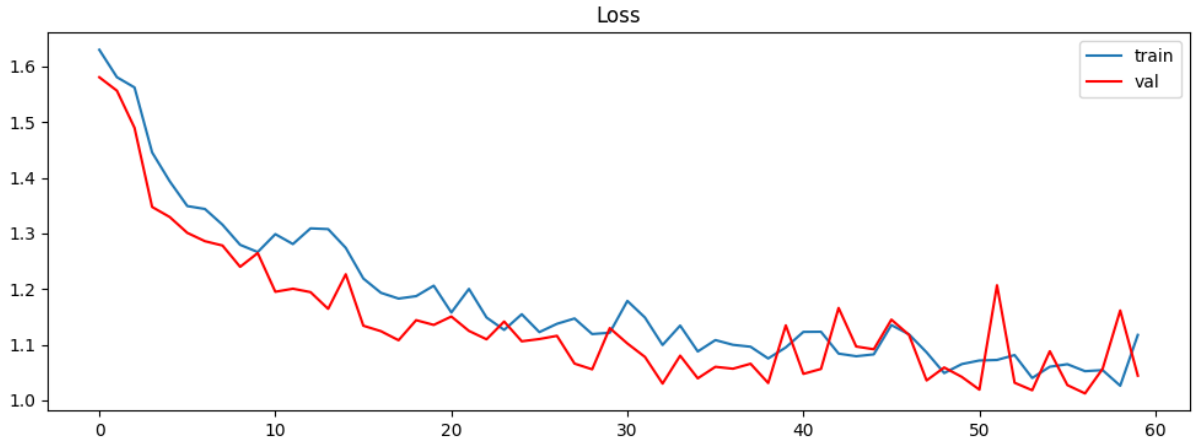
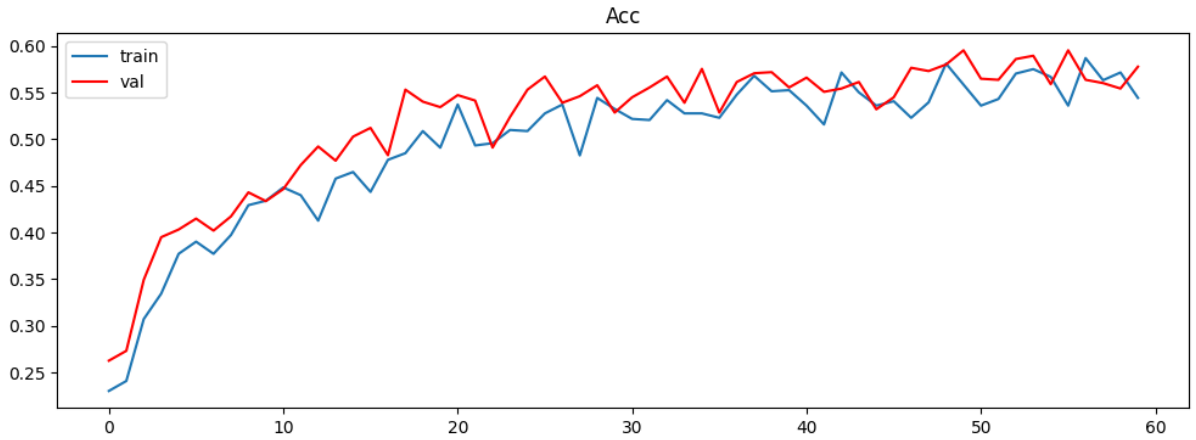
A3 Semi-supervised Learning

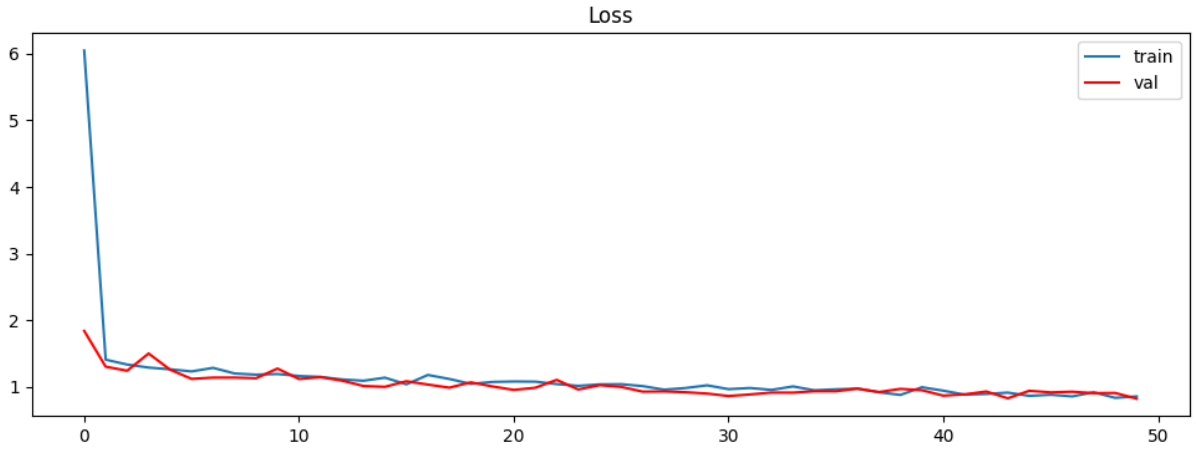
First of all, I tried to use a similar structure as several two to three convolution layers with one maxpool layer (conv-pool-conv-pool-conv-conv-conv-pool with two fc layers), and the accuracy of my training set seems quite low, which was only around 0.577960.

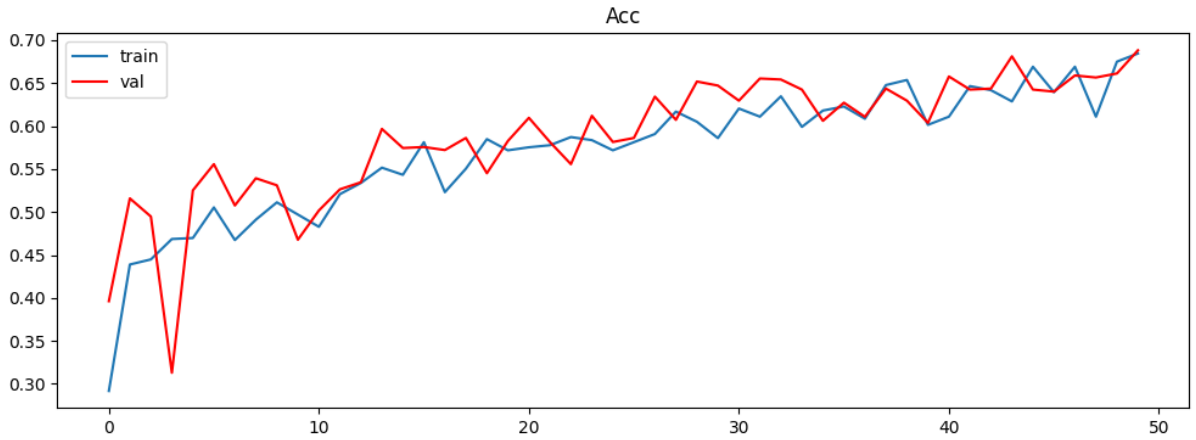




↑ The accuracy did not change much as I added more layers into my first cnn structure. So then I tried to start all over again and find some easier structure as a baseline to use this as a reference to improve the accuracy. I used four basic convolution layer groups (conv-batchnorm-relu-maxpool) and two fully-connected network layers as my baseline, and the loss function was nn.CrossEntropyLoss(), the optimizer was nn.Adam with default parameters, and the threshold was set to 0.7 which I thought it could lead to higher accuracy. The result of the baseline is as below, the accuracy of both validation and training looked great compared to my first try. The learning curve of both losses lied closely.

```
YourCNNModel(
  (conv): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_model): Sequential(
    (0): Linear(in_features=50176, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=5, bias=True)
  )
)
```





Here are several methods that I used to improve the model accuracy, and some discovered when trying different parameters, network structures, and functions.

- **Data Augmentation**
  - My baseline was :
    RandomResizedCrop / RandomHorizontalFlip / RandomAffine / RandomRotation / ToTensor / Normalize
  - I did use transforms.AutoAugment() from Internet references, but with the same network structure, the accuracy tended to be lower, thus I took the function away.
  - I also tried using more functions, for example transform.ColorJitter() and set the brightness contrast to 0.5, and the accuracy increased a bit with the same network. I think that ColorJitter did help with the model training since the images in the files seem to be taken in different light sources leading to inaccuracy.
- **Adding Convolution Layers / Adding Linear Layers**
  - Two or more convolution layers with one maxpool : the accuracy did not increase but decrease a lot. And I thought it might, due to too many redundant channels, lead to training irrelevant features.
  - Same methods above with linear layers also did not increase the accuracy.
  - So I think that the accuracy of semi-supervised learning does not have absolute correlation with the number of layers.
- **Adding Channels**
  - These parameters did help increase accuracy when there weren't too many layers, but as the layers increased, adding channels did not really help but decrease the accuracy and the time to run.
- **Adding kernel size, and padding**
  - The higher kernel size, for example 5 * 5 or 11 * 11 might decrease the accuracy.
- **Dropout**
  - I used drop out in the fully-connected layer. At first I tried :
    Linear-Batchnorm-ReLU-Dropout(0.6) –
    Linear-Batchnorm-ReLU-Dropout(0.4) –
    Linear-Batchnorm-ReLU-Dropout(0.3) –
    Linear,
  - And I found out that my model was having validation accuracy >> training accuracy, this might be due to dropping out too many. Therefore I tried several methods :
    Linear-Batchnorm-ReLU-Dropout(0.5) –
    Linear-Batchnorm-ReLU-Dropout(0.3) –
    Linear-Batchnorm-ReLU-Dropout(0.3) –
    Linear, like this by just lowering the dropout percentage, and the accuracy absolutely did not increase.
  - After googling for the reason that might cause validation accuracy >> training accuracy, letting the dropout only in one layer could be a valid method to increase accuracy, so then I left only the first layer of fully-connected layer to dropout as a percentage of 0.5. And the accuracy seems to increase and the gap between validation and training did decrease a lot.
- **ReLU or SELU**

    - I tried at once changing ReLU to SELU, and the performance decreased.
- **Adam, SGD, RMSprop, NAdam**
    - I began with Adam as a baseline optimizer, since nn.Adam is objectively a good optimizer.
    - Then I tried the other three optimizers, SGD and NAdam did not perform so well, and RMSprop was slightly better than those two.
    - Therefore, I chose Adam as my model's optimizer.
- **Learning Rate, weight_decay**
    - I used Adam optimizer and set the learning rate to 0.0001
    - Lower learning rate caused overfitting problems
    - I used Adam optimizer and tried set the learning rate to 1e-5
    - It had the problem as I found when using too much regularization, therefore I changed it to 1e-4, and the model performed better.
- **Epochs**
    - Higher epochs might improve the accuracy, but it certainly needs to find a balance point where the model does not overfit.
    - And it might depend on the network structure, so I changed to epochs based on the learning curve graph.
- **Threshold**
    - Firstly, I increased the threshold from 0.5 to 0.7, and the accuracy of unlabeled-image was better.
    - And I tried to increase the threshold from 0.7 to 0.9, and I found out that the number of used unlabeled images was only 150-300, which might not be a good quantity for training. Therefore, I changed the threshold back to 0.7, and the unlabeled image that was used increased to 1400.
    - So I think that threshold of 0.7 is an average good standard for unlabeled data.

So the main methods that helped improve performance were data augmentation, regularization, number of epochs, and threshold.
And here is the best model I had :

1. **Data Augmentation for training :**

```
transforms_train = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomAffine(degrees=0, shear=(0, 0, 0, 45)),
    transforms.RandomRotation(60),
    transforms.ColorJitter(brightness=0.5, contrast=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

2. **CNN architecture :**

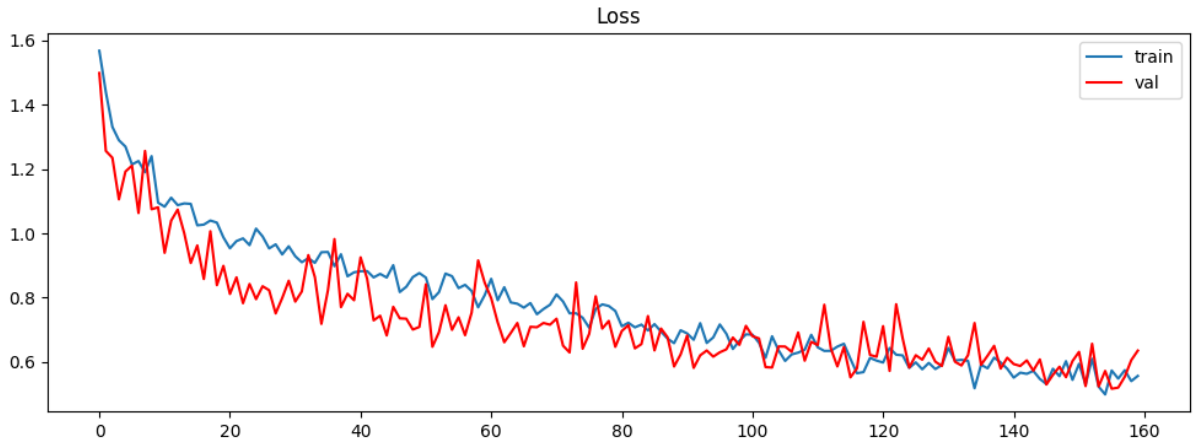
```
self.conv = nn.Sequential(
nn.Conv2d(in_channels=3, out_channels=64, kernel_size=5, stride=1,
padding=2),
nn.BatchNorm2d(64),
```

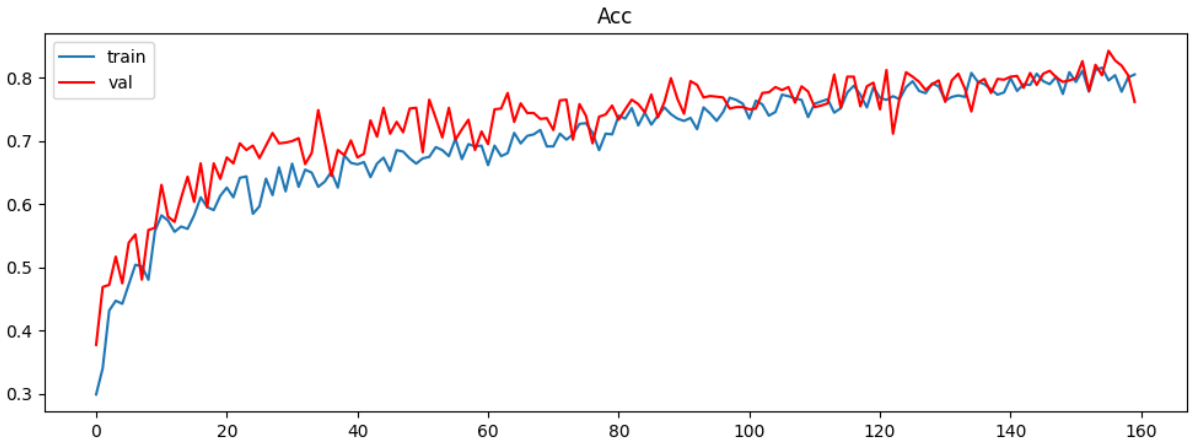
```
nn.ReLU(),
nn.MaxPool2d(2, 2),
# 64 * 112 * 112

nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1,
padding=1),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.MaxPool2d(2, 2),
# 64 * 56 * 56

nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1,
padding=1),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.MaxPool2d(2, 2),
# 128 * 28 * 28

nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=1,
padding=1),
nn.BatchNorm2d(256),
nn.ReLU(),
nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, stride=1,
padding=1),
nn.BatchNorm2d(256),
nn.ReLU(),
nn.MaxPool2d(2, 2),
# 256 * 14 * 14
nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, stride=1,
padding=1),
nn.BatchNorm2d(512),
nn.ReLU(),
nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=3, stride=1,
padding=1),
nn.BatchNorm2d(1024),
nn.ReLU(),
nn.MaxPool2d(2, 2),
# 1024 * 7 * 7

nn.Conv2d(in_channels=1024, out_channels=1024, kernel_size=3, stride=1,
padding=1),
nn.BatchNorm2d(1024),
nn.ReLU(),
```

```
nn.MaxPool2d(2, 2)
# 1024 * 3 * 3
)
self.fc_model = nn.Sequential(
nn.Linear(1024 * 3 * 3, 1024),
nn.ReLU(),
nn.Dropout(0.5),
nn.Linear(1024, 512),
nn.BatchNorm1d(512),
nn.ReLU(),
nn.Linear(512, 512),
nn.BatchNorm1d(512),
nn.ReLU(),
nn.Linear(512, 5)
)
```

3. **Loss and optimizer :**
   CrossEntropyLoss() and Adam(lr=0.0001, weight_decay=1e-4)
4. **Threshold :**
   0.7

Result of Self-training model :




Acc
train
val
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100
120
140
160