Lab 2 Report

Deep Learning

Name: Kai-Jie Lin
Student ID: 110652019

April 13, 2024

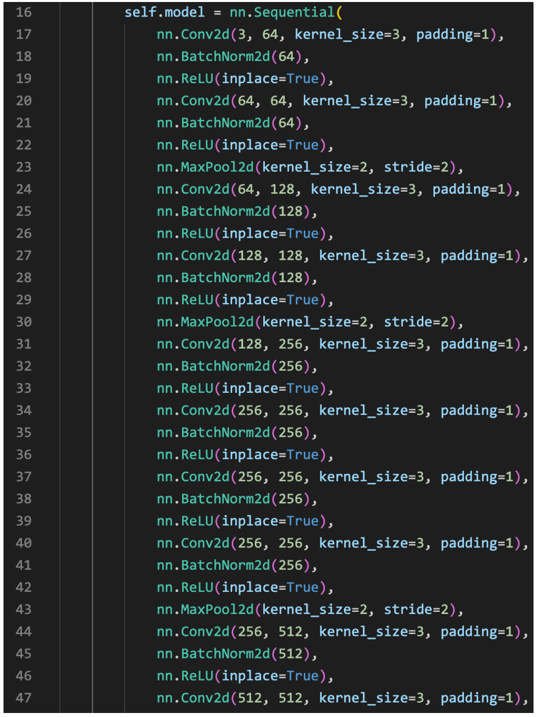
1 Introduction

In this lab, I implemented VGG19 and ResNet50 architecture with Pytorch. I use the models to classify the butterflies and moths dataset with 100 classes. Futhermore, I designed my own dataloader and data preprocessing technique to train the model. Finally, I evaluated the model with the test dataset and calcßlated the accuracy.

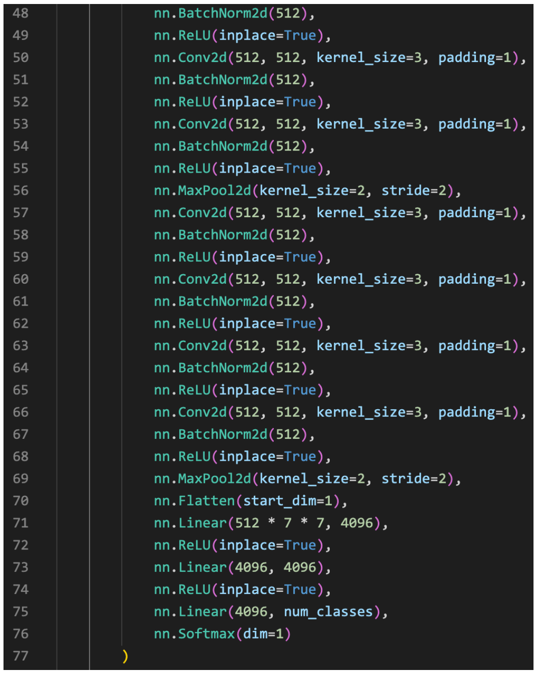
2 Implementation Details

2.1 Details of models

VGG19
Optimizer: SGD
Criterion: CrossEntropyLoss
Model architecture:



```
        self.model = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
```



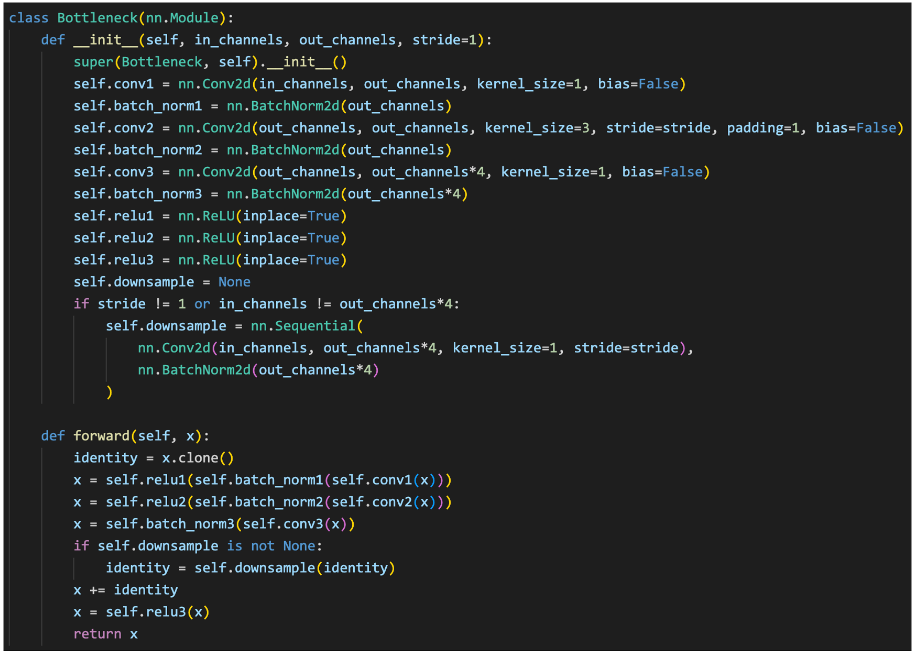
```
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Flatten(start_dim=1),
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
            nn.Softmax(dim=1)
        )
```

ResNet50

Optimizer: SGD

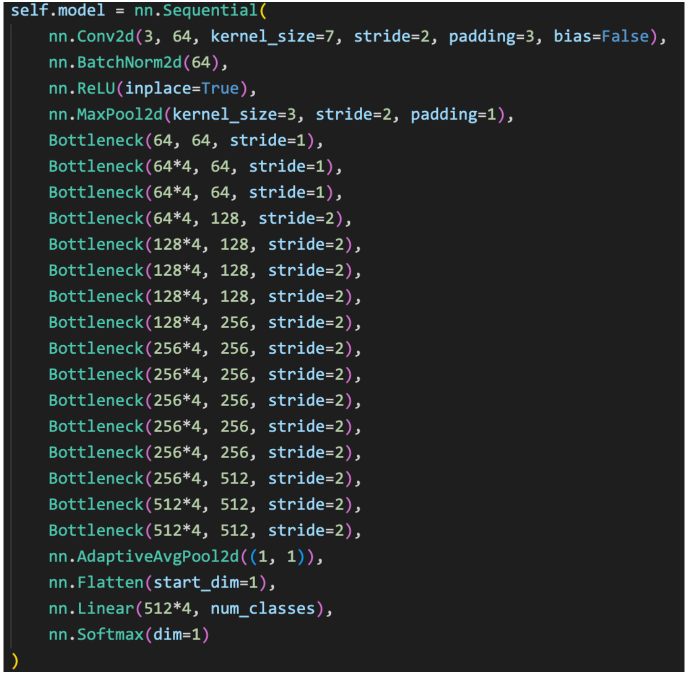
Criterion: CrossEntropyLoss

BottleNeckBlock:



```python
class Bottleneck(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False)
        self.batch_norm1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.batch_norm2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d(out_channels, out_channels*4, kernel_size=1, bias=False)
        self.batch_norm3 = nn.BatchNorm2d(out_channels*4)
        self.relu1 = nn.ReLU(inplace=True)
        self.relu2 = nn.ReLU(inplace=True)
        self.relu3 = nn.ReLU(inplace=True)
        self.downsample = None
        if stride != 1 or in_channels != out_channels*4:
            self.downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels*4, kernel_size=1, stride=stride),
                nn.BatchNorm2d(out_channels*4)
            )

    def forward(self, x):
        identity = x.clone()
        x = self.relu1(self.batch_norm1(self.conv1(x)))
        x = self.relu2(self.batch_norm2(self.conv2(x)))
        x = self.batch_norm3(self.conv3(x))
        if self.downsample is not None:
            identity = self.downsample(identity)
        x += identity
        x = self.relu3(x)
        return x
```

Model architecture:

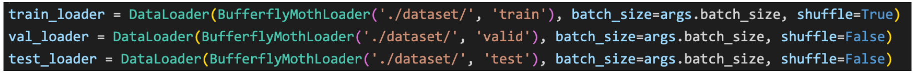


```python
self.model = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
    Bottleneck(64, 64, stride=1),
    Bottleneck(64*4, 64, stride=1),
    Bottleneck(64*4, 64, stride=1),
    Bottleneck(64*4, 128, stride=2),
    Bottleneck(128*4, 128, stride=2),
    Bottleneck(128*4, 128, stride=2),
    Bottleneck(128*4, 128, stride=2),
    Bottleneck(128*4, 256, stride=2),
    Bottleneck(256*4, 256, stride=2),
    Bottleneck(256*4, 256, stride=2),
    Bottleneck(256*4, 256, stride=2),
    Bottleneck(256*4, 256, stride=2),
    Bottleneck(256*4, 256, stride=2),
    Bottleneck(256*4, 512, stride=2),
    Bottleneck(512*4, 512, stride=2),
    Bottleneck(512*4, 512, stride=2),
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten(start_dim=1),
    nn.Linear(512*4, num_classes),
    nn.Softmax(dim=1)
)
```

2.2 Details of dataloader

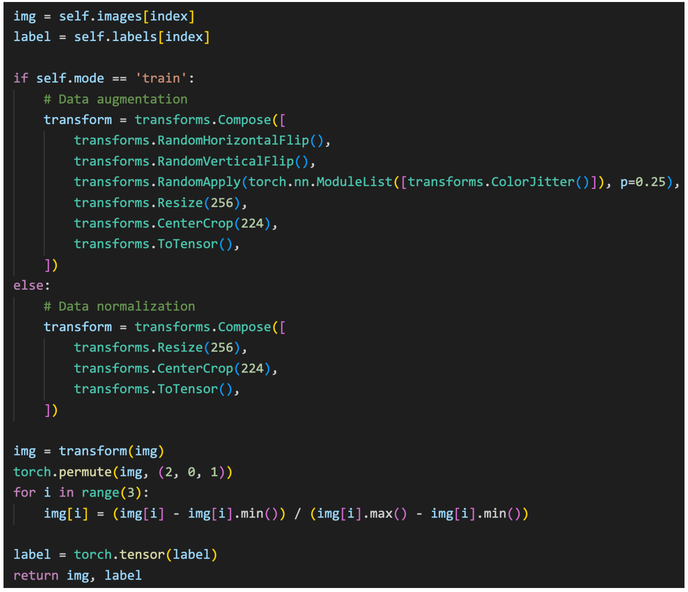
Dataloaders

I designed my own dataloader to load the dataset. Here batch size is 32.



```
train_loader = DataLoader(BufferflyMothLoader('./dataset/', 'train'), batch_size=args.batch_size, shuffle=True)
val_loader = DataLoader(BufferflyMothLoader('./dataset/', 'valid'), batch_size=args.batch_size, shuffle=False)
test_loader = DataLoader(BufferflyMothLoader('./dataset/', 'test'), batch_size=args.batch_size, shuffle=False)
```

get_item function:

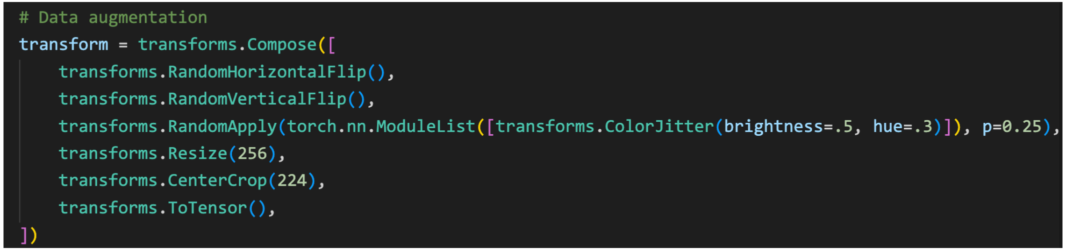


```
img = self.images[index]
label = self.labels[index]

if self.mode == 'train':
    # Data augmentation
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomApply(torch.nn.ModuleList([transforms.ColorJitter()]), p=0.25),
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ])
else:
    # Data normalization
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ])

img = transform(img)
torch.permute(img, (2, 0, 1))
for i in range(3):
    img[i] = (img[i] - img[i].min()) / (img[i].max() - img[i].min())

label = torch.tensor(label)
return img, label
```

3 Data Preprocessing

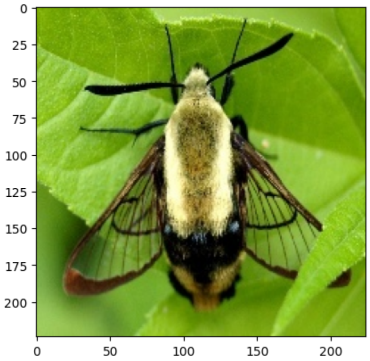
3.1 How to preprocess the data?

I use some data augmentation techniques to preprocess the data. For example, random flip the image and randomly change the brightness, contrast, saturation and hue of an image.

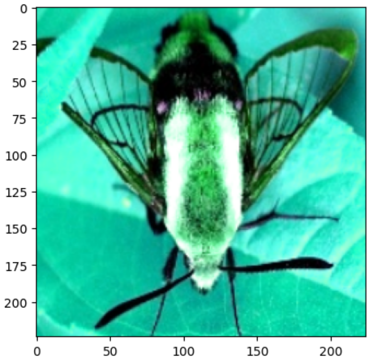


```
# Data augmentation
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomApply(torch.nn.ModuleList([transforms.ColorJitter(brightness=.5, hue=.3)]), p=0.25),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
])
```

Original image:



Augmented image:



3.2 What makes my method special?

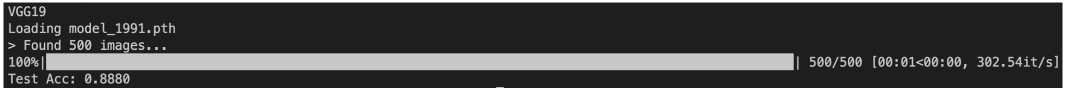
I use the data augmentation technique to increase the diversity of the dataset. This can help the model to learn more features and improve the accuracy.

4 Experimental Results

4.1 The highest accuracy

VGG19

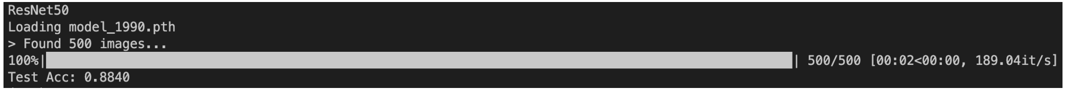
The highest accuracy on test dataset is 88.8%.



```
VGG19
Loading model_1991.pth
> Found 500 images...
100%|██████████████████████████████████████████████████| 500/500 [00:01<00:00, 302.54it/s]
Test Acc: 0.8880
```

ResNet50

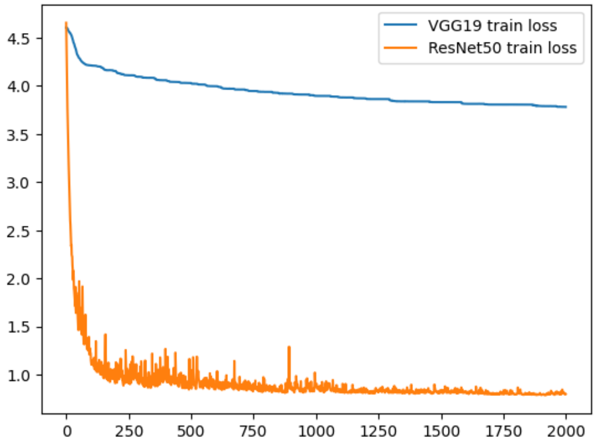
The highest accuracy on test dataset is 88.4%.



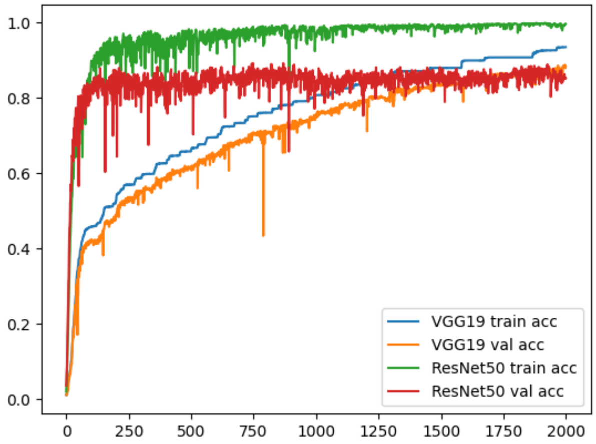
```
ResNet50
Loading model_1990.pth
> Found 500 images...
100%|██████████████████████████████████████████████████| 500/500 [00:02<00:00, 189.04it/s]
Test Acc: 0.8840
```

4.2 Comparison

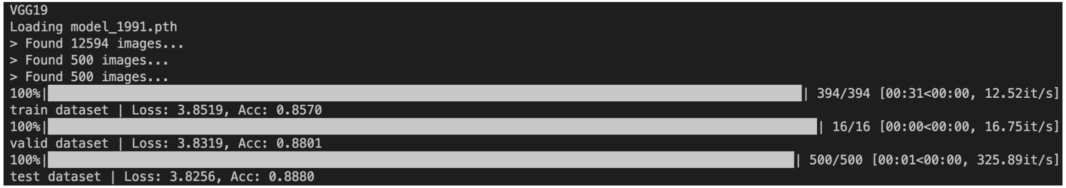
Training Loss:



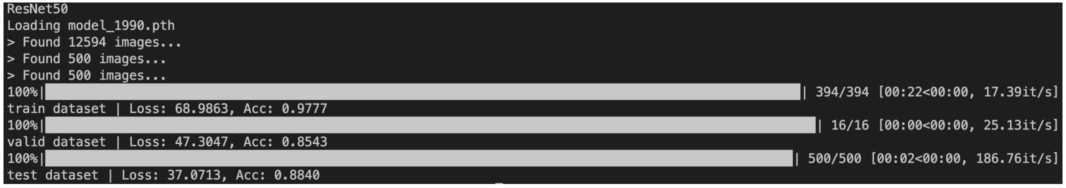
Training and Validation Accuracy per Epoch:



Test on each datasets:



```
VGG19
Loading model_1991.pth
> Found 12594 images...
> Found 500 images...
> Found 500 images...
100%|██████████| 394/394 [00:31<00:00, 12.52it/s]
train dataset | Loss: 3.8519, Acc: 0.8570
100%|██████████| 16/16 [00:00<00:00, 16.75it/s]
valid dataset | Loss: 3.8319, Acc: 0.8801
100%|██████████| 500/500 [00:01<00:00, 325.89it/s]
test dataset | Loss: 3.8256, Acc: 0.8880
```



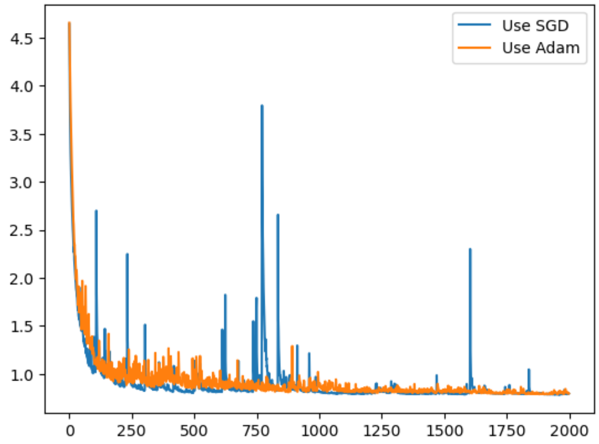
```
ResNet50
Loading model_1990.pth
> Found 12594 images...
> Found 500 images...
> Found 500 images...
100%|██████████| 394/394 [00:22<00:00, 17.39it/s]
train dataset | Loss: 68.9863, Acc: 0.9777
100%|██████████| 16/16 [00:00<00:00, 25.13it/s]
valid dataset | Loss: 47.3047, Acc: 0.8543
100%|██████████| 500/500 [00:02<00:00, 186.76it/s]
test dataset | Loss: 37.0713, Acc: 0.8840
```

5 Discussion

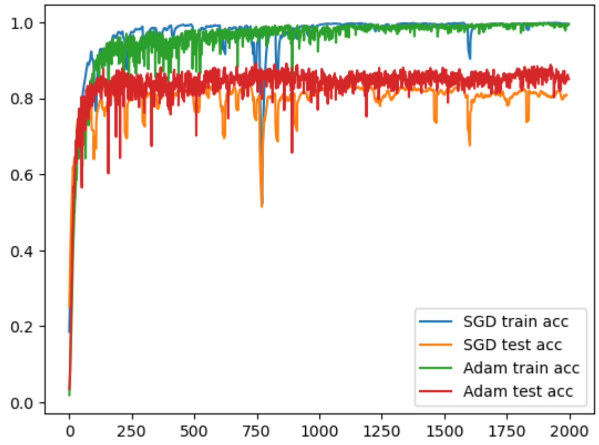
5.1 Different Optimizer

I compare the performance of the model with SGD optimizer and Adam optimizer. The result shows that the model with Adam optimizer has a higher accuracy under test dataset than the model with SGD optimizer. SGD can reach higher accuracy on training dataset but Adam can reach higher accuracy on test dataset. SGD is more likely to overfit the training dataset.

Training Loss:



Training and Test Accuracy per Epoch:


SGD train acc
SGD test acc
Adam train acc
Adam test acc