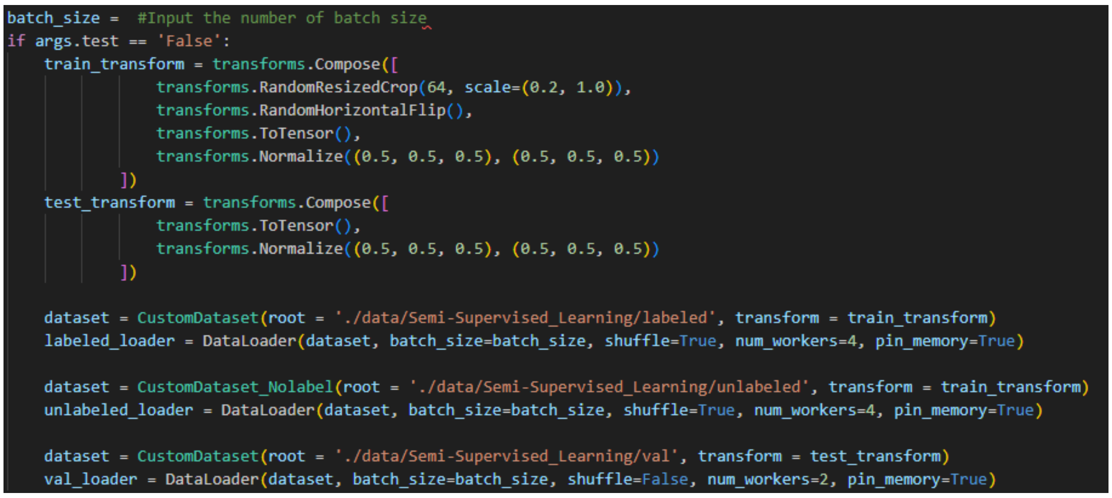


```python
def train(net1, labeled_loader, optimizer, criterion):
    net1.train()
    #Supervised_training
    for batch_idx, (inputs, targets) in enumerate(labeled_loader):
        if torch.cuda.is_available():
            inputs, targets = inputs.cuda(), targets.cuda()
        optimizer.zero_grad()

        ####################
        #Write your Code
        #Model should be optimized based on given "targets"
        ####################
```

- Students can modify the given codes as well, to achieve optimal performance.
  - ◆ Optimize a chosen model (ResNet, VGG, MobileNet, or custom CNN model).
  - ◆ Modify the parameters such as batch size, learning rate, etc.
- You don't need to set the *argparser*, just use as default.
- Evaluation to this project will be based on the classification accuracy. However, the number of parameters of deep learning model **should NOT exceed 7 Million**. Any model that are larger than the given limit will not be simply evaluated.
- We provide the code to evaluate the results and save the model in '. /log' folder as 'best_model.pt'. Do NOT change the name or path of the saved model. If changed, results will not be evaluated.

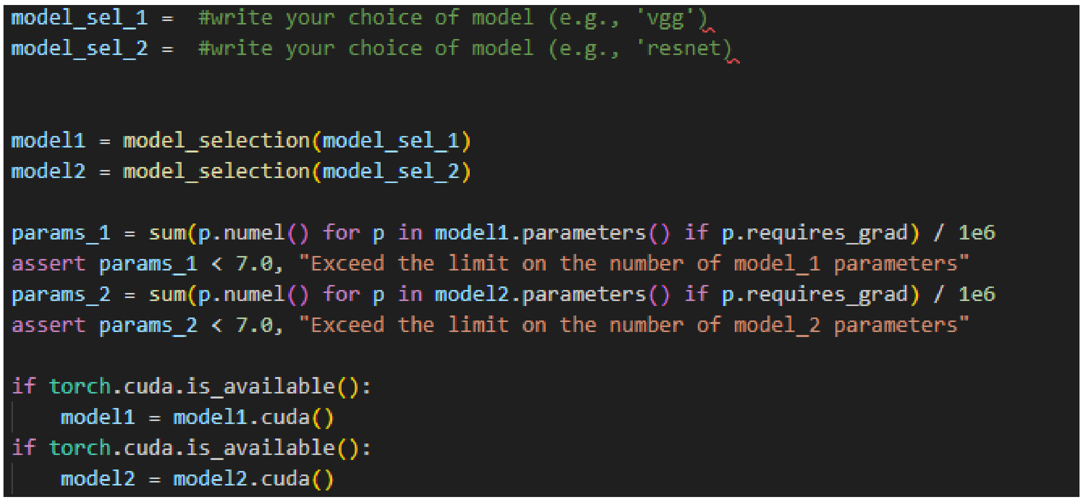
✓ Task 2: Semi-Supervised Learning

- Students should train the deep learning models to classify images into 10 labels, using both labeled and unlabeled images.
- Students should use the **co-training method** to learn **two different CNN models** simultaneously to utilize unlabeled image to boost the classification performance.
  - ◆ Take two CNN models to perform co-training for semi-supervised learning.
- Two models that are trained by labeled dataset takes in the unlabeled set and compare the results. The results of two models that match together (if they classify the same label) produce pseudo labels that can be utilized as training set.
- We provide basic codes with blanks. Students should fill in the blank and make modifications to achieve higher classification performance.
- Preparation of data

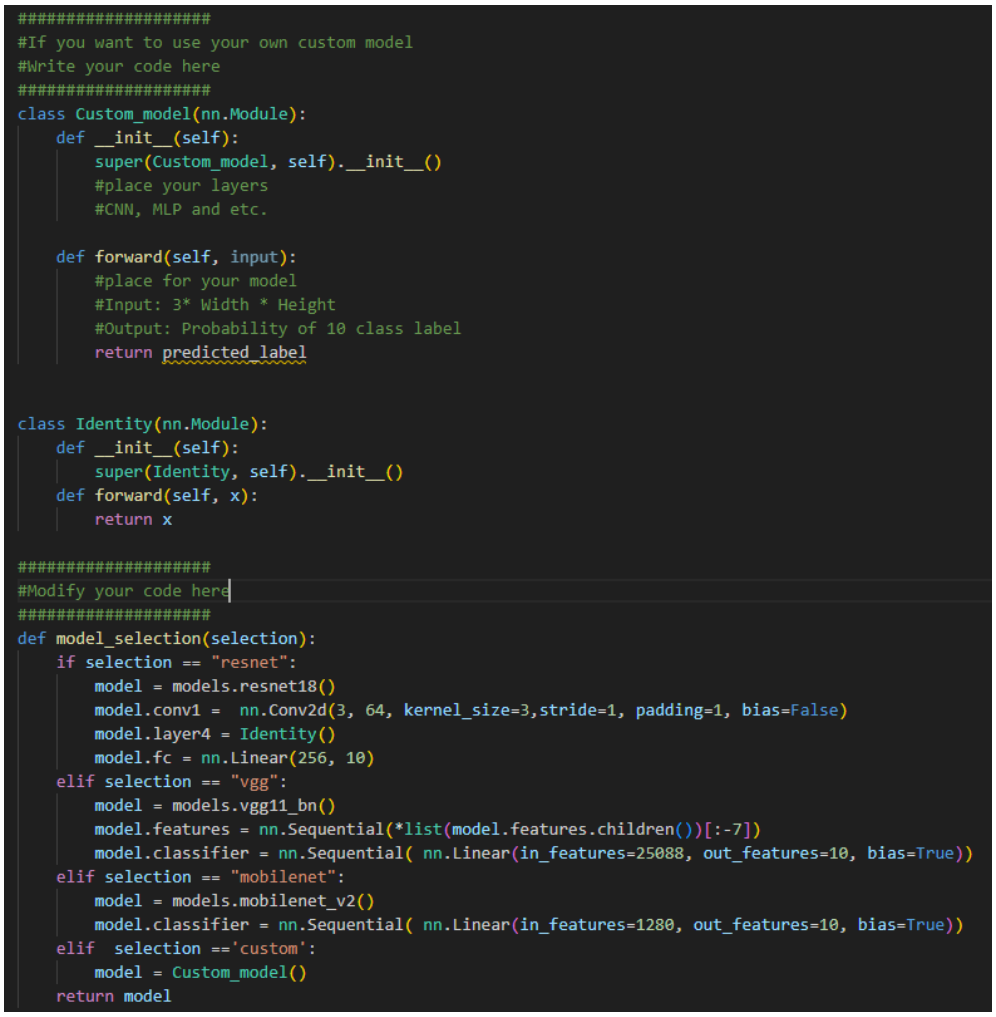


```python
batch_size =  #Input the number of batch size
if args.test == 'False':
    train_transform = transforms.Compose([
                transforms.RandomResizedCrop(64, scale=(0.2, 1.0)),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
            ])
    test_transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
            ])

    dataset = CustomDataset(root = './data/Semi-Supervised_Learning/labeled', transform = train_transform)
    labeled_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=4, pin_memory=True)

    dataset = CustomDataset_Nolabel(root = './data/Semi-Supervised_Learning/unlabeled', transform = train_transform)
    unlabeled_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=4, pin_memory=True)

    dataset = CustomDataset(root = './data/Semi-Supervised_Learning/val', transform = test_transform)
    val_loader = DataLoader(dataset, batch_size=batch_size, shuffle=False, num_workers=2, pin_memory=True)
```

- Preparation of model

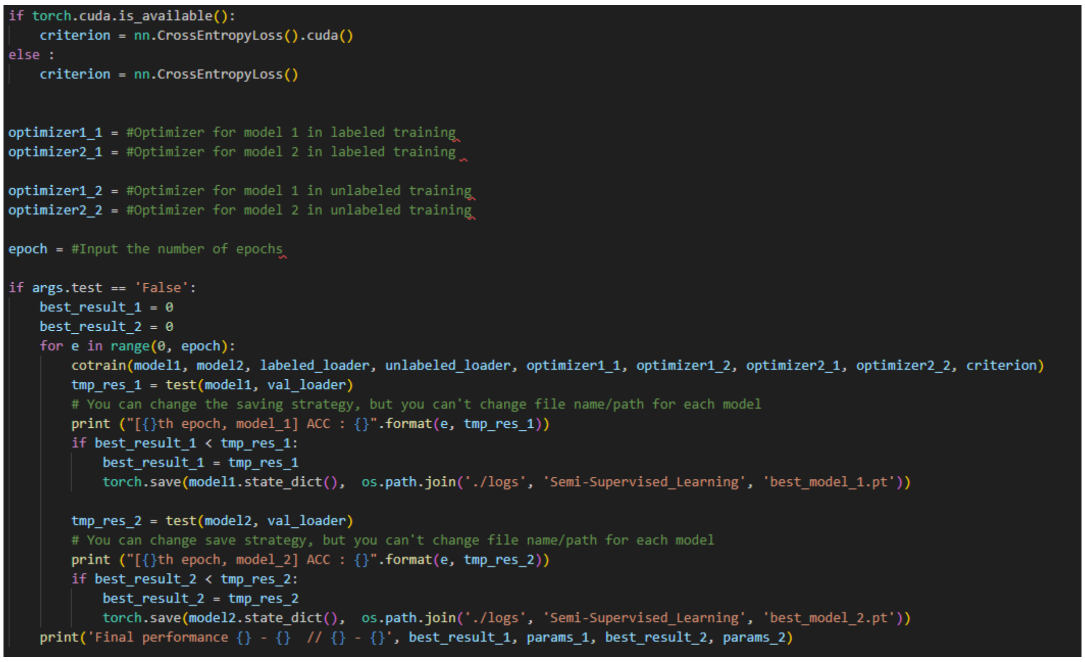


```python
model_sel_1 =  #write your choice of model (e.g., 'vgg')
model_sel_2 =  #write your choice of model (e.g., 'resnet)


model1 = model_selection(model_sel_1)
model2 = model_selection(model_sel_2)

params_1 = sum(p.numel() for p in model1.parameters() if p.requires_grad) / 1e6
assert params_1 < 7.0, "Exceed the limit on the number of model_1 parameters"
params_2 = sum(p.numel() for p in model2.parameters() if p.requires_grad) / 1e6
assert params_2 < 7.0, "Exceed the limit on the number of model_2 parameters"

if torch.cuda.is_available():
    model1 = model1.cuda()
if torch.cuda.is_available():
    model2 = model2.cuda()
```



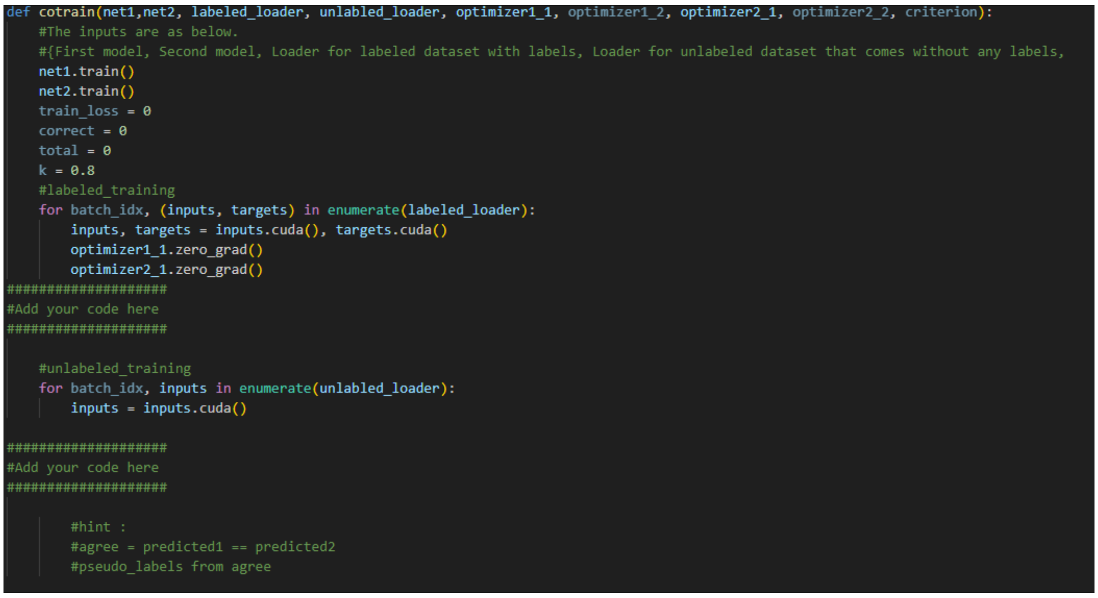
```python
####################
#If you want to use your own custom model
#Write your code here
####################
class Custom_model(nn.Module):
    def __init__(self):
        super(Custom_model, self).__init__()
        #place your layers
        #CNN, MLP and etc.

    def forward(self, input):
        #place for your model
        #Input: 3* Width * Height
        #Output: Probability of 10 class label
        return predicted_label


class Identity(nn.Module):
    def __init__(self):
        super(Identity, self).__init__()
    def forward(self, x):
        return x

####################
#Modify your code here
####################
def model_selection(selection):
    if selection == "resnet":
        model = models.resnet18()
        model.conv1 =  nn.Conv2d(3, 64, kernel_size=3,stride=1, padding=1, bias=False)
        model.layer4 = Identity()
        model.fc = nn.Linear(256, 10)
    elif selection == "vgg":
        model = models.vgg11_bn()
        model.features = nn.Sequential(*list(model.features.children())[:-7])
        model.classifier = nn.Sequential( nn.Linear(in_features=25088, out_features=10, bias=True))
    elif selection == "mobilenet":
        model = models.mobilenet_v2()
        model.classifier = nn.Sequential( nn.Linear(in_features=1280, out_features=10, bias=True))
    elif  selection =='custom':
        model = Custom_model()
    return model
```

- Main function



```python
if torch.cuda.is_available():
    criterion = nn.CrossEntropyLoss().cuda()
else :
    criterion = nn.CrossEntropyLoss()


optimizer1_1 = #Optimizer for model 1 in labeled training
optimizer2_1 = #Optimizer for model 2 in labeled training

optimizer1_2 = #Optimizer for model 1 in unlabeled training
optimizer2_2 = #Optimizer for model 2 in unlabeled training

epoch = #Input the number of epochs

if args.test == 'False':
    best_result_1 = 0
    best_result_2 = 0
    for e in range(0, epoch):
        cotrain(model1, model2, labeled_loader, unlabeled_loader, optimizer1_1, optimizer1_2, optimizer2_1, optimizer2_2, criterion)
        tmp_res_1 = test(model1, val_loader)
        # You can change the saving strategy, but you can't change file name/path for each model
        print ("[{}th epoch, model_1] ACC : {}".format(e, tmp_res_1))
        if best_result_1 < tmp_res_1:
            best_result_1 = tmp_res_1
            torch.save(model1.state_dict(),  os.path.join('./logs', 'Semi-Supervised_Learning', 'best_model_1.pt'))

        tmp_res_2 = test(model2, val_loader)
        # You can change save strategy, but you can't change file name/path for each model
        print ("[{}th epoch, model_2] ACC : {}".format(e, tmp_res_2))
        if best_result_2 < tmp_res_2:
            best_result_2 = tmp_res_2
            torch.save(model2.state_dict(),  os.path.join('./logs', 'Semi-Supervised_Learning', 'best_model_2.pt'))
    print('Final performance {} - {}  // {} - {}', best_result_1, params_1, best_result_2, params_2)
```

- Students should fill in the blank in the given code.



```
def cotrain(net1,net2, labeled_loader, unlabled_loader, optimizer1_1, optimizer1_2, optimizer2_1, optimizer2_2, criterion):
    #The inputs are as below.
    #{First model, Second model, Loader for labeled dataset with labels, Loader for unlabeled dataset that comes without any labels,
    net1.train()
    net2.train()
    train_loss = 0
    correct = 0
    total = 0
    k = 0.8
    #labeled_training
    for batch_idx, (inputs, targets) in enumerate(labeled_loader):
        inputs, targets = inputs.cuda(), targets.cuda()
        optimizer1_1.zero_grad()
        optimizer2_1.zero_grad()
####################
#Add your code here
####################

    #unlabeled_training
    for batch_idx, inputs in enumerate(unlabled_loader):
        inputs = inputs.cuda()

####################
#Add your code here
####################

        #hint :
        #agree = predicted1 == predicted2
        #pseudo_labels from agree
```

- ◆ Write the co-training strategy in the blank.
- ◆ The method how to introduce pseudo labels (by comparing confidence of the models or by collecting the same classification result from two models) can be freely modified. There is no strict answer/guideline, finding an optimal solution with highest performance is up to you.
- ◆ You can optionally apply ensemble of the two learned models to give a final classification answer or just collect the result from a better model.

- Student do not need to set the *argparser*, just use as default.
- Students can employ the models used in the supervised learning.
- Evaluation of this project will be based on the classification performance and the parameter size of the learned model is limited to 7 million.